MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

ANALYSIS OF A DISTRIBUTED ALGORITHM FOR SOLVING LINEAR EQUATIONS

OSMAN MECNUN DURU

MASTER THESIS

Department of Electrical and Electronics Engineering

Thesis Supervisor

Asst. Prof. Dr. Onur CİHAN

ISTANBUL, 2019

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

ANALYSIS OF A DISTRIBUTED ALGORITHM FOR SOLVING LINEAR EQUATIONS

OSMAN MECNUN DURU

(525014003)

MASTER THESIS

Department of Electrical and Electronics Engineering

Thesis Supervisor

Asst. Prof. Dr. Onur CİHAN

ISTANBUL, 2019

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Osman Mecnun DURU, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled "**Analysis of a Distributed Algorithm for Solving Linear Equations**", on 07.08.2019 and has been found to be satisfactory by the jury members.

Jury Members		**Signature**
Asst. Prof. Dr. Onur CİHAN Marmara University	(Advisor)	
Asst. Prof. Dr. Özlem Feyza ERKAN Beykoz University	(Jury Member)	
Asst. Prof. Dr. Salih BAYAR Marmara University	(Jury Member)	

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Osman Mecnun DURU be granted the degree of Master of Science in Department of Electrical and Electronics Engineering, Electrical and Electronics Engineering Program on 21.08.2019. (Resolution no: 2019/1708)

Director of the Institute

Prof. Dr. Bülent EKİCİ
Enstitü Müdürü

ACKNOWLEDGEMENT

First of all, I would like to thank my thesis supervisor Asst. Prof. Dr. Onur Cihan for his support and guidance and for sharing his precious time, knowledge and experience with me.

I also want to thank my colleagues from Yeni Yuzyil University, for their help and support when I was studying on my thesis.

I am also grateful to my beloved family Kamil Duru, Belgin Duru and Oğuzhan Duru, and my dear girlfriend Cansu Bulduk for their endless support throughout my education life.

August 2019 **Osman Mecnun Duru**

CONTENTS

ABSTRACT

ANALYSIS OF A DISTRIBUTED ALGORITHM FOR SOLVING LINEAR EQUATIONS

Solving a linear equation system is one of the most known and important problems in science and engineering. Such systems have numerous applications such as forecasting, estimation and linear approaches to nonlinear equations. For the solution of these systems, several distributed algorithms have been proposed in the literature, which divide large linear equation systems into small pieces and enable them to be solved faster and more accurately than solving it by a centralized processor. The basic idea of these algorithms is to consider the system as a multi-agent network structure where the autonomous agents are physically separated from each other and communicate with their neighbors nearby. The network consists of multiple agents, each agent knows only a subset of the equation and therefore none can solve the overall equation individually. In order to overcome this issue, all agents must work cooperatively in a distributed way. Each agent i has a solution vector $x_i(t)$ that takes values in R^n, and it is assumed that the only information that the agent receives from its neighbor j is its state vector. The aim of the thesis is to upgrade a well-known distributed algorithm in [1] for solving a linear algebraic equation and investigate its convergence rate. In this thesis, we consider a distributed algorithm and examine its convergence rate in networks with and without transmission delays. A new rapid method is proposed for both cases. Convergence rates of a well-known algorithm in the literature and the one proposed in this thesis are compared for un-delayed and delayed networks. Furthermore, the effect of weighting strategies on the convergence rate of the algorithm is investigated for several important network topologies as well and it is shown that for all topologies under consideration, the random walk method yields the fastest convergence for the example equation system.

ÖZETÇE

DOĞRUSAL DENKLEMLERİN ÇÖZÜMÜ İÇİN KULLANILAN DAĞITIK BİR ALGORİTMANIN ANALİZİ

Doğrusal bir denklem sistemini çözmek, bilim ve mühendislikteki en bilinen ve en önemli sorunlardan biridir. Bu tarz sistemlerin, öngörme, kestirim ve doğrusal olmayan denklemlerin doğrusal yaklaşımları gibi sayısız uygulamaları vardır. Bu sistemlerin çözümü için literatürde, büyük doğrusal denklem sistemlerini küçük parçalara bölen ve merkezi bir işlemci ile çözmekten daha hızlı ve daha doğru bir şekilde çözülmelerini sağlayan çeşitli dağıtık algoritmalar önerilmiştir. Bu algoritmaların temel fikri, sistemi, özerk etmenlerin birbirlerinden fiziksel olarak ayrıldığı ve yakındaki komşularıyla iletişim kurduğu çok etmenli bir ağ yapısı olarak görmektir. Ağ, her bir etmenin sadece denklemin bir alt kümesini bildiği birden fazla etmenden oluşur ve bu nedenle hiçbiri toplam denklemi ayrı ayrı çözemez.. Bu sorunun üstesinden gelmek için, tüm etmenler dağıtık bir şekilde işbirliği içinde çalışmalıdır. Her i etmeninin R^n de değerler alan bir çözüm vektörü $x_i(t)$ vardır ve etmen i nin komşusu j den aldığı tek bilginin, onun durum vektörü olduğu varsayılmaktadır. Bu tezin amacı, doğrusal cebirsel denklem kümesini çözmek için [1]'de önerilen dağıtık algoritmayı geliştirmek ve yakınsama oranını araştırmaktır. Bu tezde, dağıtık bir algoritma ele alınmış ve iletim gecikmeleri olan ve olmayan ağlardaki yakınsaklık oranları incelenmiştir. Her iki durum için yeni ve hızlı bir yöntem önerilmiştir. Literatürdeki iyi bilinen bir algoritmanın yakınsama oranı ve bu tezde önerilen algoritmanın yakınsama oranı gecikmesiz ve gecikmeli ağlar için karşılaştırılmıştır. Ayrıca, ağırlıklandırma yöntemlerinin algoritmanın yakınsama oranı üzerindeki etkisi de birkaç önemli ağ topolojisi için incelenmiştir ve ele alınan tüm topolojiler için, rastgele yürüyüş yönteminin, örnek denklem sistemi için en hızlı yakınsamayı sağladığı gösterilmiştir.

SYMBOLS

$\boldsymbol{A}$: Coefficients matrix

$\boldsymbol{A_i}$: Matrix of equations that agent i knows

$\boldsymbol{Adj}$: Adjacency matrix

$\boldsymbol{b}$: Constant term of an equation system

$\boldsymbol{d_i}$: Numbers of neighbors of agent i

ε : Edge set of a graph

$\boldsymbol{G}$: Graph

$\boldsymbol{N_i}$: Neighbor set of agent i

$\boldsymbol{P_i}$: Projection matrix to A_i

$\boldsymbol{R^n}$: Set of n vectors with real numbers

$\boldsymbol{\tau_{ij}}$: Time delay between agents *i* and *j*

$\boldsymbol{V}$: Vertex set of a graph

W : Electrical power unit (Watt)

$\boldsymbol{x_i(k)}$: Solution estimate of agent i at time step k

$\boldsymbol{x_{ls}}$: Least squares solution

$\boldsymbol{x_{mn}}$: Minimum norm solution

$\boldsymbol{x^*}$: Unique solution

$\boldsymbol{w_{ij}}$: Weighting coefficients that agent i uses to update its solution estimates

ABBREVIATIONS

DN **:** Degree neighbor method

EW **:** Equal weighting method

GEPP **:** Gauss elimination method with partial pivoting

IEEE **:** Institute of Electrical and Electronics Engineers

M-H **:** Metropolis-Hastings method

RW **:** Random walk method

SOR **:** Successive over-relaxation method

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Solving a set of linear algebraic equations is one of the most common problems in numerical computation since many real-world mathematical problems including forecasting, estimation and linear approaches to nonlinear equations can be modeled as linear equations [2]. There has been a growing amount of attention to solving linear equation problems and numerous methods have been developed so far. According to the dimensions of the system of interest, the solution ways also vary. Centralized methods are effective and reliable in reaching the solution where the whole equation system is known [3]. However, in some practical applications, distributed methods are needed because of the scale of the system and the physical distance of the agents that make up the system. In addition, distributed algorithms are more advantageous in terms of both efficiency and security, since more than one computer can be used in a network to solve the same large equation system together, and during the calculation, no information about the original equation system needs to be shared over the network [4].

In the following sections, the literature on centralized and distributed methods for solving linear equations are given respectively.

1.1. Literature Review and Related Works

1.1.1. Centralized Methods for Solving Linear Equations

Given a full rank $n \times n$ matrix A, the main purpose of direct methods is to obtain the unique solution $x = A^{-1}b$ by computing the inverse of A. These methods are very robust and require the knowledge of the all components of A [5]. The inverse can be obtained by the direct methods such as *LU* or *QR* decompositions. The method chosen will most likely be the Gauss elimination method with partial pivoting (GEPP). However, due to their enormous need of time and processing memory during calculations, and for the cases when A is sparse, direct methods may be impractical. In order to overcome these disadvantages, various iterative methods have been proposed to calculate the inverse of matrices [6-9].

In the cases mentioned above, it would be more preferable to propose a method for finding an approximate a solution rather than the exact solution. Five iterative methods,

which start from an initial guess $x(0)$ and produce approximations $x(1), x(2), \ldots$ in order to find the approximate solution of $Ax = b$ will be given.

1.1.1.1. Jacobi Method

The Jacobi Method finds the solution of $Ax = b$ under the condition that the diagonal elements of the square matrix A are non-zero. Let $D = diag\{A\}$ be a diagonal matrix which consists of the diagonal elements of A and $R = A - D$ be a matrix with zero diagonals. Then one can rewrite $Ax = b$ as [10],

$$(D + R)x = b\ . \tag{1.1}$$

The solution vector can be computed as,

$$x = D^{-1}(-Rx + b). \tag{1.2}$$

Thus, the estimate of *x* can be updated with the Jacobi algorithm below,

$$x(k + 1) = -D^{-1}Rx(k) + D^{-1}b. \tag{1.3}$$

where $x(k)$ is the estimate at time step k and, the expression used to update each element of x is given as;

$$x_i(k + 1) = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j(k)\right) \tag{1.4}$$

For a system of linear equations where A and $2D - A$ are positive definite, the Jacobi method converges to $A^{-1}b$ [10].

1.1.1.2. Gauss-Seidel Method

As in Jacobi method, Gauss-Seidel Method requires the diagonal elements of the matrix A are non-zero. Difference between these methods is that Gauss-Seidel method divides the A matrix into two parts L and U where L consists of the lower triangular elements of A and U consists of the strictly upper triangular elements of A. Therefore we have $A = L + U$. Then $Ax = b$ can be rewritten as [10],

$$(L + U)x = b \ , \tag{1.5}$$

and consequently

$$x = L^{-1}(-Ux + b) \ . \tag{1.6}$$

Thus, the estimate of *x* can be updated with the algorithm below,

$$x(k + 1) = L^{-1}(b - U(x(k)) \ , \tag{1.7}$$

here, the expression used to update each element of *x* is given as follows;

$$x_i(k + 1) = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j(k + 1) - \sum_{j=i+1}^{n} a_{ij}x_j(k)\right) \tag{1.8}$$

For a system of linear equations of where A is positive definite, the Gauss-Seidel method converges to $A^{-1}b$.

1.1.1.3. Successive Over-Relaxation Method

In Successive Over-Relaxation(SOR) Method, A is decomposed into the sum of D, L and U matrices where D is a diagonal matrix which has diagonal elements of A, L is a strictly lower triangular matrix and U is strictly upper triangular matrix. For an arbitrarily chosen *w*, one can rewrite $Ax = b$ as [10],

$$(D + wL)x = wb - (wU + (w - 1)D)x. \tag{1.9}$$

Thus, the estimate of x can be updated with the algorithm below,

$$x(k + 1) = (D + wL)^{-1}(wb - (wU + (w - 1)D)x(k)), \tag{1.10}$$

and the expression used to update each element of x is given as follows;

$$x_i(k+1) = (1-w)x_i(k) + \frac{w}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j(k+1) - \sum_{j>i} a_{ij}x_j(k)\right) \quad (1.11)$$

1.1.1.4. Kacmarz Method

In the Kacmarz method, the algorithm below is used to solve $Ax = b$ [10],

$$x(k+1) = x(k) + \frac{b_i - A_i x(k)}{\|A_i\|_2^2} A_i^T \quad (1.12)$$

where A_i is *i*th the row of A where i = *k* mod *m*. At each iteration step, one single row of the matrix A is used at the updating procedure of the estimate of x. For a general matrix A, the Kacmarz method converges to $A^{-1}b$.

1.1.1.5. Conjugate Gradient Method

For a symmetric and positive definite matrix A, conjugate gradient method can be used to solve $Ax = b$. A set of conjugate vectors are identified and the solution of $Ax = b$ is expressed as a linear combination of these vectors. Thus, the estimate of x is updated with the algorithm below [10],

$$x(k+1) = x(k) + \lambda(k)p(k), \quad (1.13)$$

with,

$$\lambda(k) = \frac{r(k)^T p(k)}{p(k)^T A p(k)} \quad (1.14)$$

$$r(k+1) = r(k) - \lambda(k)Ap(k) \quad (1.15)$$

$$p(k+1) = r(k+1) - \frac{r(k+1)^T Ap(k)}{p(k)^T Ap(k)} p(k) \quad (1.16)$$

where $r(0) = b - Ax(0)$ and $p(0) = r(0)$ with an arbitrary $x(0)$.

1.1.2. Distributed Algorithms for Solving Linear Equations

In order to solve linear equation systems, researchers have developed various distributed algorithms, which divide large linear equation systems into small pieces and enable them to be solved faster and more accurately than the total equation set.

1.1.2.1. Continuous Time Distributed Algorithms for Solving Linear Equations

In this section, we overview the literature on continuous time distributed algorithms which can be used to solve linear equations over networks.

In [11], authors proposed a continuous-time distributed algorithm which allows agents in an undirected, connected graph to solve a linear equation set cooperatively, where each equation is known to at least one agent. In the mentioned study, the network is assumed to have a fixed topology. The algorithm in [11] enables the agents to asymptotically converge to a solution when there are infinitely many solutions, or there is a unique solution. Furthermore, when there is no solution, it discovers there is none. In addition, the authors show that the algorithm is exponentially convergent and develop a lower bound on its convergence rate.

The algorithm proposed in [12], can be used to update estimates of agents for a linear equation set which is defined on a fixed, undirected, connected and arbitrary graph where differential equation update laws are used to ensure that all agents' states converge exponentially fast to the solution of the equation set. In this study, each distinct row of partitioned matrix $[A\ b]$ is known to only one agent and it is shown that when the aforementioned graph structure is used, exponential convergence is reached in accordance with differential geometry and linear control systems.

Another distributed algorithm for solving linear equations in continuous time is proposed in [13]. In the mentioned study, agents know only a subset of the partitioned matrix $[A\ b]$ and, for time dependent directed graph $G(t)$ and any sequence of repeatedly jointly strongly connected graphs, the agents using the algorithm are able solve the equation set asymptotically when $Ax = b$ has at least one solution. When there is no solution, limiting behavior of the system is also examined and it is shown that the agents can recognize the no solution case in a distributed way.

In [14] and [15], the authors proposed two different flows as solvers for continuous time linear algebraic equations in the form of $x = b$: "consensus + projection" flow and "projection consensus flow". Under the condition of smooth graph properties, it is shown that all agent states converge to same solution if there is at least one. When there is no exact solution, the least squares approximate solution is reached for fixed and undirected graphs.

In [16] two distributed algorithms are proposed to obtain least squares solution for both continuous-time and discrete-time cases. Each agent knows one of the linear equations and has a dynamic state. The agents aim to reach a consensus on the least squares approximate solution by sharing their state vectors over a network. Proposed algorithm is examined for different types of graphs for continuous-time networks. Under sufficient conditions, the algorithm has exponential convergence rate for both continuous-time and discrete-time.

A different approach from previous studies is discussed in [17]. In contrast to other studies in the literature, it is assumed that agents know the columns, not the rows of the matrix A. In order to solve the system of linear equations, the problem is transformed into an optimization problem with a linear constraint. Then, a continuous-time distributed algorithm based on the Lagrangian function of the optimization problem is designed. Another contribution of this study is to prove the convergence of the algorithm to a specific point by using a Lyapunov method and saddle point dynamics. The performance of the proposed algorithm is verified by numerical simulations.

1.1.2.2. Discrete Time Distributed Algorithms for Solving Linear Equations

In the past decade, several algorithms have been proposed for solving linear algebraic equations in a discrete-time setting. Main idea behind these algorithms is the same with the ones in continuous time, i.e., dividing larger linear equation sets into smaller ones and solving them with distributed computation.

In [1], a distributed algorithm is proposed for solving $Ax = b$, where A is an $n \times n$ nonsingular matrix and b is an n-vector. With the assumption that every agent i knows a distinct row of $[A\ b]$ and estimates of their neighbors, it is shown that the equation can be solved. Each agent starts with a solution to the equation known by itself

and updates its solution of $Ax = b$ at each time step by using the solutions they get from neighbor agents. The network is characterized with an undirected connected graph G and it is shown that the proposed algorithm makes all agents' estimates to converge exponentially fast to the intended solution $A^{-1}b$, for any nonsingular matrix A and any connected graph G.

In [18], a distributed algorithm is proposed, with assumption that the equation has at least one solution. Each agent has information about a subset of the rows of $[A\ b]$, and the data coming from their neighbors. Relations between neighbors are given by a time dependent directed graph $N(k)$ and it is assumed that the agent i is always considered to be a neighbor of itself. The estimates of the agents using this algorithm converge exponentially fast to the desired solution, provided that A has a solution and the graph is repeatedly jointly strongly connected. It is also shown that the algorithm can be used to solve the equation set when A and b are varying with time. It is shown that, exponential convergence can still be achieved when update instants of agents are not synchronized. For the case that $Ax = b$ has no solution, the algorithm can be used to compute the least squares solution with a slight modification.

Another algorithm with asynchronous update is proposed in [18], where each agent knows a subset of the rows of $[A\ b]$, and the network is characterized with a repeatedly jointly strongly connected and time dependent graph. There are different event times for every agent and each agent is able to communicate with their neighbors with time delays. Between two event times, each agent keeps their solution estimates constant. Under these conditions, it is shown that, the proposed algorithm provides exponential convergence.

In [19], a distributed algorithm is proposed for a linear algebraic equation $Ax = b$ with a unique solution x^*. In this algorithm, it is not essential for an agent i to calculate all of the elements of x^*, which may also be a result of communication restrictions. The main advantage of this algorithm is that each agent controls a state vector $x_i(k)$ of length less than n. It is shown that, the algorithm allows agents to iteratively update their states and finally achieve exponential convergence to the desired part of x^*.

In [20], authors have proved that the algorithm proposed in [18] is able to achieve desired solution and some further improvements are proposed. The authors have also

stated that the algorithm in [18] has some deficiencies. For the cases where the equation set has infinitely many solutions, agents' estimates converge to a solution, but the achieved solution is not clear. To eliminate this problem, they have proposed a special initialization procedure which ensures convergence to the solution with minimum Euclidian distance to a chosen vector in R^n. The algorithm in [18] depends on the idea that each agent starts updating an initial solution $x_i(0)$ that satisfies $Ax_i = b$, and the authors have also shown that with a slight modification the algorithm is able to achieve exponential convergence to a desired solution for an arbitrarily chosen initial state $x_i(0)$.

In [21], the effect of network topology on the convergence rate of the algorithm is investigated. It is shown with both numerical and analytical results that networks with smaller diameter, homogenous degree distribution and higher mean degree are able to achieve exponential convergence. For two graphs G_1 and G_2 with same length, when G_1 has a more homogeneous degree distribution than G_2 , it is easier and takes less time to visit lower degree agents from higher degree agents which yields the algorithm converges faster to the desired solution. Adding new agents to the network results in a degree distribution that is more homogenous and increases mean degree of the network, which consequently affects the performance of the algorithm in a positive way.

Another study dealing with network structure is discussed in [22]. In that study, it is assumed that each agent only has information about a subset of rows of the partitioned matrix $[A\ b]$ and communication between neighbor agents are specified with random graphs. Contrary to other studies in the literature such as [23], the network is modeled without *B*-connectivity assumption. The random Krasnoselskii-Mann iterative algorithm is used to provide convergence for any A, b and any initial agent states. The algorithm converges independently from the distribution dependency of random graphs if the weighted matrix of the graph is doubly stochastic.

The approach taken in [24] is to find a solution to linear equations from arbitrary initializations by using M-Fejer mappings. For both unique solution and infinitely many solution cases, the proposed algorithm finds a solution to the linear algebraic equation from arbitrary initializations at a geometric rate. When the equation system has a unique solution, by evaluating the mixed norm of homogeneous M-Fejer mappings, the

geometric convergence rate of the algorithm is shown. For the infinitely many solutions case, by orthogonal decompositions of the agents' estimates onto the row space and null space of A, it is proven that the algorithm still converges at a geometric rate, and it is shown how the initialization affects the final convergence point.

In [25], the algorithm proposed in [4] is discussed with some advanced perspectives. The authors; improved the numerical stability of the proposed algorithm and removed the necessary initialization procedure, and the algorithm can achieve the desired solution with minimum L_2 norm when the linear algebraic equation has multiple solutions.

While the algorithm updates the $x_i(k)$ values to reach the solution, the agent states can sometimes go out of the solution space. Once states leave the solution space, they are not able to return. To make the algorithm resistant to this kind of errors, a compensation term which keeps the iterative solutions of the agents in the solution space, is added to the algorithm. This term also ensures that for arbitrary chosen initial states, the algorithm reaches exponential convergence. When $Ax = b$ has infinitely many solutions, by using the algorithm in [4], the solution to be achieved is unclear. It is shown in this study that improving the algorithm by a special initialization step ensures that the algorithm is capable of achieving the intended solution with minimum L_2 norm.

In [26], two stochastic distributed algorithms are proposed to solve linear algebraic equations in the form of $Ax = b$ where each agent only has information about a subset of $[A\ b]$ and the communication between agents is characterized by a repeatedly jointly strongly connected, directed graph. To lessen the communication costs and the needed memory, each agent partitions its estimate vector into multiple blocks, and broadcasts one block at each step of iteration randomly. This partition can be both homogenous and heterogeneous. Two distributed algorithms are proposed for both situations and it is shown that for any linear equation with a unique solution and any strongly connected network, all agents' states reach the unique solution of the system when the proposed algorithms are utilized.

When the equation system $Ax = b$ has no solution, it is possible to obtain a least square approximate solution. In the literature, there are several studies that address this issue. In [16], two distributed algorithms are proposed to obtain least squares approximate solution for both continuous time and discrete time cases. Each agent knows one of the linear equations and every agent has a dynamic state. They aim to reach a consensus on the least squares approximate solution by sharing their state vectors over a network. A discrete-time distributed algorithm is developed by Euler's method, which converges exponentially fast to the least squares approximate solution of the equation. The effectiveness of the proposed algorithm is examined for different types of graphs.

In [27], a distributed algorithm for finding least square approximate solutions in a network with time-invariant, connected, and undirected graph is proposed and it is shown that the convergence is exponentially fast. In addition, it is not required to use carefully chosen time-varying small step sizes for convergence.

In [28], a discrete time algorithm is developed which provides exponential convergence of agents' estimates to the exact least squares solution by using Euler's discretization method. Different from the other algorithms that find least squares approximate solutions, this algorithm allows an arbitrarily chosen agent to obtain the least square solution in a finite number of time steps, by adding a finite time computation mechanism to the iteration procedure. This means that a single agent can solve the equation without communicating its neighbors, within a finite number of time steps. The main idea behind this is to reduce communication costs and time spent on calculations. Other contribution of this study is that this algorithm is applicable to networks with both directed and undirected communication graphs. Numerical examples are given to validate theoretical findings.

In [29], the proposed algorithm provides exponential convergence with finite data rates, for both unique solution and unique least squares approximate solution cases. In a network where each agent knows only one of the linear equations, as a result of data rate restrictions, each agent constructs an encoder-decoder pair to transmit its state and also generate estimates of its neighbors from the data received from them. When the equation has a unique solution, the algorithm ensures all agents' states to converge exponentially fast to that solution in a network with an undirected connected graph. It is

shown that the least squares solution can be obtained for the unique least squares solution case by choosing an appropriate time-varying step size.

The problem of finding the unique least squares solution problem is investigated in [30] and [31]. For a network with a fixed, undirected and connected graph, both algorithms ensure the exponential convergence to a least square solution without using any time varying step size. Unlike [30], the dimension of the states of the agents does not have to be the same as the dimension of the network and it is assumed that there is a self-arc at every agent i of the network. In [31], under these conditions, proposed algorithms allow all agents to obtain the same least square solution exponentially fast.

1.2. Engineering Examples of Linear Equations

Linear equation systems have many applications in physics, mathematics and engineering [32]. In this section, several applications of linear equation systems in the field of engineering will be given.

Before mentioning the applications of linear equations, it is necessary to give some basic definitions about linear functions and linear equations. Consider the following system of linear equations.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ \dots\dots\dots\dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ \dots\dots\dots\dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ \dots\dots\dots\dots + a_{mn}x_n &= b_m \end{aligned} \qquad (1.17)$$

We can rewrite these equations as $Ax = b$ where,

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}. \qquad (1.18)$$

Definition 1.1(Linear Function): A function $f(x)$ is called a linear function if the followings hold.

$$f(x+y) = f(x) + f(y),\ \forall x, y \in R^n \quad (1.19)$$

$$f(ax) = af(x), \qquad \forall x \in R^n, \forall a \in R \quad (1.20)$$

For a system defined as $Ax = b$,

- b may refer to the observation or measurement. In this case x is the vector of unknowns and can be determined from the equation $x = A^{-1}b$ if A is square and invertible
- x may refer to the action or input, when b is the result or output.
- $Ax = b$ may express a transformation or function which maps $x \in R^n$ into $b \in R^m$.

Since *i*th component of b can be expressed as $b_i = \sum_{j=1}^{n} a_{ij}x_j$, we conclude that

- *i*th row of A corresponds to *i*th output,
- *j*th column of A corresponds to *j*th input, and
- $a_{ij} = 0$ indicates that *i*th output (b_i) is independent from *j*th input (x_j) [33].

Note here that, a_{ij} is the gain factor from x_j to b_i. Furthermore, if $a_{ij} = 0$ for all $i < j$, then A is called lower triangular and the output b_i only depends on $x_1, x_2, \dots, x_i$. If $a_{ii} = 0$ for $\forall i$, A is called a diagonal matrix and the output b_i only depends on the corresponding input x_i.

Linear equations have extensive applications such as estimation, inversion, control, design, mapping, transformation, matrix multiplication as mixture of columns [34]. In this section, these interpretations and some examples will be given.

1.2.1. Estimation Applications

Some real-world estimation problems can be expressed as linear equations in the form of $Ax = b$, where b_i is the *i*th measurement or state value of the sensor (known), x_j is the *j*th unknown to be calculated and a_{ij} is the sensitivity of the *i*th sensor to the *j*th unknown parameter. To solve these equation systems, all x's that result in b must be determined [34].

1.2.2. Control and Parameter Design Applications

In the control and parameter design applications, x is the vector of design parameters or inputs which should be properly selected to meet design requirements, b is the vector of outcomes that depend on the chosen design parameters and A is the matrix which shows how these parameters affect the output. In transformation and mapping applications, x is transformed or mapped to b via a linear function $Ax = b$. Objective of such applications is to determine all x vectors that map to a given b vector [34].

1.2.3. Matrix Multiplication as Mixture of Columns

Another use of linear equations is matrix multiplication as mixture of columns. Let $a_j \in R^m$ denote the jth column of A. Then one can re-write $Ax = b$ as,

$$x_1 a_1 + x_2 a_2 + \ \dots\dots + x_n a_n = b \tag{1.21}$$

where x_j are scalars ($j = 1,2, \dots, n$). Here, b is called a mixture or a linear combination of the columns of A [34].

1.2.4. Electrical Circuits

Another engineering field where linear equations arise is electrical circuits. The circuit depicted in Figure 1.1 consists of resistors, dependent and independent sources.

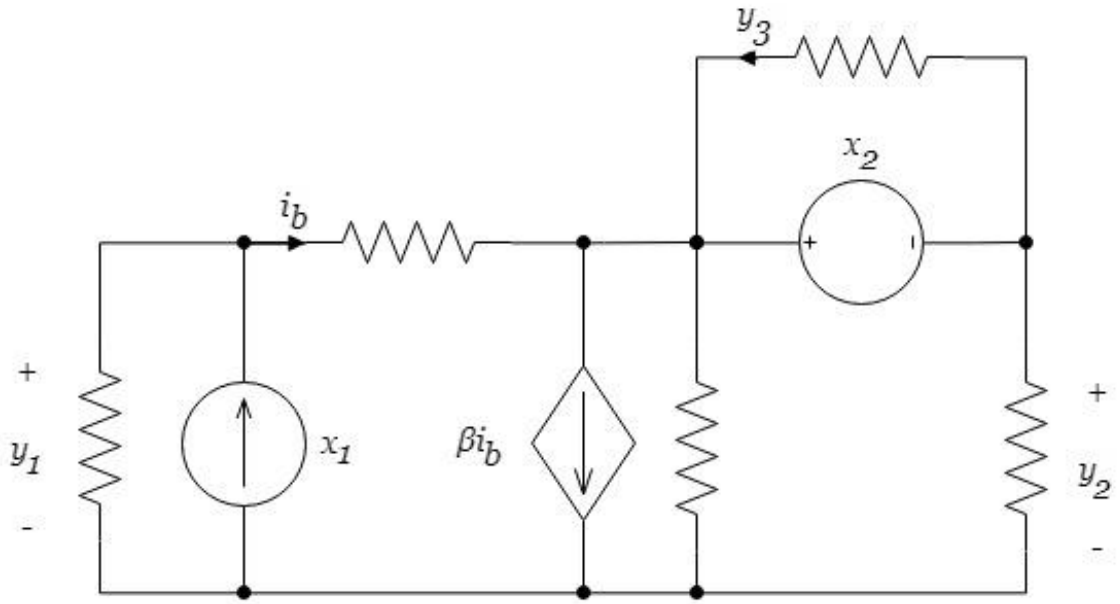

Figure 1.1. An electrical circuit [35].

In the figure above, x_j represent the current and voltage values of the independent sources whereas y_i are voltages of the resistors or currents flowing through the resistors. Equations related to voltages and currents can be written in the form of $Ax = b$. If x_j represent currents and y_i represent voltages, then, A is called the impedance matrix of the circuit [35].

1.2.5. Dynamic Systems

Another application of linear equations in physical systems is the dynamics of a mass. Consider a mass that is subject to a time varying force f illustrated in Figure 1.2.

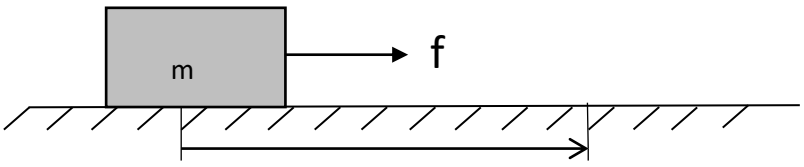

Figure 1.2. Final position of mass under the applied forces.

It is assumed that the mass is at the zero position when $t = 0$ and let $f(t)$ denote the force applied to the mass for $0 \leq t \leq n$. Suppose that $f(t)$ is constant between two consecutive time intervals and $f(t) = x_j$ for $j - 1 \leq t < j$, $j = 1,2, \ldots n$ and, y_1 and y_2 are the final position and the velocity of the mass at $t = n$, one can express the relation between the applied forces f_i and the final position and velocity of the mass as a linear equation $Ax = b$, where a_{1j} shows the effect of the force applied to the mass during $j - 1 \leq t < j$ on the final position, and a_{2j} shows the effect of the force applied to the mass during $j - 1 \leq t < j$ on final velocity [35].

1.2.6. Thermal Systems

A thermal system can also be given as an example of an application of linear equations. Figure 1.3 shows a thermal system where there are multiple heating elements which together change the temperature of the plate.

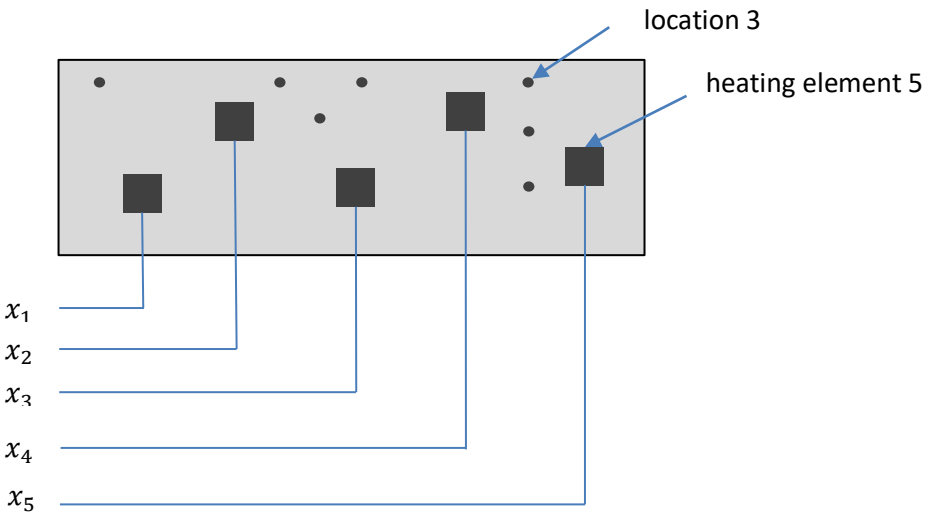

Figure 1.3. A thermal system.

Heating elements change the steady state temperatures at different locations via links between sources and the plate. In the figure, x_j represents the power of the jth heat source or heating element, and y_i is the temperature change at location i. The overall thermal equations can be written as $Ax = b$ where a_{ij} represents the effect of heating element j at location i. ith row of A represents the effects of the heaters to the

temperature at the location i and jth column of A shows mathematical expression of the steady state temperature increase corresponding to each 1 W power in the heater j [35].

1.2.7. Industrial Processes

Industrial production processes can also be modeled by linear equations. In such systems, the product inputs such as materials, parts, labor; and the products that will be formed at the end of the production processes are the inputs and the outputs of the system, respectively. Linear equations can be used to express production costs in terms of the inputs. When the system is written in the form of $Ax = b$, x_j represents the unit price of the input j, a_{ij} is the amount of the input j required to produce a unit of the product i, and y_i is the production cost of a unit of the product i. Here the ith row of the matrix A represents the total cost of all materials that must be spent for production a unit of the product i [35].

1.3. The Motivation of the Thesis

Solving linear algebraic equation systems is one of the most studied topics in physics, mathematics and engineering. These systems can be used to model many real-life situations by using multi-agent networks. In recent years, there has been a significant increase in the application areas of network systems. Examples of these systems include computer networks, wireless sensor applications, and renewable energy turbines. New types of networks and fields of applications are of interest to scientists and engineers for future works.

Several centralized methods have been proposed for the solution of systems of linear equations such as Jacobi method, Gauss-Seidel method, successive over-relaxation method, Kacmarz method and conjugate gradient method. Centralized methods, which work based on finding the inverse of the coefficients matrix A, are very robust, and they require the knowledge of whole coefficients of A. However, centralized methods are not always effective due to long processing times, insufficient flow of information, and security constraints.

For the above-mentioned reasons, several algorithms have been proposed for the solution of systems of linear equations, which take the system as a network of

independent and autonomous agents and solve the system in a distributed manner. Our main motivation in this thesis is to perform convergence rate analysis for these proposed algorithms and to propose a modification to the algorithms in the literature what yields faster convergence.

1.4. The Contributions of the Thesis

The contribution of this thesis to the literature is to increase the convergence rate of a discrete time algorithm proposed for distributed solution of linear equation systems, based on the method of estimating neighbor equations from their solutions. With this novel rapid method, state vectors of agents are aimed to converge faster to the solution of the equation system and simulation results are obtained to support this claim. In addition, this method has been shown to be applicable in case of transmission delays.

As another contribution, the effect of the weighting coefficients on the convergence rate of the algorithm is investigated. In addition to the equal weighting method, which is frequently used in the literature, different weighting methods have been examined and the best method (in terms of the convergence speed) has been determined.

1.5. The Organization of the Thesis

The remainder of this thesis is organized as follows: In Chapter 2, graph theoretical concepts and the mathematical background of the thesis are given. General structure of linear equation systems and the solution types of these systems are explained.

In Chapter 3, the problem of solving linear equations via a multi-agent network is introduced and the algorithm proposed in [1] is given. For an example equation system, convergence rate of the algorithm is examined. As a contribution, a novel rapid method based on updating the initial states and the projection matrix, by using the data that comes from neighbors, is proposed. Transmission delayed version of the method is also discussed.

In Chapter 4, the effect of weighting strategies on the convergence rate of the distributed algorithm for solving linear equations is examined for different types of network topologies. As a result of simulation studies, the fastest one among these weighting strategies is determined.

Finally, in Chapter 5, the results of the simulations are evaluated and the possible directions of future studies are discussed.

2. MATHEMATICAL PRELIMINARIES

2.1. Graph Theory

2.1.1. Notation and Definitions

Solving linear equations of the form $Ax = b$ in a distributed manner requires autonomous computational units, which are called agents, to work cooperatively. These agents form a multi agent network which is represented by a graph. In this chapter, basic concepts and definitions of graph theory which are used in network modeling are given.

Definition 2.1 : Graph A graph $G = (V, \mathcal{E})$ consists of a nonempty set $V = \{v_1, v_2, v_3, \dots v_n\}$ of vertices and a set $\mathcal{E} \subseteq V \times V$ of edges. If an edge $e_{ij} = (v_i, v_j)$ is the link between vertices v_i and v_j which represents the data flow from agent i and agent j, then the vertices $v_i$and v_j are called the ends of e [36].

Definition 2.2 : Self loop Let $G = (V, \mathcal{E})$ be a graph and an edge $e_{ij} = (v_i, v_i)$, then edge e_{ii} is called a self-loop. It means that any edge which is a single element subset of V is called a self-loop [37].

Definition 2.3 : Vertex Adjaceny Given a directed graph $G = (V, \mathcal{E})$, two vertices v_1 and v_2 are called adjacent if $e_{ij} \in \mathcal{E}$.

Definition 2.4 : Neighborhood $G = (V, \mathcal{E})$. The neighbors of the vertex v_i are the vertices that are adjacent the vertex v_i, i.e., [38].

$$N_i = \{v_j \in V : (v_j, v_i) \in \mathcal{E}\} \tag{2.1}$$

Definition 2.5 : Directed Graph A directed graph $G = (V, \mathcal{E})$ contains a nonempty vertex set $V = \{v_1, v_2, v_3, \dots v_n\}$ and an ordered edge set $\mathcal{E} \subseteq V \times V$. In a directed graph $e_{ij} = (v_i, v_j)$ denotes the edge from vertex j to vertex i. Directed graphs are also named as digraphs [36].

Definition 2.6 : Complete Graph A graph $G = (V, \mathcal{E})$ is called a complete graph if every vertex of G is connected to all others [36].

Definition 2.7 : Adjacency Matrix Adjacency matrix $Adj = [a_{ij}]$ of a graph G is a matrix consisting of zeros and ones where $a_{ij} = 1$ if $e_{ij} \in \mathcal{E}$ and $a_{ij} = 0$ otherwise.. Rows and columns of an adjacency matrix represent the nodes of the corresponding graph and the dimensions of Adj depend on the number of nodes in that graph [36].

Definition 2.8 : Walk Let $G = (V, \mathcal{E})$ be a graph. A walk w in G is obtained by joining vertices with edges, starting from a vertex in G. If the starting and end point of a walk is the same vertex, walk w is called closed; otherwise the walk is an open walk. A walk with a single vertex is named as trivial [38].

Definition 2.9 : Strong Connectedness For a directed graph G, if there exists a directed walk for every pair of vertices v_i and v_j in the vertex set, and every vertex is reachable form any other vertices, G is called strongly connected [38].

2.1.2. Some Important Network Topologies

Network topology shows how agents are physically interact with each other. Network topology can be modeled as a map that allows us to see the physical layout of connected agents and communication between them.

There are various types of topologies including fully connected topology, star topology, ring topology, path (bus) topology and tree topology. In this section some important types of network topologies will be given.

2.1.2.1. Fully Connected Topology

In a fully connected topology, every agent is connected to all other agents. The advantage of this type of topology is its robustness since the interconnectivity of the agents makes the network resistant to errors. However, it takes more configuration time and is more expensive in terms of construction cost, than other topologies [39]. Example of a fully connected network topology is given in Figure. 2.1.

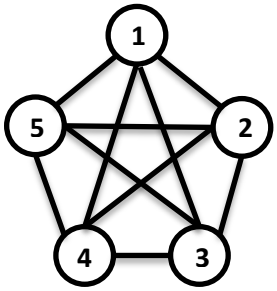

Figure 2.1. Fully connected network topology.

2.1.2.2. Star Topology

In a star topology, all agents are directly connected to a single central agent. Therefore, agents are connected indirectly to others in this topology. In computer networks, the central agent is generally a server and the other agents are clients. Data can be directly transmitted from the central agent to every other agent.

Star topologies are one of the most commonly used topologies because they allow us to control the network from a single point. As a result of that, even there is a failure at a client agent, the network will remain to process properly. However, if there is a failure at the central agent, the whole network collapses [40]. Example of a star network topology is illustrated in Figure. 2.2.

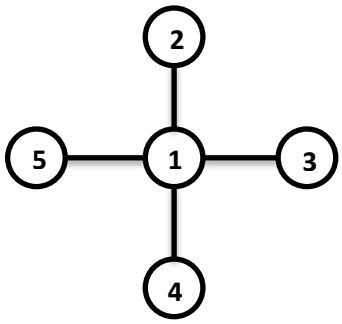

Figure 2.2. Star network topology.

2.1.2.3. Ring Topology

In this type of networks, agents are connected to each other in a circular way. Information flow in the network is circular and it can be unidirectional or bidirectional. It means that every agent has two neighbors if the information flow is bidirectional, otherwise agents have one neighbor [40]. Example of a ring network topology is given in Figure. 2.3.

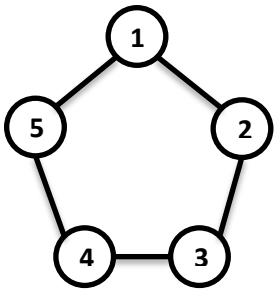

Figure 2.3. Ring network topology.

2.1.2.4. Path (Bus) Topology

Path topology is a type of network where every agent is connected on a single path which starts from the first agent and ends at the last agent. Note that one link of a network with ring topology is broken, a path topology will be formed. These types of topologies are suitable for small scale networks. However, they are vulnerable to failures since a single connection loss between agents causes whole network to be unfunctional [40]. Example of a path topology is shown in Figure. 2.4.

Figure 2.4. Path network topology.

2.2. Solution Types, Null Space and Projection Matrices

2.2.1. Null Space and Projection Matrices

Null Space

The null space of a matrix is the subspace that contains all solutions of $Ax = 0$. In other words, every solution to $Ax = 0$, is in the null space of A.

$$null(A) = \{x \in R^n | Ax = 0\} \tag{2.15}$$

Vectors in the null space of A are orthogonal to all rows of A [43].

Properties of Projection Matrix

Projection matrix P is an $n \times n$ square matrix which allows a projection from a vector space V to a subspace W. Note that for a projection matrix P, we have $P^T = P$ and $P^2 = P$, i.e., P is idempotent.

The mathematical expression of the projection matrix P, onto the nullspace of A, is defined as follows [32].

$$P = I - A^T(AA^T)^{-1}A \tag{2.16}$$

A linear equation can be defined as,

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b \tag{2.2}$$

where $x_1, x_2, \ldots, x_n$ are unknown variables, $a_1, a_2, \ldots, a_n$ are coefficients of x_i and b is called the constant term. A linear equation system consists of a set of linear equations with the same unknowns. An example of a linear equation system can be written as follows,

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ \ldots\ldots\ldots\ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ \ldots\ldots\ldots\ldots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ \ldots\ldots\ldots\ldots + a_{mn}x_n &= b_n
\end{aligned} \tag{2.3}
$$

where m is the number of equations and n is the number of unknown variables. In order to solve a system of linear equations, one must obtain scalars $(s_1, s_2, \ldots, s_n)$ that satisfy all equations when substituted $(x_1, x_2, \ldots, x_n)$ respectively. These scalar values are called solutions and the set of all solutions composes the solution set of the system.

When solving a system of linear equations, the system may have a unique solution, no solution and infinitely many solutions. A system of linear equations is called consistent if it has at least one solution, otherwise it is called inconsistent.

2.2.2. Least Squares Solution

For a system of linear equations in the form of $Ax = b$ where $A \in R^{m \times n}$, if $m > n$ (A is skinny) $b \notin range(A)$, then the linear equation set is called over-determined. The common approach to solve such equations is to find an approximate solution to $Ax = b$. In order to do that, first we must define the error vector as $h = Ax - b$, then find the $x = x_{ls}$ value that minimizes $\|h\|$. Here x_{ls} is called the least squares (approximate) solution of the linear equation $Ax = b$ [32].

Figure 2.5 illustrates the least squares approximate solution of a linear equation with two unknowns. The objective is to find the vector $x \in R^n$ that minimizes $\|Ax - b\|$ for a given $b \in R^m$.

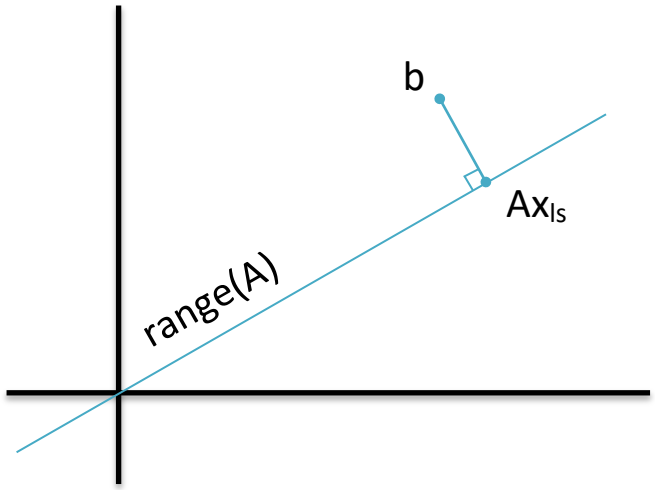

Figure 2.5 Geometric representation of the least squares solution.

In the figure above Ax_{ls} is called the orthogonal projection of b onto range(A) and it is the closest point to b in range(A).

Let $A \in R^{m \times n}$ be a full rank and skinny $(m > n)$ matrix. In order to find the least squares solution x_{ls}, we must minimize the square of the norm of the error which can be computed from

$$\|h\|^2 = x^T A^T Ax - 2b^T Ax + b^T b \tag{2.4}$$

If we take gradient of that expression and set it to zero we have

$$\nabla x \|e\|^2 = 2A^T Ax - 2A^T b = 0 \tag{2.5}$$

By organizing that expression we can get

$$A^T Ax = A^T b \tag{2.6}$$

From the initial assumptions, one can show that $A^T A$ is invertible and therefore x_{ls} least squares solution can be computed as

$$x_{ls} = (A^T A)^{-1} A^T b \tag{2.7}$$

Note that the least squares (approximate) solution x_{ls} has the following properties:

- x_{ls} is a linear function of b.
- If A is a square matrix, x_{ls} becomes $x_{ls} = A^{-1}b$.
- If $b \in range(A)$, x_{ls} solves the equation $Ax_{ls} = b$ [32].

For a full rank and skinny matrix $A \in R^{m \times n}$, the pseudo-inverse of A is defined as,

$$A^{\dagger} = (A^T A)^{-1} A^T \tag{2.8}$$

As mentioned before, Ax_{ls} is the projection of b onto $range(A)$ and it is the closest point to b in $range(A)$ and therefore

$$Ax_{ls} = P_{range(A)}(b) \tag{2.9}$$

where $P_{range(A)}$ is the projection function which maps a vector to the range space of A. Mathematical expression of the linear function $P_{range(A)}$ is given as follows

$$P_{range(A)}(b) = Ax_{ls} = A(A^T A)^{-1} A^T b \tag{2.10}$$

Furthermore, the minimum error is computed as,

$$e = Ax_{ls} - b = (A(A^T A)^{-1} A^T - I)b \tag{2.11}$$

which is orthogonal to $range(A)$:

$$\langle e, Az \rangle = b^T (A(A^T A)^{-1} A^T - I)Az = 0, \text{ for } \forall z \in R^n \tag{2.12}$$

2.2.3. Minimum Norm Solution

For a system of linear equations in the form of $Ax = b$ where $A \in R^{m \times n}$, if $m < n$ (A is fat) the linear equation set is called under determined and in that case there are more unknowns (variables) than the number of equations. Therefore, infinitely many x vectors can map to the same b value. If we assume that A is full rank, for $\forall b \in R^m$, there exist a solution set

$$\{x | Ax = b\} = \{x_p + z | z \in null(A)\} \tag{2.13}$$

where x_p is a particular solution to $Ax_p = b$. Here z can be chosen to make optimization among solutions. The dimension of solution is $\dim(null(A)) = n - m$. The mathematical expression of minimum norm solution is,

$$x_{mn} = A^T(AA^T)^{-1}b \tag{2.14}$$

Since we assume that A is full rank, we have an invertible matrix AA^T. Therefore, the problem yields to an optimization problem where the objective is minimizing $\|x\|$ subject to $Ax = b$ [42].

2.3. Summary of This Chapter

In this chapter, some important preliminaries and the mathematical background of the thesis are given. Firstly, some basic concepts and definitions related to graph theory are explained and then the network topologies used in simulation studies are explained. Unique, least squares and minimum norm solutions of linear equation systems and related mathematical expressions are further discussed.

This page is intentionally left blank.

3. SOLVING LINEAR EQUATIONS

In this chapter, we analyze a distributed algorithm for solving linear equations in the form $Ax = b$ over a multi-agent network consisting of n autonomous agents. First we define the problem of solving linear equations in Section 3.1.1, and then a well-known distributed algorithm is introduced in 3.1.2. Simulation studies and their results are given in Section 3.1.3. As a contribution to the previous studies in the literature, a novel method for rapidly solving linear equations by determining the equations of the neighbors is discussed in Section 3.2. For the cases when there is transmission delay between agents, the proposed algorithm is given in Section 3.3. Finally, in Section 3.4, simulation results and mathematical data obtained in this section are evaluated.

3.1. A Distributed Algorithm for Solving General Linear Equations

3.1.1. Problem Formulation

Let the multi-agent network be represented by a directed graph $G = (V, \mathcal{E})$ consisting of $n > 1$ autonomous agents where each agent of the network is capable of communicating with its neighbors. Let N_i denote the set of neighbors of the agent i, i.e., $N_i = \{v_j \in V : (v_j, v_i) \in \mathcal{E}\}$ and let each agent to be a neighbor of itself.

Let each agent know a subset of the linear equation $Ax = b$, i.e., a subset of the rows of $[A\ b]$. Certain parts of the total equation set are known to agents and the purpose of the agents is to solve equation system cooperatively. In particular, let A_i be the matrix consisting of the known rows of A by the agent i. Similarly, let b_i be the vector consisting of the known components of b. Solving a linear equation over a multi agent network can be expressed as, for each agent i, obtaining the solution of $Ax = b$ by using A_i, b_i and the solution estimates of neighbors. There is a state vector corresponding to each agent, and these vectors are time-varying. Agents communicate with each other in the multi-agent network and state information is exchanged among neighbors. We suppose that time is discrete and k takes values in $\{1,2, \dots\}$. At the beginning ($k = 0$) each agent sets its state $x_i(0)$ as a solution to $A_i x_i = b_i$.

The main purpose of all agents is to solve the total system of equations $Ax = b$ in a cooperative way by updating their initial solutions by taking the neighbors' solutions into account.

3.1.2. Algorithm Design

The main concept behind finding solution to a linear equation system in a distributed and iterative manner is formulating it as a consensus problem. The key idea is as follows: Each agent starts from initial estimates $x_i(0)$ which satisfy their equations $(A_i x_i(0) = b_i)$ and update their estimates by using the estimates of the neighboring agents. The update continues until each agent has the same estimates, i.e., $\lim_{t\to\infty} x_i(t) = \lim_{t\to\infty} x_j(t)$ for all i, j. In particular, let K_i be a basis matrix for the null space of A_i. Then the update of $x_i(t)$ with $x_i(t+1) = x_i(t)+K_i u_i(t), t \geq 0$ ensures, every $x_i(t)$ to satisfy $A_i x_i(t) = b_i$. Therefore, to ensure that a consensus is reached, and to solve the problem, one must choose $u_i(t)$ properly. From the idea of averaging based consensus algorithms [44] [45], one can choose $u_i(t)$ to minimize $(x_i(t) + K_i u_i(t)) - \frac{1}{d_i}\left(\sum_{j\in N_i} x_j(t)\right)$, where d_i is the number of neighbors of agent i. For instance, the following iteration can be used for solving linear equations over a multi agent network [1].

$$x_i(t+1) = x_i(t) - \frac{1}{d_i} P_i\left(d_i x_i(t) - \sum_{j\in N_i} x_j(t)\right), t \geq 1 \qquad (3.1)$$

where P_i is the projection matrix on the null space of A_i. For a nonsingular and full rank coefficients matrix A and a strongly connected graph G, the algorithm above ensures that all agents' estimates to converge successfully to the solution of the linear equation system [1].

3.1.3. Simulation Results

In this section, the convergence of the algorithm (3.1) will be investigated for an example equation system which is solved over a network whose underlying graph is directed and connected as shown in Figure3.1. Note that the graph illustrated in Figure 3.1 is connected which is a necessary condition for solving the linear equation by the distributed algorithm given in Section 3.1.2.

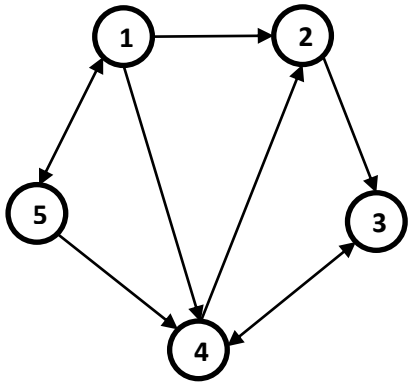

Figure 3.1. Example network graph.

Consider the following linear equation system with 5 unknowns:

$$\begin{aligned} x_1 + 2x_2 - 3x_3 + x_4 + x_5 &= 5 \\ 2x_1 - x_2 - 2x_3 &= -6 \\ 2x_2 + 2x_3 + x_4 &= 14 \\ x_1 - x_3 + 5x_4 + 2x_5 &= 28 \\ x_1 + 2x_2 + x_3 + 2x_4 + x_5 &= 21 \end{aligned} \tag{3.2}$$

The linear equation can be written in the form $Ax = b$ matrix form where

$$A = \begin{bmatrix} 1 & 2 & -3 & 1 & 1 \\ 2 & -1 & -2 & 0 & 0 \\ 0 & 2 & 2 & 1 & 0 \\ 1 & 0 & -1 & 5 & 2 \\ 1 & 2 & 1 & 2 & 1 \end{bmatrix} \text{ and } b = \begin{bmatrix} 5 \\ -6 \\ 14 \\ 28 \\ 21 \end{bmatrix} \tag{3.3}$$

Note that the linear equation system has a unique solution which is $x^* = [1,2,3,4,5]^T$. Let each agent know only one row of the partitioned matrix $[A\ b]$. At $k = 1$, each agent starts iteration by generating a solution to its own equation. While there are infinitely many candidate solutions, we select minimum norm solution. Initial states of the agents are given in Table 3.1

Table 3.1. Initial solutions of the agents.

Agent	**Initial Solutions**				
1	[5/16	5/8	-15/16	5/16	5/16]
2	[-133/100	67/100	133/100	0	0]
3	[0	311/100	311/100	39/25	0]
4	[9/10	0	-9/10	113/25	9/5]
5	[191/100	191/50	191/100	191/100	191/100]

When the algorithm given in section 3.1.2 is used to solve the equation system, estimates of the agents converge to the solution of $Ax = b$ asymptotically. Figure 3.2 illustrates the evolution of $x_1(k)$ and Figure 3.3 shows the convergence of the sum of the norms of the solution errors of the agents, which can be expressed mathematically as,

$$h(k) = \sum_{i=1}^{n} \|x_i(k) - x^*\|_2 \tag{3.4}$$

Note that the convergence is asymptotic, i.e., $\lim_{k\to\infty} h(k) = 0$ and $h(k) > 0$ for all finite k. However, in order to comment on the speed of convergence, we are interested in the number of time steps required to achieve an error value that is less than a pre-defined threshold. For instance, for $\epsilon = 10^{-2}$, the time steps required to achieve $h(k) < \epsilon$ is $k = 5958$.

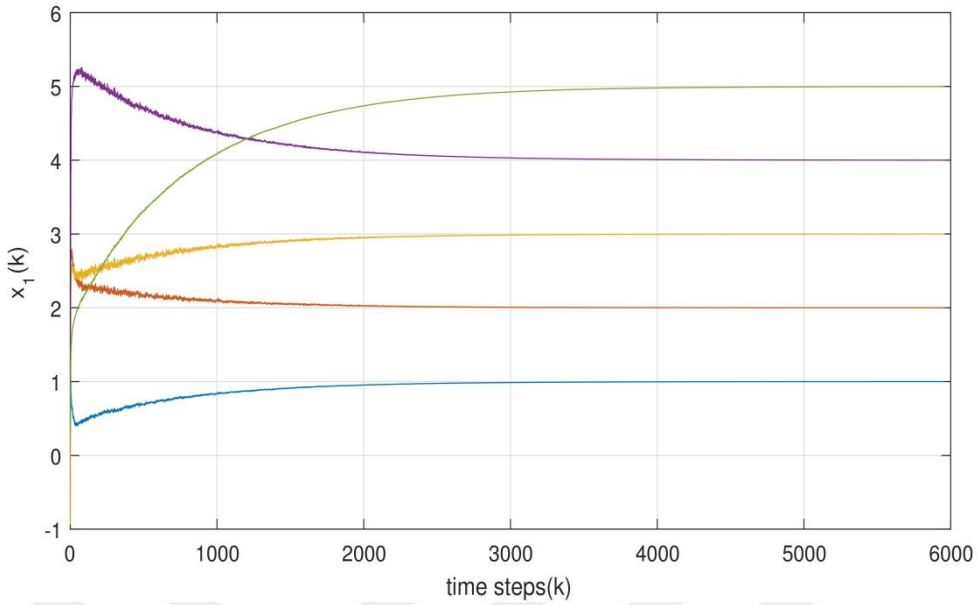

Figure 3.2. Evolution of $x_1(k)$, when algorithm (3.1) is used.

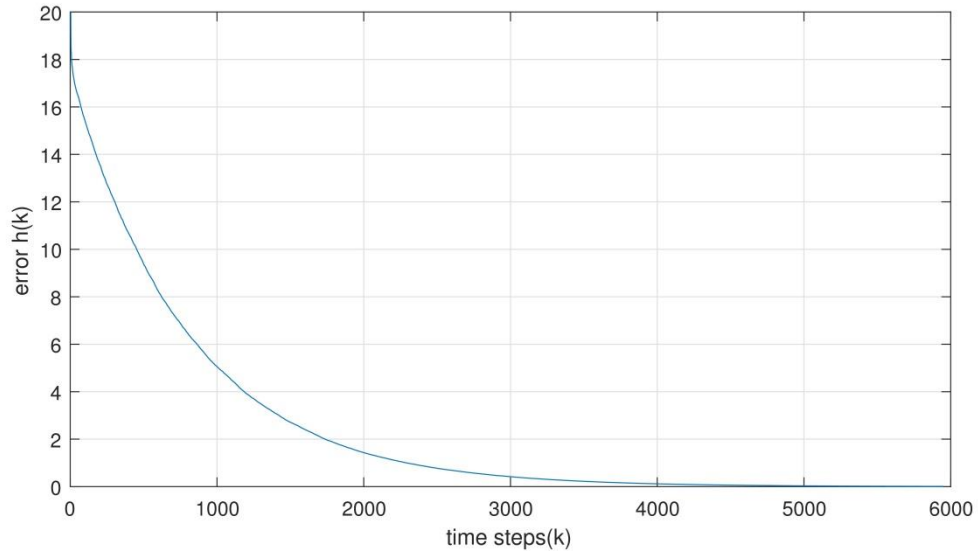

Figure 3.3. Evolution of the error ($h(k)$).

3.2. Rapid Solution of Linear Equations via Distributed Algorithms

In the literature on distributed solution of linear equations, almost all researchers have assumed that agents do not share their equations with neighbors but solution estimates for security and privacy reasons. However this assumption is not restrictive as illustrated in the following example.

Example 3.2

Consider a two agent network where agent 2 knows the equation $x_1 + x_2 = 2$ and generates solution vectors $\begin{bmatrix} 2 \\ -1 \end{bmatrix}$ and $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ in two different time steps and share them with agent 1. Since linear equation with two unknowns can be written as,

$$a_1 x_1 + a_2 x_2 - b = 0 \tag{3.5}$$

One can obtain the following two equations provided that the solutions generated by agent 2 satisfy its equation

$$\begin{aligned} 2a_1 - a_2 - b &= 0 \\ 0.5a_1 + 0.5a_2 - b &= 0 \end{aligned} \tag{3.6}$$

Furthermore, these equations can be written in matrix form as

$$\begin{bmatrix} 2 & -1 & -1 \\ 0.5 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.7}$$

The equation system above shows that the coefficients and the constant of the equation known by agent 2 can be determined by computing the nullspace of the matrix above which can be computed as $k \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ where $k \in R$. Therefore, we conclude that it is possible to determine the equation with n unknowns by using n linearly independent solutions to the equation.

3.2.1. Proposed Algorithm

Motivated by the previous example, given a linear system with n unknowns and $p < n$ linearly independent equations for each agent, the following algorithm can be used to find the solution of the system over a multi-agent network:

Algorithm 1 Algorithm for solving linear equations over networks

$P_i \leftarrow I - A_i^T(A_iA_i^T)^{-1}A_i$

$x_i(1) \leftarrow A_i^T(A_iA_i^T)^{-1}b_i$

for each $j \in N_i$ **do**

    initialize X_j as an 2D array

end for

for $k := 1$ **to** pn **do**

    for each $j \in N_i$ **do**

        Append the rows of $\left[I_p \otimes x_j(k)^T - I_p\right]$ to X_j

        $\overline{w}_{ij} \leftarrow$ A random number between 0 and 1

    end for

    for each $j \in N_i$ **do**

        $w_{ij} \leftarrow \overline{w}_{ij}/\sum_j w_{ij} = 1$

    end for

    $x_i(k+1) \leftarrow x_i(k) - P_i\left(x_i(k) - \sum_{j\epsilon N_i} w_{ij}x_j(k)\right)$

end for

$\bar{A} \leftarrow A_i$

$\bar{b} \leftarrow b_i$

for each $j \in N_i$ **do**

    Compute $\bar{A}_j$ whose columns span $null(X_j)$

    Append the first n columns of $\bar{A}_j$ as new rows of $\bar{A}$

    Append the last column of $\bar{A}_j$ to $\bar{b}$

end for

$A_i \leftarrow \bar{A}$

$b_i \leftarrow \bar{b}$

$P_i \leftarrow I - A_i^T(A_iA_i^T)^{-1}A_i$

$x_i(1) \leftarrow A_i^T(A_iA_i^T)^{-1}b_i$

$k \leftarrow pn + 1$

Receive new solutions of the neighbors

repeat

$x_i(k+1) \leftarrow x_i(k) - P_i\left(x_i(k) - \sum_{j\epsilon N_i} w_{ij}x_j(k)\right)$

$k \leftarrow k + 1$

until $\|x(k+1) - x(k)\| < \epsilon$

Note that the weighting coefficients w_{ij} are randomly chosen at each iteration, which ensures that the solution estimates generated by an agent in different iterations are not the same, i.e., they are almost surely guaranteed to be linearly independent. For $k = pn$ time steps, agent i receives the solutions of its neighbors, and solves its own equation by a distributed algorithm. At the end of that period, each agent computes the nullspace of the matrix generated by using the solutions of its neighbors and determines the equations of the neighboring agents. Then, each agent re-initializes its solution estimate which satisfies not only its equation but also the equation of its neighbors. Furthermore, agents compute the projection matrices to the nullspace of updated A_i. Then the update algorithm (1) is used with the new initial solutions and projection matrices.

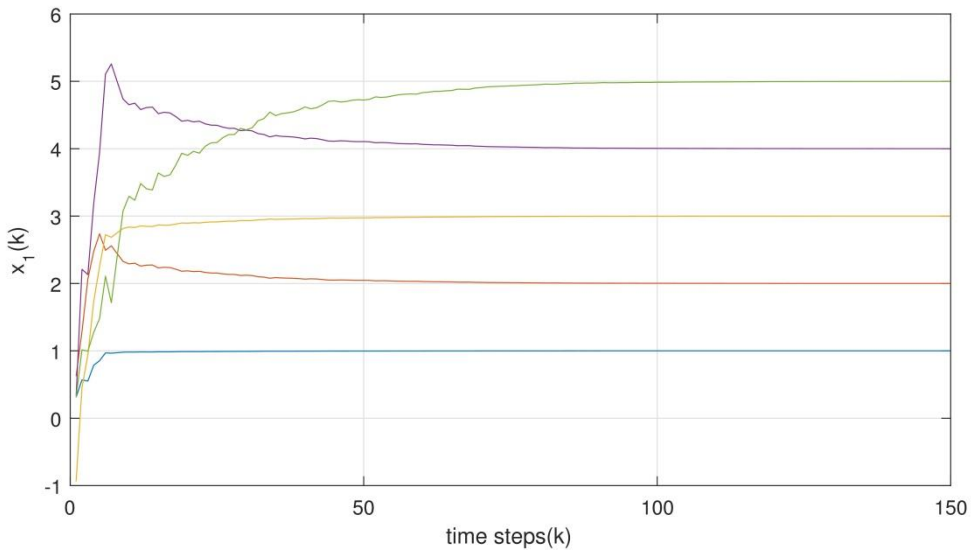

Figure 3.4. Evolution of $x_1(k)$, when the proposed distributed algorithm updated with neighbor data is used for the given equation system.

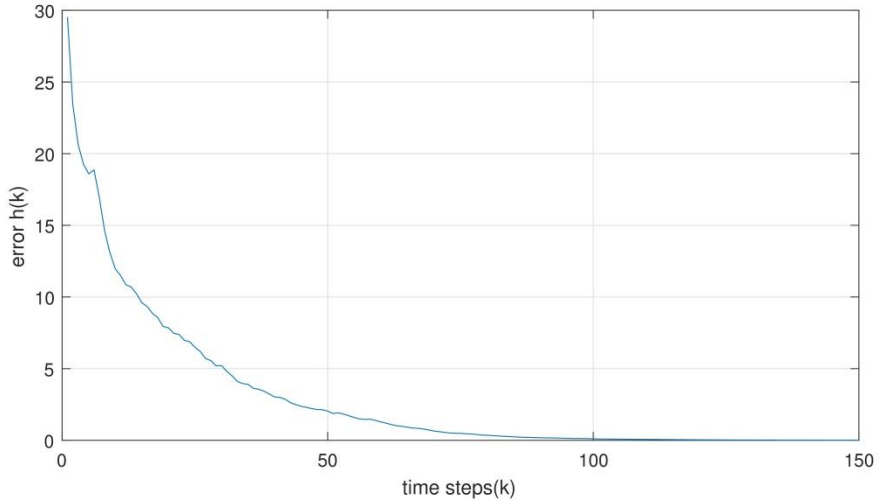

Figure 3.5. Evolution of the error ($h(k)$).

The given equation system is solved at $k = 144$ time steps which is faster than the original algorithm. This method increases the speed of convergence of the solution procedure.

3.3. Rapid Solution in Delayed Networks

In practical applications there may be transmission delays in the network due to reasons such as connectivity errors, physical difficulties, and processor capabilities [46] [47] [48]. Since delay is unavoidable in such networks, algorithms must be robust enough to overcome the adverse effect of delay. Furthermore, delay is a factor that affects the speed of the distributed algorithms. In this section, we investigate the effect of the delay on the convergence rate of the algorithm. In order to make a consistent comparison, the example equation system used in the previous sections will be reconsidered. In the existence of transmission delay, the distributed algorithm can be modified as follows [23]:

$$x_i(t+1) = x_i(t) - \frac{1}{d_i} P_i\left(d_i x_i(t) - \sum_{j \in N_i} x_j\left(t - \tau_{ij}\right)\right), t \geq 0 \qquad (3.8)$$

where $\tau_{ij} \in \{0,1, \ldots \tau_{max}\}$ is the transmission delay between agents i and j. Note that $\tau_{ii} = 0$ for all i, which means that every agent uses its own data without delay. When the original algorithm proposed in [1] is used to find the solution of given equation system, the following graph in Figure 3.6 is achieved.

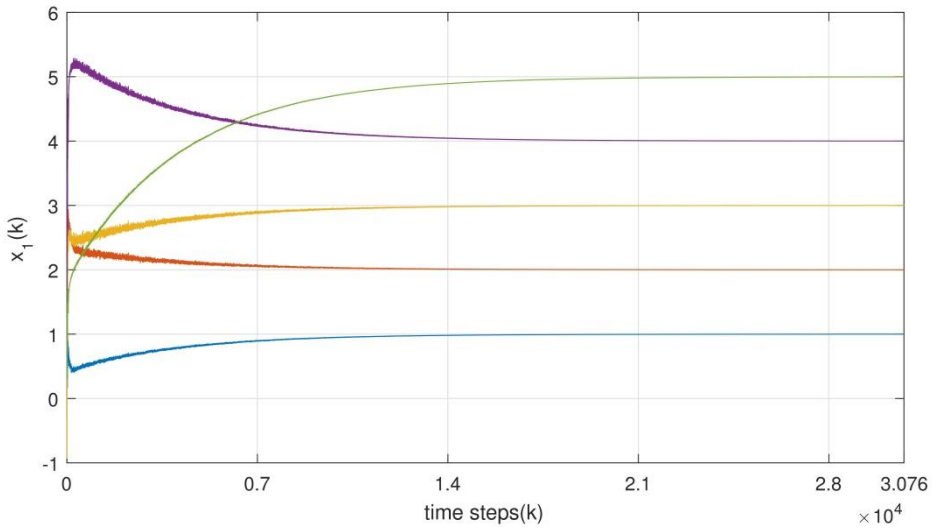

Figure 3.6. Evolution of $x_1(k)$, when algorithm (3.1) is used for the system with transmission delay.

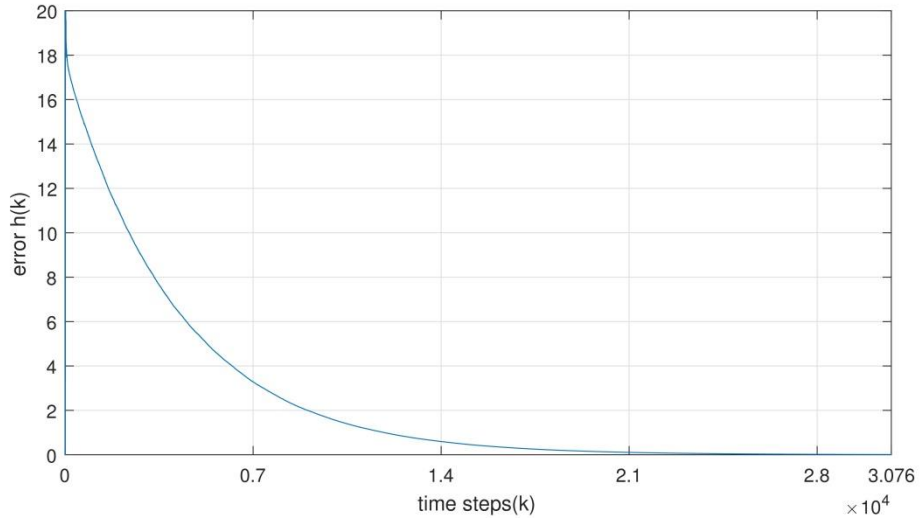

Figure 3.7. Evolution of the error ($h(k)$).

For the given equation system, in the presence of transmission delay, the solution is reached at approximately $k = 30600$ time steps as depicted in Figure 3.6. It can clearly be seen that there is a vast difference between the original delay-free system and this system in terms of convergence speed. Since this is not desirable, the approach discussed in the previous section will be adopted for systems with transmission delay. The following algorithm can be used for rapid solution in case of delay.

Algorithm 2 Algorithm for solving linear equations over networks with transmission delay

$P_i \leftarrow I - A_i^T(A_iA_i^T)^{-1}A_i$

$x_i(1) \leftarrow A_i^T(A_iA_i^T)^{-1}b_i$

for each $j \in N_i$ **do**

    initialize X_j as an 2D array

end for

for $k := 1$ **to** $pn(\tau_{max}+1)$ **do**

    for each $j \in N_i$ **do**

        Append the rows of $[I_p \otimes x_j(k)^T - I_p]$ to X_j

        $\overline{w}_{ij} \leftarrow$ A random number between 0 and 1

    end for

    for each $j \in N_i$ **do**

        $w_{ij} \leftarrow \overline{w}_{ij}/\sum_j w_{ij} = 1$

    end for

    $x_i(k+1) \leftarrow x_i(k) - P_i\left(x_i(k) - \sum_{j\epsilon N_i} w_{ij}x_j(k-\tau_{ij})\right)$

end for

$\bar{A} \leftarrow A_i$

$\bar{b} \leftarrow b_i$

for each $j \in N_i$ **do**

    Compute $\bar{A}_j$ whose columns span $null(X_j)$

    Append the first n columns of $\bar{A}_j$ as new rows of $\bar{A}$

    Append the last column of $\bar{A}_j$ to $\bar{b}$

end for

$A_i \leftarrow \bar{A}$

$b_i \leftarrow \bar{b}$

$P_i \leftarrow I - A_i^T(A_iA_i^T)^{-1}A_i$

$x_i(1) \leftarrow A_i^T(A_iA_i^T)^{-1}b_i$

$k \leftarrow pn(\tau_{max}-1)+1$

Receive new solutions of the neighbors

repeat

$x_i(k+1) \leftarrow x_i(k) - P_i\left(x_i(k) - \sum_{j\epsilon N_i} w_{ij}x_j(k-\tau_{ij})\right)$

$k \leftarrow k+1$

until $\|x(k+1) - x(k)\| < \epsilon$

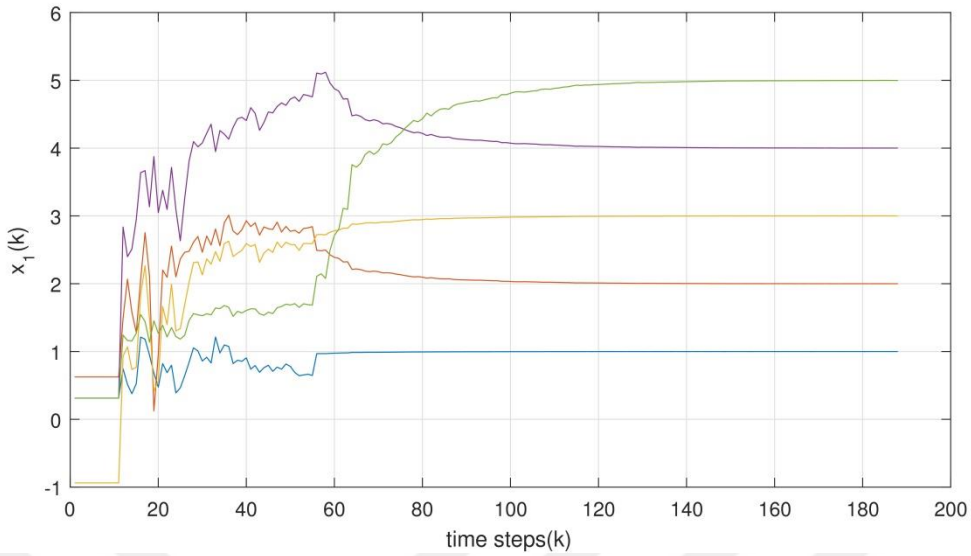

Figure 3.8. Evolution of $x_1(k)$, when the proposed distributed algorithm is used for the system with transmission delay.

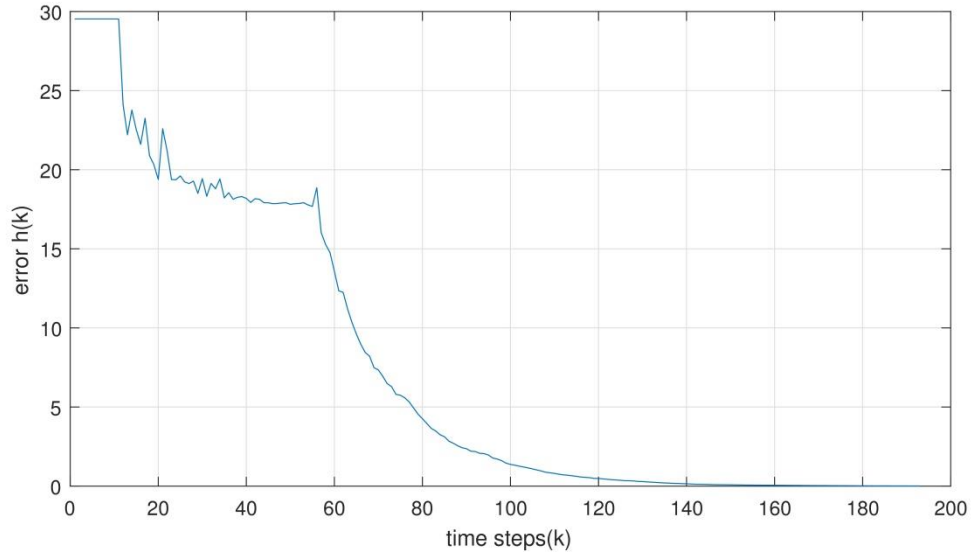

Figure 3.9. Evolution of the error ($h(k)$).

When the proposed algorithm is used, convergence to the solution is as shown in Figure 3.8 for agent 1. Note that at iteration $k = 55$, agent re-calculates P_i and re-initializes $x_i(0)$ and consequently solves the equation in approximately $k = 193$ time steps. These results clearly show that the proposed method increases the rate of convergence in delayed networks as well.

3.4. Summary of This Chapter

In this chapter, we investigate the convergence speed of a distributed algorithm proposed in [1]. The simulation studies show that the convergence speed is not satisfactory for solving an example equation system in a distributed manner. For this purpose, a rapid method is proposed by which agents are able to determine the equations of their neighbors and solve the equation in less number of steps. The same mechanism is applied to the networks with transmission delay and it is shown that the proposed method can also be used for delayed networks for faster convergence.

This page is intentionally left blank.

4. CONVERGENCE RATE ANALYSIS OF DISTRIBUTED ALGORITHMS

In this chapter, we study the effect of the choice of weighting coefficients on the convergence speed of the distributed algorithm given in Section 3.2. Consider the update algorithm proposed in [1];

$$x_i(t+1) = x_i(t) - P_i\left(x_i(t) - \sum_{j\epsilon N_i} w_{ij} x_j(t)\right), \quad t \geq 0 \tag{4.1}$$

where, w_{ij} are the weighting coefficients that agent i use to update its solution estimates.

Throughout this section, the following are assumed to be satisfied for the weighting coefficients

$$\begin{aligned} &\text{i.} && w_{ij} \geq 0, && \text{for all } i,j \text{ ,} \\ &\text{ii.} && w_{ij} = 0, && \text{if } (v_j, v_i) \notin \mathcal{E} \text{ ,} \\ &\text{iii.} && \textstyle\sum_{j=1}^{n} w_{ij} = 1 \text{ ,} && \text{for all } i \end{aligned} \tag{4.2}$$

4.1. Weighting Strategies

In order to increase the speed of convergence of the algorithm, once can treat w_{ij} as design parameters and choose them properly for faster convergence. In this section, we state several important strategies used in networks and give their mathematical expressions.

4.1.1. Random Walk Method

Random walk (RW) weighting method is a method that expresses the transition from a selected agent on a graph G to a random one of the neighbors of the agent with equal probability. In this method, weighting coefficients are defined as follows [49],

$$w_{ij} = \begin{cases} \frac{1}{d_i}, & if\ (v_j, v_i) \in \mathcal{E} \\ 0, & otherwise \end{cases} \tag{4.3}$$

where d_i represents the number of neighbors of agent i. Note that the agents do not have self-loops ($w_{ii} = 0$ for all i) in this method.

4.1.2. Equal Weighting Method

In this method, it is assumed that agents weight the solutions coming from their neighbors and their own solution equally. The mathematical expression of the weighting coefficients in the equal weighting (EW) method is given below [50].

$$w_{ij} = \begin{cases} \frac{1}{d_i+1}, & if\ \ (v_j, v_i) \in \mathcal{E}\ and\ i = j \\ 0, & otherwise \end{cases} \tag{4.4}$$

4.1.3. Degree Neighbor Method

In the Degree Neighbor (DN) weighting method, the solutions produced by an agent with many neighbors are used in the solution algorithms of their neighbors with a greater weighting coefficient. In applications where degree neighbor method is used, it is assumed that the greater the information flow on an agent, the more knowledge the agent has on the network. In this method, the weighting coefficients are selected as follows [51].

$$w_{ij} = \begin{cases} \frac{d_j}{\sum_{l \in N_i \cup \{i\}} d_l}, & if\ \ (V_j, V_i) \in \mathcal{E}\ and\ i = j \\ 0, & otherwise \end{cases} \tag{4.5}$$

4.1.4. Metropolis-Hastings Method

In the Metropolis–Hastings method, the transition probability between two connected, unique vertices is the conjugate of the larger degree, and in order to ensure that the sum of all probabilities at each vertex is 1, the self-loop probabilities are chosen properly.

Metropolis Hastings (M-H) weighting coefficients on a graph G are defined as follows [52],

$$w_{ij} = \begin{cases} min\left\{\frac{1}{d_i}, \frac{1}{d_j}\right\}, & if\ (v_j, v_i) \in \mathcal{E}\ and\ i \neq j \\ \sum_{(i,k) \in \mathcal{E}} max\left\{0, \frac{1}{d_i} - \frac{1}{d_k}\right\}, & if\ i = j \\ 0, & if\ \ (v_j, v_i) \notin \mathcal{E} \end{cases} \tag{4.6}$$

4.2. Simulation Results

In this section, the convergence rate of the given algorithm is investigated for different weighting methods and different types of graphs. Network topologies used in the simulation studies are given in Figure4.1 whose application areas were discussed in Section2.2.

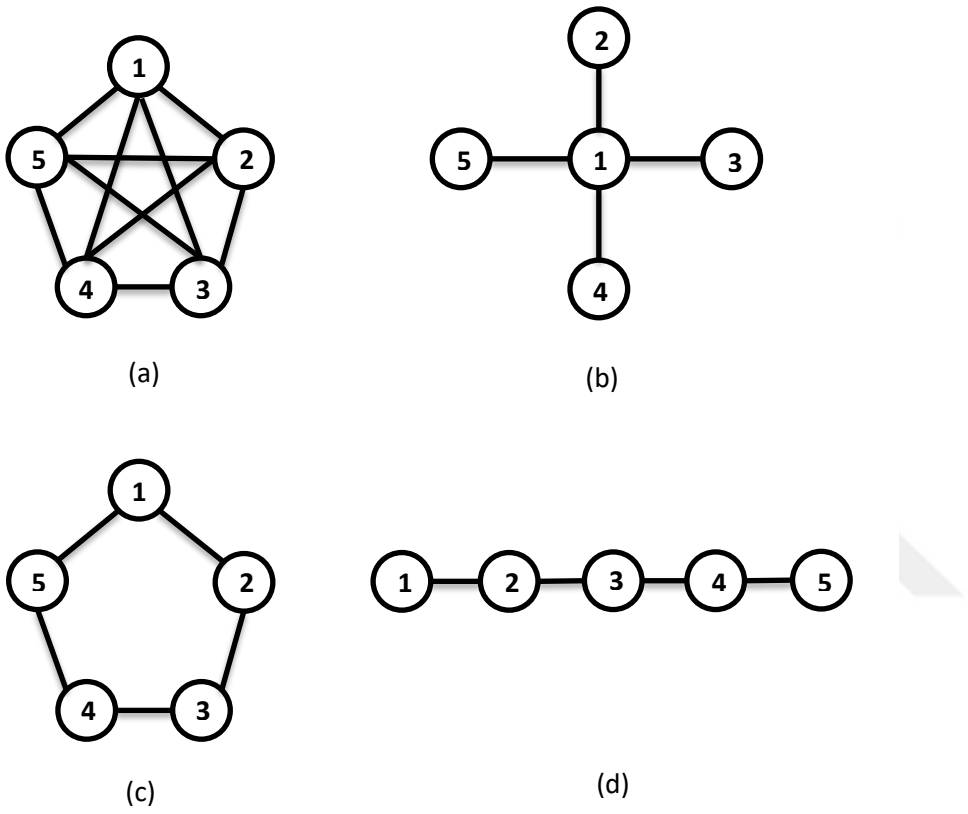

Figure 4.1. Undirected network topologies under consideration a) the fully connected network b) the star network c) the ring network d) the path network. Retrieved from [53].

4.2.1. Example Equation System

Consider the following set of equations with 5 unknowns which is to be solved by the algorithm (4.1).

$$\begin{aligned} x_1 + 2x_2 - 3x_3 + x_4 + x_5 &= 5 \\ 2x_1 - x_2 - 2x_3 &= -6 \\ 2x_2 + 2x_3 + x_4 &= 14 \\ x_1 - x_3 + 5x_4 + 2x_5 &= 28 \\ x_1 + 2x_2 + x_3 + 2x_4 + x_5 &= 21 \end{aligned} \tag{4.7}$$

Note that unique solution of the equation system can be computed as $x^* = [1,2,3,4,5]^T$. It is assumed that each agent in the network knows only a single equation of the system in (4.7). At $k = 0$, each agent starts by generating an initial solution to its own equation. For simulations, we assume that agents select their initial solutions as: $x_1(0) = 2.5e\,, x_2(0) = 6e\,, x_3(0) = 2.8e\,, x_4(0) = 4e\,, x_5(0) = 3e$ where $e = [1,1,1,1,1]^T$.

Figure 3.2. shows how agent 1 reaches the solution of the equation $Ax = b$ by using (4.1).

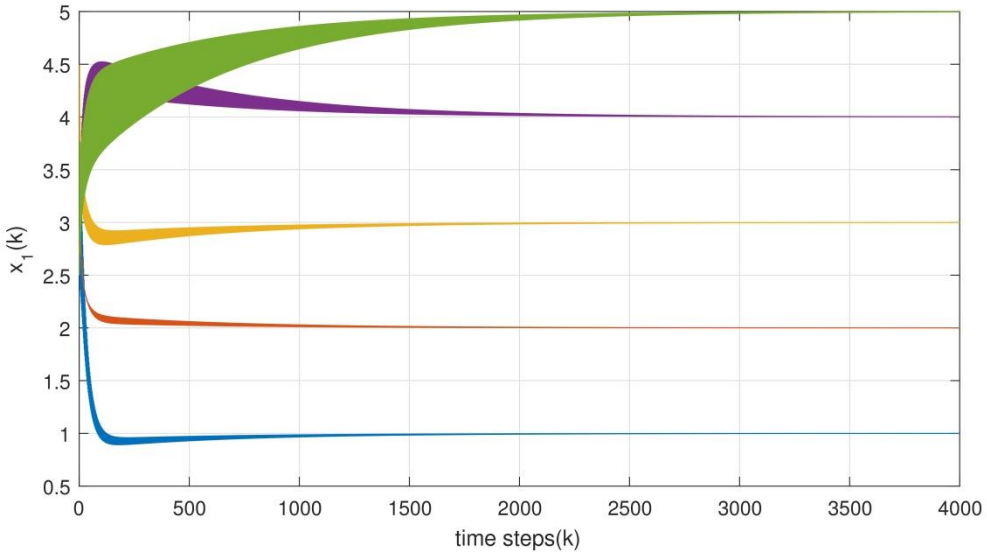

Figure 4.2. The evolution of $x_1(k)$, when the algorithm proposed in [1] is used.

Recall from Section3.2 that since the solution is asymptotic, one can consider the equation is approximately solved if the sum of the norms of errors,

$$h(k) = \sum_{i=1}^{n} \|x_i(k) - x^*\|_2 \tag{4.8}$$

is below a chosen threshold. Therefore, we will assume throughout this section that the equations are solved by the agents if $h(k) \leq \epsilon$. In particular, the number of steps required to achieve the desired accuracy can be determined as the smallest integer that satisfies $h(k) \leq \epsilon$.

The number of steps k required to reach the solution obtained from the simulation studies are given in Table 1 for different weighting methods and network topologies. The weighting methods that provide the fastest solution for each topology are marked with a star (*).

Table 4.1 Number of time steps required to reach solution for different weighting methods and network topologies.

	Fully connected	**Star**	**Ring**	**Path**
Random Walk	2098*	4159*	2750*	3654*
Equal Weighting	2623	6972	4125	5552
Degree Neighbor	2623	7192	4125	5788
Metropolis-Hastings	2098*	10399	2750*	4415

4.3. Summary of This Chapter

In this chapter, four well-known weighting methods, which can be used by agents to use the data coming from their neighbors, are given. The convergence rate of a distributed algorithm is compared for these weighting scenarios. The simulations show that Random Walk is the best method to solve the given linear equation over all networks under consideration in terms of the convergence rates.

This page is intentionally left blank.

5. CONCLUSIONS AND FUTURE WORK

Since systems of linear equations have many applications in diverse fields including engineering, economy and science, it is of great importance to solve these systems quickly and effectively. For this purpose, researchers have proposed various centralized and decentralized methods in the literature. In this thesis, a considerable amount of methods are explained, and in particular, the distributed solution of linear equation systems in discrete time is discussed. In order to solve such systems, several distributed algorithms are proposed which are given in the previous chapters. Although there are slight differences between these algorithms, they all seek to reach a consensus in agents' states. Some basic concepts are the same for all, although the necessary and sufficient conditions for the operation of these algorithms vary from algorithm to algorithm. These conditions are discussed throughout the thesis.

The algorithm proposed in [1] is used in simulations and its convergence rate is examined. It is seen from the simulation studies that with the proposed rapid method, a faster convergence is possible. This method is based on the idea of estimating neighbors' equations by utilizing the solution vectors of the neighbors. Even if agents do not share their equations with their neighbors for reasons such as security and privacy concerns, these equations can be estimated by using mechanism proposed in Section 3.2. We have shown that by using that method, the algorithm converges faster for both delay-free systems and the systems with delay.

Another contribution of the thesis is to investigate the effect of weighting strategies on the convergence rate of the distributed algorithm for solving linear equations over networks with different types of topologies. It is concluded from the simulations that, Random Walk is the fastest method of weighting for the linear equation under consideration.

Obviously, distributed solutions of linear equation systems will remain a popular area of research for years to come. In future studies, the proposed method will be put in a more compact and applicable form. In addition, a direction for future research is studying the theory of this method and providing the necessary proofs.

This page is intentionally left blank.

Bibliography

[1] S. Mou and A. S. Morse, "A Fixed-Neighbor, Distributed Algorithm for Solving a Linear Algebraic Equation," in *European Control Conference (ECC)*, Zurich, 2013.

[2] R. Mehmood and J. Crowcroft, "Parallel iterative solution method for large sparse linear equation systems," University of Cambridge, Cambridge, 2005.

[3] W. A. Smith, Elementary Numerical Analysis, Harper & Row, 1979.

[4] S. Mou, J. Liu and A. S. Morse, "A Distributed Algorithm for Solving a Linear Algebraic Equation," *IEEE Transactions on Automatic Control* , vol. 60, no. 11, pp. 2863 - 2878, 2015.

[5] M. Ylinen, A. Burian and J. Takala, "Updating matrix inverse in fixed-point representation: direct versus iterative methods," in *International Symposium on System-on-Chip*, Tampere, Finland, 2003.

[6] F. Soleymani, "A Rapid Numerical Algorithm to Compute Matrix Inversion," *International Journal of Mathematics and Mathematical Sciences*, vol. 2012, p. 11 pages, 2012.

[7] D. DasGupta, "In-Place Matrix Inversion by Modified Gauss-Jordan Algorithm," *Applied Mathematics*, vol. 4, no. 10, pp. 1392-1396, 2013.

[8] M. K. Razavi, A. Kerayechian, M. Gachpazan and S. Shateyi, "A New Iterative Method for Finding Approximate Inverses of Complex Matrices," *Abstract and Applied Analysis*, vol. 2014, p. 7 pages, 2014.

[9] F. Soleymani, "On a Fast Iterative Method for Approximate Inverse of Matrices," *Commun. Korean Math. Soc*, vol. 28, no. 2, p. 407–418, 2013.

[10] P. Wang, S. Mou, J. Lian and W. Ren, "Solving a system of linear equations: From centralized to distributed algorithms," *Annual Reviews in Control*, vol. 47, no. 8, pp. 306-322, 2019.

[11] M. Yang, "A Distributed Algorithm for Solving General Linear Equations over Networks," in *54th IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, 2015.

[12] B. D. O. Anderson, S. Mou, A. S. Morse and U. Helmke, "Decentralized Gradient Algorithm for Solution of a Linear Equation," *Numerical Algebra Control & Optimization*, vol. 6, no. 3, pp. 319-328, 2016.

[13] J. Liu, X. Chen, T. Başar and A. Nedic, "A Continuous-Time Distributed Algorithm for Solving Linear Equations," in *American Control Conference (ACC)*, Boston, 2016.

[14] G. Shi and B. D. O. Anderson, "Distributed Network Flows Solving Linear Algebraic Equations," in *American Control Conference (ACC)*, Boston, 2016.

[15] G. Shi, B. D. O. Anderson and U. Helmke, "Network Flows That Solve Linear Equations,"

IEEE Transactions on Automatic Control, vol. 62, no. 6, pp. 2659 - 2674, 2017.

[16] L. Yang, C. Lageman, B. D. O. Anderson and G. Shi, "Exponantial Least Squares Solvers for Linear Equations over Networks," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 2543-2548, 2017.

[17] K. Cao, X. Zeng and Y. Hong, "Continuous-Time Distributed Algorithms for Solving Linear Algebraic Equation," in *36th Chinese Control Conference (CCC)*, Dalian, China, 2017.

[18] S. Mou, J. Liu and A. S. Morse, "A Distributed Algorithm for Solving a Linear Algebraic Equation," in *Fifty-first Annual Allerton Conference* , Illinois, USA, 2013.

[19] S. Mou , A. S. Morse , Z. Lin, L. Wang and D. Fullmer, "A Distributed Algorithm for Efficiently Solving Linear Equations," in *IEEE 54th Annual Conference on Decision and Control (CDC)* , Osaka, Japan, 2015.

[20] X. Wang and S. Mou, "Improvement of a Distributed Algorithm for Solving Linear Equations," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 4, pp. 3113-3117, 2017.

[21] H.-T. Cao, T. E. Gibson, S. Mou and Y.-Y. Liu, "Impacts of Network Topology on the Performance of a Distributed Algorithm Solving Linear Equations," in *IEEE 55th Conference on Decision and Control (CDC)* , Las Vegas, USA, 2016.

[22] S. S. Alaviani and N. Elia, "A Distributed Algorithm for Solving Linear Algebraic Equations Over Random Networks," in *IEEE Conference on Decision and Control (CDC)* , Miami Beach, FL, USA, 2018.

[23] J. Liu , S. Mou and A. . S. Morse, "An Asynchronous Distributed Algorithm for Solvinga Linear Algebraic Equation," in *52nd IEEE Conference on Decision and Control* , Florence, Italy, 2013.

[24] P. Wang, W. Ren and Z. Duan, "Distributed Solution to Linear Equations from Arbitrary Initializations," in *American Control Conference* , Seattle, USA, 2017.

[25] X. Wang, S. Mou and D. Sun, "Further Discussions on a Distributed Algorithm for Solving Linear Algebra Equations," in *American Control Conference*, Seattle, USA, 2017.

[26] X. Gao, J. Liu and T. Başar, "Stochastic Communication-Efficient Distributed Algorithms for Solving Linear Algebraic Equations," in *IEEE Conference on Control Applications (CCA)* , Buenos Aires, Argentina, 2016.

[27] X. Wang, J. Zhou, S. Mou and M. J. Corless, "A Distributed Algorithm for Least Square Solutions of Linear Equations," 2017.

[28] T. Yang, J. George, J. Qin, X. Yi and J. Wu, "Distributed Finite-time Least Squares Solver for Network Linear Equations," 2018.

[29] J. Lei, P. Yi, G. Shi and B. D. O. Anderson, "Network Linear Equations with Finite Data Rates," in *IEEE Conference on Decision and Control (CDC)* , Miami Beach, FL, USA,

2018.

[30] X. Wang, J. Zhou, S. Mou and M. J. Corless, "A Distributed Linear Equation Solver for Least Square Solutions," in *IEEE 56th Annual Conference on Decision and Control (CDC)* , Melbourne, Australia, 2017.

[31] X. Wang, J. Zhou, S. Mou and M. J. Corless, "A Distributed Algorithm for Least Squares Solutions," *IEEE Transactions on Automatic Control*, 2019.

[32] G. Strang, Introduction to Linear Algebra, Cambridge: Cambridge Press, 2009.

[33] G. Aggarwal, "EE263: Introduction to Linear Dynamical Systems," [Online]. Available: http://ee263.stanford.edu/lectures/lin-fcts.pdf. [Accessed 3 July 2019].

[34] G. Aggarwal, "EE263: Introduction to Linear Dynamical Systems," [Online]. Available: http://ee263.stanford.edu/lectures/interpretations.pdf. [Accessed 3 July 2019].

[35] G. Aggarwal, "EE263: Introduction to Linear Dynamical Systems," [Online]. Available: http://ee263.stanford.edu/lectures/engineering_examples.pdf. [Accessed 3 July 2019].

[36] J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, New York: Elsevier Science Publishing, 1982.

[37] K. Ruohonen, "Personal Web Site," [Online]. Available: http://math.tut.fi/~ruohonen/GT_English.pdf. [Accessed 5 July 2019].

[38] D. Guichard, "Mathematics and Statistics," [Online]. Available: https://www.whitman.edu/mathematics/cgt_online/cgt.pdf. [Accessed 5 July 2019].

[39] C. S. R. Murthy and B. S. Manoj , Ad Hoc Wireless Networks, PRENTICE HALL, 2004.

[40] S. S. Meena and J. Manikandan, "Study and Evaluation of Different Topologies in Wireless Sensor Network," in *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, India, 2017.

[41] D. C. Lay, S. R. Lay and J. J. McDonald, Linear Algebra and Its Applications, Pearson Education, Inc., 2016.

[42] G. Aggarwal, "EE263: Introduction to Linear Dynamical Systems," [Online]. Available: http://ee263.stanford.edu/lectures/min-norm.pdf. [Accessed 6 July 2019].

[43] K. Hoffman, Linear Algebra, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1971.

[44] V. D. Blondel, J. M. Hendrickx, A. Olshevsky and J. N. Tsitsiklis, "Convergence in Multiagent Coordination, Consensus, and Flocking," in *44th IEEE Conference on Decision and Control*, Seville, Spain, 2005.

[45] J. N. Tsitsiklis, *Problems in Decentralized Decision Making and Computation*, Massachusetts: Massachusetts Institute of Thecnology, 1984.

[46] R. Olfati-Saber and R. M. Murray, "Consensus Problems in Networks of Agents With Switching Topology and Time-Delays," *IEEE Transactions on Automatic Control*, vol. 49,

no. 9, pp. 1520-1533, 2004.

[47] F. Xiao and L. Wang, "State consensus for multi-agent systems with switching topologies and time-varying delays," *International Journal of Control,* vol. 79, no. 10, p. 1277–1284, 2006.

[48] M. Cao, A. S. Morse and B. D. O. Anderson, "Reaching an Agreement Using Delayed Information," in *45th IEEE Conference on Decision and Control*, San Diego, CA, USA, 2006.

[49] D. A. Levin, Y. Peres and E. L. Wilmer, Markov Chains and Mixing Times, American Mathematical Society, 2009.

[50] A. Olshevsky and J. N. Tsitsiklis, "Convergence Speed in Distributed Consensus and Averaging," *SIAM Review,* vol. 53, no. 4, p. 747–772, 2011.

[51] F. Mirali and H. Werner, "Distributed Weighting Strategies for Improved Convergence Speed of First-Order Consensus," in *American Control Conference (ACC)*, Seattle, USA, 2017.

[52] S. Boyd, P. Diaconis and L. Xiao, "Fastest Mixing Markov Chain on a Graph," *SIAM REVIEW*, vol. 46, no. 4, p. 667–689, 2004.

[53] O. M. Duru and O. Cihan, "Effect of Weighting Strategies on the Convergence Rate of a Distributed Algorithm for Solving Linear Equations," in *TOK*, Istanbul, 2017.

CURRICULUM VITAE

Name Surname **:** Osman Mecnun DURU

Place and Date of Birth **:** İstanbul-08.09.1991

E-Mail **:** osmanduru59@gmail.com

EDUCATION **:**

B.Sc. **:** 2014, Karadeniz Technical University, Faculty of Engineering, Electrical and Electronics Engineering

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

Conference Presentations: *Effect of Weighting Strategies on the Convergence Rate of a Distributed Algorithm for Solving Linear Equations.* **Duru, Osman Mecnun and Cihan, Onur.** Istanbul : TOK, 2017.