Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

DIPLOMARBEIT

Nonlinear and Dynamic Average Consensus Algorithms

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplomingenieurs

unter der Leitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gerald Matz
Projektass. Dipl.-Ing. Valentin Schwarz
Institute of Telecommunications

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von
Raman Jafroudi
Portnergasse 19/8/34
1220 Wien

Wien, im September 2011 ____________________

—— To my parents ——

Abstract

This thesis deals with distributed averaging methods in wireless sensor networks. To this end, wireless sensor nodes are equipped with dedicated sensing, computing, and communication devices. Our goal is to calculate the average of measurements through the average consensus (AC).

To provide the reader with the necessary prerequisites, we summarize graph theory and different network topologies in the first part of this work. We then discuss static and dynamic variants of the AC algorithm. The static algorithm is used if all sensors measure constant signals that do not change over time. In the case where the sensors measure time-varying signals, the dynamic algorithm is used. For both the static and the dynamic case, we consider different linear and nonlinear AC weight design methods to improve performance in transient and stationary scenarios. We further study the behavior of the various versions of AC in different network topologies and different simulation settings.

Our numerical results suggest that nonlinear AC is superior to linear AC and that dynamic AC succeeds in tracking rapidly varying signal means.

Kurzfassung

Diese Diplomarbeit beschäftigt sich mit verteilten Mittelungsverfahren in drahtlosen Sensornetzwerken. Drahtlose Sensorknoten bestehen aus Komponenten für die Messung, Berechnung und Kommunikation von Daten. Unser Ziel ist es, den Mittelwert der Messwerte mit dem *Average Consensus (AC)* Algorithmus zu berechnen.

Um dem Leser das nötige Vorwissen zu vermitteln, geben wir im ersten Teil dieser Arbeit eine Einführung in die Graphentheorie und verschiedene Netzwerktopologien. Danach diskutieren wir eine statische und eine dynamische Variante des AC-Algorithmus. Die statische Mittelwertbildung wird verwendet, wenn die Sensoren konstante Signale messen, die sich nicht über die Zeit ändern. Im Fall wo die Sensoren zeitlich veränderliche Signale messen, wird die dynamische Mittelwertbildung verwendet. Sowohl für den statischen als auch für den dynamischen Fall betrachten wir verschiedene lineare und nichtlineare Methoden für den Entwurf der AC-Gewichte, um die Leistungsfähigkeit in transienten und stationären Szenarien zu verbessern. Wir untersuchen weiters das Verhalten der unterschiedlichen AC-Versionen in verschiedenen Netzwerktopologien und für verschiedene Simulationsparameter.

Unsere numerischen Ergebnisse legen den Schluss nahe, dass nichtlineare AC-Algorithmen den linearen Verfahren überlegen sind und dass dynamische AC-Verfahren schnell zeitvariante Signalmittelwerte erfolgreich schätzen können.

Acknowledgements

I would like to express my sincerest gratitude to Gerald Matz for giving me the opportunity to work on this thesis and for his continuous encouragement and devoted guidance.

I would also like to gratefully acknowledge the supervision of Valentin Schwarz, who has supported me with patient guidance, great kindness, and friendly assistance during this thesis.

Finally, I take this opportunity to express my profound gratitude to my beloved parents, Azita and Fereydoon for their support and patience during my study.

Contents

1 Introduction 1

2 Prerequisite 3

2.1 Introduction . . . 3
2.2 Graph Theory . . . 4
2.2.1 Definitions . . . 4
2.2.2 Adjacency Matrix . . . 9
2.2.3 Neighbor Set . . . 11
2.2.4 Degree Matrix . . . 12
2.2.5 Incidence Matrix . . . 14
2.2.6 Laplacian Matrix . . . 16
2.3 Network Topologies . . . 19
2.3.1 Random Geometric Graph . . . 19
2.3.2 Random Regular Graph . . . 20
2.3.3 Ring Graph . . . 20
2.3.4 Complete Graph . . . 21
2.3.5 Graph with Bottleneck . . . 22

3 Static Average Consensus 24

3.1 Introduction . . . 24
3.2 Problem Statement . . . 25
3.3 Linear Static Average Consensus Algorithm . . . 26
3.4 Convergence Conditions . . . 30
3.5 Convergence Speed . . . 32

3.6 Weight Matrix . . . 33
3.6.1 Weight Matrix based on Constant Weights . . . 33
3.6.2 Weight Matrix based on Convex Optimization . . . 36
3.6.3 Weight Matrix based on Metropolis-Hastings algorithm . . . 37
3.6.4 Influence of Eigenvalues of Weight Matrix on Convergence Speed . . . 38
3.7 Nonlinear Static Average Consensus Algorithm . . . 40
3.7.1 Nonlinearity by Varying the Weights of a Weight Matrix . . . 40
3.7.2 Nonlinearity by Combination of two Weight Matrices . . . 42

4 Dynamic Average Consensus 44

4.1 Introduction . . . 44
4.2 Linear Dynamic Average Consensus Algorithm . . . 45
4.2.1 First-Order Linear Dynamic Average Consensus Algorithm . . . 45
4.2.2 nth-Order Linear Dynamic Average Consensus Algorithm . . . 48
4.3 Nonlinear Dynamic Average Consensus Algorithm . . . 49

5 Simulation Results 51

5.1 Simulation Setup . . . 51
5.2 Linear Static Average Consensus Algorithm . . . 52
5.2.1 Influence of Number of Edges on Performance . . . 53
5.2.2 Influence of Number of Nodes on Performance . . . 56
5.2.3 Measurement from a Spatial Field . . . 57
5.2.4 Impact of the Graph Topology . . . 59
5.3 Nonlinear Static Average Consensus Algorithm . . . 64
5.3.1 Determination of Parameters of Nonlinear Function . . . 64
5.3.2 Influence of Number of Edges on Performance . . . 70
5.4 Nonlinear Static Average Consensus Algorithm with Combination of two Weight Matrices . . . 72
5.5 Linear Dynamic Average Consensus Algorithm . . . 75
5.5.1 Measurement from Time-Varying Spatial Field . . . 75
5.5.2 Influence of Frequency of Time-Varying Spatial Field on Performance . 75
5.5.3 Influence of Order of Algorithm on Performance . . . 78
5.6 Nonlinear Dynamic Average Consensus Algorithm . . . 79
5.6.1 Determination of Parameters of Nonlinear Function . . . 79
5.6.2 Measurement from Time-Varying Spatial Field . . . 80
5.6.3 Influence of Frequency of Time-Varying Spatial Field on Performance . 84

5.6.4 Influence of Order of Algorithm on Performance 87

6 Summary 90

Bibliography 92

Notation 93

List of Abbreviations 95

1 Introduction

Wireless sensor networks (WSN) are low-cost sensors which can be distributed over large area. They can accomplish simple computations and local measurements and connect to their neighbors via wireless connections to exchange informations. WSN can be applied in many different areas. Environmental monitoring, surveillance, micro-surgery, and agriculture are only a few examples.

The goal of this work is to calculate the average value of sensor measurements. One possible way is that we connect all sensors to a fusion center which collects the whole sensor measurements and do all computations. The problem of this way is that each wireless sensor has a limited communication range which makes it difficult that all sensors communicate with fusion center directly. We can solve this problem by using relaying techniques or by increasing the transmit power of sensors. But if we have too many sensors which are distributed over a large area, this will be more complicated and yield to an uneconomical method. Because of that it is more advantageous to use distributed averaging methods. We discuss in this thesis the *average consensus (AC) algorithm* as a distributed averaging method. Each sensor has locally access to its measurements and through wireless communication to measurements of its neighbors which are connected with this sensor. With this informations and by applying the AC algorithm each sensor is able to estimate a global quantity which is the average value.

We consider the static and the dynamic variants of AC algorithm. The *Static AC algorithm* is used to calculate the average value of sensor measurements which do not change over different iterations. In case of sensors which measure time-varying signals, we use *dynamic AC algorithm.* For both the static and the dynamic case, we consider different linear and nonlinear AC weight design methods to improve performance.

We model the WSN as graphs to describe them mathematically. We consider mostly *random geometric graphs* which represent wireless networks with limited communication range, but we will also consider the performance of algorithms for some other graphs like *random regular graph*, *Ring Graph*, *Complete Graph*.

This work is structured as follow:

Chapter 2: This chapter gives us a basic introduction in graph theory. Furthermore, different network topologies are discussed here.

Chapter 3: This part of the work deals with static AC algorithm with its linear and nonlinear variants. We discuss the necessary conditions for convergence to average value and the convergence speed. We will also study different AC weight design methods.

Chapter 4: In this chapter we consider the dynamic AC algorithm with its linear and nonlinear variants. The first-order algorithm and nth-order algorithm will be also discussed.

Chapter 5: This chapter shows the most important results of different simulation settings.

Chapter 6: In last part we give a summary of this work.

2

Prerequisite

2.1 Introduction

In this chapter we will consider different definitions of graph theory and their properties which we use in this work. Moreover we explain different network topologies which are applied to illustrate sensor networks. Many of following definitions can be found in [1] and [2].

2.2 Graph Theory

In this section we will consider different definitions of graphs and describe most important properties of them which we use in this work.

2.2.1 Definitions

Directed and Undirected Graphs: A *directed graph* $\mathcal{G}$ is represented by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of *vertices (nodes)* $\mathcal{V}$ and the set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The cardinality $|\mathcal{V}| = N$ corresponds to the number of nodes of the graph. We illustrate nodes by circles or points and the edges as communication links between nodes by directed lines (preferably straight lines). A directed edge (i, j) expresses the propagation from node i to node j but not vice versa.

We see in Fig. 2.1 an example of a directed graph with $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{E} = \{(1, 2), (1, 3), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6), (5, 6), (6, 7)\}$. For example node 3 can be reached from node 1 over the edge $(1, 3)$ or over the edges $(1, 2)$ and then $(2, 3)$. Propagation against the direction of edge is not allowed, for example from node 3 to node 1.

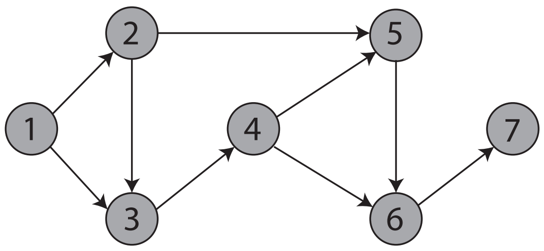


Figure 2.1: *Directed graph.*

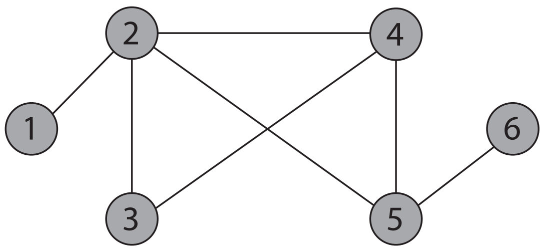


Figure 2.2: *Undirected graph.*

An *undirected graph* is similar to directed graphs except that the edges are undirected lines. An undirected edge $\{i, j\}$ expresses the propagation from node i to node j

and from node j to node i. It can be seen as a two-way street. On the other hand the directed edge (i, j) is like a one-way street, so we have a Propagation just from node i to node j and not vice versa.

In Fig. 2.2 can be seen an example of a undirected graph with $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{E} = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{4, 5\}, \{5, 6\}\}$. For example we can reach node 4 from node 1 over edges $\{1, 2\}$ and $\{2, 4\}$ or over $\{1, 2\}$, $\{2, 3\}$ and $\{3, 4\}$.

Tails and Heads: It is obvious in Fig. 2.3 that a directed edge (i, j) has the start node i and the end node j. It begins from node i and ends at node j, so node i is the *tail* and node j is the *head* of edge (i, j). If we have the edge $(i, j) \in \mathcal{E}$, node j is *adjacent* to node i. The edge (i, j) is incident to nodes i and j.

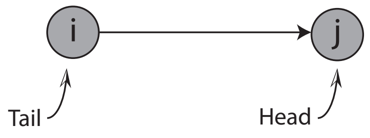


Figure 2.3: *Tail and Head.*

Degrees: The *indegree* of a node is the number of edges that terminate at the node and the *outdegree* of it is the number of edges that emanate from the node. The *degree* of a node is the sum of its indegree and outdegree.

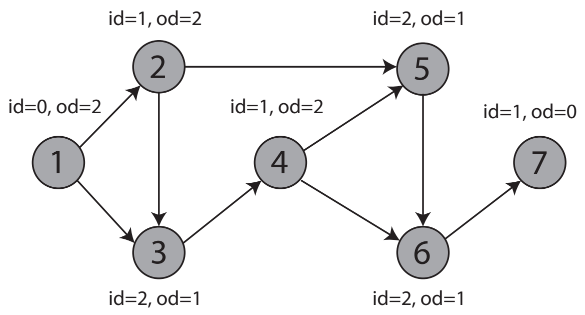


Figure 2.4: *Directed graph outdegree (od) and indegree (id) of each node.*

Fig. 2.4 shows the same graph in Fig. 2.1, but it includes also the indegree and the outdegree of each node. For example the node 2 has an indegree of 1 and an outdegree of 2, the degree of node 2 will be then 3.

The sum of indegrees of all nodes equals to the sum of outdegree of them and they are equal to the number of edges in graph. In Fig. 2.4 the sum of indegree or outdegree is 9 and this is the number of edges.

Multiedges and Loops *Multiedges* are two or more edges that connect same two nodes together. If an edge connects a node with itself, it is called a *loop*. In this work we consider only graphs without loops and multiedges.

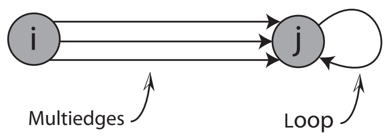


Figure 2.5: *Multiedges and loops.*

Subgraph: We have two graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. The graph $\mathcal{G}'$ is a *subgraph* of $\mathcal{G}$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. In Fig. 2.6(b) we illustrated a subgraph of the graph shown in Fig. 2.6(a) because all nodes and edges which are in Fig. 2.6(b) are all included in Fig. 2.6(a) too.

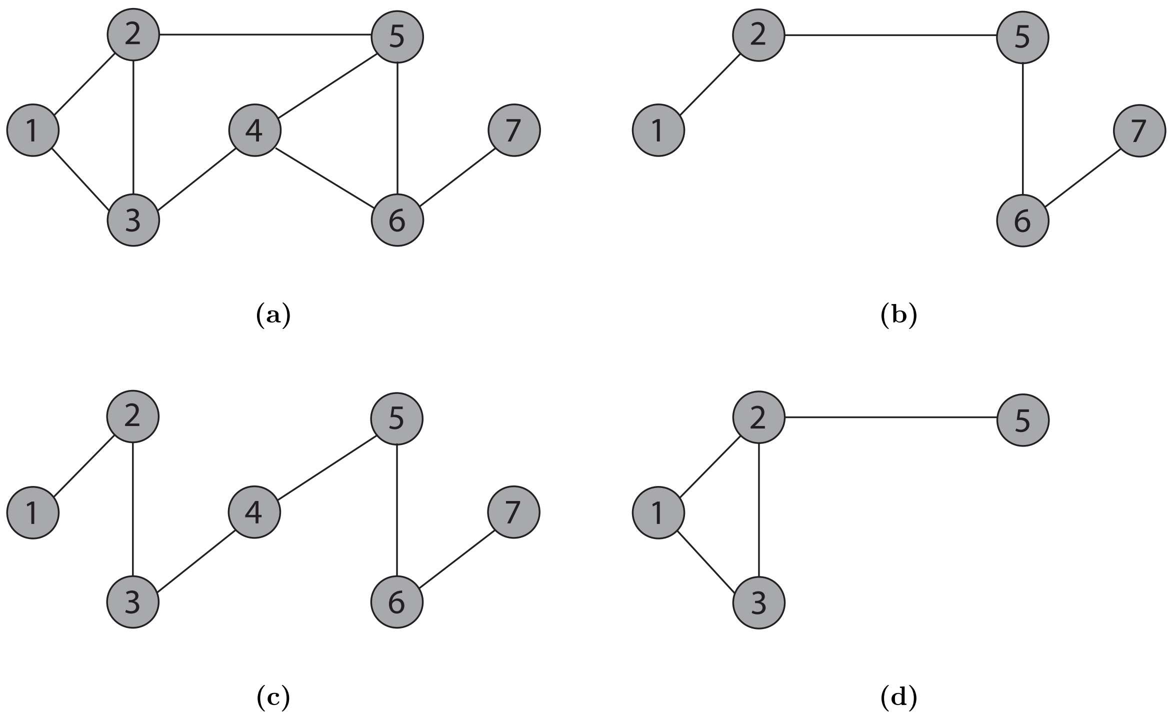


Figure 2.6: *(a) Graph $\mathcal{G}$, (b) subgraph $\mathcal{G}'$, (c) spanning subgraph $\mathcal{G}'$ and (d) subgraph $\mathcal{G}'$ induced by $\mathcal{V}'$.*

The graph $\mathcal{G}'$ is a *spanning subgraph* of $\mathcal{G}$ if $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. It is shown in Fig. 2.6(c) the spanning subgraph $\mathcal{G}'$ of 2.6(a). All nodes that we have in $\mathcal{G}$ are in $\mathcal{G}'$ too and the set of edges $\mathcal{E}' \subseteq \mathcal{E}$.

We say that $\mathcal{G}'$ is the subgraph of $\mathcal{G}$ *induced* by $\mathcal{V}'$ if in edges set $\mathcal{E}'$ we have all possible edges from $\mathcal{E}$ that connect every two nodes from $\mathcal{V}'$. In Fig. 2.6(a) we see the graph $\mathcal{G}$ and in Fig. 2.6(d) the subgraph $\mathcal{G}'$ induced by $\mathcal{V}'$.

Walk: We assume that we have a subgraph of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. This subgraph consists of a sequence of nodes i_k with the start node i_1 and the end node i_r and a sequence of edges e_k with the first edge e_1 and the last edge e_{r-1} (see Fig. 2.7). If for all $1 \leq k \leq r-1$ is either $e_k = (i_k, i_{k+1}) \in \mathcal{E}$ or $e_k = (i_{k+1}, i_k) \in \mathcal{E}$, we have a *walk* in graph $\mathcal{G}$.

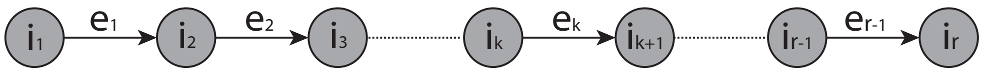


Figure 2.7: *Walk.*

In Fig. 2.8(a) we have the graph $\mathcal{G}$ and the Fig. 2.8(b) and (c) illustrates two different walks in this graph, the first walk is 1-2-5-6 and second one 1-2-4-5-2-3.

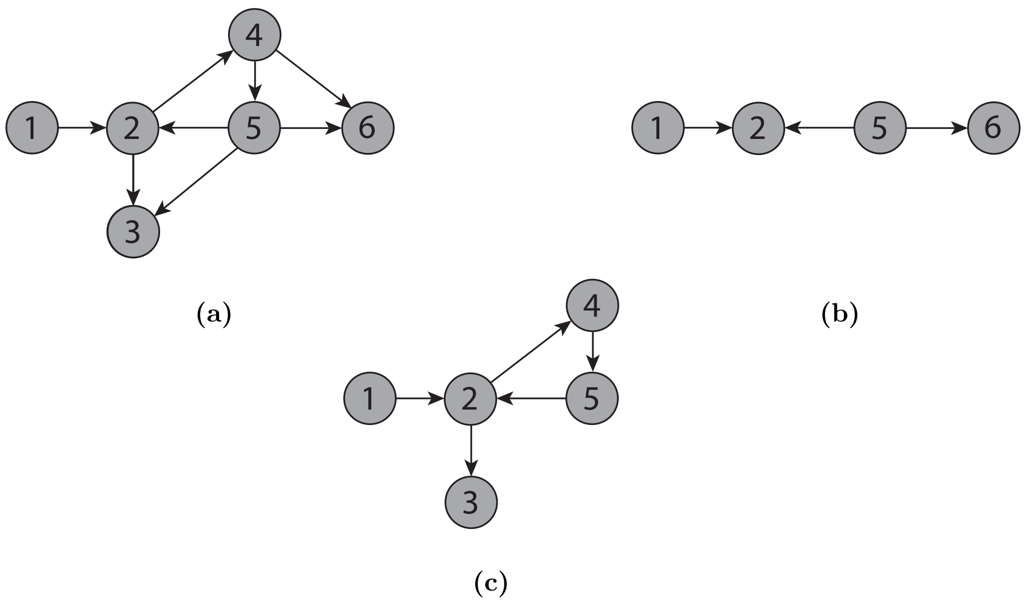


Figure 2.8: *(a) Graph $\mathcal{G}$, (b) and (c) examples of walks in graph $\mathcal{G}$.*

In a *directed walk* we have an "orientation" in comparison with undirected walk. It means for any consecutive nodes i_k and i_{k+1} on the walk, $(i_k, i_{k+1}) \in \mathcal{E}$. The walk in Fig. 2.8(b) is undirected but the walk in 2.8(c) is directed.

Path: A *path* is a walk if we visit each node of the path just once. The walk in Fig. 2.8(b) is a path but the walk in Fig. 2.8(c) is not because we visit the node 2 twice. We have in a path *forward edges* and/or *backward edges*. An edge (i, j) is forward if the path goes over node i before node j and is backward otherwise. For example in Fig. 2.8(b) the edges $(1, 2)$ and $(5, 6)$ are forward and the edge $(5, 2)$ is backward.

A *directed path* is a directed walk if we visit each node of the path just once. So, we do not have any backward edges.

Cycle: A *cycle* is a path such that start node and end node are the same. So, we have the edge (i_r, i_1) or (i_1, i_r) between i_1 the start node and i_r the last node before start node in our path. We have in cycles like paths forward and backward edges. In Fig. 2.9(a) the edge $(5, 3)$ is forward but the edges $(2, 3)$ and $(5, 2)$ are backward in cycle 2-5-3.

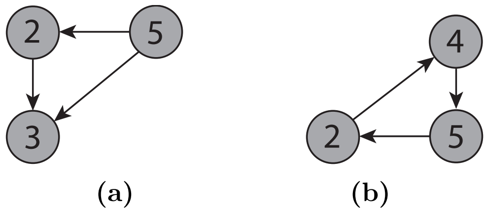


Figure 2.9: *Example of cycles.*

A *directed cycle* is a path with the edge (i_r, i_1) between the end node i_r and node i_1. We do not have any backward edge in a directed cycle. In Fig. 2.9(b) we have a directed cycle because if we assume that node 2 is the start node and node 5 is the last node before start node 2 in our path, we have an edge $(5, 2) \in \mathcal{E}$. The cycle in Fig. 2.9(a) is not a directed cycle. If we do not have any directed cycle in a graph, the graph is *acyclic*.

Connectivity: Two nodes are *connected* if there is a path between them. A graph is connected if there is a path from each node of the graph to all other nodes otherwise the graph is *disconnected*. Each connected subgraph of a graph is a component of the graph. A connected graph is illustrated in Fig. 2.10(a). The graph in Fig. 2.10(b) is disconnected because the edges $(3, 4)$ and $(2, 5)$ are deleted.

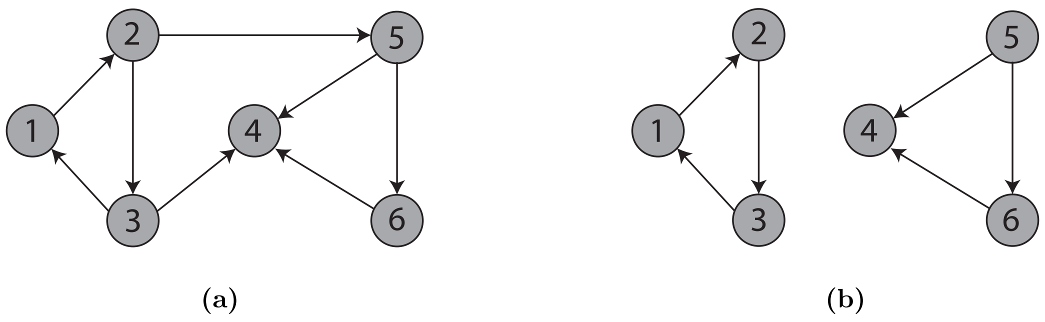


Figure 2.10: *(a) Connected and (b) disconnected graphs.*

A graph is *strongly connected* if there is a "directed" path from each node of the graph to all other nodes. It is obvious that the graph in Fig. 2.10(a) is not strongly connected because we do not have any directed path from node 4 to all other nodes but in Fig. 2.11 we have instead of edge $(3, 4)$ the edge $(4, 3)$, so this graph is strongly connected.

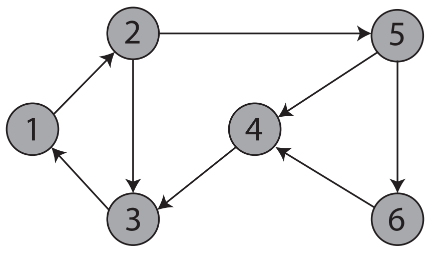


Figure 2.11: *Strongly connected graph.*

2.2.2 Adjacency Matrix

One of the possible ways to represent a graph is the adjacency matrix $\mathbf{A}$. This matrix defines neighbors of each node, so the name of it is adjacency matrix. The adjacency matrix $\mathbf{A}$ is a $N \times N$ matrix with $N = |\mathcal{V}|$ corresponds to the number of nodes of graph. Each element a_{ij} of $\mathbf{A}$ is equal to 1 if there is a directed edge from node j to node i otherwise it is 0.

For a directed graph we define each element of adjacency matrix as follow:

$$a_{ij} = \begin{cases} 1, & (j,i) \in \mathcal{E}, \\ 0, & (j,i) \notin \mathcal{E}. \end{cases} \tag{2.1}$$

For an undirected graph we replace in (2.1) the edge (j,i) by $\{i,j\}$, it leads to a symmetric adjacency matrix because if in undirected graph two nodes i and j are connected, they can communicate in both directions. In this case we define each element of adjacency matrix as follow:

$$a_{ij} = \begin{cases} 1, & \{i,j\} \in \mathcal{E}, \\ 0, & \{i,j\} \notin \mathcal{E}. \end{cases} \tag{2.2}$$

If any diagonal element of $\mathbf{A}$ is equal to 1 it means that we have a loop on this node.

In a directed graph to find the precursor nodes of node i, we should consider the ith row of $\mathbf{A}$. The number of columns of ith row that have elements equal to 1, is the number of precursor nodes of node i. For successor nodes of node i we consider the ith column of $\mathbf{A}$.

In undirected graph to find the neighbor nodes of node i, we can consider the ith row or ith column of $\mathbf{A}$ because the adjacency matrix $\mathbf{A}$ is symmetric and the edges are undirected.

Example 2.1. The graph in Fig. 2.12 that we take into account, is undirected.

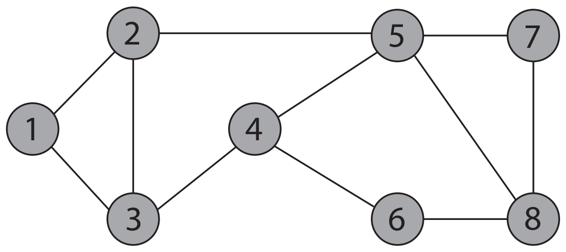


Figure 2.12: *Undirected graph.*

We write the adjacency matrix with help of (2.1) and begin with a_{11}. It is equal to zero because we do not have any loop in the graph. $a_{12} = a_{13} = 1$ because there is an edge between nodes 1 and 2 and between nodes 1 and 3. Due to missing of edges between node 1 and nodes 4, 5, 6, 7, and 8, $a_{14} = a_{15} = a_{16} = a_{17} = a_{18} = 0$. We calculate in the same manner the other elements of $\mathbf{A}$. The 8×8 adjacency matrix $\mathbf{A}$ is then as follow:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

The matrix $\mathbf{A}$ is because of undirected graph symmetric.

2.2.3 Neighbor Set

The neighbor set $\mathcal{N}_i$ defines the neighbor nodes of node i that communicate with it.

In a directed graph the elements of Neighbor Set $\mathcal{N}_i$ are neighbors which send information to node i. It means that the edges between node i and its neighbors which are in the neighbor set $\mathcal{N}_i$, emanates from these neighbors and terminates at node i, so the neighbor nodes are the tails and node i is the head of these edges. We define a neighbor set for a directed graph as below:

$$\mathcal{N}_i = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}. \tag{2.3}$$

In a undirected graph we do not have any direction thus if there is an edge between node i and other nodes, these nodes will be then in neighbor list $\mathcal{N}_i$. We define a neighbor set for a undirected graph as below:

$$\mathcal{N}_i = \{j \in \mathcal{V} : \{i, j\} \in \mathcal{E}\}. \tag{2.4}$$

The (2.4) is similar to (2.3), we replaced only the directed edge (j, i) with the undirected edge $\{i, j\}$.

Example 2.2. In this example we consider again the undirected graph in Fig. 2.12. It is obvious that for example node 1 is connected with node 2 and with node 3 which are its neighbors. We obtain the following neighbor sets for each node:

$$\mathcal{N}_1 = \{2, 3\}, \ \mathcal{N}_2 = \{1, 3, 5\},$$

$$\mathcal{N}_3 = \{1, 2, 4\}, \ \mathcal{N}_4 = \{3, 5, 6\},$$

$$\mathcal{N}_5 = \{2, 4, 7, 8\},\ \mathcal{N}_6 = \{4, 8\},$$
$$\mathcal{N}_7 = \{5, 8\},\ \mathcal{N}_8 = \{5, 6, 7\}.$$

2.2.4 Degree Matrix

The degree matrix $\mathbf{D}$ is a $N \times N$ diagonal matrix with $N = |\mathcal{V}|$ corresponds to the number of nodes.

In directed graph the diagonal elements are equal to indegree, outdegree or degree of each node depending on the application that we use. The indegree of a node is the number of edges that terminate at this node, the outdegree of a node is the number of edges that emanate from this node and the degree of a node is the sum of its indegree and outdegree. We can obtain them as follow:

Indegree:

$$d_{i,\text{in}} = \sum_{j=1}^{N} a_{ij}, \qquad \text{for } i \neq j \text{ and } i = 1 \ldots N. \tag{2.5}$$

Outdegree:

$$d_{i,\text{out}} = \sum_{j=1}^{N} a_{ji}, \qquad \text{for } i \neq j \text{ and } i = 1 \ldots N. \tag{2.6}$$

Degree:

$$d_i = d_{i,\text{in}} + d_{i,\text{out}}, \qquad \text{for } i = 1 \ldots N. \tag{2.7}$$

a_{ij} (or a_{ji}) is the element of ith row and jth column (or jth row and ith column) of adjacency matrix $\mathbf{A}$.

In undirected graph we define the diagonal elements equal to the number of connected neighbors of each node thus the sum of row or column of adjacency matrix. It is equal to number of elements of Neighbor set $\mathcal{N}_i$. The degree of node i is as below:

$$d_i = |\mathcal{N}_i| = \sum_{j=1}^{N} a_{ij} = \sum_{j=1}^{N} a_{ji}, \qquad \text{for } i \neq j. \tag{2.8}$$

The degree matrix $\mathbf{D}$ is then as follow:

$$\mathbf{D} = \begin{bmatrix} d_1 & 0 & . & . & . & 0 \\ 0 & d_2 & 0 & . & . & . \\ . & 0 & . & & & . \\ . & & & . & & . \\ . & & & & . & 0 \\ 0 & . & . & . & 0 & d_N \end{bmatrix}. \tag{2.9}$$

Example 2.3. We consider again the undirected graph which is shown in Fig. 2.12 and calculate the sum of each row of adjacency matrix

$$\mathbf{A} = \underbrace{\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\begin{bmatrix} 2 & 3 & 3 & 3 & 4 & 2 & 2 & 3 \end{bmatrix}}.$$

In Example 2.2 we have the neighbor set of each node, so we calculate the cardinally of each Neighbor set $|\mathcal{N}_i|$ and we get the following values:

$$|\mathcal{N}_1| = 2,\ |\mathcal{N}_2| = 3,\ |\mathcal{N}_3| = 3,\ |\mathcal{N}_4| = 3,$$

$$|\mathcal{N}_5| = 4,\ |\mathcal{N}_6| = 2,\ |\mathcal{N}_3| = 2,\ |\mathcal{N}_8| = 3.$$

It is obvious that the sum of each column of adjacency matrix is equal to number of elements of neighbor set of each node, so the (2.8) is satisfied. With help of Equations

(2.8) and (2.9) we calculate the 8×8 degree matrix:

$$\mathbf{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

2.2.5 Incidence Matrix

Another possible way to represent a graph is the incidence matrix $\mathbf{B}$. It is a $N \times M$ matrix with N number of nodes and M number of edges. Each row of $\mathbf{B}$ corresponds to one node and each column to one edge. We consider the incidence matrix for two cases: directed and undirected graph.

In a directed graph each element b_{il} of $\mathbf{B}$ is equal to 1 if node i and edge l are incident and l emanates from i. The element b_{il} is equal to -1 if node i and edge l are incident and l terminates at i. If i and l are not incident, b_{il} will be zero. We define each element of incidence matrix as follow:

$$b_{il} = \begin{cases} 1, & \text{if edge } l \text{ emanates from node } i, \\ -1, & \text{if edge } l \text{ terminates at node } i, \\ 0, & \text{otherwise.} \end{cases} \tag{2.10}$$

In a undirected graph we have two kinds of incidence matrices: *unoriented* and *oriented*. In an unoriented incidence matrix each element b_{il} of $\mathbf{B}$ is equal to 1 if node i and edge l are incident. If they are not incident, b_{il} will be zero. We define each element of incidence matrix as follow:

$$b_{il} = \begin{cases} 1, & \text{if edge } l \text{ and node } i \text{ are incident,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.11}$$

For an oriented incidence matrix, we assign arbitrarily a reference direction for each edge in the undirected graph which leads to a directed graph and then write the incidence matrix in sense of the matrix elements in (2.10).

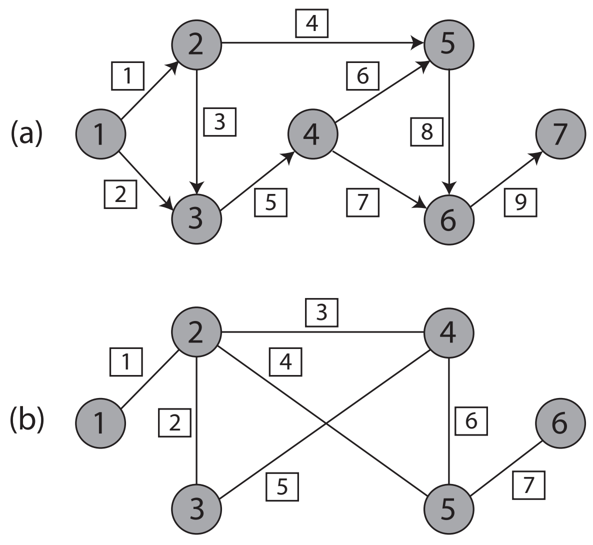


Figure 2.13: *(a) Directed and (b) undirected graph.*

Example 2.4. We consider the directed graph in Fig. 2.13(a) (this is the Fig. 2.1 with defined edge number). This graph has 7 nodes and 9 edges, so the incidence matrix $\mathbf{B}$ has a dimension of 7×9. We write the incidence matrix with help of (2.10). For example the row 2 corresponds to node 2. This node is incident with edges 1, 3 and 4. The edges 3 and 4 emanate from node 2 and the edge 1 terminates at node 2. So, the $b_{23} = b_{24} = 1$ and $b_{21} = -1$. Node 2 and the other edges 2, 5, 6, and 7 are not incident, so $b_{22} = b_{25} = b_{26} = b_{27} = 0$. To calculate other elements of incidence matrix we use the same manner. The incidence matrix $\mathbf{B}$ is as follow:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

In Fig. 2.13(b) is an undirected graph illustrated (this is the Fig. 2.2 so that the edges are numbered). We calculate the incidence matrix with help of (2.11). There are here 6 nodes and 7 edges, so the incidence matrix is a 6×7 matrix. For example the node 2 is incident with edges 1,2,3, and 4, so $b_{21} = b_{22} = b_{23} = b_{24} = 1$ but it is not incident to any other edges, so $b_{25} = b_{26} = b_{27} = b_{28} = b_{29} = 0$. The incidence matrix

B is as follow:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

2.2.6 Laplacian Matrix

Another possible representation of a graph is the Laplacian matrix **L**. It is a $N \times N$ matrix with $N = |\mathcal{V}|$ corresponds to the number of nodes.

In a directed graph we illustrate each element l_{ij} of Laplacian matrix as follow:

$$l_{ij} = \begin{cases} d_{i,in}, & \text{if } i = j, \\ -1, & \text{if } (j,i) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.12}$$

It is obvious that diagonal elements of **L** are indegree $d_{i,in}$ of node i. If $(j,i) \in \mathcal{E}$, the element l_{ij} is equal to -1. This is the same value of same element of adjacency matrix multiplied by -1.

For an undirected graph we replace in (2.12) the edge (j,i) by $\{i,j\}$, it leads to a symmetric Laplacian matrix because if we have an undirected edge between node i and j, this nodes can communicate with each other in both directions. In a undirected graph each element l_{ij} of Laplacian matrix is defined as follow:

$$l_{ij} = \begin{cases} d_i, & \text{if } i = j, \\ -1, & \text{if } \{i,j\} \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.13}$$

The Laplacian matrix is defined as difference of the degree matrix **D** and the adjacency **A**:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{2.14}$$

The Laplacian Matrix in an undirected graph could be also calculated from the oriented incidence matrix **B**,

$$\mathbf{L} = \mathbf{B}\mathbf{B}^T. \tag{2.15}$$

We consider now some spectral properties of Laplacian matrix. The sum of each row of Laplacian matrix is zero, thus vector $\mathbf{1}$ is the right eigenvector of $\mathbf{L}$:

$$\mathbf{L}\mathbf{1} = 0. \tag{2.16}$$

Another property of Laplacian matrix is that If $\mathbf{L}$ is a Laplacian matrix of an undirected graph, It is a positive semi-definite matrix, i.e., for all $\mathbf{x} \in \mathbb{R}^N$,

$$\mathbf{x}^T\mathbf{L}\mathbf{x} \geq 0. \tag{2.17}$$

There are N eigenvalues of $\mathbf{L}$ which are all real and we can order the eigenvalues as follow:

$$0 = \lambda_N(\mathbf{L}) \leq \lambda_{N-1}(\mathbf{L}) \leq \ldots \leq \lambda_1(\mathbf{L}) \leq 2\Delta \tag{2.18}$$

where $\lambda_i(.)$ denotes the ith largest eigenvalue of a matrix and $\Delta = \max_i d_i$ is the maximum degree of a graph. The first smallest eigenvalue of $\mathbf{L}$ is the zero eigenvalue $\lambda_N(L)$=0, it is the trivial eigenvalue of $\mathbf{L}$. The second smallest eigenvalue $\lambda_{N-1}(L)$ is called *algebraic connectivity*. If $\lambda_{N-1}(L)$ is greater than 0, our graph is connected. Generally number of eigenvalues of $\mathbf{L}$ that are zero, define number of components of graph which are disconnected. We can define an upper bound for the sum of eigenvalues of $\mathbf{L}$:

$$\sum_{i=1}^{N} \lambda_i = \sum_{j=1}^{N} d_i \leq \Delta N. \tag{2.19}$$

Example 2.5. In this example we consider the undirected graph in Fig. 2.12. In Examples 2.1 and 2.3 are the adjacency matrix $\mathbf{A}$ and the degree matrix $\mathbf{D}$ calculated. With help of (2.14) we calculate the Laplacian matrix

$$\mathbf{L} = \mathbf{D} - \mathbf{A} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 \end{bmatrix}.$$

There are due to 8 different nodes in graph ($N = 8$), 8 eigenvalues of $\mathbf{L}$:

$$\lambda_1 = 5.72,\ \lambda_2 = \lambda_3 = 4,\ \lambda_4 = 3.34,$$
$$\lambda_5 = 2.68,\ \lambda_6 = 1.54,\ \lambda_7 = 0.7,\ \lambda_8 = 0.$$

It is obvious that algebraic connectivity $\lambda_7 = 0.7$ is greater than zero, so our graph is connected. The graph in Fig. 2.12 has $\Delta = \max_i d_i = 4$, we see that the (2.18) satisfies:

$$\lambda_8 = 0 \leq \lambda_7 = 0.7 \leq \lambda_6 = 1.54 \leq \lambda_5 = 2.68 \leq \lambda_4 = 3.34 \leq \lambda_3 = \lambda_2 = 4 \leq \lambda_1 = 5.72 \leq 8.$$

Example 2.6. The disconnected graph in Fig. 2.14 is considered here.

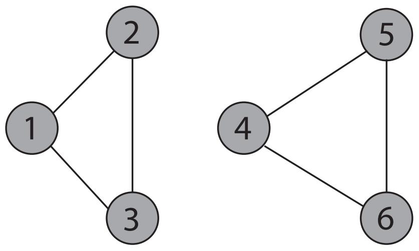


Figure 2.14: *Disconnected graph.*

We have 6 nodes, so the adjacency matrix $\mathbf{A}$, degree matrix $\mathbf{D}$, and Laplacian matrix $\mathbf{L}$ are 6×6 matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

$$\mathbf{L} = \mathbf{D} - \mathbf{A} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}.$$

There are due to 6 different nodes in graph ($N = 6$), 6 eigenvalues of $\mathbf{L}$:

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 3,\ \lambda_5 = \lambda_6 = 0.$$

We see that two eigenvalues λ_5 and λ_6 are equal to zero, so this graph is disconnected and has two components.

2.3 Network Topologies

In this section we will consider different network topologies and their properties that we use in this work to represent sensor networks.

2.3.1 Random Geometric Graph

A random geometric graph is a graph $\mathcal{G}(N, r)$. There are N nodes which are randomly and uniformly distributed on the unit square. Two nodes are connected if their Euclidean distance is at most the radius r.

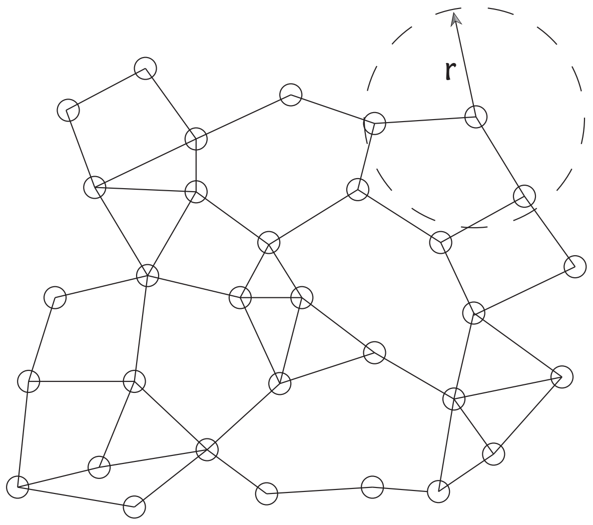


Figure 2.15: *Random geometric graph.*

Such a random geometric graph with $N = 32$ is illustrated in Fig. 2.15. It is shown that each node in the center of a circle with radius r (the dashed circle) is connected to all other nodes which are in this circle.

If we consider two nodes i and j with Euclidean distance $d(i, j)$ between them, we define the elements of adjacency matrix of random geometric graph as follow:

$$a_{ij} = a_{jj} = \begin{cases} 1, & \text{if } d(i,j) \leq r, \\ 0, & \text{otherwise.} \end{cases} \tag{2.20}$$

2.3.2 Random Regular Graph

A random regular graph $\mathcal{G}(N, d)$ is a graph with N nodes and d defines number of connected neighbors of each node.

In Fig. 2.16 we have an example of a random regular graph with $N = 32$ nodes and each node is connected to $d = 3$ other nodes. The advantage of such a graph is that the failure of one node can be compensated because we have for each node d different links nodes.

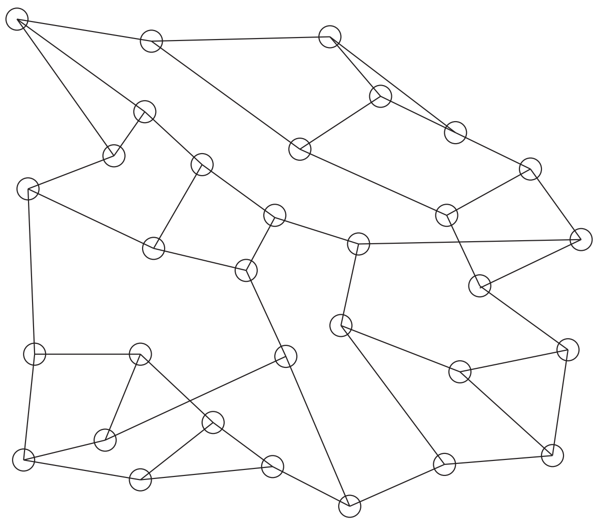


Figure 2.16: *Random regular graph (d=3).*

2.3.3 Ring Graph

Ring graph is a graph with N nodes which are connected together in form of a ring. Each node is connected with its two neighbor nodes. We can declare a ring graph as a random regular graph $\mathcal{G}(N, d)$ with $d = 2$ in that the nodes are placed in a form of a ring. One possible adjacency matrix for a ring graph is as follow:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & . & . & . & 0 \\ 1 & 0 & 1 & & & & . \\ 0 & 1 & . & . & & & . \\ . & & . & . & . & & . \\ . & & & . & . & 1 & . \\ . & & & & 1 & 0 & 1 \\ 0 & . & . & . & 0 & 1 & 0 \end{bmatrix}.$$

It is obvious that the main diagonal elements are all zero and the elements in the first diagonal above and below the main diagonal are all 1, all other elements are zero.

A ring graph with $N = 20$ is shown in Fig. 2.17. Each node has just two independent ways to transmit informations, because of that is the data transmission over such a network very slow. Another disadvantage is that the failure of two edges leads to disconnection of graph.

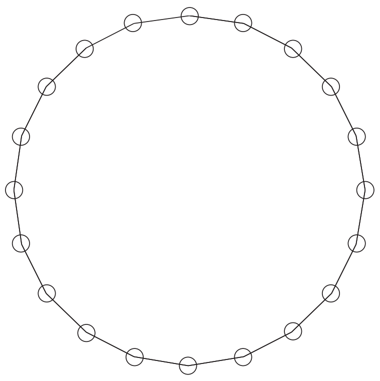


Figure 2.17: *Ring graph.*

2.3.4 Complete Graph

In a complete graph all nodes are connected to each other. A complete graph with N nodes has $\frac{N(N-1)}{2}$ edges. We can declare a complete graph as a random regular graph $\mathcal{G}(N, d)$ with $d = N - 1$. The complement graph of a complete graph is an empty graph.

The adjacency matrix of a complete graph is as follow:

$$\mathbf{A} = \mathbf{J} - \mathbf{I} = \begin{bmatrix} 0 & 1 & 1 & . & . & . & 1 \\ 1 & 0 & 1 & . & . & . & 1 \\ 1 & 1 & . & & & & . \\ . & . & & . & & & . \\ . & . & & & . & & . \\ . & . & & & & 0 & 1 \\ 1 & 1 & . & . & . & 1 & 0 \end{bmatrix}.$$

where $\mathbf{J}$ is the *unit matrix*[1] and $\mathbf{I}$ the *identity matrix*[2]. All diagonal elements of $\mathbf{A}$ are zero, other elements are equal to 1.

The advantage of this graph in comparison with other graphs is that the failure of an edge can be compensated with help of other edges and the information transmission is very fast. On the other side increase of number of edges leads to higher cost.

A complete graph with $N = 6$ nodes and 15 edges can be seen in Fig. 2.18. All nodes are incident with 5 other edges and are connected together.

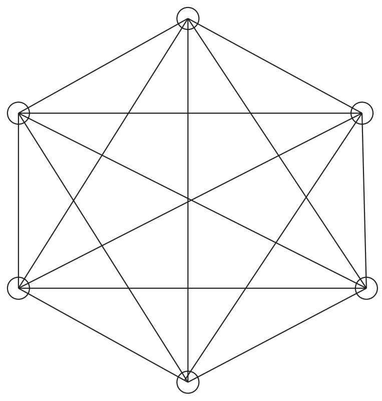


Figure 2.18: *Complete graph.*

2.3.5 Graph with Bottleneck

One special case is that we have two different graphs which are connected together over a single edge. The transmission of informations from one graph to other one can achieve just over this single edge, so it will be very slow.

[1]The unit matrix $\mathbf{J}$ is a matrix with all elements equal to one.

[2]The identity matrix $\mathbf{I}$ is a $N \times N$ matrix, The main diagonal elements are all ones and the other elements are zeros.

In Fig. 2.19 is such a graph with bottleneck illustrated. We have two different random geometric graphs which are connected over a single edge.

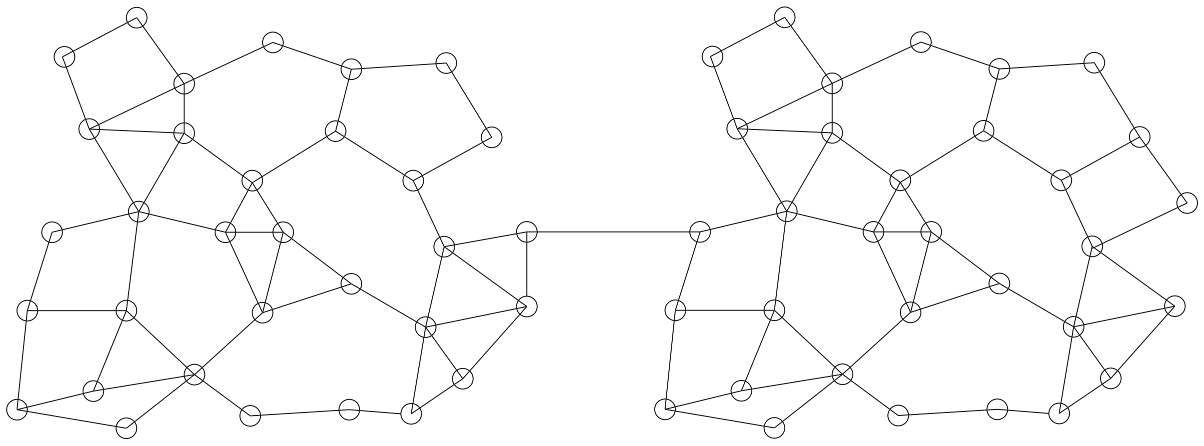


Figure 2.19: *Graph with Bottleneck.*

3 Static Average Consensus

3.1 Introduction

In this chapter we consider static average consensus (AC) and its referred algorithms. These algorithms and their structures were introduced in [3], [4], and [5].

At the beginning of this chapter in Section 3.2 we present the *problem statement* and compare different centralized and distributed averaging methods. In Section 3.3 we consider *linear static AC algorithm* and its generic form. We explain detailed the function of this algorithm. In Section 3.4 we present the *convergence conditions* which the algorithm must satisfy. After that in Section 3.5 we investigate the *convergence speed* and explain the used parameters. We will continue in Section 3.6 with a detailed discussion of the *weight matrix* and the methods to construct it. In the last section we present the *nonlinear static average algorithm* which yields a better performance in term of convergence than static AC.

3.2 Problem Statement

Let us assume a network with N different wireless sensors. These sensors are equipped with dedicated *sensing*, *computing*, and *communication* devices. Each sensor i measure a constant local value $s_i \in \mathbb{R}$ and we want to calculate the average $\overline{s}$ of these measurements:

$$\overline{s} = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{3.1}$$

Our goal is that each sensor achieve asymptotically this average value. One possible way is to connect all sensors with a fusion center which can calculate the average value. The problem of this method is that the wireless sensors have a constricted range of connection and they can not be out of this connection range from the fusion center, so the distribution of sensors on a larger area in not possible. Moreover it is required that the fusion center transmit the average value to each sensor which waste an additional energy. In a centralized architecture the failure of the fusion center leads to a collapse of the entire network.

The reliability of the sensor network can be increased by using distributed architecture. Distributed averaging can be done in different ways.

One way is that each sensor maintain a table of measured values of sensors. Initially it puts its own measured value in the table and lets the place of other sensors empty. Each sensor exchanges this table at each iteration with its connected neighbors and completes its own table. After a number of iterations dependent on number of the sensors and the connectivity, each sensor has a complete table with values of all sensors and can calculate the average value. The disadvantage of this way is that each sensor must store values of all sensors, so it needs a greater storage capacity. After each iteration each sensor transmit a higher amount of data thus we need a higher bandwidth.

Another and the better way for distributed averaging is the *average consensus (AC)*. Each sensor puts the measured value initially as its state. It communicates with connected neighbors and transmits its state. Then, the sensors use the received values and update their internal states. After a same time each sensor will have an estimation of the global average. In comparison with the before mentioned algorithm each sensor needs to store and transmit its own state to its neighbors, so we need a less storage capacity and also we have lower amount of data that is transmitted and respectively need less bandwidth.

Distributed averaging found a wide range of applications. Some of them are formation flight of unmanned air vehicles and clustered satellites, coordination of mobile robots, synchronization of coupled oscillators, load balancing for parallel processors and

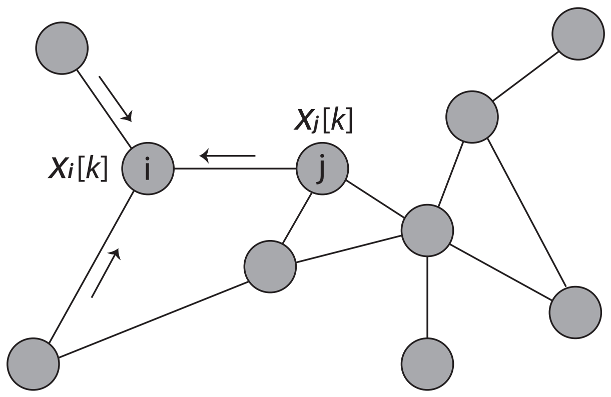


Figure 3.1: *Sensor network in which node i receives the state x_j of node j and the states of other nodes.*

network synchronization.

3.3 Linear Static Average Consensus Algorithm

Generally in wireless sensor networks (WSN), the connected sensors can communicate in both directions. Therefore we represent a wireless sensor network as a connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of nodes (sensors) $\mathcal{V}$ and the set of edges (communication links) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The cardinality $|\mathcal{V}| = N$ corresponds to the number of nodes. If two nodes i and j are connected, the edge $\{i, j\} = \{j, i\} \in \mathcal{E}$ is the communication link between them. The set of connected neighbors of node i is denoted as neighbor set $\mathcal{N}_i = \{j \in \mathcal{V} : \{i, j\} \in \mathcal{E}\}$.

In some special cases in WSN, some sensors can communicate just in one direction. For example if sensor i has a larger connection range than other sensors, it can transmit informations to them in this range, but some of these sensors are too far from sensor i and have a less range, so they can not transmit informations to sensor i. In these cases we represent a WSN as a strongly connected directed graph $\mathcal{G}$. Instead of the undirected edge we use $\{i, j\}$, the directed edge (i, j). If just node i can communicate with node j and not vice versa, the edge $(i, j) \in \mathcal{E}$. The neighbor set of node i is then $\mathcal{N}_i = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$.

In this work we concentrate us on undirected graphs so wireless sensors which can communicate in both directions.

As shown in Fig. 3.1, we denote the state of node i by $x_i[k]$ and the state of node j by $x_j[k]$ where k is the discrete time index. Node i measure the constant value s_i. The state of node i is initially equal to this measured value:

$$x_i\left[0\right] = s_i, \qquad \text{for } i = 1, \ldots, N. \tag{3.2}$$

At each iteration the neighbors of node i which are listed in neighbor set $\mathcal{N}_i$, send their states. Node i updates its own state as follow:

$$x_i\left[k+1\right] = W_{ii}x_i\left[k\right] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j\left[k\right], \qquad i = 1, \ldots, N, \tag{3.3}$$

The weight W_{ij} is the weight of edge $\{i, j\}$ (edge weights). We set $W_{ij} = 0$ for $j \notin \mathcal{N}_i$. The value W_{ii} weights the node's internal state (self-weights). Under certain conditions which we will describe later, the state of node i will asymptotically converge to the average value $\overline{s}$:

$$\lim_{k\to\infty} x_i\left[k\right] = \overline{s}. \tag{3.4}$$

The generic form of (3.3) follows as:

$$\mathbf{x}[k+1] = \mathbf{W}\mathbf{x}[k], \tag{3.5}$$

where $\mathbf{x}\left[k\right] = \left(x_1\left[k\right], \ldots x_N\left[k\right]\right)^T$ denotes the state vector consisting of the states of all nodes at discrete time k. The weight matrix $\mathbf{W}$ consists of the elements W_{ij} and belongs therefore to the set $\mathcal{J}$,

$$\mathcal{J} = \left\{\mathbf{W} \in \mathbb{R}^{N\times N} : W_{ij} = 0 \text{ if } \{i,j\} \notin \mathcal{E} \text{ and } i \neq j\right\}. \tag{3.6}$$

State vector $\mathbf{x}[k]$ will converge asymptotically to average vector:

$$\lim_{k\to\infty} \mathbf{x}\left[k\right] = \overline{\mathbf{s}}, \qquad \text{where } \overline{\mathbf{s}} = \overline{s}\,\mathbf{1}. \tag{3.7}$$

where $\mathbf{1}$ denotes a vector with all coefficients one. This applied algorithm is called as *linear static AC algorithm.*

In Fig. 3.2 we see an arbitrary graph with 150 nodes which are distributed on a unit square. The axes r_1 and r_2 define the locations of nodes which are represented as circles. If two nodes communicate with each other, we represent them as connected circles. Fig. 3.3 shows an example of AC applied at this graph. Each colored line refers to state of a node which converge to average value. We see that the states of nodes converge after 275 iterations to average value 0.4431 with precision of 10^{-4}.

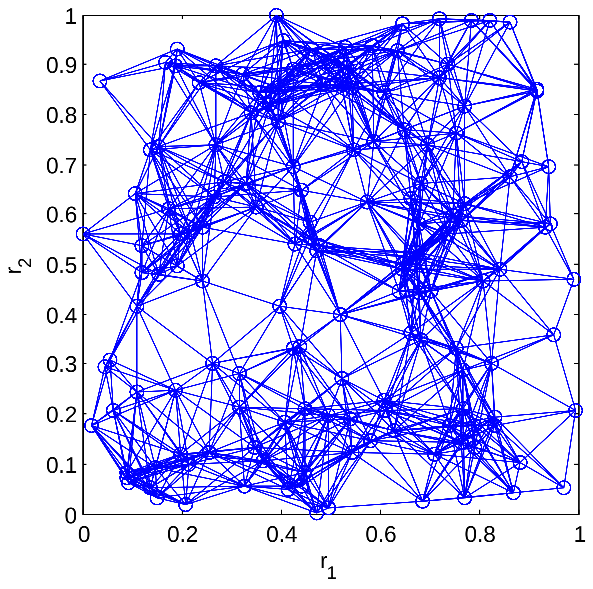


Figure 3.2: *Distribution of nodes in unit square.*

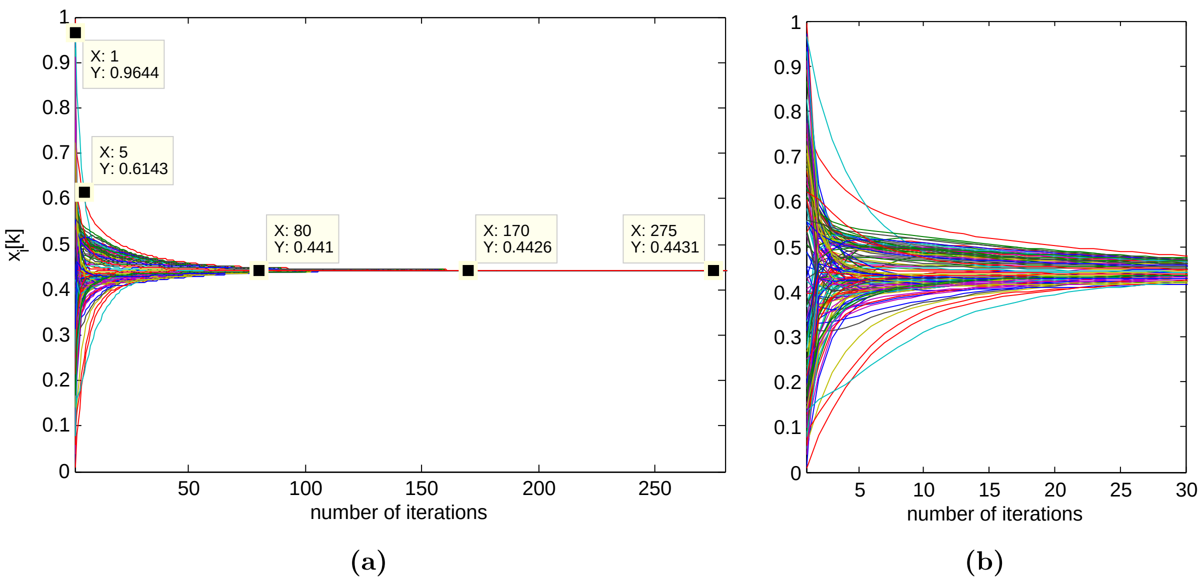


Figure 3.3: *States of nodes versus number of iterations with (a) linear static AC of a network and (b) its zoomed plot for the first 30 iterations.*

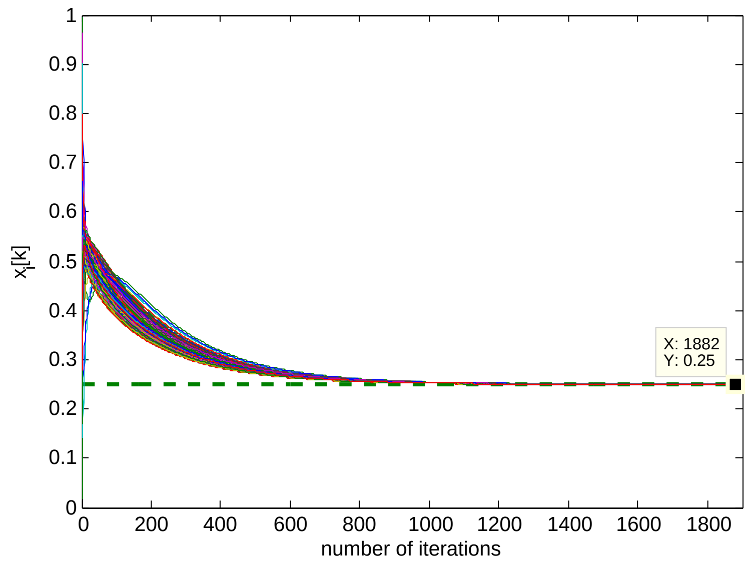


Figure 3.4: *States of nodes versus number of iterations for unconstrained consensus with single leader ($x\,[k] = 0.25$).*

As mentioned in [3], AC is a cooperative task and needs participation of all nodes. If a single node does not cooperate with the other nodes and keeps its state unchanged but still exchanges messages, the average value $\overline{s}$ in (3.1) can not be achieved. However in this case all nodes asymptotically converge to a common value, this common value is the unchanged state value of that single node. We call this node as a *leader*. Therefore we can distinguish between a *unconstrained* and *constrained* consensus. In unconstrained consensus all nodes converge to a common value but it should not be necessarily the average value but in constrained consensus all nodes converge to the average value $\overline{s}$.

Such a unconstrained consensus with a single leader is demonstrated in Fig. 3.4. We see that a single node denoted with green dashed line has a state value of $x\,[k] = 0.25$ and it keeps this state unchanged. All the nodes converge after 1882 iterations to state value of leader with precision of 10^{-4}.

If there are multiple leaders with different states, then no consensus can be asymptotically reached. In Fig. 3.5 is such a unconstrained consensus with two different leaders demonstrated. We see that two nodes have states of $x_{\text{leader},1}\,[k] = 0.25$ (denoted with green dashed line) and $x_{\text{leader},2}\,[k] = 0.75$ (denoted with blue dashed line) which stay unchanged over iterations. In this case, all nodes can not converge to a common value.

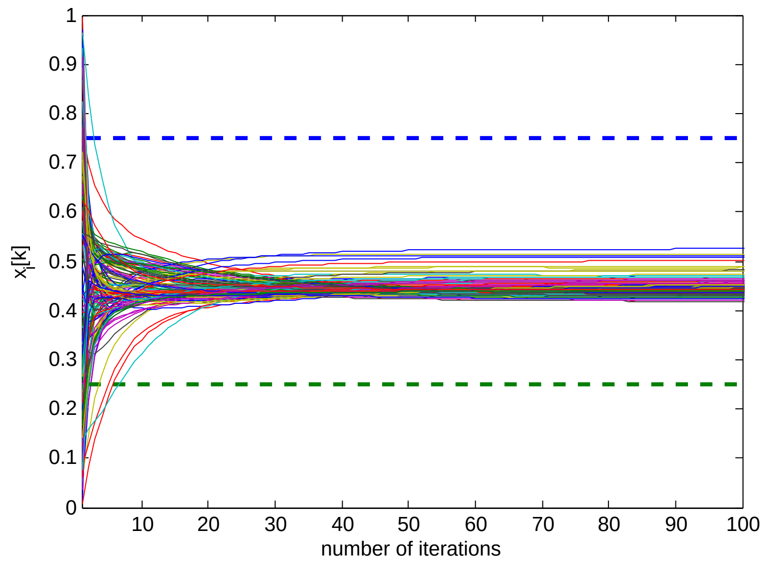


Figure 3.5: *States of nodes versus number of iterations for unconstrained consensus with two leaders ($x_{leader,1}[k] = 0.25$ and $x_{leader,2}[k] = 0.75$).*

3.4 Convergence Conditions

According to [4], the (3.5) can be reformulated as follow:

$$\mathbf{x}[k] = \mathbf{W}^k \mathbf{x}[0], \qquad \text{for } k = 1, 2, 3, \dots \tag{3.8}$$

The average vector $\bar{\mathbf{s}}$ can be transformed with help of Equations (3.1) and (3.2):

$$\bar{\mathbf{s}} = \left(\mathbf{1}^T \mathbf{x}[0]/N\right)\mathbf{1} = \left(\mathbf{1}\mathbf{1}^T/N\right)\mathbf{x}[0]. \tag{3.9}$$

The weight matrix $\mathbf{W}$ should be chosen so that for any initial state vector $\mathbf{x}[0]$, state vector $\mathbf{x}[k]$ converge for $k \to \infty$ to the average vector:

$$\lim_{k\to\infty} \mathbf{x}[k] = \lim_{k\to\infty} \mathbf{W}^k \mathbf{x}[0] = \frac{\mathbf{1}\mathbf{1}^T}{N}\mathbf{x}[0]. \tag{3.10}$$

This is equivalent to Equation

$$\lim_{k\to\infty} \mathbf{W}^k = \frac{\mathbf{1}\mathbf{1}^T}{N}. \tag{3.11}$$

To satisfy this equation, weight matrix $\mathbf{W}$ must fulfill the necessary and sufficient following conditions as defined in [4]:

$$\mathbf{1}^T\mathbf{W} = \mathbf{1}^T, \tag{3.12}$$

$$\mathbf{W}\mathbf{1} = \mathbf{1}, \tag{3.13}$$

$$\rho\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) < 1, \tag{3.14}$$

where $\rho\left(\cdot\right)$ is the *spectral radius*[1].

We give now some interpretations of this conditions:

- (3.12) shows that $\mathbf{1}$ is a left eigenvector of weight matrix $\mathbf{W}$ with associated eigenvalue one. This condition implies that the sum of the states of the nodes remains unchanged over different iterations, respectively the average of the states of the nodes remain also unchanged:

$$\mathbf{1}^T\mathbf{x}[k+1] = \mathbf{1}^T\mathbf{W}\mathbf{x}[k] = \mathbf{1}^T\mathbf{x}[k].$$

- (3.13) shows us that $\mathbf{1}$ is a right eigenvector of weight matrix $\mathbf{W}$ with associated eigenvalue one. This condition implies that $\mathbf{1}$ or any multiply of it (multiplied by a constant c) is the fix point of AC algorithm:

$$\text{if } \mathbf{x}[k] = c\mathbf{1} \Rightarrow \mathbf{x}[k+1] = \mathbf{W}\mathbf{x}[k] = c\mathbf{W}\mathbf{1} = c\mathbf{1},$$

  this means that if consensus is a fixed point, hence, if consensus is reached the states will not change.

- Condition in (3.14) ensures that AC converges to the fixed point and therefore to the average, i.e., the algorithm will be stable.

- Together with conditions in (3.12) and (3.13), the condition (3.14) implies according to [4] for the eigenvalues of weight matrix $\mathbf{W}$ (of a strongly connected graph) the following condition:

$$-1 < \lambda_N(\mathbf{W}) \leq \lambda_{N-1}(\mathbf{W}) \leq \ldots \leq \lambda_2(\mathbf{W}) < \lambda_1(\mathbf{W}) = 1 \tag{3.15}$$

[1]If λ_1, λ_2, ... are the eigenvalues of matrix $\mathbf{A}$, the spectral radius is defined as $\rho\left(\mathbf{A}\right) = \max\limits_i\left(|\lambda_i|\right)$.

3.5 Convergence Speed

To have an AC the condition in (3.11) must be satisfied. In this case, [4] defines the *asymptotic convergence factor* to measure the convergence speed as below:

$$r_{\text{asymptotic}}\left(\mathbf{W}\right) = \sup_{\mathbf{x}[0]\neq\bar{\mathbf{s}}} \lim_{k\to\infty} \left(\frac{||\mathbf{x}\left[k\right]-\bar{\mathbf{s}}||_2}{||\mathbf{x}\left[0\right]-\bar{\mathbf{s}}||_2}\right)^{1/k}, \tag{3.16}$$

which is used for calculation of *convergence time*:

$$\tau_{\text{asymptotic}} = \frac{1}{\log\left(1/r_{\text{asymptotic}}\right)}, \tag{3.17}$$

this defines the asymptotic number of iterations for the error to decrease by a factor of $1/e$.

According to [4], another factor to measure the speed of convergence is *per-step convergence factor*,

$$r_{\text{step}}\left(\mathbf{W}\right) = \sup_{\mathbf{x}[0]\neq\bar{\mathbf{s}}} \frac{||\mathbf{x}\left[k+1\right]-\bar{\mathbf{s}}||_2}{||\mathbf{x}\left[k\right]-\bar{\mathbf{s}}||_2}, \tag{3.18}$$

and the referred *convergence time* is as below:

$$\tau_{\text{step}} = \frac{1}{\log\left(1/r_{\text{step}}\right)}. \tag{3.19}$$

We know from Section 3.4 that the weight matrix $\mathbf{W}$ must hold the conditions in Equations (3.12), (3.13), and (3.14) to satisfy the property in (3.11), so according to [4], the (3.16) can be reformulated with help of spectral radius as,

$$r_{\text{asymptotic}}\left(\mathbf{W}\right) = \rho\left(\mathbf{W}-\frac{\mathbf{1}\mathbf{1}^T}{N}\right), \tag{3.20}$$

and the (3.18) with help of spectral norm as,

$$r_{\text{step}}\left(\mathbf{W}\right) = \left|\left|\mathbf{W}-\frac{\mathbf{1}\mathbf{1}^T}{N}\right|\right|_2. \tag{3.21}$$

3.6 Weight Matrix

Here we consider different possible weight matrices and their performance in dependence of their eigenvalues distribution.

3.6.1 Weight Matrix based on Constant Weights

The simplest method to design a weight matrix is to set all edge weights equal to a constant α. According to [3] and [4], the elements of $\mathbf{W}$ can be represented as below:

$$W_{ij} = \begin{cases} \alpha, & \{i,j\} \in \mathcal{E}, \\ 1 - d_i\alpha, & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{3.22}$$

where d_i is the indegree of node i. Respectively the weight matrix $\mathbf{W}$ can be written as designed in [3] and [4] as follow:

$$\mathbf{W} = \mathbf{I} - \alpha\mathbf{L}, \tag{3.23}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{L}$ is the Laplacian matrix (see Section 2.2.6).

We see in (3.22) that in each row i of the weight matrix $\mathbf{W}$, we have d_i times constant edge weights with equal values α. The value of diagonal element in each row equals to $1 - d_i\alpha$. So, the sum of each row is $d_i\alpha + 1 - d_i\alpha = 1$ which means that the condition in (3.13) holds.

Another condition that must be satisfied is the condition in (3.12). It means that the sum of each column of weight matrix must be also equal to 1. In undirected graphs the weight matrix is symmetric which means that we have the same elements in ith column and ith row, therefore, the sum of each column will be like sum of each row equal to 1. It yields in undirected graphs if the condition in (3.13) holds, the condition in (3.12) holds also. But in directed graphs, the structure of graph lead to a asymmetric weight matrix. The condition in (3.12) satisfies if the graph is *balanced*. In a balanced graph the number of edges which emanates from a node and terminates at this node are equal which means that because of design of weight matrix with defined elements in (3.22) (equal values of edge-weights), we have an equal value for sum of ith row and sum of ith column:

$$\sum_{j=1}^{N} a_{ij} = \sum_{j=1}^{N} a_{ji} = d_i\alpha + 1 - d_i\alpha = 0, \qquad \text{for } i \neq j \text{ and } i = 1 \dots N, \tag{3.24}$$

where a_{ij} are the elements of adjacency matrix (see Section 2.2.2). Therefore if a balanced graph satisfies the condition 3.13 respectively it satisfies the condition 3.12 too.

Another condition that must be satisfied is the condition $\rho\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) < 1$ (in (3.14)). From (3.23) we can express the relation between eigenvalues of weight matrix, $\lambda_i(\mathbf{W})$, and eigenvalues of Laplacian matrix, $\lambda_i(\mathbf{L})$ as

$$\lambda_i(\mathbf{W}) = 1 - \alpha\lambda_{N-i+1}(\mathbf{L}), \qquad \text{for } i = 1, 2, 3, \ldots, N, \tag{3.25}$$

where $\lambda_i(.)$ denotes the ith largest eigenvalue of a matrix. We know from (2.18) that zero is the smallest eigenvalue of $\mathbf{L}$, consequently it is obvious from (3.25) that the largest eigenvalue of $\mathbf{W}$ is one, i.e., $\lambda_N(\mathbf{L}) = 0$ and $\lambda_1(\mathbf{W}) = 1$. The possible eigenvalues of $\mathbf{W}$ are represented in (3.15), we consider now the possible eigenvalues of $\frac{\mathbf{1}\mathbf{1}^T}{N}$,

$$\lambda_1\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = 1,\ \lambda_2\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \lambda_3\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \ldots = \lambda_N\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = 0. \tag{3.26}$$

so the eigenvalues of $\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}$ are as follow:

$$\begin{aligned}
\lambda_1\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) &= \lambda_1(\mathbf{W}) - \lambda_1\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = 0,\\
\lambda_2\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) &= \lambda_2(\mathbf{W}) - \lambda_2\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \lambda_2(\mathbf{W}),\\
\lambda_3\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) &= \lambda_3(\mathbf{W}) - \lambda_3\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \lambda_3(\mathbf{W}),\\
&\ \ \vdots\\
\lambda_N\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) &= \lambda_n(\mathbf{W}) - \lambda_N\left(\frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \lambda_N(\mathbf{W}).
\end{aligned} \tag{3.27}$$

According to (3.15), $\lambda_2(\mathbf{W})$ is the largest eigenvalue and $\lambda_N(\mathbf{W})$ the smallest eigenvalue, respectively we express the condition in (3.14) as follow:

$$\rho\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) = \max\{\lambda_2(\mathbf{W}),\ -\lambda_N(\mathbf{W})\} = \max\{1 - \alpha\lambda_{N-1}(\mathbf{L}),\ \alpha\lambda_1(\mathbf{L}) - 1\} < 1. \tag{3.28}$$

We determine the range of α so that the condition 3.28 (or 3.14) satisfies. If $1-\alpha\lambda_{N-1}(\mathbf{L})$ is the maximum, the condition satisfies for all $\alpha > 0$ because in a connected graph $\lambda_{N-1}(\mathbf{L})$ is always greater than zero. If $\alpha\lambda_1(\mathbf{L}) - 1$ is the maximum, the condition satisfies for all $\alpha < \frac{2}{\lambda_1(\mathbf{L})}$. So to satisfy the condition the range of α is defined as follow:

$$0 < \alpha < \frac{2}{\lambda_1(\mathbf{L})}. \tag{3.29}$$

According to [4], the best choice of α that minimize the (3.28) is as below, we call its referred weights as *best constant (BC) weights.*

$$\alpha_{\text{bc}} = \frac{2}{\lambda_1(\mathbf{L}) + \lambda_{N-1}(\mathbf{L})}. \tag{3.30}$$

The bound defined in (3.29) utilizes the Laplacian matrix $\mathbf{L}$ but we have another choices to express this bound without knowledge of Laplacian matrix. We know from [4] that

$$\lambda_1(\mathbf{L}) \leq \max_{\{i,j\}\in\mathcal{E}} (d_i + d_j),$$

so if we put this in (3.29) we get the following bound for α independent of Laplacian matrix:

$$0 < \alpha < \frac{2}{\max_{\{i,j\}\in\mathcal{E}}(d_i + d_j)}. \tag{3.31}$$

Now if d_i=d_j=$d_{\max}$ where $d_{\max} = \Delta$ is the maximum indegree over all nodes in graph the (3.31) can be expressed as

$$0 < \alpha \leq \frac{1}{d_{\max}}. \tag{3.32}$$

As discussed in [4], the convergence is guaranteed if

$$\alpha_{\text{md}} = \frac{1}{d_{\max}}. \tag{3.33}$$

α_{md} is the *maximum degree constant* and its referred elements of weight matrix are called as *maximum degree (MD) weights.*

With help of (3.22), (3.3) can be expressed as shown in [3] as,

$$x_i[k+1] = x_i[k] + \alpha \sum_{j\in\mathcal{N}_i} (x_j[k] - x_i[k]), \qquad \text{for } i = 1,\ldots,N. \tag{3.34}$$

It is obvious that in a undirected graph or in a directed balanced graph with the weight matrix $\mathbf{W}$ with elements defined in (3.22) and the edge-weights α which are constricted in the bound defined in (3.29), the convergence conditions in (3.12), (3.13), and (3.14) are satisfied.

3.6.2 Weight Matrix based on Convex Optimization

To make convergence asymptotically faster, convex optimization is utilized. As discussed in [4], the weight matrix $\mathbf{W}$ will be designed so that the asymptotic convergence factor $r_{\text{asymptotic}}(\mathbf{W})$ from (3.20) is minimized, i.e.,

$$\begin{aligned} &\text{minimize} && r_{\text{asymptotic}}(\mathbf{W}) = \rho\left(\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right) \\ &\text{subject to} && \mathbf{W} \in \mathcal{J}, \quad \mathbf{1}^T\mathbf{W} = \mathbf{1}^T, \quad \mathbf{W}\mathbf{1} = \mathbf{1}. \end{aligned} \tag{3.35}$$

It is called as *spectral radius minimization problem.* The solving of this problem is very difficult because the spectral radius $\rho(.)$ of a matrix is not a convex function.

According [4], another method to increase the convergence speed is that per-step convergence factor $r_{\text{step}}(\mathbf{W})$ from (3.21) is minimized, i.e.,

$$\begin{aligned} &\text{minimize} && r_{\text{step}}(\mathbf{W}) = \left\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right\|_2 \\ &\text{subject to} && \mathbf{W} \in \mathcal{J}, \quad \mathbf{1}^T\mathbf{W} = \mathbf{1}^T, \quad \mathbf{W}\mathbf{1} = \mathbf{1}. \end{aligned} \tag{3.36}$$

It is called as *spectral norm minimization problem.* The solving of this problem is in comparison with (3.35) more easier because the problem is convex.

If the weight matrix $\mathbf{W}$ is symmetric, i.e., $W_{ij} = W_{ji}$, the spectral radius minimization problem and the spectral norm minimization problem are identical. In this case, both problems from 3.35 and 3.36 can be expressed as defined by [4] as:

$$\begin{aligned} &\text{minimize} && \left\|\mathbf{W} - \frac{\mathbf{1}\mathbf{1}^T}{N}\right\|_2 \\ &\text{subject to} && \mathbf{W} \in \mathcal{J}, \quad \mathbf{W} = \mathbf{W}^T, \quad \mathbf{W}\mathbf{1} = \mathbf{1}. \end{aligned} \tag{3.37}$$

which is a convex problem.

To solve the problem in (3.37) the weight matrix $\mathbf{W}$ can be replaced according to [4] as follow:

$$\mathbf{W} = \mathbf{I} - \mathbf{B}\,\text{diag}(\mathbf{w})\mathbf{B}^T, \tag{3.38}$$

The matrix $\mathbf{B}$ is an $N \times M$ oriented incidence matrix with N nodes and M edges. Here we assign arbitrarily a reference direction for each edge in the undirected graph which leads to a directed graph and then write the incidence matrix in sense of the matrix elements in (2.10) for directed graphs. The vector $\mathbf{w} \in \mathbb{R}^M$ consists of weights on all edges. For each edge we give a weight $w_l = W_{ij} = W_{ji}$ where edge l is incident with nodes i and j.

Respectively (3.37) can be reformulated as as mentioned in [4] as follow:

$$\text{minimize} \qquad \left\| \mathbf{I} - \mathbf{B}\,\mathrm{diag}(\mathbf{w})\mathbf{B}^T - \frac{\mathbf{1}\mathbf{1}^T}{N} \right\|_2 . \tag{3.39}$$

To solve this equation we use in this work the CVX a Matlab software for disciplined convex programming from [6] and [7]. The elements of vector $\mathbf{w}$ and respectively the elements of weight matrix $\mathbf{W}$ that we calculate here are called as *convex optimization (CXO) weights.*

It is obvious that the designed weight matrix $\mathbf{W}$ satisfies the condition in (3.13) because of its design expressed in (3.38). On the other hand, the weight matrix is symmetric thus, it satisfies also the condition in (3.12). Therefore, we can use the weight matrix based on convex optimization just for undirected graphs because in directed graphs the weight matrix is not symmetric. Due to spectral minimization problem that we have here, the condition (3.14) holds too.

3.6.3 Weight Matrix based on Metropolis-Hastings algorithm

As discussed in [4], another method to design the weight matrix $\mathbf{W}$ is that to assign the weight on each edge as follow:

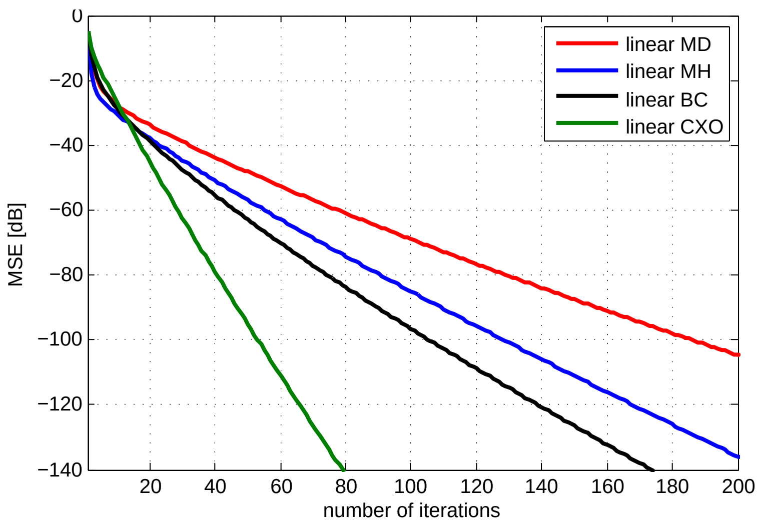
$$w_l = \frac{1}{\max\{d_i, d_j\}}, \tag{3.40}$$

for each $\{i, j\} \in \mathcal{E}$ or we can express the elements of $\mathbf{W}$ as

$$W_{ij} = \begin{cases} \frac{1}{\max\{d_i,d_j\}}, & \{i, j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.41}$$

The matrix that we designed here is a weight matrix based on Metropolis-Hastings algorithm, used to simulate a Markov chain with uniform equilibrium distribution. The elements of this matrix are called as *Metropolis-Hastings (MH) weights.*

It is obvious that because of design of weight matrix $\mathbf{W}$ the sum of each row equals to one so the $\mathbf{W}$ satisfies the condition in (3.13). To satisfy the condition in (3.12) it must be a symmetric matrix. Therefore the weight matrix based on Metropolis-Hastings algorithm is also like the weight matrix based on convex optimization just for undirected graphs.



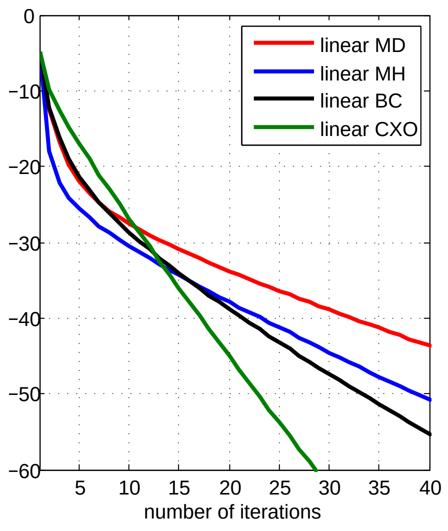


Figure 3.6: *(a) MSE of linear static AC algorithm versus number of iterations and (b) its zoomed plot for the first 40 iterations.*

	MD	MH	BC	CXO
$\rho\left(\mathbf{W} - \mathbf{1}\mathbf{1}^T/N\right)$	0.93883	0.91813	0.88964	0.78324
$\tau = 1/\log(1/\rho)$	15.8439	11.7074	8.5511	4.0931

Table 3.1: *Convergence factors and times of different weight matrices.*

3.6.4 Influence of Eigenvalues of Weight Matrix on Convergence Speed

We expressed in Section 3.6 weight matrices based on different methods. One possible example for this methods can be seen in Figure 3.6. We see the *mean square error* (MSE) of linear static AC algorithm versus number of iterations. It can be seen that the method with CXO weights is asymptotic the fastest and the method with MD weights asymptotic the slowest. The method with BC is faster than MH but both of them are slower than the method with CXO. By studying of the transient phase (beginning phase), we see that the method with MH weights is the fastest in comparison to others.

The asymptotic convergence factors and convergence times for different methods of this example, are represented in Tbl. 3.1. As mentioned before, we see that the method with CXO weights is asymptotic the fastest and the method with MD weights asymptotic the slowest. It can be seen that the method with CXO weights is about two times faster than the method with BC weights, about three times faster than MH weights and about four times faster than MD weights.

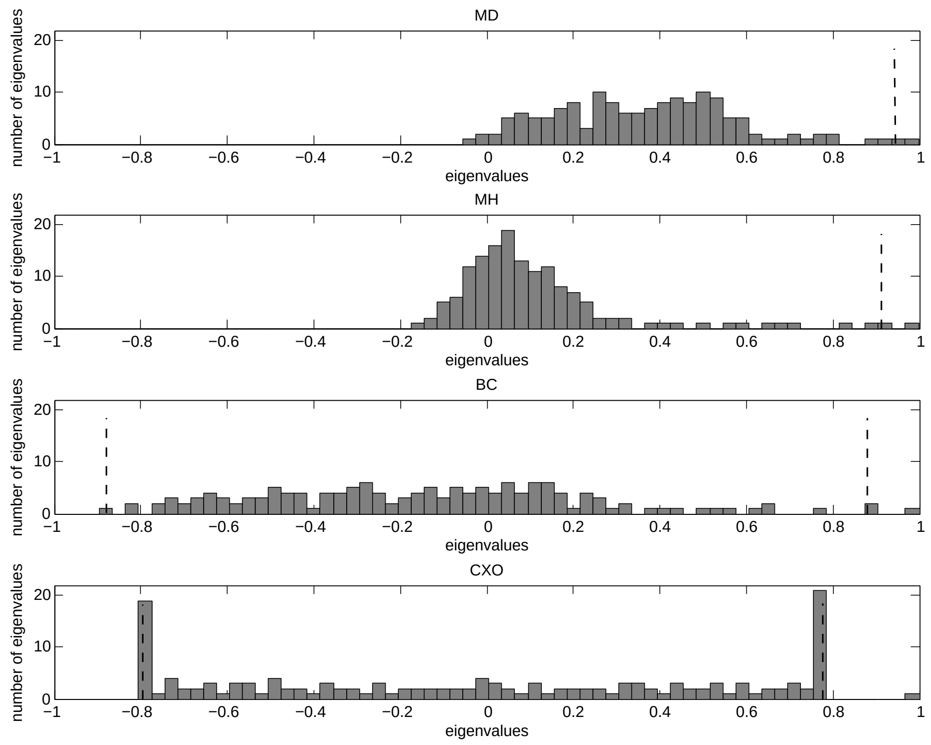


Figure 3.7: *Distribution of eigenvalues of weight matrices* $(r = 0.28)$.

We consider the distribution of eigenvalues of weight matrix $\mathbf{W}$ for each method in Figure 3.7 and come to following conclusions:

- It is obvious that we have always a single eigenvalue at one.
- We see that for a asymptotic faster method the second largest eigenvalue (algebraic connectivity) denoted by dashed lines on the right side of figure, is smaller than second largest eigenvalue of a slower method. For the MD and MH weights, the convergence factor is just dependent on the second largest eigenvalue, so $\rho\left(\mathbf{W} - \frac{\mathbf{11}^T}{N}\right) = \lambda_2(\mathbf{W})$, (from (3.28)) but for BC and CXO weights the convergence factor is dependent on the second largest eigenvalue and the smallest eigenvalue (both denoted by dashed lines), so $\rho\left(\mathbf{W} - \frac{\mathbf{11}^T}{N}\right) = \lambda_2(\mathbf{W}) = -\lambda_N(\mathbf{W})$.
- The approximately symmetric distribution of eigenvalues around zero leads to a faster asymptotically convergence. It can be seen that for the method with CXO weights, we have an approximately symmetric distribution.

3.7 Nonlinear Static Average Consensus Algorithm

We try to improve the convergence speed of AC algorithm through nonlinearity. Here we consider two different ways, ones is varying the elements (weights) of a weight matrix appropriately through a nonlinear function. Another way is that we switch through a nonlinear function between two different weight matrices here ones with MH weights (fast convergence in transient phase) and the another one with CXO weights (fast convergence in asymptotic phase) and use the advantages of both methods.

3.7.1 Nonlinearity by Varying the Weights of a Weight Matrix

To improve the convergence speed, [5] leaves the linear weights which are defined in Section 3.6 at asymptotic phase unchanged or multiplied with a constant factor but we modulate them appropriately during the transient phase so we use a new weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ which is dependent on state vector $\mathbf{x}[k]$ and its weights change over different iterations. This algorithm is called as *nonlinear static AC algorithm*. The nonlinear generic form of the linear form defined in (3.5) can be written as

$$\mathbf{x}[k+1] = \mathbf{W}'\big(\mathbf{x}[k]\big)\,\mathbf{x}[k]. \tag{3.42}$$

According to [5], the elements of weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ are defined as below:

$$W_{ij}' = \begin{cases} W_{ij}\, f\left(u_{ij}\left[k\right]\right), & \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}', & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{3.43}$$

where W_{ij} are the elements of weight matrices which are defined in Section 3.6 and are multiplied with the nonlinear function $f\left(u_{ij}\left[k\right]\right)$. The node states difference $u_{ij}\left[k\right]$ which equals to difference between states of nodes that are connected together is as follow:

$$u_{ij}\left[k\right] = x_j\left[k\right] - x_i\left[k\right]. \tag{3.44}$$

The function $f(u)$ is a continuous function with following properties:

$$f(0) = 1, \tag{3.45}$$

$$f(u) = f(-u), \tag{3.46}$$

$$-1 \leq \frac{df}{du} \leq 1. \tag{3.47}$$

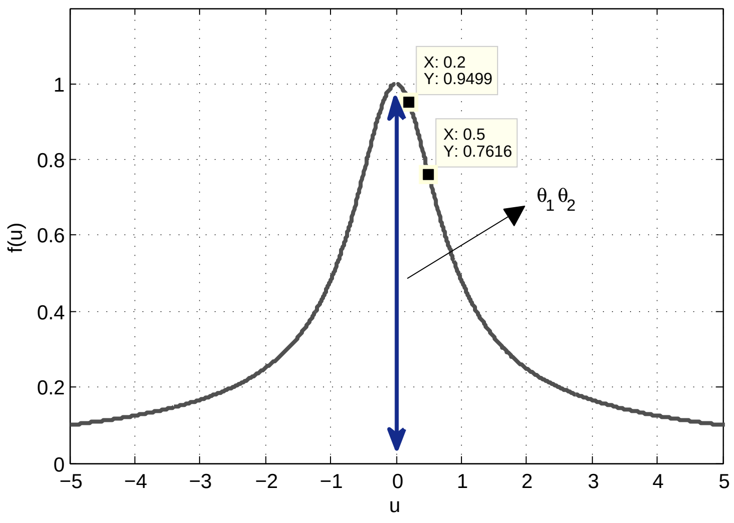


Figure 3.8: *The function $f(u)$ for $\theta_1 = 2$ and $\theta_2 = 0.5$.*

As discussed in [5], one possible function that satisfies this properties and modulate the weights appropriately is

$$f(u) = \frac{\tanh(\theta_1 u)\, \theta_2}{u}, \tag{3.48}$$

with the parameters θ_1, $\theta_2 \in \mathbb{R}_+$. θ_1 and θ_2 defines the slope of the curve and their product $\theta_1\theta_2$ defines the maximum value of it. An example of $f(u)$ with $\theta_1 = 2$ and $\theta_2 = 0.5$ is shown in Fig. 3.8.

We put now $u_{ij}[k] = x_j[k] - x_i[k]$ as input of function $f(u)$. In transient phase is the difference between node states larger (larger $u_{ij}[k]$) so the function $f(u_{ij}[k])$ returns a value which is smaller than the maximum $\theta_1\theta_2$ but after several iterations this difference between node states will be smaller (smaller $u_{ij}[k]$) and respectively the function $f(u_{ij}[k])$ returns a larger value which is closer to the maximum $\theta_1\theta_2$. Therefore each element $W_{ij}{}'$ of weight matrix $\mathbf{W}'(\mathbf{x}[k])$ takes for larger $u_{ij}[k]$ a value smaller than the product of W_{ij} and the maximum of $f(u_{ij}[k])$ $(\theta_1\theta_2)$ and for smaller $u_{ij}[k]$ a value close to this product. For example it can be seen in Fig. 3.8 that if $u_{ij}[k]$ takes at the beginning a value equal to 0.5 the $f(u_{ij})$ is then equal to 0.7616 and respectively $W_{ij}{}'[k] = 0.7616\, W_{ij}$ and after several iterations if $u_{ij}[k]$ takes a value equal to 0.2 the $f(u_{ij})$ is then equal to 0.9499 and respectively $W_{ij}{}'[k] = 0.9499\, W_{ij}$.

3.7.2 Nonlinearity by Combination of two Weight Matrices

We have seen in Fig. 3.6 that the method with MH weights is the fastest method in transient phase and the method with CXO weights is the fastest method at asymptotic phase. Now we combine this two different methods together so that the algorithm uses in transient phase the MH weights and at asymptotic phase the CXO weights. We call this algorithm as *nonlinear static AC algorithm with combination of MH and CXO weights*. We use the weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ which is dependent on node states $\mathbf{x}[k]$ and its weights change over different iterations. Similar to (3.42) the generic form is as follow:

$$\mathbf{x}[k+1] = \mathbf{W}'\big(\mathbf{x}[k]\big)\,\mathbf{x}[k].$$

The elements of weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ are defined as follow:

$$W_{ij}' = \begin{cases} W_{MH,ij}\,\left(1 - g\left(v_{ij}\,[k]\right)\right) + W_{CXO,ij}\,g\left(v_{ij}\,[k]\right), & \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}', & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{3.49}$$

where $W_{MH,ij}$ are the MH weights which are defined in (3.41), $W_{CXO,ij}$ are the CXO weights from (3.39), and the nonlinear function $g\left(v_{ij}\,[k]\right)$ for switching between MH and CXO weights. The relative node states difference $v_{ij}\,[k]$ which is the input of the nonlinear function is defined as below:

$$v_{ij}\,[k] = \frac{x_j\,[k] - x_i\,[k]}{\max\left(x_i\,[k]\,, x_j\,[k]\right)}. \tag{3.50}$$

The continuous function $g(v)$ can be expressed as follow:

$$g(v) = \frac{1}{2}\tanh\left(\gamma_1\left(|v| - \gamma_2\right)\right) + \frac{1}{2}\,, \tag{3.51}$$

with the parameters γ_1, $\gamma_2 \in \mathbb{R}_+$. γ_2 defines that in which range of inputs the function equals to 0 and in which range equals to 1, γ_1 defines the slope of the curve by switching between 0 and 1. One possible example of function $g(v)$ is shown in Fig. 3.9 with $\gamma_1 = 10$ and $\gamma_2 = 1$.

It is obvious that for a larger difference between states of neighbor nodes i and j, we get a larger value for $v_{ij}\,[k]$ and for smaller difference a smaller value. The difference of node states is in numerator and the maximum state value between node i and node j is in denominator so $v_{ij}\,[k]$ is a relative value. It consequents that $v_{ij}\,[k]$ is independent of input range of $x_i\,[k]$ and $x_j\,[k]$. To understand this better we give an example. Let

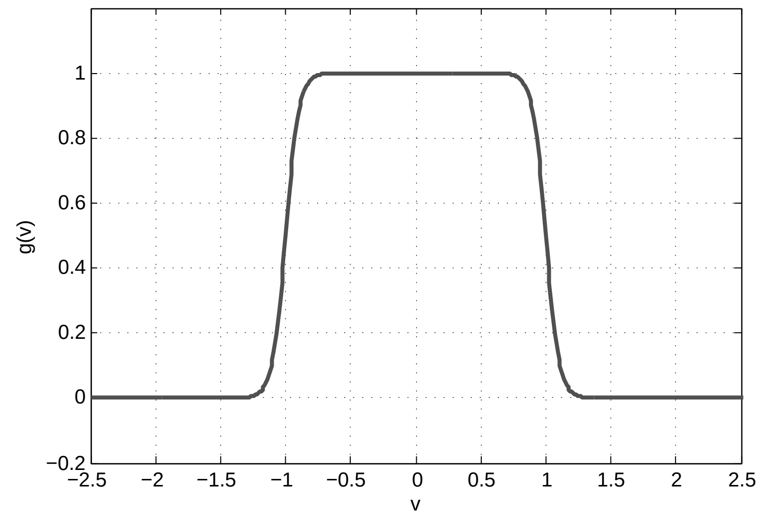


Figure 3.9: *The function $g(v)$ for $\gamma_1 = 10$ and $\gamma_2 = 1$.*

assume that the node states can take values between 0 and 1, node 1 takes the value 0.25 and node 2 the value 0.75, in this case $v_{12}\,[k]$ of neighbor nodes 1 and 2 equals to $\frac{0.75-0.25}{0.75} = 0.6677$. Now assume that the node states can take values from a wider range between 0 and 100, node 1 takes the value 25 and node 2 the value 75, $v_{12}\,[k]$ equals also to $\frac{75-25}{75} = 0.6677$. It is here obvious that for different range of input values the relative difference $v_{ij}\,[k]$ returns us an equal value. This is advantageously for us because we should not modify γ_1 and γ_2 in algorithm for any change in input range of node states. Now we put $v_{ij}\,[k]$ in function $g(v)$. It is obvious that the function equals to 0 for values greater than γ_2 or smaller than $-\gamma_2$ ($|\gamma_2| < v_{ij}\,[k]$) and equals to 1 for values between γ_2 and $-\gamma_2$ ($-\gamma_2 < v_{ij}\,[k] < \gamma_2$). γ_1 defines the switching speed between 0 and 1. We know that in transient phase MH weights and in asymptotic phase the CXO weights are faster therefore we define γ_2 so that in transient phase the function $g(v_{ij}\,[k])$ give us because of larger value of $v_{ij}\,[k]$, a value equal to 0 and respectively $W_{ij}{}'\,[k] = W_{MH,ij}$ (see (3.49)) and in asymptotic phase because of smaller value of $v_{ij}\,[k]$, a value equal to 1 and respectively $W_{ij}{}'\,[k] = W_{CXO,ij}$ (see (3.49)). In the switching time between transient and asymptotic phase, we have a combination of MH and CXO weights.

4 Dynamic Average Consensus

4.1 Introduction

In this chapter we will present the dynamic average consensus (AC) and its corresponding algorithms. This algorithms and their structures were introduced in [8].

We study at the beginning of this chapter in section 4.2, the *linear dynamic AC algorithm*. It will be studied in two different parts: the *first-order algorithm* in Section 4.2.1 and the *nth-order algorithm* in Section 4.2.2. Thereafter, we present in Section 4.3 the *nonlinear dynamic AC algorithm* and we try to improve the performance by using nonlinear methods as considered in Section 3.7.

4.2 Linear Dynamic Average Consensus Algorithm

In chapter 3 we considered sensors which measure a constant local value. This value is time independent and does not change over all iterations. Now we consider the case that an arbitrary sensor i measure at time k a local signal $s_i[k]$. In comparison to the static case the measured signal is not constant and can change over different iterations (time dependent). The average of this measured local signals is as follow:

$$\overline{s}[k] = \frac{1}{N}\sum_{i=1}^{N} s_i[k]. \tag{4.1}$$

Our goal is that each sensor reaches asymptotically this average signal.

We consider again the sensor network of Fig. 3.1. The state of node i is denoted by $x_i[k]$. Node i measure the local signal $s_i[k]$. At each iteration the neighbors of node i which are listed in neighbor set $\mathcal{N}_i$, send their states to it. All nodes apply *dynamic AC algorithm* to read the dynamic average $\overline{s}[k]$. In the next sections we explain the *first-order* and *the nth-order linear dynamic AC algorithm.*

4.2.1 First-Order Linear Dynamic Average Consensus Algorithm

As considered in [8], each node i updates its own state according to the first-order linear dynamic AC algorithm as follow:

$$x_i[k+1] = W_{ii}x_i[k] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j[k] + \Delta s_i[k] \qquad i = 1,\dots,N, \tag{4.2}$$

It is obvious that we have here in comparison to linear static AC an additional part $\Delta s_i[k]$. The $\Delta s_i[k]$ is defined as follow:

$$\Delta s_i[k] = s_i[k] - s_i[k-1], \tag{4.3}$$

and the initial state of each node i is defined as below:

$$x_i[0] = 0, \qquad \text{for } i = 1,\dots,N. \tag{4.4}$$

As defined in [8], the generic generic form of (4.2) is:

$$\mathbf{x}[k+1] = \mathbf{W}\mathbf{x}[k] + \Delta\mathbf{s}[k], \tag{4.5}$$

where $\Delta\mathbf{s}[k] = \mathbf{s}[k] - \mathbf{s}[k-1]$ with $\mathbf{s}[k] = (s_1[k], \dots s_N[k])^T$ as local signal vector con-

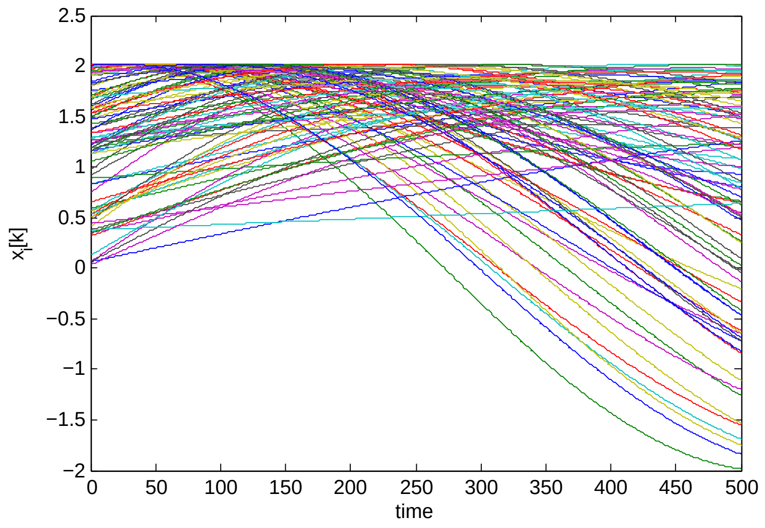


Figure 4.1: *Node measurements over time.*

sisting of local signals measured by each node. The weight matrix $\mathbf{W}$ has the constraint $\mathbf{W} \in \mathcal{J}$ defined in (3.6) and must satisfy the convergence conditions defined in (3.12), (3.13), and (3.14). This applied algorithm is called as first-order linear dynamic AC algorithm.

We consider now an example for first-order linear dynamic AC algorithm. The node measurements over time are illustrated in Fig. 4.1 so that each colored line denotes the measurement of one of this nodes. The measurements are different sinus signals with different amplitudes, frequencies, and phases. The average signal of this measurements over time is shown in Fig. 4.2.

The Fig. 4.3 shows the states of nodes versus the number of iterations. Each colored line refers to state of a single node. It can be seen in Fig. 4.3 that the node states have in transient phase larger differences with each other but after some iterations these differences are smaller and as shown in Fig. 4.2 the node states converge to an estimation of average signal.

In Fig. 4.4, we see the MSE versus number of iterations. It can be seen that the MSE decreases in the beginning until it reaches a value. Thereafter, in tracking phase, it changes slowly. It is shown in Fig. 4.4(a) that the method with CXO weights has the best tracking behavior followed by BC, MH and at the end the method with MD weights which has the worst tracking behavior. In transient phase, as shown in Fig. 4.4(b), the method with CXO weights is faster than other methods.

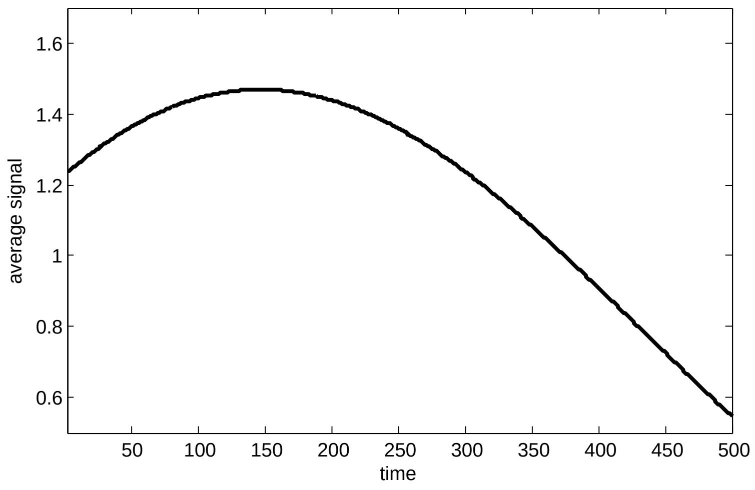


Figure 4.2: *Average signal over time.*

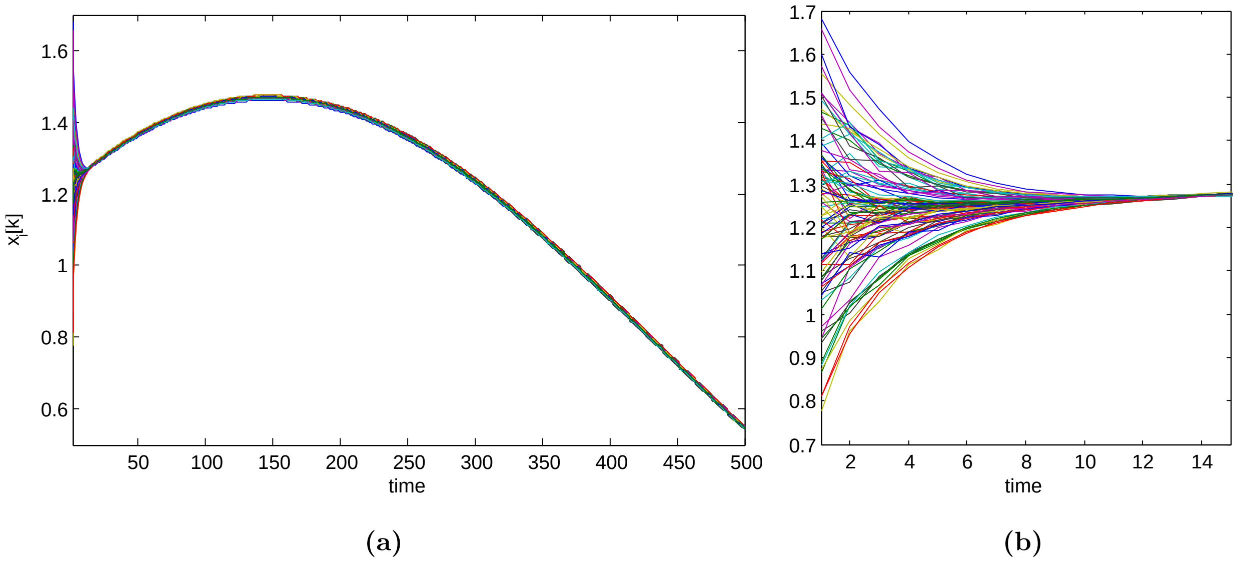


Figure 4.3: *States of nodes versus number of iterations with (a) first-order dynamic AC of a network and (b) its zoomed plot for the first 15 iterations.*

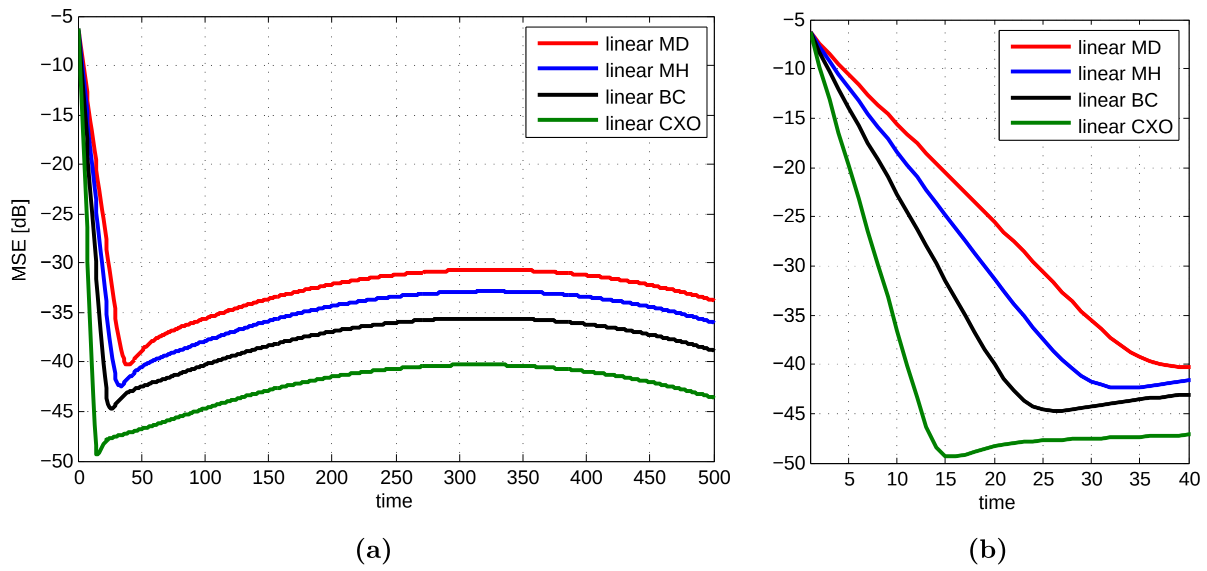


Figure 4.4: *(a) MSE of first-order linear dynamic AC algorithm versus number of iterations and (b) its zoomed plot for the first 40 iterations.*

4.2.2 nth-Order Linear Dynamic Average Consensus Algorithm

We use here the nth-order linear dynamic AC algorithm to update the state of each node. First we consider the second-order algorithm from [8],

$$\begin{aligned} x_i^{[2]}[k+1] &= W_{ii}x_i^{[2]}[k] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j^{[2]}[k] + x_i^{[1]}[k+1], \\ x_i^{[1]}[k+1] &= W_{ii}x_i^{[1]}[k] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j^{[1]}[k] + \Delta^{[2]}s_i[k], \qquad i=1,\dots,N, \end{aligned} \tag{4.6}$$

with $x_i^{[1]}[k]$ and $x_i^{[2]}[k]$ (or $x_j^{[1]}[k]$ and $x_j^{[2]}[k]$) as the first-order and second-order states of each node i (or each node j) and $\Delta^{[2]}s_i[k] = \Delta s_i[k] - \Delta s_i[k-1]$. The second-order linear dynamic AC algorithm can be extended to higher-order as considered in [8] and reach the following relations:

$$\begin{aligned} x_i^{[l]}[k+1] &= W_{ii}x_i^{[l]}[k] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j^{[l]}[k] + x_i^{[l-1]}[k+1], \qquad l=1,\dots,n, \\ x_i^{[1]}[k+1] &= W_{ii}x_i^{[1]}[k] + \sum_{j\in\mathcal{N}_i} W_{ij}x_j^{[1]}[k] + \Delta^{[n]}s_i[k], \qquad i=1,\dots,N, \end{aligned} \tag{4.7}$$

with $x_i^{[l]}$ (or $x_j^{[l]}$) the lth-order states of node i (or node j) and $\Delta^{[n]}s_i[k]$ defined as

$$\Delta^{[n]}s_i[k] = \Delta^{[n-1]}s_i[k] - \Delta^{[n-1]}s_i[k-1], \tag{4.8}$$

where $\Delta^{[1]}s_i[k] = \Delta s_i[k]$. The initial state of each node is defined:

$$x_i^{[l]}[0] = 0, \qquad \text{for } i = 1, \dots, N. \tag{4.9}$$

According to [8], the generic form of (4.7) can be written as

$$\begin{aligned} \mathbf{x}^{[l]}[k+1] &= \mathbf{W}\mathbf{x}^{[l]}[k] + \mathbf{x}^{[l-1]}[k+1], \qquad l = 1, \dots, n, \\ \mathbf{x}^{[1]}[k+1] &= \mathbf{W}\mathbf{x}^{[1]}[k] + \Delta^{[n]}\mathbf{s}[k], \end{aligned} \tag{4.10}$$

where $\mathbf{x}^{[l]}[k] = \left(x_1^{[l]}[k], \dots x_N^{[l]}[k]\right)^T$ with $l = 1, \dots, n$ is the lth-order state vector consisting of the lth-order states of the nodes at discrete time k and $\Delta^{[n]}\mathbf{s}[k] = \Delta^{[n-1]}\mathbf{s}[k] - \Delta^{[n-1]}\mathbf{s}[k-1]$ with $\mathbf{s}[k] = (s_1[k], \dots s_N[k])^T$ as local signal vector consisting of local signals measured by each node. The weight matrix $\mathbf{W}$ has again the constraint $\mathbf{W} \in \mathcal{J}$ defined in (3.6) and must satisfy the convergence conditions defined in (3.12), (3.13), and (3.14). This applied algorithm is called as nth-order linear dynamic AC algorithm.

4.3 Nonlinear Dynamic Average Consensus Algorithm

We try to improve the performance of dynamic AC algorithm by changing the linear form to nonlinear form. Therefore, we apply the weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ which its elements are defined in (3.43). This weight matrix is dependent on state vector $\mathbf{x}[k]$ and changes over different times. We put it in (4.5) and get the state vector update for the first-order nonlinear dynamic algorithm:

$$\mathbf{x}[k+1] = \mathbf{W}'\big(\mathbf{x}[k]\big)\,\mathbf{x}[k] + \Delta\mathbf{s}[k], \tag{4.11}$$

For higher-order algorithms we put the weight matrix $\mathbf{W}'\big(\mathbf{x}[k]\big)$ in (4.10) as follow:

$$\begin{aligned} \mathbf{x}^{[l]}[k+1] &= \mathbf{W}'\big(\mathbf{x}[k]\big)\,\mathbf{x}^{[l]}[k] + \mathbf{x}^{[l-1]}[k+1], \qquad l = 1, \dots, n, \\ \mathbf{x}^{[1]}[k+1] &= \mathbf{W}'\big(\mathbf{x}[k]\big)\,\mathbf{x}^{[1]}[k] + \Delta^{[n]}\mathbf{s}[k], \end{aligned} \tag{4.12}$$

This applied algorithm is called as *nth-order nonlinear dynamic AC algorithm.* It is obvious that in linear dynamic algorithm the elements of weight matrix are constant but in nonlinear dynamic algorithm, this elements changes through a nonlinear function over different times.

5

Simulation Results

5.1 Simulation Setup

We show in this chapter the results of different simulations settings. Our goal is to study the performance of different average consensus (AC) algorithms in the sense of convergence speed in transient and asymptotic phases. To describe the convergence speed we use the MSE because a faster convergence leads to faster decrease of MSE. We define the MSE for static AC, normalized and averaged over all nodes and scenarios denoted by $\epsilon\left[k\right]$ as follow:

$$\epsilon\left[k\right]=\frac{\sum_{i,q}\left(x_{i,q}\left[k\right]-\overline{s}_q\right)^2}{\sum_q\overline{s}_q^2}, \tag{5.1}$$

and for dynamic AC as,

$$\epsilon\left[k\right]=\frac{\sum_{i,q}\left(x_{i,q}\left[k\right]-\overline{s}_q\left[k\right]\right)^2}{\sum_q\overline{s}_q\left[k\right]^2}, \tag{5.2}$$

where i denotes the node and q the scenario. $x_{i,q}[k]$ represents the state of node i in scenario q and $\overline{s}_q$ (or $\overline{s}_q\left[k\right]$) the average signal in same scenario. This two equations are similar with difference that the average signal in the second one is time dependent. We can write the MSE for static AC in vector form for each scenario q as follow:

$$\epsilon_q\left[k\right]=\frac{||\mathbf{x}_q\left[k\right]-\overline{s}_q\mathbf{1}||_2^2}{||\overline{s}_q\mathbf{1}||_2^2}, \tag{5.3}$$

number of nodes	$N = 100$
number of scenarios	150 scenarios
measured local values for static AC	uniform distributed between 0 and 1 ($\mathcal{U}(0,1)$)
measured local signals for dynamic AC	measured from a time-varying field constructed by Fourier basis
graph	random geometric graph with radius $r = 0.38$

Table 5.1: *Simulation parameters which are mostly used in this work.*

and also for dynamic AC in vector form for each scenario q:

$$\epsilon_q[k] = \frac{||\mathbf{x}_q[k] - \bar{s}_q[k]\,\mathbf{1}||_2^2}{||\bar{s}_q[k]\,\mathbf{1}||_2^2}, \tag{5.4}$$

where $\mathbf{x}_q[k]$ is the state vector consisting of the states of the nodes at discrete time k and scenario q. We calculate the average of MSE over all scenarios and get $\epsilon[k]$.

In this chapter we use usually for simulations the settings as depicted in Tbl. 5.1. We consider 100 nodes which are distributed uniformly on unit square. The graph that is used, is a random geometric graph and the nodes which are in a distance of r = 0.38 of each other are connected. In static AC algorithm the local values which are measured by nodes are sampled from a uniform distribution $\mathcal{U}(0,1)$. In dynamic AC algorithm the measured local signals are from a time-varying field constructed by Fourier basis. The results are averaged over 150 different scenarios. If we do not mention any changes of this settings, we choose them for our simulations.

5.2 Linear Static Average Consensus Algorithm

Here we consider from Section 3.3 the linear static AC algorithm with different weight matrices which are presented in Section 3.6. We will see which influence has the number of edges and also the number of nodes on the performance of algorithm. The number of edges are defined with radius r of random geometric graph. The behavior of the algorithm if the nodes measure values from a spatial field, is also considered. Finally we study the performance of algorithm in different graphs and network structures.

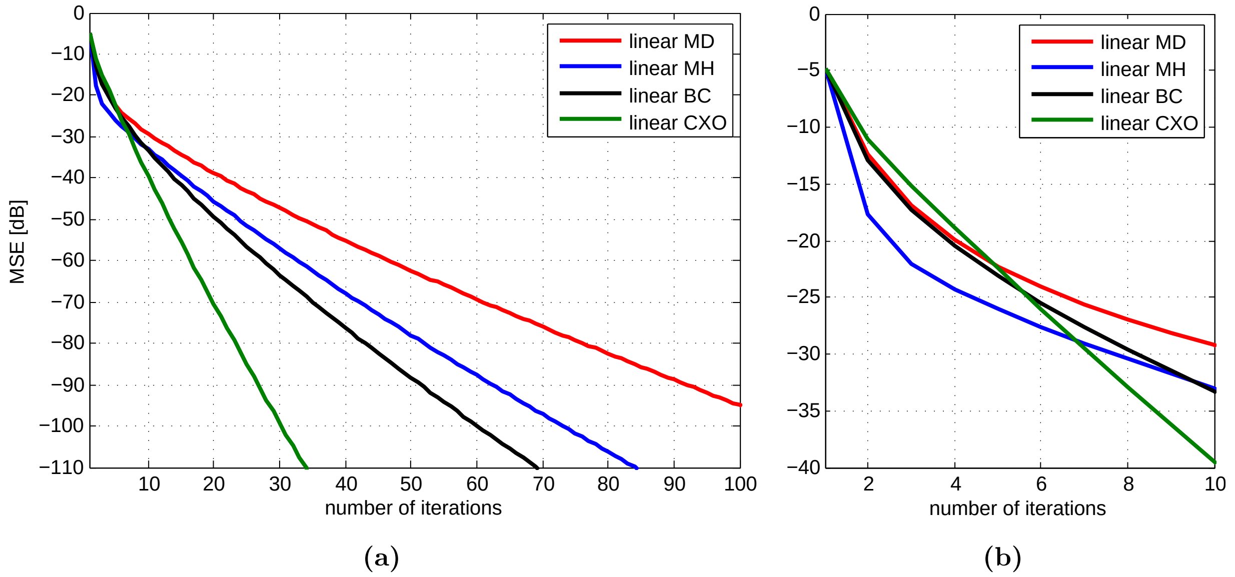


Figure 5.1: *(a) MSE of linear static AC algorithm versus number of iterations with $r = 0.38$ and (b) its zoomed plot for the first 10 iterations.*

5.2.1 Influence of Number of Edges on Performance

We consider here three different networks with $N = 100$ nodes uniformly distributed on unit square. The graph that we use is a random geometric graph and the initial states are sampled from a uniform distribution $\mathcal{U}(0, 1)$.

In the First network, we have for settings a random geometric graph with $r = 0.38$. Each node has a degree equal to 31.1975 and there are altogether 1559.88 edges (averaged over all nodes and scenarios).

The MSE of such a network versus number of iterations, is illustrated in Fig. 5.1. Each colored line denotes a different method to design the weight matrix. We consider the convergence speed in two different phases, asymptotic and transient phase.

It can be seen in Fig. 5.1(a) that in asymptotic phase the method with CXO is the fastest method and the method with MD is the slowest. It is also seen that with BC we have a faster convergence speed than with MH but both behave worse than with CXO. Although the method with CXO is the fastest method in asymptotic phase but we see in Fig. 5.1(b) that in transient phase is this method the slowest. In transient phase MH leads to a lower MSE and therefore is faster. First after $7th$ iteration the method with CXO begins to have a lower MSE. Therefore if in this graph a precision up to about 10^{-3} is desired, it is more advantageous to use MH weights than the CXO weights.

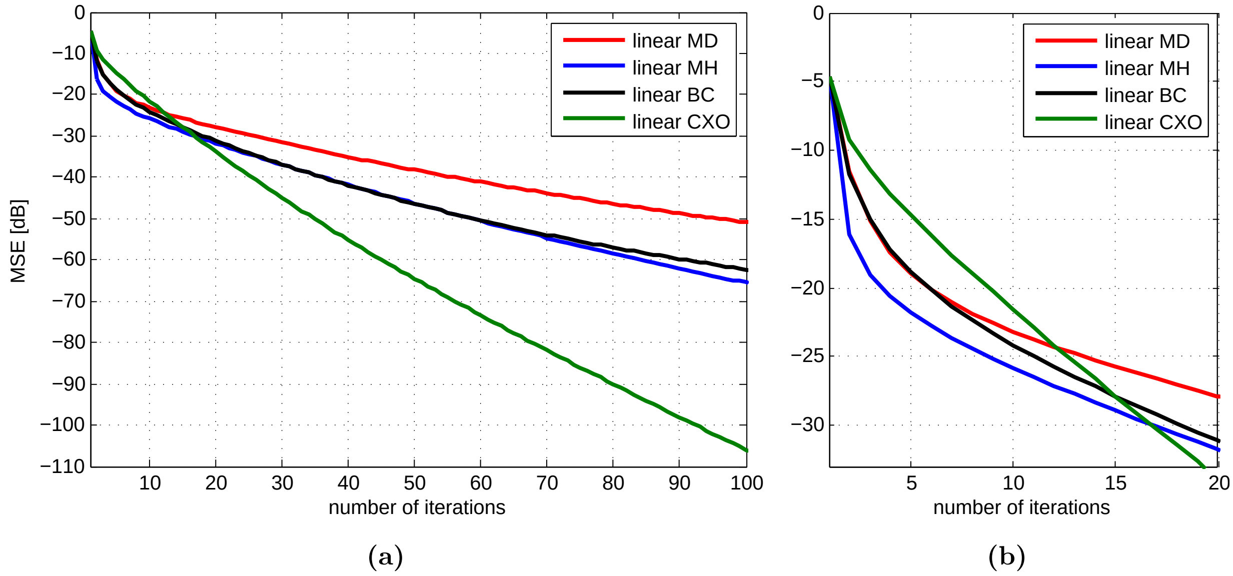


Figure 5.2: *(a) MSE of linear static AC algorithm versus number of iterations with $r = 0.26$ and (b) its zoomed plot for the first 20 iterations.*

Now let consider what happens if we reduce the radius of random geometric graph down to $r = 0.26$, it leads to our second network. In this network each node has a degree equal to 16.5512 and there are altogether 827.56 edges.

It can be seen in Fig. 5.2(a) that again the method with CXO is asymptotic the fastest and the method with MD asymptotic the slowest method but here in converse to last network, we have in asymptotic phase with MH a faster convergence speed than with BC. This is because of lower number of edges which leads to a faster convergence with MH. As shown in Fig. 5.2(b), in transient phase until about $16th$ iteration and for a precision up to about 10^{-3}, MH leads to a lower MSE and therefore, it is faster.

Now in the third network we increase the radius up to $r = 0.8$. In this network the degree of each node equals to 84.2332. We have altogether 4211.66 edges.

We see in Fig. 5.3(a) that similar to last studied two networks the method with CXO is asymptotic the fastest method and the method with MD the slowest. Here is again like the first network the convergence speed with BC asymptotically faster than with MH. As shown in Fig. 5.3(b), the main difference to the first and second networks is that here in transient phase is not more the method with MH the fastest but the method with CXO. Therefore, for a higher number of edges the algorithm with CXO has a faster convergence speed in transient and also asymptotic phase.

The MSE versus different radiuses of random geometric graph can be seen for transient phase in Fig. 5.4(a) and for asymptotic phase in Fig. 5.4(b). It can be seen that

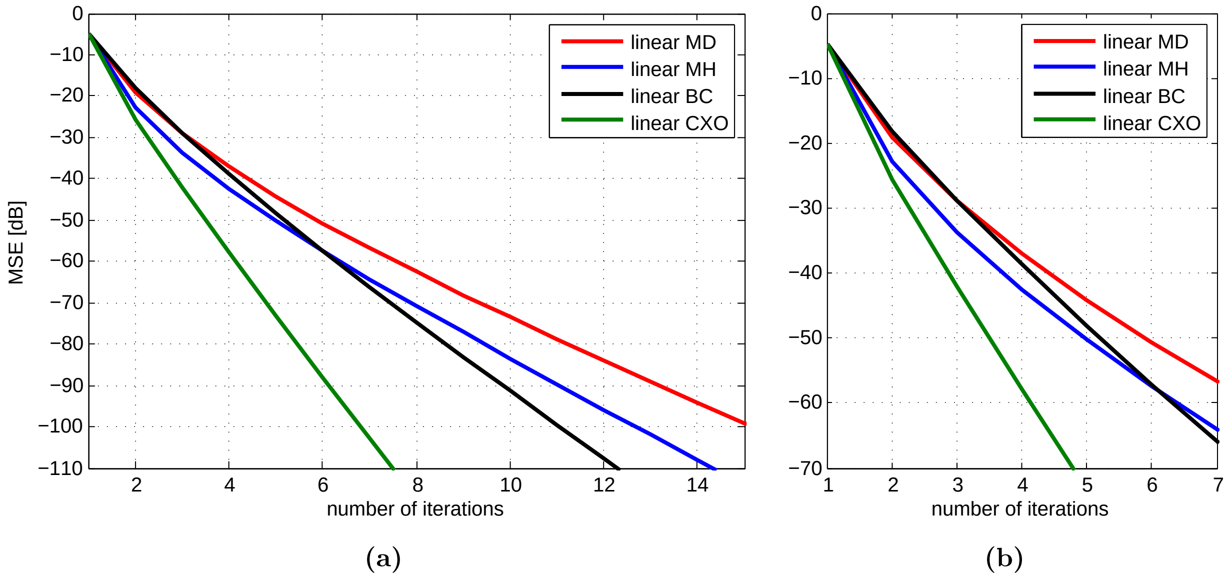


Figure 5.3: *(a) MSE of linear static AC algorithm versus number of iterations with $r = 0.8$ and (b) its zoomed plot for the first 7 iterations.*

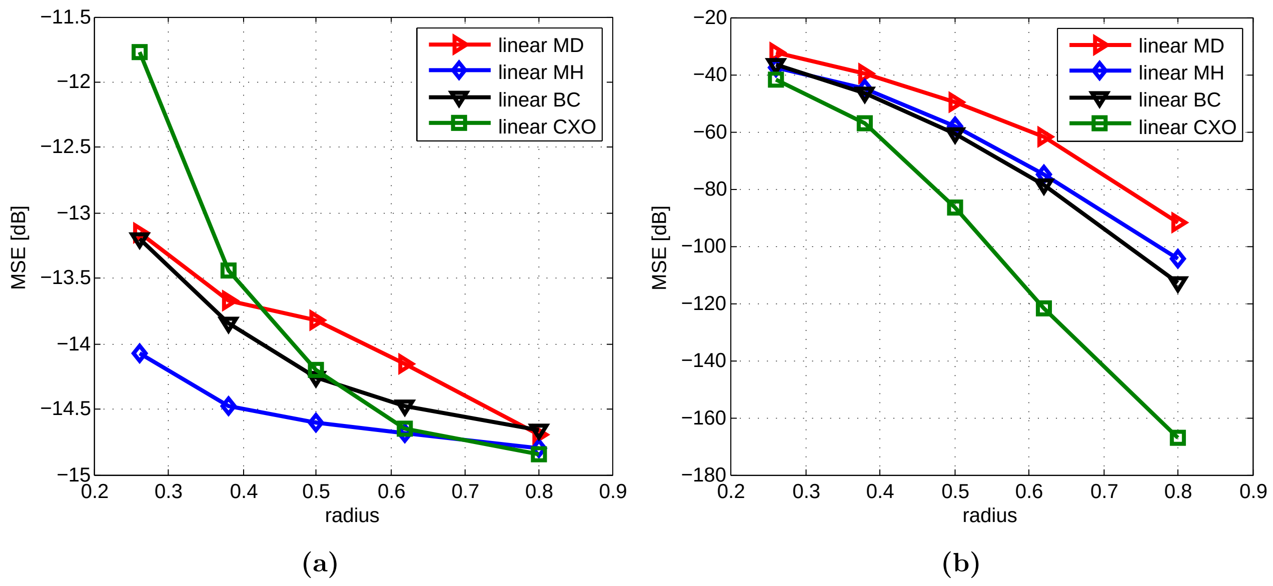


Figure 5.4: *MSE of linear static AC algorithm versus different radiuses of random geometric graph in (a) transient phase and in (b) asymptotic phase.*

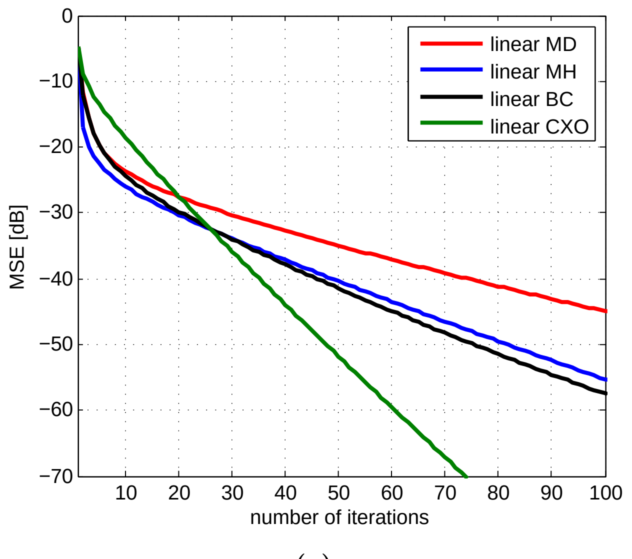
in transient phase the method with MH has the lowest MSE with radiuses up to $r = 0.6$ but with larger radiuses the MSE is lower for the method with CXO. In asymptotic phase, the method with CXO has always the lowest MSE and its distance to other methods is higher for larger radiuses.

We come from this simulations to following conclusions: In asymptotic phase is the method with CXO the fastest method and the method with MD the slowest. It is also seen that the methods with MH and BC weights behave both worse than CXO. If we have a low number of edges between nodes, MH leads to a asymptotic faster convergence than BC otherwise is always BC faster. In Transient phase is the method with MH the fastest method but for a high number of edges is the method with CXO not just in asymptotic phase but also in transient phase the fastest method. It is also seen that for a higher number of edges (increase of radius r) with each defined weight matrix of AC algorithm, the convergence speed is higher. For example in Figures 5.1, 5.2, and 5.3 the MSE of method with CXO equals approximately to 10^{-10} respectively in about $30th$, $92th$, and $7th$ iteration.

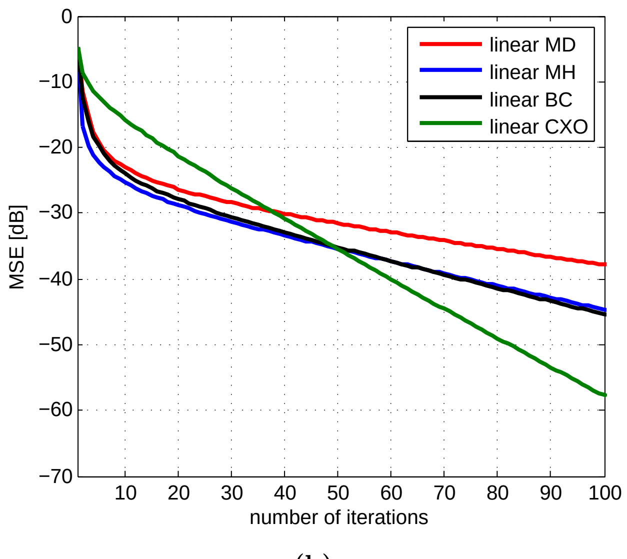
5.2.2 Influence of Number of Nodes on Performance

In Section 5.2.1 we considered the influence of number of edges on performance of algorithm. Now we want consider what happens if we change the number of nodes in our network and which influence it has on performance.

In Fig. 5.1 we considered a network with $N = 100$ nodes. Now we increase the number of nodes up to $N = 200$ and $N = 300$ and achieve two new additional networks. It is shown in Fig. 5.5(a) and (b) the MSE of each network versus number of iterations. The comparison of this three figures shows us that the increase of number of nodes, decrease the convergence speed. For example in Figures 5.1, 5.5(a), and 5.5(b) the MSE equals approximately to 10^{-5} respectively in about $13th$, $47th$ and $82th$ iteration. This is because of higher number of nodes. Each node send its state to its neighbors and this is done for all other nodes, so if the network is larger (higher number of nodes), the distribution of informations through the network needs a longer time which causes a slower convergence speed. It is also seen that the method with MH has in transient phase for a longer time (higher number of iterations), lower MSE than other methods.



(a)



(b)

Figure 5.5: *MSE of linear static AC algorithm versus number of iterations with (a) $N = 200$ nodes ($r = 0.2$) (b) $N = 300$ nodes ($r = 0.15$).*

5.2.3 Measurement from a Spatial Field

We sampled in previous simulations the initial states of nodes from a uniform distribution $\mathcal{U}(0, 1)$. In that case each node can take randomly a value between 0 and 1. Instead of that we assume, each node which is located uniformly on a unit square measure dependent on its location, a value from a spatial field. This spatial field is shown in Fig. 5.6. The location of each node on the unit square can be defined by horizontal axes r_1 and r_2. The vertical axis define the value of field in each point. It can be seen that the neighbor nodes measure values which are not too different of each other, i.e., the difference between initial states of neighbor nodes are smaller than the difference of initial states of nodes which are not located near each other. As shown in Fig. 5.7, It causes in transient phase either a higher or equal MSE for the method with MH than the method with CXO or just for a short time a lower MSE, but if we choose the initial states from a uniform distribution, as we saw in Fig. 5.1(b), the method with MH is in transient phase faster than all other methods (lower MSE).

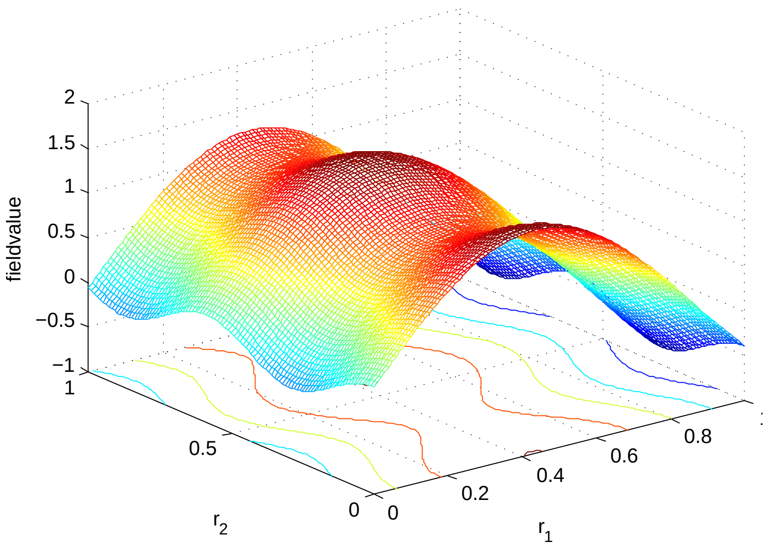
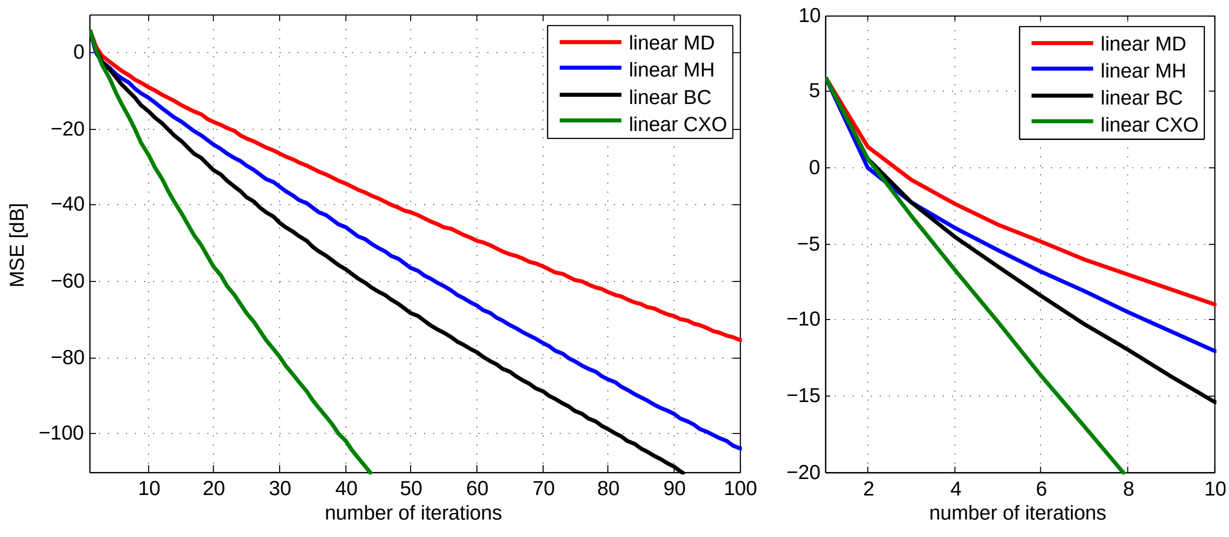


Figure 5.6: *Spatial field*



(a) **(b)**

Figure 5.7: *(a) MSE of linear static AC algorithm versus number of iterations with r = 0.38 with node measurement from a spatial field and (b) its zoomed plot for the first 10 iterations.*

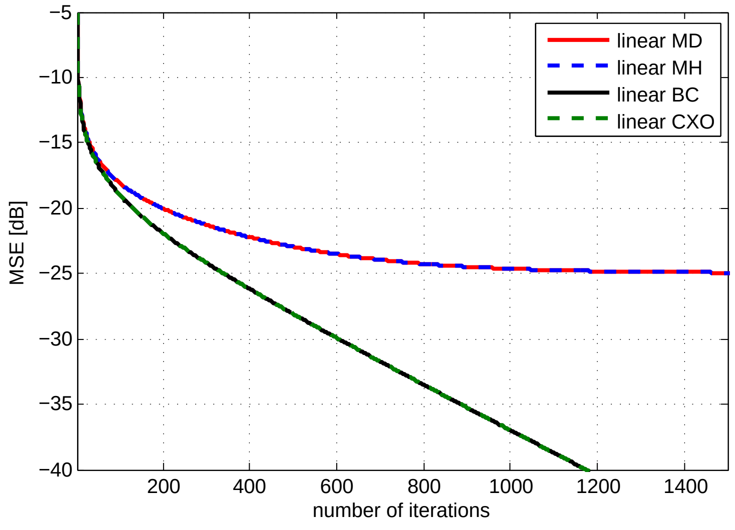


Figure 5.8: *MSE of linear static AC algorithm versus number of iterations in ring graph.*

5.2.4 Impact of the Graph Topology

We considered random geometric graphs with different number of nodes, radiuses and initial states. Now we want to study the influence of different graphs and network structures on performance of linear static AC algorithm with different weight matrices. In this graphs and network structures we have 100 nodes and the initial states are sampled from uniform distribution $\mathcal{U}(0, 1)$.

Ring Graph: We know as mentioned in Section 2.3.3 that in a ring graph all nodes are connected together in a ring form and each node is connected just with two neighbor nodes thus, the number of edges is too low.

We see in Fig. 5.8 the MSE versus number of iterations in a ring graph. It can be seen that in ring graph we achieve the same performance with MD and MH. We know from (3.22) and (3.33) that for weight matrix with MD, each edge weight W_{ij} for $\{i, j\} \in \mathcal{E}$ equals to $1/d_{\max}$. $d_{\max}$ is the maximum indegree over all nodes and equals in ring graph to 2. For a Weight matrix with MH, each edge weight W_{ij} for $\{i, j\} \in \mathcal{E}$ equals to $1/\max\{d_i, d_j\}$ where d_i and d_j are the indegrees of two neighbor nodes i and j and they have in a ring graph the same value 2. Thus, we can represent the elements

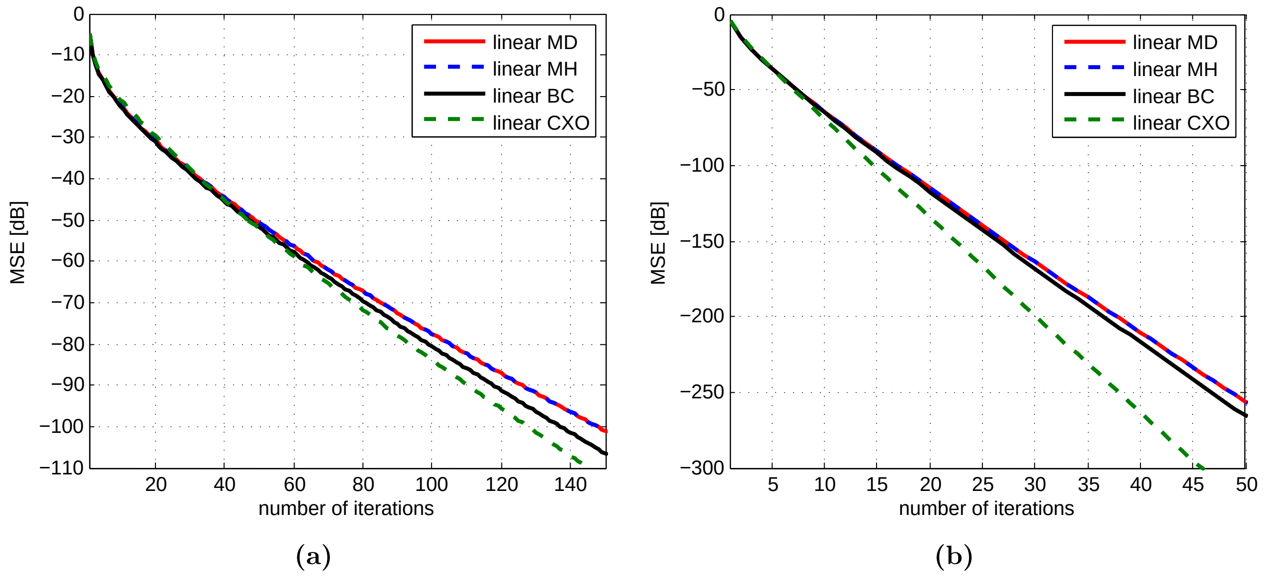


Figure 5.9: *MSE of linear static AC algorithm versus number of iterations in random regular graph with (a) $d = 3$ and (b) $d = 10$.*

of weight matrix with MD ($W_{ij,\text{MD}}$) or with MH ($W_{ij,\text{MH}}$) for a ring graph as follow:

$$W_{ij,\text{MD}} = W_{ij,\text{MH}} = \begin{cases} \frac{1}{2}, & \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

It is also seen that the performance is similar for the methods with BC and with CXO. In transient phase, we achieve for all methods a similar result. Therefore we have for a ring graph, the best performance with weight matrices with BC or CXO.

Random Regular Graph: As described in Section 2.3.2, a random regular graph $\mathcal{G}(N, d)$ has N nodes and d connected neighbors for each node. The ring graph that we studied, is a regular graph with d=2.

We consider in Fig. 5.9 the MSE versus number of iterations in random regular graphs with $d = 3$ and $d = 10$. It can be seen that because of equal number of neighbors d for each node, the method with MD and the method with MH have the same performance. Thus we can write for elements of weight matrix with MD ($W_{ij,\text{MD}}$)

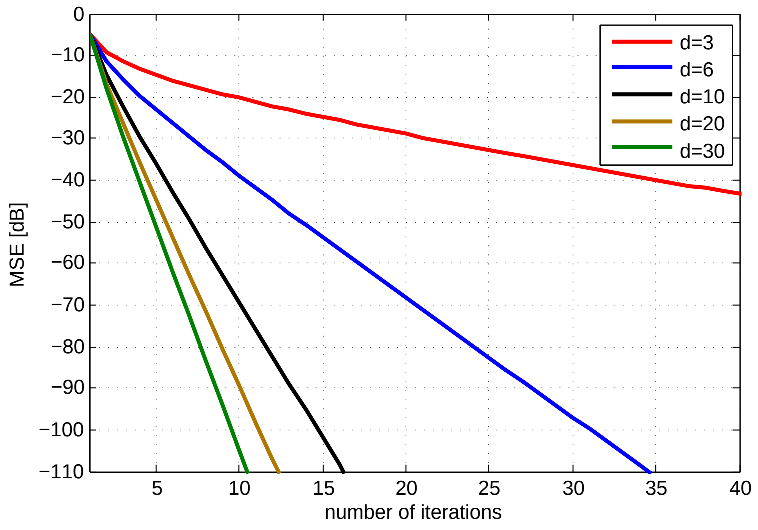


Figure 5.10: *MSE of linear static AC algorithm with CXO versus number of iterations in random regular graph for different values of d.*

or with MH ($W_{ij,\text{MH}}$) in a random regular graph, with some changes of (5.5) as,

$$W_{ij,\text{MD}} = W_{ij,\text{MH}} = \begin{cases} \frac{1}{d}, & \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

It is also seen that in asymptotic phase the method with CXO is the fastest method and the methods with MD and MH are the slowest methods. The method with BC behaves better than the methods with MH and MD but it is worse than the CXO. In transient phase all methods have approximately a similar convergence speed. Therefore we have for a random regular graph the best performance in transient and in asymptotic phase independent of number of d, with the method with CXO.

In Fig. 5.10 we consider the MSE of linear static AC algorithm with CXO versus number of iterations in random regular graphs with different values of d. It can be seen that the convergence speed increases for higher values of d. This can be explained by higher number of edges.

Complete Graph: As we studied the complete graph in Section 2.3.4, each node in this graph is connected to all other nodes, so we have a large number of edges between nodes and each node is connected to $N-1$ other nodes. We can say that a complete graph is a random regular graph with $d = N - 1$. In Fig. 5.11, we can see the MSE of

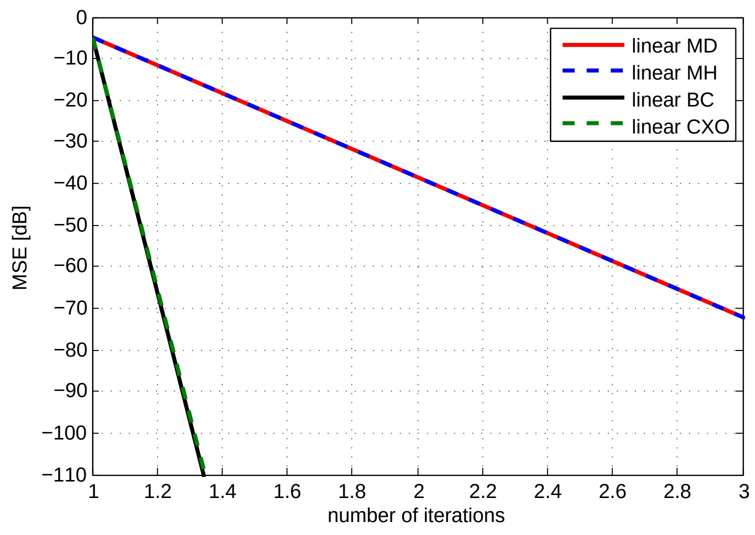


Figure 5.11: *MSE of linear static AC algorithm versus number of iterations in a complete graph.*

linear static AC algorithm versus number of iterations in a complete graph. It can be seen that the methods with MD and MH have the same performance, so the elements of weight matrix with MD ($W_{ij,\mathrm{MD}}$) or with MH ($W_{ij,\mathrm{MD}}$) in a complete graph are as follow:

$$W_{ij,\mathrm{MD}} = W_{ij,\mathrm{MH}} = \begin{cases} \frac{1}{N-1}, & \{i,j\} \in \mathcal{E}, \\ 1 - \sum_{k=1}^{N} W_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

We replaced the parameter d in (5.6) with $N-1$. The weight matrices with BC and CXO have also a similar performance. In comparison with MD and MH, they have a higher convergence speed in transient and in asymptotic phase. Therefore in a complete graph, we can reach the best results with BC or CXO.

Graph with Bottleneck: Let consider two random geometric graphs which are connected together over a single edge. We call this as a graph with bottleneck which is explained in Section 2.3.5.

We see in Fig. 5.12 the states of nodes versus number of iterations. Each colored line denotes a state of a node. It can be seen in Fig. 5.12(b) that in the beginning the node states of each side of the single edge (bottleneck) converge approximately to a common value. The common value of one side is not equal to the common value of other side. It can be seen in Fig. 5.12(a) that after this primary convergence, we have two bundle of lines so that each of them denotes the states of one side. Then

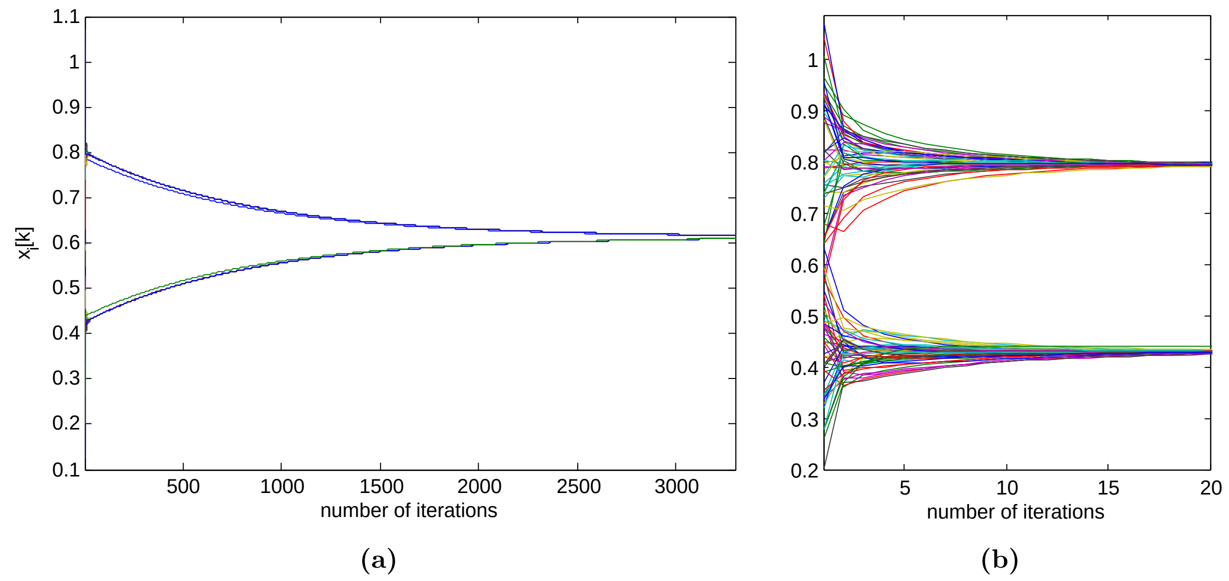


Figure 5.12: *States of nodes versus number of iterations in a graph with bottleneck: (a) linear static AC of a network and (b) its zoomed plot for the first 20 iterations.*

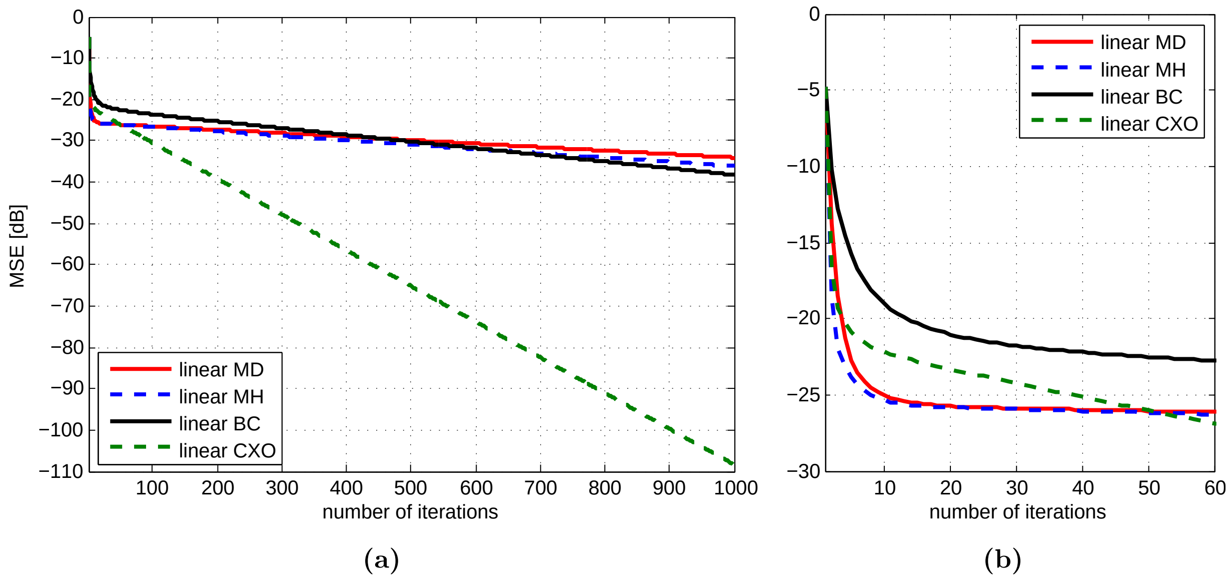


Figure 5.13: *(a) MSE of linear static AC algorithm versus number of iterations in a graph with bottleneck with* $r = 0.38$ *and (b) its zoomed plot for the first 60 iterations.*

after several iterations, this bundles converges to a common value which is the average value. This behavior can be explained as follow: we have a faster convergence in each side because of higher number of edges but the convergence of both sides together to average value is slower because of a single edge between them.

We consider in Fig. 5.13 the MSE of linear static AC algorithm versus number of iterations in a graph with bottleneck with $r = 0.38$. It can be seen in Fig. 5.13(a) that the method with CXO is asymptotic the fastest and the method with MD is the slowest. The method with BC is better than MH but both behave worse than the CXO. It can be seen in Fig. 5.13(b) that in transient phase, the method with MH is the fastest.

5.3 Nonlinear Static Average Consensus Algorithm

Here we consider the nonlinear static AC algorithm defined in Section 3.7.1. We determine the nonlinear function $f(u)$ and its parameters so that a faster convergence speed is reached. After that we study the behavior of algorithm for different radiuses of random geometric graph.

5.3.1 Determination of Parameters of Nonlinear Function

For the nonlinear static AC algorithm, we use as weight matrix, the matrices defined in Section 3.6. We let for the asymptotic phase, the weight matrix elements unchanged or multiplied with a constant factor but in transient phase we modulate them appropriately. The varying of the elements of weight matrix is done through a nonlinear function $f(u)$ defined in (3.48),

$$f(u) = \frac{\tanh(\theta_1 u)\,\theta_2}{u},$$

with the parameters θ_1, $\theta_2 \in \mathbb{R}_+$. θ_1 and θ_2 define the slope of the curve and their product $\theta_1\theta_2$ defines the maximum value of it (shown in Fig. 3.8).

We put the state difference $u_{ij}[k] = x_j[k] - x_i[k]$ as input of nonlinear function $f(u)$ where $x_i[k]$ and $x_j[k]$ are the states of two neighbor nodes i and j. The difference of states $u_{ij}[k]$ converge to zero because the state of nodes converge to a common value which is the average value. This convergence of $u_{ij}[k]$ to zero causes that $f(u_{ij}[k])$ converge to maximum value $\theta_1\theta_2$. This nonlinear function $f(u_{ij}[k])$ is multiplied with edge weights W_{ij} of linear algorithm with $\{i,j\} \in \mathcal{E}$ (defined in (3.43)). Thus, the weights of nonlinear algorithm change over different iterations and converge in asymptotic phase to weights of linear algorithm or to a multiple of them.

Now we want to determine the best possible values of this parameters for each weight matrix. We determine at the beginning the product $\theta_1\theta_2$ (the maximum value of nonlinear function $f(u)$) for a random geometric graph with radius $r = 0.38$ and for $N = 100$ nodes which their initial states are sampled from a uniform distribution $\mathcal{U}(0,1)$. The results are here averaged over 150 scenarios. We see in Fig. 5.14 the MSE of nonlinear static AC algorithm versus number of iterations for different weight matrices. Each colored line denotes the MSE of nonlinear algorithm for a defined product of θ_1 and θ_2. We choose for this parameters an equal value which is the root of their product value. Now we try to find the line which exhibits the lowest MSE and take its value as best possible product of θ_1 and θ_2. It can be seen that this product for nonlinear algorithm equals with MD to 1.7, with MH to 1.6, with BC and CXO to 1.

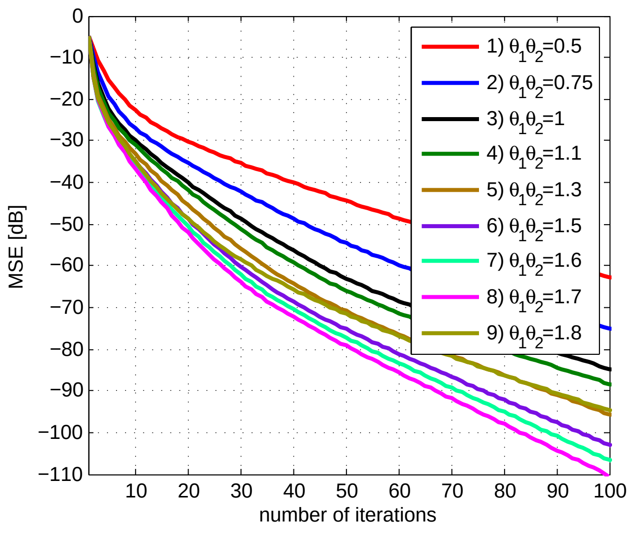
Our next goal is to find the exact values of θ_1 and θ_2 in the determined best possible product of them. As shown in Fig. 5.15, we have here again the MSE versus number of iterations. The results are here averaged over 300 scenarios. Each colored line denotes different values for θ_1 and θ_2 so that their product equals to the values that we found in Fig. 5.14. It can be seen that with θ_1 and θ_2 values of lines number 8 and 9, we do not reach the lowest MSE but with other lines we have an approximately equal MSE. To find the best possible values between these lines, we calculate the MSE averaged over all iterations and come to following results[1]:

- MD: line number 5 with $\theta_1 = 1.3038$ and $\theta_2 = 1.3038$ and an averaged MSE equal to $1.2608.10^{-3}$
- MH: line number 6 with $\theta_1 = 2$ and $\theta_2 = 0.8$ and an averaged MSE equal to $1.1802.10^{-3}$
- BC: line number 6 with $\theta_1 = 2$ and $\theta_2 = 0.5$ and an averaged MSE equal to $1.2741.10^{-3}$
- CXO: line number 5 with $\theta_1 = 1$ and $\theta_2 = 1$ and an averaged MSE equal to $1.4533.10^{-3}$

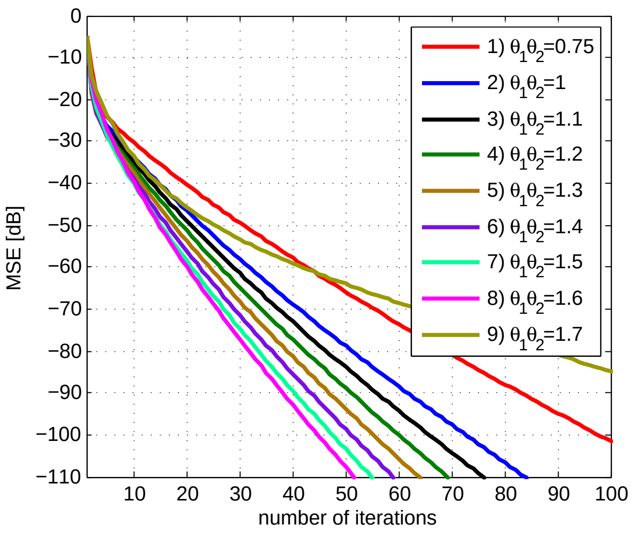
The nonlinear functions with best possible parameters θ_1 and θ_2 for different methods are demonstrated in Fig. 5.16.

These results are reached with a radius of $r = 0.38$, if we change this radius, we should calculate anew the parameters θ_1 and θ_2 in the same way. The appropriate

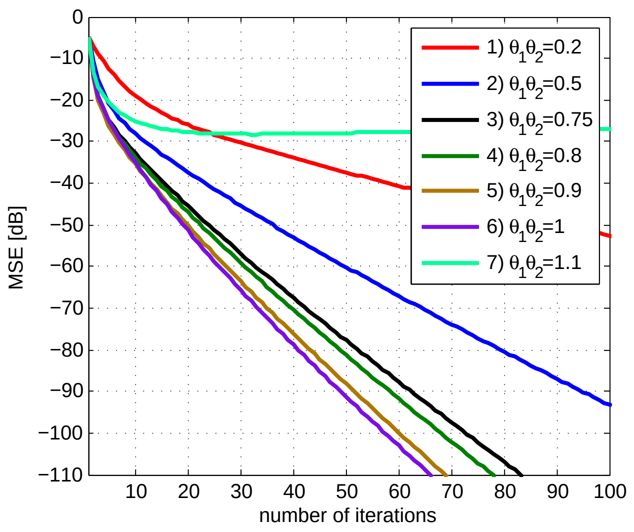
[1]It is to be attended that each calculated averaged MSE for each weight matrix, is just in favor to find the best possible values of θ_1 and θ_2 and have an overview of corresponding MSE but it is not appropriate to compare different weight matrices because we used for each simulation different random local values and communication links.



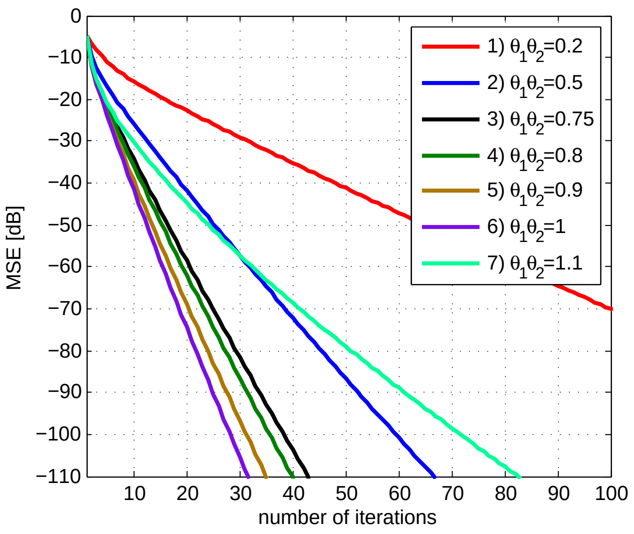
(a)



(b)

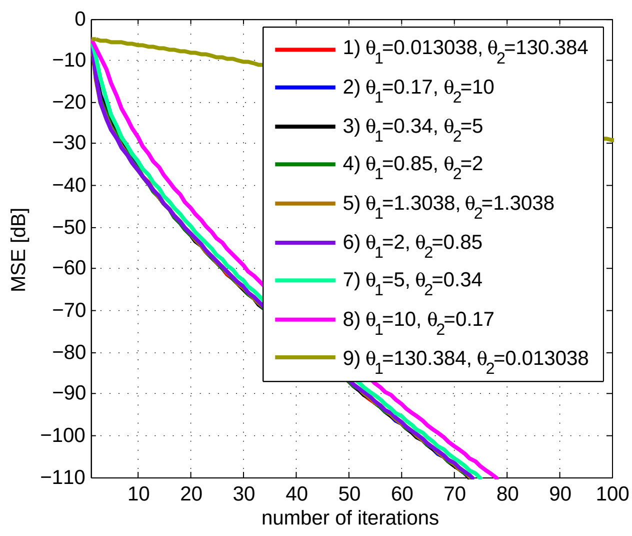


(c)

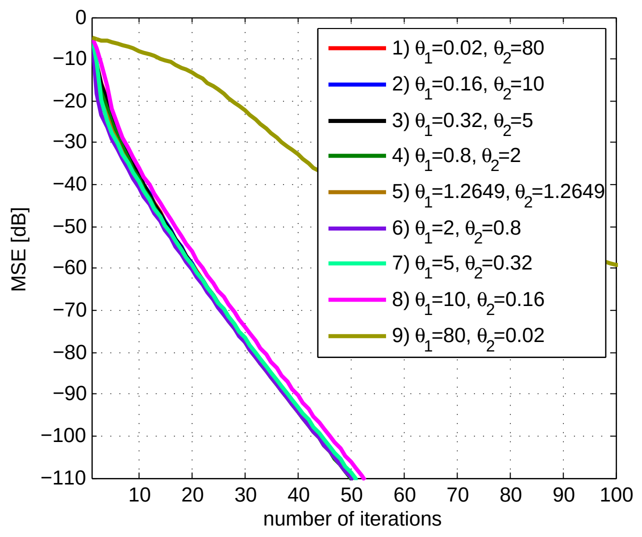


(d)

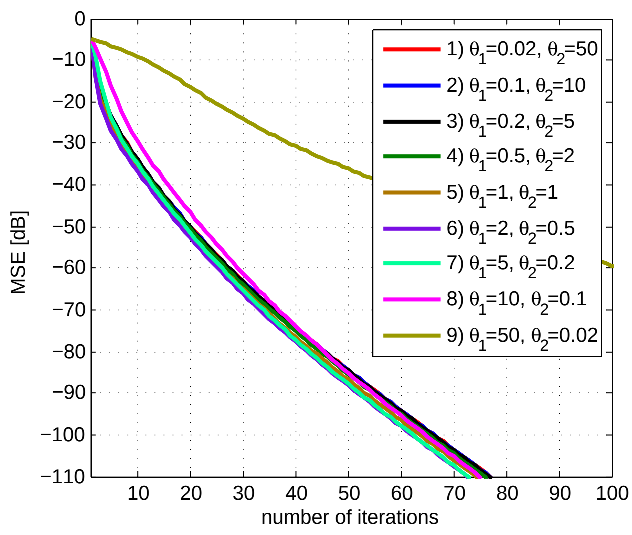
Figure 5.14: *MSE of nonlinear static AC algorithm versus number of iterations to define the best multiply of θ_1 and θ_2 with (a) MD, (b) MH, (c) BC, and (d) CXO (r=0.38).*



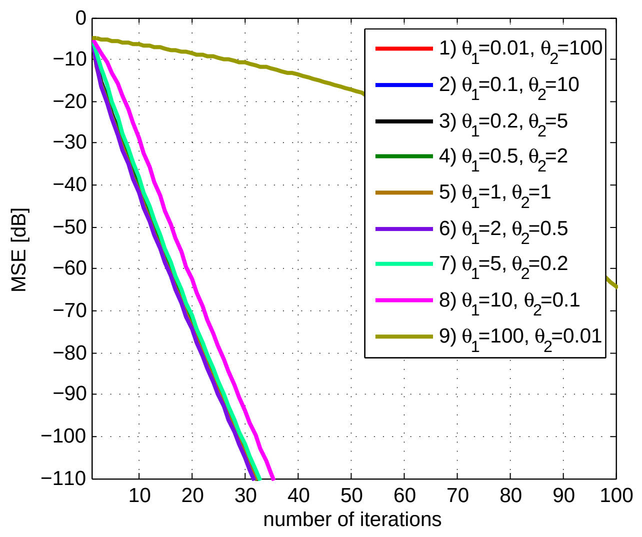
(a)



(b)



(c)



(d)

Figure 5.15: *MSE of nonlinear static AC algorithm versus number of iterations to define θ_1 and θ_2 with (a) MD, (b) MH, (c) BC, and (d) CXO (initial states sampled from $\mathcal{U}(0,1)$ and r=0.38).*

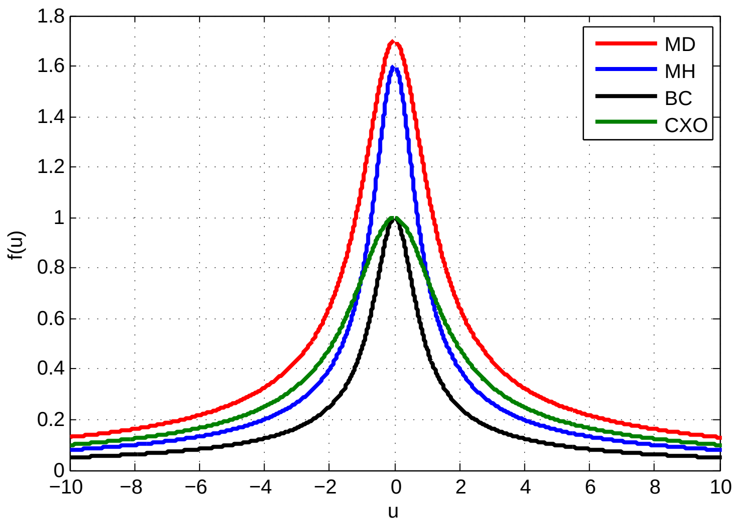


Figure 5.16: *Nonlinear functions with best possible parameters θ_1 and θ_2 for different weight matrices.*

	$r = 0.26$	$r = 0.38$	$r = 0.8$
MD	$\theta_1 = 1.3038, \theta_2 = 1.3038$	$\theta_1 = 1.3038, \theta_2 = 1.3038$	$\theta_1 = 1.1832, \theta_2 = 1.1832$
MH	$\theta_1 = 2, \theta_2 = 0.75$	$\theta_1 = 2, \theta_2 = 0.8$	$\theta_1 = 1.1401, \theta_2 = 1.1401$
BC	$\theta_1 = 2, \theta_2 = 0.5$	$\theta_1 = 2, \theta_2 = 0.5$	$\theta_1 = 2, \theta_2 = 0.5$
CXO	$\theta_1 = 1, \theta_2 = 1$	$\theta_1 = 1, \theta_2 = 1$	$\theta_1 = 1, \theta_2 = 1$

Table 5.2: *θ_1 and θ_2 for radiuses $r = 0.26$, $r = 0.38$, $r = 0.8$ and initial states sampled from $\mathcal{U}(0, 1)$ for nonlinear static AC algorithm.*

values of this parameters for radiuses $r = 0.26$ and $r = 0.8$ and also the mentioned radius $r = 0.38$ can be find in Tbl. 5.2.

Now if we sample the initial states from uniform distribution $\mathcal{U}(0, 100)$, instead from $\mathcal{U}(0, 1)$ and the radius of graph remains unchanged ($r = 0.38$), we should find anew the values of θ_1 and θ_2. The best possible product of them for each weight matrix with defined weights remains unchanged (same values as in Fig. 5.14) but we should find the exact values of them in these determined products. It is shown in Fig. 5.17 the MSE versus number of iterations. The results are here averaged over 300 scenarios. It can be seen that we get the lowest MSE for the line number 1 with $\theta_1 = 0.013$ and $\theta_2 = 130.384$ for MD, $\theta_1 = 0.02$ and $\theta_2 = 80$ for MH, $\theta_1 = 0.02$ and $\theta_2 = 50$ for BC, and $\theta_1 = 0.01$ and $\theta_2 = 100$ for CXO.

We come to conclusion that for any changes of radius of graph and respectively the structure of it and also any changes in range of initial states of nodes, we must

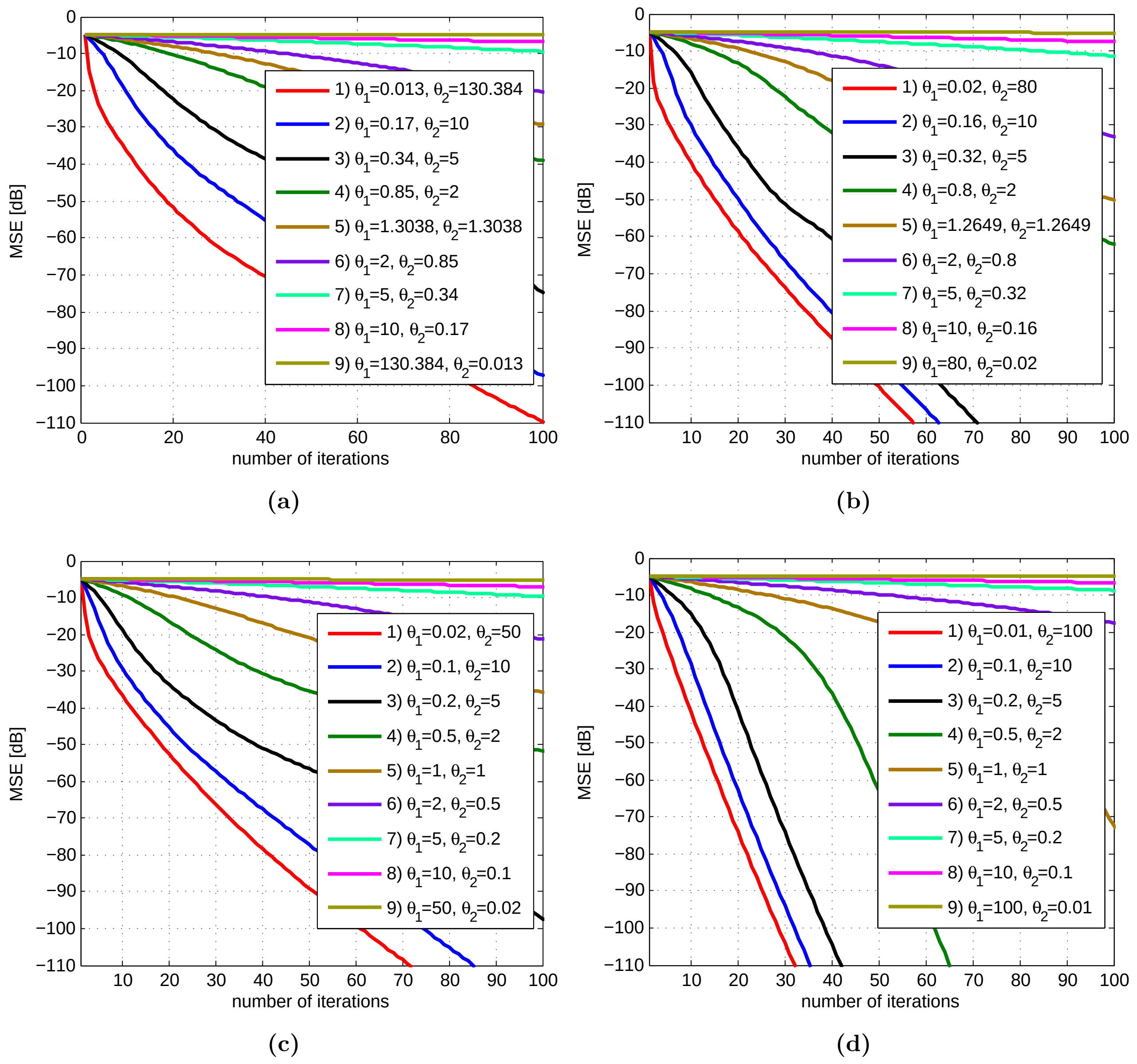


Figure 5.17: *MSE of nonlinear static AC algorithm versus number of iterations to define θ_1 and θ_2 with (a) MD, (b) MH, (c) BC, and (d) CXO (initial states sampled from $\mathcal{U}(0, 100)$ and r=0.38).*

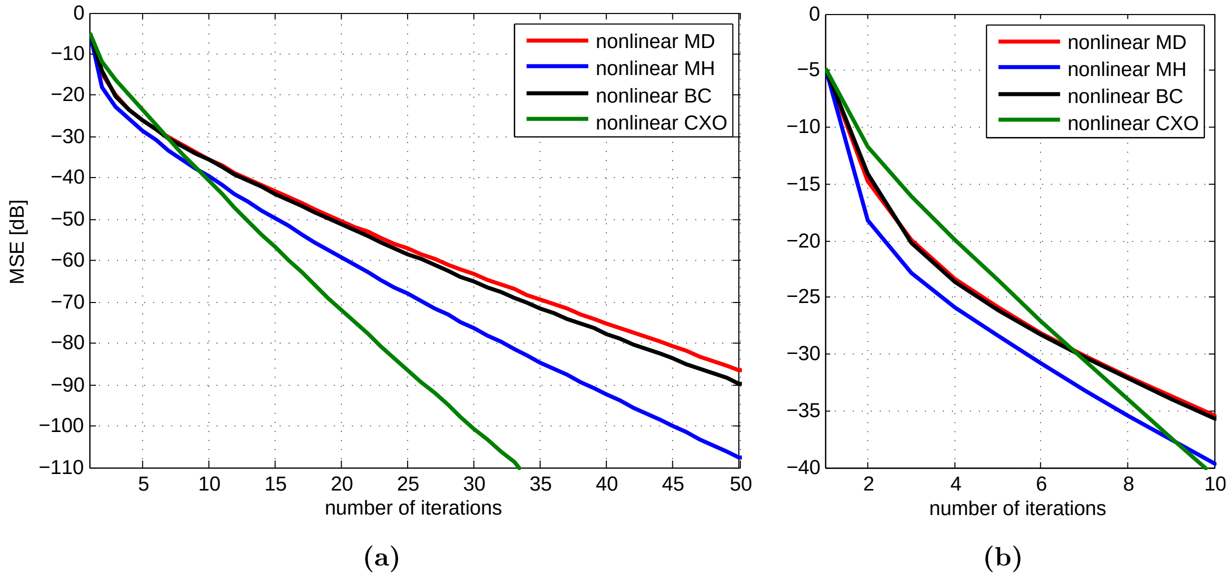


Figure 5.18: *(a) MSE of nonlinear static AC algorithm versus number of iterations with $r = 0.38$ and (b) its zoomed plot for the first 10 iterations.*

determine anew the parameters θ_1 and θ_2.

5.3.2 Influence of Number of Edges on Performance

We consider here the nonlinear static AC algorithm with calculated parameters from last Section 5.3.1. The networks that we study here, are the same three different networks of Section 5.2.1.

In the first network, we have for settings a random geometric graph with $r = 0.38$. It is shown in Fig. 5.18 the MSE of nonlinear static AC algorithm versus number of iterations. Each colored line denotes a nonlinear algorithm with different matrices.

It can be seen in Fig. 5.18(a) that method with CXO is asymptotic the fastest and the method with MD asymptotic the slowest. The method with MH is in this phase better than BC but both behave worse than the CXO. If we compare this results with linear algorithm in Fig. 5.1(a), we see a great improvement for MH through nonlinearity because in that figure was the method with MH slower than method BC but here is vice versa. It is shown in Fig. 5.18(b) that although the method with CXO is the fastest method in asymptotic phase but it is the slowest one in transient phase. In this phase, the method with MH has the lowest MSE and the fastest convergence speed.

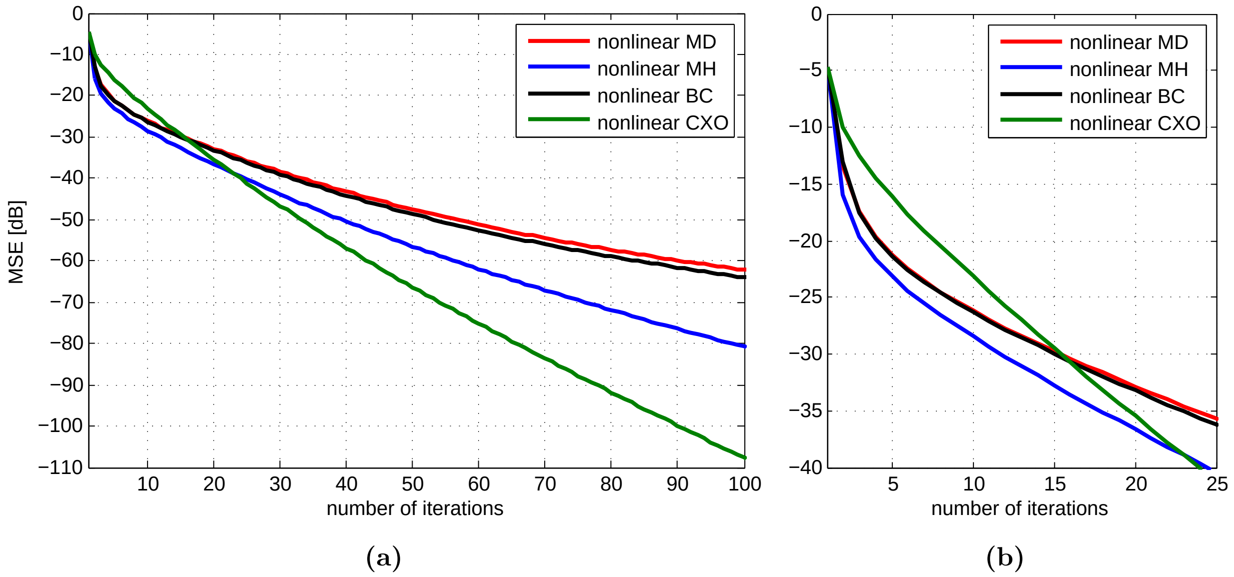
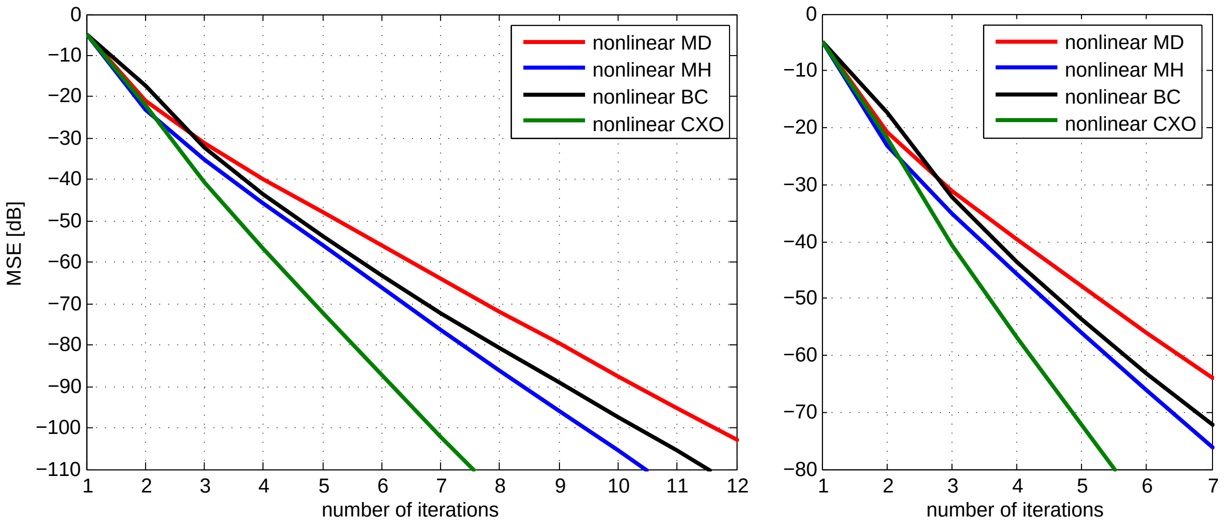


Figure 5.19: *(a) MSE of nonlinear static AC algorithm versus number of iterations with $r = 0.26$ and (b) its zoomed plot for the first 25 iterations.*

In the second network we decrease the radius down to $r = 0.26$ and hence we have a lower number of edges between nodes. It is shown in Fig. 5.19 that We reach for convergence speed similar results like the first network. The method with CXO is asymptotic the fastest and after that respectively the methods with MH, BC and MD are the asymptotic fastest methods. In transient phase has again the method with MH a faster convergence speed.

In the third network we increase the radius up to $r = 0.8$ and thus we have a higher number of edges. It can be seen in Fig. 5.20 that in asymptotic phase we have again with CXO the fastest convergence speed followed by MH, BC and at the end MD. In transient phase we have in opposite of last network with CXO a higher or approximately equal convergence speed to MH.

We come to conclusion that for nonlinear static AC algorithm is the method with CXO independent of radius of random geometric graph asymptotic the fastest method. In transient phase just for higher number of edges, we can reach with CXO a better or similar results to MH otherwise the method with MH is in this phase always faster.



(a) **(b)**

Figure 5.20: *(a) MSE of nonlinear static AC algorithm versus number of iterations with $r = 0.8$ and (b) its zoomed plot for the first 6 iterations.*

5.4 Nonlinear Static Average Consensus Algorithm with Combination of two Weight Matrices

We consider now the nonlinear static AC algorithm with combination of two weight matrices with MH and CXO which is defined in Section 3.7.2. It is shown in Figures 5.1 and 5.2 that in transient phase the method with MH and in asymptotic phase the method with CXO are the fastest methods. We combine this different methods so that in transient phase the MH and in asymptotic phase the CXO are applied. To characterize this behavior we use the $v_{ij}[k]$ from (3.50),

$$v_{ij}[k] = \frac{x_j[k] - x_i[k]}{\max(x_i[k], x_j[k])},$$

which is a time dependent relative difference between states $x_i[k]$ and $x_j[k]$ of two neighbor nodes i and j. In transient phase the MSE and also the absolute value of $v_{ij}[k]$ decrease with MH faster than with CXO, so we use in this phase MH. After some iterations with MH, the speed of decrease of MSE and the absolute value of $v_{ij}[k]$ begin to be slower than with CXO, we switch in this point from MH to CXO and call this absolute value of $v_{ij}[k]$ at this time as *switching value.*

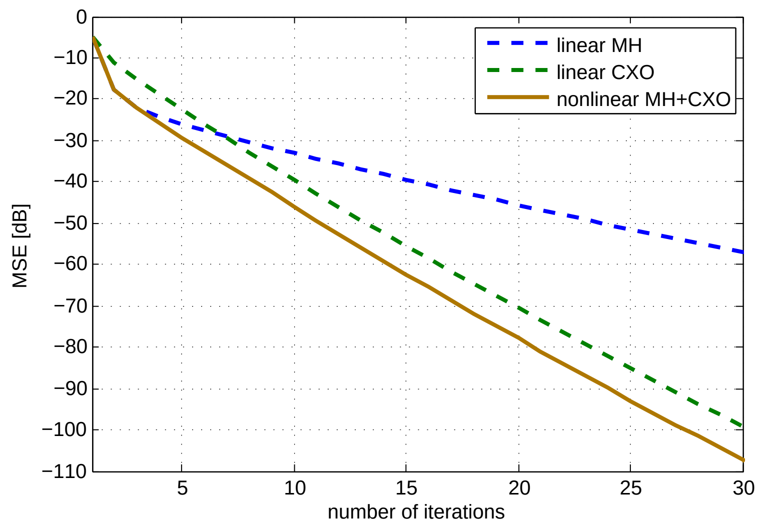


Figure 5.21: *MSE of nonlinear static AC algorithm with combination of MH and CXO versus number of iterations with* $r = 0.38$.

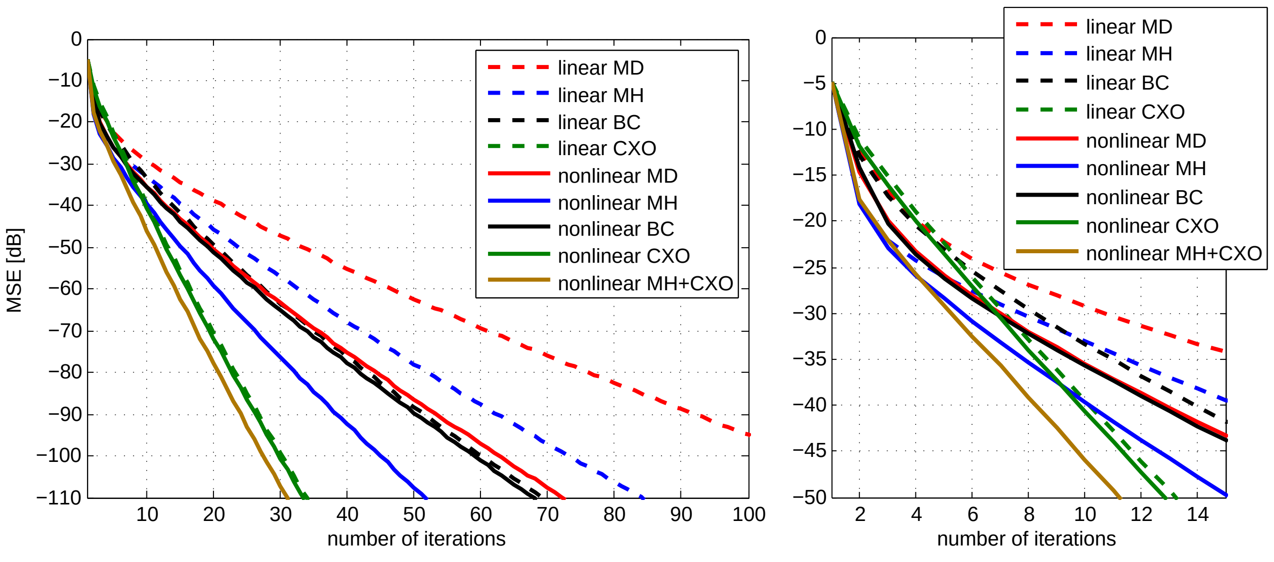
To switch between this weights we use the nonlinear function $g(v)$ defined in (3.51),

$$g(v) = \frac{1}{2} \tanh\left(\gamma_1 \left(|v| - \gamma_2\right)\right) + \frac{1}{2} \,,$$

with the parameters γ_1, $\gamma_2 \in \mathbb{R}_+$. We define with γ_2 in which range of inputs the function equals to 0 and in which range equals to 1 and with γ_1 the slope of the curve by switching between 0 and 1 (switching speed between 0 and 1).

We put the $v_{ij}\,[k]$ as input of nonlinear function $g(v)$ and set γ_2 equal to switching value of $v_{ij}\,[k]$. It causes that the nonlinear function $g(v_{ij}\,[k])$ equals to 1 for $\gamma_2 < v_{ij}\,[k]$ and for $v_{ij}\,[k] < -\gamma_2$ (in transient phase) and equals to 0 for $-\gamma_2 \leq v_{ij}\,[k] \leq \gamma_2$ (in asymptotic phase). Thus as defined in (3.49), the elements of weight matrix of this algorithm can be calculated as a sum of the multiplication of nonlinear function $g(v_{ij}\,[k])$ with MH and the multiplication of $1 - g(v_{ij}\,[k])$ with CXO.

We consider a network of 100 nodes in a random geometric graph with radius r=0.38 and the initial states are sampled from a uniform distribution $\mathcal{U}(0, 1)$. We choose for nonlinear function, $\gamma_1 = 0.18$ and $\gamma_2 = 1000$. It is shown in Fig. 5.21 the MSE of linear static AC algorithm with MH and with CXO and also the MSE of nonlinear static AC algorithm with combination of MH and CXO versus number of iterations. It can be seen that in transient phase the method with MH is faster than the method with CXO but in asymptotic phase is vice versa. Thus the nonlinear algorithm applies in this simulation until third iteration the MH and after that switches to CXO, so we get in transient phase an equal MSE with a algorithm which applies just MH and in



(a) **(b)**

Figure 5.22: *(a) MSE of linear and nonlinear static AC algorithms versus number of iterations with $r = 0.38$ and (b) its zoomed plot for the first 6 iterations.*

asymptotic phase lower MSE than a algorithm with CXO.

We reach with nonlinear static AC with combination of MH and CXO a better result in all cases where the MH is in transient phase faster than the CXO. Therefore as long as this conditions satisfy we can use this algorithm for all different radiuses of random geometric graph. Furthermore the relative form of difference of node states $v_{ij}[k]$ has the advantage that if we change the range of initial states, we should not define anew the parameters γ_1 and γ_2 in converse what we did for parameter θ_1 and θ_2 of nonlinear static AC algorithm.

To compare the performance of all static linear and nonlinear algorithms that we considered until now, We study the Fig. 5.22. We see in this figure for same settings as above, the MSE versus number of iterations for all linear and nonlinear static AC algorithms. It can be seen that we get the best result with nonlinear algorithm with combination of MH and CXO. We see also that the nonlinear algorithms have a better performance than the linear algorithms. The most improvements for nonlinear algorithms in comparison with linear ones can be seen with the methods with MD and MH.

5.5 Linear Dynamic Average Consensus Algorithm

We discussed the linear dynamic AC algorithm in Section 4.2. We consider here the performance of linear algorithm for measurements from a time-varying spatial field. Furthermore, We study the performance of linear algorithm for different frequencies of this field and also for different orders.

5.5.1 Measurement from Time-Varying Spatial Field

We consider here a network of 100 nodes distributed in a time-varying field constructed with Fourier basis. The Fourier coefficients which we use to construct the field, change over time with sinus signals with different amplitudes and phases but a defined equal frequency. It leads to a time-varying field so that each node measure a local signal from this field. Such a field is demonstrated in Fig. 5.23 on different times. The field should change very slowly (low frequency of sinus signal) otherwise the algorithm will be unstable. The graph that we use is a random geometric graph with radius $r = 0.38$.

In Fig. 5.24, we see the MSE of first-order linear dynamic AC algorithm versus number of iterations. It can be seen that in the beginning, the MSE decreases for all methods. After some iterations it stabilizes around a constant value. It can be seen that in transient phase the method with MH is for a short time faster than other methods but after some iterations the method with CXO will be the fastest method and it stabilizes itself around a lower MSE value in comparison to others. Therefore, the method with CXO has the best tracking behavior and the lowest MSE. The method with MD has the highest MSE. The method with BC has a better tracking behavior than MH but both behave worse than the CXO.

5.5.2 Influence of Frequency of Time-Varying Spatial Field on Performance

As we discussed in last Section, the nodes measure from a time-varying spatial field. The Fourier coefficients that we use to construct the field, change with sinus signals with a defined frequency. Now we want to consider the influence of varying of this frequency on performance of algorithm.

In Fig. 5.25 we see the MSE of first-order linear dynamic AC algorithm with CXO versus number of iterations for different frequencies. The results in this figure are averaged over 50 different scenarios. It can be seen that increases of frequency leads to a higher MSE and also higher fluctuation of it.

The MSE versus different frequencies is represented in Fig. 5.26. The results are

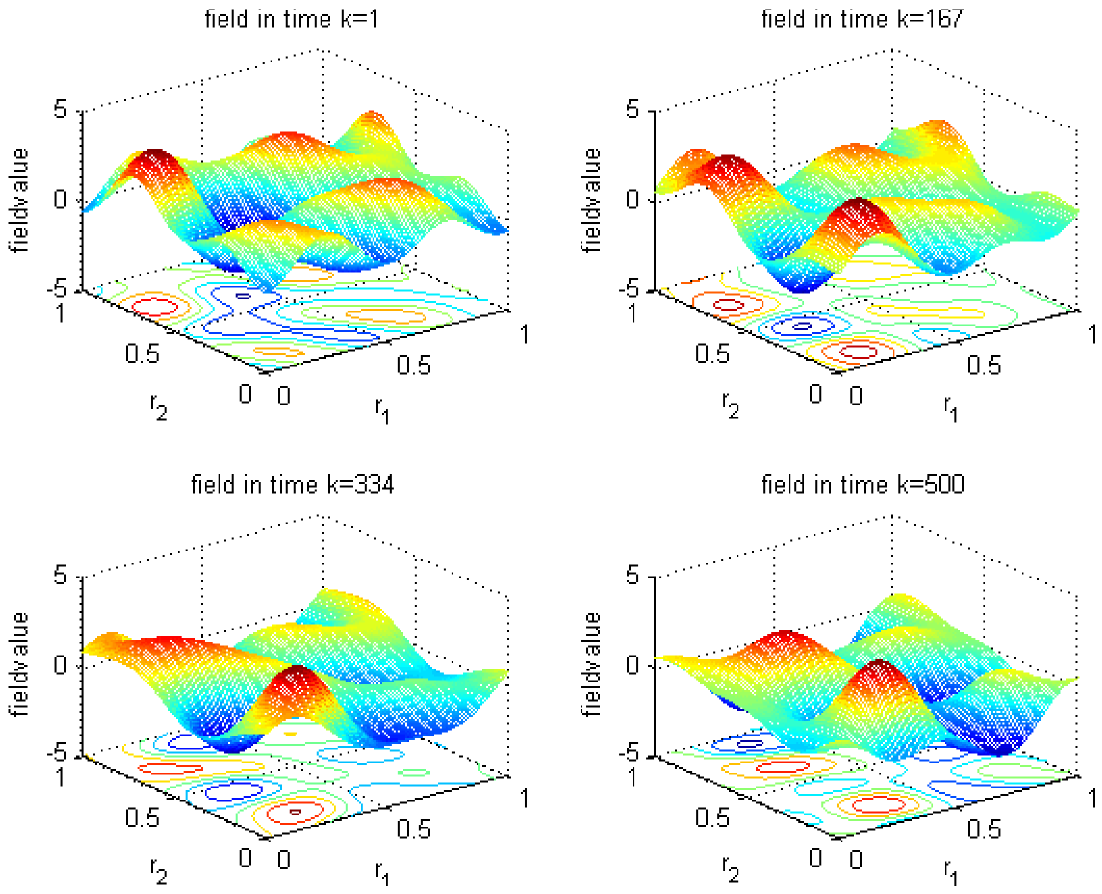
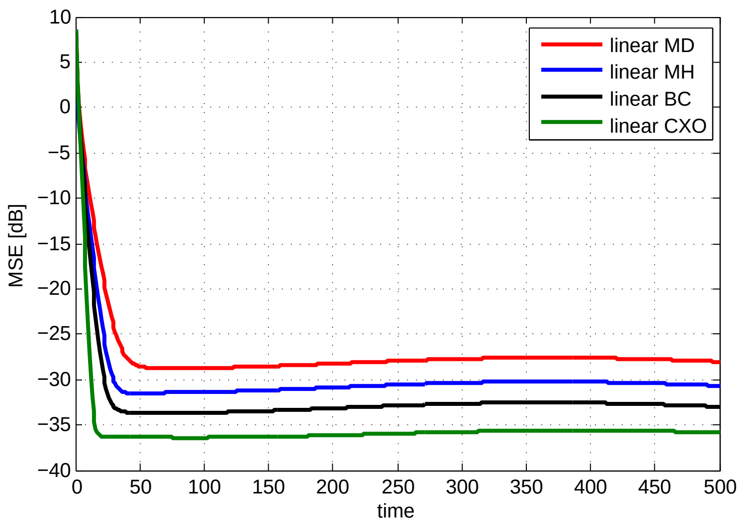
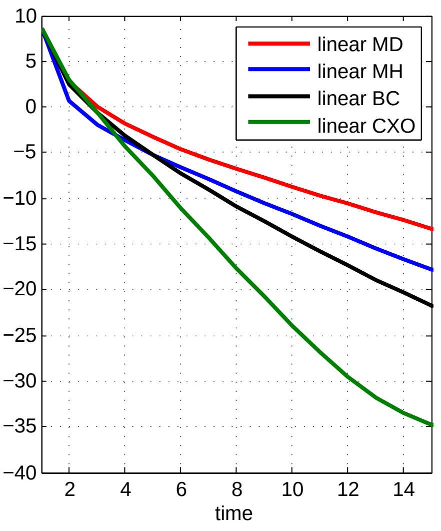


Figure 5.23: *Spatial field*



(a)



(b)

Figure 5.24: *(a) MSE of first-order linear dynamic AC algorithm versus number of iterations with* $r = 0.38$ *and (b) its zoomed plot for the first 15 iterations.*

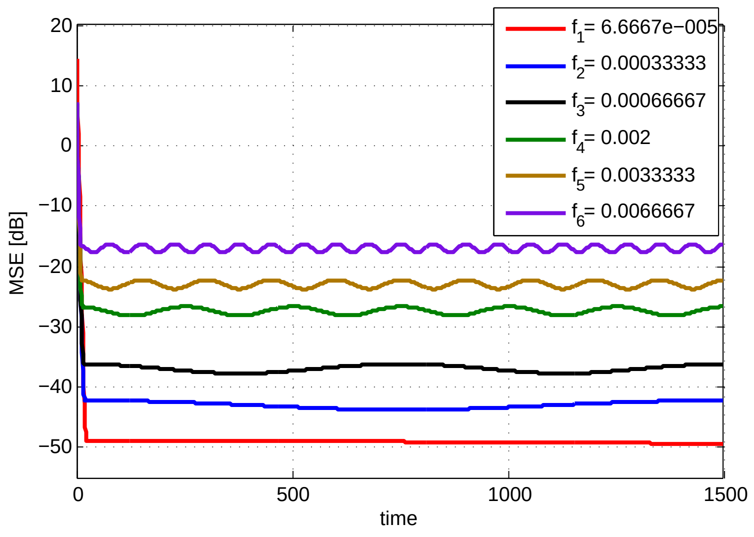


Figure 5.25: *MSE of first-order linear dynamic AC algorithm with CXO versus number of iterations for different frequencies (r=0.38).*

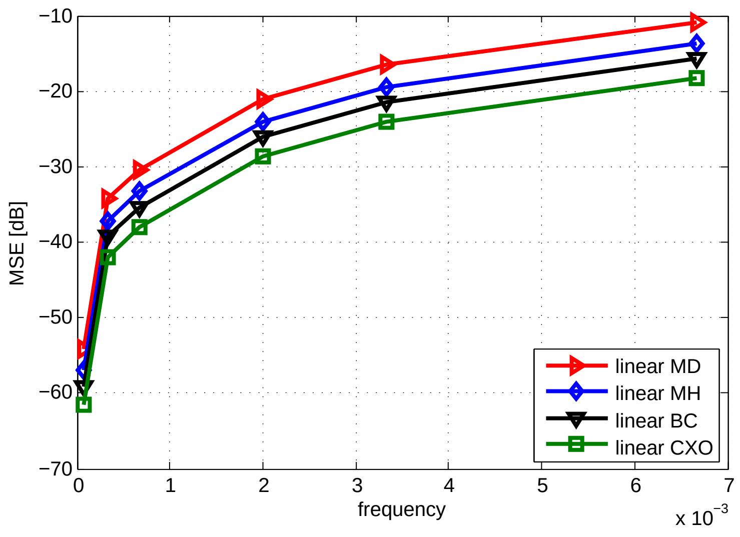
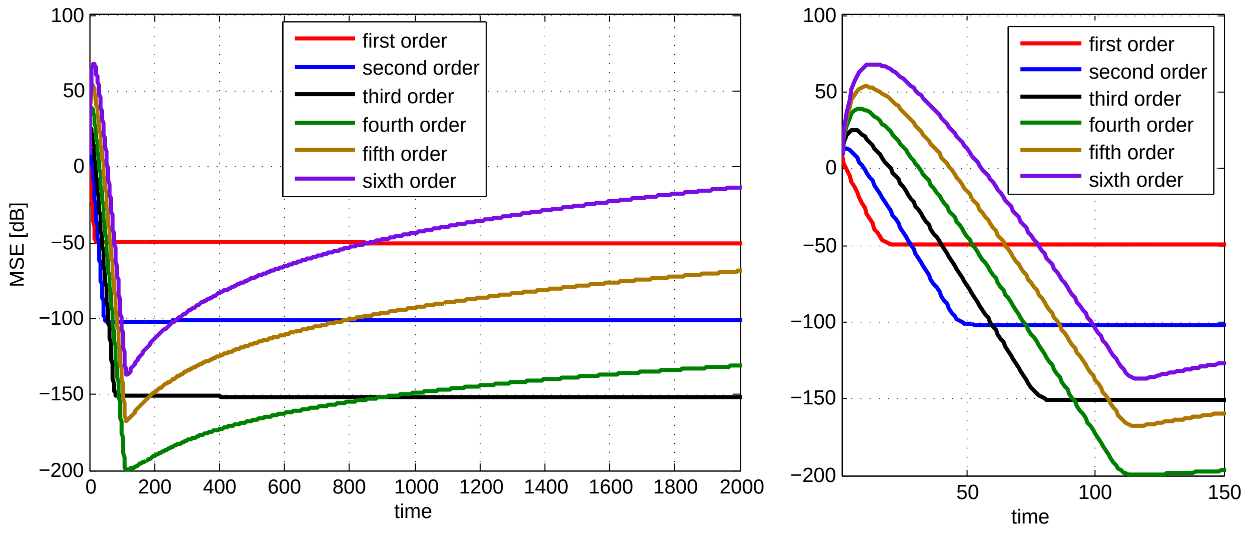


Figure 5.26: *MSE of first-order linear dynamic AC algorithm versus frequency (r=0.38).*



(a) **(b)**

Figure 5.27: *(a) MSE of linear dynamic AC algorithm with CXO versus number of iterations for different orders and (b) its zoomed plot for the first 150 iterations (r=0.38).*

here averaged over 150 different scenarios. It can be seen that for different frequencies, the method with CXO has the lowest MSE and the method with MD the highest. The method with BC has a better performance than MH but both behave worse than the CXO.

5.5.3 Influence of Order of Algorithm on Performance

We want now to consider which performance have algorithms with different orders. We study the same network as last Section. The MSE of linear dynamic AC algorithm with CXO versus number of iterations is shown for different orders in Fig. 5.27. The results in this figure are averaged over 50 different scenarios. It can be seen in Fig. 5.27 (b) that for first-order algorithm, the MSE decreases and after some iterations it will be stabilized. For second- and third-order the MSE increase in the beginning and then start to decrease. Thereafter, it will be stabilized. For algorithms which have orders higher than 3 (fourth-, fifth- and sixth-order in figure), they start in the beginning to increase and after some iterations they decrease until they reach their minimum MSE. Thereafter as shown in Fig. 5.27 (a), they begin to increase again. We considered the same simulation for other methods with MD, MH, and BC too and reached the same results.

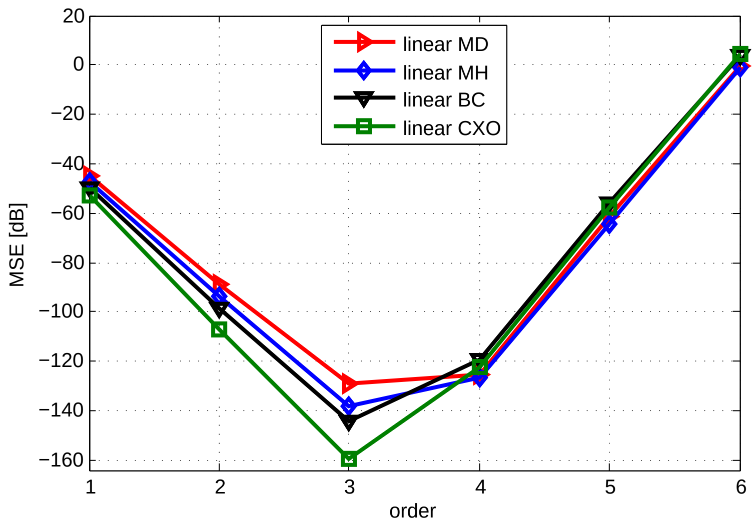


Figure 5.28: *MSE of linear dynamic AC algorithm versus order (r=0.38).*

Now we consider in Fig. 5.28 the MSE of linear dynamic AC algorithm with different weights versus order. The results are here averaged over 150 different scenarios. It can be seen that for all methods, the MSE has for third-order its minimum. We see that for this order the method with CXO has the lowest MSE and the method with MD the highest. The method with BC has a lower MSE than MH but both behave worse than the CXO.

5.6 Nonlinear Dynamic Average Consensus Algorithm

We considered the nonlinear dynamic AC algorithm in Section 4.3. In this Section we determine the parameters of nonlinear function $f(u)$ to achieve the fastest possible convergence speed. Thereafter, we will study the behavior of nonlinear algorithm for node measurements from time-varying spatial field. The performance of nonlinear algorithms for time-varying spatial field with different frequencies and orders will be also discussed.

5.6.1 Determination of Parameters of Nonlinear Function

Similar to Section 5.3.1, we want to determine the parameters θ_1 and θ_2 of nonlinear function $f(u)$ (Equation (3.48)). We consider a network with $N = 100$ nodes in a random geometric graph with radius r=0.38. The nodes measure from a time-varying spatial field constructed with Fourier basis.

MD	MH	BC	CXO
$\theta_1 = 0.36, \theta_2 = 5$	$\theta_1 = 0.85, \theta_2 = 5$	$\theta_1 = 0.2, \theta_2 = 5$	$\theta_1 = 1.0488, \theta_2 = 1.0488$

Table 5.3: *θ_1 and θ_2 for nonlinear dynamic AC algorithm.*

In the beginning we try to find the best possible product $\theta_1\theta_2$ which defines the maximum value of $f(u)$. In Fig. 5.29, we see the MSE versus number of iterations for different weight matrices and different products of parameters for nonlinear dynamic AC algorithm. The results are here averaged over 150 different scenarios. For each colored line, the parameters θ_1 and θ_2 are chosen so that each of them equals to the root of their product value. It can be seen that we have the lowest MSE for different weight matrices with MD, MH, BC, and CXO respectively for line 8 with $\theta_1\theta_2 = 1.8$, for line 7 with $\theta_1\theta_2 = 1.7$, for line 5 with $\theta_1\theta_2 = 1$, and for line 6 with $\theta_1\theta_2 = 1.1$.

Now our next goal is to find the exact values of θ_1 and θ_2 in the determined best possible product of them. We see in Fig. 5.30 the MSE versus number of iterations for different weight matrices and different values of parameters. The results are here averaged over 300 different scenarios. Each colored line denotes different values for θ_1 and θ_2 so that their product equals to the values that we found in Fig. 5.29. To find the best choices, we search for the line which is faster stabilized. The corresponding θ_1 and θ_2 are the best possible choices for their exact values. It can be seen that the fastest stabilized line with different weight matrices with MD, MH, BC, and CXO is respectively the line 3 with $\theta_1 = 0.36$ and $\theta_2 = 5$, the line 4 with $\theta_1 = 0.85$ and $\theta_2 = 2$, the line 3 with $\theta_1 = 0.2$ and $\theta_2 = 5$, and the line 5 with $\theta_1 = 1.0488$ and $\theta_2 = 1.0488$. This values can be found also in Tbl. 5.3.

5.6.2 Measurement from Time-Varying Spatial Field

We consider the same network as mentioned in last Section 5.6.1. It can be seen in Fig. 5.31 the MSE of first-order nonlinear dynamic AC algorithm with parameters calculated in the last section versus number of iterations. It can be seen in Fig. 5.31(b) that in the beginning, the MSE decreases for all methods and after some iterations it stabilizes around a constant value. We see in transient phase that the methods with MD, MH and, BC are for a short time faster than the method with CXO but after some iterations the method with CXO will be faster and it stabilizes around a MSE value which is the lowest in comparison to other methods. The method with CXO has the best tracking behavior and the lowest MSE. The methods with MD and BC have together the highest MSE. Between them is the method with MH.

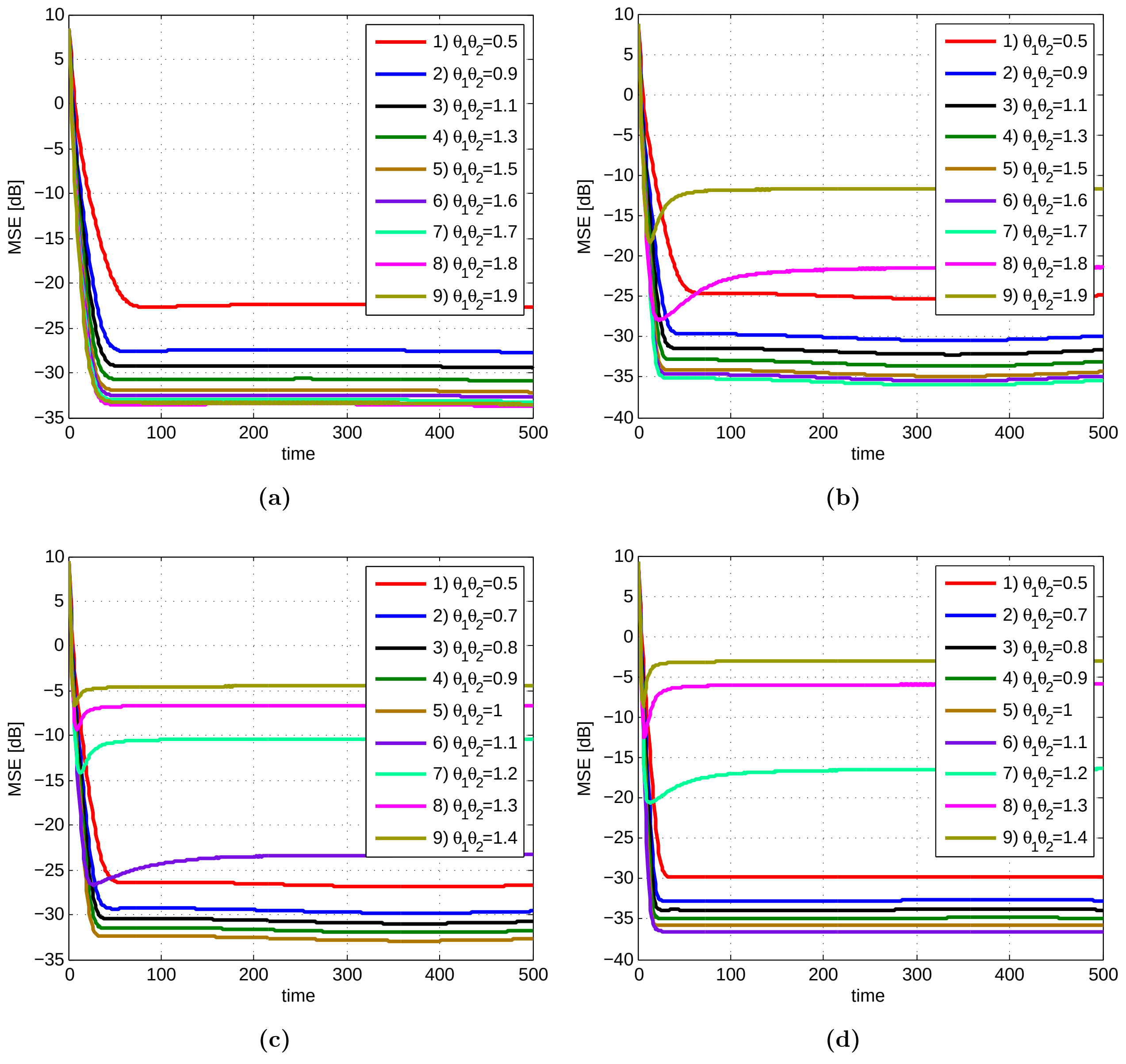
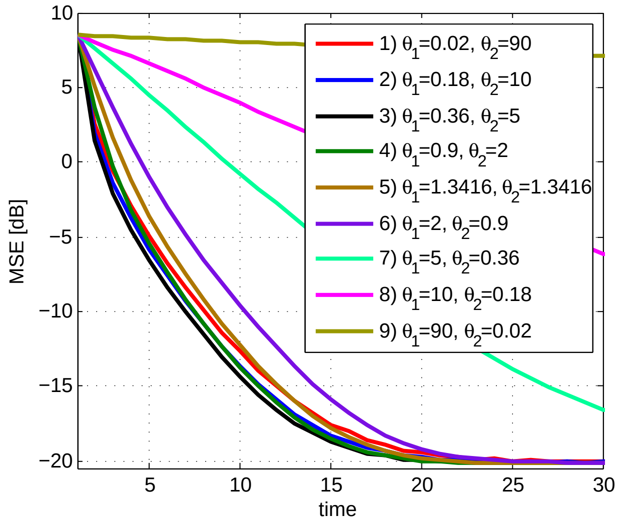
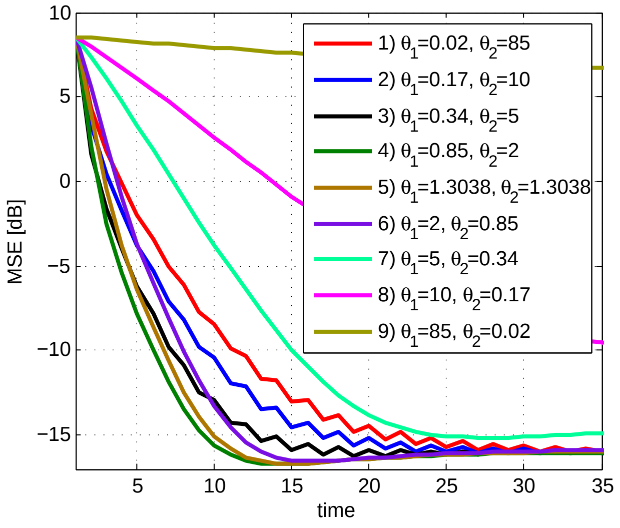


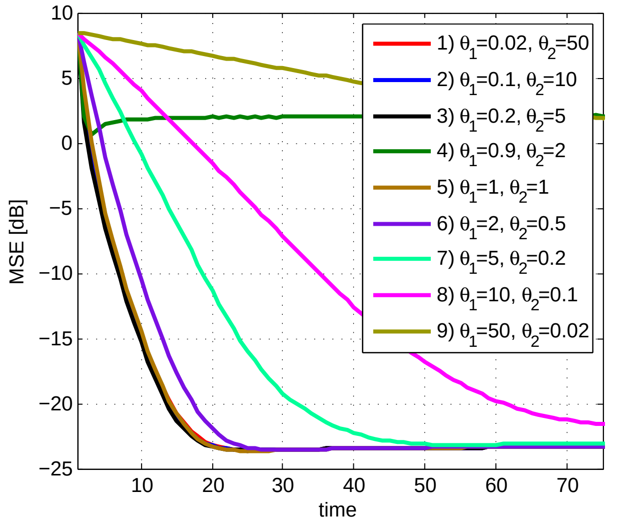
Figure 5.29: *MSE of nonlinear dynamic AC algorithm versus number of iterations to define the best multiply of θ_1 and θ_2 with (a) MD, (b) MH, (c) BC, and (d) CXO (r=0.38).*



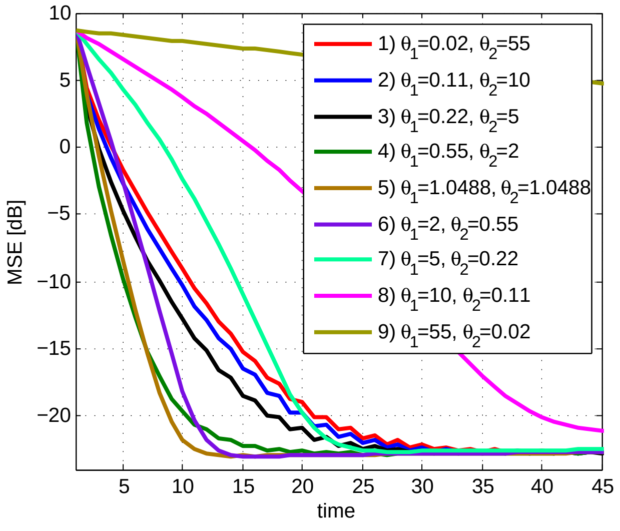
(a)



(b)

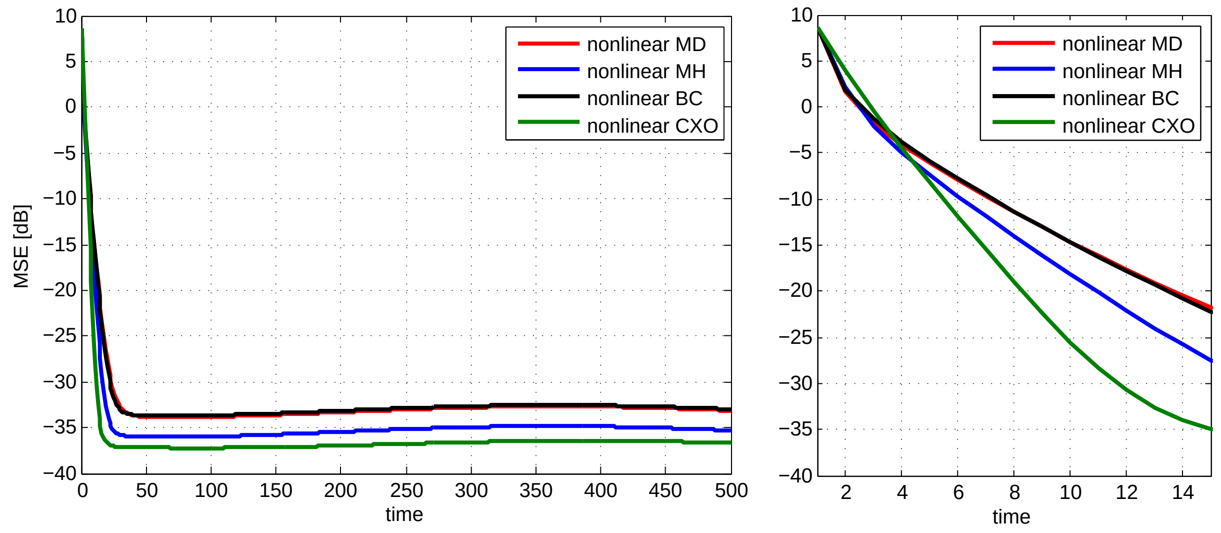


(c)



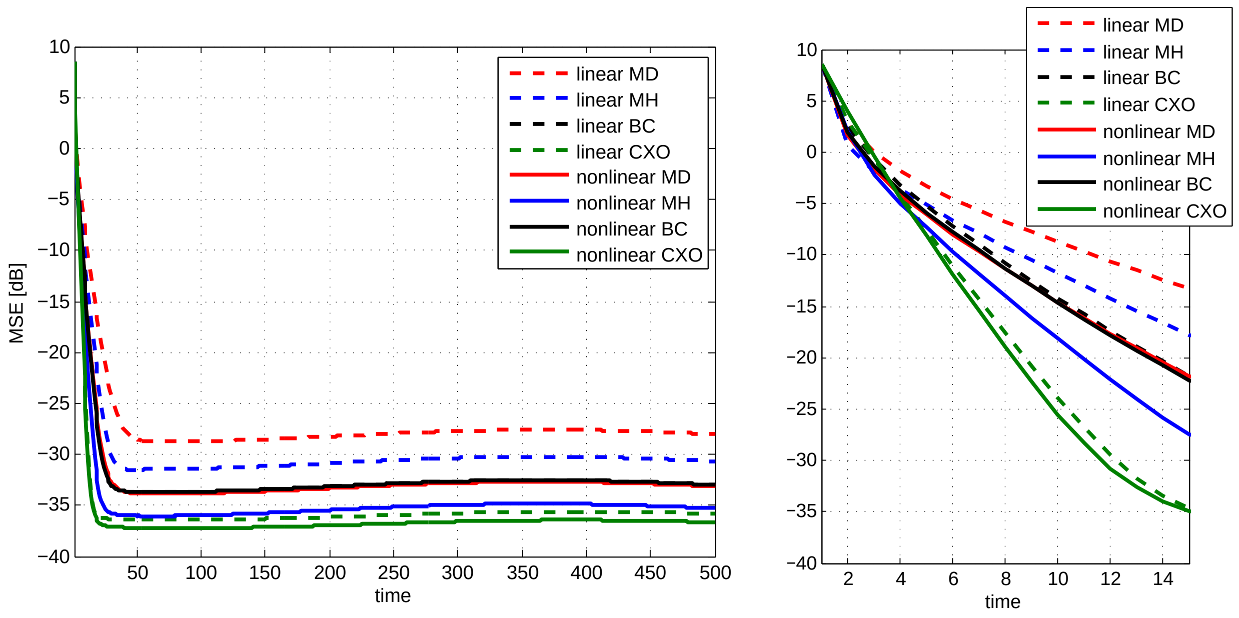
(d)

Figure 5.30: *MSE of nonlinear dynamic AC algorithm versus number of iterations to define θ_1 and θ_2 for (a) MD, (b) MH, (c) BC, and (d) CXO (r=0.38).*



(a) **(b)**

Figure 5.31: *(a) MSE of first-order nonlinear dynamic AC algorithm versus number of iterations with* $r = 0.38$ *and (b) its zoomed plot for the first 15 iterations.*



(a) **(b)**

Figure 5.32: *(a) MSE of first-order linear and nonlinear dynamic AC algorithms versus number of iterations with* $r = 0.38$ *and (b) its zoomed plot for the first 15 iterations.*

In Fig. 5.32, we see the MSE of all first-order linear and nonlinear dynamic AC algorithms with different weight matrices versus number of iterations. It can be seen that for nonlinear algorithms with MD, MH, and CXO, we achieve a lower MSE than linear algorithms. Just with BC we have similar results for both linear and nonlinear algorithms. In transient phase is for a short time the linear algorithm with MH the fastest method but after some iterations the nonlinear algorithm with CXO will be faster than other methods and it has the best tracking behavior and the lowest MSE. Therefore, we achieve the best results for most of the iterations with nonlinear algorithm with CXO.

5.6.3 Influence of Frequency of Time-Varying Spatial Field on Performance

We know that the nodes measure from a spatial field constructed with Fourier basis so that Fourier coefficients change with different sinus signals with different amplitudes, phases but a defined equal frequency. Now we want to consider the influence of different frequencies on performance of nonlinear algorithm.

We see in Fig. 5.33 the MSE versus number of iterations for different frequencies. The results are here aveaged over 50 different scenarios. It can be seen that increase of frequency, causes a higher MSE and also higher fluctuation of it.

It can be seen in Fig. 5.34 the MSE of nonlinear algorithm for different frequencies. To have a higher precision we increase the number of scenarios and average over 150 different scenarios. It can be seen that with increase of frequency, the MSE of each method will be stabilized around a higher MSE value. For all of this frequencies, we have the lowest MSE with CXO and the highest with MD and BC. Between them is the method with MH.

The MSE of all linear and nonlinear algorithms for different weight matrices and frequencies, can be seen in Fig. 5.35. It can be seen that with increase of frequency, the MSE of each weight matrix increases also but they do not change in comparison with each other, it means that for example the nonlinear method with CXO has for each frequency the lowest MSE and this remains unchanged over all frequencies.

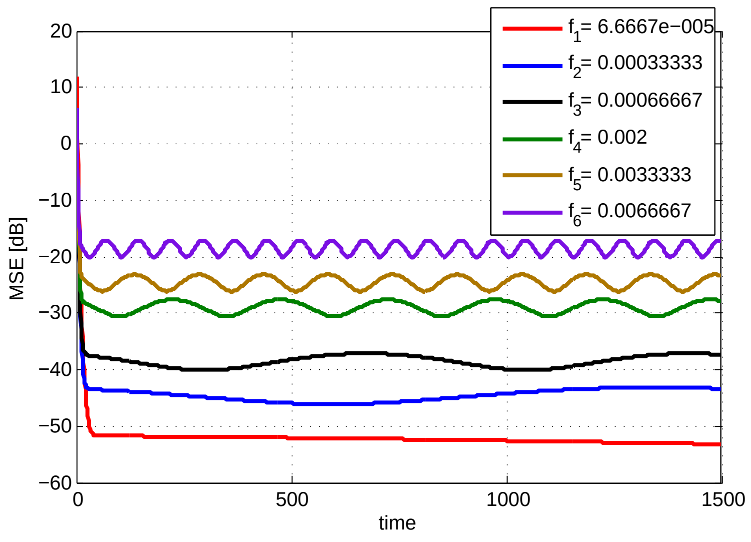


Figure 5.33: *MSE of first-order nonlinear dynamic AC algorithm with CXO versus number of iterations for different frequencies (r=0.38).*

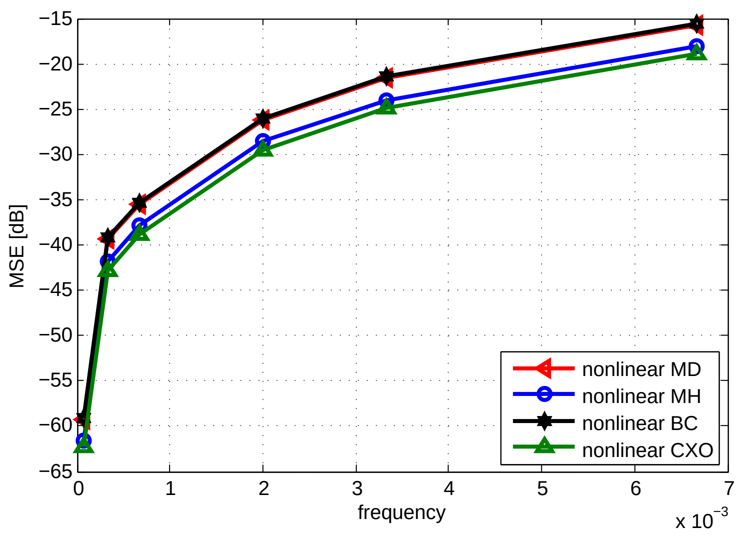
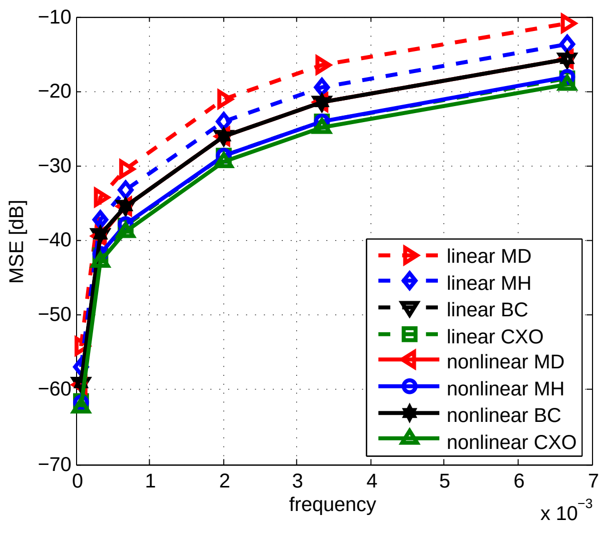
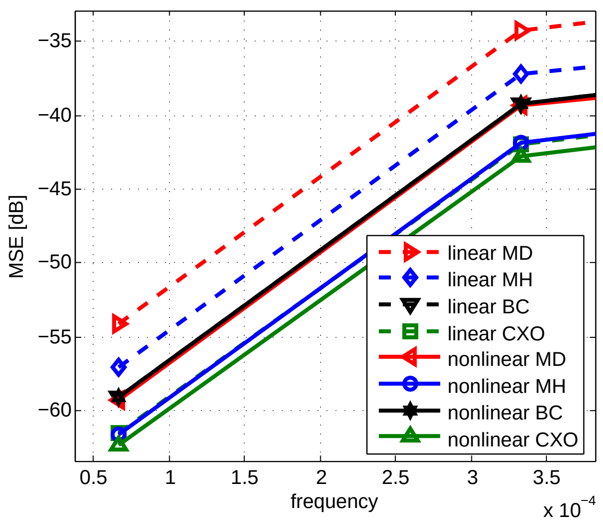


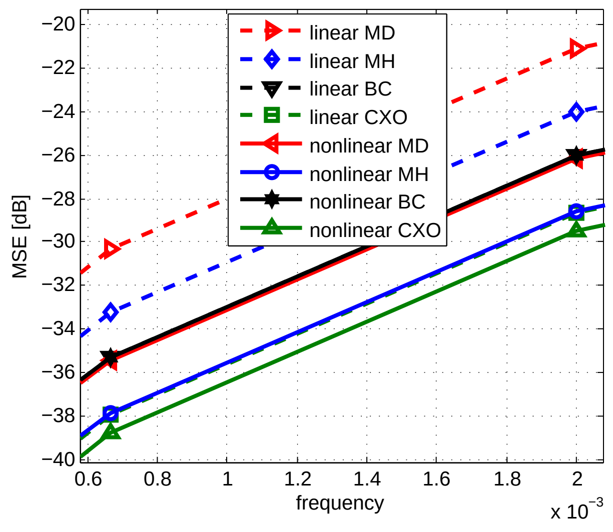
Figure 5.34: *MSE of first-order nonlinear dynamic AC algorithm versus frequency (r=0.38).*



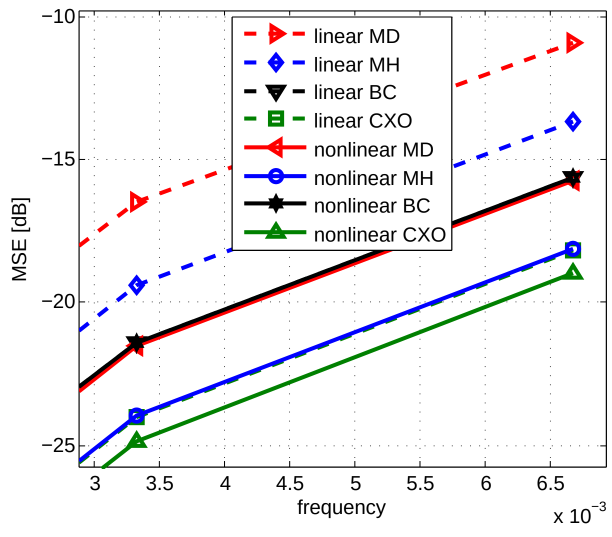
(a)



(b)



(c)



(d)

Figure 5.35: *(a) MSE of first-order dynamic AC algorithm versus frequency and its zoom for (b) f_1 and f_2, (c) f_3 and f_4, and (d) f_5 and f_6 (r=0.38).*

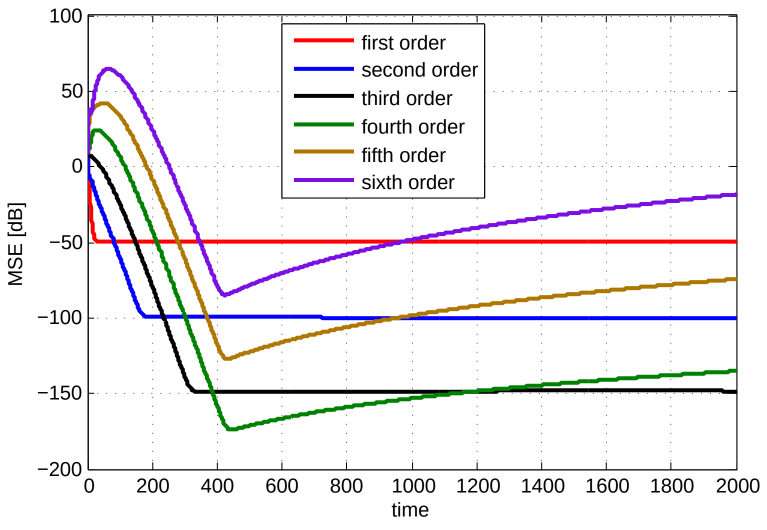
5.6.4 Influence of Order of Algorithm on Performance

In last two sections, we considered the first-order nonlinear dynamic AC algorithms, now we want to study the performance of higher-order algorithms.

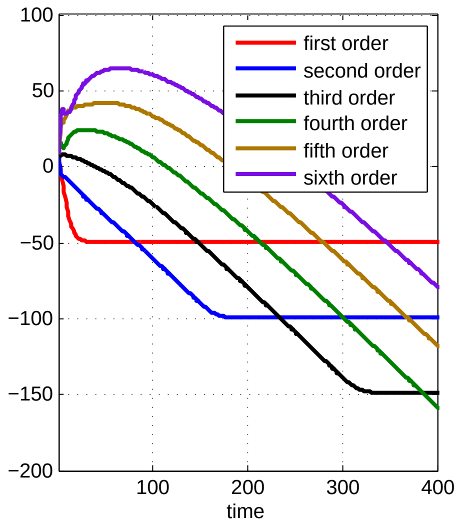
We see in Fig. 5.36 the MSE of nonlinear dynamic AC algorithm with CXO versus number of iterations for different orders. The results are here averaged over 50 different scenarios. It can be seen that for first- and second-order, the MSE deceases until it reaches a minimum value and then will be stabilized around this value. For third-order, the MSE increases in the beginning but after some iterations it starts to decrease until it reaches its minimum and then will be stabilized around this minimum. For fourth-, fifth- and sixth-order, the MSE increases also in the beginning and after some iterations, it starts to decrease until a minimum is reached. Thereafter, in converse to other algorithms with lower orders, the MSE begins again to increase. It can be seen that the nonlinear algorithm with CXO has the lowest MSE over different iterations.

We see in Fig. 5.37 the MSE of nonlinear dynamic AC algorithm versus order. The results are here averaged over 150 different scenarios. It can be seen that for all nonlinear algorithms with different weight matrices, the lowest MSE is for third-order. For this order, the method with CXO has the lowest MSE and the methods with MD and BC the highest. Between them is the method with MH.

In Fig. 5.38, we see a comparison of MSE of all linear and nonlinear algorithms with different weight matrices and orders. It can be seen as mentioned before, that for linear and nonlinear algorithms, the lowest MSE can be achieved for third-order and for the nonlinear algorithm with CXO.



(a)



(b)

Figure 5.36: *(a) MSE of nonlinear dynamic AC algorithm with CXO versus number of iterations for different orders and (b) its zoomed plot for the first 450 iterations (r=0.38).*

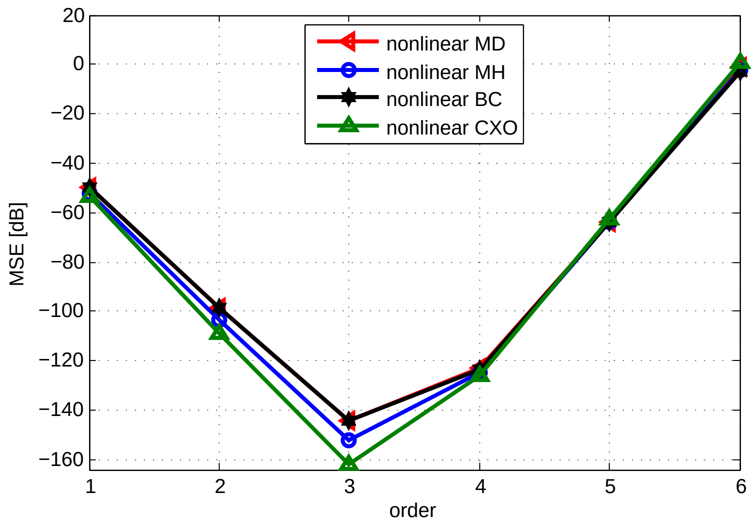
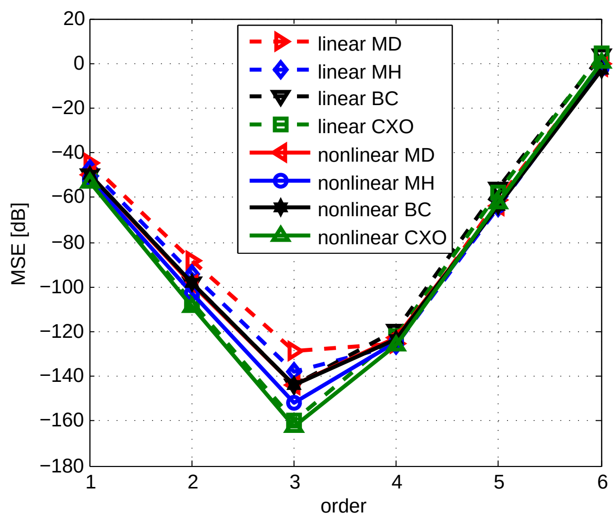
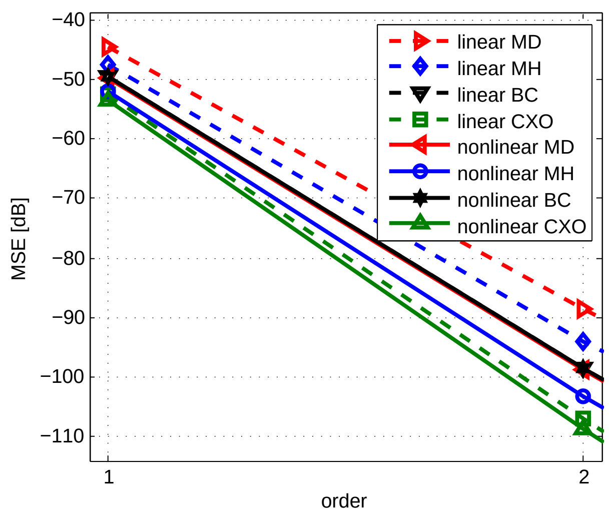


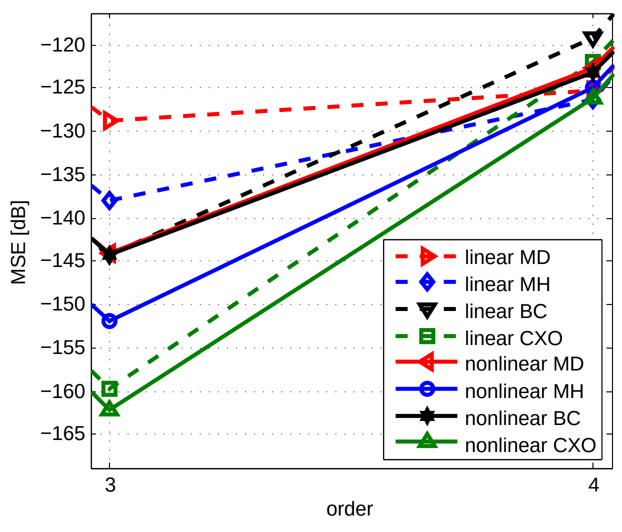
Figure 5.37: *MSE of nonlinear dynamic AC algorithm versus order(r=0.38).*



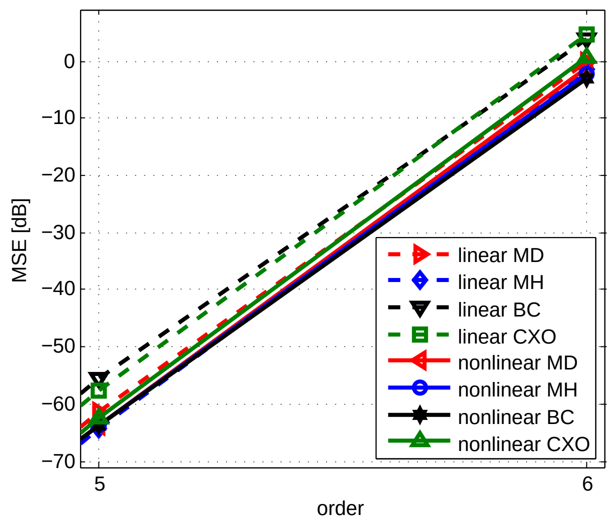
(a)



(b)



(c)



(d)

Figure 5.38: *(a) MSE of dynamic AC algorithm versus order and its zoom for (b) first- and second-, (c) third- and forth-, and (d) fifth- and sixth-order (r=0.38).*

6 Summary

In this thesis, we considered different wireless sensor networks (WSN). They consist of low-cost sensors which are distributed over an area. This sensors accomplish simple computations and local measurements and they can also communicate over wireless links with their neighbors and exchange different informations. To describe WSN mathematically, we model them as graphs. Each node represents a sensor and each edge the communication link between the sensors. We discussed a distributed averaging method to estimate a global quantity which is the average value. This distributed averaging algorithm is called as average consensus (AC) algorithm. For AC algorithm we discussed also different AC weight design methods with maximum degree (MD), Metropolis-Hasting (MH), best constant (BC), and convex optimization (CXO) weights.

In case of sensors which measure constant values so that this values do not change over different iterations, we used static AC algorithm. We considered the linear static AC algorithm and the nonlinear version of it and gave each node a state value which is initially equal to measured value and converge asymptotically to average value.

For linear static AC algorithm, our numerical results suggest that for the most of scenarios that we discussed, MH weights have a faster convergence in transient phase and CXO weights have the best performance in asymptotic phase. The performance of algorithms is dependent on network structure as like number of nodes and edges. It is also dependent from the measured values of sensors. For example if the sensors measure from a spatial field, the method with MH weights has not more in transient phase the best performance.

For nonlinear static AC algorithms we used a nonlinear function which modulates the elements of weight matrix in dependence of node states. It causes an improvement of transient phase of algorithms. The results suggest for our defined scenarios that the fastest convergence in transient phase is with MH weights and in asymptotic phase with CXO weights. We came through comparison of linear and nonlinear algorithms to conclusion that the nonlinear algorithms are superior to linear ones.

We used also the advantages of MH and CXO weights to design a new weight matrix which is a combination of MH and CXO weights. It leads to an algorithm which has the best performance in transient and also asymptotic phase. We call this algorithm as nonlinear static AC algorithm with combination of MH and CXO weights.

In case of sensors which measure time-varying signals, we used dynamic AC algorithm. We considered the linear dynamic AC algorithm and the nonlinear version of it. It is concluded from results that in our defined scenarios, we have with nonlinear algorithm with CXO weights the best tracking behavior. We considered also dynamic AC algorithms with higher-order differences. The third-order dynamic AC algorithm has in our simulations the best performance in tracking phase. We discussed also that an increase of frequency of time-varying signals, causes a worse performance.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows - theory, algorithms and applications.* Prentice Hall, 1993.

[2] V. Turau, *Algorithmische Graphentheorie (2. Aufl.).* Oldenbourg, 2004.

[3] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, pp. 215 –233, jan. 2007.

[4] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 5, pp. 4997 – 5002 Vol.5, dec. 2003.

[5] L. Georgopoulos and M. Hasler, "Nonlinear average consensus," in *Proceedings of the 2009 International Symposium on Nonlinear Theory and its Applications*, (Sapporo, Hokaido), pp. 10–13, IEICE, 2009.

[6] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21." http://cvxr.com/cvx, Apr. 2011.

[7] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control* (V. Blondel, S. Boyd, and H. Kimura, eds.), Lecture Notes in Control and Information Sciences, pp. 95–110, Springer-Verlag Limited, 2008.

[8] M. Zhu and S. Martínez, "Discrete-time dynamic average consensus," *Automatica*, vol. 46, no. 2, pp. 322–329, 2009.

Notation

Throughout this thesis, vectors are denoted by boldface letters. Time dependent quantities are denoted by $x[k]$ in the discrete time. Unless noted otherwise, the meaning of the following symbols is as stated below.

Symbol	*Meaning*
(i,j)	Directed edge
$\{i,j\}$	Undirected edge
$\mathbf{A}$	Adjacency matrix
a_{ij}	Element of adjacency matrix
$\mathbf{B}$	Incidence matrix
b_{il}	Element of incidence matrix
$\mathbf{D}$	Degree matrix
d	Number of connected node neighbors in random regular graph
d_i	Degree
$d_{i,in}$	Indegree
$d_{i,out}$	Outdegree
$\mathcal{E}$	Edges set
$f(u)$	Nonlinear function
$g(v)$	Nonlinear function
$\mathcal{G}$	Graph
$\mathbf{I}$	Identity matrix
i	Node i
$\mathbf{J}$	Unit matrix

M	Number of edges
N	Number of nodes
$\mathcal{N}_i$	Neighbor set
$\mathbf{L}$	Laplacian matrix
l_{ij}	Element of Laplacian matrix
$r_{\text{asymptotic}}$	Asymptotic convergence factor
r_{step}	Per-step convergence factor
$\overline{s}$	Average value of measurements
$\overline{\mathbf{s}}$	Average vector
$\overline{s}\,[k]$	Average signal at time k
$\overline{\mathbf{s}}\,[k]$	Average signal vector
$u_{ij}\,[k]$	Node states difference at time k
$\mathcal{V}$	Nodes set
$v_{ij}\,[k]$	Relative node states difference at time k
$\mathbf{W}$	Weight matrix
W_{ij}	Element of weight matrix
$x_i\,[k]$	State of node i at time k
$x_i^{[l]}\,[k]$	lth-order state of node i at time k
$\mathbf{x}\,[k]$	Node states vector at time k
$\mathbf{x}^{[l]}[k]$	lth-order node states vector at time k
Δ	Maximum degree of graph
$\Delta\mathbf{s}[k]$	State differences
$\Delta^{[n]}\mathbf{s}[k]$	nth-order state differences
λ_i	ith largest eigenvalue
$\rho\,(\cdot)$	Spectral radius
$\tau_{\text{asymptotic}}$	Asymptotic convergence time
τ_{step}	Per-step convergence time

List of Abbreviations

AC	**A**verage **C**onsensus
BC	**B**est **C**onstant
CXO	**C**onve**X** **O**ptimization
MD	**M**aximum **D**egree
MH	**M**etropolis **H**asting
MSE	**M**ean **S**quare **E**rror
WSN	**W**ireless **S**ensor **N**etworks