2017

Fully Distributed And Mixed Symmetric Diagonal Dominant Solvers For Large Scale Optimization

Rasul Tutunov
University of Pennsylvania, tutunov@seas.upenn.edu

Fully Distributed And Mixed Symmetric Diagonal Dominant Solvers For Large Scale Optimization

Abstract

Over the past twenty years, we have witnessed an unprecedented growth in data, inaugurating the so-called "Big Data" Epoch. Throughout these years, the exponential growth in the power of computer chips forecasted by Moore's Law has allowed us to increasingly handle such growing data progression. However, due to the physical limitations on the size of transistors we have already reached the computational limits of traditional microprocessors' architecture.Therefore, we either need conceptually new computers or distributed models of computation to allow processors to solve Big Data problems in a collaborative manner.

The purpose of this thesis is to show that decentralized optimization is capable of addressing our growing computational demands by exploiting the power of coordinated data processing. In particular, we propose an exact distributed Newton method for two important challenges in large-scale optimization: Network Flow and Empirical Risk Minimization.

The key observation behind our method is related to the symmetric diagonal dominant structure of the Hessian of dual functions correspondent to the aforementioned problems. Consequently, one can calculate the Newton direction by solving symmetric diagonal dominant (SDD) systems in a decentralized fashion.

We first propose a fully distributed SDD solver based on a recursive approximation of SDD matrix inverses with a collection of specifically structured distributed matrices. To improve the precision of the algorithm, we then apply Richardson Preconditioners arriving at an efficient algorithm capable of approximating the solution of SDD system with any arbitrary precision.

vi

Our second fully distributed SDD solver significantly improves the computational performance of the rst algorithm by utilizing Chebyshev polynomials for an approximation of the SDD matrix inverse. The particular choice of Chebyshev polynomials is motivated by their extremal properties and their recursive relation.

We then explore mixed strategies for solving SDD systems by slightly relaxing the decentralization

requirements. Roughly speaking, by allowing for one computer to aggregate some particular information from all others, one can gain quite surprising computational benefits. The key idea is to construct a spectral sparsifier of the underlying graph of computers by using local communication between them.

Finally, we apply these solvers for calculating the Newton direction for the dual function of Network Flow and Empirical Risk Minimization. On the theoretical side, we establish quadratic convergence rate for our algorithms surpassing all existing techniques. On the empirical side, we verify our superior performance in a set of extensive numerical simulations.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Ali Jadbabaie

Keywords
distributed algorithms, distributed optimization, empirical risk minimization, network flow problems, SDD solvers, sparsifiers

Subject Categories
Computer Sciences | Engineering

FULLY DISTRIBUTED AND MIXED SYMMETRIC DIAGONAL DOMINANT SOLVERS FOR LARGE SCALE OPTIMIZATION

Rasul Tutunov

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2017

Supervisor of Dissertation

Ali Jadbabaie, JR East Professor of Engineering, Institute for Data, Systems and Society, MIT

Graduate Group Chairperson

Lyle H. Ungar, Professor, Computer and Information Science

Dissertation Committee

George Pappas,
Joseph Moore Professor, Electrical and Systems Engineering, University of Pennsylvania

Michael Kearns,
Professor, Computer and Information Science, University of Pennsylvania

Daniel D. Lee,
Professor, Computer and Information Science, University of Pennsylvania

Suvrit Sra,
Principal Research Scientist (PI), Laboratory for Information and Decision Systems , MIT

FULLY DISTRIBUTED AND MIXED SYMMETRIC DIAGONAL DOMINANT SOLVERS FOR LARGE SCALE OPTIMIZATION

Dedicated to my Family.

ACKNOWLEDGEMENTS

Even though only my name is presented on the cover of this thesis, many people have contributed to its existence. I owe my gratitude to all of the people who have made this journey nothing short of marvelous.

First of all, I would like to express my sincere gratitude to my thesis supervisor, Prof. Ali Jadbabaie. Without his constant support and encouragement, this thesis would not have been written. I was truly fortunate to be guided by a scientist who foster a rigorous mathematical culture and at the same time provides very insightful "big picture" explanations. It is usually said that theory is when we know everything but nothing works and practice is when everything works but no one knows why. Throughout these years, Ali showed amazing patience and persistence explaining to me the importance of both of these aspects for - impactful research.

My first meeting with Ali happened in August 2011 in Philadelphia, where I came to pursue my doctorate degree at University of Pennsylvania. When I entered his office the first things I saw were hundreds of books and "Guernica", a reproduction of Picasso's very famous painting. It took me quite a while to realize that the office was a perfect reflection of Ali's personality: the combination of deep knowledge and the ability to look at problems abstractly by focusing on important features and removing all nonessential details. The art to separate the wheat from the chaff is something that Ali has tried to teach me all this way (I hope successfully).

I am deeply indebted to my main collaborator and colleague Dr. Haitham Bou Ammar with whom I first met in 2014. Back then, I was a third-year PhD student with huge ambitions and zero publications. Haitham showed me how pure theoretical constructions and results can be applied to the very practical fields of machine learning and artificial intelligence. He constantly motivated and encouraged me when our papers got rejected (which happened quite often, so all the bartenders in University City became our good friends). But more than exceptional professional skills, I would like to mention Haitham's personal qualities. He always was ready to listen carefully and helped with advice for any kind of problem or experience that my eager character encountered. Apart from common scientific interests, we share the same values and views on many subjects including traditions, music, and especially cuisine. He gave me an absolutely solid constructive proof that the best cuisine in the world is Lebanese. I would like to express my sincere thanks to Haitham for these fruitful years of collaboration and I would like to say that I was fortunate to find not just a good

friend but a brother.

I would like to thank my committee members for agreeing to assess my thesis and taking the time to read it through: Prof. George Pappas, Prof. Michael Kearns, Prof. Daniel D. Lee and Prof. Suvrit Sra. Your remarks and questions allowed me to look at completely different angles on this work and eventually improved it significantly.

In addition to those mentioned above, I am very grateful to many friends who helped me with their various forms of support during my graduate study. You are too many to be listed here, but you know who you are!

ABSTRACT

FULLY DISTRIBUTED AND MIXED SYMMETRIC DIAGONAL DOMINANT SOLVERS FOR LARGE SCALE OPTIMIZATION

Rasul Tutunov

Ali Jadbabaie

Over the past twenty years, we have witnessed an unprecedented growth in data, inaugurating the so-called "Big Data" Epoch. Throughout these years, the exponential growth in the power of computer chips forecasted by Moore's Law has allowed us to increasingly handle such growing data progression. However, due to the physical limitations on the size of transistors we have already reached the computational limits of traditional microprocessors' architecture.Therefore, we either need conceptually new computers or distributed models of computation to allow processors to solve Big Data problems in a collaborative manner.

The purpose of this thesis is to show that decentralized optimization is capable of addressing our growing computational demands by exploiting the power of coordinated data processing. In particular, we propose an exact distributed Newton method for two important challenges in large-scale optimization: Network Flow and Empirical Risk Minimization.

The key observation behind our method is related to the symmetric diagonal dominant structure of the Hessian of dual functions correspondent to the aforementioned problems. Consequently, one can calculate the Newton direction by solving symmetric diagonal dominant (SDD) systems in a decentralized fashion.

We first propose a fully distributed SDD solver based on a recursive approximation of SDD matrix inverses with a collection of specifically structured distributed matrices. To improve the precision of the algorithm, we then apply Richardson Preconditioners arriving at an efficient algorithm capable of approximating the solution of SDD system with any arbitrary precision.

Our second fully distributed SDD solver significantly improves the computational performance of the first algorithm by utilizing Chebyshev polynomials for an approximation of the SDD matrix inverse. The particular choice of Chebyshev polynomials is motivated by their extremal properties and their recursive relation.

We then explore mixed strategies for solving SDD systems by slightly relaxing the decentralization requirements. Roughly speaking, by allowing for one computer to aggregate some particular information from all others, one can gain quite surprising computational benefits. The key idea is to construct a spectral sparsifier of the underlying graph of computers by using local communication between them.

Finally, we apply these solvers for calculating the Newton direction for the dual function of Network Flow and Empirical Risk Minimization. On the theoretical side, we establish quadratic convergence rate for our algorithms surpassing all existing techniques. On the empirical side, we verify our superior performance in a set of extensive numerical simulations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1 : INTRODUCTION

Data analysis has become a major tool for acquiring predictive models with the goal of discovering useful information, suggesting conclusions, and supporting decision-making in a variety of fields including but not limited to heath-care (Zupan et al. (1997)), engineering (Spiliopoulou et al. (2014)), marketing (Jeffery (2010)), et cetera. Before achieving a "data product" raw information has to go through several phases. After being acquired, data is first processed and cleaned to remove errors and record duplication, as well as to recognize outliers that might have been incorrectly entered. Once cleaned, it can be analyzed to arrive at mathematical models that can be used to derive conclusions and make predictions in reality. The goal of such models is to identify relationships among the variables, such as correlation or causation that reflect inherent properties in a particular data set. Machine learning (ML) is a sub-field of computer science that is dedicated to deriving these complex predictive models. In ML, computers act without being explicitly programmed. Contrary to standard programming paradigms, in ML, a programmer designs a general architecture of a computer code that can accept inputs and derive outputs. Rather than pre-setting all required parameters to achieve the desired task, these are acquired with the aid of input data through a process of self-tuning. Here the program automatically optimizes for free-parameters to minimize a cost function that describes the task. To clarify, consider the example of classifying images of rhinos versus elephants. A general computer program that can accept images, say in pixel format, and output a class label, i.e., a discretized zero/one output, is first written. The program, however, is not explicitly told how to output a class label to an input image. Rather a form of parameterized mapping between images and class labels is assumed (e.g., logistic function, neural network, etc.). Having acquired data, self-tuning commences to determine the "best" parameters (i.e., those that optimize a pre-specified cost/error on the training data) that correctly classifies rhinos from elephants.

Searching for the "best" set of free-parameters ties machine learning to numerical optimization, which delivers a rich literature of efficient and scalable algorithms for data analysis and machine learning. Among these, maybe the most abundant are centralized first-order gradient techniques (Boyd and Vandenberghe (2004b), Ruder (2016), Andrychowicz et al. (2016a)). These algorithms come in different flavors under different names, including gradient and steepest descent (Andrychowicz et al. (2016b)), or stochastic gradient descent (Bottou (2010)). Here, parameters are tuned by iteratively

following a scaled version of the gradient of the cost function. In the standard case, the gradient is computed by running over all data points, while in the stochastic setting only one or a subset of the data is considered, leading to a more appealing algorithm for large-scale applications. Though abundant and cheap to compute, stochastic gradient-based methods are relatively slow exhibiting sub-linear convergence of the form $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$ with t being the total number of iterations (Blair (1985), Agarwal et al. (2012)). This performance is caused by the diminishing step size essential for the global convergence. Adaptive Gradient (AdaGrad) is an alternative approach that removes the step size issue by dynamically incorporating knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning. In other words, this method gives frequently occurring features low learning rates and infrequent features high learning rates. The intuition behind of this behavior is that each time an infrequent feature is seen, the algorithm should take notice. Thus, the adaptation facilitates finding and identifying very predictive but comparatively rare features (Duchi et al. (2010)). Although the convergence properties of AgaGrad are similar to to those of its stochastic counterpart, numerous empirical studies show significant improvement for the sparse data set scenarios.

Aiming at improving convergence speed limitations, the authors in Boyd et al. (2011) relied on a primal-dual decomposition technique to propose the alternating direction method of multipliers (ADMM) that achieves linear convergence of the form $\mathcal{O}\left(\frac{1}{t}\right)$. Despite being successful at efficiently reducing the error function value, it has been shown in later studies (Nishihara et al. (2015), Kadkhodaie et al. (2015)) that ADMM requires numerous iterations to achieve accurate solutions. Another class of algorithms with superior performance to these detailed earlier is the class of second-order (Newton) methods. Rather than solely relying on the gradient, Newton iterates consider the curvature of the optimization objective by taking Hessian (the matrix of second-order gradients) into account. Here, updates are done by descending in the Newton direction. Convergence results in Boyd et al. (2011) have shown that following the Newton direction leads to improved algorithms with two phases of convergence being strict and quadratic decrease. Despite the fast convergence properties, the centralized application of Newton method is restricted by the necessity to compute and invert the Hessian matrix. This situation becomes even worse for large-scale scenarios due to excessive memory requirements.

Recently, researchers and developers have met new computational challenges caused by a remarkable growth of data. With such unprecedented increase, the memory and computational requirements

are growing much faster than processing speeds, making traditional centralized solutions (such as various first order stochastic gradient descent (SGD) algorithms) inefficient. As an example let us consider the current state of the art deep neural networks (Szegedy et al. (2015), Krizhevsky et al. (2012)) operating with millions of parameters and characterizing with a very slow training time (may take several days). The typical strategy to accelerate the training procedure is adopting it to a parallel framework by allocating resources over many machines. (Dean et al. (2012)). This approach not only speeds up neural network training but also allows researchers to build more sophisticated models where different machines compute different splits of the mini-batches. Even though SGD exhibits nice scalability with respect to the complexity of a model and the size of datasets, it does not adopt well into a parallel setting: larger mini-batches and more parallel computations exhibit diminishing returns for SGD algorithms.

In the past decades, distributed programming frameworks such as Open Message Passing Interface (MPI) have endorsed rich primitives to leverage flexibility in implementing algorithms across distributed computing resources, often delivering high-performance but coming with the cost of high implementation complexity (Quinn (2003)). Despite the high performance and significant practical impact, the large-scale application of MPI methods is restricted by implementation complexities. In order to handle correctly communication and synchronization between clusters, MPI methods require a lot of programming effort as well as deep system-level understanding. The alternative models, such as Hadoop and SPARK, have recently emerged and formulated well-defined distributed programming paradigms leading to a powerful set of APIs specially built for distributed processing (Chu et al. (2007)). These abstractions make the implementation of distributed algorithms more easily accessible to researchers, but seem to come with poorly understood overheads associated with communication and data management, which make the tight control of computation vs communication cost more difficult.

Apart from serving as a computational tool for very specialized scientific purposes and companies' operational routines, distributed optimization is strongly involved in new emerging global technologies, such as Internet of Things (IoT) (Vermesan and Friess, Mattern and Floerkemeier (2010)). Simply put, IoT is the concept of connecting basically any device with an on and off switch to the Internet (and/or to each other). Cell phones, headphones, lamps, refrigerators, coffee machines and almost any device might be a part of IoT. In other words, Internet of Things can be visualized as a

giant network of connected intelligent components interacting with each other. The analyst company Gartner predicts that by 2020 the number of components in this network will be over 26 billion. According to the recent EU Commission action plan recognized the Internet of Things as a general evolution of the Internet from a network of interconnected computers to a network of interconnected objects" (IoT (2009)).

From a technical point of view, the IoT is not the result of a single novel technology; instead, several complementary technical developments provide capabilities to build a bridge between the virtual and physical world. These capabilities include but are not limited to:

- Communication and cooperation: Different components of the IoT network have the ability to interact with each other, to make use of data and services and update their state. Wireless technologies such as WPAN, GSM and UMTS, Wi-Fi, Bluetooth, ZigBee and various other wireless networking standards are currently under development.

- Distributed information processing: Intelligent objects feature a processor, plus storage capacity. These resources can be used, for example, to process and interpret sensor information, or to collectively perform more sophisticated computations.

IoTs distributed and dynamic nature, resource constraints of sensors and embedded devices as well as the amounts of generated data are challenging even for the most advanced data analysis methods known today. In particular, the IoT requires a new generation of distributed analysis methods. Many surveys (Atzori et al. (2010), Gubbi et al. (2013) Partynski and Koo (2013), Xu et al. (2014)) have strongly focused on the centralization of data in the cloud and big data analysis (so-called Master/Slave model), which follows the paradigm of parallel high-performance computing. However, bandwidth and energy can be too limited for the transmission of raw data, or it is prohibited due to privacy constraints. Such communication-constrained scenarios require decentralized analysis algorithms which at least partly work directly on the generating devices.

In the distributed setting, central problems are split across multiple processors each having access to local objectives. To clarify, consider our running example of classifying images of rhinos versus elephants. Rather than searching for a centralized solution, one can distribute the optimization across multiple processors each having access to local costs defined over random subsets of the full data set. In such a case, each processor learns a separate "chunk" of the latent model which is then

unified by incorporating consensus constraints. Recently, an increasing body of research has targeted such a setting explicitly. For instance, Google's federated learning aims at achieving collaborative model learning across multiple mobile devices without the need for centralized training data in the cloud. Here, a unified model is learned collaboratively based on only local data available on each mobile device (Konecný et al. (2016b), Konecný et al. (2016a)). Another example of the growing trend for distributed optimization is the recent work of McMahan et al. (2016). Here, the authors consider highly-dimensional optimization problems with billions of parameters and solve them with the help of tens of thousands of CPU cores. They show that this approach can lead to improved convergence allowing for algorithms that can handle such Big Data problems.

Analogous to the centralized literature, distributed optimization has also provided a rich set of algorithms for determining free parameters in a decentralized fashion by relying on communication between the involved processors. Among many approaches (e.g., distributed averaging (Olshevsky (2014)), coordinate descent (Richtárik and Takác (2013), Trofimov and Genkin (2015)) and incremental methods (Bertsekas (2015), Nedic et al. (2001)), two popular classes can be differentiated. The first is sub-gradient based, while the second relies on a decomposition-coordination procedure. Sub-gradient algorithms are similar to centralized first-order gradient methods, where they proceed by taking a gradient-related step, followed by an averaging with the neighbors at each iteration. The computation of each step is relatively cheap and can easily be implemented in a distributed fashion (Nedic and Ozdaglar (2009)). Though cheap to compute, the best known bound on the rate of sub-gradients is sub-linear. As such, these algorithms share the problems of their centralized counterparts. The second class of algorithms solve constrained problems by relying on dual methods. Their state-of-the-art technique is a distributed version of the centralized ADMM (Wei and Ozdaglar (2012), Chang et al. (2015)) that achieves a linear convergence rate with low message complexity per iteration. Apart from only achieving linear convergence, distributed ADMM has also been shown to suffer from accuracy issues as detailed in Kadkhodaie et al. (2015).

Many rate and accuracy improvements can be gained from adopting distributed second-order (Newton) methods. Though a variety of techniques have been proposed (Zargham et al. (2013) Mokhtari et al. (2015), Gurbuzbalaban et al. (2015), Eisen et al. (2016)), less progress has been made at leveraging ADMM's accuracy and convergence rate issues.
In a recent attempt Mokhtari et al. (2015), the authors propose a distributed second-order method

by using the approach in Zargham et al. (2013) to approximate the Newton direction. As detailed later, this method suffers from two problems. First, it fails to outperform ADMM and second, it faces storage and computational deficiencies for large data sets, and thus ADMM retains state-of-the-art status.

The alternative approach proposed by Eisen et al. (2016) calculates Newton direction by applying a distributed version of the Sherman-Morrison formula (Sherman and Morison (1949)). The advantages of D-BFGS relative to approximate Newton methods are that they do not require computation of Hessians, which can itself be expensive, and that they apply in any scenario in which gradients are distributedly computable irrespective of the structure of the Hessian. In terms of convergence properties, D-BFGS resembles the previously discussed Network Newton algorithm and empirically surpassed by ADMM.

In scenarios where the number of local functions is large and not all of them simultaneously available, one is interested in optimization techniques that can iteratively update the estimate for an optimal solution using partial information about component functions. The incremental Newton method (Gurbuzbalaban et al. (2015)) cycles deterministically through the component functions f_i and uses the gradient of f_i to determine the descent direction and the Hessian of f_i to construct the Hessian of the sum of component functions. Apart from requiring a global coordinator, this method suffers from immense computational requirements caused by sequential inversion of local Hessians in each iteration.

Contributions: In the previous paragraphs we discussed a general distributed optimization framework and presented a brief overview of existing decentralized algorithms. In the next paragraphs, we focus on both theoretical and practical contributions of this thesis.

Though appealing, computing the Newton direction in a distributed fashion is challenging due to the need of inverting the Hessian matrix that requires global information. In our work, a novel connection between the Hessian of a distributed optimization problem and Symmetric Diagonally Dominant (SDD) matrices is derived. Recently, SDD matrices have captured strong attention due to a series of breakthrough results (Koutis and Miller (2007), Kelner and Madry (2009), Kelner et al. (2013), Cohen et al.) starting with work of Daniel Spielman and Shang-Hua Teng (Spielman and Teng (2006)) who suggested the first almost linear time algorithm for solving SDD linear systems. Although all these algorithms have been designed only for the centralized case, the diagonal domi-

nance property gives us hope to calculate the Newton direction for a distributed problem without a heavy computational burden by attempting fast distributed SDD solvers.

In Section 2.2 we suggested two fully distributed algorithms for solving symmetric diagonally dominant systems. The first method utilizes the idea of Peng and Spielman (2013) by constructing decentralized approximated inverse chain in a recursive fashion. This chain can be visualized as a collection of matrices such that each cluster stores only the corresponding row of each of these matrices. By traversing this chain first in forward and then in backward direction we arrive at a crude solution of the SDD system that approximates an exact solution up to some constant accuracy. In order to reach any given precision, we accompany the approximated inverse chain with an iterative procedure called Richardson Preconditioners (Axelsson (1994a)) gradually improving the accuracy. To the moment of publishing, this method was the fastest fully distributed SDD solvers with time complexity proportional to a condition number of an SDD matrix.

The linear dependence on a condition number is favorable for a wide range of cluster topologies including expanders (Hoory et al. (2006)), random graphs (Hofstad (2008)), and grids (Fadel et al. (2015)). However, for configurations of clusters characterized by long diameters this behavior might lead to a slow performance. As an example, let us consider BarBell topology for clusters (Northup) where two cliques of size $\lceil \frac{n}{3} \rceil$ connected with a path graph on $\lceil \frac{n}{3} \rceil$ nodes. Due to this bottleneck part, the corresponding condition number for SDD matrices can be cubic in a number of clusters. To remedy this effect consider a polynomial approximation of SDD inverse with properly scaled Chebyshev polynomials. The particular choice of these polynomials is motivated by their extremal properties and the fast performance of the algorithm is guaranteed by the recursive relation between them (Chebyshev (1853)). Eventually, we arrive at a new SDD solver capable of computing an approximate solution with any given precision in time proportional to the square root of a condition number of an SDD matrix.

Having introduced fully decentralized techniques, we noticed that the total number of messages (in literature often referred to as communication complexity) is increasing with "spreadness" of the underlying network of clusters. To illustrate this phenomenon, let us consider Star Graph (Shao et al. (2000)) and previously discussed Barr-Bell models. In the former, all nodes are only connected to a single master node responsible for all computational burden. As a result, the message complexity in this case is only characterized by the number of clusters n. In the latter model, the information exchange between the clusters is delayed by the bottleneck part, causing the overall

message complexity to be bounded by $\mathcal{O}\left(n^{\frac{7}{2}}\right)$. Therefore, the question that follows naturally from this observation is: can we eliminate this drastic effect of cluster configuration by "slightly" relaxing the decentralization requirements. Please notice, that by completely removing this requirement we arrive at a highly impractical scenario, where all input data must be stored and processed in a single cluster. In Section 2.3, using spectral sparsifiers, we propose a mixed strategy that allows for a fixed node to aggregate some information from all others while preserving the slightly worse bound on the size of local memory of this cluster compared to the fully decentralized case. Surprisingly, apart from significantly improving the message complexity, we show that the time complexity of a new mixed algorithm is even faster than in centralized and fully decentralized scenarios.

On the practical side, we contribute by applying the proposed SDD solvers for two problems that have been in the focus of researchers for many years: Network Flow Problems and Empirical Risk Minimization.

Network Flow Problems are considered one of the most fundamental problems in industrial engendering and theoretical computer science (Ahuja et al. (1993b), Ford and Fulkerson (2010), Dahan and Amin (2016), Aly and Van Vyve (2015)). The ultimate goal here is to minimize the total cost associated with all edges connecting clusters subject to flow conservation constraints. In Section 3.1 we show that significant progress can be attained by applying primal-dual techniques and formulating an unconstrained dual problem. Careful analysis of the dual function shows that its Hessian exhibits diagonal dominant property and, therefore, concedes utilization of SDD solvers for calculating the correspondent Newton direction. As a result, we develop the first exact distributed Newton method for Network Flow problem demonstrating a quadratic convergence rate. We empirically validate this result on a variety of numerical simulations with real data against several methods, including the state-of-the-art ADAL algorithm (Chatzipanagiotis et al. (2015)).

Our second practical contribution is motivated by machine and statistical learning and has a wide range of applications in data science (Zhang (2010)), robotics (Cetto et al. (2013)), image processing (He et al. (2015)), speech recognition (Yu and Deng (2014)) et cetera. In Empirical Risk Minimization problem a true risk defined by the unknown probability distribution over the training data is approximated with an empirical risk. The latter is represented as an average over loss func-

tions computed at each data point for the chosen prediction model. In the distributed framework the associated optimization problem is formulated as a finite sum minimization problem combined with consensus constraints between the clusters. In Section 3.2.9, following primal-dual strategy we constructed the corresponding dual function and recognize that its Hessian can be represented as a product of SDD matrices. Based on this observation we calculate the Newton direction by sequentially solving a collection of SDD systems. Our exact distributed Newton method for the Empirical Risk Minimization problem is the first distributed algorithm achieving a quadratic convergence rate. Finally, we suggest Hessian-free and stochastic variation of our algorithm by applying automatic differentiation techniques and the Sherman-Morrison formula. In Section 3.2.9 we verify our superior performance by conducting a series of numerical simulations:

- against traditional decentralized stochastic gradient descent method using SPARK.
- against centralized stochastic gradient descent using streaming.
- against other distributed algorithms including state-of-the-art distributed ADMM (Wei and Ozdaglar (2012))

Finally, conclusions and future work are presented in Section 3.2.9

CHAPTER 2 : SYMMETRIC DIAGONAL DOMINANT SOLVERS

The problem of solving a system of linear equations given by symmetric diagonally dominant matrices (SDD) arises in a variety of real-world applications. For example, SDDs appear in multitude of fields including but not limited to determining solutions of partial differential equations LeVeque (2007), semi-supervised learning Zhu et al. (2003), Zhou and Schlkopf (2004), computer vision Casaca (2015), and computation of maximum flows in graphs Daitch and Spielman (2008), Madry (2013).

Recently, much progress towards solving such problems has been made. Out of the different techniques, the solution of Speilman and Teng Spielman and Teng (2006) stands out. Here, the authors exploited three components for an efficient SDD solver. Namely, using the multi-level framework suggested by Joshi (1996), low-stretch spanning tree preconditioners introduced by Boman et al. (2008), and spectral graph sparsifiers Spielman and Teng (2008), they proposed a nearly linear-time algorithm for solving SDD systems. These results were then improved by Koutis et al. Koutis et al. (2010), Cohen et al. (2014), who developed an even faster algorithm for acquiring ϵ-close solutions to SDD linear systems. Improvements have since been discovered by Kelner et al. Kelner and Madry (2009), where their algorithm relied on only low stretch trees and eliminated the need for graph sparsifiers and the multi-level framework.

Motivated by applications, much interest has been devoted to developing parallel versions of these algorithms. Koutis and Miller Koutis and Miller (2007) proposed an algorithm requiring nearly-linear work and $m^{\frac{1}{6}}$ depth (m is the total number of nonzero entries of SDD matrix) for planar graphs. This result was then extended to general graphs in Blelloch et al. (2011) leading to depth close to $m^{\frac{1}{3}}$. Since then, Peng and Spielman Peng and Spielman (2013) have proposed an efficient parallel solver requiring nearly-linear work and poly-logarithmic depth without the need for low-stretch spanning trees.

Less progress, on the other hand, has been made in the distributed version of these solvers. Contrary to the parallel setting, memory is not shared and is rather distributed in the sense that each unit abides by its own memory restrictions. Current methods, e.g., Jacobi iteration Axelsson (1994b),

Bertsekas and Tsitsiklis (1989) can be used for such distributed solutions but require substantial complexity. In Mou et al. (2015), the authors propose a gossiping framework for acquiring a solution to SDD systems in a distributed fashion. Recent work Lee et al. (2014) considers a local and asynchronous solution for solving systems of linear equations, where they acquire a bound on the number of needed multiplication proportional to the degree and condition number of the graph for one component of the solution vector.

In this chapter, we target both the theoretical investigation of distributed SDD solvers, as well as their practical applications. The first part of the work provides basic definitions and properties of SDD systems and then presents three fully distributed and one mixed algorithm for solving SDD systems in a decentralized fashion. For all proposed methods we analyze the running time and communication complexity in terms of the underlying graph topology. Finally, to provide better intuition, we consider the following special cases for the processors' graphs: *path graph, grid graph, ring graph, random graph, scale-free network, barbell graph, and Ramanujan expander.*

The second part of the work includes interesting practical applications. In Section 3.1, we consider network flow optimization - a fundamental problem with wide-ranging applicability including but not limited to, DNA sequence alignment Ahuja et al. (1993a), scheduling on uniform parallel machines Lawler et al. (1982), urban traffic flows Ahuja et al. (1993a), optimal energy allocation Gurakan et al. (2015), etc. As networks grow larger, centralized approaches to network flow optimization underperform due to the increase in time and resource complexity needed. Distributed methods for such network optimization problems present an alternative direction to cope with such increased demand. Since existing distributed methods exhibit slow convergence rate (at most linear), we propose a fully decentralized version of the Newton method using the developed SDD solvers to arrive at quadratic convergence. On the empirical side, we demonstrate that our method outperforms current algorithms for network flow in a broad set of experiments on a variety of network topologies. We also show that this outperformance arrives at no increase in the local communication exchanges between processors.

2.1. Preliminaries

Throughout the remainder of this thesis we let $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$ be a weighted connected undirected graph with $|\mathbb{V}| = n$ nodes and $\mathbb{E}$ edges. We also assume that $w_e > 0$ for all $e \in \mathbb{E}$. Each node $i \in \mathbb{V}$ represents an agent which is capable of storing information in local memory and can perform computations. Moreover, the size of local memory for each node is bounded by $\mathcal{O}(n)$ making it impossible to gather the full topology of network $\mathbb{G}$ in one node. The distributed symmetric diagonal dominant system associated with $\mathbb{G}$ is given as:

$$\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b} \tag{2.1}$$

where $\boldsymbol{L}_{\mathbb{G}} \in \mathbb{R}^{n\times n}$ such that

$$\begin{aligned} &\boldsymbol{L}_{\mathbb{G}}(ij) = 0, \quad \forall (i,j) \notin \mathbb{E} \\ &|\boldsymbol{L}_{\mathbb{G}}(ii)| \geq -\sum_{j=1}^{n} |\boldsymbol{L}_{\mathbb{G}}(ij)| \quad \forall (i) \in \mathbb{V} \end{aligned}$$

Due to the sparsity pattern of $\boldsymbol{L}_{\mathbb{G}}$, Equation (2.1) can be represented in distributed fashion among the nodes of $\mathbb{G}$. Particularly, each node $i \in \mathbb{G}$ stores $\boldsymbol{L}_{\mathbb{G}}(ij)$ for $j = 1, \ldots, n$, as well as the i^{th} component, i.e., $\boldsymbol{b}(i)$, of the demand vector $\boldsymbol{b}$. The goal is for each node to compute the corresponding component of the solution vector by only allowing local message exchange between the nodes. As a straightforward example of (2.1) is a Laplacian system, where

$$\boldsymbol{L}_{\mathbb{G}}(ij) = \begin{cases} \sum_{e\sim i} w_e & : i = j \\ -w_e & : (ij) \in \mathbb{E} \\ 0 & : otherwise \end{cases}$$

Following Koutis et al. (2010), we target an $\epsilon-$close solution of the system in 2.1, which is defined as:

Definition Let $\boldsymbol{x}^*$ be the exact solution of system $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$. A vector $\tilde{\boldsymbol{x}}$ is called ϵ approximate solution, if

$$||\tilde{\boldsymbol{x}} - \boldsymbol{x}^*||_{\boldsymbol{L}_{\mathbb{G}}} \leq \epsilon ||\boldsymbol{x}^*||_{\boldsymbol{L}_{\mathbb{G}}} \tag{2.2}$$

where $||\boldsymbol{u}||^2_{\boldsymbol{L}_{\mathbb{G}}} = \boldsymbol{u}^{\mathsf{T}}\boldsymbol{L}_{\mathbb{G}}\boldsymbol{u}$.

The important characteristic of system (2.1), which will be heavily used in our analysis is the *condition number*, defined as:

Definition Let $\{\mu_i\}_{i=1}^{p}$ be the collection of all non-zero eigenvalues of matrix $\boldsymbol{L}_{\mathbb{G}}$, such that

$$\mu_1 \leq \mu_2 \leq \ldots \leq \mu_p$$

The the ratio $\kappa(\boldsymbol{L}_{\mathbb{G}}) = \frac{\mu_p}{\mu_1}$ is called condition number of system (2.1).

In the case of graph Laplacians $\kappa(\boldsymbol{L}_{\mathbb{G}}) = \frac{\mu_n}{\mu_2} \leq \frac{1}{2}\frac{w_{max}}{w_{min}} n d_{max} diam(\mathbb{G})$, where w_{max}, w_{min} are the maximal and minimal edge weights in $\mathbb{G}$, $diam(\mathbb{G})$ its diameter, and d_{max} is the maximal unweighted degree.

Finally, we assume a synchronous model for distributed computation. Here, all message exchanges between the nodes as well as local operations are directed by a global clock. The time complexity is given in terms of the total number of time steps on the global clock needed to terminate an algorithm. The message complexity (or communication complexity) is given in terms of the total number of messages sent by nodes to terminate an algorithm.

2.2. Fully Distributed Methods

In this section, we describe two fully distributed algorithms for solving the system in (2.1). We commence by describing the algorithms and then study their time and communication complexities.

2.2.1. Algorithm I: Matrix Inverse Chain Approach

The authors in Peng and Spielman (2013) developed a near-linear time solver capable of achieving an $\epsilon-$ close approximation to the exact solution for any arbitrary $\epsilon > 0$. In this section, we propose a distributed implementation of the aforementioned solver. Before presenting our solver, however, we next brief the main machinery from Peng and Spielman (2013) needed for the exposure.

Standard Splitting and Approximations

The story of computing an approximation to the exact solution of an SDD system of linear equations starts from the standard splitting of symmetric matrices. Given a symmetric matrix $\boldsymbol{L}_{\mathbb{G}}$ the standard splitting is given by

$$\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0 \tag{2.3}$$

where $\boldsymbol{D}_0$ is a diagonal matrix that consists of the diagonal elements in $\boldsymbol{L}_{\mathbb{G}}$ while $\boldsymbol{A}_0$ is a matrix collecting the negate of the off-diagonal components in $\boldsymbol{L}_{\mathbb{G}}$. As the goal is to determine a solution of the SDD system, we will be interested in inverses of $\boldsymbol{L}_{\mathbb{G}}$. Given the splitting in Equation 2.3, the authors in Peng and Spielman (2013) prove that the inverse of $\boldsymbol{L}_{\mathbb{G}}$ can be written as

$$(\boldsymbol{D}_0 - \boldsymbol{A}_0)^{-1} = \frac{1}{2}\left[\boldsymbol{D}_0^{-1} + \left(\boldsymbol{I} + \boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)\left(\boldsymbol{D}_0 - \boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{-1}\left(\boldsymbol{I} + \boldsymbol{A}_0\boldsymbol{D}_0^{-1}\right)\right] \tag{2.4}$$

Since $\boldsymbol{D}_0 - \boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$ is also SDD we can recurse the above for the length of $d = \mathcal{O}(\log n)$ to arrive at the so-called inverse approximated chain, $\mathcal{C} = \{\boldsymbol{D}_k, \boldsymbol{A}_k\}_{k=1}^{d}$ with

$$\boldsymbol{D}_k = \boldsymbol{D}_0 \quad \text{and} \quad \boldsymbol{A}_k = \boldsymbol{D}_0\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^k} \tag{2.5}$$

Hence, the inverse at the k^{th} recursion can be written as

$$(\boldsymbol{D}_k - \boldsymbol{A}_k)^{-1} \approx \frac{1}{2}\left[\boldsymbol{D}_k^{-1} + \left(\boldsymbol{I} + \boldsymbol{D}_k^{-1}\boldsymbol{A}_k\right)\left(\boldsymbol{D}_{k+1} - \boldsymbol{A}_{k+1}\right)^{-1}\left(\boldsymbol{I} + \boldsymbol{A}_k\boldsymbol{D}_k^{-1}\right)\right]$$

Algorithm 1 : **"Crude" SDD Solver**

Input: Inverse approximated chain $\mathcal{C}$, demand vector $\boldsymbol{b}$.
Output: A "crude" approximation $\boldsymbol{x}_0$ to the exact solution $\boldsymbol{x}^*$.
Initialize: $\boldsymbol{b}_0 = \boldsymbol{b}$.
for $k = 1$ to $d = \mathcal{O}(\log n)$ **do**
    $\boldsymbol{b}_k = \left(\boldsymbol{I} + \boldsymbol{A}_{k-1}\boldsymbol{D}_{k-1}^{-1}\right)\boldsymbol{b}_{k-1}$.
end for
$\boldsymbol{x}_d = \boldsymbol{D}_d^{-1}\boldsymbol{b}_d$.
for $k = d - 1$ to 0 **do**
    $\boldsymbol{x}_k = \frac{1}{2}\left[\boldsymbol{D}_k^{-1}\boldsymbol{b}_k + \left(\boldsymbol{I} + \boldsymbol{D}_k^{-1}\boldsymbol{A}_k\right)\boldsymbol{x}_{k+1}\right]$.
end for

Given the above, the approximate solution to the SDD system can be achieved using a two-step procedure detailed in Algorithms 1 and 2.

Algorithm 1 describes a set of instructions used to acquire a crude approximation to the real solution of the SDD system. The algorithm runs a forward and a backward loop with the number of steps equal to the length of the inverse approximate chain d[1]. In the forward loop (lines 4-6) intermediate vectors are constructed and used in the backward loop (lines 8-10) to determine a constant error (see 2.1) solution to the exact solution of the SDD system. Since the approximation incurs a constant error to the real inverse $\boldsymbol{L}_{\mathbb{G}}^{-1}$, the authors in Peng and Spielman (2013) then introduce the Richardson preconditioning scheme detailed in Algorithm 2 to arrive at any arbitrary precision

Algorithm 2 : **"Exact" SDD Solver**

1: **Input:** Inverse approximated chain $\mathcal{C}$, demand vector $\boldsymbol{b}$, precision parameter ϵ.
2: **Output:** $\epsilon-$close approximation $\tilde{\boldsymbol{x}}$, to the exact solution $\boldsymbol{x}^*$.
3: **Initialize** $\boldsymbol{y}_0 = \boldsymbol{0}$.
4: Set $\boldsymbol{\chi}$ to be the crude solution returned by Algorithm 1.
5: **for** $k = 1$ to $d = \mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ **do**
6: Set $\boldsymbol{u}_k^{(1)} = \boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_{k-1}$.
7: Set $\boldsymbol{u}_k^{(2)}$ by calling Algorithm 1 with $\boldsymbol{b} = \boldsymbol{u}_k^{(1)}$.
8: Update $\boldsymbol{y}_k = \boldsymbol{y}_{k-1} - \boldsymbol{u}_k^{(2)} + \boldsymbol{chi}$
9: **end for**
10: Set $\tilde{\boldsymbol{x}} = \boldsymbol{y}_q$.

Algorithm 2 uses Algorithm 1 as a sub-routine to drive the "crude"-solution to $\epsilon-$close approximate one for any $\epsilon > 0$ in $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ iterations.

The authors in Peng and Spielman (2013) show that the above iteration scheme can be parallelized across multiple processors leading to an algorithm which can acquire ϵ-approximate solutions in nearly linear time. As our goal is to determine the Newton direction of network flow problems in a distributed fashion, i.e., using only local information exchange, we next present a distributed version of Algorithms 1 and 2. Our strategy is similar to that in Peng and Spielman (2013) with crucial differences related to the type and length of the inverse approximate chains as detailed below.

Distributed SDD Solvers: Methodology

A key ingredient enabling efficient solvers for SDD systems is the introduction of the inverse approximate chain which rendered a parallelized implementation. Since our interest lies in a distributed solution for determining the Newton direction, the first step needed for the development of our SDD solver is an inverse chain which can be computed in a distributed fashion using only local commu-

[1]The exact length of d is given below

nication exchange among the nodes of $\mathbb{G}$. As noted in Peng and Spielman (2013), a collection of matrices $\mathcal{C} = \{\boldsymbol{D}_0, \boldsymbol{A}_0 \ldots, \boldsymbol{D}_d, \boldsymbol{A}_d\}$ is an inverse approximate chain to $\boldsymbol{L}_{\mathbb{G}}$ if there exist positive real numbers $\epsilon_0, \ldots, \epsilon_d$ such that the following three conditions are satisfied:

1. $\boldsymbol{D}_k - \boldsymbol{A}_k \approx_{\epsilon_{k-1}} \boldsymbol{D}_{k-1} - \boldsymbol{A}_{k-1}\boldsymbol{D}_{k-1}^{-1}\boldsymbol{A}_{k-1}$, for all $k = \{1, \ldots, d\}$.

2. $\boldsymbol{D}_k \approx_{\epsilon_{k-1}} \boldsymbol{D}_{k-1}$ and

3. $\boldsymbol{D}_d \approx_{\epsilon_d} \boldsymbol{D}_d - \boldsymbol{A}_d$.

The "$\approx_\epsilon$" defines the notion of approximation for matrices previously introduced in Peng and Spielman (2013) where $\boldsymbol{X} \approx_\epsilon \boldsymbol{Y}$ is written to indicate

$$\exp(-\epsilon)\boldsymbol{X} \preceq \boldsymbol{Y} \preceq \exp(\epsilon)\boldsymbol{X}$$

where $\boldsymbol{X} \preceq \boldsymbol{Y}$ reflects that $\boldsymbol{X} - \boldsymbol{Y}$ is positive semidefinite.

Starting from $\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0$, and defining the inverse chain as in Equation 2.5, it is easy to verify that $\mathcal{C}$ is an inverse approximate chain to $\boldsymbol{L}_{\mathbb{G}}$ as it satisfies all three of the above conditions. Hence, it can be used for computing a crude solution to an SDD system. To derive the distributed algorithm, we further examine the update equations of Algorithm 1. Studying the forward loop, the intermediate vectors $\boldsymbol{b}_k$ are constructed according to

$$\boldsymbol{b}_k = \left(\boldsymbol{I} + \boldsymbol{A}_{k-1}\boldsymbol{D}_{k-1}^{-1}\right)\boldsymbol{b}_{k-1} = \boldsymbol{b}_{k-1} + \underbrace{\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\ldots\boldsymbol{A}_0\boldsymbol{D}_0^{-1}}_{\text{a product of } 2^k \text{matrices}}\boldsymbol{b}_{k-1}$$

Examining each product of the 2^k matrices, e.g., $\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{b}_0$, we recognize that such a product can be computed locally by each node. This is true as the first part of the product, i.e., $\boldsymbol{z} = \boldsymbol{D}_0^{-1}\boldsymbol{b}_0$ is a simple scaling, while the second, i.e., $\boldsymbol{A}_0\boldsymbol{z}$ can be performed completely in a distributed fashion due to the sparsity pattern of $\boldsymbol{A}_0$. Consequently, the overall product of the $2k$ terms can be also distributed across the network provided the usage of a recursive update rule. Therefore, for a node i, the first part (i.e., forward loop) of the distributed solver can be concisely summarized using the set of instructions detailed in Algorithm 3.

Clearly, lines 6-8 are executing the distributed computation of the product of the $2k$ terms above, while line 9 of the algorithm is computing the i^{th} component of $\boldsymbol{b}_k$ based on the recursive scheme de-

scribed above. After running for a total of d iterations, Algorithm 3 returns the i^{th} component of intermediate vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$. Further, since the length of the chain $d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\frac{w_{max}}{w_{min}}n^3\right)\rceil$ with w_{max} and w_{min} denoting the largest and smallest edge weights, each node needs to have memory size of $\mathcal{O}(\max\{d, d_{max}\}) = \mathcal{O}(n)$.

Algorithm 3 : **Forward Loop: Distributed "Crude" Solver**

Input: The i^{th} row of matrices $\boldsymbol{A}_0, \boldsymbol{D}_0$, the i^{th} component of vector $\boldsymbol{b}$, the length $d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\frac{w_{max}}{w_{min}}n^3\right)\rceil$ of approximated inverse chain.
Output: The i^{th} components of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$.
for $k = 1$ to d **do**
  Set $l = 2^{k-1}$.
  Update $\left[\boldsymbol{u}_1^{(k-1)}\right]_i = \left[\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{b}_{k-1}\right]_i$.
  for $j = 2$ to l **do**
    Update $\left[\boldsymbol{u}_j^{(k-1)}\right]_i = \left[\boldsymbol{A}_0\boldsymbol{D}_0^{-1}\boldsymbol{u}_{j-1}^{(k-1)}\right]_i$.
  end for
  Set $[\boldsymbol{b}_k]_i = [\boldsymbol{b}_{k-1}]_i - \left[\boldsymbol{u}_l^{(k-1)}\right]_i$.
end for
1

Analogous to Algorithm 1, the distributed solver commences by running a backward loop to compute the i^{th} component of the crude solution $[\boldsymbol{x}_0]_i$. Using a similar analysis to that of the forward rule, the recursive update equations are represented in Algorithm 4. Algorithm 4 distributes the computations of the products involved in determining the i^{th} component of $\boldsymbol{x}_0$ using recursion. Furthermore, it uses the same chain length as that in Algorithm 3.

Algorithm 4 : **Backward Loop: Distributed "Crude" Solver**

Input: The i^{th} row of matrices $\boldsymbol{A}_0, \boldsymbol{D}_0$, the i^{th} component of vector $\boldsymbol{b}$, the length $d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\frac{w_{max}}{w_{min}}n^3\right)\rceil$ of approximated inverse chain, and $[\boldsymbol{b}_d]_i$ as returned by Algorithm 3.
Output: The i^{th} components of "crude" solution $[\boldsymbol{x}_0]_i$.
Set $[\boldsymbol{x}_d]_i = \frac{[\boldsymbol{b}_d]_i}{[\boldsymbol{D}_0]_{ii}}$
for $k = d-1$ to 1 **do**
  Set $l = 2^k$.
  Update $\left[\boldsymbol{\eta}_1^{(k+1)}\right]_i = \left[\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{x}_{k+1}\right]_i$.
  for $j = 2$ to l **do**
    Update $\left[\boldsymbol{\eta}_j^{(k+1)}\right]_i = \left[\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{\eta}_{j-1}^{(k+1)}\right]_i$.
  end for
  Set $[\boldsymbol{x}_k]_i = \frac{1}{2}\left[\frac{[\boldsymbol{b}_k]_i}{[\boldsymbol{D}_0]_{ii}} + [\boldsymbol{x}_{k+1}]_i + \left[\boldsymbol{\eta}_l^{(k+1)}\right]_i\right]$.
end for
Set $[\boldsymbol{x}_0]_i = \frac{1}{2}\left[\frac{[\boldsymbol{b}]_i}{[\boldsymbol{D}_0]_{ii}} + [\boldsymbol{x}_1]_i + \left[\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\boldsymbol{x}_1\right]_i\right]$

Having developed an algorithm which computes a crude approximation to an SDD system of linear equations, we now provide an exact distributed solver which can drive $\boldsymbol{x}_0$ to an ϵ-close solution for any arbitrary $\epsilon > 0$. Similar to the previous analysis, each node i receives the i^{th} row of $\boldsymbol{L}_{\mathbb{G}}$, the i^{th} component of the right-hand side vector $[\boldsymbol{b}]_i$, the length of the inverse chain d, and a precision parameter ϵ as inputs. The algorithm then determines the i^{th} component of the ϵ-close approximation to the real solution $\boldsymbol{x}^*$ as detailed in Algorithm 5. It should be noted that Algorithm 5 is a simple distributed implementation of the Exact SDD solver, where the products are computed locally based on the sparsity pattern of $\boldsymbol{L}_{\mathbb{G}}$.

Algorithm 5 : **"Exact" Distributed SDD Solver**

1: **Input:** The i^{th} row of matrices $\boldsymbol{A}_0, \boldsymbol{D}_0$, the i^{th} component of vector $\boldsymbol{b}$, precision parameter ϵ.
2: **Output:** The i^{th} components of $\epsilon-$ approximate solution $[\tilde{\boldsymbol{x}}]_i$.
3: **Initialize:** $[\boldsymbol{y}_0]_i = 0$ and $[\boldsymbol{\chi}]_i$ by running Algorithms 3 and 4 with $[\boldsymbol{A}_0]_{i1}, \ldots, [\boldsymbol{A}_0]_{in}, [\boldsymbol{D}_0]_{ii}, [\boldsymbol{b}]_i$ and d as inputs.
4: **for** $t = 1$ to $q = \mathcal{O}\left(\frac{1}{\epsilon}\right)$ **do**
5: Set $\left[\boldsymbol{u}_t^{(1)}\right]_i = \left[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_{t-1}\right]_i$.
6: Set $\left[\boldsymbol{u}_t^{(2)}\right]_i$ by running Algorithms 3 and 4 with $[\boldsymbol{A}_0]_{i1}, \ldots, [\boldsymbol{A}_0]_{in}, [\boldsymbol{D}_0]_{ii}, [\boldsymbol{u}_t^{(1)}]_i$ and d as inputs.
7: Update $[\boldsymbol{y}_t]_i = [\boldsymbol{y}_{t-1}]_i - \left[\boldsymbol{u}_t^{(2)}\right]_i + [\boldsymbol{\chi}]_i$.
8: **end for**
9: Set $[\tilde{\boldsymbol{x}}]_i = [\boldsymbol{y}_q]_i$

Having developed the solvers, we next illustrate the most important theoretical results attained by our distributed SDD solver and compare to current literature.

Theoretical Guarantees

In this section, we provide theoretical justification for the correctness of Algorithm 5. Namely, we show that the distributed solver is capable of acquiring ϵ-close approximations to the exact solution of the SDD system and provide its iteration count in terms of the network's properties. These results are summarized in the following theorem:

Theorem 2.2.1 *The distributed SDD solver described in Algorithm 5 uses local communication exchange to compute an ϵ-approximate solution of the SDD system $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$ in the following number of rounds*

$$\mathcal{O}\left(\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}\log\left(\frac{1}{\epsilon}\right)\right)$$

where $\kappa(\boldsymbol{L}_{\mathbb{G}})$ is condition number of $\mathbb{G}$, and w_{max}, w_{min} are maximal and minimum edge weights.

To arrive at such results, we require the analysis of both the crude and the exact distributed SDD solvers. The remainder of this section is dedicated to the proof of the above theorem. It proceeds by presenting two essential lemmas. The first shows that the distributed crude solver (i.e., Algorithms 3 and 4) returns a constant error approximation to the exact solution of the SDD system. The second demonstrates that the exact solver in Algorithm 5 drives the crude solution to an ϵ-close one and provides its iteration count.

To arrive at a crude approximation to the real solution of the SDD system, we need to show that the procedure described in Algorithms 3 and 4 is capable of approximating the inverse of $\boldsymbol{L}_{\mathbb{G}}$ and providing a good enough approximation to the exact solution. The accuracy of this approximation and the needed iteration count has also to be quantified. We summarize these results for the case of graph Laplacian $\boldsymbol{L}_{\mathbb{G}}$ in the following

Lemma 2.2.2 *Let $\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting. Let the length of the inverse chain is defined as $d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\frac{w_{max}}{w_{min}}n^3\right)\rceil$. Further, let $\boldsymbol{Z}'$ be the operator defined by the "crude" solver, such that $\boldsymbol{x}_0 = \boldsymbol{Z}'\boldsymbol{b}$. Then*

1. *$\epsilon_d < \frac{1}{3}\ln 2$.*
2. *$\boldsymbol{Z}' \approx_{\epsilon_d} \boldsymbol{L}_{\mathbb{G}}^{-1}$, and*
3. *$\mathcal{O}\left(2^d\right)$ rounds is required to arrive at the crude solution $\boldsymbol{x}_0$.*

The derived bounds depend on the length of the inverse approximate chain d. The choice of d has to be made in such a way to guarantee that $\epsilon_d \leq \frac{1}{3}\ln 2$. As mentioned in Lemma 2.2.2 a value satisfying the above condition is given by

$$d = \lceil \log\left(2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\frac{w_{max}}{w_{min}}n^3\right)\rceil, \;\; i.e., \boldsymbol{D}_0 \approx_{\epsilon_d} \boldsymbol{D}_0 - \boldsymbol{D}_0\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^d} \quad \text{with } \epsilon_d < \frac{1}{3}\ln 2$$

After attaining a crude solution to the SDD system, our strategy was the usage of the exact solver in Algorithm 5 to drive it to an ϵ-approximate one for any $\epsilon > 0$. In what comes next, we show the exact solver is capable of achieving such a solution.

Lemma 2.2.3 *Let $\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0$ be the standard splitting. If $\epsilon_d < \frac{1}{3}\ln 2$, then Algorithm 5*

requires $\mathcal{O}\left(\log \frac{1}{\epsilon}\right)$ *iterations to return the* i^{th} *component of the* $\epsilon-$*close solution to* $\boldsymbol{x}^*$ *and requires* $\mathcal{O}\left(2^d \log \frac{1}{\epsilon}\right)$ *rounds.*

Theorem 2.2.1 follows immediately from Lemmas 2.2.2 and 2.2.3. This result provides us with time complexity $\mathcal{O}\left(d_{max}\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}\log\left(\frac{1}{\epsilon}\right)\right)$ and message complexity $\mathcal{O}\left(m\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}\log\left(\frac{1}{\epsilon}\right)\right)$.

2.2.2. Algorithm II: Chebyshev polynomials Approach

Although Algorithm 5 is at $\log n$ factor faster than other distributed solvers (see Section 2.2.3), it suffers from linear dependency on condition number $\kappa(\boldsymbol{L}_{\mathbb{G}})$ for both time and message complexities. Therefore, the practical application of such technique is restricted by graphs with small condition numbers, such as expanders (Hoory et al. (2006)).

Next, we propose a novel distributed SDD solver that acquires an ϵ-close solution in time and message complexities sub-linearly dependent on the condition number of the processing graph. We achieve such a reduction by exploiting well-known polynomial representation of the inverse of the graph Laplacian. Our method aims at constructing a set of Chebyshev polynomials that reduce the differential to the optimal solution as quickly as possible.

Polynomial Representation.

Similar to the previous approach, we consider the same story of computing an approximation to the exact solution of an SDD system of linear equations that starts from standard splittings of symmetric matrices. Given a symmetric matrix, say $\boldsymbol{L}_{\mathbb{G}}$, the standard splitting is given by $\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0$. The authors in Peng and Spielman (2013) exploited the fact that the inverse of $\boldsymbol{L}_{\mathbb{G}}$ can be written as:

$$(\boldsymbol{D}_0 - \boldsymbol{A}_0)^{-1} = \boldsymbol{D}_0^{-\frac{1}{2}}\left[\boldsymbol{I} - \boldsymbol{D}_0^{-\frac{1}{2}}\boldsymbol{A}_0\boldsymbol{D}_0^{-\frac{1}{2}}\right]^{-1}\boldsymbol{D}_0^{-\frac{1}{2}} =$$

$$\boldsymbol{D}_0^{-\frac{1}{2}}\prod_{k\geq 0}\left(\boldsymbol{I} + \left[\boldsymbol{D}_0^{-\frac{1}{2}}\boldsymbol{A}_0\boldsymbol{D}_0^{-\frac{1}{2}}\right]^{2^k}\right)\boldsymbol{D}_0^{-\frac{1}{2}} \approx \boldsymbol{D}_0^{-\frac{1}{2}}\prod_{k=0}^{\mathcal{O}(\log T)}\left(\boldsymbol{I} + \left[\boldsymbol{D}_0^{-\frac{1}{2}}\boldsymbol{A}_0\boldsymbol{D}_0^{-\frac{1}{2}}\right]^{2^k}\right)\boldsymbol{D}_0^{-\frac{1}{2}} =$$

$$\hat{p}_T(\boldsymbol{L}_{\mathbb{G}}).$$

where $\hat{p}_T(\boldsymbol{L}_{\mathbb{G}})$ is a polynomial of degree $T = 2^d \sim \kappa(\boldsymbol{L}_{\mathbb{G}})$ where $\kappa(\boldsymbol{L}_{\mathbb{G}})$ is the condition number, chosen to guarantee accuracy properties of the approximate solution, $\tilde{\boldsymbol{x}} = \hat{p}_T(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{b}$, to $\boldsymbol{x}$ with respect to definition (2.1). Our strategy for proposing a distributed SDD solver that exhibits sub-linear time

and message dependencies on the condition number of the processing graph relies on determining a "better" polynomial expansion[2] for $(\boldsymbol{D}_0 - \boldsymbol{A}_0)^{-1}$ than $\hat{p}_T(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{b}$. Formally, our goal is to determine a solution vector in the following form:

$$\boldsymbol{x}_k = p_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{b} \tag{2.6}$$

where $p_k(\boldsymbol{L}_{\mathbb{G}})$ is a polynomial of degree k. Consequently, the differential between $\boldsymbol{x}_k - \boldsymbol{x}^*$ can be written as:

$$\boldsymbol{x}_k - \boldsymbol{x}^* = p_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{b} - \boldsymbol{x}^* = p_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x}^* - \boldsymbol{x}^* = (p_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{L}_{\mathbb{G}} - \boldsymbol{I})\boldsymbol{x}^* =$$
$$(\boldsymbol{L}_{\mathbb{G}}p_k(\boldsymbol{L}_{\mathbb{G}}) - \boldsymbol{I})\boldsymbol{x}^* = -q_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{x}^*$$

where $q_k(\boldsymbol{L}_{\mathbb{G}}) = -\boldsymbol{L}_{\mathbb{G}}p_k(\boldsymbol{L}_{\mathbb{G}}) + \boldsymbol{I}$. Notice, that between polynomials $q_k(\boldsymbol{L}_{\mathbb{G}})$ and $p_k(\boldsymbol{L}_{\mathbb{G}})$ there is one to one correspondence. Given $p_k(\boldsymbol{L}_{\mathbb{G}})$ we can easily construct $q_k(\boldsymbol{L}_{\mathbb{G}})$. On the other hand, for any degree k, $q_k(\boldsymbol{L}_{\mathbb{G}})$ polynomial, we can recover $p_k(\boldsymbol{L}_{\mathbb{G}})$ using[3]

$$p_k(\boldsymbol{L}_{\mathbb{G}}) = \boldsymbol{L}_{\mathbb{G}}^{-1}(\boldsymbol{I} - q_k(\boldsymbol{L}_{\mathbb{G}}))$$

Plugging the above result back in Equation 2.6, we arrive at the following representation for the solution $\boldsymbol{x}_k$:

$$\boldsymbol{x}_k = \boldsymbol{L}_{\mathbb{G}}^{-1}(\boldsymbol{I} - q_k(\boldsymbol{L}_{\mathbb{G}}))\boldsymbol{b} \tag{2.7}$$

Hence, we recognize that instead of seeking $p_k(\boldsymbol{L}_{\mathbb{G}})$, one can think of trying to construct polynomials $q_k(\boldsymbol{L}_{\mathbb{G}})$ that reduce the term $\boldsymbol{x}_k - \boldsymbol{x}^*$ as fast as possible. This intuition can be formalized in terms of the properties of $q_k(\boldsymbol{L}_{\mathbb{G}})$ by requiring the polynomial to have a minimal degree, as well as to satisfy the following two conditions for a given precision parameter ϵ

$$q_k(0) = 1 \tag{2.8}$$
$$|q_k(\mu_i)| \leq \epsilon \quad \text{for all} \quad i = 1, \ldots, p$$

[2] As shall be seen later, better here means a polynomial with a lower degree.

[3] Please note that for the case of singular $\boldsymbol{L}_{\mathbb{G}}$, we can safely replace $\boldsymbol{L}_{\mathbb{G}}^{-1}$ by the pseudo-inverse $\boldsymbol{L}_{\mathbb{G}}^{\dagger}$. This is true since in such a scenario $\boldsymbol{b} \in (\ker(\boldsymbol{L}_{\mathbb{G}}))^{\perp}$ and $\boldsymbol{L}_{\mathbb{G}}^{\dagger}\boldsymbol{L}_{\mathbb{G}}^{r}\boldsymbol{b} = \boldsymbol{L}_{\mathbb{G}}^{r-1}\boldsymbol{b}$ for $r = 1, \ldots, k$

with μ_i being the i^{th} nonzero eigenvalue of $\boldsymbol{L}_{\mathbb{G}}$. The first condition is a result of observing that $q_k(z) = -zp_k(z) + 1$ (analogous to $q_k(\boldsymbol{L}_{\mathbb{G}}) = -\boldsymbol{L}_{\mathbb{G}}p_k(\boldsymbol{L}_{\mathbb{G}}) + \boldsymbol{I}$ is unity when evaluated at $z = 0$. The second, on the other hand, guarantees an ϵ-approximate solution to $\boldsymbol{x}^*$:

$$||\boldsymbol{x}_k - \boldsymbol{x}^*||^2_{\boldsymbol{L}_{\mathbb{G}}} = ||q_k(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{x}^*||^2_{\boldsymbol{L}_{\mathbb{G}}} \leq ||q_k(\boldsymbol{L}_{\mathbb{G}})||^2_2||\boldsymbol{x}^*||^2_{\boldsymbol{L}_{\mathbb{G}}} = \max_i |q_k(\mu_i)|^2||\boldsymbol{x}^*||^2_{\boldsymbol{L}_{\mathbb{G}}} \leq \epsilon^2||\boldsymbol{x}^*||^2_{\boldsymbol{L}_{\mathbb{G}}}$$

In other words, finding $q_k(z)$ that has minimal degree and that satisfies the conditions in Equation 2.8 guarantees an efficient and an ϵ-approximate solution to $\boldsymbol{x}^*$.

Candidate Solutions & Chebyshev Polynomials.

Among other potential solutions, Chebyshev polynomials of the first kind resemble a good candidate for determining $q_k(z)$. These forms are defined as

$$T_k(z) = \begin{cases} \cos\left(k \arccos(z)\right), & \text{if } z \in [-1, 1] \\ \frac{1}{2}\left((z + \sqrt{z^2 - 1})^k + (z - \sqrt{z^2 - 1})^k\right), & \text{otherwise} \end{cases} \tag{2.9}$$

Interestingly, $T_k(z) \leq 1$ on $[-1, 1]$ and among all polynomials of degree k with a leading coefficient 1, the polynomial $T_k(z)$ acquires its sharpest increase outside the range $[-1, 1]$. At this stage, we are ready to consider $q_k(z)$ in terms of $T_k(s)$. We posit that a good candidate is $q^*_k(z)$, which we define as

$$q^*_k(z) = \frac{T_k\left(\frac{\mu_p + \mu_1 - 2z}{\mu_p - \mu_1}\right)}{T_k\left(\frac{\mu_p + \mu_1}{\mu_p - \mu_1}\right)} \tag{2.10}$$

Next, we will demonstrate that $q^*_k(z)$ is indeed a good candidate since it meets the requirements in Equation 2.8 and allows for an efficient distributed SDD solver. First, it is easy to see that the polynomial defined in Equation 2.10 attains a unity value when evaluated at $z = 0$ (i.e. $q^*_k(0) = 1$). As such the first condition of Equation 2.8 is met. When it comes to second, we further recognize

that for any $z \in [\mu_1, \mu_p]$, $|q_k^*(z)|^2$ is also bounded as

$$|q_k^*(z)|^2 \leq T_k^{-2}\left(\frac{\mu_p+\mu_1}{\mu_p-\mu_1}\right) = T_k^{-2}\left(\frac{\kappa(\boldsymbol{L}_\mathbb{G})+1}{\kappa(\boldsymbol{L}_\mathbb{G})-1}\right) =$$
$$4\left(\left(\frac{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1}{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}-1}\right)^k + \left(\frac{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}-1}{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1}\right)^k\right)^{-2} \leq 4\left(\frac{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}-1}{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1}\right)^{2k} \leq$$
$$4e^{-\frac{4k}{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1}}$$

where $\kappa(\boldsymbol{L}_\mathbb{G}) = \frac{\mu_p}{\mu_1}$ is a condition number of $\boldsymbol{L}_\mathbb{G}$. Therefore, for the solution vector $\tilde{\boldsymbol{x}}_k = \boldsymbol{L}_\mathbb{G}^{-1}(\boldsymbol{I} - q_k^*(\boldsymbol{L}_\mathbb{G}))\boldsymbol{b}$ the corresponding error is given as:

$$||\tilde{\boldsymbol{x}}_k - \boldsymbol{x}^*||^2_{\boldsymbol{L}_\mathbb{G}} \leq 4e^{-\frac{4k}{\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1}}||\boldsymbol{x}^*||^2_{\boldsymbol{L}_\mathbb{G}} \tag{2.11}$$

Hence, by choosing $k_\epsilon = \lceil \frac{1}{2}(\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})}+1)\ln\frac{2}{\epsilon}\rceil$ the solution vector

$$\tilde{\boldsymbol{x}}_{k_\epsilon} = \boldsymbol{L}_\mathbb{G}^{-1}\left[\frac{T_{k_\epsilon}\left(\frac{\mu_p+\mu_1}{\mu_p-\mu_1}\right)\boldsymbol{I} - T_{k_\epsilon}\left(\frac{((\mu_p+\mu_1)\boldsymbol{I}-2\boldsymbol{L}_\mathbb{G}}{\mu_p-\mu_1}\right)}{T_{k_\epsilon}\left(\frac{\mu_p+\mu_1}{\mu_p-\mu_1}\right)}\right]\boldsymbol{b} \tag{2.12}$$

satisfies the accuracy requirement (2.1).

Distributed Challenges & Solution.

Having met the requirements of Equation 2.8 and proposed an approximate solution $\tilde{\boldsymbol{x}}_k$, at this stage we are ready to commence with the distributed implementation of our solver. However, we recognize the following two challenges hindering its direct distributed implementation. First, we note that computing the minimum and maximum non-zero eigenvalues of $\boldsymbol{L}_\mathbb{G}$ requires global information. The second relates to the product with $\boldsymbol{L}_\mathbb{G}^{-1}$ needed in Equation 2.12 of $\tilde{\boldsymbol{x}}_{k_\epsilon}$. In this section, we detail the solutions to above two problems for the case when $\boldsymbol{L}_\mathbb{G}$ is graph Laplacian and derive our distributed SDD solver, which is used later to compute the Newton direction

1. **Parameters μ_1 and μ_p.** As clear from the previous section, our method requires the computation of the second-minimum and maximum eigenvalues of $\boldsymbol{L}_\mathbb{G}$. The computation of these, however, requires global information and hence are difficult to determine in a distributed fashion. As a substitute for the exact values of μ_1 and μ_p, one can use the well-known eigenvalue

bounds determined as

$$\mu_1 \geq \underline{\mu} = \frac{4}{n^2}$$
$$\mu_p \leq \bar{\mu} = 2n$$

2. **Multiplication on $\boldsymbol{L}_{\mathbb{G}}^{-1}$.** We start by noting that the second issue faced relates to the computational inefficiency when attempting to compute the coefficients of $T_{k_\epsilon}\left(\frac{\mu_p+\mu_1}{\mu_p-\mu_1}\right)\boldsymbol{I} - T_{k_\epsilon}\left(\frac{((\mu_p+\mu_1)\boldsymbol{I}-2\boldsymbol{L}_{\mathbb{G}}}{\mu_p-\mu_1}\right)$ where performing it naively will potentially lead to linear dependency on the condition number of the processing graph. To illustrate, let us, in fact, consider the naive approach by assuming that each node i has access to the following decomposition of $T_{k_\epsilon}(z)$:

$$T_{k_\epsilon}(z) = 1 + \alpha_1 z + \alpha_2 z^2 + \ldots + \alpha_{k_\epsilon} z^{k_\epsilon}$$

where $\alpha_1, \ldots, \alpha_{k_\epsilon}$ are coefficients one for each power of the polynomial. For ease of exposition, let us further denote

$$c_1 = \frac{\bar{\mu}+\underline{\mu}}{\bar{\mu}-\underline{\mu}} \qquad c_2 = \frac{2}{\bar{\mu}-\underline{\mu}}$$

Using the above, the numerator in Equation (2.12): can be written as

$$\boldsymbol{L}_{\mathbb{G}}^{-1}\left[T_{k_\epsilon}(c_1)\boldsymbol{I} - T_{k_\epsilon}(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})\right]\boldsymbol{b} = \boldsymbol{L}_{\mathbb{G}}^{-1}\left[\sum_{i=1}^{k_\epsilon}\alpha_i c_1^i\boldsymbol{I} - \sum_{i=1}^{k_\epsilon}\alpha_i(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})^i\right]\boldsymbol{b} =$$
$$\boldsymbol{L}_{\mathbb{G}}^{-1}\left[\sum_{i=1}^{k_\epsilon}\alpha_i\left[(c_1\boldsymbol{I})^i - (c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})^i\right]\right]\boldsymbol{b}$$

The first term (i.e $c_1^i\boldsymbol{I}$) is easily computable. The second, on the other hand, can be computed by rewriting the term $(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})^i$ explicitly in terms of $\boldsymbol{L}_{\mathbb{G}}$ for each node i. Unfortunately, this procedure is inefficient as it boils-down to a total of $)(k_\epsilon^2)$ of matrix vector multiplications of the form $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{u}$. Taking into account the expression for k_ϵ, we end up with an algorithm exhibiting linear dependency on the condition number $\kappa(\boldsymbol{L}_{\mathbb{G}})$. Instead, our goal is to show that $\tilde{\boldsymbol{x}}_{k_\epsilon}$ can be computed in fully distributed way in $\mathcal{O}(k_\epsilon)$ rounds. The crucial property for

us here is the recursive relation of Chebyshev polynomials:

$$\begin{aligned} T_0(z) &= 1, \\ T_1(z) &= z, \\ T_k(z) &= 2zT_{k-1}(z) - T_{k-2}(z) \end{aligned} \tag{2.13}$$

Denote by

$$\begin{aligned} \boldsymbol{\Delta}_k &= \boldsymbol{L}_{\mathbb{G}}^{-1}\left[T_k(c_1)\boldsymbol{I} - T_k(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})\right]\boldsymbol{b} \\ \boldsymbol{\Omega}_k &= T_k(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})\boldsymbol{b} \\ \Theta_k &= T_k(c_1) \end{aligned}$$

Therefore, the solution vector (2.12) can be written as $\tilde{\boldsymbol{x}}_{k_\epsilon} = \frac{\boldsymbol{\Delta}_{k_\epsilon}}{\Theta_{k_\epsilon}}$ and recursive relation gives:

$$\begin{aligned} \boldsymbol{\Delta}_k &= 2c_1\boldsymbol{\Delta}_{k-1} - \boldsymbol{\Delta}_{k-2} + 2c_2\boldsymbol{\Omega}_{k-1} \\ \boldsymbol{\Omega}_k &= 2(c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})\boldsymbol{\Omega}_{k-1} - \boldsymbol{\Omega}_{k-2} \\ \Theta_k &= 2c_1\Theta_{k-1} - \Theta_{k-2} \end{aligned} \tag{2.14}$$

with initials given by:

$$\begin{aligned} \boldsymbol{\Delta}_1 &= c_2\boldsymbol{b} & \boldsymbol{\Omega}_1 &= [c_1\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}}]\boldsymbol{b} & \Theta_1 &= c_1 \\ \boldsymbol{\Delta}_0 &= \boldsymbol{0} & \boldsymbol{\Omega}_0 &= \boldsymbol{b} & \Theta_0 &= 1 \end{aligned}$$

Algorithm 6 summarizes these results and provides a fully distributed computation of vector $\tilde{\boldsymbol{x}}_{k_\epsilon}$ in $\mathcal{O}(k_\epsilon)$ rounds. Clearly, lines 8-10 are executing relations (2.14) in a fully distributed way. Indeed, each matrix vector multiplication $(c_i\boldsymbol{I} - c_2\boldsymbol{L}_{\mathbb{G}})\boldsymbol{u}$ can be computed locally by a single message exchange between the neighboring nodes. Moreover, the total number of such multiplications is bounded by $\mathcal{O}(k_\epsilon)$ and this fact establishes the following

Theorem 2.2.4 *The distributed SDD solver described in Algorithm 6 uses local communication exchange to compute an ϵ-approximate solution of the SDD system $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$ in the following number*

of rounds

$$\mathcal{O}\left(\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}}\log\left(\frac{1}{\epsilon}\right)\right)=\mathcal{O}\left(\sqrt{nd_{max}diam(\mathbb{G})\frac{w_{max}}{w_{min}}}\log\left(\frac{1}{\epsilon}\right)\right)$$

where $\kappa(\boldsymbol{L}_{\mathbb{G}})$ *is condition number of* $\mathbb{G}$, $d_{max}, diam(\mathbb{G})$ *are its maximal degree and diameter and* w_{max}, w_{min} *are maximal and minimum edge weights.*

This result provides us with time complexity $\mathcal{O}\left(d_{max}\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}}\log\left(\frac{1}{\epsilon}\right)\right)$ and message complexity $\mathcal{O}\left(m\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})\frac{w_{max}}{w_{min}}}\log\left(\frac{1}{\epsilon}\right)\right)$.

Algorithm 6 : **Chebyshev SDD Solver**

1: **Input:** The i^{th} row of matrices $\boldsymbol{A}_0, \boldsymbol{D}_0$, the i^{th} component of vector $\boldsymbol{b}$, precision parameter ϵ.
2: **Output:** The i^{th} components of $\epsilon-$ approximate solution $[\tilde{\boldsymbol{x}}]_i$.
3: Set $\bar{\mu}=2n$, $\underline{\mu}=\frac{4}{n^2}$ and $c_1=\frac{\bar{\mu}+\underline{\mu}}{\bar{\mu}-\underline{\mu}}$, $c_2=\frac{2}{\bar{\mu}-\underline{\mu}}$, $\kappa(\boldsymbol{L}_{\mathbb{G}})=\frac{\bar{\mu}}{\underline{\mu}}$
4: $k_\epsilon=\lceil\frac{1}{2}(\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})}+1)\ln\frac{2}{\epsilon}\rceil$.
5: $[\boldsymbol{\Delta}_0]_i=0 \quad [\boldsymbol{\Omega}_0]_i=[\boldsymbol{b}]_i \quad \Theta_0=1$.
6: $[\boldsymbol{\Delta}_1]_i=c_2[\boldsymbol{b}]_i \quad [\boldsymbol{\Omega}_1]_i=[(c_1\boldsymbol{I}-c_2\boldsymbol{L}_{\mathbb{G}})\,\boldsymbol{b}]_i \quad \Theta_1=c_1$.
7: **for** $k=2$ to k_ϵ **do**
8: $\Theta_k=2c_1\Theta_{k-1}-\Theta_{k-2}$.
9: $[\boldsymbol{\Omega}_k]_i=[2(c_1\boldsymbol{I}-c_2\boldsymbol{L}_{\mathbb{G}})\boldsymbol{\Omega}_{k-1}]_i-[\boldsymbol{\Omega}_{k-2}]_i$.
10: $[\boldsymbol{\Delta}_k]_i=2c_1[\boldsymbol{\Delta}_{k-1}]_i-[\boldsymbol{\Delta}_{k-2}]_i+2c_2[\boldsymbol{\Omega}_{k-1}]_i$
11: **end for**
12: Set $[\tilde{\boldsymbol{x}}]_i=\frac{[\boldsymbol{\Delta}_{k_\epsilon}]_i}{\Theta_{k_\epsilon}}$

Adaptive Method

To finalize this section, we suggest a modification of Algorithm 6 allowing us to achieve faster implementation depending on the specific instance of the problem. One can notice, the second condition in (2.8) can be relaxed depending of the decomposition of vector $\boldsymbol{b}$ in the basis formed by eigenvectors of $\boldsymbol{L}_{\mathbb{G}}$. Indeed, assume that $\boldsymbol{b}\in\text{Span}\{\boldsymbol{u}_{p-r},\ldots,\boldsymbol{u}_p\}$. Then, one can simplify condition 2.8 as follows:

$$\begin{aligned}&q_k(0)=1\\&|q_k(\mu_i)|\leq\epsilon \quad \text{for all} \quad i=p-r,\ldots,p\end{aligned}$$

Here, the threshold r can be guessed using a binary search strategy and by testing the quality of the solution at each round. This intuition is formalized for the case of the unweighted graph Laplacian $\boldsymbol{L}_{\mathbb{G}}$ in Algorithm 7. The adjustment of vector $\boldsymbol{b}$ to the proper interval $[\mu_{p-r},\mu_p]$ is carried out in lines 10-11. In other words, once the condition $\delta\leq\epsilon\frac{\sqrt{2}}{n^2}b$ is reached, the corresponding vector $\tilde{\boldsymbol{x}}_{k_\epsilon}$

satisfies (2.2). Indeed, using $\mu_p \leq 2n$ and $\mu_1 \geq \frac{4}{n^2}$ gives:

$$\delta \leq \epsilon \frac{\sqrt{2}}{n^2} b \leq \epsilon \sqrt{\frac{\mu_1}{n\mu_p}} b \leq \epsilon \sqrt{\frac{\mu_1}{n\mu_p}} ||\boldsymbol{b}||_2 \leq \epsilon \sqrt{\frac{\mu_1}{n}} \sqrt{\boldsymbol{b}^\mathsf{T} \boldsymbol{L}_\mathbb{G}^\dagger \boldsymbol{b}}$$

Since $\boldsymbol{b}^\mathsf{T} \boldsymbol{L}_\mathbb{G}^\dagger \boldsymbol{b} = \boldsymbol{x}^{*\mathsf{T}} \boldsymbol{L}_\mathbb{G} \boldsymbol{x}^*$ and $||\tilde{\boldsymbol{x}}_{k_\epsilon} - \boldsymbol{x}^*||_{\boldsymbol{L}_\mathbb{G}} \leq \sqrt{\frac{n}{\mu_1}}\delta$:

$$||\tilde{\boldsymbol{x}}_{k_\epsilon} - \boldsymbol{x}^*||_{\boldsymbol{L}_\mathbb{G}}^2 \leq \frac{n}{\mu_1}\delta^2 \leq \epsilon^2 \boldsymbol{b}^\mathsf{T} \boldsymbol{L}_\mathbb{G}^\dagger \boldsymbol{b} = \epsilon^2 ||\boldsymbol{x}^*||_{\boldsymbol{L}_\mathbb{G}}^2$$

As for the running time, the t^{th} iteration of Algorithm 7 requires $\mathcal{O}\{d_{max} 2^{\frac{t}{2}} \ln \frac{2}{\epsilon} + diam(\mathbb{G})\}$ time steps. Consequently, in the worst case ($r = p - 1$) it consummates to $\mathcal{O}\{d_{max}\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})} \ln \frac{2}{\epsilon} + diam(\mathbb{G})\lceil \log n \rceil\}$. Similarly, the communication complexity in the worst case is given by $\mathcal{O}\left(m\sqrt{\kappa(\boldsymbol{L}_\mathbb{G})} \log\left(\frac{1}{\epsilon}\right)\right)$. One can notice, these characteristics are almost identical to the guarantees on Algorithm 6, though for special instances of vector $\boldsymbol{b}$ much faster performance can be attained. For example, if $\boldsymbol{b} \in \text{Span}\{\boldsymbol{u}_{\lfloor \frac{p}{2} \rfloor}, \ldots, \boldsymbol{u}_p\}$, then Algorithm 7 terminates in $\mathcal{O}(n \ln \frac{1}{\epsilon})$ time steps and utilizes $\mathcal{O}(m \ln \frac{1}{\epsilon})$ messages in total.

Algorithm 7 : **Adaptive Chebyshev SDD Solver**

Input: The i^{th} row of matrices $\boldsymbol{A}_0, \boldsymbol{D}_0$, the i^{th} component of vector $\boldsymbol{b}$, precision parameter ϵ.
Output: The i^{th} components of $\epsilon-$ approximate solution $[\tilde{\boldsymbol{x}}]_i$.
Set $\bar{\mu} = 2n$ and $t = 1$.
Compute $b = \max_i\{|[\boldsymbol{b}]_i|\}$ using maximum consensus protocol.
for $t = 1$ to $\lceil \log \frac{n^3}{2} \rceil$ **do**
    Set $\underline{\mu} = \frac{\bar{\mu}}{2^t}$ $c_1 = \frac{\bar{\mu}+\underline{\mu}}{\bar{\mu}-\underline{\mu}}$, $c_2 = \frac{2}{\bar{\mu}-\underline{\mu}}$, $k_\epsilon = \lceil \frac{1}{2}(\sqrt{\frac{\bar{\mu}}{\underline{\mu}}} + 1) \ln \frac{2}{\epsilon} \rceil$.
    Using (2.14) compute the i^{th} component of vector

$$\tilde{\boldsymbol{x}}_{k_\epsilon} = \boldsymbol{L}_\mathbb{G}^{-1} \left[\frac{T_{k_\epsilon}(c_1 \boldsymbol{I}) - T_{k_\epsilon}(c_1 \boldsymbol{I} - c_2 \boldsymbol{L}_\mathbb{G})}{T_{k_\epsilon}(c_1 \boldsymbol{I})} \right] \boldsymbol{b} \tag{2.15}$$

    Set $[\boldsymbol{\delta}]_i = [\boldsymbol{L}_\mathbb{G} \tilde{\boldsymbol{x}}_{k_\epsilon}]_i - [\boldsymbol{b}]_i$.
    Compute $\delta = \max_i\{|[\boldsymbol{\delta}]_i|\}$ using maximum consensus protocol.
    if $\delta > \epsilon \frac{\sqrt{2}}{n^2} b$ **then** $t = t + 1$
    else $[\tilde{\boldsymbol{x}}]_i = [\tilde{\boldsymbol{x}}_{k_\epsilon}]_i$ **break**.
end for

2.2.3. Comparisons to Existing Literature

Both our methods are faster than state-of-the-art methods used for iteratively solving linear systems. Typical linear methods, such as Jacobi iteration Axelsson (1994b), are guaranteed to converge if the matrix is strictly diagonally dominant. We proposed a distributed algorithm that generalizes this

setting, where it is guaranteed to converge in the SDD scenario. Furthermore, the time complexity of linear techniques is $\mathcal{O}(n^{1+\beta})\log n$. Hence, a case of strict diagonal dominance leading to a complexity of $\mathcal{O}(n^4)$ can easily be constructed. Our approaches, therefore, not only generalize the assumptions made by linear methods, but the first and second methods are also faster by a factor of $\log n$ and $n^{\frac{3}{2}}\log n$ respectively. Furthermore, such algorithms require additional iterations to perform decentralized vector norm computations. Contrary to these methods which lead to additional approximation errors to the real solution, our approach resolves these issues by eliminating the need for such a consensus framework.

In centralized solvers, on the other hand, nonlinear methods (e.g., conjugate gradient descent Kaasschieter (1989), Nocedal and Wright (2006), etc.) typically offer computational advantages over linear methods (e.g., Jacobi Iteration) for iteratively solving linear systems. These techniques, however, cannot be easily decentralized. For instance, the stopping criteria for nonlinear methods require the computation of weighted norms of residuals (e.g., $||\boldsymbol{p}||_{\boldsymbol{L}_{\mathbb{G}}}$ with $\boldsymbol{p}_k$ being the search direction at iteration k). Using the approach in Olshevsky (2014), this requires the calculation of the top singular value of $\boldsymbol{L}_{\mathbb{G}}$ which amounts to a power iteration on $\boldsymbol{L}_{\mathbb{G}}^{\frac{1}{2}}\boldsymbol{L}_{\mathbb{G}}^{\frac{1}{2}}$ leading to loss of sparsity. Furthermore, conjugate gradient methods require global computations of inner products.

Another existing technique to which we compare our results is the recent work in Lee et al. (2014), Mou et al. (2015). The authors consider a local and asynchronous solution for solving systems of linear equations. In their work, a complexity bound, for one component of the solution vector is derived. This amounts to $\mathcal{O}\left(\min\{d_{max}\epsilon^{\frac{\ln d_{max}}{\ln ||\boldsymbol{G}||_2}}, \frac{d_{max}n\ln\epsilon}{\ln||\boldsymbol{G}||_2}\}\right)$ with ϵ being the precision parameter, d_{max} the maximal degree of $\mathbb{G}$, and $\boldsymbol{G}$ is defined as $\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x}+\boldsymbol{z}$ which can be directly mapped to $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$. The relevant scenario to our work is when $\boldsymbol{L}_{\mathbb{G}}$ is positive semi-definite and $\boldsymbol{G}$ is symmetric. Here, the bound on the number of multiplications is given by $\mathcal{O}\left(\min\{d_{max}^{\frac{\kappa(\boldsymbol{L}_{\mathbb{G}})+1}{2}}\ln\frac{1}{\epsilon}, \frac{\kappa(\boldsymbol{L}_{\mathbb{G}})+1}{2}nd_{max}\ln\frac{1}{\epsilon}\}\right)$ with $\kappa(\boldsymbol{L}_{\mathbb{G}})$ being the condition number of $\boldsymbol{L}_{\mathbb{G}}$. In the general case, when the degree depends on the number of nodes (i.e., $d_{max} = d_{max}(n)$), the minimum in the above bound will be the result of the second term $\left(\frac{\kappa(\boldsymbol{L}_{\mathbb{G}})+1}{2}nd_{max}\ln\frac{1}{\epsilon}\right)$ leading to $\mathcal{O}(d_{max}(n)n\kappa(\boldsymbol{L}_{\mathbb{G}})\ln\frac{1}{\epsilon})$. Hence, in such a general setting, our techniques outperform Lee et al. (2014) by a factor of n and $n^{\frac{5}{2}}$ respectively.

Finally, Rebeschini and Tatikonda (2016) suggested a notably new approach for solving distributed Laplacian linear systems based on the message passing model. The key idea here was to establish

a connection between Laplacian linear systems and the min-sum algorithm which is a popular distributed routine to optimize a cost function that has a graph structure (Moallemi and Van Roy (2009), Moallemi and Van Roy (2010)). This was achieved by constructing a particular factor graph that encoded the topology of the underlying graph of processors defined by a given Laplacian linear system. As a result, authors suggested a general framework to analyze the convergence of the min-sum paradigm, which goes beyond the typical assumption of scaled-diagonal dominance. In particular, it was shown that the convergence rate of the algorithm for some classes of graphs, such as $\frac{d}{2}-$connected cycles and $\frac{d}{2}-$dimensional tori, where d is even, is characterized as $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$, where t is the iteration time. Using this empirical evidence it was shown that in these graphs, potentially under some additional assumptions on the problem inputs (depending on the norm used for the analysis), the min-sum algorithm could yield $\epsilon-$approximate solutions for both Laplacian linear systems with running time $\mathcal{O}(\frac{n}{\epsilon^2})$, i.e. linear in n. The obvious drawback of this complexity result is related to quadratic dependence on $\frac{1}{\epsilon}$ term preventing us from computing highly accurate solutions. In contrast to the message passing algorithm, the distributed SDD solvers presented in this thesis abide by $\log\frac{1}{\epsilon}$ dependence on the accuracy parameter. Indeed, using the fact that the diameter of any $d-$regular graph on n nodes is bounded by $\mathcal{O}(\frac{n}{d})$ for the fully distributed algorithm presented in section 2.2.2, the overall time complexity is given as $\mathcal{O}(n\log\frac{1}{\epsilon})$. In addition, we study the performance of the proposed SDD solvers for any arbitrary graph topology, while in Rebeschini and Tatikonda (2016) authors focused only on very special forms of $d-$regular graphs.

2.3. Mixed Method

In this section, we investigate a new approach for solving SDD systems based on mixing decentralized and centralized strategies. To illustrate, let us consider the following straightforward centralized procedure to solve system (2.1):

Algorithm 8 : **Naive Centralized Algorithm**

1: **Input** The i^{th} row of matrices $\boldsymbol{L}_{\mathbb{G}}$, the i^{th} component of vector $\boldsymbol{b}$, precision parameter ϵ.
2: **Output:** The i^{th} components of $\epsilon-$ approximate solution $[\tilde{\boldsymbol{x}}]_i$.
3: Call a standard leader election protocol to come at leader node

$$i_{leader} = \text{LeaderElection}(\mathbb{G})$$

4: Collect the whole matrix $\boldsymbol{L}_{\mathbb{G}}$ and vector $\boldsymbol{b}$ in node i_{leader}.
5: Find $\epsilon-$approximate solution $\tilde{\boldsymbol{x}}$ to (2.1) at node i_{leader}.
6: Distribute components $[\tilde{\boldsymbol{x}}]_i$ among the nodes in $\mathbb{G}$

This algorithm combines both centralized and decentralized approaches to solve system (2.1). Specifically, Algorithm 8 firstly identifies the leader node using any distributed leader election protocol (Raynal (2013)), and then implements one of the fast SDD solvers (Spielman and Teng (2006), Cohen et al. (2014) or Blelloch et al. (2011)) at this node. The following summarizes both pros and cons of this approach:

1. **Pros**: time and communication complexities of Algorithm 8 are given by $\tilde{\mathcal{O}}(m \log \frac{1}{\epsilon})$ and $\mathcal{O}(m)$, respectively.[4]

2. **Cons**: to store $\boldsymbol{L}_{\mathbb{G}}$ in the leader node the corresponding local memory size should be at least $\mathcal{O}(m)$.

Therefore, it is interesting to ask if we can design some mixed procedure that retains all strong traits of Algorithm 8, and, at the same time, has more moderate requirements concerning the local memory of the leader node. The following Theorem confirms the above:

Theorem 2.3.1 *Let $\mathbb{G}$ be a weighted connected graph with n nodes and m edges. Consider the distributed SDD system*

$$\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$$

associated with $\mathbb{G}$. Then, there is a randomized mixed algorithm that computes $\epsilon-$approximate solution of this system in $\tilde{\mathcal{O}}(n \log^2 \frac{1}{\epsilon})$ time steps and $\tilde{\mathcal{O}}(m \log^2 \frac{1}{\epsilon})$ messages in total. Moreover, the size of the local memory for the leader node is bounded by $\tilde{\mathcal{O}}(n)$.

Before commencing with the technicalities of the mixed technique, notice that the above theorem establishes the existence of a mixed algorithm that actually improves the benefits of Algorithm 8 rather than just preserving them. Indeed, the time complexity is advanced to $\tilde{\mathcal{O}}(n)$, while the message complexity is almost linear in m. Finally, the local memory requirement for the leader node is reduced to $\tilde{\mathcal{O}}(n)$, which almost replicates the bound on the local memory for fully distributed solvers.

Next, we introduce additional machinery needed to establish the mixed method.

[4]Please note that that $\tilde{\mathcal{O}}$ hides polylog(n) factors.

2.3.1. Spanners and Sparsifiers

A sparsifier $\mathbb{H}$ of the graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$ is its sparse sub-graph that is similar to $\mathbb{G}$ with respect to some useful measure. A variety of sparsifiers have been considered: cut sparsifiers, spanners, spectral sparsifiers, etc. The last play a crucial role in the new mixed approach. As such, we present its definition below:

Definition Let $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$ be a weighted graph and p be the path connecting two endpoints of an edge $e \in \mathbb{E}$. Then the stretch of e with respect to p is given by:

$$st_p(e) = \sum_{e' \in p} \frac{w_e}{w_{e'}}$$

Definition For a given weighted graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$ its sub-graph $\mathbb{H} = (\mathbb{V}, \mathbb{E}', \mathbb{W}')$ is called $\log n$-spanner for $\mathbb{G}$ if

$$stretch_{\mathbb{H}}(e) \leq 2 \log n$$

where $stretch_{\mathbb{H}}(e) = \min_{p \in \mathbb{H}} st_p(e)$

For two weighted graphs $\mathbb{G}_1 = (\mathbb{V}, \mathbb{E}, \mathbb{W}_1)$ and $\mathbb{G}_1 = (\mathbb{V}, \mathbb{E}, \mathbb{W}_2)$, denote

$$\mathbb{G}_1 + \mathbb{G}_2 = (\mathbb{V}, \mathbb{E}, \mathbb{W}_1 + \mathbb{W}_2)$$
$$\beta \mathbb{G}_1 = (\mathbb{V}, \mathbb{E}, \beta \mathbb{W}_1)$$

The following definition specifies the concept of $\log n$ spanners:

Definition Let $\mathbb{G}$ be a weighted graph and $\mathbb{H}_1, \ldots, \mathbb{H}_t$ be sub-graphs of $\mathbb{G}$ such that $\mathbb{H}_i$ is a $\log n-$spanner for the graph $\mathbb{G} - \sum_{i=1}^{i-1} \mathbb{H}_j$. Then sub-graph $\mathbb{H} = \sum_{i=1}^{t} \mathbb{H}_i$ is called a $t-$ bundle spanner of $\mathbb{G}$.

Finally, the concept of spectral sparsification induces the proximity measure between quadratic forms defined by Laplacians of two graphs:

Definition For a given weighted graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$ its sub-graph $\mathbb{H} = (\mathbb{V}, \mathbb{E}', \mathbb{W}')$ is called ϵ_1-spectral sparsifier if

$$(1 - \epsilon_1) \boldsymbol{x}^{\mathsf{T}} \boldsymbol{L}_{\mathbb{H}} \boldsymbol{x} \leq \boldsymbol{x}^{\mathsf{T}} \boldsymbol{L}_{\mathbb{G}} \boldsymbol{x} \leq (1 + \epsilon_1) \boldsymbol{x}^{\mathsf{T}} \boldsymbol{L}_{\mathbb{H}} \boldsymbol{x} \tag{2.16}$$

for all $\boldsymbol{x} \in \mathbb{R}^{|\mathbb{V}|}$

Algorithm 9 : **Decentralized Spectral Sparsification** Koutis (2014)

Input Graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{W})$, sparsification parameter ϵ_1.
Output: ϵ_1-spectral sparsifier $\mathbb{H} = (\mathbb{V}, \mathbb{E}', \mathbb{W}')$ with expected number of edges $|\mathbb{E}'| = \mathcal{O}\left(\frac{1}{\epsilon_1^2} n \log^{c_1} n\right)$, where c_1 is some constant.
Set $\mathbb{G}_0 = \mathbb{G}$ and $\beta = \frac{\epsilon_1}{\lceil \log n \rceil}$.
for $i = 1$ to $\lceil \log n \rceil$ **do**
    Compute $\frac{24 \log^2 n}{\beta^2}-$ bundle spanner $\hat{\mathbb{H}}$ for $\mathbb{G}_{i-1}$.
    Set $\tilde{\mathbb{G}} = \hat{\mathbb{H}}$
    for each edge $e \notin \hat{\mathbb{H}}$ **do**
        with probability $\frac{1}{4}$ add e to $\tilde{\mathbb{G}}$ with weight $4w_e$
    end for
    Set $\mathbb{G}_i = \tilde{\mathbb{G}}$.
end for
Set $\mathbb{H} = \mathbb{G}_{\lceil \log n \rceil}$

In Koutis (2014) the authors suggest a simple fully distributed algorithm for spectral graph sparsification. The key component of the algorithm is a decentralized computation of $\frac{24 \log^2 n}{\beta^2}-$ bundle spanner using the technique proposed by Baswana and Sen (2007). The following result from Koutis (2014) provides all necessary theoretical guarantees:

Theorem 2.3.2 *The output $\mathbb{H}$ of algorithm 9 on input $\mathbb{G}$ and parameter ϵ_1 satisfies (2.16) with high probability. The expected number edges in $\mathbb{H}$ is at most $\mathbb{O}(\frac{1}{\epsilon_1^2} n \log^6 n)$. In the synchronous distributed model, it can be implemented to run in $\mathcal{O}(\frac{1}{\epsilon_1^2} \log^7 n)$ rounds with $\mathcal{O}(\frac{1}{\epsilon_1^2} m \log^6 n)$ communication complexity, using messages of size $\mathcal{O}(\log n)$.*

2.3.2. Fast Mixed SDD Solver

Before we proceed to the mixed solver, we need to emphasize the problem with tuning the precision parameters for both the solver and the sparsifier routines. Notice that, although, $\boldsymbol{L}_{\mathbb{H}}$ is spectrally close to $\boldsymbol{L}_{\mathbb{H}}$ the corresponding precision parameter should be chosen in such a way to guarantee (2.2) for the exact solution of the original system (2.1). Second, the running time, communication complexity, as well as the expected number of edges in $\mathbb{H}$ have a quadratic dependence on $\frac{1}{\epsilon_1}$. Hence, an accurate spectral approximation of $\boldsymbol{L}_{\mathbb{G}}$ (corresponding to small values of ϵ_1) can blow up all three of these performance measures. Next, we present a new mixed solver (Algorithm 10) along with its theoretical analysis and performance guarantees.

To better understand the performance of this algorithm, we further analyze its time and communi-

Algorithm 10 : **Fast Mixed SDD Solver**

1: **Input:** The i^{th} row of matrices $\boldsymbol{L}_{\mathbb{G}}$, the i^{th} component of vector $\boldsymbol{b}$, precision parameter ϵ.
2: **Output:** The i^{th} components of $\epsilon-$ approximate solution $[\tilde{\boldsymbol{x}}]_i$.
3: Set sparsification value $\epsilon_1 = \frac{1}{16 \log \frac{1}{\epsilon} + 8 \log 12}$, precision value $\epsilon_2 = \frac{\epsilon^{16}}{12^8}$.
4: Build ϵ_1-spectral sparsifier $\mathbb{H} = (\mathbb{V}, \mathbb{E}', \mathbb{W})$ for $\mathbb{G}$ using Algorithm 9.
5: Call a standard leader election protocol to come at leader node

$$i_{leader} = \text{LeaderElection}(\mathbb{G})$$

6: Collect the whole matrix $\boldsymbol{L}_{\mathbb{H}}$ and vector $\boldsymbol{b}$ in node i_{leader}.
7: Call the fast centralized solver for the system $\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x} = \boldsymbol{b}$ in the leader node v_{leader} with precision parameter ϵ_2.
8: Distribute components $[\tilde{\boldsymbol{x}}]_i$ among the nodes in $\mathbb{G}$.

cation complexities. The following step-by-step analysis establishes these results:

1. In **line 4**, Algorithm 9 constructs in $\mathcal{O}\left(\log^7 n \log^2 \frac{1}{\epsilon}\right)$ rounds an ϵ_1-spectral sparsifier $\mathbb{H}$ with expected number of edges $\mathcal{O}\left(n \log^6 n \log^2 \frac{1}{\epsilon}\right)$. In each round, a node passes its adjacency list constant number of times. Therefore, the overall time complexity of the this line is $\mathcal{O}\left(n \log^7 n \log^2 \frac{1}{\epsilon}\right)$ and communication complexity is $\mathcal{O}\left(m \log^7 n \log^2 \frac{1}{\epsilon}\right)$. The length of each message is bounded by $\mathcal{O}(\log n)$.

2. In **line 5**, we use the standard leader election technique, which requires $\mathcal{O}(n)$ and $\mathcal{O}(m)$ time and message complexities, respectively.

3. In **line 6**, the simple edge by edge passing algorithm collects the topology of $\mathbb{H}$ in node i_{leader} in $\mathcal{O}\left(n \log^6 n \log^2 \frac{1}{\epsilon}\right)$ time steps and the same total amount of messages.

4. In **line 7**, we apply the fast SDD solver with precision parameter ϵ_2 to $\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x} = \boldsymbol{b}$ requires $\mathcal{O}\left(n \log^{6.5} n \log^3 \frac{1}{\epsilon}\right)$ time steps. No message exchange is needed here because all computations take place at the leader node.

5. In **line 8**, we broadcast the solution vector to other nodes requiring $\mathcal{O}(n)$ and $\mathcal{O}(m)$ time and message complexities, respectively.

Hence, the overall time complexity is given by $\mathcal{O}\left(\max\{n \log^7 n \log^2 \frac{1}{\epsilon},\ \ n \log^{6.5} n \log^3 \frac{1}{\epsilon}\}\right) = \tilde{\mathcal{O}}(n \log^3 \frac{1}{\epsilon})$ and the total communication complexity is bounded by $\mathcal{O}\left(m \log^7 n \log^2 \frac{1}{\epsilon}\right) = \tilde{\mathcal{O}}(m \log^2 \frac{1}{\epsilon})$. The length of each message is at most $\mathcal{O}(\log n)$. The last step is to prove the correctness of Algorithm 10 as well as to validate the choice of parameters ϵ_1 and ϵ_2, which we show next.

2.3.3. Theoretical Guarantees

Denote $\boldsymbol{x}^*_{\mathbb{G}}$ and $\boldsymbol{x}^*_{\mathbb{H}}$ as the exact solutions of systems $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x} = \boldsymbol{b}$ respectively, and $\tilde{\boldsymbol{x}}_{\mathbb{H}}$ be ϵ_2-approximate solution of $\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x} = \boldsymbol{b}$. Therefore, according to Definition 2.1:

$$(1-\epsilon_2)^2||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \leq (1+\epsilon_2)^2||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \tag{2.17}$$

Since $\mathbb{H}$ is ϵ_1-spectral sparsifier for $\mathbb{G}$:

$$(1-\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{G}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \leq (1+\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{G}}} \tag{2.18}$$

Combining (2.17) and (2.18) gives:

$$(1+\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{G}}} \geq (1-\epsilon_2)^2||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}}$$
$$(1-\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{G}}} \leq (1+\epsilon_2)^2||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}}$$

or

$$\frac{(1-\epsilon_2)^2}{1+\epsilon_1}||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{G}}} \leq \frac{(1+\epsilon_2)^2}{1-\epsilon_1}||\boldsymbol{x}^*_{\mathbb{H}}||^2_{\boldsymbol{L}_{\mathbb{H}}} \tag{2.19}$$

Fix some $\delta > 0$ and denote $\hat{\boldsymbol{L}}_{\mathbb{H}} = \boldsymbol{L}_{\mathbb{H}} + \delta\mathbf{1}\mathbf{1}^\mathsf{T}$ and $\hat{\boldsymbol{L}}_{\mathbb{G}} = \boldsymbol{L}_{\mathbb{G}} + \delta\mathbf{1}\mathbf{1}^\mathsf{T}$. Then for the inverses we have:

$$\hat{\boldsymbol{L}}^{-1}_{\mathbb{H}} = \boldsymbol{L}^{\dagger}_{\mathbb{H}} + \frac{1}{\delta}\mathbf{1}\mathbf{1}^\mathsf{T}$$
$$\hat{\boldsymbol{L}}^{-1}_{\mathbb{G}} = \boldsymbol{L}^{\dagger}_{\mathbb{G}} + \frac{1}{\delta}\mathbf{1}\mathbf{1}^\mathsf{T}$$

Moreover, for any $\boldsymbol{x} \in \mathbb{R}^n$:

$$(1-\epsilon_1)\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{G}}\boldsymbol{x} \leq \boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{H}}\boldsymbol{x} \leq (1+\epsilon_1)\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{G}}\boldsymbol{x} \tag{2.20}$$

Assuming $\epsilon_1 \leq \frac{1}{2}$ and denoting $\alpha = \ln(1+2\epsilon_1)$ gives:

$$e^{-\alpha}\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{G}}\boldsymbol{x} \leq \boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{H}}\boldsymbol{x} \leq e^{\alpha}\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{L}}_{\mathbb{G}}\boldsymbol{x}$$

In other words, $\hat{\boldsymbol{L}}_{\mathbb{H}} \approx_\alpha \hat{\boldsymbol{L}}_{\mathbb{G}}$, and consequently $\hat{\boldsymbol{L}}_{\mathbb{H}}^{-1} \approx_\alpha \hat{\boldsymbol{L}}_{\mathbb{G}}^{-1}$. Therefore, for any $\boldsymbol{x} \in \mathbb{R}^n$:

$$e^{-\alpha}\boldsymbol{x}^\mathsf{T}\left(\boldsymbol{L}_{\mathbb{G}}^\dagger + \frac{1}{\delta}\mathbf{1}\mathbf{1}^\mathsf{T}\right)\boldsymbol{x} \leq \boldsymbol{x}^\mathsf{T}\left(\boldsymbol{L}_{\mathbb{H}}^\dagger + \frac{1}{\delta}\mathbf{1}\mathbf{1}^\mathsf{T}\right)\boldsymbol{x} \leq e^{\alpha}\boldsymbol{x}^\mathsf{T}\left(\boldsymbol{L}_{\mathbb{G}}^\dagger + \frac{1}{\delta}\mathbf{1}\mathbf{1}^\mathsf{T}\right)\boldsymbol{x}$$

In particular, for $\boldsymbol{b} \in \mathbf{1}^\perp$:

$$e^{-\alpha}\boldsymbol{b}^\mathsf{T}\boldsymbol{L}_{\mathbb{G}}^\dagger\boldsymbol{b} \leq \boldsymbol{b}^\mathsf{T}\boldsymbol{L}_{\mathbb{H}}^\dagger\boldsymbol{b} \leq e^{\alpha}\boldsymbol{b}^\mathsf{T}\boldsymbol{L}_{\mathbb{G}}^\dagger\boldsymbol{b} \tag{2.21}$$

or equivalently, using $\boldsymbol{b}^\mathsf{T}\boldsymbol{L}_{\mathbb{G}}^\dagger\boldsymbol{b} = ||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2$ and $\boldsymbol{b}^\mathsf{T}\boldsymbol{L}_{\mathbb{H}}^\dagger\boldsymbol{b} = ||\boldsymbol{x}_{\mathbb{H}}^*||_{\boldsymbol{L}_{\mathbb{H}}}^2$

$$e^{-\alpha}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq ||\boldsymbol{x}_{\mathbb{H}}^*||_{\boldsymbol{L}_{\mathbb{H}}}^2 \leq e^{\alpha}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2$$

Hence, using the above result in (2.19) gives:

$$\frac{(1-\epsilon_2)^2}{1+\epsilon_1}e^{-\alpha}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq \frac{(1+\epsilon_2)^2}{1-\epsilon_1}e^{\alpha}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2$$

or equivalently,

$$\frac{(1-\epsilon_2)}{\sqrt{(1+\epsilon_1)(1+2\epsilon_1)}}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}} \leq (1+\epsilon_2)\sqrt{\frac{1+2\epsilon_1}{1-\epsilon_1}}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}} \tag{2.22}$$

Using $\epsilon_1 = -\frac{1}{\log\epsilon_0}$ and $\epsilon_2 = \epsilon_0$ equation (2.22) can be written as:

$$\underbrace{\frac{(1-\epsilon_0)}{\sqrt{\left(1-\frac{1}{\log\epsilon_0}\right)\left(1-\frac{2}{\log\epsilon_0}\right)}}}_{A}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}} \leq \underbrace{(1+\epsilon_0)\sqrt{\frac{1-\frac{2}{\log\epsilon_0}}{1+\frac{1}{\log\epsilon_0}}}}_{B}||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}$$

For $\epsilon_0 \leq \frac{1}{5}$: $A \geq 1 - 2\epsilon_0^{\frac{1}{8}}$ and $B \leq 1 + 2\epsilon_0^{\frac{1}{8}}$, hence:

$$\left(1-2\epsilon_0^{\frac{1}{8}}\right)||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}} \leq ||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}} \leq \left(1+2\epsilon_0^{\frac{1}{8}}\right)||\boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}} \tag{2.23}$$

Equation (2.23) allows us to evaluate the quality of the solution $\tilde{\boldsymbol{x}}_{\mathbb{H}}$:

$$||\tilde{\boldsymbol{x}}_{\mathbb{H}} - \boldsymbol{x}_{\mathbb{G}}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq \left[\left(1+2\epsilon_0^{\frac{1}{8}}\right)^2 + 1\right]||\boldsymbol{x}^*||_{\boldsymbol{L}_{\mathbb{G}}}^2 - 2\tilde{\boldsymbol{x}}_{\mathbb{H}}^\mathsf{T}\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x}_{\mathbb{G}}^* \tag{2.24}$$

To finalize the analysis the last term in (2.24) should be lower bounded. This can be achieved by applying results (2.18), (2.21) and (2.23):

$$
\begin{aligned}
&2\tilde{\boldsymbol{x}}_{\mathbb{H}}^{\mathsf{T}}\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x}_{\mathbb{G}}^{*} = 2\tilde{\boldsymbol{x}}_{\mathbb{H}}^{\mathsf{T}}\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x}_{\mathbb{H}}^{*} \geq \tilde{\boldsymbol{x}}_{\mathbb{H}}^{T}\boldsymbol{L}_{\mathbb{H}}\tilde{\boldsymbol{x}}_{\mathbb{H}} + (1-\epsilon_2^2)\boldsymbol{x}_{\mathbb{H}}^{*\mathsf{T}}\boldsymbol{L}_{\mathbb{H}}\boldsymbol{x}_{\mathbb{H}}^{*} = \\
&(1-\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}}^2 + (1-\epsilon_2^2)\boldsymbol{b}^{\mathsf{T}}\boldsymbol{L}_{\mathbb{H}}^{\dagger}\boldsymbol{b} \geq \\
&(1-\epsilon_1)||\tilde{\boldsymbol{x}}_{\mathbb{H}}||_{\boldsymbol{L}_{\mathbb{G}}}^2 + (1-\epsilon_2^2)e^{-\alpha}\boldsymbol{x}_{\mathbb{G}}^{*\mathsf{T}}\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x}_{\mathbb{G}} \geq \\
&(1-\epsilon_1)\left(1-2\epsilon_0^{\frac{1}{8}}\right)^2||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2 + \frac{(1-\epsilon_2^2)}{1+2\epsilon_1}||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2 = \\
&\left[\left(1+\frac{1}{\log\epsilon_0}\right)\left(1-2\epsilon_0^{\frac{1}{8}}\right)^2 + \frac{(1-\epsilon_0^2)}{\left(1-\frac{2}{\log\epsilon_0}\right)}\right]||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2
\end{aligned}
$$

For $\epsilon_0 \leq \frac{1}{5}$ we have $-\frac{2}{\log\epsilon_0 - 2} \leq \epsilon_0^{\frac{1}{8}}$ and $-\frac{1}{\log\epsilon_0} \leq \epsilon_0^{\frac{1}{8}}$. Applying this results to (2.24) and choosing $\epsilon_0 = \frac{\epsilon^{16}}{12^8}$ gives:

$$
\begin{aligned}
&||\tilde{\boldsymbol{x}}_{\mathbb{H}} - \boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq \\
&\left[\left(1+2\epsilon_0^{\frac{1}{8}}\right)^2 + 1 - \left(\left(1+\frac{1}{\log\epsilon_0}\right)\left(1-2\epsilon_0^{\frac{1}{8}}\right)^2 + \frac{(1-\epsilon_0^2)}{\left(1-\frac{2}{\log\epsilon_0}\right)}\right)\right]||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2 \leq \\
&12\epsilon_0^{\frac{1}{8}}||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2 = \epsilon^2||\boldsymbol{x}_{\mathbb{G}}^{*}||_{\boldsymbol{L}_{\mathbb{G}}}^2
\end{aligned}
\tag{2.25}
$$

In other words, $\tilde{\boldsymbol{x}}_{\mathbb{H}}$ is $\epsilon-$approximate solution of system $\boldsymbol{L}_{\mathbb{G}}\boldsymbol{x} = \boldsymbol{b}$. Notice, that the choice of parameters ϵ_1 and ϵ_2 allows to bound the time and message complexities of Algorithm 10 in terms of poly $\left(\log\left(\frac{1}{\epsilon}\right)\right)$ avoiding computational issues mentioned before.

Finally, the local memory requirement for the leader node is dictated by the necessity to store $\boldsymbol{L}_{\mathbb{H}}$ and it is characterized by the number of edges in the sparsifier $\mathbb{H}$, i.e. given by $\mathcal{O}\left(n\log^6 n\log^2\frac{1}{\epsilon}\right) = \tilde{\mathcal{O}}(n\log^2\frac{1}{\epsilon})$. Ignoring $\log n$ factors, this result almost replicates the worst case bound for local memory in fully distributed method, given by $\mathcal{O}(n)$.

2.4. Special Cases

To finalize the study of distributed solvers we illustrate the performance of the proposed techniques for different graph topologies. In particular, the following cases are considered: Path Graph $\mathbb{P}_n$, Grid Graph $\mathbb{GG}_{k\times l}$, Ring Graph $\mathbb{R}_n$, Star graph $\mathbb{S}_n$, Erdos-Renyi graph $\mathbb{ER}_{n,p}$, Scale Free Network

$\mathbb{SF}_n$ and Bar-Bell Graph $\mathbb{BBG}_n$ and Ramanujan Expander $\mathbb{REG}_{d,n}$. To bound the condition number of these graphs we use the following well-known bounds (Chung (1997), Mohar (1991)):

$$\mu_n(\boldsymbol{L}_{\mathbb{G}}) \leq 2d_{max} \quad \text{and} \quad \mu_2(\boldsymbol{L}_{\mathbb{G}}) \geq \frac{4}{ndiam(\mathbb{G})}$$

where $d_{max}, diam(\mathbb{G})$ are maximal degree and diameter of $\mathbb{G}$ respectively.

1. **Path graph $\mathbb{P}_n$.** In a path graph all nodes can be enumerated in such way that edges are given as $\mathbb{E} = \{(i, i+1)\}_{i=1}^{n-1}$. Figure 1 shows path graph $\mathbb{P}_8$:

Figure 1: Path graph $\mathbb{P}_8$

Exploiting that $diam(\mathbb{P}_n) = n - 1$ for the condition number of a path graph $\mathbb{P}_n$:

$$\kappa(\boldsymbol{L}_{\mathbb{P}_n}) \leq n^2$$

2. **Grid Graph $\mathbb{GG}_{k\times l}$:** Let $\mathbb{P}_k$,$\mathbb{P}_l$ be path graphs with k and l nodes respectively, such that $kl = n$. The *Cartesian product* of $\mathbb{P}_k \times \mathbb{P}_l$ is called a grid graph $\mathbb{SG}_{k\times l}$. In other words, the vertex set of $\mathbb{GG}_{k\times l}$ is given as a collection of ordered pairs $\{[i,j]\}_{i\in\mathbb{P}_k, j\in\mathbb{P}_l}$ and two vertices $[i_1, j_1]$, $[i_2, j_2]$ are adjacent if and only if either i_1 adjacent to i_2 in $\mathbb{P}_k$ or j_1 adjacent to j_2 in $\mathbb{P}_l$. The example of $\mathbb{GG}_{4\times 8}$ is presented in Figure 2:

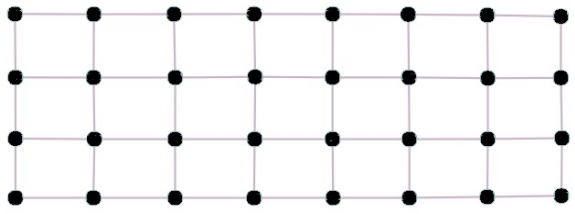


Figure 2: Grid Graph $\mathbb{GG}_{4\times 8}$.

Being the Cartesian product allows to associate the spectrum of $\mathbb{GG}_{k\times l}$ in terms of spectrum of $\mathbb{P}_k$ and $\mathbb{P}_l$. In particular, let λ_i, ν_j be eigenvalues of $\mathbb{P}_k$ and $\mathbb{P}_l$, then according to Mohar (1991) we have $\mu_{i,j} = \lambda_i + \nu_j$ is the eigenvalue of $\mathbb{GG}_{k\times l}$. The reverse is also true, i.e.

for any eigenvalue μ of graph $\mathbb{GG}_{k\times l}$ there are eigenvalues λ_i, ν_j of graphs $\mathbb{P}_k$ and $\mathbb{P}_l$ such that $\mu = \lambda_i + \nu_j$. Therefore, using that the spectrum of a path graph $\mathbb{P}_n$ is represented as $\{2 - 2\cos\frac{2\pi r}{n}\}_{r=1}^{\frac{n}{2}}$ then:

$$\kappa(\boldsymbol{L}_{\mathbb{GG}_{k\times l}}) \leq \frac{2}{\pi^2}\max\{k^2, l^2\}$$

In the case $k = l = \sqrt{n}$ this result implies $\kappa(\boldsymbol{L}_{\mathbb{GG}_{\sqrt{n}\times\sqrt{n}}}) \leq \frac{2}{\pi^2}n$

3. **Ring Graph $\mathbb{R}_n$:** A ring graph $\mathbb{R}_n$ consists of a sequence of n vertices starting and ending at the same vertex and each two consecutive nodes in the sequence adjacent to each other. The example of $\mathbb{R}_{10}$ is presented in Figure 3. Notice, the diameter of ring graph is given by $\frac{n}{2}$, therefore, the condition number can be bounded by:

$$\kappa(\boldsymbol{L}_{\mathbb{R}_n}) \leq \frac{n^2}{2}$$

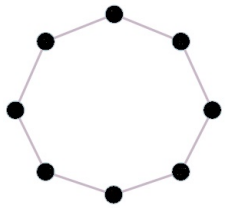


Figure 3: Ring graph $\mathbb{R}_8$

4. **Star Graph $\mathbb{S}_n$.** In a star graph the edge set is given as $\mathbb{E} = \{(1, j)\}_{j=2}^{n}$. In other words, vertices $2, \ldots, n$ are only connected to node 1. This node is called the central node. Figure 4 presents the star graph $\mathbb{S}_9$. The spectrum of Laplacian of $\mathbb{S}_n$ is given as:

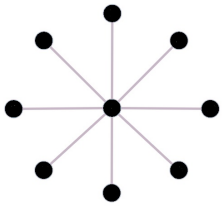


Figure 4: Star graph $\mathbb{S}_9$

$$Spect(\boldsymbol{L}_{\mathbb{S}_n}) = \{0, \underbrace{1, \ldots, 1}_{n-2}, n\}$$

Therefore, the condition number is equal to:

$$\kappa(\boldsymbol{L}_{\mathbb{S}_n}) = n$$

5. **Erdos-Renyi graph $\mathbb{ER}_{n,p}$.** In the model introduced by Erdos and Renyi (Erdos and Renyi (1959)), all graphs are constructed on a fixed vertex set $\mathbb{V} = \{1, \ldots, n\}$ where each edge has a fixed probability p of being present or absent, independently of the other edges. Figure 5 shows the instance of $\mathbb{ER}_{n,p}$ for $p = 0.1$ and $p = 0.2$:

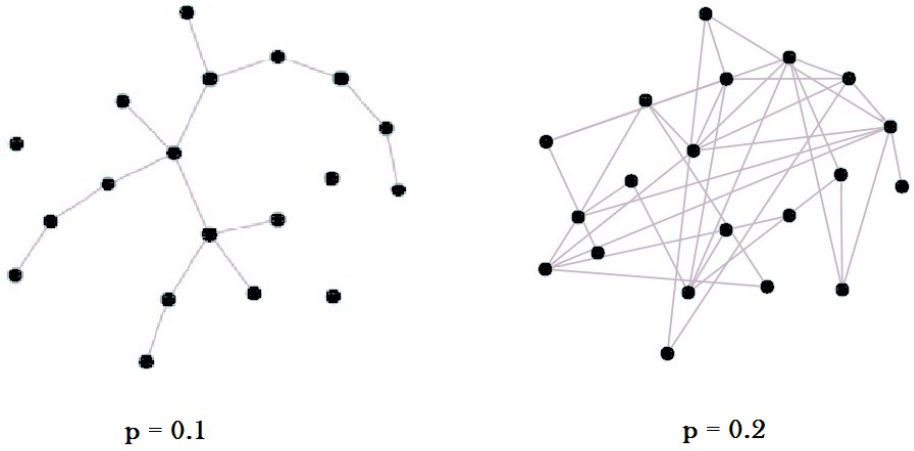


Figure 5: Erdos-Renyi graph $\mathbb{ER}_{n,p}$ with $p = 0.1$ and $p = 0.2$

The important features of Erdos-Renyi graphs strongly depend on a choice of parameter p. For instance, if $p \geq \frac{\log n}{n}$, then with high probability graph $\mathbb{ER}_{n,p}$ is an expander with a node degree $d \sim \mathcal{O}(\log n)$ and a diameter $diam(\mathbb{ER}_{n,p}) \sim \mathcal{O}(\log n)$. Consequently, one can bound the condition number of $\mathbb{ER}_{n,p}$ by:

$$\kappa(\boldsymbol{L}_{\mathbb{ER}_{n,p}}) \leq \frac{n}{2} \log^2 n$$

6. **Scale Free Network $\mathbb{SF}_n$:** This graph topology was first mentioned in de Solla Price (1965) by analyzing the network of scientific citations. Formally speaking, a scale free network is generated according to the following recursive procedure:

   (a) Start with $\mathbb{G}_{(1)}$ - the graph with one single node.

(b) Contract $\mathbb{G}_{(n)}$ from $\mathbb{G}_{(n-1)}$ by adding a node n with a single undirected edge from n to i, chosen according to $\mathbb{P}r(i = s) = \frac{d(s)_{\mathbb{G}_{(n-1)}}}{2n-1}$, with $d(s)_{\mathbb{G}_{(n-1)}}$ being the degree on node s in graph $\mathbb{G}_{(n-1)}$.

(c) $\mathbb{SF}_n = \mathbb{G}_{(n)}$.

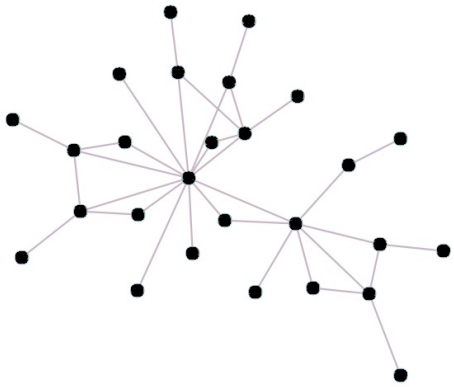


Figure 6: Scale Free Network $\mathbb{SF}_n$

Figure 6 presents the result of this process. In Bollobas et al. (2001) and Bollobas and Riordan (2004) the authors achieve the bounds for a maximal degree and a diameter of a scale free network. Precisely, with high probability

$$diam(\mathbb{SF}_n) \sim \mathcal{O}(\log n)$$
$$d_{max} \sim \mathcal{O}(\sqrt{n})$$

Hence, for the condition number we obtain the bound:

$$\kappa(\boldsymbol{L}_{\mathbb{SF}_n}) \leq \frac{n\sqrt{n}}{2} \log n$$

7. **Bar-Bell Graph $\mathbb{BBG}_n$:** Bar Bell graph with n nodes consist of two cliques $\mathbb{K}_{\lceil \frac{n}{3} \rceil}$ connected by a path graph $\mathbb{P}_{\lceil \frac{n}{3} \rceil}$. This topology is characterized by the following features:

$$diam(\mathbb{BBG}_n) \sim \mathcal{O}(n)$$
$$d_{max}(\mathbb{BBG}_n) \sim \mathcal{O}(n)$$

Therefore, for the condition number we have:

$$\kappa(\mathbb{BBG}_n) \leq \frac{n^3}{2}$$

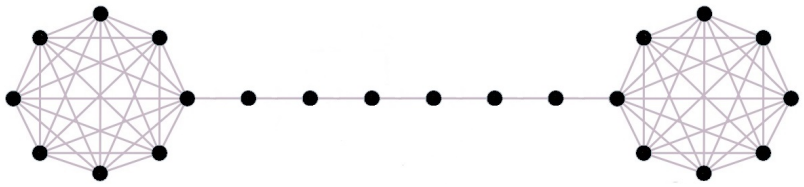


Figure 7: Bar Bell Graph $\mathbb{BBG}_8$

8. **Ramanujan Expander Graph $\mathbb{REG}_{d,n}$:** Let $d \geq 3$ and consider $d-$ regular graph $\mathbb{G}$ on n nodes. Denote $\{\lambda_i\}_{i=1}^n$ be the collection of eigenvalues of adjacency matrix $\boldsymbol{A}_{\mathbb{G}}$ of $\mathbb{G}$ arranged in decreasing order, i.e $d = \lambda_0 > \lambda_1 \geq \ldots \geq \lambda_{n-1} \geq -d$. The graph $\mathbb{G}$ is called Ramanujan Expander graph if

$$\max_{i:|\lambda_i| \neq d} |\lambda_i| \leq 2\sqrt{d-1}$$

There is a simple connection between eigenvalues of Laplacian and adjacency matrix of $\mathbb{REG}_{d,n}$. Indeed, using $\boldsymbol{L}_{\mathbb{REG}_{d,n}} = d\boldsymbol{I} - \boldsymbol{A}_{\mathbb{REG}_{d,n}}$:

$$\mu_i(\boldsymbol{L}_{\mathbb{REG}_{d,n}}) = d - \lambda_i$$

Therefore, $\mu_2(\boldsymbol{L}_{\mathbb{REG}_{d,n}}) = d - \lambda_1$, $\mu_n(\boldsymbol{L}_{\mathbb{REG}_{d,n}}) = d - \lambda_{n-1}$, and for the condition number:

$$\kappa(\boldsymbol{L}_{\mathbb{REG}_{d,n}}) = \frac{d - \lambda_{n-1}}{d - \lambda_1} \leq \frac{2d}{d - \lambda_1} \leq \frac{2d}{d - 2\sqrt{d-1}}$$

Figure 8 shows the example of $\mathbb{REG}_{3,20}$:

It is worth mentioning, that the explicit construction of such graphs for a fixed d and $n \to \infty$ has only been described in the case $d-1$ is prime Lubotzky et al. (1988), Margulis (1988) or a prime power Morgenstern (1994). In the recent work by Cohen (2016), the authors proposed the polynomial algorithm for constructing bipartite Ramanujan graphs of all degrees and all sizes. The general construction of a Ramanujan graph for non-bipartite case is still an open problem.

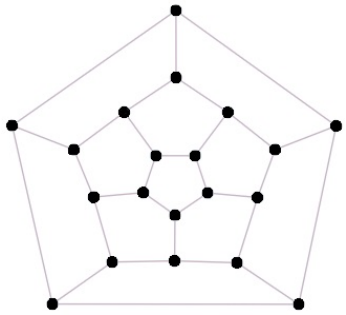


Figure 8: Ramanujan Expander Graph $\mathbb{REG}_{3,20}$

Time and message complexity results are presented in Tables[5] 1 and 2

$\mathbb{G}$	$\kappa(\boldsymbol{L}_{\mathbb{G}})$	**Alg 5**	**Alg 6**	**Alg 10**
$\mathbb{P}_n$	n^2	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
$\mathbb{GG}_{\sqrt{n}\times\sqrt{n}}$	$\frac{2}{\pi^2}n$	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(n)$
$\mathbb{R}_n$	$\frac{n^2}{2}$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
$\mathbb{S}_n$	n	$\mathcal{O}(n^2)$	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n)$
$\mathbb{ER}_{n,p}$	$\frac{n}{2}\log^2 n$	$\mathcal{O}(\frac{n}{2}\log^3 n)$	$\mathcal{O}(\sqrt{n}\log^2 n)$	$\mathcal{O}(n\log n)$
$\mathbb{SF}_n$	$\frac{n\sqrt{n}}{2}\log n$	$\mathcal{O}(n^2\log n)$	$\mathcal{O}(n^{\frac{5}{4}}\sqrt{\log n})$	$\mathcal{O}(n)$
$\mathbb{BBG}_n$	$\frac{n^3}{2}$	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{\frac{5}{2}})$	$\tilde{\mathcal{O}}(n)$
$\mathbb{REG}_{d,n}$	$\frac{2d}{d-2\sqrt{d-1}}$	$\mathcal{O}\left(\frac{d^2}{d-2\sqrt{d-1}}\right)$	$\mathcal{O}\left(d\sqrt{\frac{d}{d-2\sqrt{d-1}}}\right)$	$\tilde{\mathcal{O}}(n)$

Table 1: Time Complexity For Different Graph Topologies

$\mathbb{G}$	$\kappa(\boldsymbol{L}_{\mathbb{G}})$	**Alg 5**	**Alg 6**	**Alg 10**
$\mathbb{P}_n$	n^2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
$\mathbb{GG}_{\sqrt{n}\times\sqrt{n}}$	$\frac{2}{\pi^2}n$	$\mathcal{O}(n^2)$	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n)$
$\mathbb{R}_n$	$\frac{n^2}{2}$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
$\mathbb{S}_n$	n	$\mathcal{O}(n^2)$	$\mathcal{O}(n\sqrt{n})$	$\mathcal{O}(n)$
$\mathbb{ER}_{n,p}$	$\frac{n}{2}\log^2 n$	$\mathcal{O}(n^2\log^3 n)$	$\mathcal{O}(n\sqrt{n}\log^2 n)$	$\mathcal{O}(n\log^{\frac{3}{2}} n)$
$\mathbb{SF}_n$	$\frac{n\sqrt{n}}{2}\log n$	$\mathcal{O}(n^3\log n)$	$\mathcal{O}(n^{\frac{9}{4}}\sqrt{\log n})$	$\mathcal{O}(n)$
$\mathbb{BBG}_n$	$\frac{n^3}{2}$	$\mathcal{O}(n^5)$	$\mathcal{O}(n^{\frac{9}{2}})$	$\tilde{\mathcal{O}}(n^2)$
$\mathbb{REG}_{d,n}$	$\frac{2d}{d-2\sqrt{d-1}}$	$\mathcal{O}\left(\frac{d^2n}{d-2\sqrt{d-1}}\right)$	$\mathcal{O}\left(dn\sqrt{\frac{d}{d-2\sqrt{d-1}}}\right)$	$\tilde{\mathcal{O}}(dn)$

Table 2: Message Complexity For Different Graph Topologies

[5]For sparse graphs with $|\mathbb{E}| \sim \tilde{\mathcal{O}}(n)$ the construction of sparsifier in Alg 10 is unnecessary

CHAPTER 3 : APPLICATION OF SDD SOLVERS FOR LARGE SCALE OPTIMIZATION

In the second part of this work we discuss two practical applications of the proposed SDD solvers: *Network Flow Problem* and *Empirical Risk Minimization*. Particularly, we develop a distributed version of an exact Newton method for these problems and establish super-linear convergence.

3.1. Network Flow Optimization

Network flow optimization is a fundamental problem with wide-ranging applicability including but not limited to DNA sequence alignment Ahuja et al. (1993a), scheduling on uniform parallel machines Lawler et al. (1982), urban traffic flows Ahuja et al. (1993a), optimal energy allocation Gurakan et al. (2015), etc. As networks grow larger, centralized approaches to network flow optimization underperform due to the increase in time and resource complexity needed. Distributed methods for such network optimization problems present an alternative direction to cope with such increased demand. Conventional methods for distributed network optimization are based on subgradient descent in either the primal or dual domains. For a large class of problems, these techniques yield iterations that can be implemented in a distributed fashion using only local information. Their applicability, however, is limited by increasingly slow convergence rates. The Newton method Boyd and Vandenberghe (2004b) is known to overcome this limitation leading to improved convergence rates.
However, computing exact Newton directions based only on local information is challenging due to the need to invert the Hessian of the dual (distributed among the nodes of a graph) which typically requires global information. Consequently, authors in Jadbabaie et al. (2009), Wei et al. (2010) proposed approximate algorithms for determining these Newton iterates in a distributed fashion. Accelerated Dual Descent (ADD) Jadbabaie et al. (2009), for instance, exploits the fact that the dual Hessian is the weighted Laplacian of the network and uses a truncated Neumann expansion of the inverse to determine an approximation to the Newton step. ADD allows for a trade-off between accurate Hessian approximations and communication costs through the N-Hop design, where increased N allows for more accurate inverse approximations at the expense of increased cost, and lower values of N reduce accuracy but improve computational times. However, the effectiveness of these approaches highly depends on the accuracy of the truncated Hessian inverse which is used to approximate the Newton step. In fact, as we will show, the approximation error can be large,

leading to an inaccurate computation of the Newton step.

3.1.1. Problem Formulation

In this section, we formalize the Network flow problem and introduce essential results which enable distributed computation of the Newton direction. Crucially, we show the Hessian of the dual function to be a SDD matrix. Consequently, proposed SDD solvers can be used to compute the Newton direction in a distributed fashion up to any precision.

We consider a network represented by a directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with node set $\mathbb{V} = \{1, \ldots, n\}$ and edge set $\mathbb{E} = \{1, \ldots, E\}$. The flow vector is denoted by $\boldsymbol{x} = [x^{(e)}]_{e \in \mathbb{E}}$, where $x^{(e)}$ represents the flow on edge e. Flow conservation constraints are compactly represented by

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{A}$ is $n \times E$ node-edge incidence matrix of network $\mathbb{G}$ which is defined as

$$\boldsymbol{A}_{i,j} = \begin{cases} -1 & : \text{if edge } j \text{ leaves node } i \\ 1 & : \text{if edge } j \text{ enters node } i \\ 0 & : otherwise \end{cases}$$

Vector $\boldsymbol{b} \in \mathbf{1}^{\perp}$ denotes total sources. Consequently, $[\boldsymbol{b}]_i > 0$ (or $[\boldsymbol{b}]_i < 0$) indicates $[\boldsymbol{b}]_i$ units of external flow enters (or leaves) node i. The goal of the network flow is to minimize a sum of costs at all edges. Hence, we define $\Phi_e : \mathbb{R} \to \mathbb{R}^+$ to be the cost associated with each edge $e \in \mathbb{E}$. Here, $\Phi_e(x^{(e)})$ denotes the cost on edge e evaluated at the e^{th} edge flow $x^{(e)}$. We assume that these functions are strictly convex and twice differentiable. Given the above, the minimum cost network optimization problem can be written as

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{e=1}^{E} \Phi_e \left(x^{(e)} \right) \tag{3.1}$$

$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

3.1.2. Newton Method for Dual Descent

Typical methods for optimizing the problem in Equation (3.1) are based on descending in the dual function. The main difference among these techniques depends on the descent direction. Dual subgradients, for example, optimize the above objective by taking sub-gradient steps in the dual. Though successful, the applicability of sub-gradient descent is hindered by slow convergence rates. This motivates the consideration of Newton-type methods, which descend along a scaled version of the sub-gradient.

To formulate these iteration schemes, we start by defining the Lagrangian, $\mathscr{L}(\cdot) : \mathbb{R}^E \times \mathbb{R}^N \to \mathbb{R}$:

$$\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{e=1}^{E} \Phi_e(x^{(e)}) + \boldsymbol{\lambda}^{\mathsf{T}}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

where $\boldsymbol{\lambda}$ is a dual vector with each component being associated to a node. The dual function $q(\lambda)$ is then derived as

$$q(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}\in\mathbb{R}^E} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \inf_{\boldsymbol{x}\in\mathbb{R}^E} \left(\Phi_e(x^{(e)}) + \boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{x}\right) - \boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{b} =$$
$$\sum_{e=1}^{E} \inf_{x^{(e)}} \left(\Phi_e(x^{(e)}) + (\boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{A})^e x^{(e)}\right) - \boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{b}$$

Hence, it can be clearly seen that the evaluation of the dual function $q(\boldsymbol{\lambda})$ decomposes into E one-dimensional optimization problems. We assume that each of these optimization problems has a unique optimal solution which is guaranteed by the strict convexity of the costs Φ_e. Denoting the solutions by $x^{(e)}(\boldsymbol{\lambda})$ and using the first order optimality conditions, it can be seen that for each edge e, $x^{(e)}(\boldsymbol{\lambda})$ is given by

$$x^{(e)}(\boldsymbol{\lambda}) = [\dot{\Phi}_e]^{-1}([\boldsymbol{\lambda}]_j - [\boldsymbol{\lambda}]_i) \tag{3.2}$$

where $i, j \in \mathbb{V}$ denote the source and destiny nodes of edge $e = (i, j)$; see Zargham et al. (2013) for detailed description of the above derivation. Therefore, for an edge e, the evaluation of $x^{(e)}(\boldsymbol{\lambda})$ can be performed based on local information about the edge's cost function and the dual variables of the incident nodes, i and j.

Typically, the goal is to maximize the dual in terms of the dual variables. In this work, we consider

the minimization of the negative of the dual leading to:

$$\max_{\boldsymbol{\lambda}\in\mathbb{R}^n} q(\boldsymbol{\lambda}) \Longleftrightarrow \min_{\boldsymbol{\lambda}\in\mathbb{R}^n} \tilde{q}(\boldsymbol{\lambda}) \quad \text{where} \quad \tilde{q}(\boldsymbol{\lambda}) = -q(\boldsymbol{\lambda})$$

To minimize $\tilde{q}(\boldsymbol{\lambda})$, a second order method which descends along a scaled version of the gradient can be used. Such iterates are given by:

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \alpha_k \boldsymbol{d}_k, \quad \text{for all } k \geq 0 \tag{3.3}$$

with $\boldsymbol{d}_k$ being the Newton direction at iteration k, and α_k denoting the step size. The Newton direction is computed as the solution to the following linear system of equations:

$$\boldsymbol{H}_k \boldsymbol{d}_k = -\boldsymbol{g}_k \tag{3.4}$$

with $\boldsymbol{H}_k = \boldsymbol{H}(\boldsymbol{\lambda}_k) = \nabla^2 \tilde{q}(\boldsymbol{\lambda})$ being the Hessian of $\tilde{q}(\boldsymbol{\lambda})$ and $\boldsymbol{g}_k = \boldsymbol{g}(\boldsymbol{\lambda}_k) = \nabla \tilde{q}(\boldsymbol{\lambda}_k)$ denoting the gradient, both evaluated at $\boldsymbol{\lambda} = \boldsymbol{\lambda}_k$.

3.1.3. Distributed Newton Method For Network Flow Problem

The goal of this chapter is to propose a method which can compute the Newton direction for network flow problems in a distributed fashion. Our computational restriction is that each node, i, can compute the i^{th} component of the sought direction based on only local communication exchange with its neighbors. Namely, given the i^{th} row of $\boldsymbol{H}_k$ and the i^{th} entry of $\boldsymbol{g}_k$, the node has to determine the i^{th} component, $[\boldsymbol{d}_k]_i$, of $\boldsymbol{d}_k$. If such a solution can be attained, the iteration scheme in Equation (3.3) can be fully distributed across all nodes as:

$$[\boldsymbol{\lambda}_{k+1}]_i = [\boldsymbol{\lambda}_k]_i + \alpha_k [\boldsymbol{d}_k]_i$$

Having updated the dual variables, each node can perform local primal updates based on Equation (3.27), which is in itself distributable across the nodes of the network. To avoid conflicts, we adopt a strategy where each node i updates the flows on all of its outgoing edges based on the dual variables of its neighbors.

A distributed computational procedure as detailed above requires the dual gradient and Hessian to

share properties enabling decentralization. In Lemma 3.1.2, we show two such results. First, we demonstrate that the gradient is readily distributable across the network. Secondly, we prove the Hessian to be an SDD matrix, which we then exploit to distribute the computation of the Newton direction. Our results in Lemma 3.1.2 are based on the following assumptions:

Assumption 3.1.1 *The cost functions, $\Phi_e(\cdot)$, in Equation (3.1) are*

1. *twice continuously differentiable, i.e., $\gamma \leq \ddot{\Phi}_e \leq \Gamma$, with γ and Γ are constants; and*
2. *Hessian Lipschitz continuous for all edges $e \in \mathbb{E}$, i.e., $\left|\ddot{\Phi}_e(x) - \ddot{\Phi}_e(\hat{x})\right| \leq \delta|x - \hat{x}|$ for all $x, \hat{x} \in \mathbb{R}$*

Lemma 3.1.2 *The function $\tilde{q}(\boldsymbol{\lambda})$ abides by the following two properties:*

1. *The gradient and the Hessian of $\tilde{q}(\boldsymbol{\lambda})$ is given by*

$$\nabla\tilde{q}(\boldsymbol{\lambda}) = \boldsymbol{g}(\boldsymbol{\lambda}) = -\boldsymbol{A}\boldsymbol{x}(\boldsymbol{\lambda}) + \boldsymbol{b}$$
$$\nabla^2\tilde{q}(\boldsymbol{\lambda}) = \boldsymbol{H}(\boldsymbol{\lambda}) = \boldsymbol{A}\left[\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))\right]^{-1}\boldsymbol{A}^{\mathsf{T}}$$

2. *Denote $\mu_n(\boldsymbol{L}_{\mathbb{G}})$ be the largest eigenvalue of unweighted Laplacian of $\mathbb{G}$. Then, for constant $B = \frac{\delta}{\gamma^3}\mu_n(\boldsymbol{L}_{\mathbb{G}})$ and for any $\bar{\boldsymbol{\lambda}}, \boldsymbol{\lambda} \in \mathbb{R}^N$:*

$$||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq B||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2$$

   i.e. $\boldsymbol{H}(\boldsymbol{\lambda})$ is Lipschitz continuous with constant B.

The first result in Lemma 3.1.2 shows that the gradient can be distributed across the nodes of $\mathbb{G}$ which is true due to the sparsity pattern of the incidence matrix $\boldsymbol{A}$. Namely, the i^{th} element, $[\boldsymbol{g}_k]_i$, of the gradient $\boldsymbol{g}(\boldsymbol{\lambda})$ at iteration k can be computed as:

$$[\boldsymbol{g}_k]_i = -\sum_{e=(i,j)} x^{(e)}(\boldsymbol{\lambda}_k) + \sum_{e=(j,i)} x^{(e)}(\boldsymbol{\lambda}_k) + [\boldsymbol{b}]_i \tag{3.5}$$

The second result demonstrates that the Hessian is the weighted Laplacian of the graph $\mathbb{G}$ with the i^{th} row containing the weights of the edges corresponding to node i. Consequently, a distributed

solution as described above is achievable as long as the pseudo inverse of the Hessian can be computed locally. Being the weighted Laplacian of the graph, it is easy to see that the Hessian is also a SDD matrix, where for any $\boldsymbol{\lambda}$:

$$\boldsymbol{H}(\boldsymbol{\lambda})_{ii} \geq -\sum_{j \neq i} \boldsymbol{H}(\boldsymbol{\lambda})_{ij}$$

Consequently, all our solvers developed in the previous section are readily applicable to the computation of $\epsilon-$approximate solution to the exact Newton direction $\boldsymbol{d}_k$. Formally, we consider the following iteration scheme

$$[\boldsymbol{\lambda}_{k+1}]_i = [\boldsymbol{\lambda}_k]_i + \alpha_k [\tilde{\boldsymbol{d}}_k]_i \tag{3.6}$$

where $[\tilde{\boldsymbol{d}}_k]_i$ is the i^{th} component of the approximation to the real Newton direction $\boldsymbol{d}_k$ and α_k is a step size. The following Lemma studies the change of the norm of dual gradient for the scheme (3.6) and plays a crucial role for the convergence analysis:

Lemma 3.1.3 *Let us consider iteration scheme given by (3.6) and denote*

$$\boldsymbol{\epsilon}_k = \boldsymbol{H}_k \tilde{\boldsymbol{d}}_k + \boldsymbol{g}_k$$

be the approximation error vector corresponding to $\tilde{\boldsymbol{d}}_k$. Then for any $\alpha_k \in (0, 1]$

$$||\boldsymbol{g}_{k+1}||_2 \leq (1 - \alpha_k)||\boldsymbol{g}_k||_2 + \alpha_k^2 B \frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_k||_2^2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \alpha_k^2 B \frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}_k||_2^2 \tag{3.7}$$

where B is defined in Lemma 3.1.2 and $\mu_2(\boldsymbol{L}_{\mathbb{G}})$ is the smallest nonzero eigenvalue of unweighted Laplacian of $\mathbb{G}$.

3.1.4. Distributed Backtracking Line Search

To guarantee global convergence, a step size α_k needs to be selected appropriately. Our new method is inspired by the well known Armijo's rule

$$||\boldsymbol{g}(\boldsymbol{\lambda}_{k+1})||_2 \leq (1 - \sigma\alpha_k)||\boldsymbol{g}(\boldsymbol{\lambda}_k)||_2$$

where $\sigma \in \left(0, \frac{1}{2}\right]$. The decentralized computation of the dual norm $||\boldsymbol{g}(\cdot)||_2$ can be implemented by a distributed consensus based scheme. However, such computation suffers from two main drawbacks:

it is inaccurate and the fastest technique requires $\mathcal{O}(n)$ time steps. To avoid both of these issues, we construct a new method based on maximum consensus protocol:

Algorithm 11 : **Distributed Backtracking Line Search**

1: **Input:** The constants $\sigma \in \left(0, \frac{1}{2}\right]$ and $\beta \in (0,1)$, parameters $\epsilon, \Gamma, \gamma, \delta$. The i^{th} component of dual gradients $[\boldsymbol{g}_{k+1}]_i$ and $[\boldsymbol{g}_k]_i$.
2: **Output:** step size α_k.
3: Set $m_i = 0$
4: Compute $\max_i\{[\boldsymbol{g}_k]_i\}$ using maximal consensus protocol.
5: **while** $[\boldsymbol{g}_{k+1}]_i > (1-\sigma\beta^{m_i})\sqrt{n}\max_i\{[\boldsymbol{g}_k]_i\} + 2\epsilon\frac{n\gamma^2}{\delta\Gamma}$ **do**
6: $\quad m_i = m_i + 1$.
7: **end while**
8: Compute $\hat{m} = \max_i\{m_i\}$ using maximal consensus protocol.
9: Set $\alpha_k = \beta^{\hat{m}}$.

Algorithm 11 requires only $\mathcal{O}(diam(\mathbb{G}))$ time steps and conducts only exact computations. The following Lemma studies the change of step size given by the proposed backtracking line search procedure:

Lemma 3.1.4 *Let step size α_k be chosen according to Algorithm 11 and let $\boldsymbol{g}_k$ be the dual gradient evaluated at $\boldsymbol{\lambda}_k$. Then*

1. *If $||\boldsymbol{g}_k||_2 \leq \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ then $\alpha_k = 1$*
2. *If $||\boldsymbol{g}_k||_2 > \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ then $\alpha_k \geq \beta\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\max_i\{|[\boldsymbol{g}_k]_i|\}}$*

where B is a constant defined in Lemma 3.1.2 and $\mu_2(\boldsymbol{L}_{\mathbb{G}}), \mu_n(\boldsymbol{L}_{\mathbb{G}})$ are the smallest and largest nonzero eigenvalues of the unweighted Laplacian of $\mathbb{G}$.

3.1.5. Accurate Distributed Newton Method

Given the above approximation of the Newton direction, in this section, we analyze the iteration scheme of the distributed Newton method. We show that our method acquires super-linear convergence within a neighborhood of the optimal value similar to standard Newton methods. Our main results on the two-phase convergence guarantees are summarized in Theorem 3.1.5

Theorem 3.1.5 *Let γ, Γ, δ B be the constants defined in Assumption 3.1.1 and Lemma 3.1.2,*

$\mu_2(\boldsymbol{L}_{\mathbb{G}})$ and $\mu_n(\boldsymbol{L}_{\mathbb{G}})$ representing the smallest and largest nonzero eigenvalues of unweighted Laplacian of $\mathbb{G}$, $\epsilon \leq \frac{\beta}{8}\frac{\delta}{n\sqrt{n}}\left(\frac{\gamma}{\Gamma}\right)\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}$ be the precision parameter for the SDD solver. Consider iteration scheme given by $[\boldsymbol{\lambda}_{k+1}]_i = [\boldsymbol{\lambda}_k]_i + \alpha_k[\tilde{\boldsymbol{d}}_k]_i$ where the step size α_k is calculated by Algorithm 11. Then, this iteration scheme exhibits two convergence phases:

1. ***Strict Decreases Phase*** *If $||\boldsymbol{g}_k||_2 > \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$, then*

$$||\boldsymbol{g}_{k+1}||_2 - ||\boldsymbol{g}_k||_2 \leq -\frac{1}{8}\frac{\beta}{\delta\sqrt{n}}\frac{\gamma^3}{\Gamma^2}\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}$$

   where parameter $\beta \in (0,1)$.

2. ***Quadratic Decreases Phase*** *If $||\boldsymbol{g}_k||_2 \leq \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$, then for any $l \geq 1$:*

$$||\boldsymbol{g}_{k+l}||_2 \leq \frac{1}{2^{2^l}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}} + \tilde{B} + \frac{\hat{\Lambda}}{\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}}\left[\frac{2^{2^l-1}-1}{2^{2^l}}\right] \tag{3.8}$$

   where

$$\tilde{B} = \frac{1}{2}\epsilon\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\Gamma\delta}\left[\sqrt{\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma}{\mu_n(\boldsymbol{L}_{\mathbb{G}})\Gamma}} + \frac{\epsilon}{2}\right] \sim \mathcal{O}(\epsilon)$$
$$\hat{\Lambda} = \tilde{B}\frac{4B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\left[1 + \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right] \sim \mathcal{O}(\epsilon)$$

The following result follows directly from (3.8) and establishes the asymptotic limit for the dual gradient after passing strict decrease phase:

Corollary 3.1.6 *Let k_0 designate the first iteration such that $||\boldsymbol{g}_{k_0}||_2 \leq \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$. Then for the next iterations dual gradient converges quadratically to*

$$\lim_{l\to\infty}||\boldsymbol{g}_{k_0+l}||_2 = \tilde{B} + \frac{1}{2}\frac{\hat{\Lambda}}{\delta}\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\frac{\gamma^3}{\Gamma^2} \sim \mathcal{O}(\epsilon)$$

In other words, tuning precision parameter ϵ one can approximate the solution vector $\boldsymbol{x}^* = \boldsymbol{x}(\boldsymbol{\lambda}^*)$ with any arbitrary precision.

3.1.6. Experiments

This section provides empirical validation of our distributed Newton method. We performed two sets of experiments on three different network topologies: 1) random (both small and large in sizes), 2) bar-bell, and 3) star-bar graphs. The usage of different typologies allows us to better understand the effect of good and bad mixing times on the performance of our technique.

To ensure state-of-the-art performance, we compared the proposed algorithm to six benchmark solvers: 1) Distributed SDD-Newton (Tutunov et al. (2016)), 2) Augmented Lagrangian for Distributed Optimization (ADAL) (Chatzipanagiotis et al. (2015)), 3) Accelerated Dual Descent (ADD) with two different splittings (Zargham et al. (2013)), 4) dual sub-gradients, and 5) the fully distributed algorithms for convex optimization (Mosk-Aoyama et al. (2007)) (FDA).

In all experiments, we used $\Phi_e(x^{(e)}) = exp(x^{(e)}) + exp(-x^{(e)})$ to represent the cost function on the edges of the network. The flow vectors, $\boldsymbol{b}$, were chosen so that the first component corresponded to 1 and the last to -1 with all others being 0.

Feasibility & Objective Value Results

In this section, we report the performance of all algorithms on various network typologies. The parameter details for each of the network typologies are detailed below:

1. **Small Random Graphs:** We refer to a 20-node 60-edge network as a small random one. Here, edges were generated uniformly at random. Typical, condition numbers for these networks ranged between 8-15. For ease of exposure, random small networks are referred to as "sRandom Graph" in Figure 9.

2. **Large Random Graphs:** We refer to an 80-node 200-edge network as a large random one. Again, edges were generated uniformly at random. Condition numbers for such networks varied between 19-32. In Figure 10, we refer to large random networks as "lRandom Graph".

3. **Bar-Bell Graphs:** A bar-bell graph is a network consisting of two cliques connected by a line graph. In Figure 11 we considered a bar-bell network with 30 nodes. In this network, the condition number can resemble high values in the order of hundreds.

4. **Bar-Star Graphs:** A bar-star graph is a network resembling similarities to the bar-bell graphs

with two star-shapes connected by a line graph. Here, the condition number can also resemble high orders. In Figure 12, we refer to Bar-Star networks as "Bar-Star Graph".

A gradient threshold of 10^{-5} was used to assess convergence. For the solver in Algorithm 5, the length of the chain d was chosen according to

$$d = \lceil \log \left(2 \ln \left(\frac{\sqrt[3]{2}}{\sqrt[3]{2} - 1} \right) \kappa(\boldsymbol{L}_{\mathbb{G}}) \right) \rceil$$

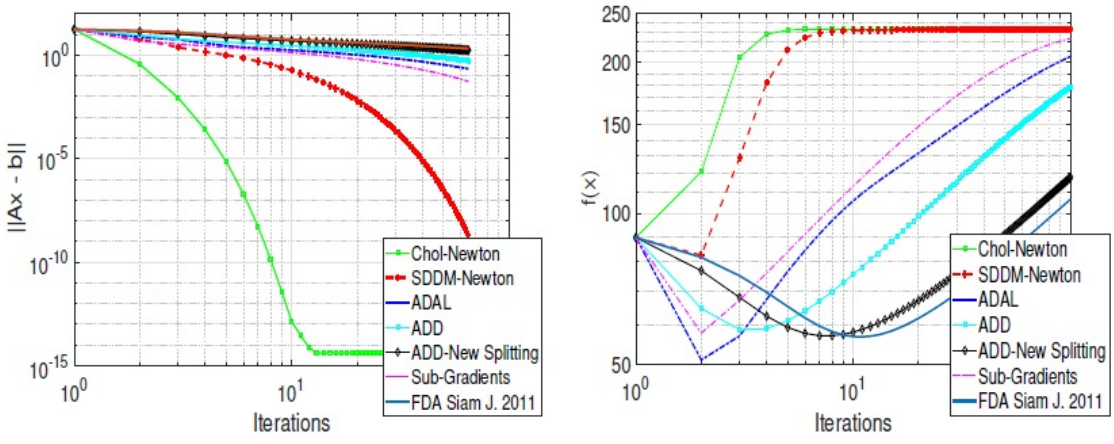
and for the solver in Algorithm 6 the degree of Chebyshev polynomial was set to

$$k_{\epsilon} = \lceil \frac{1}{2} (\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})} + 1) \ln \frac{2}{\epsilon} \rceil$$

where $\kappa(\boldsymbol{L}_{\mathbb{G}})$ is the condition number of the graph $\mathbb{G}$ and $\epsilon = 10^{-3}$ is the accuracy in approximating the Newton direction. In our experiments, we relaxed this choice to a fixed constant step-size. We varied its values between [0.1, 0.2, 0.4] and similar performance to that reported in Figures 9, 10, 11, 12 was observed. Step-sizes for all other algorithms were chosen as suggested per the corresponding paper.

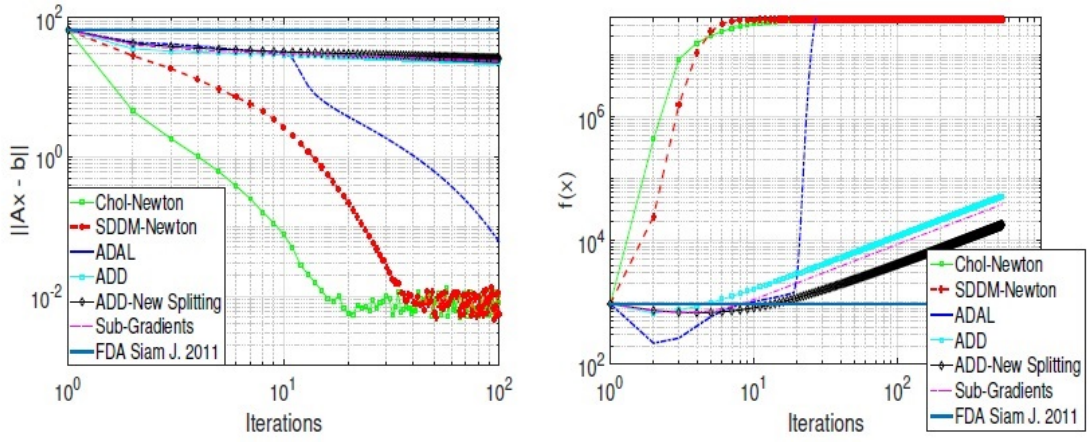
We assessed the performance of all methods using two evaluation criteria: 1) feasibility error $||\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}||_2$ with k being the iteration count, and 2) objective value $f(\boldsymbol{x}_k) = \sum_e \Phi_e(x^{(e)})$. Results on the four typologies are reported in Figures 9, 10, 11, 12. We first recognize that our proposed method is capable of outperforming others in both evaluation criteria. On small random graphs, for instance, our distributed Newton method achieves a low feasibility error of 10^{-6} in $10^{2.6}$ to $10^{2.8}$ for the second best being ADAL. This is also true on other typologies. For example, on bar-bell graphs we achieved a low feasibility error in about $10^{1.7}$ iterations compared to $10^{2.5}$ for ADAL. Though comparable to our performance, it is worth noting that ADAL does not adhere to the distributed framework we detailed before due to the need for global information in computing dual updates.[1]

[1] Increased versions of these figures are presented in Appendix A.20



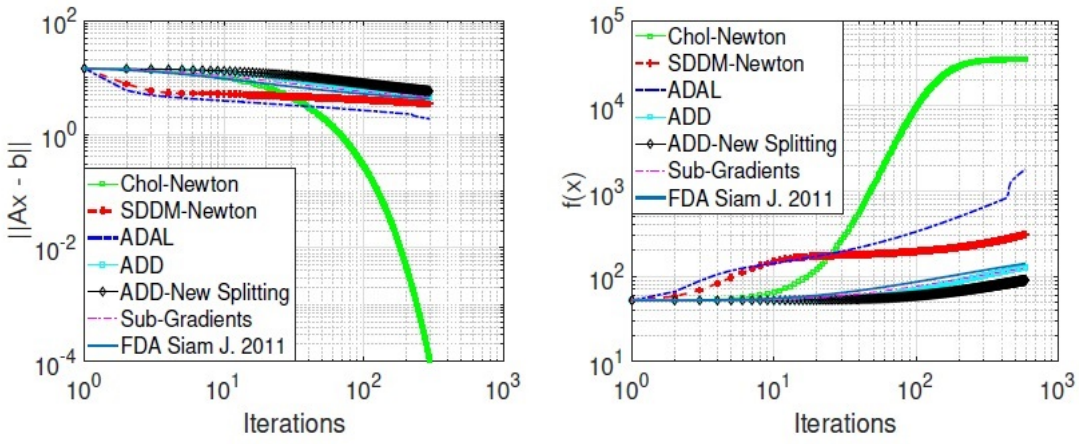
(a) sRandom Graph (b) sRandom Graph

Figure 9: Experimental Results for Small Random Graph



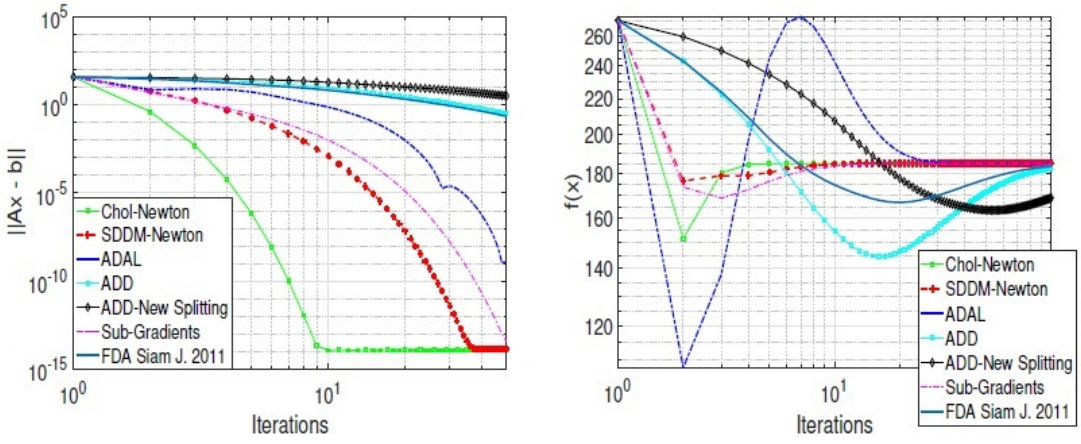
(a) lRandom Network (b) lRandom Network

Figure 10: Experimental Results for Large Random Graph



(a) Bar-Bell Graph (b) Bar-Bell Graph

Figure 11: Experimental Results for Bar-Bell Graph



(a) Bar-Star Graph (b) Bar-Star Graph

Figure 12: Experimental Results for Bar-Star Graph

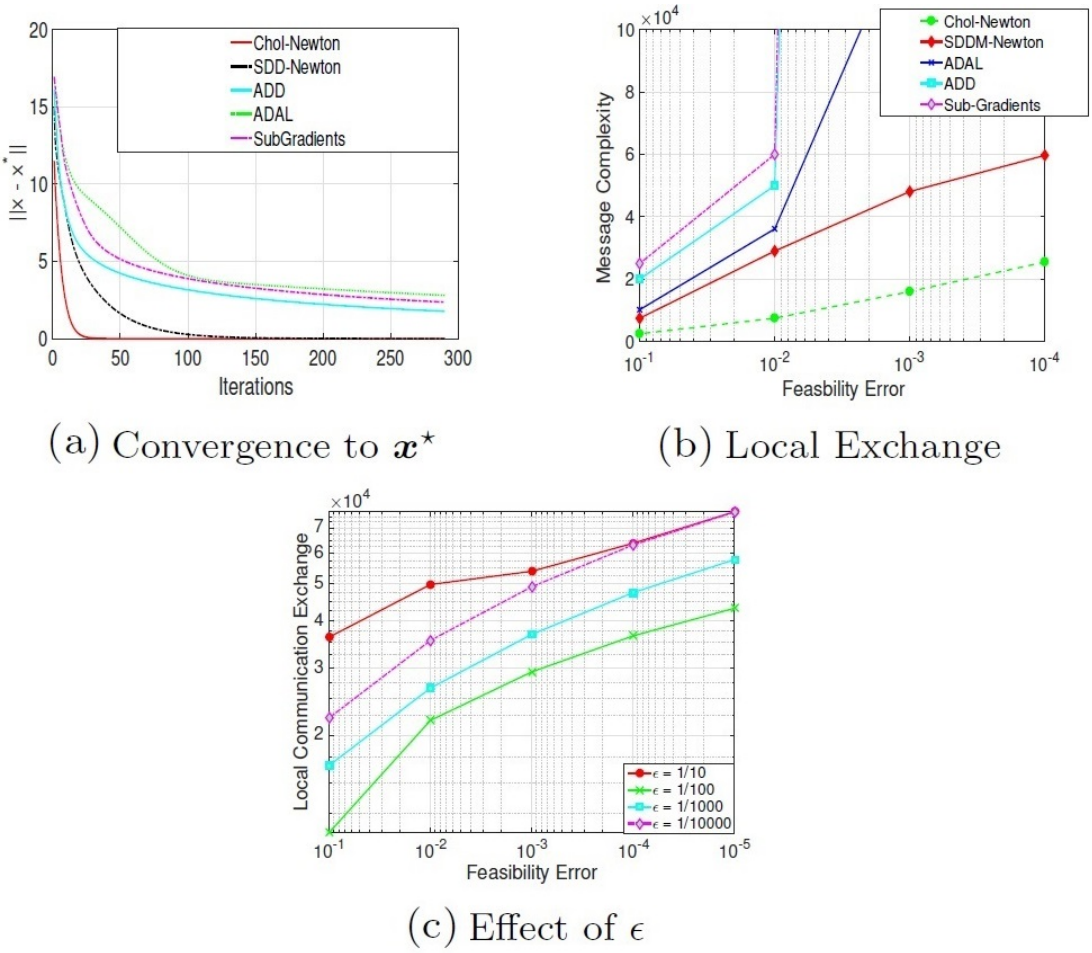


Figure 13: Experimental results: convergence, communication overhead, accuracy effect

Though successful, it is interesting to ask the question of whether our algorithm is capable of retrieving the exact optimal flow $\boldsymbol{x}^*$. We computed $\boldsymbol{x}^*$ using the centralized Newton method running on the large random network of 80 nodes and 200 edges. We then traced $||\boldsymbol{x}_k - \boldsymbol{x}^*||_2$ for all algorithms. Results reported in Figure 13 (a) show that our techniques are capable of achieving a 0 value of the norm after $\sim 10^2$ iterations compared to values $> 10^3$ for the other methods.

Communication Cost

One might argue that the improvements we achieved above arrive at high communication overhead between the processors of the network. This can be true, since at every iteration our fully distributed solvers require $\mathcal{O}\left(\kappa(\boldsymbol{L}_{\mathbb{G}})\log\frac{1}{\epsilon}\right)$ and $\mathcal{O}\left(\sqrt{\kappa(\boldsymbol{L}_{\mathbb{G}})}\log\frac{1}{\epsilon}\right)$ local exchanges respectively, with $\kappa(\boldsymbol{L}_{\mathbb{G}})$

being the condition number of the graph. To better understand this phenomenon, we conducted an experiment with a random graph of 20 nodes and 60 edges generated uniformly at random. We measured the local communication exchange between processors as a function of the feasibility error which varied from 10^{-1} to 10^{-4} These results are shown in Figure 13 (b),(c). First, it is clear that all algorithms are relatively comparable at low error demands. As these demands increase so does the communication cost for all approaches. Our methods' growth, however, is slower compared to that of others, which can become exponential for ADD and sub-gradients.

3.2. Empirical Risk Minimization

Data analysis through machine and statistical learning has become an important tool in a variety of fields including artificial intelligence, biology, medicine, finance, and marketing. Though arising in diverse applications, these problems share key characteristics, such as an extremely large number (in the order of tens of millions) of training examples typically residing in high-dimensional spaces. With this unprecedented growth in data, the need for distributed computation across multiple processing units is ever-pressing. This direction holds the promise for algorithms that are both rich enough to capture the complexity of modern data, and scalable enough to handle Big Data efficiently.

In the distributed setting, central problems are split across multiple processors each having access to local objectives. We are interested in cases when the global objective is non-separable. Therefore, when attempting to distribute the optimization of the objective, multiple copies of the minimizer have to be created. Then, our goal is not only to minimize a sum of local costs, but also to ensure consensus (agreement) on the minimizer across all processors Nedić and Ozdaglar (2008). To clarify, consider the example of linear regression in which the goal is to find a latent model for a given data-set. Rather than searching for a centralized solution, one can distribute the optimization across multiple processors, each having access to local costs defined over random subsets of the full data-set. In such a case, each processor learns a separate chunk of the latent model, which is then unified by incorporating consensus constraints.

Generally, there are two popular classes of algorithms for distributed optimization. The first is sub-gradient based, while the second relies on a decomposition-coordination procedure. Sub-gradient

algorithms proceed by taking a gradient-related step followed by an averaging with neighbors at each iteration. The computation of each step is relatively cheap and can be implemented in a distributed fashion Nedić and Ozdaglar (2008). Though cheap to compute, the best known convergence rate of subgradient methods is relatively slow given by $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$ with t being the total number of iterations Wei and Ozdaglar (2012), Goffin (1977). The second class of algorithms solves constrained problems by relying on dual methods. One of the well-known methods (state-of-the-art) from this class is the Alternating Direction Method of Multipliers (ADMM) Boyd et al. (2011). ADMM decomposes the original problem to two subproblems which are then solved sequentially leading to updates of dual variables. In Wei and Ozdaglar (2012), the authors show that ADMM can be fully distributed over a network leading to improved convergence rates in the order of $\mathcal{O}\left(\frac{1}{t}\right)$.

Apart from accuracy problems inherent to ADMM-based methods Kadkhodaie et al. (2015), much rate improvement can be gained from adopting second-order (Newton) methods. Though a variety of techniques have been proposed Wei et al. (2010), Scheinberg and Tang (2013), less progress has been made at leveraging ADMM's accuracy and convergence rate issues. In a recent attempt [4], [5], the authors propose a distributed second-order method for general consensus by using the approach in Zargham et al. (2013) to compute the Newton direction. As detailed in our experiments, this method suffers from two problems. First, it fails to outperform ADMM and second, it faces storage and computational deficiencies for large data-sets, and thus ADMM retains state-of-the-art status.

3.2.1. Problem Formulation

Similar to previous setting, consider a network of n agents represented by a connected undirected graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with $|\mathbb{V}| = n$ and $|\mathbb{E}| = m$. Each agent, i, corresponding to a node, can exchange information among its first-hop neighborhood denoted by $\mathbb{N}(i) = \{j \in \mathbb{V} : (i, j) \in \mathbb{E}\}$. The size of such $\mathbb{N}(i)$ is referred to as the degree of node i, i.e., $d(i) = |\mathbb{N}(i)|$. In the general form, the goal is for each agent to determine an unknown vector $\boldsymbol{x}_i \in \mathbb{R}^p$ which minimizes a sum of multivariate cost functions $\{f_i\}_{i=1}^n$ distributed over the network while abiding by consensus constraints:

$$\min_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n} f(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n) = \min_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n} \sum_{i=1}^{n} f_i(\boldsymbol{x}_i) \tag{3.9}$$

$$s.t.\ \boldsymbol{x}_1 = \boldsymbol{x}_2 = \ldots = \boldsymbol{x}_n$$

Though multiple attempts have been made at distributing the global consensus problem, the majority of these works suffer from the following drawbacks. The first line of work is that introduced in Wei and Ozdaglar (2012). This work mostly focuses on the univariate and separable settings and suffers from scalability when generalized to the multivariate case. The second, on the other hand, is that of Boyd et al. (2011), where the focus is mainly on a parallelized setting and not a distributed one. Parallel methods assume shared memory that can become restrictive for problems with large data sets. In this work, we focus on a true distributed setting where each processor abides by its own memory constraints and the framework does not invoke any central node. We start by introducing a set of vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$, each in $\mathbb{R}^n$. Each vector $\boldsymbol{y}_j$ acts as a collector for every dimension of the solution across all nodes. In other words, each vector $\boldsymbol{y}_i$ contains the i^{th} components of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$:

$$\boldsymbol{y}_1 = [[\boldsymbol{x}_1]_1, \ldots, [\boldsymbol{x}_n]_1]^\mathsf{T}, \ldots, \boldsymbol{y}_p = [[\boldsymbol{x}_1]_p, \ldots, [\boldsymbol{x}_n]_p]^\mathsf{T}$$

Clearly, each vector of the collection of $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$ is locally distributed among the nodes of graph $\mathbb{G}$, since each node $i \in \mathbb{V}$ needs only to have access to the i^{th} components of such vectors. Consequently, we can rewrite the problem of Equation (3.9) in an equivalent distributed form:

$$\begin{aligned} &\min_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_p} f(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p) = \min_{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_p} \sum_{i=1}^{n} f_i([\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i) \\ &s.t. \quad \boldsymbol{L}_\mathbb{G}\boldsymbol{y}_1 = \boldsymbol{0}, \quad \boldsymbol{L}_\mathbb{G}\boldsymbol{y}_2 = \boldsymbol{0}, \quad \ldots, \boldsymbol{L}_\mathbb{G}\boldsymbol{y}_p = \boldsymbol{0} \end{aligned} \tag{3.10}$$

To finalize the definition, we write the problem in Equation (3.10) in a vectorized format as:

$$\begin{aligned} &\min_{\boldsymbol{y}} \sum_{i=1}^{n} f_i([\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i) \\ &s.t. \underbrace{\boldsymbol{I}_{p\times p} \otimes \boldsymbol{L}_\mathbb{G}}_{\boldsymbol{M}} \underbrace{\boldsymbol{y}}_{\boldsymbol{y}\in\mathbb{R}^{np}} = \boldsymbol{0} \end{aligned} \tag{3.11}$$

where $\boldsymbol{M} = \boldsymbol{I}_{p\times p} \otimes \boldsymbol{L}_\mathbb{G}$ is a block-diagonal matrix with Laplacian diagonal entries, and $\boldsymbol{y}$ is a vector concatenating $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$. At this stage, our aim is to solve the problem in Equation (3.11)using dual techniques. Before presenting properties of the dual problem, we next introduce a standard assumption Boyd and Vandenberghe (2004b) on the associated functions f_i^s:

Assumption 3.2.1 *The cost functions, $f_i(\cdot)$, in Equation (3.9) are*

1. *twice continuously differentiable, i.e.,* $\gamma \boldsymbol{I}_{p\times p} \leq \nabla^2 f_i(\cdot) \leq \Gamma \boldsymbol{I}_{p\times p}$*, with* γ *and* Γ *are constants; and*

2. *Hessian Lipschitz continuous for all* $i \in \mathbb{V}$*, i.e.,* $\left|\left|\nabla^2 f_i(\boldsymbol{x}_i) - \nabla^2 f_i(\hat{\boldsymbol{x}}_i)\right|\right|_2 \leq \delta ||\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i||_2$ *for all* $\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i \in \mathbb{R}^p$

Please note that though these assumptions seem quite restrictive, in our empirical evaluation (Section VI) we assess our method on a broader class of functions (e.g., non-smooth $L1$ regularized least squares).

3.2.2. primal-dual Technique

Following the standard primal-dual method for general consensus problem (3.11), we first introduce a vector of dual variables $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1^\mathsf{T}, \ldots, \boldsymbol{\lambda}_p^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^{np}$, where each $\boldsymbol{\lambda}_i \in \mathbb{R}^n$ are Lagrange multipliers, one for each dimension of the unknown vector. For distributed computations, we assume that each node i, needs only to store its corresponding components $[\boldsymbol{\lambda}_1]_i, \ldots, [\boldsymbol{\lambda}_p]_i$. Consequently, the Lagrangian of Equation (3.11) can be written as follows:

$$\mathscr{L}(\boldsymbol{y}, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \left(f_i([\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i) + [\boldsymbol{y}_1]_i [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_1]_i + \ldots + [\boldsymbol{y}_p]_i [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_p]_i \right)$$

Hence, the dual function is given as:

$$q(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \inf_{[\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i} \left(f_i([\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i) + [\boldsymbol{y}_1]_i [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_1]_i + \ldots + [\boldsymbol{y}_p]_i [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_p]_i \right) \tag{3.12}$$

Having determined the dual variable $\boldsymbol{\lambda}$, we still require a decentralized procedure which allows us to infer about the corresponding primal $\boldsymbol{y}(\boldsymbol{\lambda})$. Using the above, the primal variables are determined as the solution to the following system of differential equations:

$$\frac{\partial f_i(\cdot)}{\partial [\boldsymbol{y}_1]_i} = -[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_1]_i, \quad \ldots, \quad \frac{\partial f_i(\cdot)}{\partial [\boldsymbol{y}_p]_i} = -[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_p]_i \tag{3.13}$$

Clearly, Equation (3.13) is locally defined for each node $i \in \mathbb{V}$, where for each $r = 1, \ldots, p$:

$$-[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_r]_i = \sum_{j \in \mathbb{N}(i)} [\boldsymbol{\lambda}_r]_j - d(i) [\boldsymbol{\lambda}_r]_i \tag{3.14}$$

Hence, each node i can construct its own system of equations by collecting $\{[\boldsymbol{\lambda}_1]_j, \ldots, [\boldsymbol{\lambda}_p]_j\}$ from its neighbors $j \in \mathbb{N}(i)$ without the need for full communication. Denoting the solution of the partial differential equations as:

$$[\boldsymbol{y}_1]_i = \phi_1^{(i)}([\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_1]_i, \ldots, [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_p]_i), \quad \ldots, \quad [\boldsymbol{y}_p]_i = \phi_p^{(i)}([\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_1]_i, \ldots, [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_p]_i)$$

we can show the following essential theoretical guarantee on the partial derivatives:

Lemma 3.2.2 *Let $z_1 = [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_1]_i, z_2 = [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_2]_i, \ldots, z_p = [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_p]_i$. Under Assumption 3.2.1, the functions $\phi_1^{(i)}, \ldots, \phi_p^{(i)}$ exhibit bounded partial derivatives with respect to $z_1, \ldots, z_p$. In other words, for any $r = 1, \ldots, p$:*

$$\left|\frac{\partial \phi_r^{(i)}}{\partial z_1}\right| \leq \frac{\sqrt{p}}{\gamma}, \quad \ldots, \quad \left|\frac{\partial \phi_r^{(i)}}{\partial z_p}\right| \leq \frac{\sqrt{p}}{\gamma}$$

for any $[z_1, \ldots, z_p] \in \mathbb{R}^p$.

The above result is crucial in our analysis, as an obvious corollary is that each function, $\phi_r^{(i)}$, is Lipschitz continuous, i.e., for any two vectors $\tilde{\boldsymbol{z}} = [\tilde{z}_1, \ldots, \tilde{z}_p]$ and $\boldsymbol{z} = [z_1, \ldots, z_p]$:

$$\left|\phi_r^{(i)}(\tilde{\boldsymbol{z}}) - \phi_r^{(i)}(\boldsymbol{z})\right| \leq \frac{\sqrt{p}}{\gamma}||\tilde{\boldsymbol{z}} - \boldsymbol{z}||_2 \tag{3.15}$$

Our method for computing the Newton direction relies on the fact that the system of equations described by the Hessian of the dual problem can be solved using SDD solvers. We prove this property in the following

Lemma 3.2.3 *The function $q(\boldsymbol{\lambda}) = q(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_p)$ abides by the following properties:*

1. *Let $\boldsymbol{y}(\boldsymbol{\lambda})$ be the primal variable corresponding to dual vector $\boldsymbol{\lambda}$. Then the gradient and the Hessian of $q(\boldsymbol{\lambda})$ are given by*

$$\nabla q(\boldsymbol{\lambda}) = \boldsymbol{g}(\boldsymbol{\lambda}) = \boldsymbol{M}\boldsymbol{y}(\boldsymbol{\lambda})$$
$$\nabla^2 q(\boldsymbol{\lambda}) = \boldsymbol{H}(\boldsymbol{\lambda}) = -\boldsymbol{M}\left[\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))\right]^{-1}\boldsymbol{M}$$

2. *Denote $\mu_n(\boldsymbol{L}_{\mathbb{G}})$ as the largest eigenvalue of the unweighted Laplacian of $\mathbb{G}$ and constants δ, γ*

are given in Assumption 3.2.1. Then, for constant $B = p\delta\left(\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\gamma}\right)^3$ *and for any* $\bar{\boldsymbol{\lambda}}, \boldsymbol{\lambda} \in \mathbb{R}^{np}$*:*

$$||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq B|\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2$$

i.e. $\boldsymbol{H}(\boldsymbol{\lambda})$ *is Lipschits continuous with constant* B*.*

The first result in Lemma 3.2.3 shows that the gradient can be distributed across the nodes of $\mathbb{G}$ which is true due to the sparsity pattern of the incidence matrix $\boldsymbol{M}$. Moreover, it demonstrates the specific factorization of the Hessian, which is crucial for decentralized computation of Newton direction.

3.2.3. Distributed Computation of Newton Direction

We solve the consensus problem using Newton-like techniques, where our method follows the approximate Newton direction in the dual: $\boldsymbol{\lambda}^{[k+1]} = \boldsymbol{\lambda}^{[k]} + \alpha_k \tilde{\boldsymbol{d}}^k$ where k is the iteration number, and $\alpha^{[k]}$ the step-size, $\tilde{\boldsymbol{d}}^{[k]}$ is the ϵ_0-approximation to the exact Newton direction at iteration k. For efficient operation, the main goal is to accurately approximate the Newton direction in a fully distributed fashion. This can be achieved with the help of the SDD properties of the dual Hessian proved earlier. Recalling that exact Newton computes[2]:

$$\boldsymbol{H}^{[k]}\boldsymbol{d}^{[k]} = -\boldsymbol{g}^{[k]}$$

or equivalently,

$$\boldsymbol{M}[\nabla^2 f(\boldsymbol{y}^{[k]})]^{-1}\boldsymbol{M}\boldsymbol{d}^{[k]} = \boldsymbol{M}\boldsymbol{y}^{[k]} \tag{3.16}$$

we notice that Equation (3.16) can be simplified to the following SDD linear system:

$$\boldsymbol{M}\boldsymbol{d}^{[k]} = \nabla^2 f(\boldsymbol{y}^{[k]})\boldsymbol{y}^{[k]} \tag{3.17}$$

This equation is by itself SDD which can be split into p distributed SDD systems and solved in a distributed fashion using solvers proposed in Algorithms 5 and 6. Having attained that solution, we map the system (3.17) to p-SDD systems by introducing: $\boldsymbol{d}^{[k]} = \left(\left(\boldsymbol{d}_1^{[k]}\right)^{\mathsf{T}}, \ldots, \left(\boldsymbol{d}_p^{[k]}\right)^{\mathsf{T}}\right)^{\mathsf{T}}$ with each $\boldsymbol{d}_r^{[k]} \in \mathbb{R}^n$. It is easy to see that this can be split to the following collection of p linear systems

[2]Above we used the following notation $\boldsymbol{y}^{[k]} = \boldsymbol{y}(\boldsymbol{\lambda}^{[k]}), \boldsymbol{g}^{[k]} = \boldsymbol{M}\boldsymbol{y}^{[k]}$ and $\boldsymbol{H}^{[k]} = -\boldsymbol{M}[\nabla^2 f(\boldsymbol{y}^{[k]})]^{-1}\boldsymbol{M}$

for $r = 1, \ldots, n$.

$$\boldsymbol{L}_{\mathbb{G}} \boldsymbol{d}_1^{[k]} = \boldsymbol{b}_1^{[k]}, \quad \ldots, \quad \boldsymbol{L}_{\mathbb{G}} \boldsymbol{d}_p^{[k]} = \boldsymbol{b}_p^{[k]} \tag{3.18}$$

where $\boldsymbol{b}_1^{[k]}, \ldots, \boldsymbol{b}_p^{[k]} \in \mathbb{R}^n$ defined as:

$$\begin{aligned}
[\boldsymbol{b}_1^{[k]}]_r &= \sum_{l=1}^{p} \frac{\partial^2 f_r(\cdot)}{\partial [\boldsymbol{y}_1]_r \partial [\boldsymbol{y}_l]_r} [\boldsymbol{y}_l^{[k]}]_r \\
&\vdots \\
[\boldsymbol{b}_p^{[k]}]_r &= \sum_{l=1}^{p} \frac{\partial^2 f_r(\cdot)}{\partial [\boldsymbol{y}_p]_r \partial [\boldsymbol{y}_l]_r} [\boldsymbol{y}_l^{[k]}]_r
\end{aligned}$$

for $r = 1, \ldots, n$. Interestingly, the above computations can be performed completely locally by noting that each node $r \in \mathbb{V}$ can compute the r^{th} component of each vector $\boldsymbol{b}_1^{[k]}, \ldots, \boldsymbol{b}_p^{[k]}$. This is true as such a node stores f_r as well as the variables $[\boldsymbol{y}_1^{[k]}]_r, \ldots, [\boldsymbol{y}_p^{[k]}]_r$. Before commencing to the convergence analysis, the final step needed is to establish the connection between the approximate solutions:

Lemma 3.2.4 *Let $\tilde{\boldsymbol{d}}_1^{[k]}, \ldots, \tilde{\boldsymbol{d}}_p^{[k]}$ be ϵ_0 close solutions of systems (3.17). Then $\tilde{\boldsymbol{d}}^{[k]}$ is $\epsilon-$approximate solution to system the original (3.16), with $\epsilon = \epsilon_0 \sqrt{\frac{\Gamma}{\gamma} \frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}$.*

This lemma establishes the connection between accuracy of solutions for system (3.17) and the original Hessian system (3.16) and allows us to approximate a real Newton direction with arbitrary precision ϵ by properly tuning parameter ϵ_0 in Algorithms 5 and 6.

3.2.4. Distributed Newton Method For Empirical Risk Minimization

Decentralized computation of approximate Newton direction leads us to a decentralized Newton-type iteration scheme $\boldsymbol{\lambda}^{[k+1]} = \boldsymbol{\lambda}^{[k]} + \alpha_k \tilde{\boldsymbol{d}}^{[k]}$, or for each node $i \in \mathbb{V}$ given by a collection of the following p updates:

$$\begin{cases}
[\boldsymbol{\lambda}_1^{[k+1]}]_i = [\boldsymbol{\lambda}_1^{[k]}]_i + \alpha_k [\tilde{\boldsymbol{d}}_1^{[k]}]_i \\
[\boldsymbol{\lambda}_2^{[k+1]}]_i = [\boldsymbol{\lambda}_2^{[k]}]_i + \alpha_k [\tilde{\boldsymbol{d}}_2^{[k]}]_i \\
\vdots \\
[\boldsymbol{\lambda}_p^{[k+1]}]_i = [\boldsymbol{\lambda}_p^{[k]}]_i + \alpha_k [\tilde{\boldsymbol{d}}_p^{[k]}]_i
\end{cases} \tag{3.19}$$

where $[\tilde{\boldsymbol{d}}_r^{[k]}]_i$ is the i^{th} component of the approximation to a part $\boldsymbol{d}_r^{[k]}$ of a real Newton direction and α_k is a step size. The following Lemma studies the change of the norm of dual gradient for the scheme (3.19) and plays a crucial role for the convergence analysis:

Lemma 3.2.5 *Let us consider iteration scheme given by (3.19) and denote*

$$\boldsymbol{\epsilon}^{[k]} = \boldsymbol{H}^{[k]}\tilde{\boldsymbol{d}}^{[k]} + \boldsymbol{g}^{[k]}$$

be the approximation error vector corresponding to $\tilde{\boldsymbol{d}}^{[k]}$*. Then for any* $\alpha_k \in (0, 1]$

$$||\boldsymbol{g}^{[k+1]}||_2 \leq (1-\alpha_k)||\boldsymbol{g}^{[k]}||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k]}||_2^2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}^{[k]}||_2^2 \tag{3.20}$$

where B *is defined in Lemma 3.2.3 and* $\mu_2(\boldsymbol{L}_{\mathbb{G}})$ *is the smallest nonzero eigenvalue of unweighted Laplacian of* $\mathbb{G}$*.*

3.2.5. Distributed Backtracking Line Search for Empirical Risk Minimization

To guarantee global convergence a step size α_k needs to be selected appropriately. Similarly to Section 3.1.4 we follow Armijo's rule:

$$||\boldsymbol{g}(\boldsymbol{\lambda}^{[k+1]})||_2 \leq (1-\sigma\alpha_k)||\boldsymbol{g}(\boldsymbol{\lambda}^{[k]})||_2$$

where $\sigma \in \left(0, \frac{1}{2}\right]$. The inexact decentralized computation of the dual norm $||\boldsymbol{g}(\cdot)||_2$ can be implemented by a distributed consensus-based scheme in $\mathcal{O}(np)$ time steps. As an exact and fast alternative we propose the following:

Where we use that $\boldsymbol{g}^{[k]} = \boldsymbol{M}\boldsymbol{y}^{[k]} = \left(\left(\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_1^{[k]}\right)^{\mathsf{T}}, \ldots, \left(\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_1^{[k]}\right)^{\mathsf{T}}\right)^{\mathsf{T}}$. Algorithm 12 requires only $\mathcal{O}(diam(\mathbb{G}))$ time steps and conducts only exact computations. The following Lemma studies the change of step size given by the proposed backtracking line search procedure:

Lemma 3.2.6 *Let step size* α_k *is chosen according to Algorithm 12 and let* $\boldsymbol{g}^{[k]}$ *be the dual gradient evaluated at* $\boldsymbol{\lambda}^{[k]}$*. Then*

Algorithm 12 : **Distributed Line Search for ERM**

Input: The constants $\sigma \in \left(0, \frac{1}{2}\right]$ and $\beta \in (0,1)$, parameters $\epsilon, \Gamma, \gamma, \delta$. The i^{th} component of dual gradients $\{[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_r^{[k+1]}]_i\}_{r=1}^p$ and $\{[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_r^{[k]}]_i\}_{r=1}^p$ for $r = 1, \ldots, p$
Output: step size α_k.
Set $m_i = 0$.
Compute $\eta_i = \max_r\{|[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_r^{[k]}]_i|\}$.
Compute $\max_i\{\eta_i\}$ using maximal consensus protocol.
while $\max_r\{|[\boldsymbol{L}_{\mathbb{G}} \boldsymbol{y}_r^{[k+1]}]_i|\} > (1 - \sigma\beta^{m_i})\sqrt{n}\max_i\{\eta_i\} + 2\epsilon\frac{n\gamma^2}{p\delta\Gamma}$ **do**
    $m_i = m_i + 1$.
end while
Compute $\hat{m} = \max_i\{m_i\}$ using maximal consensus protocol.
Set $\alpha_k = \beta^{\hat{m}}$.

1. *If* $||\boldsymbol{g}^{[k]}||_2 \leq \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ *then* $\alpha_k = 1$

2. *If* $||\boldsymbol{g}^{[k]}||_2 > \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ *then* $\alpha_k \geq \beta\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2 \max_i\{\eta_i\}}$

where B is a constant defined in Lemma 3.1.2 and $\mu_2(\boldsymbol{L}_{\mathbb{G}}), \mu_n(\boldsymbol{L}_{\mathbb{G}})$ are the smallest and largest nonzero eigenvalues of the unweighted Laplacian of $\mathbb{G}$.

3.2.6. Accurate Distributed Newton Method For Empirical Minimization Problem

Given the above approximation of the Newton direction, in this section, we analyze the iteration scheme of the distributed Newton method. Similarly as in the case of Network Flop problem, our method acquires super-linear convergence within a neighborhood of the optimal value similar to standard Newton methods. Our main results on the two-phase convergence guarantees are summarized in Theorem 3.2.7

Theorem 3.2.7 *Let γ, Γ, δ B be the constants defined in Assumption 3.2.1 and Lemma 3.2.3, $\mu_2(\boldsymbol{L}_{\mathbb{G}})$ and $\mu_n(\boldsymbol{L}_{\mathbb{G}})$ representing the smallest and largest nonzero eigenvalues of the unweighted Laplacian of $\mathbb{G}$, $\epsilon \leq \frac{\beta}{8}\frac{\gamma^3}{\Gamma^2 p\delta}\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{\mu_n^3(\boldsymbol{L}_{\mathbb{G}})}$ be the precision parameter for the SDD solver. Consider iteration scheme given by scheme (3.19) with the step size α_k is calculated by Algorithm 12. Then, this iteration scheme exhibits two convergence phases:*

1. ***Strict Decreases Phase*** *If $||\boldsymbol{g}^{[k]}||_2 > \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$, then*

$$||\boldsymbol{g}^{[k+1]}||_2 - ||\boldsymbol{g}^{[k]}||_2 \leq -\frac{\beta}{8\sqrt{n}p\delta}\frac{\gamma^3}{\Gamma^2}\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{\mu_n^3(\boldsymbol{L}_{\mathbb{G}})}$$

where parameter $\beta \in (0, 1)$.

2. ***Quadratic Decreases Phase*** *If* $||\boldsymbol{g}^{[k]}||_2 \leq \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$, *then for any* $l \geq 1$*:*

$$||\boldsymbol{g}^{[k+l]}||_2 \leq \frac{1}{2^{2^l} \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}} + \hat{B} + \frac{\tilde{\Lambda}}{\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}} \left[\frac{2^{2^l-1}-1}{2^{2^l}}\right] \tag{3.21}$$

where

$$\hat{B} = \frac{1}{2}\epsilon \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\mu_n(\boldsymbol{L}_{\mathbb{G}})p\Gamma\delta} \left[\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\gamma}{\Gamma}} + \frac{\epsilon}{2}\right] \sim \mathcal{O}(\epsilon)$$
$$\tilde{\Lambda} = \hat{B}\frac{4B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})} \left[1 + \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right] \sim \mathcal{O}(\epsilon)$$

The following result follows directly from (3.21) and establishes the asymptotic limit for dual gradient after passing strict decrease phase:

Corollary 3.2.8 *Let* k_0 *designate the first iteration such that* $||\boldsymbol{g}^{[k_0]}||_2 \leq \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$. *Then for the next iterations dual gradient converges quadratically to*

$$\lim_{l\to\infty} ||\boldsymbol{g}^{[k_0+l]}||_2 = \hat{B} + \frac{1}{2}\frac{\tilde{\Lambda}}{p\delta}\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{\mu_n^3(\boldsymbol{L}_{\mathbb{G}})}\frac{\gamma^3}{\Gamma^2} \sim \mathcal{O}(\epsilon)$$

In other words, tuning precision parameter ϵ one can approximate the solution vector $\boldsymbol{y}^* = \boldsymbol{y}(\boldsymbol{\lambda}^*)$ with any arbitrary precision.

3.2.7. Do We Need Hessians?

Despite the fast convergence rates, the traditional Newton method is often regarded skeptically for many practical large-scale applications. There are two main reasons for such skepticism. These drawbacks are related to the space and computational demands for storing and operating with Hessian matrices. In this section, we discuss several techniques that can be used to address these issues. We focus on the distributed framework presented earlier and present the precise step of the Distributed Newton Method involving local Hessian operations. Next, we present two Hessian-free approaches.

Problem with Hessian Calculations:

In Section 3.2.3, we discussed the computation of the Newton direction $\boldsymbol{d}^k = \left(\left(\boldsymbol{d}_1^{[k]}\right)^\mathsf{T}, \ldots, \left(\boldsymbol{d}_p^{[k]}\right)^\mathsf{T} \right)^\mathsf{T}$ for dual function and reduced this problem to a collection of p systems:

$$\boldsymbol{L}_{\mathbb{G}} \boldsymbol{d}_1^{[k]} = \boldsymbol{b}_1^{[k]}, \ldots, \boldsymbol{L}_{\mathbb{G}} \boldsymbol{d}_p^{[k]} = \boldsymbol{b}_p^{[k]},$$

where p is the dimensionality of the unknown parameter and k indicates the current iteration of the algorithm, which will be dropped for simplicity. We showed that each vector $\boldsymbol{b}_j$ on the right hand side is distributed across the nodes of graph $\mathbb{G}$ and can be locally computed, i.e., for $i = 1, \ldots, n$:

$$\begin{aligned}
[\boldsymbol{b}_1]_i &= \sum_{r=1}^{p} \frac{\partial^2 f_i}{\partial [\boldsymbol{y}_1]_i \partial [\boldsymbol{y}_r]_i} [\boldsymbol{y}_r]_i \\
&\vdots \\
[\boldsymbol{b}_p]_i &= \sum_{r=1}^{p} \frac{\partial^2 f_i}{\partial [\boldsymbol{y}_p]_i \partial [\boldsymbol{y}_r]_i} [\boldsymbol{y}_r]_i
\end{aligned}$$

Let $\boldsymbol{\mathcal{Y}}_i = ([\boldsymbol{y}_1]_i, \ldots, [\boldsymbol{y}_p]_i)^\mathsf{T}$ be a vector stored in node i and collecting the i^{th} components of vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$. According to the above equations, each node $i \in \mathbb{V}$ computes the following Hessian-vector product:

$$\begin{bmatrix} [\boldsymbol{b}_1]_i \\ \vdots \\ [\boldsymbol{b}_p]_i \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f_i}{\partial^2 [\boldsymbol{y}_1]_i} & \cdots & \frac{\partial^2 f_i}{\partial [\boldsymbol{y}_1]_i \partial [\boldsymbol{y}_p]_i} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_i}{\partial [\boldsymbol{y}_p]_i \partial [\boldsymbol{y}_1]_i} & \cdots & \frac{\partial^2 f_i}{\partial^2 [\boldsymbol{y}_p]_i} \end{bmatrix} \begin{bmatrix} [\boldsymbol{y}_1]_i \\ \vdots \\ [\boldsymbol{y}_p]_i \end{bmatrix} = \nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i) \boldsymbol{\mathcal{Y}}_i,$$

where $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)$ is the Hessian of local function f_i evaluated at $\boldsymbol{\mathcal{Y}}_i$. In other words, component-wise computation of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_p$ can be attained by locally calculating the following Hessian-vector products:

$$\nabla^2 f_1(\boldsymbol{\mathcal{Y}}_1) \boldsymbol{\mathcal{Y}}_1, \ \ldots, \ \nabla^2 f_n(\boldsymbol{\mathcal{Y}}_n) \boldsymbol{\mathcal{Y}}_n \tag{3.22}$$

The straightforward computation of these products requires $\mathcal{O}(p^2)$ space for the local memory of each node. The situation with time complexity is even worse. With a certain pre-processing requiring $\mathcal{O}(p^{2+\epsilon})$, the Hessian-vector product can be computed in $\mathcal{O}\left(\frac{p^2}{(\epsilon \log p)^2}\right)$ time, with ϵ being the accuracy parameter Williams (2007). These characteristics restrict our method from being applied on high-

dimensional problems. In the next sections we address this issue and present three Hessian-free approaches for the Distributed Newton Method.

The Method of Finite Differences:

The first method computes a Hessian-vector product $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$ using the well-known finite difference formula:

$$\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i = \frac{\nabla f_i((1+t)\boldsymbol{\mathcal{Y}}_i) - \nabla f_i((1-t)\boldsymbol{\mathcal{Y}}_i)}{2t} + o(t^2) \tag{3.23}$$

Given a precision parameter t, this result speeds up the computation of $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$ by evaluating the local gradient of function f_i at points $(1+t)\boldsymbol{\mathcal{Y}}_i$ and $(1-t)\boldsymbol{\mathcal{Y}}_i$. In practice, parameter t requires careful consideration, which restricts the application of finite differences. On one hand, the truncation error in (3.23) reduces quadratically with t and, therefore, prefers smaller values. On the other hand, choosing parameter t too small implies taking the ratio over t in equation (3.23), and hence, magnifying the rounding errors in the nominator of (3.23). In other words, a lower value of parameter t reduces the truncation error but increases the rounding error, and a higher value of t has the opposite effect. The next lemma establishes the optimal value for parameter t balancing the trade-off between truncation and rounding errors:

Lemma 3.2.9 *Let $\frac{\partial f_i}{\partial [\boldsymbol{y}_r]_i}$ be the partial derivative of local function f_i with respect to r^{th} component on its argument and $\frac{\partial \tilde{f}_i}{\partial [\boldsymbol{y}_r]_i}$ its numerical representation. Then, the total error in approximation (3.23) is minimized at*

$$t^* \approx \mathcal{O}(\sqrt[3]{\epsilon_{machine}}) \tag{3.24}$$

where $\epsilon_{machine}$ is the machine precision parameter (usually $\epsilon_{machine} \approx 10^{-16}$).

Proof See Appendix.

Automatic Differentiation:

Previously, we considered finite differences for Hessian-vector multiplications and discussed the main drawback of this technique caused by the interplay between truncation and rounding errors. Here, we illustrate the second approach for computing Hessian-vector products based on *Automatic Differentiation* (AD) Baydin et al. (2015). In a nutshell, AD exploits the fact that all numerical computations

can be decomposed into compositions of a finite collection of elementary operations. The derivative of a complex function then can be computed by applying a chain rule to the derivatives of these elementary operations. The first step of AD is representing the input function $f_i : \mathbb{R}^p \to \mathbb{R}$ in the form of its *computational graph*:

Definition Let $f_i(\boldsymbol{x}) : \mathbb{R}^p \to \mathbb{R}$ be the input function. Then its computational graph $\mathcal{G}(f_i) = (\mathcal{V}_{f_i}, \mathcal{E}_{f_i})$ is constructed recursively as follows:

1. For each component $[\boldsymbol{x}]_j$ of the argument vector, add node v to $\mathcal{V}_{f_i}$. These nodes are referred to as input nodes

2. Let $u, w \in \mathcal{V}_{f_i}$ and $val(u), val(w)$ be the corresponding numerical values. Then repeat the following steps:

   (a) For each elementary arithmetic operation $val(u) \bullet val(w)$ where $\bullet \in \{+, -, \times, /\}$ add node v to $\mathcal{V}_{f_i}$ and directed edges $(u, v), (w, v)$ to $\mathcal{E}_{f_i}$.

   (b) For any fundamental elementary function [3] $g(val(u))$ add node v to $\mathcal{V}_{f_i}$ and directed edge (u, v) to $\mathcal{E}_{f_i}$

3. The process finishes when no more computation is left.

In other words, a computational graph is a representation of a composite function as a network of connected nodes, where each node is an elementary operation or elementary function. Let us illustrate this definition with a simple example $f_i(x_1, x_2) = e^{x_1} + x_1 x_2 - \cos(x_2)$.

Node Set $\mathcal{V}_{f_i}$	**Operation**	**Edge Set $\mathcal{E}_{f_i}$**
v_1	x_1	
v_2	x_2	
v_3	e^{x_1}	(v_1, v_3)
v_4	$x_1 x_2$	$(v_1 v_4)$, (v_2, v_4)
v_5	$\frac{\pi}{2}$	
v_6	$x_2 + \frac{\pi}{2}$	(v_2, v_6), (v_5, v_6)
v_7	$\sin\left(x_2 + \frac{\pi}{2}\right)$	(v_6, v_7)
v_8	$e^{x_1} + x_1 x_2$	(v_3, v_8), (v_4, v_8)
v_9	$e^{x_1} + x_1 x_2 - \sin\left(x_2 + \frac{\pi}{2}\right)$	(v_7, v_9), (v_8, v_9)

Table 3: Decomposition of $f(x_1, x_2) = e^{x_1} + x_1 x_2 - \cos(x_2)$ into elementary operations and functions

[3] There are eight fundamental elementary function: $g_1(x) = c$, $g_2(x) = x$, $g_3(x) = \frac{1}{x}$, $g_4(x) = \sqrt[n]{x}$, $g_5(x) = \sin(x)$, $g_6(x) = e^x$, $g_7(x) = \ln(x)$, $g_8(x) = arccos(x)$

In the first and second columns, Table 3 collects the set of nodes $\mathcal{V}_{f_i}$ of a computational graph $\mathcal{G}_{f_i}$ with corresponding elementary arithmetic operations and fundamental elementary functions[4]. The third column maintains the edge set $\mathcal{E}_{f_i}$ following the relation between numerical operations. Figure 14 presents the computational graph for function $f_i(x_1, x_2)$.

AD can operate in two modes: *forward* and *reverse*. In the forward mode, for given vectors

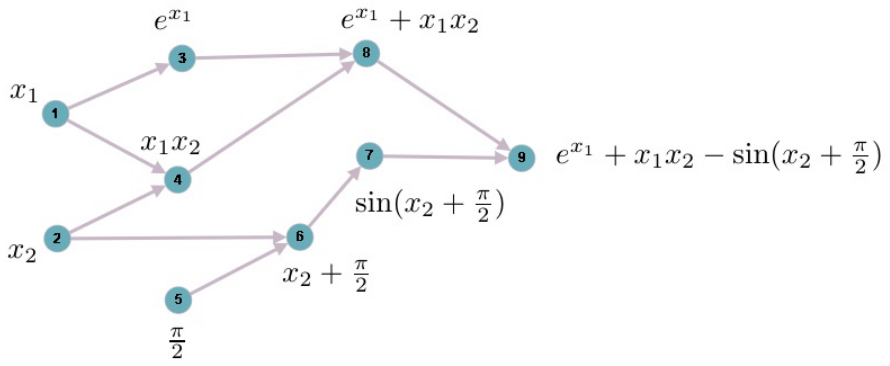


Figure 14: Computational graph for $f(x_1, x_2) = e^{x_1} + x_1x_2 - \cos(x_2)$

$\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^p$, one can compute a directional derivative $\nabla^\mathsf{T} f_i(\boldsymbol{a})\boldsymbol{b}$ by traversing the computational graph $\mathcal{G}_{f_i}$ in a forward way and applying a chain rule. The formal description of the AD method in forward mode is given in Algorithm 13:

Algorithm 13 : Forward Mode

1: **Input:** Function $f_i : \mathbb{R}^p \to \mathbb{R}$, computational graph $\mathcal{V}_{f_i}$, vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^p$.
2: **Output:** $\nabla^\mathsf{T} f_i(\boldsymbol{a})\boldsymbol{b}$.
3: Set $val(v_i) = [\boldsymbol{a}]_i$ for all input nodes $v_1, \ldots, v_p$.
4: Compute $val(v_l)$ for the rest nodes using $\mathcal{G}_{f_i}$.
5: Set derivative $\dot{val}(v_i) = [\boldsymbol{b}]_i$ for all input nodes $v_1, \ldots, v_p$.
6: Compute derivative $\dot{val}(v)$ recursively for the rest of the nodes using a chain rule.
7: Output $\dot{val}(w) = \nabla^\mathsf{T} f_i(\boldsymbol{a})\boldsymbol{b}$, where w is the last node in $\mathcal{V}_{f_i}$.

To illustrate the forward mode of AD, let us consider $f_i(x_1, x_2)$ again. Namely, let us consider the computation of $\nabla^\mathsf{T} f_i(\boldsymbol{a})\boldsymbol{b}$ with $\boldsymbol{a} = (1, 1)^\mathsf{T}$ and $\boldsymbol{b} = (2, 5)^\mathsf{T}$ in Table 4. The time and space complexity of Algorithm 13 are characterized by the size of the computational graph, i.e $\mathcal{O}(|\mathcal{V}_{f_i}| + |\mathcal{E}_{f_i}|)$.

[4]The term $\cos(x_2)$ is written in the equivalent form $\sin\left(x_2 + \frac{\pi}{2}\right)$

Computation of values	**Computation of derivatives**
$val(v_1) = a_1 = 1$	$\dot{val}(v_1) = b_1 = 2$
$val(v_2) = a_2 = 1$	$\dot{val}(v_2) = b_2 = 5$
$val(v_3) = e^{val(v_1)} = e$	$\dot{val}(v_3) = e^{val(v_1)}\dot{val}(v_1) = 2e$
$val(v_4) = val(v_1)val(v_2) = 1$	$\dot{val}(v_4) = val(v_1)\dot{val}(v_2) + \dot{val}(v_1)val(v_2) = 7$
$val(v_5) = \frac{\pi}{2}$	$\dot{val}(v_5) = 0$
$val(v_6) = val(v_2) + val(v_5) = 1 + \frac{\pi}{2}$	$\dot{val}(v_6) = \dot{val}(v_2) + \dot{val}(v_5) = 5$
$val(v_7) = \sin(val(v_6)) = \cos(1)$	$\dot{val}(v_7) = \cos(val(v_6))\dot{val}(v_6) = -5\sin(1)$
$val(v_8) = val(v_3) + val(v_4) = e + 1$	$\dot{val}(v_8) = \dot{val}(v_3) + \dot{val}(v_4) = 2e + 7$
$val(v_9) = val(v_8) + val(v_7) = e + 1 - \cos(1)$	$\dot{val}(v_9) = \dot{val}(v_8) + \dot{val}(v_7) = 2e + 7 + 5\sin(1)$

Table 4: Computation of a directional derivative $\nabla^\mathsf{T} f_i(\boldsymbol{a})\boldsymbol{b}$ for $f(x_1, x_2) = e^{x_1} + x_1x_2 - \cos(x_2)$ with $\boldsymbol{a} = (1, 1)^\mathsf{T}, \boldsymbol{b} = (2, 5)^\mathsf{T}$ using AD in forward mode.

In the reverse mode, for a given vector $\boldsymbol{a} \in \mathbb{R}^p$, the AD method calculates the whole gradient vector $\nabla f_i(\boldsymbol{a})$ evaluated at $\boldsymbol{a}$. It starts with the final node $w \in \mathcal{V}_{f_i}$ with a corresponding value $val(w) = f_i$ and then computes derivatives with respect to each sub-expression recursively using a chain rule and computational graph $\mathcal{G}_{f_i}$. As a result, AD in backward mode traverses the graph in the reverse direction and constructs a collection of variables referred to as *adjoin* and defined:

$$\bar{val}(v) = \frac{\partial f_i}{\partial val(v)}$$

where $v \in \mathcal{V}_{f_i}$. Algorithm 14 presents the detailed description of AD in backward mode:

Algorithm 14 : **Backward Mode**

1: **Input:** Function $f_i(\boldsymbol{x}) : \mathbb{R}^p \to \mathbb{R}$, computational graph $\mathcal{V}_{f_i}$, vector $\boldsymbol{a} \in \mathbb{R}^p$.
2: **Output:** Vector $\nabla f_i(\boldsymbol{a})$.
3: Set the first adjoint variable $\bar{val}(w) = \frac{\partial f_i}{\partial val(w)} = 1$.
4: Compute adjoint variables $\bar{val}(v)$ for all $v \in \mathcal{V}_{f_i}$ recursively using $\bar{val}(v) = \frac{\partial f_i}{\partial val(v)}|_{\boldsymbol{a}}$ and dependency given in computational graph $\mathcal{G}_{f_i}$.
5: Output $\frac{\partial f_i}{\partial [\boldsymbol{x}]_j}(\boldsymbol{a}) = \bar{val}(v_j)$ for $j = 1, \ldots, p$.

To illustrate the backward mode of AD, we calculate the gradient of function $f_i(x_1, x_2)$ at point $\boldsymbol{a} = (1, 1)^\mathsf{T}$ in Table 5. Similarly to the forward mode, the time and space complexity of Algorithm 14 are characterized by the size of the computational graph, i.e $\mathcal{O}(|\mathcal{V}_{f_i}| + |\mathcal{E}_{f_i}|)$.

Computation of variables	**Computation of adjoin variables**
$val(v_1) = a_1 = 1$	$\bar{val(w)} = 1$
$val(v_2) = a_2 = 1$	$\bar{val(v_9)} = \frac{\partial f_i}{\partial val(v_9)} = \frac{\partial f_i}{\partial val(w)} \frac{\partial val(w)}{\partial val(v_9)} = \bar{val(w)} = 1$
$val(v_3) = e$	$\bar{val(v_8)} = \frac{\partial f_i}{\partial val(v_8)} = \frac{\partial f_i}{\partial val(v_9)} \frac{\partial val(v_9)}{\partial val(v_8)} = \bar{val(v_9)} = 1$
$val(v_4) = 1$	$\bar{val(v_7)} = \frac{\partial f_i}{\partial val(v_7)} = \frac{\partial f_i}{\partial val(v_9)} \frac{\partial val(v_9)}{\partial val(v_7)} = -\bar{val(v_9)} = -1$
$val(v_5) = \frac{\pi}{2}$	$\bar{val(v_6)} = \frac{\partial f_i}{\partial val(v_6)} = \frac{\partial f_i}{\partial val(v_7)} \frac{\partial val(v_7)}{\partial val(v_6)} = \bar{val(v_7)} \cos(val(v_6)) = \sin(1)$
$val(v_6) = 1 + \frac{\pi}{2}$	$\bar{val(v_5)} = \frac{\partial f_i}{\partial val(v_5)} = \frac{\partial f_i}{\partial val(v_6)} \frac{\partial val(v_6)}{\partial val(v_5)} = val(v_6) = \sin(1)$
$val(v_7) = \cos(1)$	$\bar{val(v_4)} = \frac{\partial f_i}{\partial val(v_4)} = \frac{\partial f_i}{\partial val(v_8)} \frac{\partial val(v_8)}{\partial val(v_4)} = \bar{val(v_8)} = 1$
$val(v_8) = e + 1$	$\bar{val(v_3)} = \frac{\partial f_i}{\partial val(v_3)} = \frac{\partial f_i}{\partial val(v_8)} \frac{\partial val(v_8)}{\partial val(v_3)} = \bar{val(v_8)} = 1$
$val(v_9)e + 1 - \cos(1)$	$\bar{val(v_2)} = \frac{\partial f_i}{\partial val(v_2)} = \frac{\partial f_i}{\partial val(v_6)} \frac{\partial val(v_6)}{\partial val(v_2)} + \frac{\partial f_i}{\partial val(v_4)} \frac{\partial val(v_4)}{\partial val(v_2)} =$ $\bar{val(v_6)} + \bar{val(v_4)} val(v_1) = \sin(1) + 1$
	$\bar{val(v_1)} = \frac{\partial f_i}{\partial val(v_1)} = \frac{\partial f_i}{\partial val(v_4)} \frac{\partial val(v_4)}{\partial val(v_1)} + \frac{\partial f_i}{\partial val(v_3)} \frac{\partial val(v_3)}{\partial val(v_1)} =$ $\bar{val(v_4)} val(v_2) + \bar{val(v_3)} e^{val(v_1)} = 1 + e$
	$\frac{\partial f_i}{\partial x_1}(\boldsymbol{a}) = \bar{val(v_1)} = 1 + e \quad \frac{\partial f_i}{\partial x_2}(\boldsymbol{a}) = \bar{val(v_2)} = \sin(1) + 1$

Table 5: Computation of gradient $\nabla f(\boldsymbol{a})$ for $f(x_1, x_2) = e^{x_1} + x_1 x_2 - \cos(x_2)$ at vector $\boldsymbol{a} = [1, 1]^\mathsf{T}$ using AD in backward mode.

The application of AD methods for computing Hessian-vector products $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)$ is based on the following observation:

$$\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i = \nabla_{\boldsymbol{x}}[\nabla_{\boldsymbol{x}}^\mathsf{T} f_i(\boldsymbol{x})\boldsymbol{\mathcal{Y}}_i]|_{\boldsymbol{x}=\boldsymbol{\mathcal{Y}}_i}$$

Therefore, using the forward mode for computing directional derivatives, one can construct function $g_i(\boldsymbol{x}) = \nabla^\mathsf{T} f_i(\boldsymbol{x})\ \boldsymbol{\mathcal{Y}}_i$ and then calculate gradient $\nabla g_r\ (\boldsymbol{\mathcal{Y}}_i)$ by calling backward mode on $g_i(\boldsymbol{x})$. We describe these steps formally in Algorithm 15:

Algorithm 15 : **Hessian-vector Product Algorithm via AD**

1: **Input:** Function $f_i(\boldsymbol{x}) : \mathbb{R}^p \to \mathbb{R}$, its computational graph $\mathcal{G}_{f_i}$, vector $\boldsymbol{\mathcal{Y}}_i$.

2: **Output:** Vector $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$.

3: Construct $g_i\ (\boldsymbol{x}) = \nabla^\mathsf{T} f_i(\boldsymbol{x})\boldsymbol{\mathcal{Y}}_i$ using Algorithm 13.

4: Compute gradient $\nabla g_i\ (\boldsymbol{\mathcal{Y}}_i)$ using Algorithm 14.

5: Set $\nabla g_i\ (\boldsymbol{\mathcal{Y}}_i) = \nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$.

As was mentioned, the time and space complexity of AD in forward or backward modes are characterized by the size of the corresponding computational graph $\mathcal{G}_{f_i}$. Therefore, to guarantee the efficient computation, one needs to ensure graph $\mathcal{G}_{f_i}$ has size $\mathcal{O}(p^{1+\delta})$ for some $\delta < 1$. Although for an arbitrary function f_i this cannot be assured, many loss functions used in machine learning (linear/logistic regression) have computational graphs of size $\mathcal{O}(p)$[5]. To illustrate this, in the Appendix, we construct the computational graphs for two popular machine learning objectives:

Linear Regression Loss:

$$f_i(\boldsymbol{x}) = \left(a - \Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}\right)^2 + \mu_i ||\boldsymbol{x}||_2^2$$

Logistic Regression Loss:

$$f_i(\boldsymbol{x}) = -\left[a \log \frac{1}{1 + e^{-\Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}}} + (1-a)\log\left(1 - \frac{1}{1 + e^{-\Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}}}\right)\right] + \mu_i ||\boldsymbol{x}||_2^2$$

where $(\boldsymbol{b}, a)$ represents input/output pair,$\Phi(\boldsymbol{b})$ is a feature representation of the input vector $\boldsymbol{b}$, and μ_i is a regularization coefficient. In Figure 15 we demonstrate the effect of dimensionality on the time complexity of one iteration of Distributed Newton method and standard gradient descent applied for the linear regression model. For the former method, the slope of the constructed line is characterized by the square root of the condition number of the local objective.

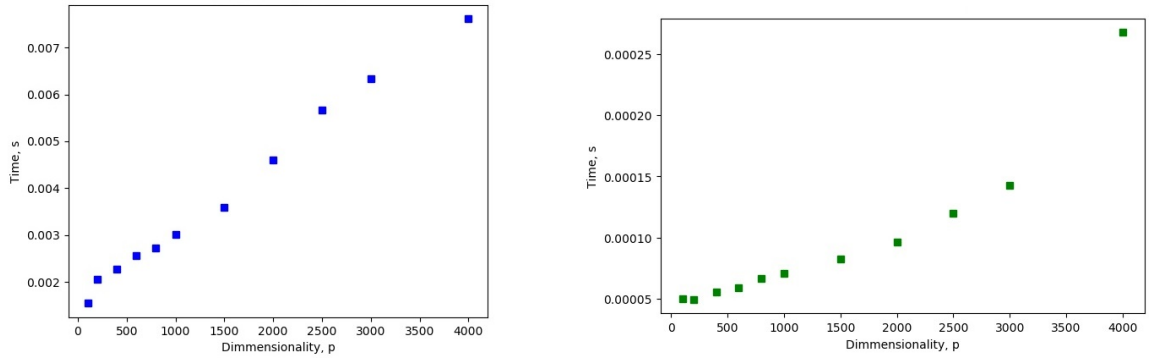


Figure 15: The effect of dimensionality using Automatic Differentiation

[5]For Neural Network, the computational graph is proportional to $\mathcal{O}(p^2)$. In this case, the complexity of Hessian-vector computation is equal to the complexity of gradient computation and bounded by $\mathcal{O}(p^2)$

3.2.8. Too Many Training Points

Earlier in Section 3.2.1, we considered empirical risk minimization where the total number of additive terms in the risk objective is equal to the number of processors, i.e. nodes of graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. Consequently, this objective function is distributed across the nodes such that each receives exactly one local function $f_i(\cdot)$. However, in this section we focus on the large-scale setting, where each node $i \in \mathbb{V}$ accumulates a collection of N functions $\{\ell_{ij}(\cdot)\}_{j=1}^{N}$ from the risk objective. We would like to understand the effect of the parameter N on the performance of the Distributed Newton Method and discuss efficient methods to cope with a large number of additive terms.

Large-scale Setting for Empirical Risk Minimization

We start with the formulation of empirical risk minimization motivated by a machine learning setup. For a given collection of data points $\mathcal{D} = \{a, \boldsymbol{b}\}_{k=1}^{\mathcal{N}}$ and loss functions $l(\cdot) : \mathbb{R}\times\mathbb{R} \to \mathbb{R}^+$ the associated empirical risk is defined as:

$$\mathcal{R}_{[\mathcal{D},\ell]}(\boldsymbol{x}) = \frac{1}{\mathcal{N}} \sum_{k=1}^{\mathcal{N}} \ell(h(\boldsymbol{b}_k, \boldsymbol{x}), a_k) = \frac{1}{\mathcal{N}} \sum_{k=1}^{\mathcal{N}} \ell_k(\boldsymbol{x})$$

where $h()$ is a prediction model parameterized by $\boldsymbol{x}$ and $\ell_k(\boldsymbol{x}) = \ell(h(\boldsymbol{b}_k, \boldsymbol{x}), a_k) + \eta||\boldsymbol{x}||_2^2$ is a regularized loss imposed by model $h()$ on a training point $(a_k, \boldsymbol{b}_k)$. The next step is to distribute the empirical risk among n computational units (i.e. processors) such that each receives $N = \frac{\mathcal{N}}{n}$ training examples [6]. As a result, we achieve a large-scale form of empirical risk minimization:

$$\min_{\boldsymbol{x}\in\mathbb{R}^p} \mathcal{R}_{[\mathcal{D},\ell]}(\boldsymbol{x}) \iff \min_{\boldsymbol{x}\in\mathbb{R}^p} \sum_{k=1}^{\mathcal{N}} \ell_k(\boldsymbol{x})$$

and the corresponding global consensus problem has the following form:

$$\min_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n} f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \min_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n} \sum_{i=1}^{n} \sum_{j=1}^{N} \ell_{ij}(\boldsymbol{x}_i) \tag{3.25}$$

$$s.t.\ \boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n \in \mathbb{R}^p$$

[6] For simplicity we assume that $\frac{\mathcal{N}}{n} \in \mathbb{N}$

It is easy to see that the above formulation can be converted to the original global consensus problem (3.9) by simply denoting the local objective for node $i \in \mathbb{V}$ as:

$$f_i(\boldsymbol{x}_i) = \sum_{j=1}^{N} \ell_{ij}(\boldsymbol{x}_i) \tag{3.26}$$

This structure of the local objective will have two effects on the performance of the proposed Distributed Newton Method. First of all, the time for computation of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_p$ will scale linearly with N. Indeed, following the discussion in the previous section, each node i computes N Hessian-vector products $\nabla^2 \ell_{ij}(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$ due to:

$$\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i = \sum_{j=1}^{N} \nabla^2 \ell_{ij}(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$$

Therefore, using the Hessian-free techniques discussed in Section 3.2.7, the total time complexity for each node $i \in \mathbb{V}$ increases by factor N. The second effect is related to a primal-dual computation and will be discussed in detail in the next paragraphs.

Primal Variable Computation

Our proposed Distributed Newton Method is formulated for the dual problem; however, it is crucial for the consecutive iterations to maintain the "bridge" between primal and dual variables. This relation was established in Section 3.2.2 by posing for the given dual vectors $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_p$. the following local optimization problems:

$$\min_{[\boldsymbol{y}_j]_1, \ldots, [\boldsymbol{y}_p]_i} f_i([\boldsymbol{y}_j]_1, \ldots, [\boldsymbol{y}_p]_i) + \sum_{j=1}^{p} [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_j]_i [\boldsymbol{y}_j]_i \qquad i = 1, \ldots, n$$

Therefore, using the form of local function $f_i(\cdot)$ from (3.26), the connection between primal and dual variables is given as:

$$\min_{[\boldsymbol{y}_j]_1, \ldots, [\boldsymbol{y}_p]_i} \sum_{j=1}^{N} \ell_{ij}([\boldsymbol{y}_j]_1, \ldots, [\boldsymbol{y}_p]_i) + \sum_{j=1}^{p} [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_j]_i [\boldsymbol{y}_j]_i \qquad i = 1, \ldots, n \tag{3.27}$$

This is a collection of unconstrained optimization problems with strongly convex objectives formulated as finite sum minimizations. It is worth mentioning that for some prediction models

h optimization problem (3.27) has a closed-form solution. For example, let us consider a regularized linear regression case with $h(\boldsymbol{b}, \boldsymbol{x}) = \Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}$ and quadratic loss function $\ell(h(\boldsymbol{b}, \boldsymbol{x}), a) = (a - h(\boldsymbol{b}, \boldsymbol{x}))^2 + \eta||\boldsymbol{x}||_2^2$. Therefore, primal-dual dependence is given by quadratic optimization:

$$\min_{\boldsymbol{\mathcal{Y}}_i} \sum_{j=1}^{N} \left(a_{ij} - \Phi^{\mathsf{T}}(\boldsymbol{b_{ij}})\boldsymbol{\mathcal{Y}}_i\right)^2 + N\eta||\boldsymbol{\mathcal{Y}}_i||_2^2 + \boldsymbol{c}_i^{\mathsf{T}}\boldsymbol{\mathcal{Y}}_i$$

with $\boldsymbol{c}_i = ([\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_1]_i, \ldots, [[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_p]_i)^{\mathsf{T}}$. Applying simple derivation one can obtain a closed-form expression for the minimizer of the above least square problem:

$$\boldsymbol{\mathcal{Y}}_i^* = -\frac{1}{2}\left[\sum_{j=1}^{N} \Phi(\boldsymbol{b_{ij}})\Phi^{\mathsf{T}}(\boldsymbol{b_{ij}}) + N\eta\boldsymbol{I}_{p\times p}\right]^{-1} \boldsymbol{c}_i$$

Further, we discuss techniques to compute optimal vector $\boldsymbol{\mathcal{Y}}_i^*$ efficiently without computing and storing the Hessian inverse. Unfortunately, only a few prediction models allow a closed-form solution for primal-dual dependence. For instance, there is no explicit relation between primal and dual variables for logistic regression models. To address these cases, in the next paragraphs we describe both deterministic and stochastic methods to solve the optimization problem (3.27).

Approximate Newton Method

As mentioned in the previous section, the "bridge" between primal and dual variables can be established by locally solving the optimization problem (3.27) in each node $i \in \mathbb{V}$. Let

$$\hat{f}_i(\boldsymbol{\mathcal{Y}}_i) = \sum_{j=1}^{N} \ell_{ij}(\boldsymbol{\mathcal{Y}}_i) + \sum_{j=1}^{p} [\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_j]_i [\boldsymbol{y}_j]_i$$

be a local objective for the optimization problem (3.27). Then one can rewrite (3.27) simply as a strongly convex problem:

$$\min_{\boldsymbol{\mathcal{Y}}_i} \hat{f}_i(\boldsymbol{\mathcal{Y}}_i) \tag{3.28}$$

Moreover, due to Assumption 3.2.1, the above objective function satisfies standard premises for a centralized Newton method Boyd and Vandenberghe (2004a):

$$\begin{aligned} &\gamma \boldsymbol{I} \preceq \nabla^2 \hat{f}_i(\cdot) \preceq \Gamma \boldsymbol{I} \\ &\left|\left|\nabla^2 \hat{f}_i(\boldsymbol{\mathcal{Y}}_i) - \nabla^2 \hat{f}_i(\tilde{\boldsymbol{\mathcal{Y}}}_i)\right|\right|_2 \leq \delta ||\boldsymbol{\mathcal{Y}}_i - \tilde{\boldsymbol{\mathcal{Y}}}_i||_2, \quad \forall \ \boldsymbol{\mathcal{Y}}_i, \tilde{\boldsymbol{\mathcal{Y}}}_i \in \mathbb{R}^p \end{aligned} \tag{3.29}$$

Therefore, considering its fast convergence rate, it is reasonable to apply Newton's method to solve the optimization problem (3.28):

$$\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} + \alpha_{it} \Delta \boldsymbol{\mathcal{Y}}_i^{[t]} \tag{3.30}$$

where α_{it} is a step-size computed by a backtracking line search, and $\Delta \boldsymbol{\mathcal{Y}}_i^{[t+1]}$ is a Newton direction at iteration t given as:

$$\Delta \boldsymbol{\mathcal{Y}}_i^{[t]} = -[\nabla^2 \hat{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})]^{-1} \nabla \hat{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) \tag{3.31}$$

The main challenge now is to efficiently compute the Newton direction from both time and memory perspectives. To achieve this goal, let us first introduce the following notations:

$$\begin{aligned} &\boldsymbol{H}_{ij} \triangleq \nabla^2 \ell_{ij}, \qquad \boldsymbol{H}_i \triangleq \nabla^2 \hat{f}_i = \sum_{j=1}^{N} \boldsymbol{H}_{ij}, \\ &\boldsymbol{h}_{ij} \triangleq \nabla \ell_{ij}, \qquad \boldsymbol{h}_i \triangleq \nabla \hat{f}_i = \sum_{j=1}^{N} \boldsymbol{h}_{ij} + \boldsymbol{c}_i \end{aligned}$$

Therefore, for Newton direction we can immediately write (for clarity reasons we drop further iteration index t):

$$\Delta \boldsymbol{\mathcal{Y}}_i = -\boldsymbol{H}_i^{-1} \boldsymbol{h}_i$$

Following the polynomial approximation scheme proposed in Section 2.2.2 and using Chebyshev polynomials with guarantees (3.29), the $\zeta-$ approximate solution[7] of the above system can be

[7] Recall, $\zeta-$approximate solution implies $||\widetilde{\Delta \boldsymbol{\mathcal{Y}}}_i - \Delta \boldsymbol{\mathcal{Y}}_i^*||_{\boldsymbol{H}_i} \leq \zeta ||\Delta \boldsymbol{\mathcal{Y}}_i^*||_{\boldsymbol{H}_i}$

computed as follows:

$$\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i = -\boldsymbol{H}_i^{-1}\left[\frac{T_{k_\zeta}\left(\frac{\Gamma+\gamma}{\Gamma-\gamma}\right)\boldsymbol{I} - T_{k_\zeta}\left(\frac{(\Gamma+\gamma)\boldsymbol{I}-2\boldsymbol{H}_i}{\Gamma-\gamma}\right)}{T_{k_\zeta}\left(\frac{\Gamma+\gamma}{\Gamma-\gamma}\right)}\right]\boldsymbol{h}_i \tag{3.32}$$
$$\text{with} \qquad k_\zeta = \lceil \frac{1}{2}\left(\sqrt{\frac{\Gamma}{\gamma}}+1\right)\ln\frac{2}{\zeta}\rceil$$

Similar to Section 2.2.2, the fast computation of vector (3.32) can be performed by exploiting the recursive relation of Chebyshev polynomials (2.13). Denote by

$$\begin{aligned}
\boldsymbol{\Delta}_{i,k} &= \boldsymbol{H}_i^{-1}\left[T_k\left(\frac{\Gamma+\gamma}{\Gamma-\gamma}\right)\boldsymbol{I} - T_k\left(\frac{(\Gamma+\gamma)\boldsymbol{I}-2\boldsymbol{H}_i}{\Gamma-\gamma}\right)\right]\boldsymbol{h}_i \\
\boldsymbol{\Omega}_{i,k} &= T_k\left(\frac{(\Gamma+\gamma)\boldsymbol{I}-2\boldsymbol{H}_i}{\Gamma-\gamma}\right)\boldsymbol{h}_i \\
\Theta_k &= T_k\left(\frac{\Gamma+\gamma}{\Gamma-\gamma}\right)
\end{aligned}$$

Therefore, the solution vector (3.32) can be written as $\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i = -\frac{\boldsymbol{\Delta}_{i,k_\zeta}}{\Theta_{k_\zeta}}$ and the recursive relation gives:

$$\begin{aligned}
\boldsymbol{\Delta}_{i,k} &= 2\frac{\Gamma+\gamma}{\Gamma-\gamma}\boldsymbol{\Delta}_{i,k-1} - 2\boldsymbol{\Delta}_{i,k-2} + \frac{4}{\Gamma-\gamma}\boldsymbol{\Omega}_{i,k-1} \\
\boldsymbol{\Omega}_{i,k} &= 2\left[\frac{\Gamma+\gamma}{\Gamma-\gamma}\boldsymbol{I} - \frac{2}{\Gamma-\gamma}\boldsymbol{H}_i\right]\boldsymbol{\Omega}_{i,k-1} - \boldsymbol{\Omega}_{i,k-2} \\
\Theta_k &= 2\frac{\Gamma+\gamma}{\Gamma-\gamma}\Theta_{k-1} - \Theta_{k-2}
\end{aligned}$$

with initials given by:

$$\begin{aligned}
\boldsymbol{\Delta}_{i,1} &= \frac{2}{\Gamma-\gamma}\boldsymbol{h}_i & \boldsymbol{\Omega}_{i,1} &= \frac{\Gamma+\gamma}{\Gamma-\gamma}\boldsymbol{h}_i - \frac{2}{\Gamma-\gamma}\boldsymbol{H}_i\boldsymbol{h}_i & \Theta_1 &= \frac{\Gamma+\gamma}{\Gamma-\gamma} \\
\boldsymbol{\Delta}_{i,0} &= \boldsymbol{0} & \boldsymbol{\Omega}_{i,0} &= \boldsymbol{h}_i & \Theta_0 &= 1
\end{aligned}$$

As a result we arrive at Algorithm 16 that computes $\zeta-$approximation of Newton direction for iteration scheme (3.30) and executes $\mathcal{O}(k_\zeta)$ matrix vector multiplications of the form $\boldsymbol{H}_i\boldsymbol{v}$:

Each such multiplication can be further decomposed into a summation of Hessian-vector products:

$$\boldsymbol{H}_i\boldsymbol{v} = \boldsymbol{H}_{i1}\boldsymbol{v} + \boldsymbol{H}_{i2}\boldsymbol{v} + \ldots + \boldsymbol{H}_{iN}\boldsymbol{v}$$

Algorithm 16 : **Approximate Newton Direction Computation**

Input: Functions $\{\ell_{ij}(\cdot)\}_{j=1}^{N}$, current iteration vector $\boldsymbol{\mathcal{Y}}_i$, vector $\boldsymbol{c}_i = \left([\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_1]_i, \ldots, [[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_p]_i\right)^{\intercal}$, parameters Γ, γ and ζ
Output: $\zeta-$approximation of Newton direction $\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i$.
Set $w_1 = \frac{\Gamma+\gamma}{\Gamma-\gamma}$ and $w_2 = \frac{2}{\Gamma-\gamma}$, and $k_\zeta = \lceil \frac{1}{2}\left(\sqrt{\frac{\Gamma}{\gamma}}+1\right)\ln\frac{2}{\zeta}\rceil$
Compute $\boldsymbol{h}_i = \sum_{j=1}^{N}\nabla\ell_{ij}(\boldsymbol{\mathcal{Y}}_i) + \boldsymbol{c}_i$ and $\boldsymbol{H}_i\boldsymbol{h}_i = \sum_{j=1}^{N}\boldsymbol{H}_{ij}(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{h}_i(\boldsymbol{\mathcal{Y}}_i)$
Set $\boldsymbol{\Delta}_{i,0} = \boldsymbol{0}$, $\boldsymbol{\Omega}_{i,0} = \boldsymbol{h}_i$ and $\Theta_0 = 1$.
    $\boldsymbol{\Delta}_{i,1} = w_2\boldsymbol{h}_i$, $\boldsymbol{\Omega}_{i,1} = w_1\boldsymbol{h}_i - w_2\boldsymbol{H}_i\boldsymbol{h}_i$ and $\Theta_1 = w_1$.
for $k = 2$ to k_ζ **do**
    $\Theta_k = 2w_1\Theta_{k-1} - \Theta_{k-2}$.
    $\boldsymbol{\Omega}_{i,k} = 2\left[w_1\boldsymbol{I} - w_2\boldsymbol{H}_i\right]\boldsymbol{\Omega}_{i,k-1} - \boldsymbol{\Omega}_{i,k-2}$.
    $\boldsymbol{\Delta}_{i,k} = 2w_1\boldsymbol{\Delta}_{i,k-1} - 2\boldsymbol{\Delta}_{i,k-2} + 2w_2\boldsymbol{\Omega}_{i,k-1}$
end for
Set $\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i = -\frac{\boldsymbol{\Delta}_{i,k_\zeta}}{\Theta_{k_\zeta}}$

Therefore, the total running time of Algorithm 16 can be bounded by $\mathcal{O}\left(N\sqrt{\frac{\Gamma}{\gamma}}\mathcal{T}\ln\frac{1}{\zeta}\right)$ where $\mathcal{T}$ is a running time for computing Hessian-vector product $\nabla^2\ell_{ij}(\cdot)\boldsymbol{v}$. In particular, for a wide range of machine learning objectives $\ell_{ij}(\cdot)$, including linear and logistic regression models, etc., this bound boils down to $\mathcal{O}\left(Np\sqrt{\frac{\Gamma}{\gamma}}\ln\frac{1}{\zeta}\right)$ by exploiting Hessian-free methods from Section 3.2.7.

Next, we study the effect of the precision parameter ζ on the convergence properties of the Newton method (3.30). Our first result explores the change of local gradient $\hat{f}_i(\cdot)$ between two consecutive iterations of the Approximated Newton Method:

$$\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} + \alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i^{[t]} \tag{3.33}$$

where the approximated Newton direction vector $\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i^{[t]}$ computed by Algorithm 16 and α_{it} is a step size chosen according to the approximated backtracking line search procedure, presented in Algorithm 17

Lemma 3.2.10 *For each node $i \in \mathbb{V}$ consider iteration scheme given by (3.33) and denote*[8]

$$\boldsymbol{\epsilon}_i^{[t]} = \boldsymbol{H}_i^{[t]}\widetilde{\Delta\boldsymbol{\mathcal{Y}}}_i^{[t]} + \boldsymbol{h}_i^{[t]}$$

[8]We denote $\boldsymbol{H}_i^{[t]} = \boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})$ and $\boldsymbol{h}_i^{[t]} = \boldsymbol{h}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})$.

be the approximation error vector corresponding to $\widetilde{\Delta \boldsymbol{\mathcal{Y}}_i}^{[t]}$. *Then for any* $\alpha_{it} \in (0, 1]$

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1 - \alpha_{it})||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}^2 \delta \frac{1}{\gamma^2}||\boldsymbol{h}_i^{[t]}||_2^2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \alpha_{it}^2 \delta \frac{1}{\gamma^2}||\boldsymbol{\epsilon}_i^{[t]}||_2^2 \tag{3.34}$$

Proof See Appendix.

Algorithm 17 : **Approximated Backtracking Line Search**

1: **Input:** The constants $\sigma \in (0, \frac{1}{2}]$, $\beta \in (0, 1)$, parameters Γ, γ, δ and ζ, gradients $||\boldsymbol{h}_i^{[t+1]}||_2$, $||\boldsymbol{h}_i^{[t]}||_2$.
2: **Output:** $\alpha_{it}-$ the step size for Approximated Newton Method
3: Set $m_i = 0$.
4: **while** $||\boldsymbol{h}_i^{[t+1]}||_2 > (1 - \sigma\beta^{m_i})||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$ **do**
5: $\quad m_i = m_i + 1$
6: **end while**
7: Set $\alpha_{it} = \beta^{m_i}$

Next, we establish guarantees on step size α_{it} computed by the approximated backtracking line search procedure:

Lemma 3.2.11 *Let step size* α_{it} *is chosen according to Algorithm 17 and let* $\boldsymbol{h}_i^{[t]}$ *be the gradient of* $\hat{f}_i$ *evaluated at* $\boldsymbol{\mathcal{Y}}_i^{[t]}$. *Then,*

1. *If* $||\boldsymbol{h}_i^{[t]}||_2 \leq \frac{\gamma^2}{2\delta}$, *then* $\alpha_{it} = 1$.

2. *If* $||\boldsymbol{h}_i^{[t]}||_2 > \frac{\gamma^2}{2\delta}$, *then* $\alpha_{it} \geq \beta\frac{\gamma^2}{2\delta||\boldsymbol{h}_i^{[t]}||_2}$.

where parameters Γ, γ, δ *are given in (3.29)*

Proof See Appendix.

The next theorem studies the convergence properties of the Approximated Newton Method given in (3.33):

Theorem 3.2.12 *Consider the iteration scheme* $\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} + \alpha_{it}\widetilde{\Delta \boldsymbol{\mathcal{Y}}_i}^{[t]}$ *where* α_{it} *is a step size defined by Algorithm 17. Let parameters* Γ, γ, δ *be given according to conditions (3.29) and accuracy parameter for Algorithm 16 satisfies* $\zeta \leq 2\sqrt{\frac{\gamma}{\Gamma}}$. *Then, this iteration scheme exhibits two convergence phases:*

1. ***Strict Decrease Phase:*** *If* $||\boldsymbol{h}_i^{[t]}||_2 > \frac{\gamma^2}{2\delta}$, *then*

$$||\boldsymbol{h}_i^{[t+1]}||_2 - ||\boldsymbol{h}_i^{[t]}||_2 \leq -\frac{\beta\gamma^2}{8\delta}$$

2. ***Quadratic Decrease Phase:*** *If* $||\boldsymbol{h}_i^{[t]}||_2 \leq \frac{\gamma^2}{2\delta}$, *then*

$$||\boldsymbol{h}_i^{[t+m]}||_2 \leq \frac{1}{2^{2^m}\frac{\delta}{\gamma^2}} + \tilde{B}_i + \frac{\hat{\Lambda}_i}{\frac{\delta}{\gamma^2}}\left[\frac{2^{2^m-1}-1}{2^{2^m}}\right] \tag{3.35}$$

where:

$$\tilde{B}_i = \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{2\delta}\left[1 + \frac{\zeta}{2}\sqrt{\frac{\gamma}{\Gamma}}\right] \sim \mathcal{O}(\zeta)$$
$$\hat{\Lambda}_i = 4\frac{\delta}{\gamma^2}\tilde{B}_i\left[1 + \frac{\delta}{\gamma^2}\tilde{B}_i\right] \sim \mathcal{O}(\zeta)$$

Proof See Appendix.

The following result follows directly from (3.35) and establishes the asymptotic limit for gradient $\boldsymbol{h}_i^{[t]}$ after passing the strict decrease phase:

Corollary 3.2.13 *Let* t_0 *designate the first iteration such that* $||\boldsymbol{h}_i^{t_0}||_2 \leq \frac{\gamma^2}{2\delta}$. *Then for the next iterations the norm of the gradient* $\boldsymbol{h}_i$ *converges quadratically to*

$$\lim_{l\to\infty} ||\boldsymbol{h}_i^{t_0+l}||_2 = \tilde{B}_i + \frac{1}{2}\frac{\hat{\Lambda}_i\gamma^2}{\delta} \sim \mathcal{O}(\zeta)$$

In other words, tuning precision parameter ζ, one can approximate the solution vector $\boldsymbol{\mathcal{Y}}_i^*$ with any arbitrary precision.

Stochastic Method

Next, we relax the deterministic nature of the local Newton method and consider its stochastic counterpart. Recall that the primal-dual relation is established by the following optimization problem:

$$\min_{\boldsymbol{\mathcal{Y}}_i} \hat{f}_i(\boldsymbol{\mathcal{Y}}_i) = \sum_{j=1}^{N} \ell_{ij}(\boldsymbol{\mathcal{Y}}_i) + \sum_{j=1}^{p}[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_j]_i[\boldsymbol{y}_j]_i$$

or equivalently:

$$\min_{\boldsymbol{\mathcal{Y}}_i} \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i) = \frac{1}{N}\sum_{j=1}^{N}\left[\ell_{ij}(\boldsymbol{\mathcal{Y}}_i) + \sum_{j=1}^{p}[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{\lambda}_j]_i[\boldsymbol{y}_j]_i\right] \tag{3.36}$$

Following the properties in (3.29), it is easy to see that function $\tilde{f}_i$ is twice differentiable with:

$$\frac{\Gamma}{N}\boldsymbol{I} \preceq \nabla^2\tilde{f}_i \preceq \frac{\Gamma}{N}\boldsymbol{I}$$
$$\left|\left|\nabla^2\tilde{f}_i(\boldsymbol{\mathcal{Y}}_i) - \nabla^2\tilde{f}_i(\tilde{\boldsymbol{\mathcal{Y}}}_i)\right|\right|_2 \leq \frac{\delta}{N}||\boldsymbol{\mathcal{Y}}_i - \tilde{\boldsymbol{\mathcal{Y}}}_i||_2, \quad \forall\ \boldsymbol{\mathcal{Y}}_i, \tilde{\boldsymbol{\mathcal{Y}}}_i \in \mathbb{R}^p$$

Previously, we suggested a deterministic approximated Newton method for the above problem. Notice that, in each iteration of this method, one has to calculate N gradients and Hessians. Hence, as N grows large, the application of such technique becomes problematic. To remedy this problem, stochastic Quasi-Newton methods proposed in Byrd et al. (2016) can be used. The algorithm operates in epochs by updating the solution vector $\boldsymbol{\mathcal{Y}}_i^{[t]}$ using Newton iteration. The crucial component of this method is an approximate computation of the Hessian inverse based on the BFGS formula given in Algorithm 19. This approximation is achieved by collecting a set of correction pairs $\{\boldsymbol{s}_j, \boldsymbol{z}_j\}$ throughout the most recent $\tilde{m}$ epochs. Such approach allows operating only with the sampled gradient and Hessians of function $\tilde{f}_i$. The full description of the SQN method for problem (3.36) is presented in Algorithm 18:

The convergence guarantees for the SQN method is based on spectral properties of the Hessian inverse approximation given by Algorithm 19. Next, we present a slight modification of Lemma 3.1 in Byrd et al. (2016) proving bounds for the spectrum of Hessian inverse approximation:

Lemma 3.2.14 *Let* $\widetilde{\boldsymbol{H}}_i^{[r]}$ *be the output of Algorithm 19, then*

$$\nu_1\boldsymbol{I} \preceq \widetilde{\boldsymbol{H}}_i^{[r]} \preceq \nu_2\boldsymbol{I}$$

where constants $\nu_1 = \frac{N}{\Gamma(\tilde{m}+p)}$ *and* $\nu_2 = N\left[\left(\frac{p}{\gamma}+\frac{1}{\Gamma}\right)\left(1+\frac{\Gamma}{\gamma}\right)^{\tilde{m}} - \frac{1}{\Gamma}\right]$.

Proof See Appendix.

The following theorem in Byrd et al. (2016) establishes the convergence properties of the SQN algorithm:

Algorithm 18 : **Stochastic Quasi Newton Method** Byrd et al. (2016)

Input: Initial value $\boldsymbol{\mathcal{Y}}_i^{[0]}$, sampling sizes b_g, b_H, number of iterations $T = SL$, the length of the epoch L, number of epochs $S = \lceil \frac{T}{L} \rceil$.
Output $\boldsymbol{\mathcal{Y}}_i^{[T]}$.
Set epoch count $r = -1$ and average vector $\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]} = \mathbf{0}$;
for $t = 1$ to T **do**
    Sample u.r. with replacement subset $\mathcal{S}_{g,t} \subset \{1, \ldots, N\}$ and:

$$\widetilde{\nabla} \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^{[t]}) = \frac{1}{|\mathcal{S}_{g,t}|} \sum_{j \in \mathcal{S}_{g,t}} \left[\nabla \ell_{ij}(\boldsymbol{\mathcal{Y}}_i^{[t]}) + \sum_{j=1}^{p} [\boldsymbol{L}\boldsymbol{\lambda}_j]_i \boldsymbol{e}_j \right]$$

    at point $\boldsymbol{\mathcal{Y}}_i^{[t]}$.
    Set $\overline{\boldsymbol{\mathcal{Y}}}_i^{[t]} = \overline{\boldsymbol{\mathcal{Y}}}_i^{[t]} + \frac{1}{L} \boldsymbol{\mathcal{Y}}_i^{[t]}$.
    if $t \leq 2L$ **then**
        Set $\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} - \beta_t \widetilde{\nabla} \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})$.
    else
        Compute inverse hessian approximation $\widetilde{\boldsymbol{H}}_i^{[r]}$ using Algorithm 19.
        Set $\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} - \beta_t \widetilde{\boldsymbol{H}}_i^{[r]} \widetilde{\nabla} \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})$.
    end if
    if $\mod \{t, L\} = 0$ **then**
        Set $r = r + 1$.
        if $r > 0$
            Sample u.r. with replacement subset $\mathcal{S}_H \subset \{1, \ldots, N\}$ and:

$$\widetilde{\nabla}^2 \tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) = \frac{1}{|\mathcal{S}_H|} \sum_{j \in S_h} \left[\nabla^2 \ell_{ij}(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) \right]$$

            at point $\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}$.
            Compute $\boldsymbol{s}^{[r]} = \overline{\boldsymbol{\mathcal{Y}}}_i^{[r]} - \overline{\boldsymbol{\mathcal{Y}}}_i^{[r-1]}$, $\boldsymbol{z}^{[r]} = \widetilde{\nabla}^2 \tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) \boldsymbol{s}^{[r]}$
        end if
    Set $\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]} = \mathbf{0}$.
    end if
end for

Algorithm 19 : **Hessian Inverse Approximation** Byrd et al. (2016)

1: **Input:**Epoch count r, memory parameter M, collection of correction pair $\{\boldsymbol{s}^{[j]}, \boldsymbol{z}^{[j]}\}$ for $j = r - \tilde{m} + 1, \ldots, r$ with $\tilde{m} = \min\{r, M\}$.

2: **Output** $\widetilde{\boldsymbol{H}}_i^{[r]}$.

3: Set $\widetilde{\boldsymbol{H}}_i^{[r]} = \frac{\boldsymbol{s}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}\boldsymbol{I}$

4: **for** $j = r - \tilde{m} + 1, \ldots, r$ **do**

5: Set $\rho_j = \frac{1}{\boldsymbol{z}^{[j]\mathsf{T}}\boldsymbol{s}^{[j]}}$ and

$$\widetilde{\boldsymbol{H}}_i^{[r]} = \left(\boldsymbol{I} - \rho_j \boldsymbol{s}^{[j]}\boldsymbol{z}^{[j]\mathsf{T}}\right)\widetilde{\boldsymbol{H}}_i^{[r]}\left(\boldsymbol{I} - \rho_j \boldsymbol{z}^{[j]}\boldsymbol{s}^{[j]\mathsf{T}}\right) + \rho_j \boldsymbol{s}^{[j]}\boldsymbol{s}^{[j]\mathsf{T}}$$

6: **end for**

Theorem 3.2.15 *Consider the iteration procedure given in Algorithm 18 with step size* $\beta_t = \frac{\gamma\nu_1 b_g}{(\nu_2\Gamma)^2}$ *where* b_g *is the size of gradient samplings. Then the SQN algorithm exhibits linear convergence rate*

$$\mathbb{E}_{\mathcal{S}_{g,t-1}}\left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^{[t]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^*)\right] \leq \left(1 - \tilde{\delta}\right)^t \left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[0]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^*)\right]$$

where $\mathcal{S}_{g,t-1}$ *is random sampling from* $\{1, \ldots, N\}$ *for computing the subsampled gradient at* $t-1$ *iteration,* $\boldsymbol{\mathcal{Y}}_i^{[0]}$ *and* $\boldsymbol{\mathcal{Y}}_i^*$ *are initial and optimal values for (3.36), and*

$$\tilde{\delta} = \left(\frac{\gamma\nu_1}{\Gamma\nu_2}\right)^2 \frac{b_g}{N}$$

Proof See Appendix.

To finalize, let us make two important remarks distinguishing the proposed analysis from the original one in Byrd et al. (2016):

- *Convergence rate:* We improve the convergence rate $\mathcal{O}\left(\frac{1}{k}\right)$ achieved in Byrd et al. (2016) to a linear rate.

- *Step-size:* A linear convergence rate is obtained by using a constant step size $\beta_t = \frac{\gamma\nu_1}{(\nu_2\Gamma)^2} b_g$ rather than diminishing step size $\beta_t \approx \frac{1}{t}$ given in the original paper. This makes the method more attractive from a practical point of view, because diminishing step size in practice quickly stagnates the iteration scheme $\boldsymbol{\mathcal{Y}}_i^{[t+1]} = \boldsymbol{\mathcal{Y}}_i^{[t]} - \beta_t \widetilde{\boldsymbol{H}}_i^{[r]} \widetilde{\nabla} \tilde{f}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})$ at some non-optimal value.

3.2.9. Experiments

We analyze our numerical performance in three different scenarios: against the centralized stochastic gradient descent (SGD) method using streaming model, against the decentralized SGD using SPARK model, and against other fully distributed approaches on different graph topologies.

Comparison with the centralized stochastic gradient descent in a streaming model

For the first scenario, we simulate a distributed network according to the Erdos-Renyi model with 50 nodes and 100 edges. The synthetic data for the linear regression model consists of $N = 10^5$ data points, each represented as $p = N^\alpha$ dimensional vector with $\alpha \in [0.25, 0, 5]$. An ϵ of $1/10$ was provided to the distributed solver for determining the approximate Newton direction. Step sizes were determined separately for each algorithm using a grid-search-like technique over $\{0.01, 0.1, 0.2, 0.3, 0.5, 0.6, 0.9, 1\}$ to ensure best operating conditions.
In this experiment, for each value of the dimensional parameter p we measure the time needed for both approaches to reach the stopping criteria. The latter is chosen as a threshold for the norm of the gradient of the primal objective and it is formulated according to the upper bound on the empirical risk error given by $\mathcal{O}(\frac{1}{N})$ (as was shown in Shalev-Shwartz et al. (2009)). The results are represented in Figure 17 and demonstrate that the distributed Newton method outperforms stochastic gradient descent (SGD) for the chosen values of α.

Interestingly, we noticed that for smaller values of the parameter α, controlling the dimensional complexity of the problem, the major effect on the performance is imposed by the condition number of the objective function. In other words, if the ratio $\frac{\Gamma}{\gamma}$ of parameters from Assumption 3.2.1 is high, then gradient descent methods are more favorable. We believe this behavior is caused by the local Hessian inversion procedure called the Distributed Newton Method in each iteration and implemented according to Approximated Newton Algorithm 16. As was shown in Section 3.2.8 the performance of this algorithm scales linearly with $\sqrt{\frac{\Gamma}{\gamma}}$. For low dimensional setups, this ratio plays a major role in the performance of the Distributed Newton Method.

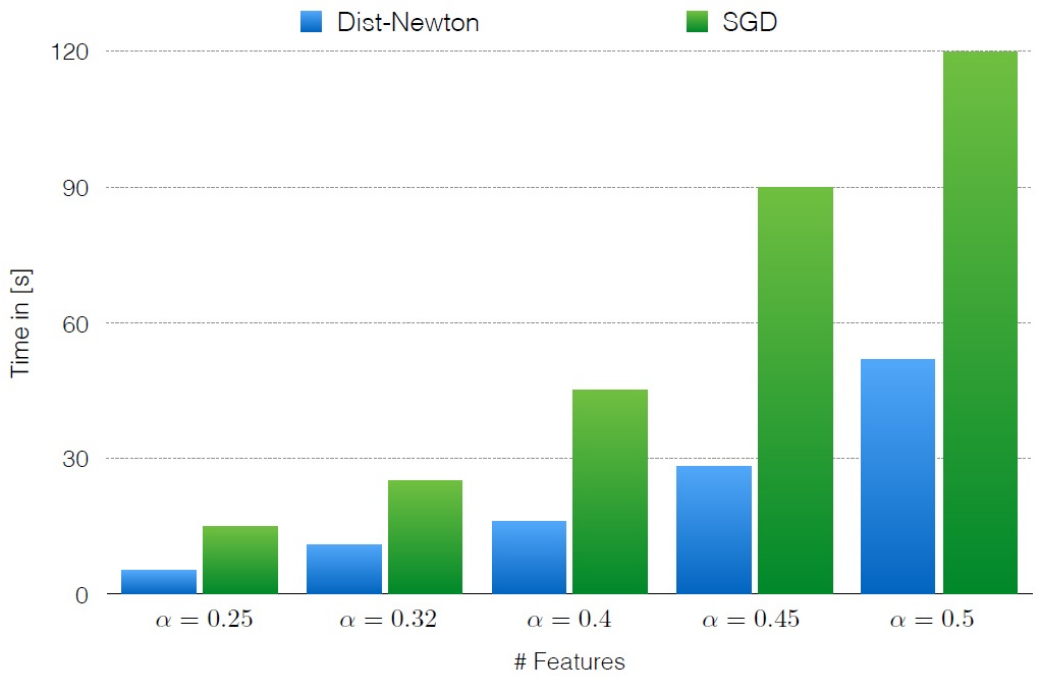


Figure 16: The effect of dimensionality

Comparison with the decentralized SGD on SPARK model

For this experiment, we implement both the Distributed Newton Method and the stochastic gradient descent (SGD) in a SPARK computational model. In particular, we split the data set among $n = 9$ computational clusters connected to a central master node.

For the stochastic gradient method, the central node collects the sampled gradients from peripheral clusters and then formulates the unbiased estimate $\hat{\nabla} f(\boldsymbol{x})$ for the total gradient over all finite sum. This estimate is used by the central cluster to perform gradient descent update:

$$\boldsymbol{x}^{[k+1]} = \boldsymbol{x}^{[k]} - \beta^{[k]} \hat{\nabla} f(\boldsymbol{x}^{[k]})$$
$$with \quad \sum_{k=1}^{\infty} \beta^{[k]} = \infty \;\; \& \;\; \sum_{k=1}^{\infty} \left(\beta^{[k]}\right)^2 < \infty$$

which is distributed then to all peripheral clusters. In this experiment, we consider the logistic regression model applied to synthetic data sets with $N = 10^6$ and number of features given by $p = N^{\alpha}$ with $\alpha \in [0.25, 0, 5]$. For each value of the dimensional parameter p, we measure the

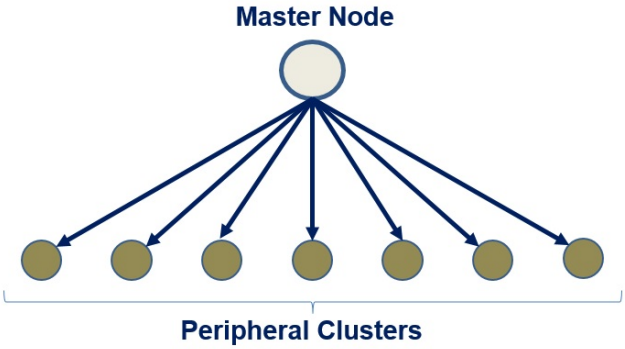


Figure 17: SPARK computational model

time needed for both approaches to reach the stopping criteria. As in the previous experiment, the stopping criteria is chosen as a threshold for the norm of the gradient of the primal objective given by the upper bound on the empirical risk error.

The central node connected with a collection of peripheral clusters forms a star graph topology $\mathbb{S}_n$, described as a special case in Section 2.4. As reported in Table 1, the time complexity of our fully distributed Chebyshev solver is given as $\mathcal{O}(n\sqrt{n})$, with n being the total number of clusters. The additional factor n caused by the aggregation and processing information from peripheral clusters in the central node. Next, we present an acceleration technique tailored specifically for unweighted Laplacians of a star graph $\mathbb{S}_n$ allowing us to solve a Laplacian system

$$\boldsymbol{L}_{\mathbb{S}_n}\boldsymbol{x} = \boldsymbol{b} \tag{3.37}$$

in $\mathcal{O}(n)$ time. This new technique is based on the spectrum of the unweighted Laplacian $\boldsymbol{L}_{\mathbb{S}_n}$ of a star graph given as:

$$Spectr(\boldsymbol{L}_{\mathbb{S}_n}) = \{0, \underbrace{1, 1 \ldots, 1}_{n-2}, n\}$$

Moreover, the eigenvector $\boldsymbol{u}_n$ corresponding to the largest eigenvalue allows a closed-form expression:

$$\boldsymbol{u}_n = \frac{1}{\sqrt{(n-1)n}} \begin{bmatrix} n-1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$$

Therefore, using spectral decomposition for $\boldsymbol{L}_{\mathbb{S}_n} = \sum_{j=2}^{n-1} \boldsymbol{u}_j \boldsymbol{u}_j^\mathsf{T} + n\boldsymbol{u}_n \boldsymbol{u}_n^\mathsf{T}$ and $\boldsymbol{L}_{\mathbb{S}_n}^\dagger = \sum_{j=2}^{n-1} \boldsymbol{u}_j \boldsymbol{u}_j^\mathsf{T} + \frac{1}{n}\boldsymbol{u}_n \boldsymbol{u}_n^\mathsf{T}$ we immediately arrive at the closed-form expression for the exact solution of system (3.37):

$$\boldsymbol{x}^* = \boldsymbol{L}_{\mathbb{S}_n}^\dagger \boldsymbol{b} = \boldsymbol{L}_{\mathbb{S}_n} \boldsymbol{b} + \frac{1 - n^2}{n} (\boldsymbol{u}_n^\mathsf{T} \boldsymbol{b}) \boldsymbol{u}_n \tag{3.38}$$

One can see that the computational complexity of this technique is restrained by the matrix vector $\boldsymbol{L}_{\mathbb{S}_n} \boldsymbol{b}$ and the inner $\boldsymbol{u}_n^\mathsf{T} \boldsymbol{b}$ products and both these operations can be performed in $\mathcal{O}(n)$ time.[9]
We apply this accelerated technique in the distributed computation of the Newton direction for SPARK model experiments. The results are represented in Figure 18 and demonstrate that the Distributed Newton Method outperforms stochastic gradient descent for the chosen values of α.

α	SGD	DNewton
0.25	137.0	27.2
0.32	400.0	61.6
0.4	560.0	88.0
0.45	1730.0	109.2
0.5	3190.2	175.0

Figure 18: The effect of dimensionality for the SPARK model

Similarly to the previous experiment, for smaller values of the parameter α, controlling the dimensional complexity of the problem the major effect on the performance is imposed by the condition number of the objective function. In other words, if the ratio $\frac{\Gamma}{\gamma}$ of parameters from Assumption 3.2.1 is high, then SGD methods are more favorable.

Comparison with Fully Distributed Approaches

Finally, we evaluate our method against five other approaches: 1) distributed Newton ADD, an adaptation of ADD (Zargham et al. (2013)) that we introduce to compute the Newton direction of general consensus, 2) distributed ADMM (Wei and Ozdaglar (2012)), 3) distributed averaging (Olshevsky (2014)), an algorithm solving general consensus using local averaging, 4) Network Newton 1 and 2 (Mokhtari et al. (2015)), and 5) distributed gradients (Nedic and Ozdaglar (2009)). We

[9]Notice, this approach is only suitable for unweighted Laplacian $\boldsymbol{L}_{\mathbb{S}_n}$ of a star graph, because of the specific structure of the spectrum of $\boldsymbol{L}_{\mathbb{S}_n}$ as well as the known expression for the largest eigenvector $\boldsymbol{u}_n$.

are chiefly interested in the convergence speeds of both the objective value and the consensus error. The objective value plots demonstrate whether our method is capable of reducing the value of the objective/cost function, while consensus plots allow us to comprehend the violation of the constraints. Furthermore, comparison against ADMM positions us with respect to the state-of-the-art, while comparisons against ADD and Network Newton sheds light on the accuracy of our Newton direction approximation.

To simulate a real-world distributed environment, we used the Matlab parallel pool running on an 8-core server. After generating the processors' graph structure with random edge assignment (see below for specifics on the node-edge configuration), we split nodes equally across the 8 cores. Hence, each processor was assigned a collection of nodes for performing computations.

Benchmark Data Sets We performed three sets of experiments on standard machine learning problems: 1) linear regression, 2) logistic regression, and 3) reinforcement learning. We transformed centralized problems to fit within the distributed consensus framework. This can be easily achieved by factoring the summation running over all the available training examples to partial summations across multiple processors while introducing consensus. We considered both synthetic as well as real-world data sets:

Synthetic Regression Task: We created a data set for regression with 108 data points each being an 80-dimensional vector. The task parameter vector was generated as a linear combination of these features. The training data set $\boldsymbol{D}$ was generated from a standard normal distribution in 80 dimensions. The training labels were given as $\boldsymbol{y} = \boldsymbol{D}\boldsymbol{x} + \boldsymbol{\xi}$, where each element in $\boldsymbol{\xi}$ was an independent univariate Gaussian noise.

MNIST Data The MNIST data set is a large database of handwritten digits which has been used as a benchmark for classification algorithms Lecun and Cortes. The goal is to classify among 10 different digits amounting from 0 to 9. After reading each image, we perform dimensionality reduction to reduce the number of features of each instance image to 150 features using principle component analysis and follow a one-versus-all classification scheme.

London Schools Data The London Schools data set consists of examination scores from 15,362 students in 139 schools. This is a benchmark regression task with a goal of predicting examination scores of each student. We use the same feature encoding used in Kumar and Daume (2012), where four school-specific categorical variables along with three student-specific categorical variables are

encoded as a collection of binary features. In addition, we use the examination year and a bias term as additional features, giving each data instance 27 features.

Reinforcement Learning We considered the policy search framework to control a double cart-pole system (DCP). As detailed in Bou-Ammar et al. (2015), the DCP adds a second inverted pendulum to the standard cart-pole system, with six parameters and six state features. The goal is to balance both poles upright. We generated 20,000 rollouts each with a length of 150 time steps.

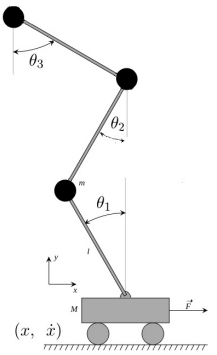
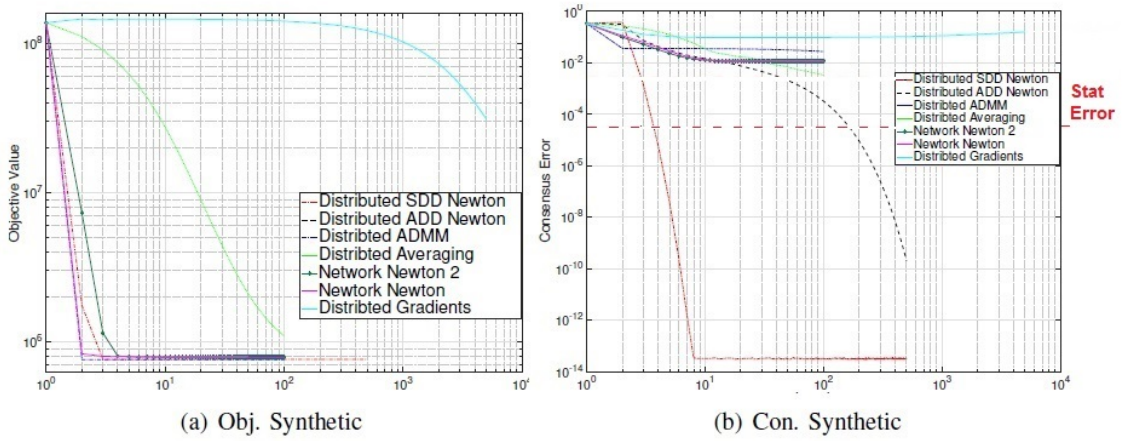


Figure 19: Double Cart Pole System

Benchmark Results

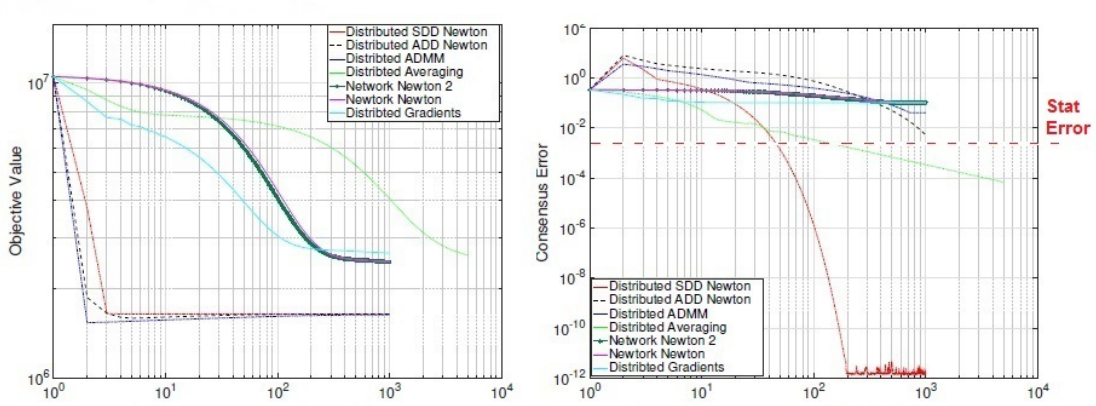
1. *Linear Regression Results:*In this section, we report regression results on the synthetic and London school data-sets. **Synthetic Data:** We randomly distributed the regression objective over a network of 100 nodes and 250 edges. The edges were chosen uniformly at random. An ϵ of 1/10 was provided to the distributed solvers for determining the approximate Newton direction. Step sizes were determined separately for each algorithm using a grid-search-like-technique over $\{0.01, 0.1, 0.2, 0.3, 0.5, 0.6, 0.9, 1\}$ to ensure best operating conditions. We used the local objective and the consensus error as performance metrics. Results shown in Figure 20 demonstrate that our method (titled Distributed SDD Newton) outperforms all other techniques in both objective value and consensus error. Namely, distributed SDD Newton converges to the optimal value in about 40 iterations compared to about 200 for the second-best performing algorithm. It is also interesting to recognize that the worst performing algorithms were distributed gradients and Network Newton 1 and 2 from Mokhtari et al. (2015). Furthermore, it is worth noting that although some algorithms appear similar to ours in terms of objective values, the solution derived by SDD-Newton is more accurate in terms of feasibility,

a criterion essential for achieving consensus.



(a) Obj. Synthetic (b) Con. Synthetic

Figure 20: Experimental Results on Linear Regression with a synthetic data set.



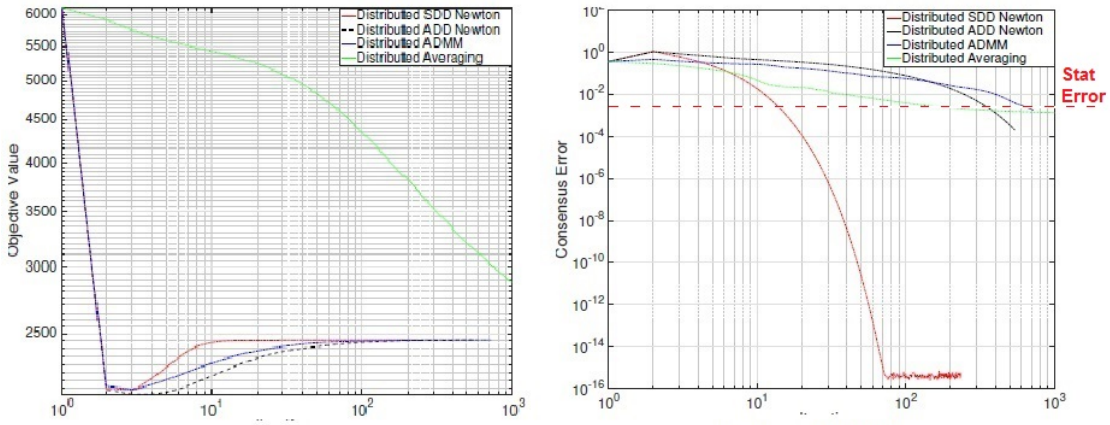
(a) Objective London Schools (b) Consensus London Schools

Figure 21: Experimental Results on Linear Regression with a real data set.

London Schools Data: We repeated the above experiments on the London Schools data-set. The same parametric setting for the SDD solver and for the step sizes was used. The graph topology, however, was set to 50 nodes and 150 edges generated uniformly at random. Results depicted in Figure 21 confirm previous conclusions showing the SDDNewton outperforms other techniques by a significant margin (in the order of 1000 iterations)

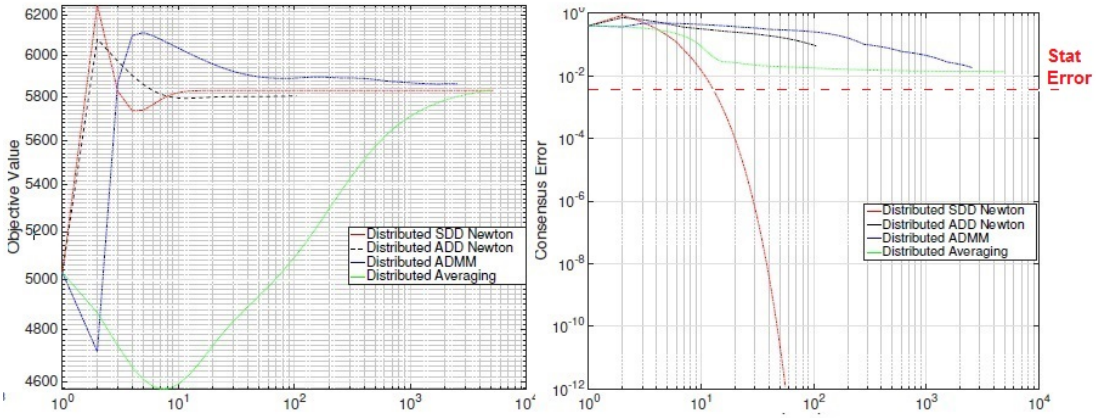
2. *Logistic Regression Results* We chose the most successful algorithms from previous experiments to perform image classification. We considered both smooth ($L2$ norms) and non-smooth ($L1$

norms) regularization forms on latent parameters. The processor graph was set to 10 nodes and 20 edges generated uniformly at random. Results depicted in Figures 22 and 23 demonstrate that our algorithm is again capable of outperforming state-of-the-art methods.



(a) Obj. MNIST L_2 (b) Con. MNIST L_2

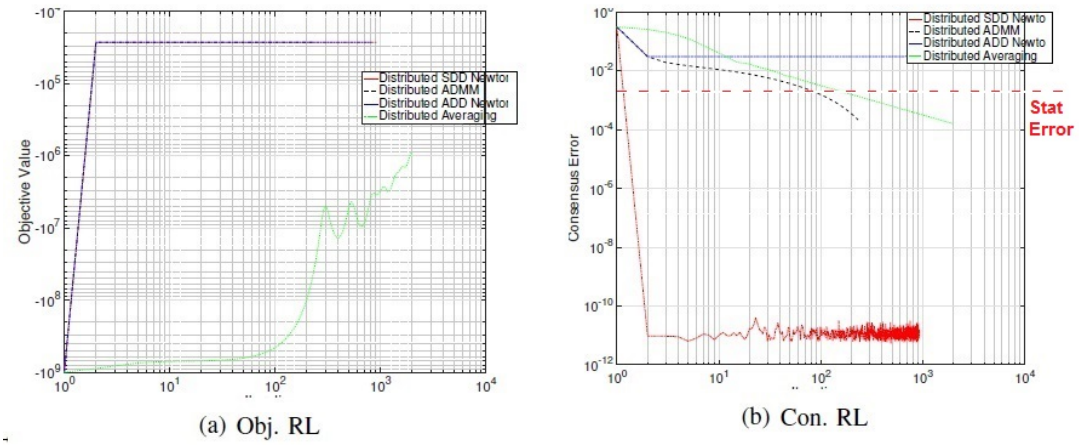
Figure 22: Experimental Results on Logistic Regression with L_2 regularization.



(a) Obj. MNIST L_1 (b) Con. MNIST L_1

Figure 23: Experimental Results on Logistic Regression with L_1 regularization.

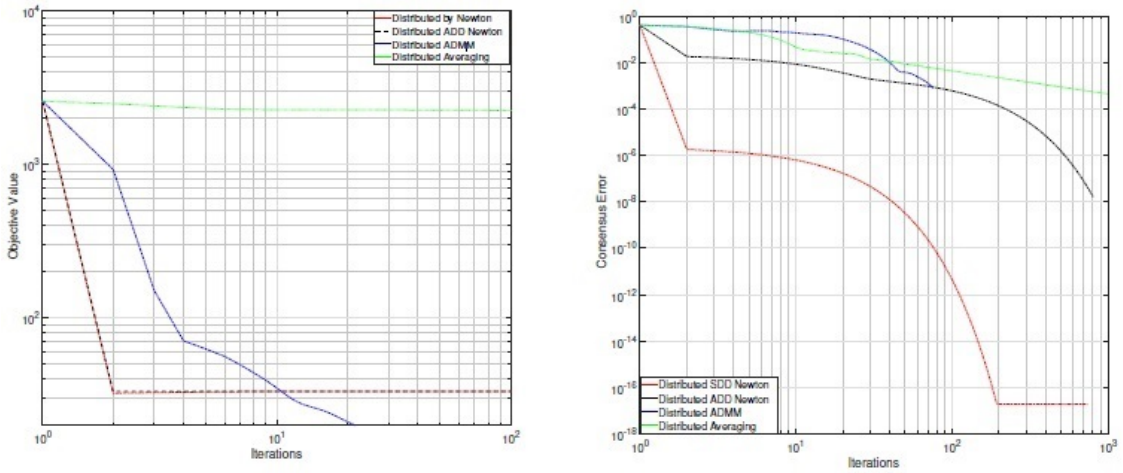
3. *Reinforcement Learning Results:* Finally, the above were repeated for controlling the DCP task. We split 20,000 trajectories across a graph of 120 processors and 250 edges. Results shown in Figure 24 demonstrate that our method again outperforms state-of-the-art where it is capable of achieving low consensus error after a couple of iterations.



(a) Obj. RL (b) Con. RL

Figure 24: Experimental Results on Reinforcement Learning.

fMRI Experiment Having shown that our approach outperforms others on relatively dense benchmark data-sets, we are now interested in the performance on sparse data sets where the number of features is much larger than the number of inputs. To do so, we used the functional Magnetic Resonance Imaging (fMRI) data set from Singh et al. (2007). The goal in these experiments is to classify the cognitive state (i.e., whether looking at a picture or a sentence) of a subject based on fMRI data. Six subjects were considered in total. Each had 40 trials that lasted for 27 seconds attaining in total 54 images per subject. After preprocessing as described in Singh et al. (2007), we acquired a sparse data-set with 240 input data points, each having 43,720 features. We then performed logistic regression with an L1 regularization and reported objective values and consensus errors. Figure 25 demonstrates the objective value and consensus errors on the fMRI data-set. First, it is clear that our approach outperforms others on both criteria. It is worth noting that the second-best performing algorithm to ours is Distributed ADDNewton, an alternative approach we propose in this paper for computing the Newton direction. Distributed ADMM and Distributed Averaging perform the worst on such a sparse problem. Second, Figure 25(b) clearly manifests the drawback of ADMM which requires substantial numbers of iterations for converging to the optimal feasible point. Due to the size of the feature set (i.e., 43,720), even small deviations from the optimal model can lead to significant errors in the value of the objective function. This motivates the need for the accurate solutions as acquired by our method.



(a) Con. fMRI (b) Obj. fMRI

Figure 25: Experimental Results on fMRI images.

Communication Overhead & Running Times Similar to the Network Flow Problem, it could seem that the speed and accuracy of the proposed approach come at the expense of a higher communication overhead since our approach utilizes distributed solvers while other approaches allow only a few message exchanges. To test this, we conducted a final experiment measuring local communication exchange with respect to accuracy requirements. For that, we chose the London Schools data set as all algorithms performed relatively well. Results reported in Figure 26(a) demonstrate that this increase is negligible compared to other methods. Clearly, as accuracy improves so does the communication overhead of all other algorithms. Distributed solvers proposed in Algorithms 5 and 6 have a growth rate proportional to the condition number and square root of the condition number of the graph respectively, which is much slower compared to the exponential growth observed with other techniques. Finally, Figure 26(b) reports running times till convergence on the same data set.

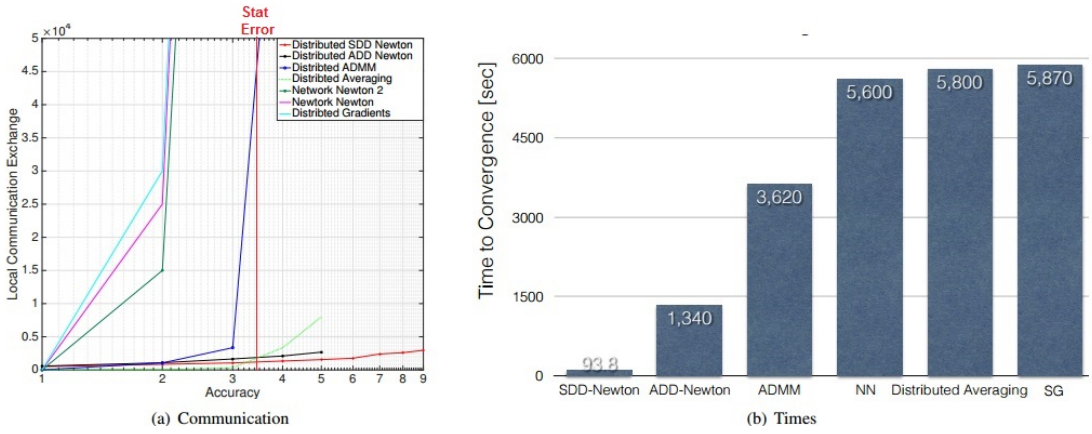


Figure 26: Experimental results: communication overhead, CPU running times

CHAPTER 4 : CONCLUSION

In this chapter, we emphasize the contributions of this thesis and present future directions that could be taken to extend these results.

4.1. Thesis Summary

The purpose of this thesis was to develop new algorithmic tools that can be applied in a distributed setting and improve the performance of decentralized optimization methods. The developed algorithms target the problem of solving symmetric diagonal dominant linear systems in a fully distributed and mixed way.

The first fully distributed algorithm in Section 2.2 was based on the recursive preconditioning of the original SDD matrix with a collection of matrices called an approximated inverse chain. This method can be considered as a decentralized version of the parallel SDD solver proposed by Peng and Spielman (2013). The distributed adaptation of their technique allows us to kill two birds with one stone: to construct a short approximated chain, reducing the overall computational burden and allowing nodes to operate only with their local information without any need for a global coordinator. As a result, we arrive at a fully distributed SDD solver with time complexity characterized by the condition number of matrix of the system.

The linear dependence on the condition number restricted the application range of the first solver to graphs with "small" diameters. To remedy this issue and cover arbitrary graph topologies, we proposed the second fully distributed SDD solver. The key idea for improving the performance of the previous algorithm was a better approximation of the SDD inverse by carefully scaled Chebyshev polynomials. Due to their extremal properties and recursive relation connecting them we were able to design a fully distributed SDD solver with performance characterized by a square root of the condition number. We further optimized this algorithm by designing a distributed binary search procedure to adjust the computation for a structure of the right-hand side vector of a system.

It was observed that fully decentralized models have unavoidable drawbacks in comparison to their centralized counterparts. In particular, the message communication between the clusters charges additional costs on the behavior of distributed algorithms. Moreover, we noticed that this message exchange directly related to the topology of the underlying graph of clusters. In Section 2.3 we carefully studied this relation and introduced a new mixed algorithm based on distributed construction of a spectral sparsifier. The size of this sub-graph allowed us to collect its full topology in one single node previously chosen by applying leader election protocols. Finally, we applied the centralized SDD solver to a sparsified SDD system with carefully chosen precision parameters. As a result, our new mixed SDD solver not only improved the communication complexity of fully distributed techniques but surprisingly surpassed central SDD solvers in terms of time complexity.

In Sections 3.1 and 3.2 based on the proposed SDD solvers we designed an exact distributed Newton method for the Network Flow problem and Empirical Risk Minimization. The particular choice of these applications was motivated by their importance for practical interest. On the theoretical side, we showed that the introduced algorithms exhibited quadratic convergence rates and outperformed all existing methods. On the practical side, we verified this behavior on a wide range of numerical simulations including various graph topologies, a different number of features and different forms of regularization terms.

4.2. Future Work

4.2.1. Non-Convex Case

Due to the recent success of Deep Learning techniques in speech recognition, computer vision, and artificial intelligence, it is interesting to extend our work for these methods and to address the immense computational scale of deep neural networks. Despite the non-convex nature of the activation function, we hope to develop second order distributed techniques based on higher order Taylor approximation. In particular, in a recent work (Agarwal et al. (2017)), the authors exploit the curvature information by considering the cubic approximation of a non-convex function. The descent direction then is computed as a solution of a symmetric system of linear equations abiding by the diagonal dominant property. Interestingly, in contrast with a convex case, this cubic regularized Newton method attains only linear convergence rate to a local minimum of the activation

function. From the practical point of view, training of multi-layered neural networks implies serious computational challenges for researchers. Therefore, adopting distributed models promises to reduce the complexity of the problem.

4.2.2. Experiments with SPARK

On the implementation side, the interesting direction is to consider large cluster configurations with hundreds or even thousands of computational units. As was shown in Section 2.4, the size of the network does not necessarily decrease its computational power (for instance, with sparse Ramanujan graphs). Collecting these units into "efficient" network configurations could potentially lead to significant acceleration of the proposed techniques.

APPENDIX

A.1. Proof of Lemma 2.2.2

Proof For a better exposure each statement is proved separately:

1. The proof of the first part will be given as a collection of claims:

   Claim: Let κ be the condition number of $\boldsymbol{L}_{\mathbb{G}} = \boldsymbol{D}_0 - \boldsymbol{A}_0$, and $\{\lambda_i\}_{i=1}^n$ denote the eigenvalues of $\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$. Then, $|\lambda_i| \leq 1 - \frac{1}{\kappa}$, for all $i = 1, \ldots, n$

   Proof See Proposition 5.3 in Peng and Spielman (2013).

   Notice that if λ_i represented an eigenvalue of $\boldsymbol{D}_0^{-1}\boldsymbol{A}_0$, then λ_i^r is an eigenvalue of $\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^r$ for all $r \in \mathbb{N}$. Therefore, for the spectral radius of matrix $(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0)^{2^d}$ we have: $\rho\left(\left(\boldsymbol{D}_0^{-1}\boldsymbol{A}_0\right)^{2^d}\right) \leq \left(1 - \frac{1}{\kappa}\right)^{2^d}$.

   Claim: Let $\boldsymbol{M}$ be an SDD matrix and consider the splitting $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$, with $\boldsymbol{D}$ being non negative diagonal and $\boldsymbol{A}$ being symmetric non negative. Further, assume that the eigenvalues of $\boldsymbol{D}^{-1}\boldsymbol{A}$ lie between $-\alpha$ and β. Then:

   $$(1 - \beta)\boldsymbol{D} \preceq \boldsymbol{D} - \boldsymbol{A} \preceq (1 + \alpha)\boldsymbol{D}$$

   Proof See Proposition 5.4 in Peng and Spielman (2013).

   Combining the above results, gives

   $$\left[1 - \left(1 - \frac{1}{\kappa}\right)^{2^d}\right] \boldsymbol{D}_d \preceq \boldsymbol{D}_d - \boldsymbol{A}_d \preceq \left[1 + \left(1 - \frac{1}{\kappa}\right)^{2^d}\right] \boldsymbol{D}_d.$$

   Hence, to guarantee that $\boldsymbol{D}_d \approx_{\epsilon_d} \boldsymbol{D}_d - \boldsymbol{A}_d$, the following system must be satisfied:

   $$e^{-\epsilon_d} \leq 1 - \left(1 - \frac{1}{\kappa}\right)^{2^d}, \quad \text{and} \quad e^{\epsilon_d} \geq 1 + \left(1 - \frac{1}{\kappa}\right)^{2^d}.$$

Introducing γ for $\left(1-\frac{1}{\kappa}\right)^{2^d}$, we arrive at:

$$\epsilon_d \geq \ln\left(\frac{1}{1-\gamma}\right), \quad \text{and} \quad \epsilon_d \geq \ln(1+\gamma).$$

Hence, $\epsilon_d \geq \max\left\{\ln\left(\frac{1}{1-\gamma}\right), \ln(1+\gamma)\right\} = \ln\left(\frac{1}{1-\gamma}\right)$. Now, notice that if $d = \lceil \log c\kappa \rceil$ then, $\gamma = \left(1-\frac{1}{\kappa}\right)^{2^d} = \left(1-\frac{1}{\kappa}\right)^{c\kappa} \leq \frac{1}{e^c}$. Hence, $\ln\left(\frac{1}{1-\gamma}\right) \leq \ln\left(\frac{e^c}{e^c-1}\right)$. This gives $c = \lceil 2\ln\left(\frac{\sqrt[3]{2}}{\sqrt[3]{2}-1}\right)\rceil$, implying $\epsilon_d = \ln\left(\frac{e^c}{e^c-1}\right) < \frac{1}{3}\ln 2$.

2. In Algorithms 3 and 4, each node i computes $[\boldsymbol{b}_1]_i, [\boldsymbol{b}_2]_i, \ldots, [\boldsymbol{b}_d]_i$ and $[\boldsymbol{x}_d]_i, [\boldsymbol{x}_{d-1}]_i, \ldots, [\boldsymbol{x}_0]_i$, respectively. These are determined using the inverse approximated chain as follows

$$\begin{aligned} \boldsymbol{b}_k &= (\boldsymbol{I} + (\boldsymbol{A}_{k-1}\boldsymbol{D}_{k-1}^{-1})\boldsymbol{b}_{k-1} = \boldsymbol{b}_{k-1} + (\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^{2^{k-1}}\boldsymbol{b}_{k-1}. \\ \boldsymbol{x}_k &= \frac{1}{2}[\boldsymbol{D}_k^{-1}\boldsymbol{b}_k + (\boldsymbol{I} + \boldsymbol{D}_k^{-1}\boldsymbol{A}_k)\boldsymbol{x}_{k+1}] = \frac{1}{2}[\boldsymbol{D}_0^{-1}\boldsymbol{b}_k + \boldsymbol{x}_{k+1} + (\boldsymbol{D}_0^{-1}\boldsymbol{A}_0)^{2^k}\boldsymbol{x}_{k+1}]. \end{aligned} \tag{A.1}$$

Hence, Lemma 4.3 from Peng and Spielman (2013) gives $\boldsymbol{Z}' \approx_{\epsilon_d} \boldsymbol{L}_{\mathbb{G}}^{-1}$.

3. Considering the computation of $[\boldsymbol{b}_1]_k, \ldots, [\boldsymbol{b}_d]_k$ in (A.9) we have:

$$\begin{aligned} [\boldsymbol{b}_k]_i &= [\boldsymbol{b}_{k-1}]_i + [(\boldsymbol{A}_0\boldsymbol{D}_0^{-1})^{2^{k-1}}\boldsymbol{b}_{k-1}]_i = [\boldsymbol{b}_{k-1}]_i + [\underbrace{\boldsymbol{C}_0 \ldots \boldsymbol{C}_0}_{l}\boldsymbol{b}_{k-1}]_i \\ &= [\boldsymbol{b}_{k-1}]_i + [\underbrace{\boldsymbol{C}_0 \ldots \boldsymbol{C}_0}_{l-1}\boldsymbol{u}_1^{(k-1)}]_i = [\boldsymbol{b}_{k-1}]_i + \left[\boldsymbol{u}_l^{(k-1)}\right]_i. \end{aligned}$$

with $\boldsymbol{C}_0 = \boldsymbol{A}_0\boldsymbol{D}_0^{-1}$, $l = 2^{k-1}$, and $\boldsymbol{u}_{j+1}^{(k-1)} = \boldsymbol{C}_0\boldsymbol{u}_j^{(k-1)}$ for $j = 1, \ldots, l-1$. Due to the sparsity of $\boldsymbol{C}_0$, node i computes $[\boldsymbol{u}_{j+1}^{(k-1)}]_i$ based on the components of $\boldsymbol{u}_j^{(k-1)}$ attained from its neighbors. For each k, the computing $[\boldsymbol{b}_k]_i$ requires $\mathcal{O}\left(d_{max}2^{k-1}\right)$ time steps, where d_{max} is a maximal degree in $\mathcal{G}$. Therefore, the overall computation of the values $[\boldsymbol{b}_1]_i, [\boldsymbol{b}_2]_i, \ldots, [\boldsymbol{b}_d]_i$ is $\mathcal{O}\left(d_{max}2^d\right)$. Similar analysis can be applied to determine the computational complexity of $[\boldsymbol{x}_d]_i, [\boldsymbol{x}_{d-1}]_i, \ldots, [\boldsymbol{x}_1]_i$, in Algorithm 4. We arrive that the total time complexity of Algorithms 3 and 4 is $\mathcal{O}\left(d_{max}2^d\right)$.

A.2. Proof of Lemma 2.2.3

Proof Note that the iterations of Algorithm 5 correspond to a distributed version of the preconditioned Richardson iteration scheme

$$\boldsymbol{y}_t = [\boldsymbol{I} - \boldsymbol{Z}'\boldsymbol{L}_{\mathbb{G}}]\boldsymbol{y}_{t-1} + \boldsymbol{Z}'\boldsymbol{b}_0.$$

with $\boldsymbol{y}_0 = \boldsymbol{0}$ and $\boldsymbol{Z}'$ being the operator defined by Algorithms 4 and 5. From Lemma 3.3 it is clear that $\boldsymbol{Z}' \approx_{\epsilon_d} \boldsymbol{L}_{\mathbb{G}}^{-1}$. Applying Lemma 4.4 from Peng and Spielman (2013) , provides that Algorithm 5 requires $\mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$ iterations to return the i^{th} component of the $\epsilon-$close approximation to $\boldsymbol{x}^\star$. Finally, since Algorithm 5 uses Algorithms 3 and 4 as subroutines, it follows that for each node i only communication between neighbors is allowed. Consequently, the time complexity of Algorithm 5 is given by $\mathcal{O}\left(d_{max}2^d \log\left(\frac{1}{\epsilon}\right)\right)$.

A.3. Proof of Theorem 2.3.2

Proof The proof can be found in Koutis (2014).

A.4. Proof of Lemma 3.1.2

Proof We study each statement separately:

1. Instead of Network Flop Problem (3.1) we consider more general optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{x}} f(x) \\ &s.t.\ \ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{A} \in \mathbb{R}^{n\times p},\ \ \boldsymbol{b} \in \mathbb{R}^n \end{aligned} \tag{A.2}$$

   where $f(x)$ twice differentiable strongly convex function, and unknown variable $\boldsymbol{x} \in \mathbb{R}^p$. One can see, that problem (3.1) is a special case of (A.2) with $\boldsymbol{A}$ being an incidence matrix of graph $\mathbb{G}$. Let $q(\boldsymbol{\lambda})$ be the corresponding dual for (A.2), with dual variable $\boldsymbol{\lambda} \in \mathbb{R}^n$. We will show

that:

$$\nabla^2 q(\boldsymbol{\lambda}) = -\boldsymbol{A}[\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\boldsymbol{A}^\mathsf{T} \tag{A.3}$$
$$\nabla q(\boldsymbol{\lambda}) = \boldsymbol{A}\boldsymbol{x}(\boldsymbol{\lambda}) - \boldsymbol{b}$$

where $\boldsymbol{x}(\boldsymbol{\lambda}) = argmin_{\boldsymbol{x}} f(\boldsymbol{x}) + \boldsymbol{\lambda}^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ minimizes the Lagrangian of problem (A.2). Let us denote $\boldsymbol{x}(\boldsymbol{\lambda}) = \boldsymbol{x}^+$

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2p} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{np} \end{bmatrix}, \qquad \boldsymbol{x}^+ = \begin{bmatrix} x_1^+(\boldsymbol{\lambda}) \\ x_2^+(\boldsymbol{\lambda}) \\ \vdots \\ x_p^+(\boldsymbol{\lambda}) \end{bmatrix}, \nabla f(\boldsymbol{x}^+) = \begin{bmatrix} z_1(\boldsymbol{x}^+) \\ z_2(\boldsymbol{x}^+) \\ \vdots \\ z_p(\boldsymbol{x}^+) \end{bmatrix} \tag{A.4}$$

The optimal primal variable $\boldsymbol{x}(\boldsymbol{\lambda})$ satisfies:

$$\nabla f(\boldsymbol{x}(\boldsymbol{\lambda})) + \boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda} = \boldsymbol{0}. \tag{A.5}$$

Using Fenchel's conjugate, the dual function can be written as:

$$q(\boldsymbol{\lambda}) = -\boldsymbol{b}^\mathsf{T}\boldsymbol{\lambda} - f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}) \tag{A.6}$$

Therefore,

$$\nabla q(\boldsymbol{\lambda}) = -\boldsymbol{b} - \nabla f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}) \tag{A.7}$$

Denoting $\boldsymbol{u} = -\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}$, then the k^{th} component of vector $\nabla f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda})$ can be written as:

$$[\nabla f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda})]_k = \sum_{j=1}^{p} \frac{\partial f^*}{\partial u_j}\frac{\partial u_j}{\partial \lambda_k} = -\begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{kp} \end{bmatrix} \times \begin{bmatrix} \frac{\partial f^*}{\partial u_1} \\ \frac{\partial f^*}{\partial u_2} \\ \vdots \\ \frac{\partial f^*}{\partial u_p} \end{bmatrix}_{-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}}$$

Applying result (A.5) and the relation between the gradients of function and its Fenchel's

conjugate:

$$\nabla f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}) = -\boldsymbol{A}\nabla_{\boldsymbol{u}} f^*(\boldsymbol{u})|_{-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}} = -\boldsymbol{A}\nabla_{\boldsymbol{u}} f^*(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}) = \tag{A.8}$$
$$-\boldsymbol{A}\nabla_{\boldsymbol{u}} f^*(\nabla f(\boldsymbol{x}^+)) = -\boldsymbol{A}\boldsymbol{x}(\boldsymbol{\lambda})$$

Therefore, the result (A.7) gives:

$$\nabla q(\boldsymbol{\lambda}) = -\boldsymbol{b} + \boldsymbol{A}\boldsymbol{x}(\boldsymbol{\lambda}) \tag{A.9}$$

which establishes the claim for the dual gradient.

Taking the gradient in (A.9) gives:

$$\nabla^2 q(\boldsymbol{\lambda}) = \boldsymbol{A} \underbrace{\begin{bmatrix} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_n} \\ \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_n} \end{bmatrix}}_{\boldsymbol{F}(\boldsymbol{x}^+)} \tag{A.10}$$

Hence, we target matrix $\boldsymbol{F}(\boldsymbol{x}^+)$ to obtain the form of dual Hessian. Equation (A.5) gives:

$$\nabla f(\boldsymbol{x}^+) = -\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}$$

On the next step, we take partial derivative $\frac{\partial}{\partial \lambda_j}$ for both sides of the above equation for

$j = 1, \ldots, n$. For simplicity, consider $\frac{\partial}{\partial \lambda_1}$:

$$\frac{\partial}{\partial \lambda_1} \nabla f(\boldsymbol{x}^+) = \begin{bmatrix} \frac{\partial}{\partial \lambda_1} z_1(\boldsymbol{x}^+) \\ \frac{\partial}{\partial \lambda_1} z_2(\boldsymbol{x}^+) \\ \vdots \\ \frac{\partial}{\partial \lambda_1} z_p(\boldsymbol{x}^+) \end{bmatrix} =$$

$$\begin{bmatrix} \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \cdots + \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \cdots + \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \vdots \\ \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} + \cdots + \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} \end{bmatrix} =$$

$$\underbrace{\begin{bmatrix} \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} & \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} & \cdots & \frac{\partial z_1(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \\ \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} & \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} & \cdots & \frac{\partial z_2(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \\ \vdots & & \ddots & \vdots \\ \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_1^+(\boldsymbol{\lambda})} & \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_2^+(\boldsymbol{\lambda})} & \cdots & \frac{\partial z_p(\boldsymbol{x}^+)}{\partial x_p^+(\boldsymbol{\lambda})} \end{bmatrix}}_{\nabla^2 f(\boldsymbol{x}^+)} \begin{bmatrix} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \vdots \\ \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} \end{bmatrix} = \nabla^2 f(\boldsymbol{x}^+) \begin{bmatrix} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} \\ \vdots \\ \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} \end{bmatrix} =$$

$$\frac{\partial}{\partial \lambda_1}(-\boldsymbol{A}^\mathsf{T}\boldsymbol{\lambda}) = -\begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1p} \end{bmatrix}$$

Repeating this for $\frac{\partial}{\partial \lambda_2}, \ldots, \frac{\partial}{\partial \lambda_p}$ gives:

$$\nabla^2 f(\boldsymbol{x}^+) \underbrace{\begin{bmatrix} \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_1^+(\boldsymbol{\lambda})}{\partial \lambda_n} \\ \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_2^+(\boldsymbol{\lambda})}{\partial \lambda_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_1} & \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_2} & \cdots & \frac{\partial x_p^+(\boldsymbol{\lambda})}{\partial \lambda_n} \end{bmatrix}}_{\boldsymbol{F}(\boldsymbol{x}^+)} = -\underbrace{\begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & & \ddots & \vdots \\ a_{1p} & a_{2p} & \cdots & a_{np} \end{bmatrix}}_{\boldsymbol{A}^\mathsf{T}}$$

Therefore, for matrix $\boldsymbol{F}(\boldsymbol{x}^+) = -[\nabla^2 f(\boldsymbol{x}^+)]^{-1}\boldsymbol{A}^\mathsf{T}$ and combining this result with (A.10) gives:

$$\nabla^2 q(\boldsymbol{\lambda}) = -\boldsymbol{A}[\nabla^2 f(\boldsymbol{x}^+)]^{-1}\boldsymbol{A}^\mathsf{T}$$

2. Recall that:

$$||\boldsymbol{A}||_2 = \sup_{\boldsymbol{v}:\boldsymbol{v}\neq\boldsymbol{0}} \frac{||\boldsymbol{A}\boldsymbol{v}||_2}{||\boldsymbol{v}||_2}$$

Fix arbitrary $\boldsymbol{v} \neq \boldsymbol{0}$, then using the first claim:

$$||[\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})]\boldsymbol{v}||_2^2 =$$
$$\boldsymbol{v}^\mathsf{T}\boldsymbol{A}\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right)\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right)\boldsymbol{A}^\mathsf{T}\boldsymbol{v} \leq$$
$$\mu_n(\boldsymbol{L}_\mathbb{G})\boldsymbol{v}^\mathsf{T}\boldsymbol{A}\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right)^2\boldsymbol{A}^\mathsf{T}\boldsymbol{v} \leq$$
$$\mu_n^2(\boldsymbol{L}_\mathbb{G})\mu_n^2\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right)||\boldsymbol{v}||_2^2$$

Hence,

$$||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq \mu_n(\boldsymbol{L}_\mathbb{G})\mu_n\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right) \tag{A.11}$$

Due to Assumption 3.1.1:

$$\mu_n\left([\nabla^2 f(\boldsymbol{x}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{x}(\boldsymbol{\lambda}))]^{-1}\right) \leq \max_{e\in\mathbb{E}}\left|\frac{1}{\ddot{\Phi}_e(x^{(e)}(\bar{\boldsymbol{\lambda}}))} - \frac{1}{\ddot{\Phi}_e(x^{(e)}(\boldsymbol{\lambda}))}\right| \leq$$
$$\frac{\delta}{\gamma^2}|x^{(e)}(\bar{\boldsymbol{\lambda}}) - x^{(e)}(\boldsymbol{\lambda})|$$

To analyze the last term, notice that $\left|\frac{\partial}{\partial\lambda_i}[\dot{\Phi}_e]^{-1}(\boldsymbol{\lambda})\right| = \frac{1}{\ddot{\Phi}_e\left([\dot{\Phi}_e]^{-1}(\boldsymbol{\lambda})\right)} \leq \frac{1}{\gamma}$. Therefore, function $[\dot{\Phi}_e]^{-1}(\boldsymbol{\lambda})$ is Lipschitz continuous with constant $\frac{1}{\gamma}$. Using $x^{(e)}(\boldsymbol{\lambda}) = [\dot{\Phi}_e]^{-1}(\lambda_i - \lambda_j)$ it implies:

$$|x^{(e)}(\bar{\boldsymbol{\lambda}}) - x^{(e)}(\boldsymbol{\lambda})| \leq \frac{1}{\gamma}||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2$$

Combining these results gives:

$$||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq \frac{\mu_n(\boldsymbol{L}_\mathbb{G})\delta}{\gamma^3}||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2 = B||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2.$$

A.5. Proof of Lemma 3.1.3

Proof Using definition of $\boldsymbol{\epsilon}_k$ for the dual gradient we have:

$$\begin{aligned}
&\boldsymbol{g}(\boldsymbol{\lambda}_k + \alpha_k \tilde{\boldsymbol{d}}_k) = \boldsymbol{g}(\boldsymbol{\lambda_k}) + \int_0^1 \boldsymbol{H}(\boldsymbol{\lambda}_k + t\alpha_k \tilde{\boldsymbol{d}}_k)\alpha_k \tilde{\boldsymbol{d}}_k dt = \\
&\boldsymbol{g}(\boldsymbol{\lambda_k}) + \int_0^1 \left[\boldsymbol{H}(\boldsymbol{\lambda}_k + t\alpha_k \tilde{\boldsymbol{d}}_k) - \boldsymbol{H}(\boldsymbol{\lambda}_k)\right] \alpha_k \tilde{\boldsymbol{d}}_k dt + \alpha_k \int_0^1 \boldsymbol{H}(\boldsymbol{\lambda}_k)\tilde{\boldsymbol{d}}_k dt = \\
&\boldsymbol{g}(\boldsymbol{\lambda_k}) + \int_0^1 \left[\boldsymbol{H}(\boldsymbol{\lambda}_k + t\alpha_k \tilde{\boldsymbol{d}}_k) - \boldsymbol{H}(\boldsymbol{\lambda}_k)\right] \alpha_k \tilde{\boldsymbol{d}}_k dt + \alpha_k(\boldsymbol{\epsilon}_k - \boldsymbol{g}(\boldsymbol{\lambda}_k))
\end{aligned}$$

Applying $\boldsymbol{g}_{k+1} = \boldsymbol{g}(\boldsymbol{\lambda}_k + \alpha_k \tilde{\boldsymbol{d}}_k)$, $\boldsymbol{g}_k = \boldsymbol{g}(\boldsymbol{\lambda}_k)$ and Lemma 3.1.2:

$$\begin{aligned}
&||\boldsymbol{g}_{k+1}||_2 \leq (1-\alpha_k)||\boldsymbol{g}_k||_2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \frac{1}{2}\alpha_k^2 B||\tilde{\boldsymbol{d}}_k||_2^2 = \\
&(1-\alpha_k)||\boldsymbol{g}_k||_2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \frac{1}{2}\alpha_k^2 B||\boldsymbol{H}^\dagger(\boldsymbol{\lambda}_k)(\boldsymbol{g}_k - \boldsymbol{\epsilon}_k)||_2^2 \leq \\
&(1-\alpha_k)||\boldsymbol{g}_k||_2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \alpha_k^2 B||\boldsymbol{H}^\dagger(\boldsymbol{\lambda}_k)||_2^2(||\boldsymbol{g}_k||_2^2 + ||\boldsymbol{\epsilon}_k||_2^2)
\end{aligned}$$

Investigating the explicit form of dual Hessian gives $||\boldsymbol{H}^\dagger(\boldsymbol{\lambda}_k)||_2 \leq \frac{\Gamma}{\mu_2(\boldsymbol{L}_\mathbb{G})}$. Hence,

$$||\boldsymbol{g}_{k+1}||_2 \leq (1-\alpha_k)||\boldsymbol{g}_k||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_\mathbb{G})}||\boldsymbol{g}_k||_2^2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_\mathbb{G})}||\boldsymbol{\epsilon}_k||_2^2.$$

A.6. Proof of Lemma 3.1.4

Proof Combining $||\boldsymbol{g}_k||_2 \leq \frac{\mu_2^2(\boldsymbol{L}_\mathbb{G})}{2B\Gamma^2}$ with Lemma 3.1.3 implies:

$$||\boldsymbol{g}_{k+1}||_2 \leq \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}_k||_2 + \alpha_k||\boldsymbol{\epsilon}_k||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_\mathbb{G})}||\boldsymbol{\epsilon}_k||_2^2$$

Since $||\boldsymbol{\epsilon}_k||_2 \leq \epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}_k||_2$, $||\boldsymbol{g}_k||_2 \leq \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ and $\alpha_k \leq 1$, then

$$||\boldsymbol{g}_{k+1}||_2 \leq \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}_k||_2 + \alpha_k\epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}_k||_2 + \alpha_k^2\epsilon^2 B\frac{\Gamma^3}{\mu_2^3(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\gamma}||\boldsymbol{g}_k||_2^2 \leq$$
$$\left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}_k||_2 + \epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}_k||_2 + \epsilon^2 B\frac{\Gamma^3}{\mu_2^3(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\gamma}||\boldsymbol{g}_k||_2^2 \leq$$
$$\left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}_k||_2 + \frac{1}{2}\epsilon\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\Gamma\delta}\left[\sqrt{\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma}{\mu_n(\boldsymbol{L}_{\mathbb{G}})\Gamma}} + \frac{\epsilon}{2}\right] = \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}_k||_2 + \tilde{B}$$

where we denote $\tilde{B} = \frac{1}{2}\epsilon\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\Gamma\delta}\left[\sqrt{\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})\gamma}{\mu_n(\boldsymbol{L}_{\mathbb{G}})\Gamma}} + \frac{\epsilon}{2}\right] \leq 2\epsilon\frac{n\gamma^2}{\Gamma\delta}$ for $\epsilon \leq \frac{2}{n}\sqrt{\frac{2\gamma}{n\Gamma}}$. Since $||\boldsymbol{g}_{k+1}||_2 \geq |[\boldsymbol{g}_{k+1}]_i|$ and $||\boldsymbol{g}_k||_2 \leq \sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}$, then

$$|[\boldsymbol{g}_{k+1}]_i| \leq \left(\frac{3}{2} - \alpha_k\right)\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\} + 2\epsilon\frac{n\gamma^2}{\Gamma\delta}$$

Notice that if $m_i = 0$ the $\frac{3}{2} - \beta^{m_i} \leq 1 - \sigma\beta^{m_i}$ Therefore, for $m_i = 0$ we have

$$|[\boldsymbol{g}_{k+1}]_i| \leq (1 - \sigma\beta^{m_i})\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\} + 2\epsilon\frac{n\gamma^2}{\Gamma\delta}$$

In other words, Algorithm 11 returns $\alpha_k = \beta^0 = 1$.

For the case $||\boldsymbol{g}_k||_2 > \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ consider $\bar{\alpha}_k = \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}}$. Because $||\boldsymbol{g}_k||_2 \leq \sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}$

and $||\boldsymbol{g}_k||_2 > \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ then $\bar{\alpha}_k < 1$. Hence, applying $\bar{\alpha}_k$ with $\epsilon \leq \frac{2}{n}\sqrt{\frac{2\gamma}{n\Gamma}}$ for (3.7) gives:

$$
\begin{aligned}
&||\boldsymbol{g}_{k+1}||_2 \leq (1-\bar{\alpha}_k)||\boldsymbol{g}_k||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_k||_2^2 + \bar{\alpha}_k||\boldsymbol{\epsilon}_k||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}_k||_2^2 = \\
&||\boldsymbol{g}_k||_2 + \bar{\alpha}_k||\boldsymbol{\epsilon}_k||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}_k||_2^2 - \bar{\alpha}_k||\boldsymbol{g}_k||_2\left[1 - \bar{\alpha}_k B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_k||_2\right] \leq \\
&||\boldsymbol{g}_k||_2 + \bar{\alpha}_k\epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}_k||_2 + \bar{\alpha}_k^2\epsilon^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}||\boldsymbol{g}_k||_2^2 - \\
&\bar{\alpha}_k||\boldsymbol{g}_k||_2\left[1 - \frac{||\boldsymbol{g}_k||_2}{2\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}}\right] \leq ||\boldsymbol{g}_k||_2 + \bar{\alpha}_k\epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}_k||_2 + \\
&\bar{\alpha}_k^2\epsilon^2 B\frac{\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}||\boldsymbol{g}_k||_2^2 - \frac{1}{2}\bar{\alpha}_k||\boldsymbol{g}_k||_2 = \left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}_k||_2 + \\
&\epsilon\sqrt{\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}}\frac{||\boldsymbol{g}_k||_2}{\frac{2B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}} + \epsilon^2\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}\frac{1}{4\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}}\frac{||\boldsymbol{g}_k||_2^2}{n\max_i\{[\boldsymbol{g}_k]_i^2\}} \leq \\
&\left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}_k||_2 + \tilde{B} \leq \left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}_k||_2 + 2\epsilon\frac{n\gamma^2}{\delta\Gamma}
\end{aligned}
$$

In other words, we establishes:

$$||\boldsymbol{g}_{k+1}||_2 \leq (1 - \sigma\bar{\alpha}_k)||\boldsymbol{g}_k||_2 + 2\epsilon\frac{n\gamma^2}{\delta\Gamma}$$

Applying again $||\boldsymbol{g}_{k+1}||_2 \geq |[\boldsymbol{g}_{k+1}]_i|$ and $||\boldsymbol{g}_k||_2 \leq \sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\}$ gives:

$$|[\boldsymbol{g}_{k+1}]_i| \leq (1 - \sigma\bar{\alpha}_k)\sqrt{n}\max_i\{|[\boldsymbol{g}_k]_i|\} + 2\epsilon\frac{n\gamma^2}{\delta\Gamma}$$

Therefore, Algorithm 11 returns $\alpha_k \geq \beta\bar{\alpha}_k = \beta\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\max_i\{|[\boldsymbol{g}_k]_i|\}}$.

A.7. Proof of Theorem 3.1.5

Proof We will proof the above theorem by handling each of the cases separately. We start by considering the case when $||\boldsymbol{g}_k||_2 > \frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$. Then, according to Lemma 3.1.4: $\alpha_k \geq \beta\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\max_i\{|[\boldsymbol{g}_k]_i|\}}$ and Equation (3.7) we have:

$$||\boldsymbol{g}_{k+1}||_2 \leq (1 - \frac{1}{2}\beta\bar{\alpha}_k)||\boldsymbol{g}_k||_2 + 2\epsilon\frac{n\gamma^2}{\delta\Gamma}$$

Choosing $\epsilon \leq \frac{\beta}{8}\frac{\delta}{n\sqrt{n}}\left(\frac{\gamma}{\Gamma}\right)\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}$ implies $2\epsilon\frac{n\gamma^2}{\delta\Gamma} \leq \frac{1}{4}\beta\bar{\alpha}_k||\boldsymbol{g}_k||_2$ and

$$||\boldsymbol{g}_{k+1}||_2 - ||\boldsymbol{g}_k||_2 \leq -\frac{1}{4}\beta\bar{\alpha}_k)||\boldsymbol{g}_k||_2 \leq -\frac{1}{4}\beta\frac{||\boldsymbol{g}_k||_2}{2\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}\max_i\{|[\boldsymbol{g}_k]|_i\}} \leq$$
$$-\frac{1}{8}\beta\frac{1}{\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}} = -\frac{\beta}{8\sqrt{n}\delta}\frac{\gamma^3}{\Gamma^2}\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}$$

The the quadratic decrease phase we use the result of Lemma 3.1.4 and induction:

1. For $m = 1$ applying $\alpha_k = 1$ in Equation (3.7):

$$||\boldsymbol{g}_{k+1}||_2 \leq \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_k||_2^2 + \tilde{B} \leq \frac{1}{4\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}} + \tilde{B}$$

   This result validates the claim for $m = 1$.

2. Assume (3.8) is correct for some $m > 0$.

3. Using $\alpha_{k+m+1} = 1$ in Equation (3.7) and denoting $u = 2^{2^m}$ gives :

$$\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_{k+m+1}||_2 \leq \left[\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_{k+m}||_2\right]^2 + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} \leq$$
$$\left[\frac{1}{u} + \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})} + \hat{\Lambda}\frac{\frac{1}{2}u - 1}{u}\right]^2 + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} =$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{\frac{1}{2}u^2 - 1}{u^2} - \hat{\Lambda}\frac{\frac{1}{2}u^2 - 1}{u^2} + \hat{\Lambda}\frac{u-2}{u^2} + \tilde{B}\frac{2B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{1}{u} +$$
$$\left(\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right)^2\left[\tilde{B}^2 + 2\tilde{B}\hat{\Lambda}\frac{1}{\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}}\frac{(u-2)}{u} + \hat{\Lambda}^2\frac{1}{\left(\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right)^2}\frac{(u-2)^2}{4u^2}\right]$$

Since $\tilde{B} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B}^2 = \frac{\hat{\Lambda}}{4\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}}$, then

$$\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}_{k+m+1}||_2 \leq$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{\frac{1}{2}u^2-1}{u^2} - \hat{\Lambda}\frac{\frac{1}{2}u^2-1}{u^2} + \hat{\Lambda}\frac{u-2}{u^2} + \tilde{B}\frac{2B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{1}{u} +$$
$$\left(\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right)^2\left[\hat{\Lambda}\frac{1}{4\left(\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right)^2} - \frac{\tilde{B}}{\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}} + \frac{2\tilde{B}\hat{\Lambda}}{\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}}\left(\frac{1}{2}-\frac{1}{u}\right) + \frac{\hat{\Lambda}^2}{\left(\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\right)^2}\frac{(u-2)^2}{u^2}\right] =$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{(u^2-2)}{2u^2} + \frac{\hat{\Lambda}}{u^2}\left[-\frac{1}{2}u^2+u-1\right] + \frac{\hat{\Lambda}}{4} + \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{2}{u} +$$
$$\tilde{B}\hat{\Lambda}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\left[1-\frac{2}{u}\right] + \hat{\Lambda}^2\left(\frac{u-2}{2u}\right)^2 = \frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{(u^2-2)}{2u^2} -$$
$$\frac{\hat{\Lambda}}{u^2}\left(\frac{u}{2}-1\right)^2 + \hat{\Lambda}^2\left(\frac{1}{2}-\frac{1}{u}\right)^2 + \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\left[-1+\frac{2}{u}+\hat{\Lambda}-\frac{2}{u}\hat{\Lambda}\right] =$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{(u^2-2)}{2u^2} - \left(\frac{1}{2}-\frac{1}{u}\right)^2\left(\hat{\Lambda}-\hat{\Lambda}^2\right) - \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}(1-\hat{\Lambda})\left(1-\frac{2}{u}\right) \leq$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\tilde{B} + \hat{\Lambda}\frac{(u^2-2)}{2u^2} = \frac{1}{2^{2^{m+1}}} + \tilde{B}\frac{B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})} + \hat{\Lambda}\left[\frac{2^{2^{m+1}-1}-1}{2^{2^{m+1}}}\right]$$

The last step follows due to $u > 2$ and $\hat{\Lambda} < 1$ (choosing ϵ small enough).

Hence, claim (3.8) is correct.

A.8. Proof of Lemma 3.2.2

Proof Using the definition of $z_1, \dots z_p$ system (3.13) can be written as:

$$\begin{cases} \frac{\partial f_i}{\partial \phi_1^{(i)}} = -z_1 \\ \frac{\partial f_i}{\partial \phi_2^{(i)}} = -z_2 \\ \vdots \\ \frac{\partial f_i}{\partial \phi_p^{(i)}} = -z_p \end{cases} \tag{A.12}$$

Taking the derivative with respect to z_1 in each equation of system (A.12) gives:

$$\begin{cases} \frac{\partial^2 f_i}{\partial(\phi_1^{(i)})^2}\frac{\partial \phi_1^{(i)}}{\partial z_1} + \frac{\partial^2 f_i}{\partial \phi_1^{(i)}\partial \phi_2^{(i)}}\frac{\partial \phi_2^{(i)}}{\partial z_1} + \ldots + \frac{\partial^2 f_i}{\partial \phi_1^{(i)}\partial \phi_p^{(i)}}\frac{\partial \phi_p^{(i)}}{\partial z_1} = -1 \\ \frac{\partial^2 f_i}{\partial \phi_2^{(i)}\partial \phi_1^{(i)}}\frac{\partial \phi_1^{(i)}}{\partial z_1} + \frac{\partial^2 f_i}{\partial(\phi_2^{(i)})^2}\frac{\partial \phi_2^{(i)}}{\partial z_1} + \ldots + \frac{\partial^2 f_i}{\partial \phi_2^{(i)}\partial \phi_p^{(i)}}\frac{\partial \phi_p^{(i)}}{\partial z_1} = 0 \\ \vdots \\ \frac{\partial^2 f_i}{\partial \phi_p^{(i)}\partial \phi_1^{(i)}}\frac{\partial \phi_1^{(i)}}{\partial z_1} + \frac{\partial^2 f_i}{\partial \phi_p^{(i)}\partial \phi_2^{(i)}}\frac{\partial \phi_2^{(i)}}{\partial z_1} + \ldots + \frac{\partial^2 f_i}{\partial(\phi_p^{(i)})^2}\frac{\partial \phi_p^{(i)}}{\partial z_1} = 0 \end{cases}$$

Let $\boldsymbol{u}_1 = [\frac{\partial \phi_1^{(i)}}{\partial z_1}, \frac{\partial \phi_2^{(i)}}{\partial z_1}, \ldots, \frac{\partial \phi_p^{(i)}}{\partial z_1}]^\intercal$ then the above result can be written in matrix vector form:

$$[\nabla^2 f_i]\boldsymbol{u}_1 = -\boldsymbol{e}_1$$

where $\boldsymbol{e}_1 = [1, 0 \ldots, 0] \in \mathbb{R}^p$. Similarly we have:

$$[\nabla^2 f_i]\boldsymbol{u}_2 = -\boldsymbol{e}_2 \;\; [\nabla^2 f_i]\boldsymbol{u}_3 = -\boldsymbol{e}_3, \;\ldots\; [\nabla^2 f_i]\boldsymbol{u}_p = -\boldsymbol{e}_p$$

with $\boldsymbol{u}_r = [\frac{\partial \phi_1^{(i)}}{\partial z_r}, \frac{\partial \phi_2^{(i)}}{\partial z_r}, \ldots, \frac{\partial \phi_p^{(i)}}{\partial z_r}]^\intercal$. Combining all these equations gives:

$$[\nabla^2 f_i]\boldsymbol{U} = -\boldsymbol{I}_{p\times p} \tag{A.13}$$

where

$$\boldsymbol{U} = \begin{bmatrix} \frac{\partial \phi_1^{(i)}}{\partial z_1} & \frac{\partial \phi_1^{(i)}}{\partial z_2} & \cdots & \frac{\partial \phi_1^{(i)}}{\partial z_p} \\ \frac{\partial \phi_2^{(i)}}{\partial z_1} & \frac{\partial \phi_2^{(i)}}{\partial z_2} & \cdots & \frac{\partial \phi_2^{(i)}}{\partial z_p} \\ \vdots & & \ddots & \vdots \\ \frac{\partial \phi_p^{(i)}}{\partial z_1} & \frac{\partial \phi_p^{(i)}}{\partial z_2} & \cdots & \frac{\partial \phi_p^{(i)}}{\partial z_p} \end{bmatrix}$$

Notice, Equation (A.13) implies:

$$\boldsymbol{U} = -[\nabla^2 f_i]^{-1}$$

Hence, using Assumption 3.2.1: $||\boldsymbol{U}||_2 \leq \frac{1}{\gamma}$, and:

$$|U_{ij}| \leq ||\boldsymbol{U}||_F \leq \sqrt{p}||\boldsymbol{U}||_2 \leq \frac{\sqrt{p}}{\gamma}$$

A.9. Proof of Lemma 3.2.3

Proof The proof of the first claim can be found in the proof of Lemma 3.1.2.

Following the strategy as in Lemma 3.1.2 and using $\boldsymbol{M} \preceq \mu_n(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{I}$:

$$
\begin{aligned}
&||[\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})]\boldsymbol{v}||_2^2 = ||[\boldsymbol{M}([\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1})\boldsymbol{M}\boldsymbol{v}]||_2^2 = \\
&\boldsymbol{v}^{\mathsf{T}}\boldsymbol{M}([\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1})\boldsymbol{M}^{\boldsymbol{2}}([\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1})\boldsymbol{M}\boldsymbol{v} \leq \\
&\mu_n^2(\boldsymbol{L}_{\mathbb{G}})\boldsymbol{v}^{\mathsf{T}}\boldsymbol{M}([\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1})^2\boldsymbol{M}\boldsymbol{v} \leq \\
&\mu_n^2(\boldsymbol{L}_{\mathbb{G}})\mu_{max}^2(|[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}|)\boldsymbol{v}^{\mathsf{T}}\boldsymbol{M}^2\boldsymbol{v} \leq \\
&\mu_n^4(\boldsymbol{L}_{\mathbb{G}})\mu_{max}^2(|[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}|)||\boldsymbol{v}||_2^2
\end{aligned}
$$

Therefore,

$$
||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq \mu_n^2(\boldsymbol{L}_{\mathbb{G}})\mu_{max}(|[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}|) \tag{A.14}
$$

To bound the term $\mu_{max}(|[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}|)$ we study study the properties of primal Hessian more carefully:

Claim: For primal Hessian $\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))$ the following properties are true

$$
\gamma \preceq \nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda})) \preceq \Gamma \tag{A.15}
$$

$$
\begin{aligned}
&\mu_{max}(|[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}|) \leq \\
&\delta \max_{i \in \mathbb{V}} \sqrt{\sum_{k=1}^{p} \left([\boldsymbol{y}_k]_i(\bar{\boldsymbol{\lambda}}) - [\boldsymbol{y}_k]_i(\boldsymbol{\lambda})\right)^2}
\end{aligned} \tag{A.16}
$$

for any $\bar{\boldsymbol{\lambda}}, \boldsymbol{\lambda} \in \mathbb{R}^p$.

Proof Firstly, notice that for any $j \neq i$ and any $r = 1 \ldots, p$:

$$
\frac{\partial^2 f}{\partial [\boldsymbol{y}_1]_i \partial [\boldsymbol{y}_r]_j} = \frac{\partial^2 f}{\partial [\boldsymbol{y}_2]_i \partial [\boldsymbol{y}_r]_j} = \ldots = \frac{\partial^2 f}{\partial [\boldsymbol{y}_p]_i \partial [\boldsymbol{y}_r]_j} = 0
$$

Hence, the sparsity pattern of primal Hessian allows the symmetric reordering of rows and columns

such that $\nabla^2 f(\boldsymbol{\lambda})$ is transformed into the block diagonal matrix:

$$\boldsymbol{W}(\boldsymbol{\lambda}) = \begin{bmatrix} \nabla^2 f_1(\boldsymbol{\lambda}) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \nabla^2 f_2(\boldsymbol{\lambda}) & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \nabla^2 f_p(\boldsymbol{\lambda}) \end{bmatrix}$$

The matrix $\boldsymbol{W}(\boldsymbol{\lambda})$ preserves the important properties of $\nabla^2 f(\boldsymbol{\lambda})$. Particularly, the spectrum of these two matrices are the same. Indeed, let $\boldsymbol{T}_{ij}$ is the operator that swaps i^{th} and j^{th} rows of some arbitrary matrix $\boldsymbol{A}$ and let $\bar{\boldsymbol{A}}$ be the result of such transformation. Then, $\bar{\boldsymbol{A}} = \boldsymbol{T}_{ij}\boldsymbol{A}\boldsymbol{T}_{ij}$, and using $\boldsymbol{T}_{ij}^2 = \boldsymbol{I}$:

$$det(\bar{\boldsymbol{A}} - \mu\boldsymbol{I}) = det(\boldsymbol{T}_{ij}\boldsymbol{A}\boldsymbol{T}_{ij} - \mu\boldsymbol{I}) = det(\boldsymbol{T}_{ij}(\boldsymbol{A} - \mu\boldsymbol{I})\boldsymbol{T}_{ij}) =$$
$$det(\boldsymbol{A} - \mu\boldsymbol{I})det(\boldsymbol{T}_{ij}^2) = det(\boldsymbol{A} - \mu\boldsymbol{I})$$

Since $\boldsymbol{W}(\boldsymbol{\lambda})$ is constructed from $\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))$ by symmetric reordering rows and columns, then $Spectrum(\boldsymbol{W}(\boldsymbol{\lambda})) = Spectrum(\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda})))$. Using the Assumption 3.2.1 it implies:

$$\gamma \preceq \boldsymbol{W}(\boldsymbol{\lambda}) \preceq \Gamma$$

To prove (A.16), notice that if $\bar{\boldsymbol{A}} = \boldsymbol{T}_{ij}\boldsymbol{A}\boldsymbol{T}_{ij}$ and $\boldsymbol{A}$ is invertible, then so $\bar{\boldsymbol{A}}$ and using $\boldsymbol{T}_{ij}^{-1} = \boldsymbol{T}_{ij}$:

$$det(\bar{\boldsymbol{A}}^{-1} - \mu\boldsymbol{I}) = det(\boldsymbol{T}_{ij}^{-1}\boldsymbol{A}^{-1}\boldsymbol{T}_{ij}^{-1} - \mu\boldsymbol{I}) = det(\boldsymbol{T}_{ij}(\boldsymbol{A}^{-1} - \mu\boldsymbol{I})\boldsymbol{T}_{ij}) =$$
$$det(\boldsymbol{A}^{-1} - \mu\boldsymbol{I})$$

Denote $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_l\}$ is a collection of operators that swaps the rows of matrix $\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))$ to transform it to $\boldsymbol{W}(\boldsymbol{\lambda})$, i.e.

$$\boldsymbol{W}(\boldsymbol{\lambda}) = \boldsymbol{T}_1 \cdots \boldsymbol{T}_l \nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))\boldsymbol{T}_l \cdots \boldsymbol{T}_1$$

Then $[\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1} = \boldsymbol{T}_l \cdots \boldsymbol{T}_1 \boldsymbol{W}^{-1}(\boldsymbol{\lambda}) \boldsymbol{T}_1 \cdots \boldsymbol{T}_l$, and using the Assumption 3.2.1:

$$
\begin{aligned}
&\mu_{max}(|||[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}||) = \\
&\mu_{max}(\boldsymbol{T}_l \cdots \boldsymbol{T}_1 |\boldsymbol{W}^{-1}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{W}^{-1}(\boldsymbol{\lambda})| \boldsymbol{T}_1 \cdots \boldsymbol{T}_l) \leq \mu_{max}(|\boldsymbol{W}^{-1}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{W}^{-1}(\boldsymbol{\lambda})|) \leq \\
&\max_{i \in \mathbb{V}} \mu_{max}(|[\nabla^2 f_i([\boldsymbol{y}_1]_i(\bar{\boldsymbol{\lambda}}), \dots [\boldsymbol{y}_p]_i(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f_i([\boldsymbol{y}_1]_i(\boldsymbol{\lambda}), \dots [\boldsymbol{y}_p]_i(\boldsymbol{\lambda}))]^{-1}|) = \\
&\max_{i \in \mathbb{V}} |||[\nabla^2 f_i([\boldsymbol{y}_1]_i(\bar{\boldsymbol{\lambda}}), \dots [\boldsymbol{y}_p]_i(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f_i([\boldsymbol{y}_1]_i(\boldsymbol{\lambda}), \dots [\boldsymbol{y}_p]_i(\boldsymbol{\lambda}))]^{-1}|||_2 \leq \\
&\frac{\delta}{\gamma^2} \max_{i \in \mathbb{V}} ||([\boldsymbol{y}_1]_i(\bar{\boldsymbol{\lambda}}), \dots [\boldsymbol{y}_p]_i(\bar{\boldsymbol{\lambda}})) - ([\boldsymbol{y}_1]_i(\boldsymbol{\lambda}), \dots [\boldsymbol{y}_p]_i(\boldsymbol{\lambda}))||_2 = \\
&\frac{\delta}{\gamma^2} \max_{i \in \mathbb{V}} \sqrt{\sum_{k=1}^{p} \left([\boldsymbol{y}_k]_i(\bar{\boldsymbol{\lambda}}) - [\boldsymbol{y}_k]_i(\boldsymbol{\lambda})\right)^2}
\end{aligned}
$$

which establishes (A.16).

Consider the term $\left([\boldsymbol{y}_k]_i(\bar{\boldsymbol{\lambda}}) - [\boldsymbol{y}_k]_i(\boldsymbol{\lambda})\right)$. Using the result (3.15) we can write:

$$
\begin{aligned}
&\left|y_k(i)(\bar{\boldsymbol{\lambda}}) - y_k(i)(\boldsymbol{\lambda})\right| = |\phi_k^{(i)}([\boldsymbol{L}_{\mathbb{G}} \bar{\boldsymbol{\lambda}}_1]_i, \dots, [\boldsymbol{L}_{\mathbb{G}} \bar{\boldsymbol{\lambda}}_p]_i) - \phi_k^i([\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_1]_i, \dots, [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_p]_i)| \leq \\
&\frac{\sqrt{p}}{\gamma} \sqrt{\sum_{r=1}^{p} \left([\boldsymbol{L}_{\mathbb{G}} \bar{\boldsymbol{\lambda}}_r]_i - [\boldsymbol{L}_{\mathbb{G}} \boldsymbol{\lambda}_r]_i\right)^2} = \frac{\sqrt{p}}{\gamma} \sqrt{\sum_{r=1}^{p} \left[\boldsymbol{L}_{\mathbb{G}} (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)\right]_i^2} \leq \frac{\sqrt{p}}{\gamma} \sqrt{\sum_{r=1}^{p} ||\boldsymbol{L}_{\mathbb{G}} (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)||_2^2} = \\
&\frac{\sqrt{p}}{\gamma} \sqrt{\sum_{r=1}^{p} (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)^\intercal \boldsymbol{L}_{\mathbb{G}}^2 (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)} \leq \frac{\sqrt{p}}{\gamma} \sqrt{\mu_n^2(\boldsymbol{L}_{\mathbb{G}}) \sum_{r=1}^{p} (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)^\intercal (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)} = \\
&= \mu_n(\boldsymbol{L}_{\mathbb{G}}) \frac{\sqrt{p}}{\gamma} ||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2
\end{aligned}
$$

where

$$
\sum_{r=1}^{p} (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r)^\intercal (\bar{\boldsymbol{\lambda}}_r - \boldsymbol{\lambda}_r) = ||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2^2
$$

is used. Hence,

$$
\left([\boldsymbol{y}_k]_i(\bar{\boldsymbol{\lambda}}) - [\boldsymbol{y}_k]_i(\boldsymbol{\lambda})\right)^2 \leq \mu_n^2(\boldsymbol{L}_{\mathbb{G}}) \frac{p}{\gamma^2} ||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2^2
$$

Combining this result with (A.16) gives:

$$
\mu_{max}(|||[\nabla^2 f(\boldsymbol{y}(\bar{\boldsymbol{\lambda}}))]^{-1} - [\nabla^2 f(\boldsymbol{y}(\boldsymbol{\lambda}))]^{-1}||) \leq \frac{\delta}{\gamma^2} \mu_n(\boldsymbol{L}_{\mathbb{G}}) \frac{p}{\gamma} ||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2
$$

and applying it to (A.14) gives:

$$||\boldsymbol{H}(\bar{\boldsymbol{\lambda}}) - \boldsymbol{H}(\boldsymbol{\lambda})||_2 \leq \delta p \left(\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\gamma}\right)^3 ||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_{\boldsymbol{2}} = B||\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}||_2$$

A.10. Proof of Lemma 3.2.4

Proof Let $\boldsymbol{d}^{*[k]} = \left(\left(\boldsymbol{d}_1^{*[k]}\right)^{\mathsf{T}}, \ldots, \left(\boldsymbol{d}_p^{*[k]}\right)^{\mathsf{T}}\right)^{\mathsf{T}}$ be the exact solution to system (3.17). One can see that $\boldsymbol{d}^{*[k]}$ is also exact solution of the original system (3.16). Indeed, $\boldsymbol{M}\boldsymbol{d}^{*[k]} = [\nabla^2 f(\boldsymbol{y}^{[k]})]\boldsymbol{y}^{[k]}$ implies:

$$[\nabla^2 f(\boldsymbol{y}^{[k]})]^{-1}\boldsymbol{M}\boldsymbol{d}^{*[k]} = \boldsymbol{y}^{[k]} \quad \Longrightarrow \quad \boldsymbol{M}[\nabla^2 f(\boldsymbol{y}^{[k]})]^{-1}\boldsymbol{M}\boldsymbol{d}^{*[k]} = \boldsymbol{M}\boldsymbol{y}^{[k]}.$$

Due to the definition of vectors $\tilde{\boldsymbol{d}}_1^{[k]}, \ldots, \tilde{\boldsymbol{d}}_p^{[k]}$ and using $\boldsymbol{H}^{[k]} \succeq \frac{\Gamma}{\mu_2(\boldsymbol{L})}\boldsymbol{M}$:

$$\begin{aligned}
&||\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]}||^2_{\boldsymbol{H}^{[k]}} = (\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]})^{\mathsf{T}}\boldsymbol{M}[\nabla^2 f(\boldsymbol{y}^{[k]})]^{-1}\boldsymbol{M}(\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]}) \leq \\
&\frac{1}{\gamma}(\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]})^{\mathsf{T}}\boldsymbol{M}^2(\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]}) \leq \frac{\mu_n(\boldsymbol{L})}{\gamma}||\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]}||^2_{\boldsymbol{M}} \leq \epsilon_0^2\frac{\mu_n(\boldsymbol{L})}{\gamma}||\boldsymbol{d}^{*[k]}||^2_{\boldsymbol{M}} \leq \\
&\epsilon_0^2\frac{\mu_n(\boldsymbol{L})}{\gamma}\frac{\Gamma}{\mu_2(\boldsymbol{L})}||\boldsymbol{d}^{*[k]}||^2_{\boldsymbol{H}^{[k]}}
\end{aligned}$$

Hence, we have:

$$||\tilde{\boldsymbol{d}}^{[k]} - \boldsymbol{d}^{*[k]}||_{\boldsymbol{H}^{[k]}} \leq \epsilon_0\sqrt{\frac{\mu_n(\boldsymbol{L})}{\mu_2(\boldsymbol{L})}\frac{\Gamma}{\gamma}}||\boldsymbol{d}^{*[k]}||_{\boldsymbol{H}^{[k]}} = \epsilon||\boldsymbol{d}^{*[k]}||_{\boldsymbol{H}^{[k]}} \tag{A.17}$$

This finishes the proof of the second statement of the lemma.

A.11. Proof of Lemma 3.2.5

Proof Using definition of $\boldsymbol{\epsilon}^{[k]}$ for the dual gradient we have:

$$\begin{aligned}
&\boldsymbol{g}(\boldsymbol{\lambda}^{[k]} + \alpha_k\tilde{\boldsymbol{d}^{[k]}}) = \boldsymbol{g}(\boldsymbol{\lambda}^{[k]}) + \int_0^1 \boldsymbol{H}(\boldsymbol{\lambda}^{[k]} + t\alpha_k\tilde{\boldsymbol{d}}^{[k]})\alpha_k\tilde{\boldsymbol{d}}^{[k]}dt = \\
&\boldsymbol{g}(\boldsymbol{\lambda}^{[k]}) + \int_0^1 \left[\boldsymbol{H}(\boldsymbol{\lambda}^{[k]} + t\alpha_k\tilde{\boldsymbol{d}}^{[k]}) - \boldsymbol{H}(\boldsymbol{\lambda}^{[k]})\right]\alpha_k\tilde{\boldsymbol{d}}^{[k]}dt + \alpha_k\int_0^1 \boldsymbol{H}(\boldsymbol{\lambda}^{[k]})\tilde{\boldsymbol{d}}^{[k]}dt = \\
&\boldsymbol{g}(\boldsymbol{\lambda}^{[k]}) + \int_0^1 \left[\boldsymbol{H}(\boldsymbol{\lambda}^{[k]} + t\alpha_k\tilde{\boldsymbol{d}^{[k]}}) - \boldsymbol{H}(\boldsymbol{\lambda}^{[k]})\right]\alpha_k\tilde{\boldsymbol{d}}^{[k]}dt + \alpha_k(\boldsymbol{\epsilon}^{[k]} - \boldsymbol{g}(\boldsymbol{\lambda}^{[k]}))
\end{aligned}$$

Applying $\boldsymbol{g}^{[k+1]} = \boldsymbol{g}(\boldsymbol{\lambda}^{[k]} + \alpha_k \tilde{\boldsymbol{d}}^{[k]})$, $\boldsymbol{g}^{[k]} = \boldsymbol{g}(\boldsymbol{\lambda}^{[k]})$ and Lemma 3.2.3:

$$
\begin{aligned}
&||\boldsymbol{g}^{[k+1]}||_2 \leq (1-\alpha_k)||\boldsymbol{g}^{[k]}||_2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \frac{1}{2}\alpha_k^2 B||\tilde{\boldsymbol{d}}^{[k]}||_2^2 = \\
&(1-\alpha_k)||\boldsymbol{g}^{[k]}||_2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \frac{1}{2}\alpha_k^2 B||\boldsymbol{H}^{\dagger}(\boldsymbol{\lambda}^{[k]})(\boldsymbol{g}^{[k]} - \boldsymbol{\epsilon}^{[k]})||_2^2 \leq \\
&(1-\alpha_k)||\boldsymbol{g}^{[k]}||_2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \alpha_k^2 B||\boldsymbol{H}^{\dagger}(\boldsymbol{\lambda}^{[k]})||_2^2(||\boldsymbol{g}^{[k]}||_2^2 + ||\boldsymbol{\epsilon}^{[k]}||_2^2)
\end{aligned}
$$

Investigating the explicit form of dual Hessian gives $||\boldsymbol{H}^{\dagger}(\boldsymbol{\lambda}^{[k]})||_2 \leq \frac{\Gamma}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}$. Hence,

$$
||\boldsymbol{g}^{[k+1]}||_2 \leq (1-\alpha_k)||\boldsymbol{g}^{[k]}||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k]}||_2^2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}^{[k]}||_2^2.
$$

A.12. Proof of Lemma 3.2.6

Proof Combining $||\boldsymbol{g}^{[k]}||_2 \leq \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ with Lemma 3.2.5 implies:

$$
||\boldsymbol{g}^{[k+1]}||_2 \leq \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}^{[k]}||_2 + \alpha_k||\boldsymbol{\epsilon}^{[k]}||_2 + \alpha_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}^{[k]}||_2^2
$$

Since $||\boldsymbol{\epsilon}^{[k]}||_2 \leq \epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2$, $||\boldsymbol{g}^{[k]}||_2 \leq \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ and $\alpha_k \leq 1$, then

$$
\begin{aligned}
&||\boldsymbol{g}^{[k+1]}||_2 \leq \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}^{[k]}||_2 + \alpha_k\epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2 + \alpha_k^2\epsilon^2 B\frac{\Gamma^3}{\mu_2^6(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n^2(\boldsymbol{L}_{\mathbb{G}})}{\gamma}||\boldsymbol{g}^{[k]}||_2^2 \leq \\
&\left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}^{[k]}||_2 + \epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2 + \epsilon^2 B\frac{\Gamma^3}{\mu_2^6(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n^2(\boldsymbol{L}_{\mathbb{G}})}{\gamma}||\boldsymbol{g}^{[k]}||_2^2 \leq \\
&\left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}^{[k]}||_2 + \frac{1}{2}\epsilon\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\mu_n(\boldsymbol{L}_{\mathbb{G}})p\Gamma\delta}\left[\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\gamma}{\Gamma}} + \frac{\epsilon}{2}\right] = \left(\frac{3}{2} - \alpha_k\right)||\boldsymbol{g}^{[k]}||_2 + \hat{B}
\end{aligned}
$$

where we denote $\hat{B} = \frac{1}{2}\epsilon\frac{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})\gamma^2}{\mu_n(\boldsymbol{L}_{\mathbb{G}})p\Gamma\delta}\left[\frac{\mu_2(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\gamma}{\Gamma}} + \frac{\epsilon}{2}\right] \leq 2\epsilon\frac{n\gamma^2}{p\Gamma\delta}$ for $\epsilon \leq \frac{4}{n^3}\sqrt{\frac{\gamma}{\Gamma}}$. Since $||\boldsymbol{g}^{[k+1]}||_2 \geq \max_r\{|[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_r^{[k+1]}]_i|\}$ and $||\boldsymbol{g}^{[k]}||_2 \leq \sqrt{n}\max_i\{\eta_i\}$, then

$$
\max_r\{|[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_r^{[k+1]}]_i|\} \leq \left(\frac{3}{2} - \alpha_k\right)\sqrt{n}\max_i\{\eta_i\} + 2\epsilon\frac{n\gamma^2}{p\Gamma\delta}
$$

Notice that if $m_i = 0$ the $\frac{3}{2} - \beta^{m_i} \leq 1 - \sigma\beta^{m_i}$ Therefore, for $m_i = 0$ we have

$$
\max_r\{|[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_r^{[k+1]}]_i|\} \leq (1 - \sigma\beta^{m_i})\sqrt{n}\max_i\{\eta_i\} + 2\epsilon\frac{n\gamma^2}{p\Gamma\delta}
$$

In other words, Algorithm 12 returns $\alpha_k = \beta^0 = 1$.

For the case $||\boldsymbol{g}^{[k]}||_2 > \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ consider $\bar{\alpha}_k = \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\sqrt{n}\max_i\{\eta_i\}}$. Because $||\boldsymbol{g}^{[k]}||_2 \leq \sqrt{n}\max_i\{\eta_i\}$ and $||\boldsymbol{g}^{[k]}||_2 > \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$ then $\bar{\alpha}_k < 1$. Hence, applying $\bar{\alpha}_k$ with $\epsilon \leq \frac{4}{n^3}\sqrt{\frac{\gamma}{\Gamma}}$ for (3.20) gives:

$$
\begin{aligned}
&||\boldsymbol{g}^{[k+1]}||_2 \leq (1-\bar{\alpha}_k)||\boldsymbol{g}^{[k]}||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k]}||_2^2 + \bar{\alpha}_k||\boldsymbol{\epsilon}^{[k]}||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}^{[k]}||_2^2 = \\
&||\boldsymbol{g}^{[k]}||_2 + \bar{\alpha}_k||\boldsymbol{\epsilon}^{[k]}||_2 + \bar{\alpha}_k^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{\epsilon}^{[k]}||_2^2 - \bar{\alpha}_k||\boldsymbol{g}^{[k]}||_2\left[1 - \bar{\alpha}_k B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k]}||_2\right] \leq \\
&||\boldsymbol{g}^{[k]}||_2 + \bar{\alpha}_k\epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2 + \bar{\alpha}_k^2\epsilon^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}||\boldsymbol{g}^{[k]}||_2^2 - \\
&\bar{\alpha}_k||\boldsymbol{g}^{[k]}||_2\left[1 - \frac{||\boldsymbol{g}^{[k]}||_2}{2\sqrt{n}\max_i\{\eta_i\}}\right] \leq ||\boldsymbol{g}^{[k]}||_2 + \bar{\alpha}_k\epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2 + \\
&\bar{\alpha}_k^2\epsilon^2 B\frac{\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\frac{\mu_n^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}||\boldsymbol{g}^{[k]}||_2^2 - \frac{1}{2}\bar{\alpha}_k||\boldsymbol{g}^{[k]}||_2 = \left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}^{[k]}||_2 + \\
&\epsilon\frac{\mu_n(\boldsymbol{L}_{\mathbb{G}})}{\mu_n(\boldsymbol{L}_{\mathbb{G}})}\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{g}^{[k]}||_2\frac{||\boldsymbol{g}^{[k]}||_2}{\frac{2B\Gamma^2}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}\max_i\{\eta_i\}} + \epsilon^2\frac{\mu_n^2(\boldsymbol{L}_{\mathbb{G}})}{\mu_2^2(\boldsymbol{L}_{\mathbb{G}})}\frac{\Gamma}{\gamma}\frac{1}{4\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}}\frac{||\boldsymbol{g}^{[k]}||_2^2}{n\max_i\{\eta_i^2\}} \leq \\
&\left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}^{[k]}||_2 + \hat{B} \leq \left(1 - \frac{\bar{\alpha}_k}{2}\right)||\boldsymbol{g}^{[k]}||_2 + 2\epsilon\frac{n\gamma^2}{p\delta\Gamma}
\end{aligned}
$$

In other words, we establishes:

$$||\boldsymbol{g}^{[k+1]}||_2 \leq (1 - \sigma\bar{\alpha}_k)||\boldsymbol{g}^{[k]}||_2 + 2\epsilon\frac{n\gamma^2}{p\delta\Gamma}$$

Applying again $||\boldsymbol{g}^{[k+1]}||_2 \geq \max_r\{|[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_r^{[k+1]}]_i|\}$ and $||\boldsymbol{g}^{[k]}||_2 \leq \sqrt{n}\max_i\{\eta_i\}$ gives:

$$\max_r\{|[\boldsymbol{L}_{\mathbb{G}}\boldsymbol{y}_r^{[k+1]}]_i|\} \leq (1 - \sigma\bar{\alpha}_k)\sqrt{n}\max_i\{\eta_i\} + 2\epsilon\frac{n\gamma^2}{p\Gamma\delta}$$

Therefore, Algorithm 12 returns $\alpha_k \geq \beta\bar{\alpha}_k = \beta\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\max_i\{\eta_i\}}$.

A.13. Proof of Theorem 3.2.7

Proof We will proof the above theorem by handling each of the cases separately. We start by considering the case when $||\boldsymbol{g}^{[k]}||_2 > \frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2}$. Then, according to Lemma 3.2.6: $\alpha_k \geq \beta\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{2B\Gamma^2\max_i\{\eta_i\}}$

and Equation (3.20) we have:

$$||\boldsymbol{g}^{[k+1]}||_2 \leq (1 - \frac{1}{2}\beta\bar{\alpha}_k)||\boldsymbol{g}^{[k]}||_2 + 2\epsilon\frac{n\gamma^2}{p\delta\Gamma}$$

Choosing $\epsilon \leq \frac{\beta}{8}\frac{\gamma^3}{\Gamma^2 p\delta}\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{\mu_n^3(\boldsymbol{L}_{\mathbb{G}})}$ implies $2\epsilon\frac{n\gamma^2}{p\delta\Gamma} \leq \frac{1}{4}\beta\bar{\alpha}_k||\boldsymbol{g}^{[k]}||_2$ and

$$||\boldsymbol{g}^{[k+1]}||_2 - ||\boldsymbol{g}^{[k]}||_2 \leq -\frac{1}{4}\beta\bar{\alpha}_k||\boldsymbol{g}^{[k]}||_2 \leq -\frac{1}{4}\beta\frac{||\boldsymbol{g}^{[k]}||_2}{2\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}\max_i\{\eta_i\}} \leq$$
$$-\frac{1}{8}\beta\frac{1}{\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\sqrt{n}} = -\frac{\beta}{8\sqrt{n}p\delta}\frac{\gamma^3}{\Gamma^2}\frac{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}{\mu_n^3(\boldsymbol{L}_{\mathbb{G}})}$$

The the quadratic decrease phase we use the result of Lemma 3.2.6 and induction:

1. For $m = 1$ applying $\alpha_k = 1$ in Equation (3.20):

$$||\boldsymbol{g}^{[k+1]}||_2 \leq \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k]}||_2^2 + \hat{B} \leq \frac{1}{4\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}} + \hat{B}$$

   This result validates the claim for $m = 1$.

2. Assume (3.21) is correct for some $m > 0$.

3. Using $\alpha_{k+m+1} = 1$ in Equation (3.20) and denoting $u = 2^{2^m}$ gives :

$$\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k+m+1]}||_2 \leq \left[\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k+m]}||_2\right]^2 + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} \leq$$
$$\left[\frac{1}{u} + \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})} + \tilde{\Lambda}\frac{\frac{1}{2}u - 1}{u}\right]^2 + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} =$$
$$\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{\frac{1}{2}u^2 - 1}{u^2} - \tilde{\Lambda}\frac{\frac{1}{2}u^2 - 1}{u^2} + \tilde{\Lambda}\frac{u-2}{u^2} + \hat{B}\frac{2B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\frac{1}{u} +$$
$$\left(\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right)^2\left[\hat{B}^2 + 2\hat{B}\tilde{\Lambda}\frac{1}{\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}}\frac{(u-2)}{u} + \tilde{\Lambda}^2\frac{1}{\left(\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right)^2}\frac{(u-2)^2}{4u^2}\right]$$

Since $\hat{B} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B}^2 = \frac{\tilde{\Lambda}}{4\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}}$, then

$$
\begin{aligned}
&\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}||\boldsymbol{g}^{[k+m+1]}||_2 \leq \\
&\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{\frac{1}{2}u^2-1}{u^2} - \tilde{\Lambda}\frac{\frac{1}{2}u^2-1}{u^2} + \tilde{\Lambda}\frac{u-2}{u^2} + \hat{B}\frac{2B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\frac{1}{u} + \\
&\left(\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right)^2\left[\tilde{\Lambda}\frac{1}{4\left(\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right)^2} - \frac{\hat{B}}{\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}} + \frac{2\hat{B}\tilde{\Lambda}}{\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}}\left(\frac{1}{2}-\frac{1}{u}\right) + \frac{\tilde{\Lambda}^2}{\left(\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\right)^2}\frac{(u-2)^2}{u^2}\right] = \\
&\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{(u^2-2)}{2u^2} + \frac{\tilde{\Lambda}}{u^2}\left[-\frac{1}{2}u^2+u-1\right] + \frac{\tilde{\Lambda}}{4} + \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\frac{2}{u} + \\
&\hat{B}\tilde{\Lambda}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\left[1-\frac{2}{u}\right] + \tilde{\Lambda}^2\left(\frac{u-2}{2u}\right)^2 = \frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{(u^2-2)}{2u^2} - \\
&\frac{\tilde{\Lambda}}{u^2}\left(\frac{u}{2}-1\right)^2 + \tilde{\Lambda}^2\left(\frac{1}{2}-\frac{1}{u}\right)^2 + \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\left[-1+\frac{2}{u}+\tilde{\Lambda}-\frac{2}{u}\tilde{\Lambda}\right] = \\
&\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{(u^2-2)}{2u^2} - \left(\frac{1}{2}-\frac{1}{u}\right)^2\left(\tilde{\Lambda}-\tilde{\Lambda}^2\right) - \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}(1-\tilde{\Lambda})\left(1-\frac{2}{u}\right) \leq \\
&\frac{1}{u^2} + \frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})}\hat{B} + \tilde{\Lambda}\frac{(u^2-2)}{2u^2} = \frac{1}{2^{2^{m+1}}} + \hat{B}\frac{B\Gamma^2}{\mu_2^4(\boldsymbol{L}_{\mathbb{G}})} + \tilde{\Lambda}\left[\frac{2^{2^{m+1}-1}-1}{2^{2^{m+1}}}\right]
\end{aligned}
$$

The last step follows due to $u > 2$ and $\hat{\Lambda} < 1$ (choosing ϵ small enough).

Hence, claim (3.21) is correct.

A.14. Proof of Lemma 3.2.9

Proof Using finite difference formula (3.23) the r^{th} component of Hessian-vector product $\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i$ is given as:

$$[\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i]_r \approx \frac{\frac{\partial f_i}{\partial[\boldsymbol{y}_r]_i}((1+t)\boldsymbol{\mathcal{Y}}_i) - \frac{\partial f_i}{\partial[\boldsymbol{y}_r]_i}((1-t)\boldsymbol{\mathcal{Y}}_i)}{2t}$$

The total error of the this approximation consists of rounding truncation parts. Let $\mathcal{B}(\boldsymbol{\mathcal{Y}}_i, t)$ is the ball in $\mathbb{R}^p$ centered at $\boldsymbol{\mathcal{Y}}_i$ with radius t and let $A = \sup_{\boldsymbol{x}\in\mathcal{B}(\boldsymbol{\mathcal{Y}}_i,t)}\left|\frac{\partial f_i}{\partial[\boldsymbol{y}_r]_i}\right|$, then the rounding error can be bounded as follows:

$$
\begin{aligned}
&\left|\frac{\frac{\partial \tilde{f}_i}{\partial[\boldsymbol{y}_r]_i}((1+t)\boldsymbol{\mathcal{Y}}_i) - \frac{\partial \tilde{f}_i}{\partial[\boldsymbol{y}_r]_i}((1-t)\boldsymbol{\mathcal{Y}}_i)}{2t} - \frac{\frac{\partial f_i}{\partial[\boldsymbol{y}_r]_i}((1+t)\boldsymbol{\mathcal{Y}}_i) - \frac{\partial f_i}{\partial[\boldsymbol{y}_r]_i}((1-t)\boldsymbol{\mathcal{Y}}_i)}{2t}\right| \\
&\leq \frac{A\epsilon_{machine}}{t}
\end{aligned}
$$

For the truncation error, using $B = \sup_{\boldsymbol{x}\in\mathcal{B}(\boldsymbol{\mathcal{Y}}_i,t)} \left|\sum_{h=1}^{p}\sum_{q=1}^{p}\sum_{s=1}^{p}\frac{\partial^3 f_i(\boldsymbol{x})}{\partial x_h \partial x_q \partial x_s}\right|$ in the Taylor expansion:

$$\left|\frac{\frac{\partial f_i}{\partial [\boldsymbol{y}_r]_i}((1+t)\boldsymbol{\mathcal{Y}}_i) - \frac{\partial f_i}{\partial [\boldsymbol{y}_r]_i}((1-t)\boldsymbol{\mathcal{Y}}_i)}{2t} - \underbrace{[\boldsymbol{y}_r]_i \frac{\partial f_i}{\partial [\boldsymbol{y}_r]_i}(\boldsymbol{\mathcal{Y}}_i)}_{[\nabla^2 f_i(\boldsymbol{\mathcal{Y}}_i)\boldsymbol{\mathcal{Y}}_i]_r}\right| \leq \frac{t^2 B}{6}$$

Hence, the total error can be bounded as

$$Error_{total} \leq \frac{A\epsilon_{machine}}{t} + \frac{t^2 B}{6} \tag{A.18}$$

which is minimized at $t^* = \sqrt[3]{\frac{3A\epsilon_{machine}}{B}} \approx \mathcal{O}(\sqrt[3]{\epsilon_{machine}})$

A.15. Proof of Lemma 3.2.10

Proof Using definition of $\boldsymbol{\epsilon}_i^{[t]}$ for the gradient of function $\hat{f}_i(\cdot)$ we have:

$$\begin{aligned}
&\boldsymbol{h}_i(\boldsymbol{\mathcal{Y}}_i^{[t]} + \alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]}) = \boldsymbol{h}_i(\boldsymbol{\mathcal{Y}}_i^{[t]}) + \int_0^1 \boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]} + s\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]})\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]} ds = \\
&\boldsymbol{h}_i^{[t]} + \int_0^1 \left[\boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]} + s\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]}) - \boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})\right]\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]} ds + \alpha_{it}\int_0^1 \boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]} ds = \\
&\boldsymbol{h}_i^{[t]} + \int_0^1 \left[\boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]} + s\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]}) - \boldsymbol{H}_i(\boldsymbol{\mathcal{Y}}_i^{[t]})\right]\alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]} ds + \alpha_{it}(\boldsymbol{\epsilon}_i^{[t]} - \boldsymbol{h}_i^{[t]})
\end{aligned}$$

Applying $\boldsymbol{h}_i^{[t+1]} = \boldsymbol{h}_i(\boldsymbol{\mathcal{Y}}_i^{[t]} + \alpha_{it}\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]})$, $\boldsymbol{h}_i^{[t]} = \boldsymbol{h}(\boldsymbol{\mathcal{Y}}_i^{[t]})$ and guarantees (3.29):

$$\begin{aligned}
&||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1-\alpha_{it})||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \frac{1}{2}\alpha_{it}^2\delta||\widetilde{\Delta\boldsymbol{\mathcal{Y}}_i}^{[t]}||_2^2 = \\
&(1-\alpha_{it})||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \frac{1}{2}\alpha_{it}^2\delta||\boldsymbol{H}_i^{\dagger}(\boldsymbol{\mathcal{Y}}_i^{[t]})(\boldsymbol{h}_i^{[t]} - \boldsymbol{\epsilon}_i^{[t]})||_2^2 \leq \\
&(1-\alpha_{it})||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \alpha_{it}^2\delta||\boldsymbol{H}_i^{\dagger}(\boldsymbol{\mathcal{Y}}_i^{[t]})||_2^2(||\boldsymbol{h}_i^{[t]}||_2^2 + ||\boldsymbol{\epsilon}_i^{[t]}||_2^2)
\end{aligned}$$

Investigating the explicit form of dual Hessian gives $||\boldsymbol{H}_i^{\dagger}(\boldsymbol{\mathcal{Y}}_i^{[t]})||_2 \leq \frac{1}{\gamma}$. Hence,

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1-\alpha_{it})||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}^2\delta\frac{1}{\gamma^2}||\boldsymbol{h}_i^{[t]}||_2^2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \alpha_{it}^2\delta\frac{1}{\gamma^2}||\boldsymbol{\epsilon}_i^{[t]}||_2^2$$

A.16. Proof of Lemma 3.2.11

Proof Combining $||\boldsymbol{h}_i^{[t]}||_2 \leq \frac{\gamma^2}{2\delta}$ with Lemma 3.2.10 implies:

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq \left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \alpha_{it}^2 \frac{\delta}{\gamma^2}||\boldsymbol{\epsilon}_i^{[t]}||_2^2$$

Since $||\boldsymbol{\epsilon}_i^{[t]}||_2 \leq \zeta\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{h}_i^{[t]}||_2$, $||\boldsymbol{h}_i^{[t]}||_2 \leq \frac{\gamma^2}{2\delta}$ and $\alpha_{it} \leq 1$, then

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq \left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}\zeta\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{h}_i^{[t]}||_2 + \alpha_{it}^2\zeta^2\frac{\delta}{\gamma^2}\frac{\Gamma}{\gamma}||\boldsymbol{h}_i^{[t]}||_2^2 \leq$$
$$\left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{h}_i^{[t]}||_2 + \zeta^2\frac{\delta\Gamma}{\gamma^3}||\boldsymbol{h}_i^{[t]}||_2^2 \leq$$
$$\left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{2\delta} + \zeta^2\frac{\delta\Gamma}{\gamma^3}\frac{\gamma^4}{4\delta^2} = \left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \tilde{B}_i$$

where we denote $\tilde{B}_i = \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{2\delta}\left[1 + \frac{\zeta}{2}\sqrt{\frac{\gamma}{\Gamma}}\right]$ and for $\zeta \leq 2\sqrt{\frac{\gamma}{\Gamma}}$ it implies that $\tilde{B}_i \leq \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$. Hence,

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq \left(\frac{3}{2} - \alpha_{it}\right) ||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$$

Notice that if $m_i = 0$ the $\frac{3}{2} - \beta^{m_i} \leq 1 - \sigma\beta^{m_i}$ Therefore, for $m_i = 0$ we have

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1 - \sigma\beta^{m_i}) ||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$$

In other words, Algorithm 17 returns $\alpha_{it} = \beta^0 = 1$.

For the case $||\boldsymbol{h}_i^{[t]}||_2 > \frac{\gamma^2}{2\delta}$ consider $\bar{\alpha}_{it} = \frac{\gamma^2}{2\delta||\boldsymbol{h}_i^{[t]}||_2}$. Because $||\boldsymbol{h}_i^{[t]}||_2 > \frac{\gamma^2}{2\delta}$ then $\bar{\alpha}_{it} < 1$. Hence,

applying $\bar{\alpha}_{it}$ with $\zeta \leq 2\sqrt{\frac{\gamma}{\Gamma}}$ for (3.2.10) gives:

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1-\bar{\alpha}_{it})||\boldsymbol{h}_i^{[t]}||_2 + \bar{\alpha}_{it}^2\frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t]}||_2^2 + \bar{\alpha}_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \bar{\alpha}_{it}^2\frac{\delta}{\gamma^2}||\boldsymbol{\epsilon}_i^{[t]}||_2^2 =$$
$$||\boldsymbol{h}_i^{[t]}||_2 + \bar{\alpha}_{it}||\boldsymbol{\epsilon}_i^{[t]}||_2 + \bar{\alpha}_{it}^2\frac{\delta}{\gamma^2}||\boldsymbol{\epsilon}_i^{[t]}||_2^2 - \bar{\alpha}_{it}||\boldsymbol{h}_i^{[t]}||_2\left[1-\bar{\alpha}_{it}\frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t]}||_2\right] \leq$$
$$||\boldsymbol{h}_i^{[t]}||_2 + \bar{\alpha}_{it}\zeta\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{h}_i^{[t]}||_2 + \bar{\alpha}_{it}^2\zeta^2\frac{\delta\Gamma}{\gamma^3}||\boldsymbol{h}_i^{[t]}||_2^2 - \frac{1}{2}\bar{\alpha}_{it}||\boldsymbol{h}_i^{[t]}||_2 \leq$$
$$\left(1-\frac{\bar{\alpha}_{it}}{2}\right)||\boldsymbol{h}_i^{[t]}||_2 + \frac{\gamma^2}{2\delta||\boldsymbol{h}_i^{[t]}||_2}\zeta\sqrt{\frac{\Gamma}{\gamma}}||\boldsymbol{h}_i^{[t]}||_2 + \frac{\gamma^4}{4\delta^2||\boldsymbol{h}_i^{[t]}||_2^2}\frac{\delta}{\gamma^2}\zeta^2\frac{\Gamma}{\gamma}||\boldsymbol{h}_i^{[t]}||_2^2 \leq$$
$$\left(1-\frac{\bar{\alpha}_{it}}{2}\right)||\boldsymbol{h}_i^{[t]}||_2 + \tilde{B}_i \leq \left(1-\frac{\bar{\alpha}_{it}}{2}\right)||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$$

In other words, we establishes:

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq (1-\sigma\bar{\alpha}_{it})||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$$

Therefore, Algorithm 17 returns $\alpha_{it} \geq \beta\bar{\alpha}_{it} = \beta\frac{\gamma^2}{2\delta||\boldsymbol{h}_i^{[t]}||_2}$.

A.17. Proof of Theorem 3.2.12

Proof We will proof the above theorem by handling each of the cases separately. We start by considering the case when $||\boldsymbol{h}_i^{[t]}||_2 > \frac{\gamma^2}{2\delta}$. Then, according to Lemma 3.2.10: $\alpha_{it} \geq \beta\frac{\gamma^2}{2\delta||\boldsymbol{h}_i^{[t]}||_2}$ and Equation (3.34) we have:

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq \left(1-\frac{1}{2}\beta\bar{\alpha}_{it}\right)||\boldsymbol{h}_i^{[t]}||_2 + \zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta}$$

Choosing $\zeta \leq \frac{\beta}{8}\sqrt{\frac{\gamma}{\Gamma}}$ implies $\zeta\sqrt{\frac{\Gamma}{\gamma}}\frac{\gamma^2}{\delta} \leq \frac{1}{4}\beta\bar{\alpha}_{it}||\boldsymbol{h}_i^{[t]}||_2$ and

$$||\boldsymbol{h}_i^{[t+1]}||_2 - ||\boldsymbol{h}_i^{[t]}||_2 \leq -\frac{1}{4}\beta\bar{\alpha}_{it}||\boldsymbol{h}_i^{[t]}||_2 \leq -\frac{1}{4}\beta\frac{\gamma^2||\boldsymbol{h}_i^{[t]}||_2}{2\delta||\boldsymbol{h}_i^{[t]}||_2} =$$
$$-\frac{\beta\gamma^2}{8\delta}$$

The the quadratic decrease phase we use the result of Lemma 3.2.11 and induction:

1. For $m = 1$ applying $\alpha_{it} = 1$ in Equation (3.34):

$$||\boldsymbol{h}_i^{[t+1]}||_2 \leq \frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t]}||_2^2 + \tilde{B}_i \leq \frac{1}{4\frac{\delta}{\gamma^2}} + \tilde{B}_i$$

   This result validates the claim for $m = 1$.

2. Assume (3.35) is correct for some $m > 0$.

3. Using $\alpha_{i(t+m+1)} = 1$ in Equation (3.34) and denoting $u = 2^{2^m}$ gives :

$$\frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t+m+1]}||_2 \leq \left[\frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t+m]}||_2\right]^2 + \frac{\delta}{\gamma^2}\tilde{B}_i \leq$$
$$\left[\frac{1}{u} + \tilde{B}_i\frac{\delta}{\gamma^2} + \hat{\Lambda}_i\frac{\frac{1}{2}u-1}{u}\right]^2 + \frac{\delta}{\gamma^2}\tilde{B}_i =$$
$$\frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{\frac{1}{2}u^2-1}{u^2} - \hat{\Lambda}_i\frac{\frac{1}{2}u^2-1}{u^2} + \hat{\Lambda}_i\frac{u-2}{u^2} + \tilde{B}_i\frac{2\delta}{\gamma^2}\frac{1}{u} +$$
$$\left(\frac{\delta}{\gamma^2}\right)^2\left[\tilde{B}_i^2 + 2\tilde{B}_i\hat{\Lambda}_i\frac{1}{\frac{\delta}{\gamma^2}}\frac{(u-2)}{u} + \hat{\Lambda}_i^2\frac{1}{\left(\frac{\delta}{\gamma^2}\right)^2}\frac{(u-2)^2}{4u^2}\right]$$

   Since $\tilde{B}_i + \frac{\delta}{\gamma^2}\tilde{B}_i^2 = \frac{\hat{\Lambda}_i}{4\frac{\delta}{\gamma^2}}$, then

$$\frac{\delta}{\gamma^2}||\boldsymbol{h}_i^{[t+m+1]}||_2 \leq$$
$$\frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{\frac{1}{2}u^2-1}{u^2} - \hat{\Lambda}_i\frac{\frac{1}{2}u^2-1}{u^2} + \hat{\Lambda}_i\frac{u-2}{u^2} + \tilde{B}_i\frac{2\delta}{\gamma^2}\frac{1}{u} +$$
$$\left(\frac{\delta}{\gamma^2}\right)^2\left[\hat{\Lambda}_i\frac{1}{4\left(\frac{\delta}{\gamma^2}\right)^2} - \frac{\tilde{B}_i}{\frac{\delta}{\gamma^2}} + \frac{2\tilde{B}_i\hat{\Lambda}_i}{\frac{\delta}{\gamma^2}}\left(\frac{1}{2}-\frac{1}{u}\right) + \frac{\hat{\Lambda}_i^2}{\left(\frac{\delta}{\gamma^2}\right)^2}\frac{(u-2)^2}{u^2}\right] =$$
$$\frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{(u^2-2)}{2u^2} + \frac{\hat{\Lambda}_i}{u^2}\left[-\frac{1}{2}u^2 + u - 1\right] + \frac{\hat{\Lambda}_i}{4} + \tilde{B}_i\frac{\delta}{\gamma^2}\frac{2}{u} +$$
$$\tilde{B}_i\hat{\Lambda}_i\frac{\delta}{\gamma^2}\left[1-\frac{2}{u}\right] + \hat{\Lambda}_i^2\left(\frac{u-2}{2u}\right)^2 = \frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{(u^2-2)}{2u^2} -$$
$$\frac{\hat{\Lambda}_i}{u^2}\left(\frac{u}{2}-1\right)^2 + \hat{\Lambda}_i^2\left(\frac{1}{2}-\frac{1}{u}\right)^2 + \tilde{B}_i\frac{\delta}{\gamma^2}\left[-1+\frac{2}{u}+\hat{\Lambda}_i-\frac{2}{u}\hat{\Lambda}_i\right] =$$
$$\frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{(u^2-2)}{2u^2} - \left(\frac{1}{2}-\frac{1}{u}\right)^2\left(\hat{\Lambda}_i - \hat{\Lambda}_i^2\right) - \tilde{B}_i\frac{\delta}{\gamma^2}(1-\hat{\Lambda}_i)\left(1-\frac{2}{u}\right) \leq$$
$$\frac{1}{u^2} + \frac{\delta}{\gamma^2}\tilde{B}_i + \hat{\Lambda}_i\frac{(u^2-2)}{2u^2} = \frac{1}{2^{2^{m+1}}} + \tilde{B}_i\frac{\delta}{\gamma^2} + \hat{\Lambda}_i\left[\frac{2^{2^{m+1}-1}-1}{2^{2^{m+1}}}\right]$$

The last step follows due to $u > 2$ and $\hat{\Lambda}_i < 1$ (choosing ζ small enough).

Hence, claim (3.35) is correct.

A.18. Proof Of Lemma 3.2.14

Proof Following exactly the same strategy as in Byrd et al. (2016) instead of studying the properties of approximated hessian inverse $\widetilde{\boldsymbol{H}}_i^{[r]}$ we will study the corresponding properties of hessian approximation $\widetilde{\boldsymbol{B}}_i^{[r]}$ given by BFGS formula:

$$\begin{aligned}
&\widetilde{\boldsymbol{B}}_i^{[r]} = \widetilde{\boldsymbol{B}}_i^{[r],0} = \frac{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{s}^{[r]}}\boldsymbol{I}, \\
&\widetilde{\boldsymbol{B}}_i^{[r],k+1} = \widetilde{\boldsymbol{B}}_i^{[r],k} - \frac{\widetilde{\boldsymbol{B}}_i^{[r],k}\boldsymbol{s}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\boldsymbol{B}}_i^{[r],k}}{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\boldsymbol{B}}_i^{[r],k}\boldsymbol{s}^{[j_k]}} + \frac{\boldsymbol{z}^{[j_k]}\boldsymbol{z}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}} \\
&\text{for } k = 0, \ldots, \tilde{m}-1 \text{ and } j_k = t - \tilde{m} + 1 + k \\
&\widetilde{\boldsymbol{B}}_i^{[r+1],0} = \widetilde{\boldsymbol{B}}_i^{[r],\tilde{m}} = \widetilde{\boldsymbol{B}}_i^{[r+1]}
\end{aligned}$$

For $\widetilde{\boldsymbol{B}}_i^{[r],0}$ we have:

$$\widetilde{\boldsymbol{B}}_i^{[r],0} = \frac{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{s}^{[r]}}\boldsymbol{I} = \frac{\boldsymbol{s}^{[r]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^2\boldsymbol{s}^{[r]}}{\boldsymbol{s}^{[r]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[r]}}\boldsymbol{I} = \frac{(\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}))^{\frac{1}{2}}\boldsymbol{s}^{[r]})^{\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})(\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}))^{\frac{1}{2}}\boldsymbol{s}^{[r]})}{(\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}))^{\frac{1}{2}}\boldsymbol{s}^{[r]})^{\mathsf{T}}(\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}))^{\frac{1}{2}}\boldsymbol{s}^{[r]})}\boldsymbol{I}$$

Hence, using $\frac{\gamma}{N}\boldsymbol{I} \preceq \widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) \preceq \frac{\Gamma}{N}\boldsymbol{I}$ gives:

$$\frac{\gamma}{N}\boldsymbol{I} \leq \frac{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{s}^{[r]}}\boldsymbol{I} \leq \frac{\Gamma}{N}\boldsymbol{I} \tag{A.19}$$

Applying the above result for the trace of $\widetilde{\boldsymbol{B}}_i^{[r+1],0}$ gives:

$$\begin{aligned}
&\mathrm{Tr}(\widetilde{\boldsymbol{B}}_i^{[r+1],0}) = \mathrm{Tr}(\widetilde{\boldsymbol{B}}_i^{[r],0}) + \sum_{k=1}^{\tilde{m}}\frac{||\boldsymbol{z}^{[j_k]}||_2^2}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}} - \sum_{i=1}^{\tilde{m}}\frac{||\widetilde{\boldsymbol{B}}_i^{[r],k}\boldsymbol{s}^{[j_k]}||_2^2}{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\boldsymbol{B}}_i^{[r],k}\boldsymbol{s}^{[j_k]}} \leq \\
&\mathrm{Tr}(\widetilde{\boldsymbol{B}}_i^{[r],0}) + \frac{\Gamma}{N}\tilde{m} \leq \frac{\Gamma}{N}(\tilde{m}+p)
\end{aligned}$$

which immediately establishes the upperbound for largest eigenvalue of matrix $\widetilde{\boldsymbol{B}}_i^{[r+1],0}$. Therefore,

$$\boldsymbol{H}_t \succeq \nu_1\boldsymbol{I}$$

with $\nu_1 = \frac{N}{\Gamma(\tilde{m}+p)}$.

To establish the lower bound we study BFGS formula for hessian inverse, given in Algorithm 19:

$$\widetilde{\boldsymbol{H}}_i^{[r]} = \widetilde{\boldsymbol{H}}_i^{[r],0} = \frac{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{s}^{[r]}}{\boldsymbol{z}^{[r]\mathsf{T}}\boldsymbol{z}^{[r]}}\boldsymbol{I},$$
$$\widetilde{\boldsymbol{H}}_i^{[r],k+1} = \left[\boldsymbol{I} - \frac{\boldsymbol{s}^{[j_k]}\boldsymbol{z}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right]\widetilde{\boldsymbol{H}}_i^{[r],k}\left[\boldsymbol{I} - \frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right] + \frac{\boldsymbol{s}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}$$
$$\text{for } k = 0, \ldots, \tilde{m}-1 \text{ and } j_k = r - \tilde{m} + 1 + k$$
$$\widetilde{\boldsymbol{H}}_i^{[r+1],0} = \widetilde{\boldsymbol{H}}_i^{[r],\tilde{m}} = \widetilde{\boldsymbol{H}}_i^{[r+1]}$$

Using $\frac{\gamma}{N}\boldsymbol{I} \preceq \widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) \preceq \frac{\Gamma}{N}\boldsymbol{I}$, $\mathrm{Tr}(\boldsymbol{CD}) = \mathrm{Tr}(\boldsymbol{D}^\mathsf{T}\boldsymbol{C}^\mathsf{T})$ and cyclic property of the trace:

$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],0}) = \frac{\boldsymbol{s}^{[r]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[r]}}{\boldsymbol{s}^{[r]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^2\boldsymbol{s}^{[r]}} \leq \frac{N}{\gamma}p$$
$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k+1}) =$$
$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k}) + \mathrm{Tr}\left(\frac{\boldsymbol{s}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}_{j_i}^\mathsf{T}\boldsymbol{s}_{j_i}}\right) - 2\,\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) + \mathrm{Tr}\left(\frac{\boldsymbol{s}^{[j_k]}\boldsymbol{z}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) =$$
$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k}) + \mathrm{Tr}\left(\frac{\boldsymbol{s}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) - 2\,\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) + \frac{||\boldsymbol{s}^{[j_k]}||_2^2}{(\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]})^2}\boldsymbol{z}^{[j_k]\mathsf{T}}\widetilde{\boldsymbol{H}}_i^{[r],k}\boldsymbol{z}^{[j_k]} \leq$$
$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k}) + \frac{||\boldsymbol{s}^{[j_k]}||_2^2}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}} - 2\,\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) + \frac{N}{\gamma}\,\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{z}^{[j_k]}}{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[j_k]}}$$
$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k}) + \frac{N}{\gamma} - 2\,\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) + \frac{\Gamma}{\gamma}\,\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k})$$

Notice,

$$\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\frac{\boldsymbol{z}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}}{\boldsymbol{z}^{[j_k]\mathsf{T}}\boldsymbol{s}^{[j_k]}}\right) =$$
$$\frac{1}{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[j_k]}}\,\mathrm{Tr}\left(\widetilde{\boldsymbol{H}}_i^{[r],k})\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[j_k]}\boldsymbol{s}^{[j_k]\mathsf{T}}\right) = \frac{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[j_k]}}{\boldsymbol{s}^{[j_k]\mathsf{T}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})\boldsymbol{s}^{[j_k]}} \geq 0$$

The last inequality follows from the fact that $\widetilde{\boldsymbol{H}}_i^{[r],k}$ and $\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})$ are positive semi definite and, hence, $\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})$ is positive semi definite. Indeed,

$$\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]}) = \widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}} =$$
$$\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{-\frac{1}{2}}\left[\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}\right]\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}$$

Hence, the spectrum of matrices $\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})$ and $\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}\widetilde{\boldsymbol{H}}_i^{[r],k}\widetilde{\nabla}^2\tilde{f}_i(\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]})^{\frac{1}{2}}$ are the same, and the latter matrix is positive semi definite. Applying this result gives:

$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],0}) \leq \frac{N}{\gamma}p$$

$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k+1}) \leq \mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],k})\left[1+\frac{\Gamma}{\gamma}\right]+\frac{N}{\gamma}$$

The above recursive inequality gives:

$$\mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],\tilde{m}}) \leq \mathrm{Tr}(\widetilde{\boldsymbol{H}}_i^{[r],0})\left[1+\frac{\Gamma}{\gamma}\right]^{\tilde{m}}+\frac{N}{\gamma}\sum_{q=0}^{\tilde{m}-1}\left[1+\frac{\Gamma}{\gamma}\right]^{q} \leq$$
$$N\left[\left(\frac{p}{\gamma}+\frac{1}{\Gamma}\right)\left[1+\frac{\Gamma}{\gamma}\right]^{\tilde{m}}-\frac{1}{\Gamma}\right]$$

Hence, using that $\widetilde{\boldsymbol{H}}_i^{[r],\tilde{m}} = \widetilde{\boldsymbol{H}}_i^{[r+1]}$

$$\widetilde{\boldsymbol{H}}_i^{[r+1]} \preceq \nu_2\boldsymbol{I}$$

with $\nu_2 = N\left[\left(\frac{p}{\gamma}+\frac{1}{\Gamma}\right)\left[1+\frac{\Gamma}{\gamma}\right]^{\tilde{m}}-\frac{1}{\Gamma}\right]$.

A.19. Proof of Theorem 3.2.15

Proof To simplify the description we drop index i and denote $\boldsymbol{\mathcal{Y}}_i^{[t]} = \boldsymbol{\mathcal{Y}}^{[t]}$, $\overline{\boldsymbol{\mathcal{Y}}}_i^{[r]} = \overline{\boldsymbol{\mathcal{Y}}}^{[r]}$, $\boldsymbol{\mathcal{Y}}_i^* = \boldsymbol{\mathcal{Y}}^*$ then:

$$\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t+1]}) = \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]} - \beta_t\widetilde{\boldsymbol{H}}_i^{[r+1]}\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})) \leq$$
$$\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \beta_t\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})^{\intercal}\widetilde{\boldsymbol{H}}_i^{[r+1]}\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) + \frac{\Gamma}{2N}(\beta_t\nu_2)^2||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2$$

Taking the expectation $\mathbb{E}_{\mathcal{S}_{g,t}}$ from both sides and using Lemma 3.2.14:

$$\mathbb{E}_{\mathcal{S}_{g,t}}[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t+1]})] \leq \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \beta_t\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})^{\intercal}\widetilde{\boldsymbol{H}}_i^{[r+1]}\mathbb{E}_{\mathcal{S}_{g,t}}[\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})] + \frac{\Gamma}{2N}(\beta_t\nu_2)^2\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right] =$$
$$\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \beta_t\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})^{\intercal}\widetilde{\boldsymbol{H}}_i^{[r+1]}\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) + \frac{\Gamma}{2N}(\beta_t\nu_2)^2\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right] \leq$$
$$\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \beta_t\nu_1||\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2 + \frac{\Gamma}{2N}(\beta_t\nu_2)^2\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right]$$

To proceed, we need to bound $\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right]$ and $||\nabla\tilde{f}_r(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2$. Let us denote $\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]}) = \ell_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})+\sum_{j=1}^{p}[\boldsymbol{\mathcal{Y}}^{[t]}]_j[\boldsymbol{L}\boldsymbol{\lambda}_{\boldsymbol{j}}]_i$ then using Cauchy-Schwarz inequality and $\mathbb{P}r_{\mathcal{S}_{g,t}}\left[\mathbb{1}\{j\in\mathcal{S}_{g,t}\}\mathbb{1}\{i\in\mathcal{S}_{g,t}\}\right] = \frac{b_g}{N}\frac{b_g-1}{N}$:

$$\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right] = \frac{1}{|\mathcal{S}_{g,t}|^2}\left[\sum_{r=1}^{N}\sum_{j=1}^{N}\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})^{\mathsf{T}}\nabla\tilde{\ell}_{ir}(\boldsymbol{\mathcal{Y}}^{[t]})\mathbb{E}_{\mathcal{S}_{g,t}}[\mathbb{1}\{j\in\mathcal{S}_{g,t}\}\mathbb{1}\{i\in\mathcal{S}_{g,t}\}]\right] =$$
$$\frac{1}{b_g^2}\left[\sum_{j=1}^{N}||\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\frac{b_g}{N} + 2\frac{b_g(b_g-1)}{N^2}\sum_{r\neq j}^{N}\nabla\tilde{\ell}_{rj}(\boldsymbol{\mathcal{Y}}^{[t]})^{\mathsf{T}}\nabla\tilde{\ell}_{ir}(\boldsymbol{\mathcal{Y}}^{[t]})\right] \leq$$
$$\frac{1}{b_gN}\left[\sum_{j=1}^{N}||\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2 + 2\frac{(b_g-1)}{N}\sum_{r\neq j}^{N}||\nabla\tilde{\ell}_{rj}(\boldsymbol{\mathcal{Y}}^{[t]})||_2||\nabla\tilde{\ell}_{ri}(\boldsymbol{\mathcal{Y}}^{[t]})||_2\right] \leq$$
$$\frac{1}{b_gN}\left[\sum_{j=1}^{N}||\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2\right]^2 \leq \frac{1}{b_g}\sum_{j=1}^{N}||\nabla\tilde{l}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2$$

where $|\mathcal{S}_{g,k}| = b_g$. Due to (3.29), the curvature of function $\tilde{\ell}_{ij}$ satisfies:

$$\frac{\gamma}{N}\boldsymbol{I} \preceq \nabla^2\tilde{\ell}_{ij} \preceq \frac{\Gamma}{N}\boldsymbol{I}$$

and it is easy to establish the following bounds:

$$||\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2 \geq 2\frac{\gamma}{N}[\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{*})] \tag{A.20}$$
$$||\nabla\tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2 \leq 2\frac{\Gamma}{N}[\hat{l}_{rj}(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{\ell}_{ij}(\boldsymbol{\mathcal{Y}}^{*})]$$

Therefore, we establish the following bound:

$$\mathbb{E}_{\mathcal{S}_{g,t}}\left[||\widetilde{\nabla}\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2\right] \leq \frac{2\Gamma}{b_g}\left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{*})\right] \tag{A.21}$$

The next step is to bound the term $||\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2$. Because $\frac{\gamma}{N}\boldsymbol{I} \preceq \nabla^2\tilde{f}_i \preceq \frac{\Gamma}{N}\boldsymbol{I}$, hence similarly as in (A.20):

$$||\nabla\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]})||_2^2 \geq 2\frac{\gamma}{N}\left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{*})\right] \tag{A.22}$$

Therefore, combining results (A.21) and (A.22):

$$\mathbb{E}_{\mathcal{S}_{g,t}}[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t+1]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{*})] \leq \left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^{*})\right]\left[1 + \frac{(\nu_2\Gamma)^2}{b_gN}\beta_t^2 - 2\frac{\gamma\nu_1}{N}\beta_t\right]$$

Taking expectation $\mathbb{E}_{\mathcal{S}_{g,t-1}}$ from both sides gives:

$$\mathbb{E}_{\mathcal{S}_{g,t}}[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t+1]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^*)] \leq \left[\frac{(\nu_2\Gamma)^2}{b_g N}\beta_t^2 - 2\frac{\gamma\nu_1}{N}\beta_t + 1\right] \mathbb{E}_{\mathcal{S}_{g,t-1}}\left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^*)\right]$$

Therefore, choosing step size $\beta_t = \frac{\gamma\nu_1 b_g}{(\nu_2\Gamma)^2}$ immediately gives:

$$\mathbb{E}_{\mathcal{S}_{g,t}}[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[t+1]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^*)] \leq \left[1 - \left(\frac{\gamma\nu_1}{\Gamma\nu_2}\right)^2 \frac{b_g}{N}\right]^{t+1} \left[\tilde{f}_i(\boldsymbol{\mathcal{Y}}^{[0]}) - \tilde{f}_i(\boldsymbol{\mathcal{Y}}^*)\right]$$

achieving linear convergence rate.

A.20. Experiments for Network Flow Problem

In this paragraph we present increased versions of Figures 9 - 13:

Figure 27: Experimental Results for Small Random Graph

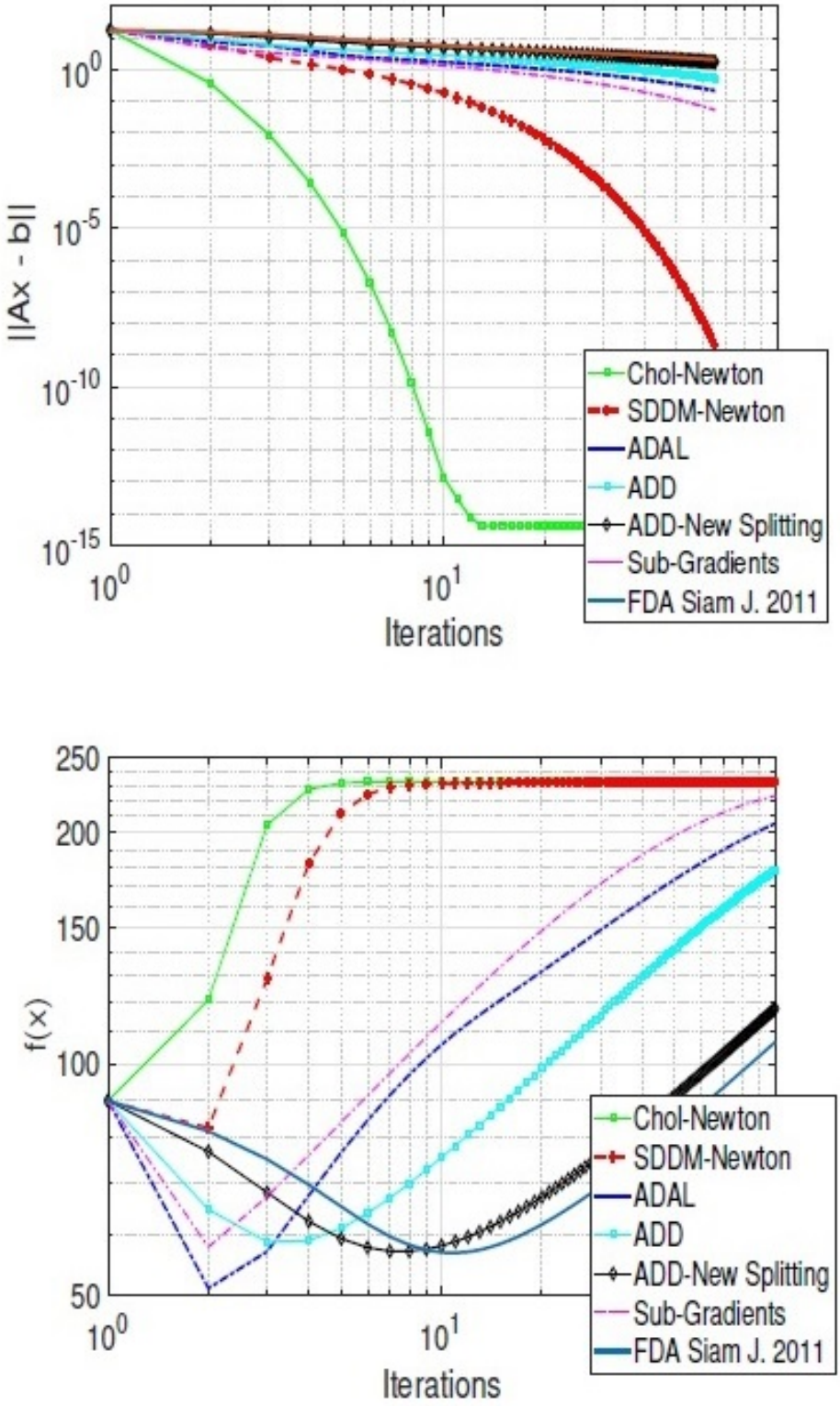


Figure 28: Experimental Results for Large Random Graph

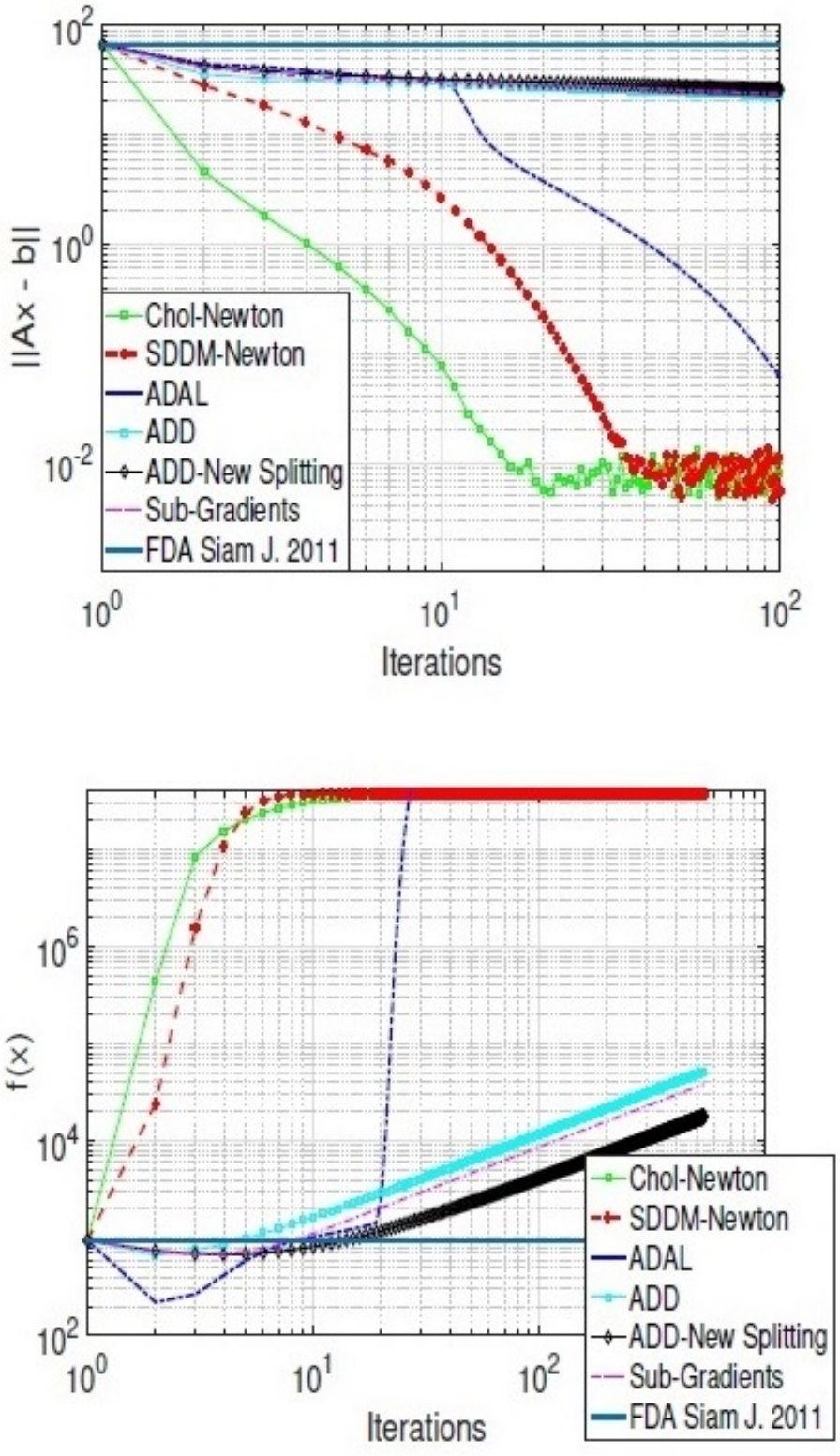
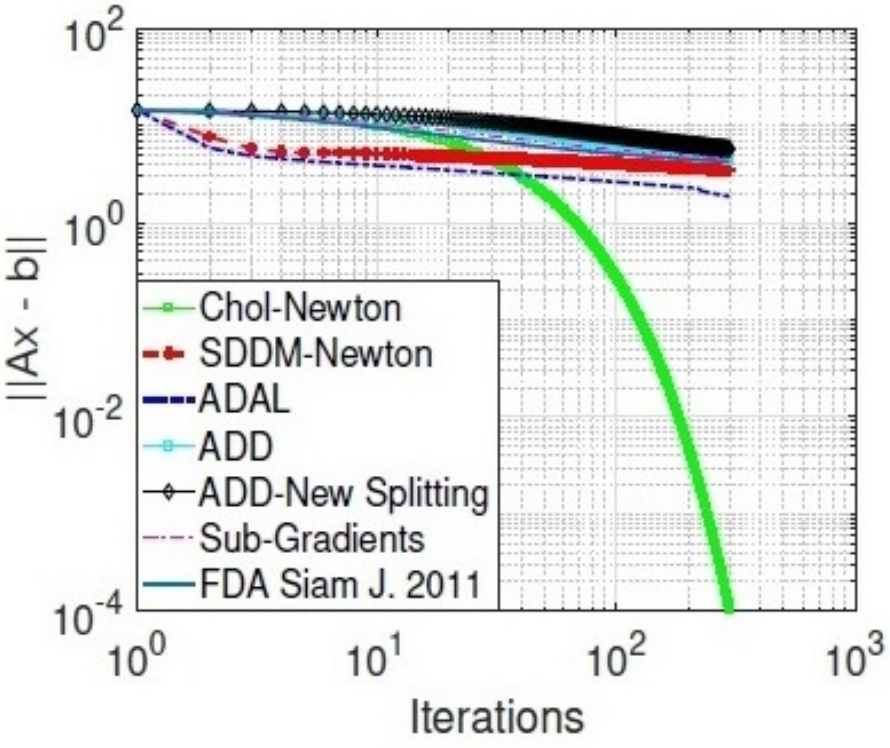


Figure 29: Experimental Results for Bar-Bell Graph



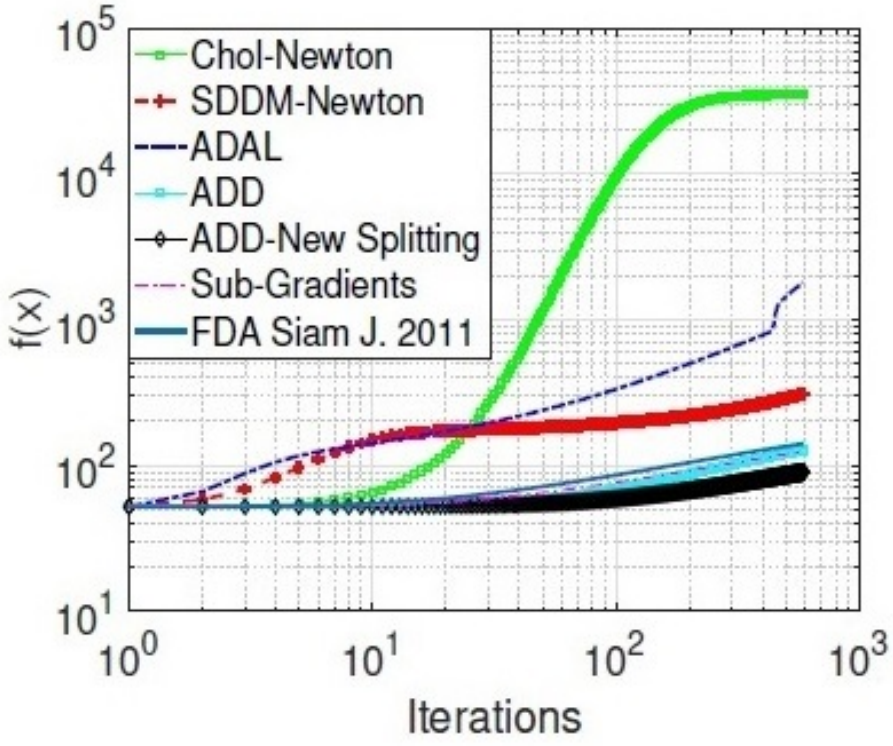


Figure 30: Experimental Results for Bar-Star Graph

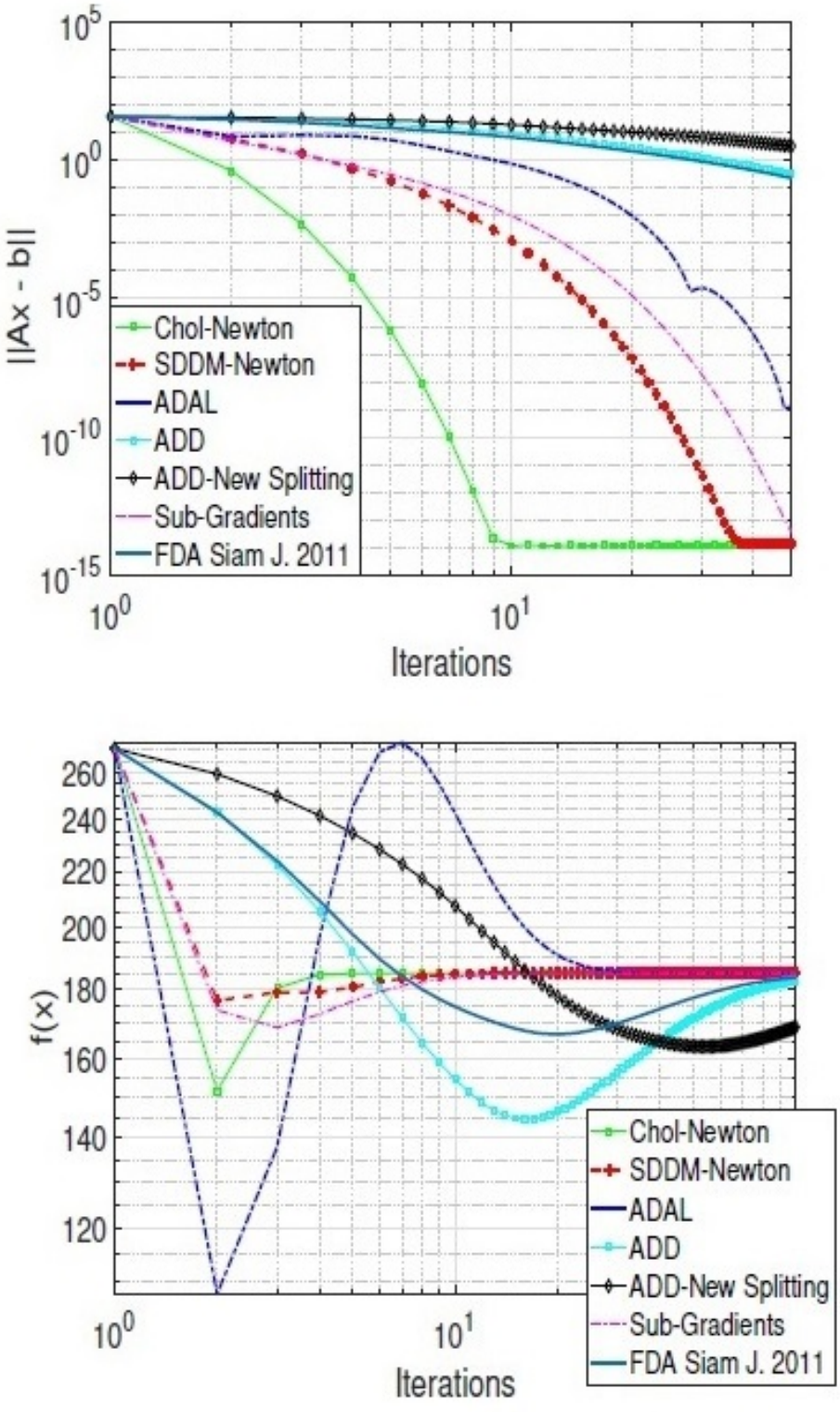
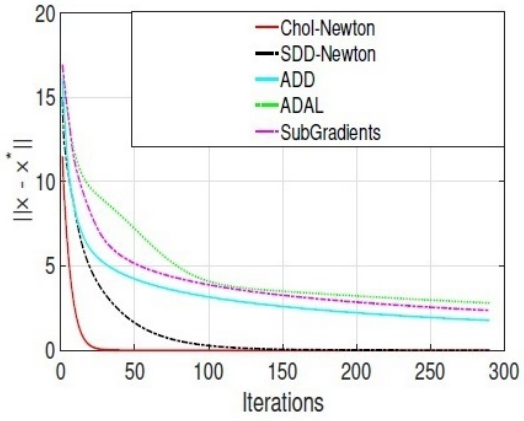
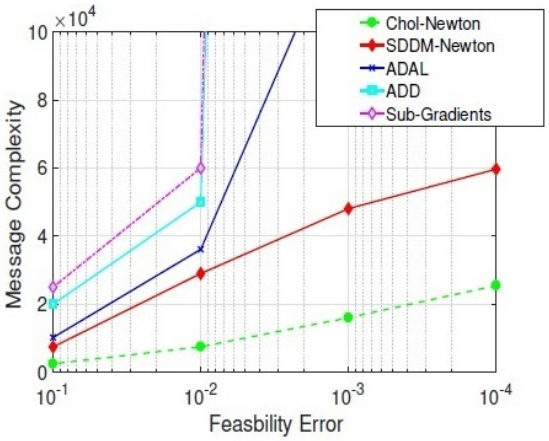


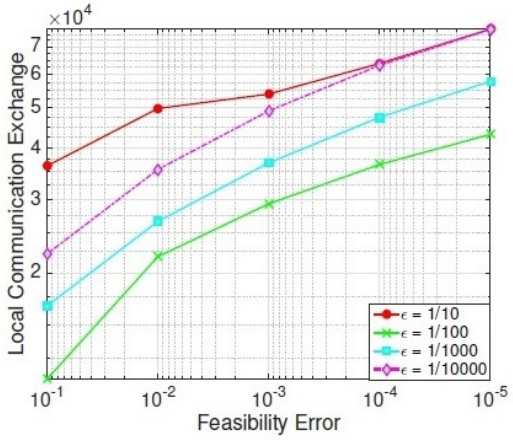
Figure 31: Experimental results: convergence, communication overhead, accuracy effect



(a) Convergence to $\boldsymbol{x}^\star$



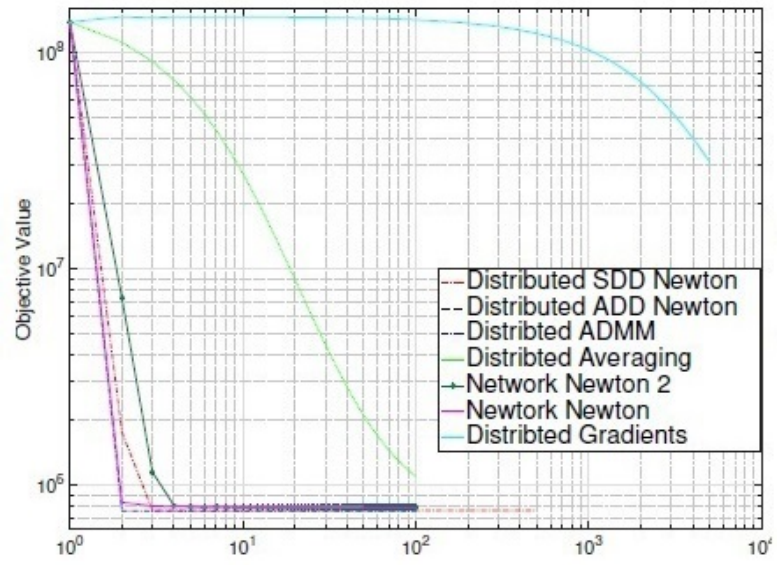
(b) Local Exchange



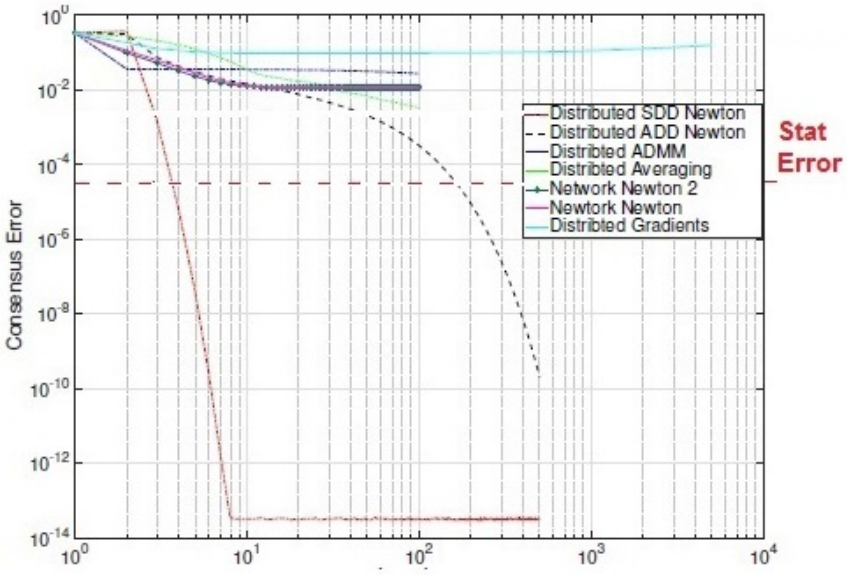
(c) Effect of ϵ

A.21. Experiment for Empirical Risk Minimization Problem

In this paragraph we present increased versions of Figures 20 - 26:

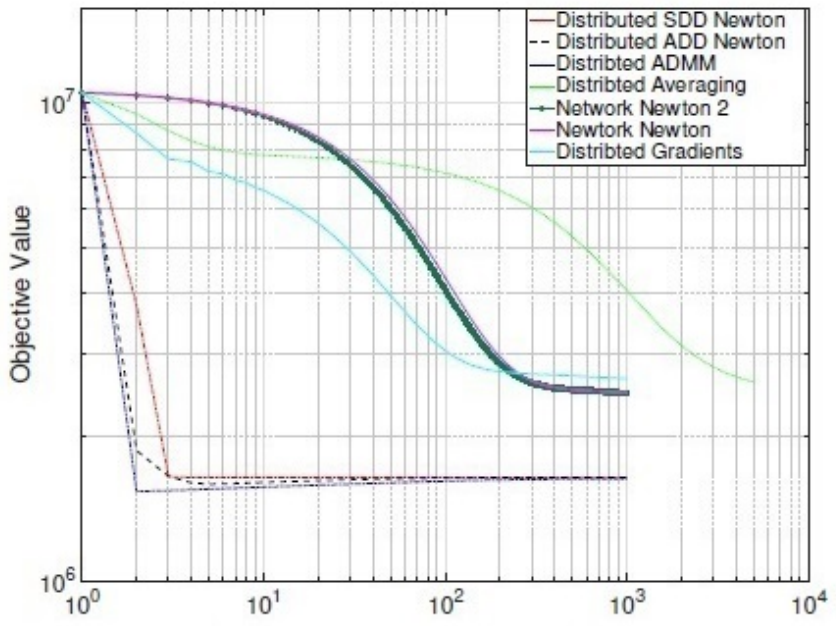


(a) Obj. Synthetic

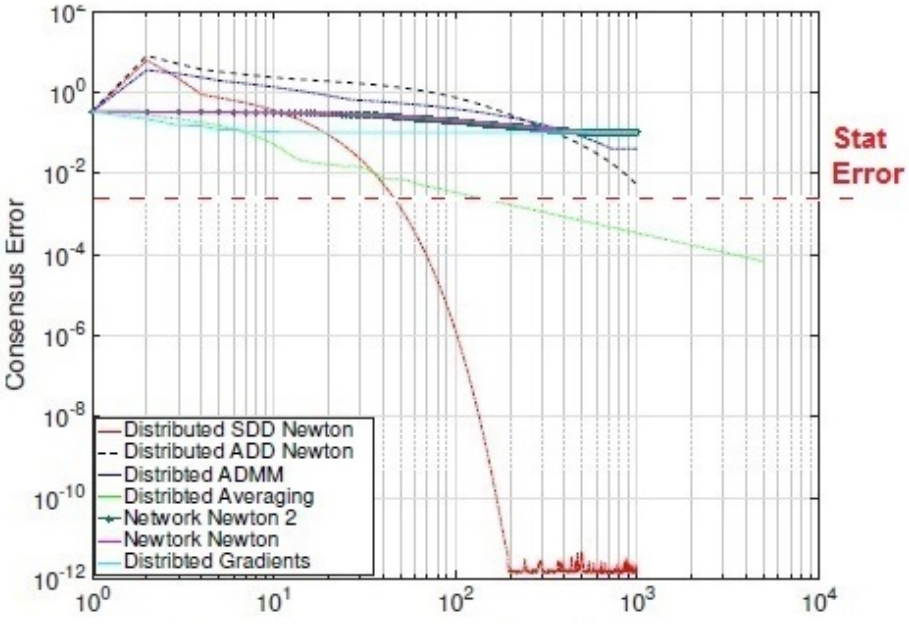


(b) Con. Synthetic

Figure 32: Experimental Results on Linear Regression with a synthetic data set.

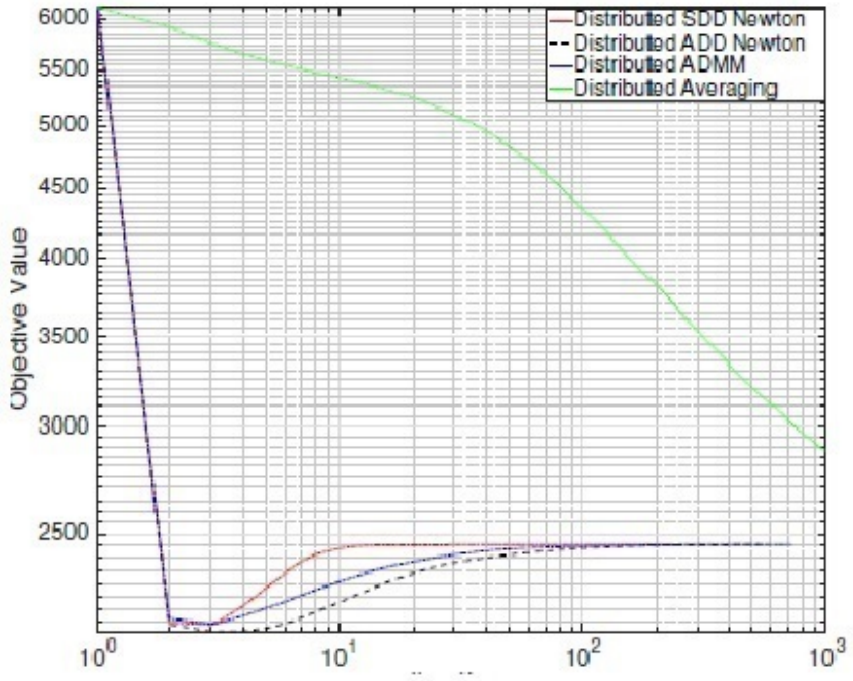


(a) Objective London Schools

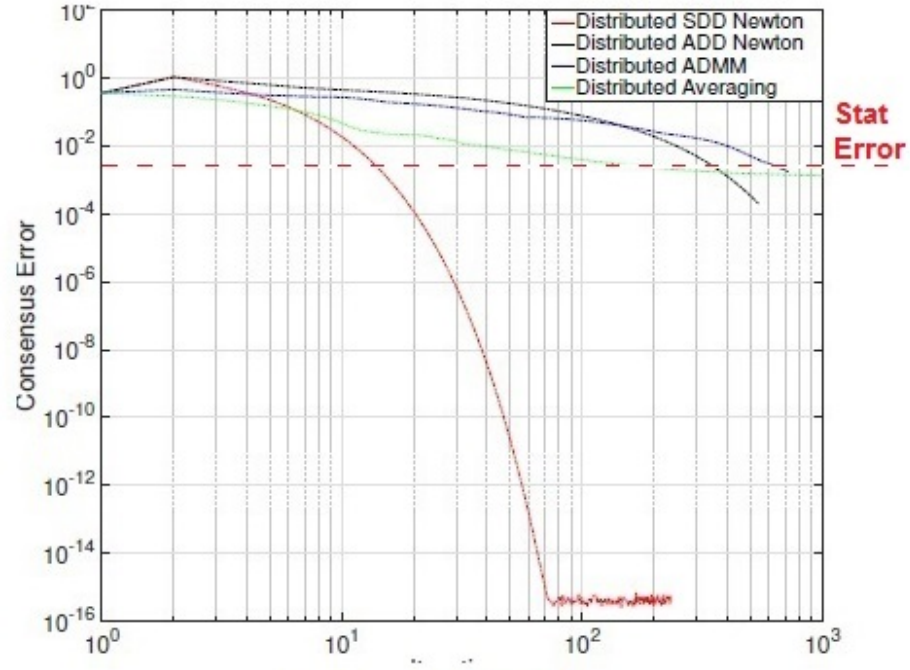


(b) Consensus London Schools

Figure 33: Experimental Results on Linear Regression with a real data set.

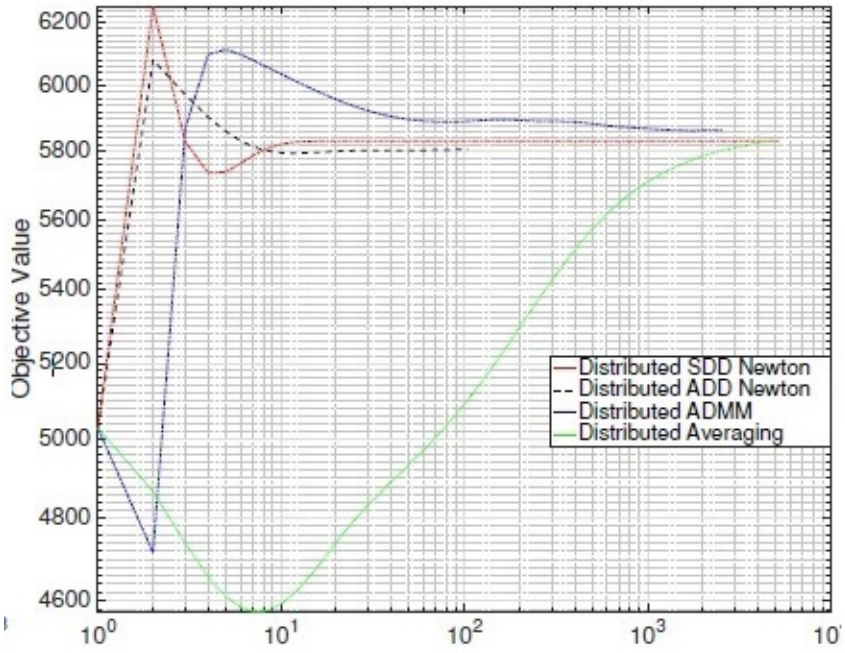


(a) Obj. MNIST L_2

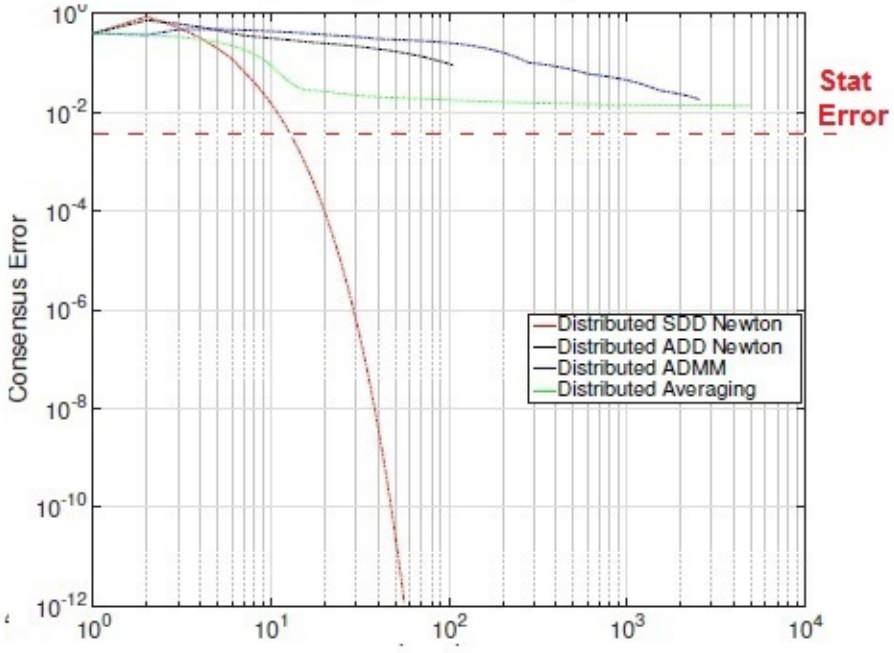


(b) Con. MNIST L_2

Figure 34: Experimental Results on Logistic Regression with L_2 regularization.

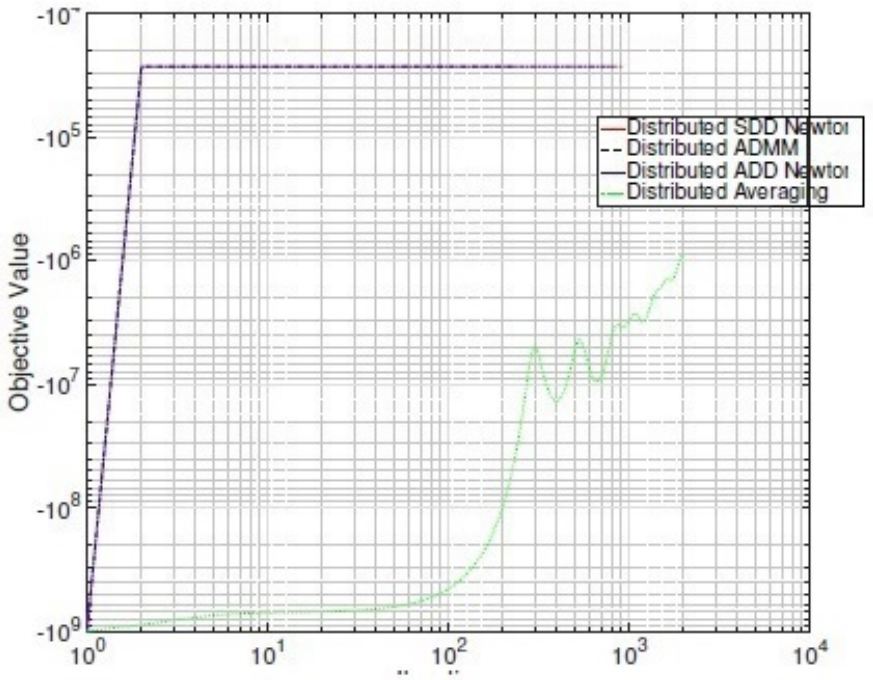


(a) Obj. MNIST L_1

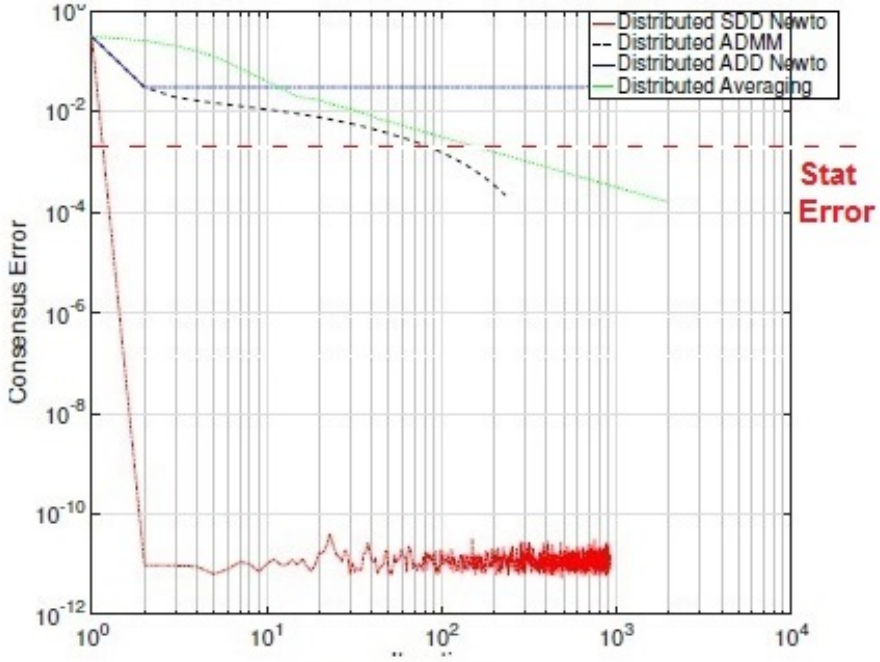


(b) Con. MNIST L_1

Figure 35: Experimental Results on Logistic Regression with L_1 regularization.

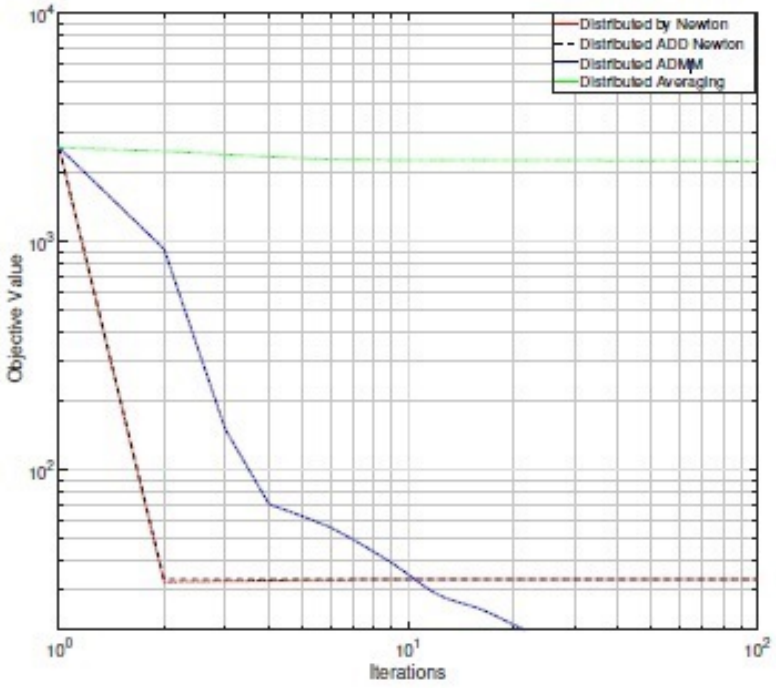


(a) Obj. RL

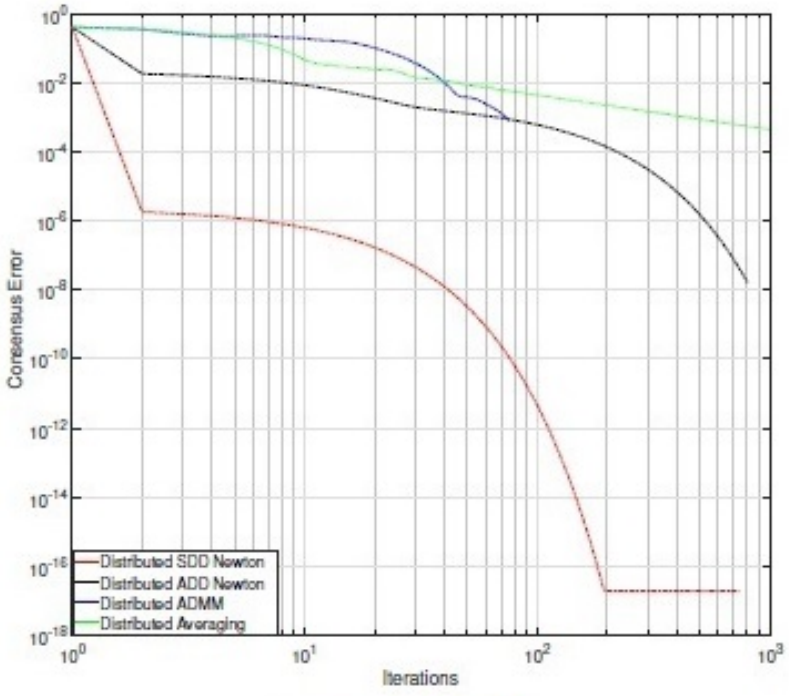


(b) Con. RL

Figure 36: Experimental Results on Reinforcement Learning.

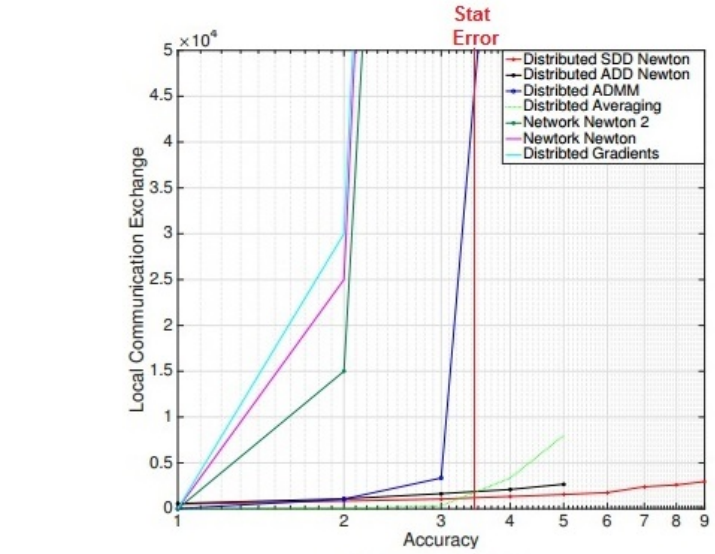


(a) Con. fMRI

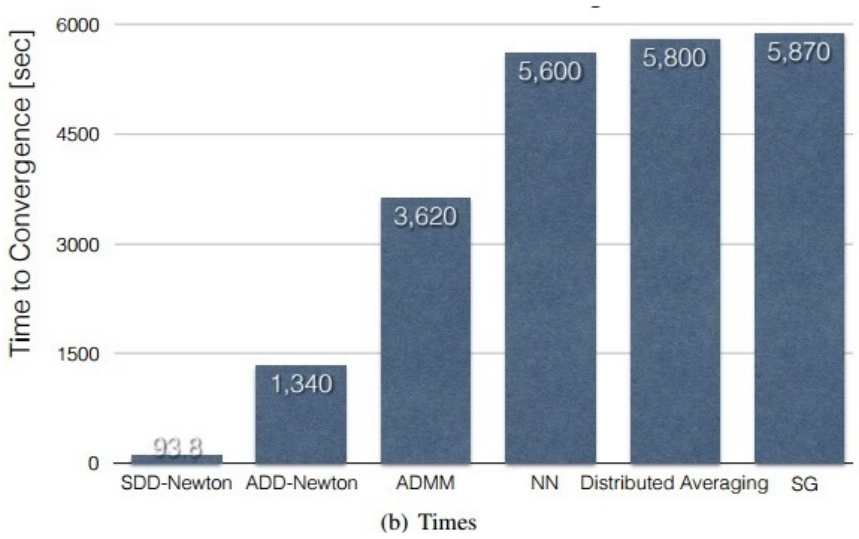


(b) Obj. fMRI

Figure 37: Experimental Results on fMRI images.



(a) Communication



(b) Times

Figure 38: Experimental results: communication overhead, CPU running times

A.22. Computational Graph For Linear Regression

In this paragraph we consider linear regression loss function $f_i(\boldsymbol{x}) = \left(a - \Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}\right)^2 + \mu_i||\boldsymbol{x}||_2^2$ and construct computational graph $\mathcal{G}_{g_i}$ for directional derivative $g_i(\boldsymbol{x}) = \nabla^{\mathsf{T}} f_i(\boldsymbol{x}) \times \boldsymbol{\mathcal{Y}}_i$. Our construction

is based on closed form expression for $g_i(\boldsymbol{x})$:

$$g_i(\boldsymbol{x}) = 2\sum_{j=1}^{p}\left((a - \Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}[\Phi(\boldsymbol{b})]_j) + \mu_i[\boldsymbol{x}]_j\right)[\boldsymbol{y}_j]_i$$

The input nodes $v_1, \ldots, v_p$ correspond to components $[\boldsymbol{x}]_1, \ldots, [\boldsymbol{x}]_p$ and the rest nodes are arranged according to Definition. We start with computational binary tree for scalar product $\Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x} = \sum_{j=1}^{p}[\Phi(\boldsymbol{b})]_j[\boldsymbol{x}]_j$:

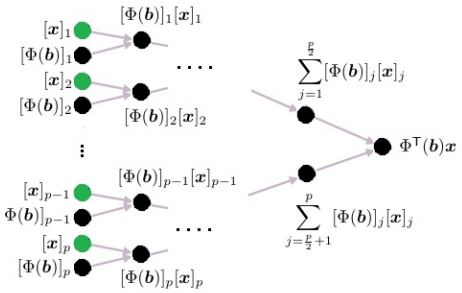


Figure 39: Computational binary tree for $\Phi^{\mathsf{T}}(\boldsymbol{b})\boldsymbol{x}$

Notice, that for any $\boldsymbol{r}, \boldsymbol{s} \in \mathbb{R}^p$ one can construct similar computational tree for computing scalar product $\boldsymbol{r}^{\mathsf{T}}\boldsymbol{s}$. Moreover, the size of such tree is bounded by $\mathcal{O}(p)$. Using such trees, the computational graph of directional derivative $g_i(\boldsymbol{x})$ is given as:

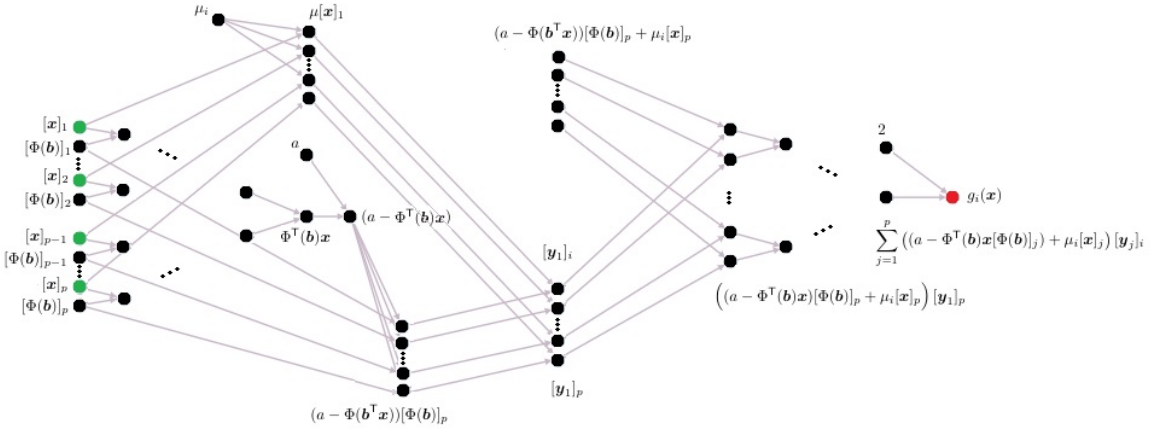


Figure 40: Computational graph $\mathcal{G}_{g_i}$ for $g_i(\boldsymbol{x})$. Input nodes are marked by green. Last node is marked by red.

Easy to see that the size of graph $\mathcal{G}_{g_i}$ is bounded by $\mathcal{O}(p)$.

A.23. Computational Graph For Logistic Regression

In this paragraph we consider linear regression loss function

$$f_i(\boldsymbol{x}) = \left[a \log \frac{1}{1 + e^{-\Phi^\mathsf{T}(\boldsymbol{b})\boldsymbol{x}}} + (1-a) \log \left(1 - \frac{1}{1 + e^{-\Phi^\mathsf{T}(\boldsymbol{b})\boldsymbol{x}}} \right) \right] + \mu_i ||\boldsymbol{x}||_2^2$$

and construct computational graph $\mathcal{G}_{g_i}$ for directional derivative $g_i(\boldsymbol{x}) = \nabla^\mathsf{T} f_i(\boldsymbol{x}) \times \boldsymbol{\mathcal{Y}}_i$. Our construction is based on closed form expression for $g_i(\boldsymbol{x})$:

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{p} \left(2\mu_i [\boldsymbol{x}]_j + (a-1)[\Phi(\boldsymbol{b})]_j + \frac{[\Phi(\boldsymbol{b})]_j}{1 + e^{-\Phi^\mathsf{T}(\boldsymbol{b})\boldsymbol{x}}} \right) [\boldsymbol{y}_j]_i$$

The input nodes $v_1, \ldots, v_p$ correspond to components $[\boldsymbol{x}]_1, \ldots, [\boldsymbol{x}]_p$ and the rest nodes are arranged according to Definition.

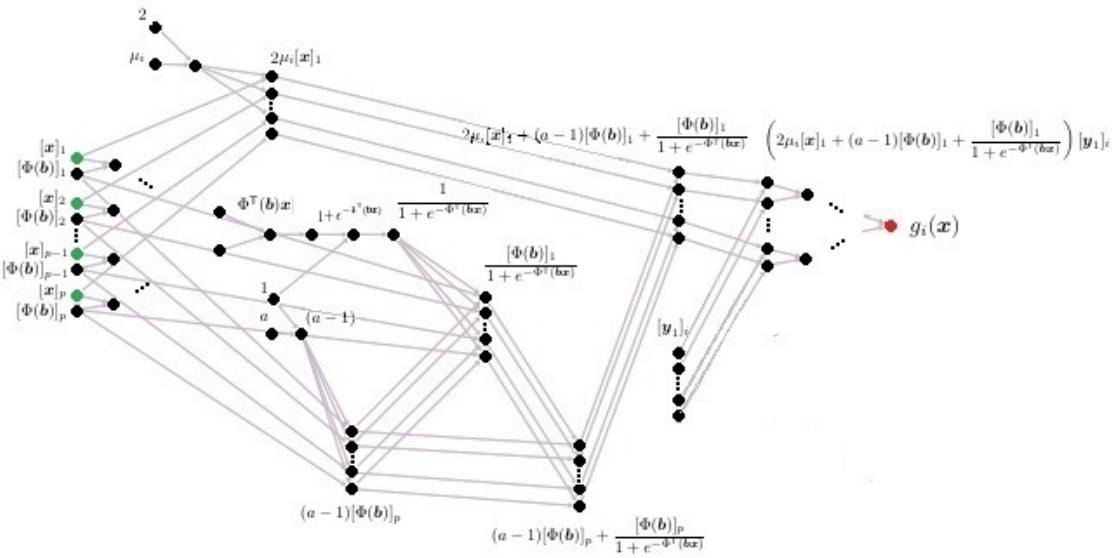


Figure 41: Computational graph $\mathcal{G}_{g_i}$ for $g_i(\boldsymbol{x})$. Input nodes are marked by green. Last node is marked by red.

Similarly to linear regression case, the size of graph $\mathcal{G}_{g_i}$ is bounded by $\mathcal{O}(p)$.

BIBLIOGRAPHY

European Commission: Internet of Things An action plan for Europe. 2009. URL COM(2009) 278,http://eur-lex.europa.eu/LexUriServ/site/en/com/2009/com20090278en01.pdf(2009).

A. Agarwal, P. L. Bartlett, P. Ravikumar, and M. J. Wainwright. Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization. *IEEE Trans. Information Theory*, 58 (5):3235–3249, 2012. URL http://dx.doi.org/10.1109/TIT.2011.2182178.

N. Agarwal, Z. Allen-Zhu, B. Bullins, E. Hazan, and T. Ma. Finding Approximate Local Minima Faster Than Gradient Descent. In *STOC*, 2017.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993a. ISBN 0-13-617549-X.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993b. ISBN 0-13-617549-X.

A. Aly and M. Van Vyve. *Securely Solving Classical Network Flow Problems*, pages 205–221. Springer International Publishing, Cham, 2015. ISBN 978-3-319-15943-0.

M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016a. URL http://arxiv.org/abs/1606.04474.

M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016b. URL http://arxiv.org/abs/1606.04474.

L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15): 2787 – 2805, 2010. ISSN 1389-1286. URL http://www.sciencedirect.com/science/article/pii/S1389128610001568.

O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, NY, USA, 1994a. ISBN 0-521-44524-8.

O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, NY, USA, 1994b. ISBN 0-521-44524-8.

S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30:532–563, 2007. ISSN 1042-9832. doi: 10.1002/rsa.v30:4. URL http://dx.doi.org/10.1002/rsa.v30:4.

A. G. Baydin, B. A. Pearlmutter, and A. A. Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.

D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *CoRR*, abs/1507.01030, 2015. URL http://arxiv.org/abs/1507.01030.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-648700-9.

C. Blair. Problem complexity and method efficiency in optimization (a. s. nemirovsky and d. b. yudin). *SIAM Review*, 27(2):264–265, 1985. URL http://dx.doi.org/10.1137/1027074.

G. E. Blelloch, A. Gupta, I. Koutis, G. L. Miller, R. Peng, and K. Tangwongsan. Near linear-work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *CoRR*, abs/1111.1750, 2011. URL http://arxiv.org/abs/1111.1750.

B. Bollobas and O. Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, Jan. 2004. ISSN 0209-9683.

B. Bollobas, O. Riordan, J. Spencer, and G. Tusndy. The degree sequence of a scale-free random graph process. *RANDOM STRUCTURES AND ALGORITHMS*, 18:279–290, 2001.

E. G. Boman, B. Hendrickson, and S. A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM J. Numerical Analysis*, 46(6):3264–3284, 2008. doi: 10.1137/040611781. URL http://dx.doi.org/10.1137/040611781.

L. Bottou. *Large-Scale Machine Learning with Stochastic Gradient Descent*, pages 177–186. Physica-Verlag HD, Heidelberg, 2010.

H. Bou-Ammar, E. Eaton, J. Luna, and P. Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3345–3351. AAAI Press, 2015.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004a. ISBN 0521833787.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004b.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1): 1–122, Jan. 2011. ISSN 1935-8237. doi: 10.1561/2200000016.

R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016. URL http://dx.doi.org/10.1137/140954362.

W. Casaca. Graph laplacian for spectral clustering and seeded image segmentation. In U. C. S. Pablo, editor, *2015 XLI Latin American Computing Conference (CLEI), Special Edition*, pages 167–186, Arequipa-Peru, October 2015. CLEI, CLEI. ISBN 978-9972-825-91-0. URL http://clei.org/clei2015/144096.

J. A. Cetto, J. Filipe, and J.-L. Ferrier. *Informatics in Control Automation and Robotics: Revised and Selected Papers from the International Conference on Informatics in Control Automation and ...* Springer Publishing Company, Incorporated, 2013. ISBN 3642267416, 9783642267413.

T. Chang, M. Hong, W. Liao, and X. Wang. Asynchronous distributed ADMM for large-scale optimization- part I: algorithm and convergence analysis. *CoRR*, abs/1509.02597, 2015. URL http://arxiv.org/abs/1509.02597.

N. Chatzipanagiotis, D. Dentcheva, and M. M. Zavlanos. An augmented lagrangian method for

distributed optimization. *Mathematical Programming*, 152(1):405–434, 2015. ISSN 1436-4646. doi: 10.1007/s10107-014-0808-7. URL http://dx.doi.org/10.1007/s10107-014-0808-7.

P. Chebyshev. *Theorie des mecanismes connus sous le nom de parallelogrammes*. Number pt. 1. Imprimerie de l'Academie imperiale des sciences, 1853. URL https://books.google.com/books?id=o7AwkgAACAAJ.

C.-T. Chu, S. K. Kim, Y. an Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng. Map-reduce for machine learning on multicore. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, 2007. URL http://papers.nips.cc/paper/3150-map-reduce-for-machine-learning-on-multicore.pdf.

F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

M. B. Cohen. Ramanujan graphs in polynomial time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 276–281. IEEE, 2016.

M. B. Cohen, R. Kyng, G. L. Miller, J. Pachocki, R. Peng, A. Rao, and S. C. Xu.

M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC 14, pages 343–352, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2710-7. doi: 10.1145/2591796.2591833. URL http://doi.acm.org/10.1145/2591796.2591833.

M. Dahan and S. Amin. Security games in network flow problems. *submitted to Math of OR*, 2016. URL http://cps-forces.org/pubs/144.html.

S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. *CoRR*, abs/0803.0988, 2008. URL http://arxiv.org/abs/0803.0988.

D. J. de Solla Price. Networks of scientific papers. *Science*, 149(3683):510–515, 1965. ISSN 0036-8075.

J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1223–1231, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999271.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html.

M. Eisen, A. Mokhtari, and A. Ribeiro. A decentralized quasi-newton method for dual formulations of consensus optimization. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1951–1958, Dec 2016.

P. Erdos and A. Renyi. On random graphs. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

E. Fadel, V. Gungor, L. Nassef, N. Akkari, M. A. Maik, S. Almasri, and I. F. Akyildiz. A survey on wireless sensor networks for smart grid. *Comput. Commun.*, 71(C):22–33, Nov. 2015. ISSN 0140-3664. URL http://dx.doi.org/10.1016/j.comcom.2015.09.006.

D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010. ISBN 0691146675, 9780691146676.

J. L. Goffin. On convergence rates of subgradient optimization methods. *Mathematical Programming*, 13(1):329–347, 1977. ISSN 1436-4646. doi: 10.1007/BF01584346.

J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, Sept. 2013. ISSN 0167-739X. URL http://dx.doi.org/10.1016/j.future.2013.01.010.

B. Gurakan, O. Ozel, and S. Ulukus. Optimal energy and data routing in networks with energy cooperation. *CoRR*, abs/1509.05395, 2015. URL http://arxiv.org/abs/1509.05395.

M. Gurbuzbalaban, A. Ozdaglar, and P. Parrilo. A globally convergent incremental newton method. *Mathematical Programming*, 151(1):283–313, 2015. ISSN 1436-4646. URL http://dx.doi.org/10.1007/s10107-015-0897-y.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

R. V. D. Hofstad. Random graphs and complex networks. In *In preparation*, 2008.

S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *BULL. AMER. MATH. SOC.*, 43(4):439–561, 2006.

A. Jadbabaie, A. Ozdaglar, and M. Zargham. A distributed newton method for network optimization. In *Proceedings of IEEE CDC*, 2009.

M. Jeffery. *Data-Driven Marketing: The 15 Metrics Everyone in Marketing Should Know*. Wiley, 2010. ISBN 9780470595688. URL https://books.google.com/books?id=2mr3WoTiqRgC.

A. Joshi. Topics in optimization and sparse linear systems. Technical report, Champaign, IL, USA, 1996.

E. F. Kaasschieter. A general finite element preconditioning for the conjugate gradient method. *BIT Numerical Mathematics*, 29(4):824–849, 1989. ISSN 1572-9125. doi: 10.1007/BF01932748. URL http://dx.doi.org/10.1007/BF01932748.

M. Kadkhodaie, K. Christakopoulou, M. Sanjabi, and A. Banerjee. Accelerated alternating direction method of multipliers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 497–506, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783400.

J. A. Kelner and A. Madry. Faster generation of random spanning trees. *CoRR*, abs/0908.1448, 2009. URL http://arxiv.org/abs/0908.1448.

J. A. Kelner, L. Orecchia, A. Sidford, and Z. Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. *CoRR*, abs/1301.6628, 2013. URL http://arxiv.org/abs/1301.6628.

J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016a. URL http://arxiv.org/abs/1610.02527.

J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016b. URL http://arxiv.org/abs/1610.05492.

I. Koutis. A simple parallel algorithm for spectral sparsification. *CoRR*, abs/1402.3851, 2014. URL http://arxiv.org/abs/1402.3851.

I. Koutis and G. L. Miller. A linear work, o(n1/6) time, parallel algorithm for solving planar laplacians. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1002–1011, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL http://dl.acm.org/citation.cfm?id=1283383.1283491.

I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD systems. *CoRR*, abs/1003.2958, 2010. URL http://arxiv.org/abs/1003.2958.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

A. Kumar and H. Daume. Learning task grouping and overlap in multi-task learning. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1383–1390, New York, NY, USA, 2012. ACM. URL http://icml.cc/2012/papers/690.pdf.

E. L. Lawler, M. G. Luby, and V. V. Vazirani. Scheduling open shops with parallel machines. *Oper. Res. Lett.*, 1(4):161–164, Sept. 1982. ISSN 0167-6377. doi: 10.1016/0167-6377(82)90021-9. URL http://dx.doi.org/10.1016/0167-6377(82)90021-9.

Y. Lecun and C. Cortes. The MNIST database of handwritten digits.

C. E. Lee, A. E. Ozdaglar, and D. Shah. Solving systems of linear equations: Locally and asynchronously. *CoRR*, abs/1411.2647, 2014. URL http://arxiv.org/abs/1411.2647.

R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2007.

A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. ISSN 0209-9683. doi: 10.1007/BF02126799.

A. Madry. Navigating central path with electrical flows: from flows to matchings, and back. *CoRR*, abs/1307.2205, 2013. URL http://arxiv.org/abs/1307.2205.

G. A. Margulis. Explicit Group-Theoretical Constructions of Combinatorial Schemes and Their Application to the Design of Expanders and Concentrators. *Probl. Peredachi Inf.*, 24(1):51–60, 1988.

F. Mattern and C. Floerkemeier. From active data management to event-based systems and more. chapter From the Internet of Computers to the Internet of Things, pages 242–259. Springer-Verlag, 2010. URL http://dl.acm.org/citation.cfm?id=1985625.1985645.

H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016. URL `http://arxiv.org/abs/1602.05629`.

C. C. Moallemi and B. Van Roy. Convergence of min-sum message passing for quadratic optimization. *IEEE Transactions on Information Theory*, 55(5), 2009.

C. C. Moallemi and B. Van Roy. Convergence of min-sum message-passing for convex optimization. *IEEE Transactions on Information Theory*, 56(4), 2010.

B. Mohar. The laplacian spectrum of graphs. In *Graph Theory, Combinatorics, and Applications*, pages 871–898. Wiley, 1991.

A. Mokhtari, Q. Ling, and A. Ribeiro. Network Newton-Part I: Algorithm and Convergence. *ArXiv e-prints*, Apr. 2015.

M. Morgenstern. Existence and explicit constructions of q + 1 regular ramanujan graphs for every prime power q. *J. Comb. Theory Ser. B*, 62(1):44–62, Sept. 1994. ISSN 0095-8956. doi: 10.1006/jctb.1994.1054. URL `http://dx.doi.org/10.1006/jctb.1994.1054`.

D. Mosk-Aoyama, T. Roughgarden, and D. Shah. *Fully Distributed Algorithms for Convex Optimization Problems*, pages 492–493. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75142-7.

S. Mou, J. Liu, and A. S. Morse. A distributed algorithm for solving a linear algebraic equation. *CoRR*, abs/1503.00808, 2015. URL `http://arxiv.org/abs/1503.00808`.

A. Nedić and A. Ozdaglar. Distributed subradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control, forthcoming*, 2008.

A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, Jan 2009.

A. Nedic, D. Bertsekas, and V. Borkar. Distributed asynchronous incremental subgradient methods. *Studies in Computational Mathematics*, 8(C):381–407, 2001. ISSN 1570-579X.

R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan. A general analysis of the convergence of admm. In D. Blei and F. Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 343–352. JMLR Workshop and Conference Proceedings, 2015. URL `http://jmlr.org/proceedings/papers/v37/nishihara15.pdf`.

J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Berlin, 2006. ISBN 978-0387-30303-1. URL `http://opac.inria.fr/record=b1120179`. NEOS guide http://www-fp.mcs.anl.gov/otc/Guide/.

A. Northup. A study of semiregular graphs.

A. Olshevsky. Linear Time Average Consensus on Fixed Graphs and Implications for Decentralized Optimization and Multi-Agent Control. *ArXiv e-prints*, Nov. 2014.

D. Partynski and S. G. M. Koo. Integration of smart sensor networks into internet of things: Challenges and applications. In *Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, GREENCOM-ITHINGS-CPSCOM '13, pages 1162–1167, Washington, DC,

USA, 2013. IEEE Computer Society. ISBN 978-0-7695-5046-6. URL http://dx.doi.org/10.1109/GreenCom-iThings-CPSCom.2013.202.

R. Peng and D. A. Spielman. An efficient parallel solver for SDD linear systems. *CoRR*, abs/1311.3286, 2013. URL http://arxiv.org/abs/1311.3286.

M. J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003. ISBN 0071232656.

M. Raynal. *Leader Election Algorithms*, pages 77–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38123-2.

P. Rebeschini and S. Tatikonda. A new approach to Laplacian solvers and flow problems. *ArXiv e-prints*, Nov. 2016.

P. Richtárik and M. Takác. Distributed coordinate descent method for learning with big data. *CoRR*, abs/1310.2059, 2013. URL http://arxiv.org/abs/1310.2059.

S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

K. Scheinberg and X. Tang. Complexity of inexact proximal newton methods. *CoRR*, abs/1311.6547, 2013. URL http://arxiv.org/abs/1311.6547.

S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Stochastic convex optimization. In *COLT*, 2009.

G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 3–16. IEEE, 2000.

Sherman and W. Morison. Abstracts of papers. *Ann. Math. Statist.*, 20(4):620–624, 12 1949.

V. Singh, K. P. Miyapuram, and R. S. Bapi. Detection of cognitive states from fMRI data using machine learning techniques. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 587–592. Morgan Kaufmann Publishers Inc., 2007.

D. A. Spielman and S. Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006. URL http://arxiv.org/abs/cs/0607105.

D. A. Spielman and S. Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008. URL http://arxiv.org/abs/0808.4134.

M. Spiliopoulou, L. Schmidt-Thieme, and R. Janning, editors. *Data analysis, machine learning and knowledge discovery*. Studies in Classification, Data Analysis, and Knowledge Organization. Springer International Publishing, 2014. ISBN 9783319015958 3319015958.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

I. Trofimov and A. Genkin. *Distributed Coordinate Descent for L1-regularized Logistic Regression*, pages 243–254. Springer International Publishing, Cham, 2015.

R. Tutunov, H. Bou-Ammar, and A. Jadbabaie. A distributed newton method for large scale consensus optimization. *CoRR*, abs/1606.06593, 2016. URL `http://arxiv.org/abs/1606.06593`.

O. Vermesan and P. Friess, editors. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communication. River, Aalborg. ISBN 978-87-92982-73-5. URL `http://www.internet-of-things-year=2013`.

E. Wei and A. E. Ozdaglar. Distributed alternating direction method of multipliers. In *CDC*, pages 5445–5450. IEEE, 2012. ISBN 978-1-4673-2065-8.

E. Wei, A. Ozdaglar, and A. Jadbabaie. A distributed newton method for network utility maximization. *LIDS Technical Report 2832*, 2010.

R. Williams. Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In *IN SODA*, pages 1–11. ACM Press, 2007.

L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014. ISSN 1551-3203.

D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014. ISBN 1447157788, 9781447157786.

M. Zargham, A. Ribeiro, and A. Jadbabaie. Accelerated dual descent for constrained convex network flow optimization. In *52nd IEEE Conference on Decision and Control*, pages 1037–1042, Dec 2013. doi: 10.1109/CDC.2013.6760019.

X. Zhang. *Empirical Risk Minimization*, pages 312–312. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8.

D. Zhou and B. Schlkopf. A regularization framework for learning from graph data. In *ICML Workshop on Statistical Relational Learning*, pages 132–137, 2004.

X. Zhu, J. Lafferty, and Z. Ghahramani. Semi-supervised learning: From gaussian fields to gaussian processes. Technical report, School of CS, CMU, 2003.

B. Zupan, E. T. Keravnou, and N. Lavrac. *Intelligent Data Analysis in Medicine and Pharmacology*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. ISBN 0792380002.