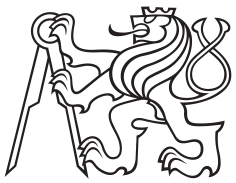
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Distributed signal processing in radio communication networks

Jakub Kolář
Open Electronic Systems

May 2017
kolarj39@fel.cvut.cz
Supervisor: prof. Ing. Jan Sýkora, CSc.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Radioelectronics

BACHELOR PROJECT ASSIGNMENT

Student: **Jakub Kolář**

Study programme: Open Electronic Systems

Title of Bachelor Project: **Distributed Signal Processing in Radio Communication Networks**

Guidelines:

Student will get acquainted with fundamental principles of the algorithms on graphs with a special focus on distributed signal processing algorithms for radio communication networks. Main focus of the work should be on average consensus algorithm on the graph. Student should apply this algorithm on selected simple scenarios in communication networks, e.g. distributed time or carrier synchronisation. The algorithms should be implemented in Matlab including suitable graphical demonstration of the distributed convergence process and algorithms performance.

Bibliography/Sources:

[1] Lin Xiao, Stephen Boyd, Seung-Jean Kim: Distributed average consensus with least-mean-square deviation. Journal of Parallel and Distributed Computing, 2007, Volume 67 Number 1
[2] U.Spagnolini: Distributed signal processing and synchronization, tutorial 2013

Bachelor Project Supervisor: prof. Jan Sýkora Ing., CSc.

Valid until the end of the summer semester of academic year 2017/2018

L.S.

doc. Mgr. Petr Páta, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 27, 2017

Acknowledgement / Declaration

TBD. In Prague, 26. 5. 2017

.....

Abstrakt / Abstract

Cílem této bakalářské práce je seznámení se s lineárním konsenzuálním algoritmem nad grafy. Jde o iterativní algoritmus, který pracuje s hodnotami přiřazenými vrcholům grafu a jehož cílem je, aby asymptoticky byly tyto hodnoty ve všech vrcholech stejné a průměrem hodnot počátečních. Studovaný algoritmus toho docílí jen komunikací mezi dvojicemi vrcholů přes existující hrany grafu. V práci využíváme maticovou reprezentaci grafu a pro nalezení vhodných parametrů algoritmu používáme Laplaceovu matici. Algoritmus je tak převeden na opakující se maticové násobení, jehož význam zachovává původní úlohu. Dále uvádíme názorné ukázky toho, jak algoritmus pracuje v případě ideálních aktualizací a základní řešení v případě ovlivňování aktualizací šumem s nulovou střední hodnotou. Součástí práce je implementace algoritmu v Matlabu. V práci nejsou uvedeny složitější důkazy a odvození, ale odkazujeme se na příslušnou uvedenou literaturu. Na závěr uvádíme několik jednoduchých příkladů využívajících popsané iterativní schéma.

Klíčová slova: Graf, Laplaceova matice, Lineární konsenzus, Aktualizace s rušením, Odhad parametru

Main scope of this Bachelor's thesis is to provide an introduction to Linear average consensus algorithm over graph. It is an iterative algorithm, that works with values assigned to vertices of graph and the goal is to asymptotically obtain in all vertices an average of initial values, respecting the topology of the graph. In the text we use a matrix representation of graph and to design the parameters of algorithm is used Laplacian matrix. Using matrix representation, the algorithm that solves the original problem is transferred to iterative matrix multiplication. Next, we provide examples of run of the algorithm in case of ideal and reliable communication and also an outline of solution in case of zero-mean noisy updates. Included are also Matlab scripts with algorithm implementation. More difficult formal proofs and derivations are not included, but we provide references, where to find them in literature. Finally, we provide few simple examples of application of the described algorithm.

Keywords: Graph, Laplacian, Linear average consensus algorithm, Noisy updates, Estimation

Contents /

/ Figures

Chapter 1
Introduction

Topic of distributed algorithms used in radio communication has been for last decades subject of research. Throughout this Thesis we study and comment the basic results found in the corresponding literature, with emphasis on the understanding of Average consensus algorithm.

1.1 Motivation

To begin with an idea of an average consensus algorithm, let's make a thought experiment. We are looking for an average quantity, for example an average temperature in a room, with a group of wireless communication devices, that can exchange information, provided they are in range to reach each other. We deploy these thermometers in the room randomly, with no special requirements on a topology. Next, let's consider, that for each pair of the thermometers we can decide, whether they can exchange information or not - meaning we know all neighbors of all devices, that are mutually in range to communicate.

Now, we can encode our experiment settings to a graph. A very natural way to represent this graph is drawing it. To do so, we simply take all thermometers as different vertices. Of course, every vertex always knows a result of its own measurement. By an edge between two vertices we mark the situation, that these two nodes can exchange information. Which means, every node can get know also the value that measures its neighbor. This ought to be only a very simple illustration how to transfer a physically realizable experiment to the terms of graphs.

Finally, as we shall see, if we fulfill some basic convergence conditions on the properties of this graph, the average consensus algorithm acts like this: We synchronously update the value in each node by some increment, which depends only on the old value in this node and the values of its direct neighbors in the graph. By doing this long enough, we obtain in each node a value, which goes in limit to the average of all initially measured values.

1.2 Outline

A Graph theory provides a very elegant way to represent information encoded by graphs as matrices. In the first chapter we will provide some basic definitions to the Graph theory and properties of these important matrices. Using matrices, we will also briefly mention some very useful results from Matrix analysis, because also a serious object of our interest will be topic of eigenvalues of matrices. We will define a Laplacian of a graph and show some of its basic properties. In next chapter, we will provide a description of the average consensus algorithm and show some examples with graphically illustrated solution. In the last part of this thesis, we will try to implement this algorithm, to easily solve some typical problems in area of wireless digital communication - such as time synchronization. In a very simple case, we can also show, how do the nonidealities change the result of algorithm, in our case additive zero mean noise.

Chapter 2
Graph theory

2.1 Motivation

It is commonly known, that the elements of Graph theory were set by a mathematician Leonhard Euler in the 18th century , when He solved a problem called Seven Bridges of Königsberg [1].

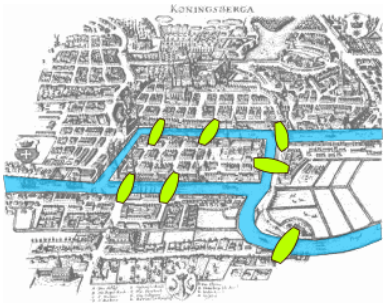


Figure 2.1. View on city Konigsberg with marked bridges [2] .

The problem was formulated like this: River Pregole flows through the city and creates two islands in there; these two islands were connected with the city by 7 bridges (see figure above) and the question was, whether it is possible to take a walk through the city in such a way, to pass each bridge exactly once [1]. Euler described this problem as a graph, where the edges represented the seven bridges, and the vertices were the separated parts of the city. He proved, that in this case, it is impossible to pass each bridge only once and showed, that such a way exists if and only if each vertex of the graph is of an even degree [1].

Nowadays, a very modern and interesting example of usage of the Graph theory is visualization and simulation of the communication networks, such as the Internet, mobile network etc. A typical task is to find the best route from a source to a destination location with respect to a given specific metric. This metric can depend on many parameters. Typical choices are e.g. a number of intermediate devices, end-to-end delay or a bandwidth of the connectivity. These Graph algorithms are often based on a very efficient improvement of basic Depth-first search. (This is a case of the famous Dijkstra's algorithm used by OSPF routing protocol, to find the best way from each node to all the others with edges with a given cost.)

Next interest, we will briefly meet also term Spanning tree. In Ethernet-based communication is very important to avoid loops in a network, because they might cause a congestion of the network and failure of the service. A Spanning tree of a graph is a factor of this graph, which originates by removing some of its edges, in a way to preserve all vertices reachable, and removes all cycles in the graph [3]. In the chapter about Laplacian we shall see how to simply find a count of these Spanning trees.

To finish this motivation part, a very nice example of the usage of distributed algorithm is a Network time protocol (NTP). It is important to have globally synchronized time between server computers. Few of the servers are connected to the reference clocks and next, to them hierarchically connected servers, are stochasticaly analyzing properties of neighbor's time to obtain final value of time, that they use and provide to next users [4].

2.2 Notation

In the sequel we will use the following notation. Scalars are denoted by small leters, such as a, d, α. Vectors are marked as e.g. $\boldsymbol{u}$ or $\boldsymbol{v}$. We use column vectors. Especically $\boldsymbol{1}$ is a column vector of all ones (in the text always of length N as number of nodes). Matrices are marked by capitals, such as $\boldsymbol{L}, \boldsymbol{P}$. $\boldsymbol{I}$ stands for Identity matrix. Single elements of vectors and matrices are denoted as scalars with lower indices with meaning of the position, e.g. v_i or p_{ij}, respectively. Finally in last chapter we use e.g. $\boldsymbol{x_i}$ to mark vector that is in i-th node.

2.3 Definitions

A graph $G = (V, E)$ is described by a pair of its vertices V and edges E. $V = \{v_1, v_2, ..., v_N\}$ is a set of N vertices. By the vertices we understand the points connected by the edges. An edge (v_i, v_j) means a connection between vertices v_i and v_j.

2.3.1 Undirected graph

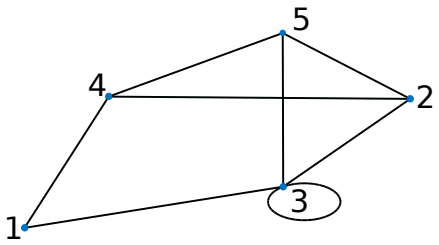


Figure 2.2. Example of a simple undirected graph to demonstrate basic definitions.

The graph $G = (V, E)$ above could be described by set of vertices $V = \{1; 2; 3; 4; 5\}$ and set of edges $E = \{(1, 4); (1, 3); (2, 4); (2, 5); (3, 2); (3, 3); (3, 5); (4, 5)\}$.

Set of neighbors $\mathcal{N}_i$ of a vertex v_i is $\mathcal{N}_i = \{v_j \in V | (v_i, v_j) \in E\}$. For example $\mathcal{N}_4 = \{1; 2; 5\}$. Degree of a node is $d_i = |\mathcal{N}_i|$.

2.3.2 Directed graph

For directed graph holds the same as for undirected with only difference, we distinguish the edges (v_i, v_j) and (v_j, v_i). Then, for a degree of a node in directed graph, we have to consider only neighbors available via oriented edges, $d_i^{IN} = |\mathcal{N}_i|$. Drawing the figure, we distinguish the orientation of edges with arrows.

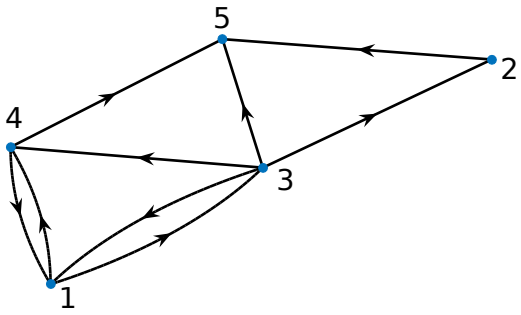


Figure 2.3. Example of a directed graph.

2.3.3 Adjacency matrix

Adjacency matrix is a very natural way of a full graph description. This matrix is $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ and for graph G with N vertices describes inner connectivity of the graph with information, to what all vertices goes an edge from a given vertex. Its values $a_{i,j}$ are defined as [5]:

$$a_{i,j} = \begin{cases} 1 & \text{if there is the edge } (v_i, v_j), \\ 0 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Adjacency matrix of a graph from figure 2.2 reads

$$\boldsymbol{A}_{2.2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \tag{2.2}$$

We can see, that Adjacency matrix of an undirected graph is symmetric.

And adjacency matrix of a graph from figure 2.3 is

$$\boldsymbol{A}_{2.3} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.3}$$

For directed graph the Adjacency matrix generally is not symmetric.

For undirected graphs allowing *Weighted graphs* means, that for each pair of vertices i, j we assign a certain weight $a_{i,j}$, that satisfies conditions: 1) $a_{i,j} = a_{j,i}$, 2) $a_{i,j} \geq 0$ and 3) $a_{ij} \neq 0$ if and only if vertices i and j are not connected by an edge. This is only a generalization of the Adjacency matrix definition above.

2.3.4 Degree matrix

A Degree matrix $\boldsymbol{D} \in \mathbb{R}^{N\times N}$ is a diagonal matrix bearing an information about degree of each vertex. Its diagonal elements are $d_i = \sum_{i\neq j} a_{i,j}$ and all non diagonal elements are equal to 0 [6]. For example Degree matrix of undirected graph from figure 2.2 reads $\boldsymbol{D}_{2.2} = diag\{2, 3, 4, 3, 3\}$. Next, for the case of directed graph we have to consider only incoming edges. Which for graph from figure 2.3 means $\boldsymbol{D}_{2.3} = diag\{2, 1, 1, 2, 3\}$. And also let's define $\Delta = \max_i(d_i)$.

2.3.5 Incidence matrix

An Incidence matrix of a directed graph provides for each edge an information about an initial and terminal vertex. For a graph with N vertices and L edges the Incidence matrix $\boldsymbol{E} \in \mathbb{R}^{N \times L}$ elements $e_{i,j}$ are defined as [7]:

$$e_{i,j} = \begin{cases} 1 & \text{if edge } j \text{ begins in the vertex } i, \\ -1 & \text{if edge } j \text{ ends in the vertex } i\ , \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

So Incidence matrix for graph on figure 2.3 reads

$$\boldsymbol{E}_{2.3} = \begin{pmatrix} -1 & -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{2.5}$$

We can see, this matrix is very rare. In each column contains only one pair of 1 and -1. The adjacency matrix provides same information but with typically smaller matrix.

2.4 Laplacian matrix

The structure of following section about Laplacian is based mainly on [8]. Supplementary references are added at corresponding paragraphs.

2.4.1 Definitions

Now, in previous text we defined Adjacency and Degree matrix of a graph G. Next, we define the *Laplacian matrix* $\boldsymbol{L}(G)$ of a graph [8],

$$\boldsymbol{L}(G) = \boldsymbol{D}(G) - \boldsymbol{A}(G). \tag{2.6}$$

Matrix $\boldsymbol{L}(G)$ for a graph with N vertices is $\boldsymbol{L}(G) \in \mathbb{R}^{N \times N}$. Loops do not affect $\boldsymbol{L}(G)$. To make hold some important results from Linear algebra and Matrix analysis, we will next consider only undirected graphs without loops. Which means, that the corresponding Adjacency matrix will be symmetric. Because the $\boldsymbol{A}(G)$ is symmetric, the $\boldsymbol{L}(G)$ comes to be also symmetric.

Using the Incidence matrix of graph $\boldsymbol{E}(G)$, we can find the Laplacian matrix of graph G as

$$\boldsymbol{L}(G) = \boldsymbol{E}(G)\boldsymbol{E}^T(G). \tag{2.7}$$

The Laplacian definitions in Equations and are equivalent.

Let's denote $\mu(G, x)$ the characteristic polynomial of $\boldsymbol{L}(G)$, i.e.

$$\mu(G, x) = \det(\boldsymbol{L} - x\boldsymbol{I}). \tag{2.8}$$

Roots of this characteristic polynomial are called *Laplacian eigenvalues* of G. As it is common in literature, we will denote them $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$, enumerated with lower indices in an increasing order with counting multiplicities. N denotes the number of vertices. The set $\{\lambda_1, \lambda_2, ..., \lambda_N\}$ is called the *spectrum* of $\boldsymbol{L}(G)$ and is in interest of *spectral graph theory* [8].

2.4.2 Basic properties

Theorem 2.1. If $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is symmetric then $\boldsymbol{A}$ has real eigenvalues.

Proof: Ommited. For example [9], page number 92.

Theorem 2.2. Let G be an undirected graph without loops. Then 0 is an eigenvalue for the Laplacian matrix of G with an eigenvector $(1, 1, ..., 1)^T$.

Proof: Found in [10]. If G is an undirected graph then the sum of the entries in row i of Adjacency matrix $\boldsymbol{A}$ gives exactly the degree d_i of vertex i. So we can write:

$$\boldsymbol{A}\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} = \begin{pmatrix}d_1\\d_2\\\vdots\\d_N\end{pmatrix}. \tag{2.9}$$

And from that:

$$\boldsymbol{L}(G)\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} = (\boldsymbol{D}(G) - \boldsymbol{A}(G))\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} = \begin{pmatrix}d_1 - d_1\\d_2 - d_2\\\vdots\\d_N - d_N\end{pmatrix} = \begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix} = 0\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix}. \tag{2.10}$$

In which we easily recognize the relation holding for eigenvalues.

Theorem 2.3. The Laplacian matrix $\boldsymbol{L}(G)$ is positive semi-definite and singular [11].

Proof: Let λ be an eigenvalue and $\boldsymbol{v}$ its corresponding eigenvector. Then

$$\boldsymbol{L}\boldsymbol{v} = \lambda\boldsymbol{v}, \tag{2.11}$$

$$\lambda = \boldsymbol{v}^T\boldsymbol{L}\boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{E}\boldsymbol{E}^T\boldsymbol{v} = (\boldsymbol{v}^T\boldsymbol{E})(\boldsymbol{E}^T\boldsymbol{v}) = (\boldsymbol{E}^T\boldsymbol{v})^T(\boldsymbol{E}^T\boldsymbol{v}) = \|\boldsymbol{E}^T\boldsymbol{v}\|^2 \geq 0. \tag{2.12}$$

$\boldsymbol{L}$ is singular, because sum of all elements in each column is zero.

2.4.3 Bounds for eigenvalues

Theorem 2.4. *Gershgorin circle theorem* [9]. Consider matrix $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ and $i = 1, 2, ..., N$. Let's denote

$$r_i = \sum_{j=1;\ i\neq j}^{N} = |a_{ij}|, \quad K_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \leq r_i\}. \tag{2.13}$$

The K_i sets are called *Gershgorin circles*. It holds for all eigenvalues $\{\lambda_1, \lambda_2, ..., \lambda_N\}$ of the matrix $\boldsymbol{A}$, that they are all localized in the union of *Gershgorin circles* $\{K_1 \cup K_2 \cup ... \cup K_N\}$ in the Complex plane.

Proof: Let λ be an eigenvalue of $\boldsymbol{A}$ and its corresponding eigenvector $\boldsymbol{x} = (x_1, x_2, ..., x_N)$. So holds $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. Let x_k be the biggest absolute value number in vector $\boldsymbol{x}$. Then $\lambda x_k = \sum_{j=1}^{N} a_{kj}x_j$. Next move the $a_{kk}x_k$ summand from RHS to LHS. We obtain $x_k(\lambda - a_{kk}) = \sum_{j=1;j\neq k}^{N} a_{kj}x_j$. Now we take an absolute value of this equation, divide by x_k and using Triangle inequality we go to:

$$|\lambda - a_{kk}| = \left|\frac{\sum_{j=1;j\neq k}^{N} a_{kj}x_j}{x_k}\right| \leq \sum_{j=1;j\neq k}^{N}\left|\frac{a_{kj}x_j}{x_k}\right| \leq \sum_{j=1;j\neq k}^{N}|a_{kj}| = r_k. \tag{2.14}$$

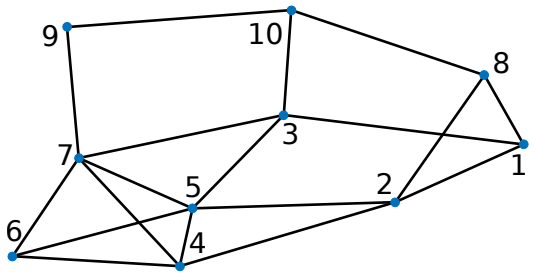


Figure 2.4. Graph to present Gershgorin theorem.

Example 2.5. To present Gershgorin theorem practically, consider the graph in Figure 2.4 ith its Laplacian matrix:

$$\boldsymbol{L} = \begin{pmatrix} 3 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 4 & 0 & -1 & -1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & 0 & -1 & 0 & -1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 5 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & -1 & 5 & 0 & -1 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{pmatrix}. \tag{2.15}$$

And a characteristic polynomial: $\mu(G,x) = x^{10} - 36x^9 + 561x^8 - 4\,954x^7 + 27\,236x^6 - 96\,318x^5 + 218\,121x^4 - 303\,398x^3 + 233\,888x^2 - 75\,870x$. Note, that $\boldsymbol{L}$ is a symmetric matrix and 0 is clearly a root of $\mu(G,x)$.

Numerically solving $\mu(G,x) = 0$ we obtain the following eigenvalues (rounding for 3 decimal points): $\{0;\ 1,274;\ 1,416;\ 3,100;\ 3,233;\ 3,936;\ 4,826;\ 5,280;\ 6,458;\ 6,476\}$. All values are real and non-negative. Finally, plotting the graph with marked eigenvalues and Gershgorin circles. As expected, all eigenvalues are included in the circles.

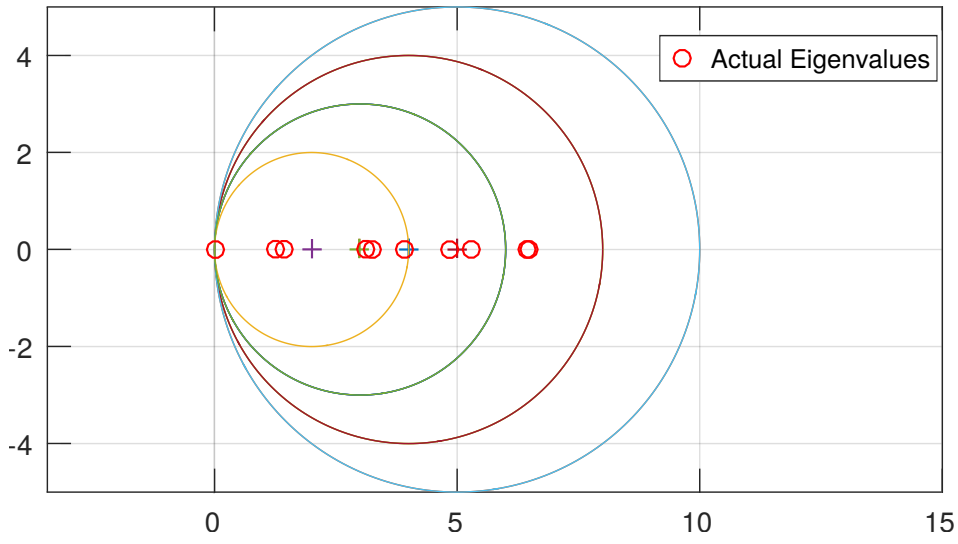


Figure 2.5. Plot of Eigenvalues and according Gershgorin circles.

Since Laplacian matrix has an interesting construction, e.g. its rows and columns sum up to zero, there may be found some interesting properties holding for its eigenvalues. We provide them here according to [8].

Theorem 2.6. Let G be a graph with N vertices. Then holds:

- $$\lambda_2 \leq \frac{N}{N-1} \min_i \{d(v_i) | v_i \in V(G)\}, \tag{2.16}$$
- $$\lambda_N \leq \max_i \{d(v_i) + d(v_j) | \, (v_i, v_j) \in E(G)\}, \tag{2.17}$$
- if G is a simple graph, then
$$\lambda_N \leq N, \tag{2.18}$$
- $$\sum_{m=1}^{N} \lambda_m = 2|E(G)| = \sum_{v_i} d(v_i), \tag{2.19}$$
- $$\lambda_N \geq \frac{N}{N-1} \max_i \{d(v_i) | v_i \in V(G)\} [8]. \tag{2.20}$$

One is recommended to check these rules e.g. on the matrix from Example 2.5. These may be found very useful for example when using numerical methods for solving the roots of $\mu(G, x)$. It is well-known, that we don't have exact analytical formulas to obtain roots of polynomials with higher degree than 5. Using these bounds, we know where the roots must be, respective where they can not be.

2.4.4 Matrix tree theorem

$\boldsymbol{L}(G)$ may be also referred to as Kirchhoff matrix due to the following theorem. We revise: A *tree* is a connected, acyclic graph; a *spanning tree* of graph G is a tree which origins as a subgraph, preserving the set of vertices $V(G)$ and removing some of its edges to avoid cycles. A spanning tree may be found only for connected graphs.

An (i, j)-cofactor of a matrix is a determinant of a sub-matrix created by deleting the i-th and the j-th column from this matrix [12].

Theorem 2.7. *Kirchhoff's Matrix-Tree Theorem.* Let G be a connected graph with its Laplacian matrix $\boldsymbol{L}(G)$. Then all $\boldsymbol{L}(G)$ cofactors are equal and this common value is the number of spanning trees of G [13].

Proof: Omitted. Is based on decomposing the Laplacian matrix into product of Incidence matrix and its transpose and then usage of Cauchy-Binet formula [13].

Example 2.8. For graph from Figure 2.4 we could so find 7587 spanning-trees.

2.4.5 Eigen value λ_2

We call graph G with N vertices connected if there is a path from any vertex v_i to any other vertex v_j , $\forall i, j \in \{1, 2, ..., N\}$.

Eigenvalue λ_2 is also called *graph connectivity* [8]. This eigenvalue is probably the most important from the whole spectrum, or at least in context of our consensus algorithm is in center of interest. Holds, that $\lambda_2 > 0$ if and only if the graph is connected and the multiplicity of 0 as eigenvalue is the number of connected components [8]. We emphasize, λ_2 is directly proportional to the rate of connectivity, i.e. high λ_2 means dense graph.

Diameter of a graph G, $diam(G)$, is the biggest number of edges we have to pass, to get from one vertex to another.

There exist some properties and bounds for λ_2. Very interesting and easily interpretable, in context of the connectivity term, is the following one. Let's consider graph G with N vertices and diameter $diam(G)$. Then holds [8]:

$$\lambda_2 \geq \frac{4}{N \cdot diam(G)}. \tag{2.21}$$

2.4.6 Operations with disjoint graphs

Very detailed reading about this part of Laplacian topic may be found beside in [8] also in [14]. Let's now briefly mention what happens with Laplacian of a graph, that is not connected.

Considering the definition of the Laplacian $\boldsymbol{L}(G) = \boldsymbol{D}(G) - \boldsymbol{A}(G)$, we are not surprised, that Laplacian of a graph consisting of k mutually disjoint sets of vertices will have block diagonal form obtained from matrices $\boldsymbol{L}(G_1), \boldsymbol{L}(G_2), ..., \boldsymbol{L}(G_k)$ [8].

Theorem 2.9. Let G be a graph created as a union of disjoint graphs $G_1, G_2, ..., G_k$. Then holds [8]:

$$\mu(G, x) = \prod_{i=1}^{k} \mu_i(G_i, x). \tag{2.22}$$

Example 2.10.

Let's take a graph from the following figure, consisting of two disjoint components with vertices $\{1, 2, 3\}$ and $\{4, 5, 6, 7\}$.

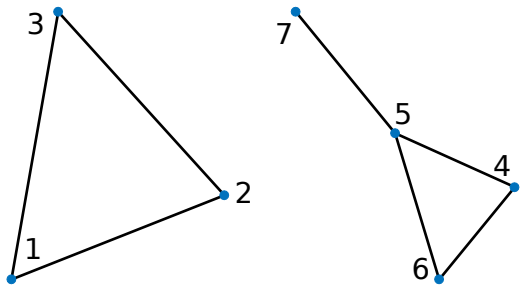


Figure 2.6. Example of a graph with two disconnected components.

Laplacian matrix of the whole graph reads:

$$\boldsymbol{L}(G) = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}. \tag{2.23}$$

$\boldsymbol{L}(G)$ is a block diagonal matrix consisting of submatrices $\boldsymbol{L}(G_{\{1,2,3\}})$ and $\boldsymbol{L}(G_{\{4,5,6,7\}})$. For characteristic polynomial holds:

$\mu(G, x) = \mu(G_{\{1,2,3\}}, x)\mu(G_{\{4,5,6,7\}}, x) = (x^3 - 6x^2 + 9x)(x^4 - 8^3 + 19x^2 - 12x) = x^7 - 14x^6 + 76x^5 - 198x^4 + 243x^3 - 108x^2$. Note, that 0 is clearly double root corresponding to the 2 components of graph.

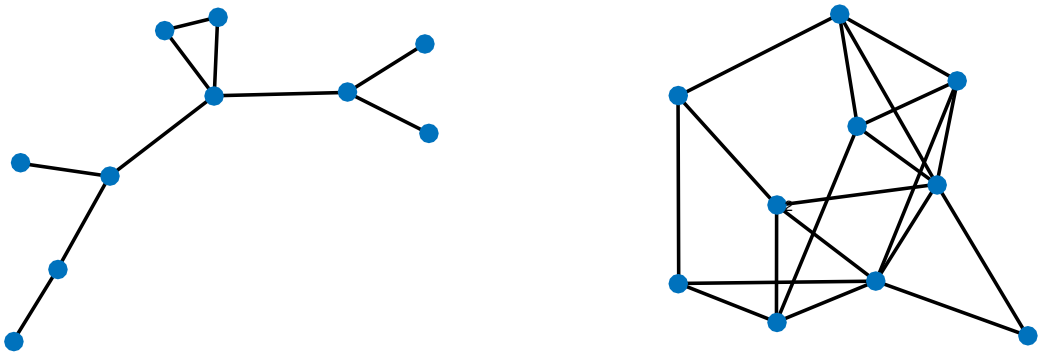


Figure 2.7. Another demonstration of Connectivity eigenvalue meaning.

Example 2.11. To get more intuition about *Connectivity eigenvalue* λ_2, consider two graphs in Figure 2.7 below.

The eigenvalues of the spare and low connected graph on the left are

$$\lambda_1^S = 0,\ \lambda_2^S \approx 0.21, ...,\ \lambda_{10}^S \approx 5.46.$$

The eigenvalues of the dense graph on the right are

$$\lambda_1^D = 0,\ \lambda_2^D \approx 1.65, ...,\ \lambda_{10}^D \approx 7.56.$$

We notice, that λ_2^S of the spare graph is almost eight times smaller than λ_2^D of relatively dense graph. In next chapter we shall see, that the *Connectivity eigenvalue* directly limits the speed of convergence.

Chapter 3
Linear average consensus algorithm

In this chapter, let us consider an undirected and connected graph $G = (V, E)$ with N vertices and edges (v_i, v_j) between vertices i, j, where $i, j \in \{1, 2, ..., N\}$. We denote an initial value $x_i(0)$ the value assigned to the i-th vertex (node, agent) in time $t = 0$, $t \in \mathbb{Z}$. Then $x_i(t)$ refers to the value in the i-th vertex in time t. Our goal is for $t \to \infty$, using only communication between vertices compute in all N vertices of the graph an average value of those initial values. Based on a matrix-like description of graph G, we want to construct matrix $\boldsymbol{P}$, whose components p_{ij} will suit this average consensus algorithm, in a form of iterative matrix multiplication [15].

In this chapter, subject of our interest will be a linear, discrete-time consensus algorithm. A detailed description of the following is in [16], [17] both containing also rich references to other publications.

3.1 Distributed algorithms

A step-by-step introduction to the theory of Distributed algorithms may be found in a book [18].

In this chapter about Linear average consensus algorithm we will assume, that:

- The Topology of the graph is fixed. Our first goal will be to find only a static algorithm, that works for all time of computing with constant Adjacency and Incidence matrices.
- The communication between vertices is reliable. So all updates for a given agent always reach a destination. In a real case, a very good level of reliability might have been reached using e.g. some ARQs algorithms used for an Ethernet network. However, this would have slowed the algorithm down.
- All nodes have globally synchronized clocks with one central time, so that the computations are synchronized (practically, we could use for example clock ticks from GPS satellites).
- We always know an initial state of each vertex, i.e. an input value to the algorithm.

3.2 Introduction

Assume this *linear* update equation

$$\boldsymbol{x}(t+1) = \boldsymbol{P}(t)\boldsymbol{x}(t), \tag{3.1}$$

where $\boldsymbol{x}(t) = (x_1(t), x_2(t), ..., x_N(t))^T \in \mathbb{R}^N$ and for all values of t, $\boldsymbol{P}(t) \in \mathbb{R}^{N \times N}$ is a *stochastic matrix*, i.e. $p_{ij}(t) \geq 0$ and $\sum_{j=1}^{N} p_{ij} = 1, \forall i, j \in 1, 2, ..., N$. Meaning, that all values in each row sum up to 1. The p_{ij} components are also often referred to as *weights* [17].

Now, let's rewrite the equation above expanding a matrix multiplication:

$$x_i(t+1) = \sum_{j=1}^{N} p_{ij}(t)x_i(t) = x_i(t) + \sum_{j=1;j\neq i}^{N} p_{ij}(x_j(t) - x_i(t)). \tag{3.2}$$

This equation is for given $\boldsymbol{P}(t)$ a general form of a *linear consensus algorithm*, that may be usually found in the literature. Frankly spoken, all the theory behind linear consensus algorithm aims to find the best matrix $\boldsymbol{P}(t)$, such as the consensus is reached.

Formally defined, we say that $\boldsymbol{P}(t)$ solves *consensus problem*, if for all i holds

$$\lim_{t\to\infty} x_i(t) = \alpha, \forall i. \tag{3.3}$$

Then, for a solution of the *average consensus problem* must be in an addition to the previous condition fulfilled also

$$\alpha = \frac{1}{N}\sum_{i=1}^{N} x_i(0). \tag{3.4}$$

Next, we call $\boldsymbol{P}(t)$ *doubly stochastic*, if holds also $\sum_{j=1}^{N} p_{ij} = 1, \forall j \in 1, 2, ..., N$. So both, rows and columns sum up to 1. Note, that if $\boldsymbol{P}(t)$ is stochastic and symmetric, $\boldsymbol{P}(t) = \boldsymbol{P}(t)^T$, then $\boldsymbol{P}(t)$ is doubly stochastic.

The $\boldsymbol{P}(t)$ matrix may be considered as: 1) constant $\boldsymbol{P}(t) = \boldsymbol{P}$, i.e. we set up only one matrix at the beginning of the computation, to be used for the whole run of an algorithm, 2) a deterministic time variable matrix, 3) randomly variable matrix; it is the most general case bearing also most complexities [16]. For simplicity, we will firstly concern only case 1).

3.3 General convergence conditions

Next, let's formulate some conditions for our constant $\boldsymbol{P}$ matrix. We call a matrix $\boldsymbol{P}$ *compatible* with a graph G, if $p_{ij} = 0$ for $j \notin \mathcal{N}_i$ (i.e. i-th node is not in a set of neighbors of the j-th node.) Still considering an undirected graph, we can write:

$$\boldsymbol{P} = \boldsymbol{P}^T. \tag{3.5}$$

We define terms *irreducible* and *primitive* matrices: we call matrix $\boldsymbol{A}$ irreducible if its associated graph G is strongly connected; and we call $\boldsymbol{A}$ primitive, if it is an irreducible stochastic matrix, that has exactly one strictly greatest modulus of eigenvalue [19].

Theorem 3.1. Perron-Frobenius theorem [19]. Let $\boldsymbol{P}$ be a primitive non-negative matrix with left and right eigenvectors $\boldsymbol{w}$ and $\boldsymbol{v}$, respectively, satisfying $\boldsymbol{Pv} = \boldsymbol{v}$, $\boldsymbol{w}^T\boldsymbol{P} = \boldsymbol{w}^T$ and $\boldsymbol{v}^T\boldsymbol{w} = 1$. Then

$$\lim_{t\to\infty} \boldsymbol{P}^t = \boldsymbol{v}\boldsymbol{w}^T. \tag{3.6}$$

(Note, the upper index t in expression $\boldsymbol{P}^t$ means power of matrix.) Perron-Frobenius theorem may be found in many stronger forms, but for us, this minimalistic form will be sufficient.

Now, let's add the desired property to make the algorithm *average.* We define an averaging matrix $\frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T$, where $\boldsymbol{1}$ denotes a column vector of N ones. Note, $\boldsymbol{1}\boldsymbol{1}^T$ is $N \times N$ matrix of all ones, however, $\boldsymbol{1}^T\boldsymbol{1} = N$, is a scalar. When multiplying this rank-one matrix with a vector $\boldsymbol{z} \in \mathbb{R}^N$, $\overline{\boldsymbol{z}} = \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{z}$, we obtain a column vector $\overline{\boldsymbol{z}}$ with all components equal to the average of all N components of the $\boldsymbol{z}$ vector [17].

What we ask about the algorithm is

$$\lim_{t\to\infty} \boldsymbol{x}(t) = lim_{t\to\infty} \boldsymbol{P}^t \ \boldsymbol{x}(0) = \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{x}(0), \tag{3.7}$$

which is for arbitrary vector $\boldsymbol{x}(0)$ equivalent to the

$$lim_{t\to\infty} \boldsymbol{P}^t = \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T. \tag{3.8}$$

Next, according to the above Perron-Frobenius theorem (3.6) and equation (3.7), our next naturally appearing condition for $\boldsymbol{P}$ is to have it doubly stochastic , i.e. :

$$\boldsymbol{P}\boldsymbol{1} = \boldsymbol{1}, \tag{3.9}$$

and

$$\boldsymbol{1}^T\boldsymbol{P} = \boldsymbol{1}^T. \tag{3.10}$$

Explicitly summarizing: to reach the convergence of the *average* consensus algorithm, the increasing powers of (not unique) stochastic matrix $\boldsymbol{P}$ must converge and moreover converge to a doubly stochastic matrix $\frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T$.

So far, we wrote down some conditions for $\boldsymbol{P}$ matrix, however it should be clear, that they definitely do not determine any unique matrix and still leave a lot of freedom how to choose it. But as there are more ways to construct $\boldsymbol{P}$ matrix, to reach convergence, for all of them will be necessary to always hold it compatible with a given graph. We must not forget, that $\boldsymbol{P}$ corresponds to a physically realizable information exchange in the graph G, so this condition allows communication only over existing edges.

Theorem 3.2. Equation

$$lim_{t\to\infty} \boldsymbol{P}^t = \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T \tag{3.11}$$

holds if and only if

$$\boldsymbol{1}^T\boldsymbol{P} = \boldsymbol{1}^T, \tag{3.12}$$

i.e. $\boldsymbol{1}$ is left eigenvector of $\boldsymbol{P}$ with eigenvalue 1,

$$\boldsymbol{P}\boldsymbol{1} = \boldsymbol{1}, \tag{3.13}$$

i.e. $\boldsymbol{1}$ is also right eigenvalue of $\boldsymbol{P}$ and

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

where $\rho(.)$ denotes the spectral radius of a matrix, i.e. the greatest absolute value of an eigenvalue.

Proof: Complete proof may be found in [20].

3.4 Heuristics based on the Laplacian matrix

Basis material for following section comes mainly from [20].

There have been developed some simple heuristics for choosing matrix $\boldsymbol{P}$, that fulfills the established conditions from the previous section. They are based on the construction of the Laplacian matrix, shown in Graph Theory chapter. So then, let us heuristically take

$$\boldsymbol{P} = \boldsymbol{I} - \alpha\boldsymbol{L}, \ \alpha \in \mathbb{R}. \tag{3.15}$$

$\boldsymbol{P}$ is often refered to as Perron matrix, because of Perron's (1880 – 1975) work in last century [21].

This $\boldsymbol{P}$ evaluates edges of graph with a value α. The first great advantage of this choice, is that such a matrix $\boldsymbol{P}$ will be automatically compatible with the graph, while it bears information about connected, respectively disconnected vertices and also, in this way, as we build the $\boldsymbol{P}$ matrix like a subtraction of an Identity matrix and some specified multiple of a Laplacian matrix, this substract-origined matrix is of course a stochastic matrix, regarding to the property of the Laplacian matrix, that all its rows sum up exactly to zero [17].

The elements of $\boldsymbol{P}$ are

$$p_{ij} = \begin{cases} \alpha & \text{if there is the edge } (v_i, v_j), \\ 1 - d_i\alpha & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \tag{3.16}$$

where we remind d_i is the degree of vertex i.

Now on, we can from equation $\boldsymbol{P} = \boldsymbol{I} - \alpha\boldsymbol{L}$ determine an expression linking eigenvalues of matrix $\boldsymbol{P}$ with eigenvalues of matrix $\boldsymbol{L}$.

Theorem 3.3. [17]:

$$\lambda_i(\boldsymbol{P}) = 1 - \alpha\lambda_{N-i+1}(\boldsymbol{L}), i = 1, 2, ..., N, \tag{3.17}$$

where $\lambda_i(.)$ stands for the $i-$th smallest eigenvalue of the symmetric matrix.

Proof: Quite simple, this can be verified writing the equation for characteristic polynomial of matrix $\alpha\boldsymbol{L}$:

$$\det\left(\alpha\boldsymbol{L} - \lambda\boldsymbol{I}\right), \tag{3.18}$$

which is using $\alpha\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{P}$ equal to

$$\det\left((\boldsymbol{I} - \boldsymbol{P}) - \lambda\boldsymbol{I}\right) = \det\left((1 - \lambda)\,\boldsymbol{I} - \boldsymbol{P}\right). \tag{3.19}$$

Next we want to solve characteristic equation $\det\left((1 - \lambda)\,\boldsymbol{I} - \boldsymbol{P}\right) = 0$. In this, we can see from RHS of (3.19), that the spectrum of $\alpha\boldsymbol{L}$ will be exactly the spectrum of $(1-\lambda(\boldsymbol{P}))$. And since holds

$$\det(a\boldsymbol{X}) = a\det(\boldsymbol{X}),$$

the proof is complete.

We have seen, that for Laplacian matrix of connected graph holds

$$\lambda_1(\boldsymbol{L}) = 0. \tag{3.20}$$

Then we can using previous theorem immediately write

$$\lambda_N(\boldsymbol{P}) = 1. \tag{3.21}$$

This is for us extremely useful! Since because of Equation (3.21) the matrix $\boldsymbol{P}$ is primitive and holds Equation (3.6) from Perron-Frobenius theorem, i.e. because there is exactly one greatest eigenvalue the the limit in Equation (3.11) is guaranteed to exist.

Theorem 3.4. (Convergence condition.) Consider a network of agents given by a strongly connected graph G with N nodes and Perron matrix $\boldsymbol{P} = \boldsymbol{I} - \alpha\boldsymbol{L}$. Applying the distributed consensus algorithm

$$\boldsymbol{x}(t + 1) = \boldsymbol{P}\boldsymbol{x}(t), \tag{3.22}$$

where $\alpha \in (0, \Delta]$, consensus is asymptotically reached for all initial states [22].

We already showed in Chapter 2, that Laplacian matrix is always positive semidefinite. Because of this property, we have to necessarily choose

$$\alpha > 0, \tag{3.23}$$

to successfully accomplish the convergence condition [20]

$$\rho\left(\boldsymbol{P} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) < 1. \tag{3.24}$$

The spectral radius of a matrix $\left(\boldsymbol{P} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$ may be then expressed as

$$\rho\left(\boldsymbol{P} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) = \max\{\lambda_{N-1}(\boldsymbol{P}), -\lambda_1(\boldsymbol{P})\} = \max\{1 - \alpha\lambda_2(\boldsymbol{L}), \alpha\lambda_N(\boldsymbol{L}) - 1\}. \tag{3.25}$$

Using the condition $\rho\left(\boldsymbol{P} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) < 1$ we can write

$$0 < \alpha < \frac{2}{\lambda_N(\boldsymbol{L})}. \tag{3.26}$$

Finally, according to [20], the choice of α to minimize $\rho\left(\boldsymbol{P} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$ is

$$\alpha^* = \frac{2}{\lambda_N(\boldsymbol{L}) + \lambda_2(\boldsymbol{L})}. \tag{3.27}$$

This is the best possible choice based on the Laplacian matrix. However, very useful is to select the coefficient as stated in [22]

$$0 < \alpha_\Delta < \frac{1}{\Delta}, \tag{3.28}$$

which choice depends only on the maximum degree in the graph, assuming the graph is strongly connected. It is useful, because we do not have to look for the eigenvalues of Laplacian matrix. Next interesting reason is, if in implementation the topology of graph would have changed, typically in a way that one of the nodes breaks and shuts down, the number Δ can only decrease, which implies the coefficient α_Δ to increase. Therefore, we don't have to be worried of algorithm divergence. Proof of asymptotical convergence may be found e.g. in [22].

Matlab script used in following Examples 3.5, 3.6 and 3.7 may be found in Appendix A.1.

Example 3.5. Let us take an undirected graph from Figure 2.2, initializing simply the $i-$th vertex with value i , $i = 1, 2, ..., 10$. In the figure 3.1 are shown the time varying values in each vertex. We can clearly see, that the all values converge to the expected value $5, 5$.

Example 3.6. In next example in Figure 3.2, let's again take exactly the same graph 2.2, but now we will during the run of algorithm fix the value in vertex $v_3 = 3$. It is quite interesting, however not surprising, that this modification causes all values to converge to the value 3.

Example 3.7. For this moment the last experiment in Figure 3.3, that we want to present over topology from Figure 2.2, is adding reasonably small Additive White Gaussian Noise (AWGN) to the updates. Noisy updates will be main subject of the next chapter.

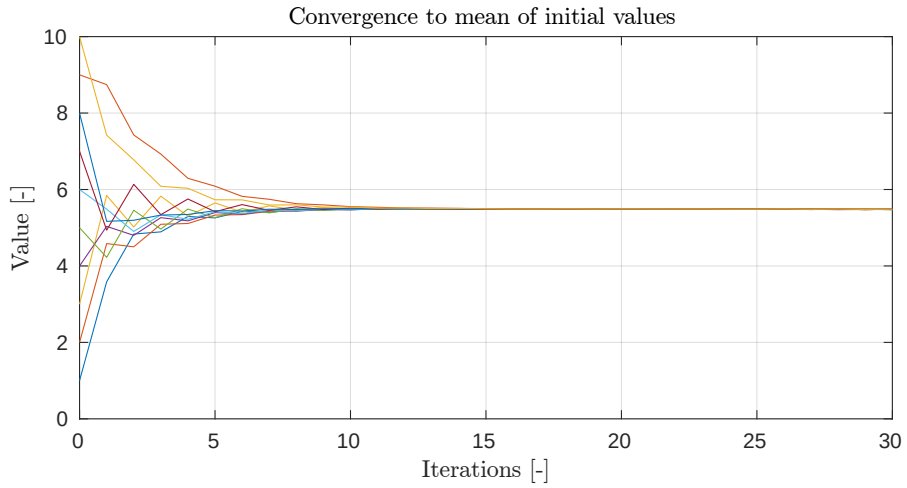


Figure 3.1. Average consensus algorithm on graph from Figure 2.2. The nodes are initialized with values 1 - 10 (vertical axis). Using exchanging updates between neighbors, the consensus is reached in 10 iterations.

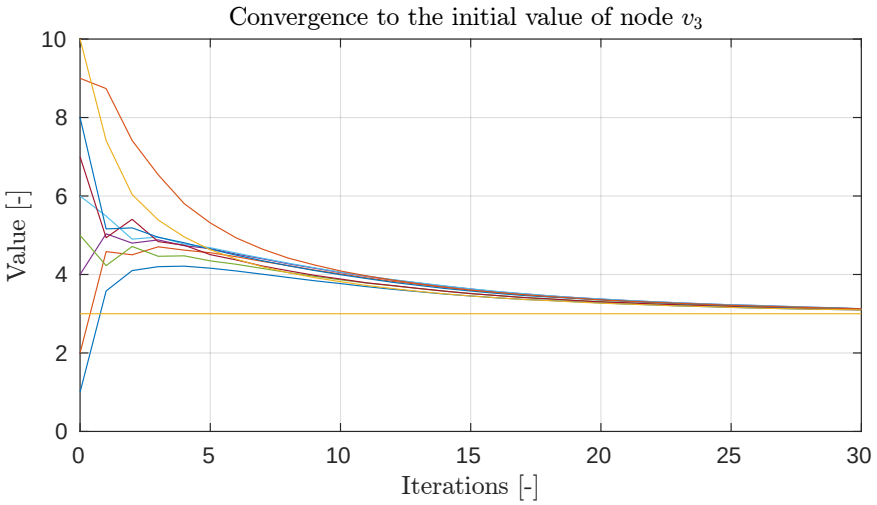


Figure 3.2. Average consensus algorithm in graph from Figure 2.2, where we again initialized nodes with values 1-10, but this time we during run of the algorithm fix the value in $v_3, x_{v_3}(t) = 3\ \forall t \geq 0$. We can see, that consequently all nodes converge to the fixed value.

3.4.1 The Metropolis-Hastings weighting method

As mentioned before, the choice to construct $\boldsymbol{P}$ is not unique. Although the Laplacian based approach is in the related basic literature probably the most common, we will for illustration give one other satisfactory example [23].

The Metropolis-Hastings weighting method coefficients of $\boldsymbol{P}^{MH}$ matrix are

$$p_{ij}^{MH} = \begin{cases} \frac{1}{1+\max(d_i,\ d_j)} & \text{for } j \in \mathcal{N}_i,\ i \neq j, \\ 1 - \sum_{j \in \mathcal{N}_i} p_{ij} & \text{if } i = j, \\ 0 & \text{otherwise [23].} \end{cases} \tag{3.29}$$

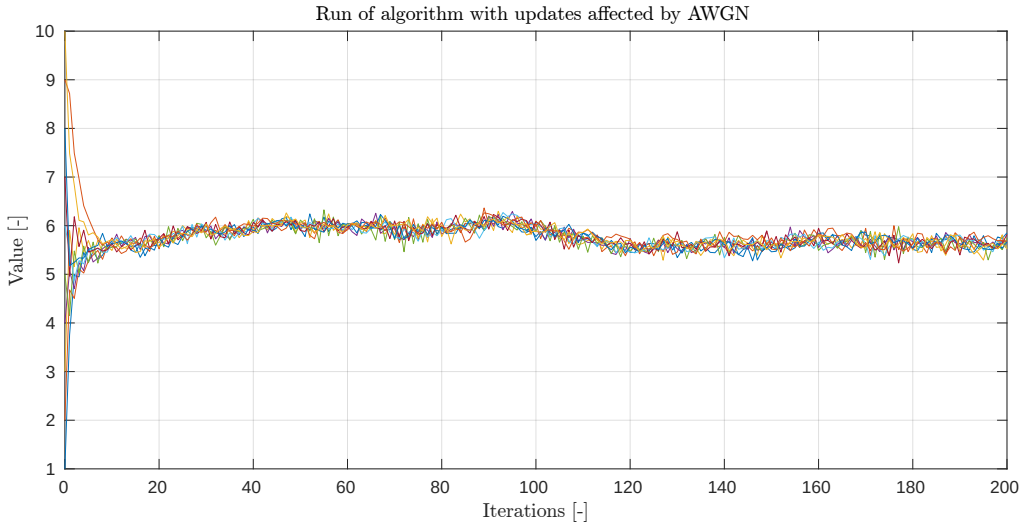


Figure 3.3. Run of average consensus algorithm on graph from Figure 2.2 with noisy updates. We can see, that adding noise to the updates in run of simple version of algorithm causes the algorithm to lose convergence property.

3.5 Average consensus algorithm with additive noise

Source for this section is [17].

In previous text, we assumed during the run of algorithm a perfectly reliable communication. Now, let's have a look, what happens, when this holds no more. The simplest case to begin with, is an Additive noise $\boldsymbol{w}(t) \in \mathbb{R}^N = (w_1(t), w_2(t), ..., w_N(t))$. In detail, the $\boldsymbol{w}(t)$ is a random variable with zero mean and unit variance. Mathematically formulated, the average consensus algorithm affected by additive noise will be described as

$$\boldsymbol{x}(t+1) = \boldsymbol{P}(\boldsymbol{x}(t) + \boldsymbol{w}(t)) \tag{3.30},$$

where we expect that $\boldsymbol{P}$ fulfills the convergence conditions stated before. Note, that now the sequence of the vectors $\boldsymbol{x}(t)$ becomes to be a *stochastic process*, i.e. a set of random variables parameterized by the time t.

By $\mathbb{E}[.]$, we denote an Expectation operator, (i.e. the Mean). Then since in (3.30) we expected a zero mean, it implies that holds

$$\mathbb{E}[\boldsymbol{x}(t+1)] = \boldsymbol{P}\mathbb{E}[\boldsymbol{x}(t)].$$

So using the operator $\mathbb{E}[.]$, the algorithm doesn't even know about the noise, while its mean is zero. This models a situation, when to the updates that are being exchanged is added some noise. However, the values in each vertex do not converge at all. To present this, let's define a function

$$a(t) = \frac{1}{N}\boldsymbol{1}^T\boldsymbol{x}(t), \tag{3.31}$$

that provides an average value of a vector $\boldsymbol{x}(t)$ (i.e. a scalar) [17]. Then

$$a(t+1) = \frac{1}{N}\boldsymbol{1}^T\boldsymbol{x}(t+1) = \frac{1}{N}\boldsymbol{1}^T\left(\boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{w}(t)\right) = \frac{1}{N}\boldsymbol{1}^T\boldsymbol{P}\boldsymbol{x}(t) + \frac{1}{N}\boldsymbol{1}^T\boldsymbol{P}\boldsymbol{w}(t) =$$
$$= \left|\boldsymbol{1}^T\boldsymbol{P} = \boldsymbol{1}^T\right| = a(t) + \frac{1}{N}\boldsymbol{1}^T\boldsymbol{w}(t).$$

Note, the expression $\frac{1}{N}\boldsymbol{1}^T\boldsymbol{w}(t)$ is nothing but a sequence of random variables, which implies that $a(t)$ has the following properties [17]:

$$\mathbb{E}[a(t)] = a(0) \tag{3.32}$$

and

$$\mathbb{E}\left[a(t) - \mathbb{E}[a(t)]\right]^2 = t. \tag{3.33}$$

(Note, in Eq. (3.33) we used a fact the $var[\boldsymbol{w}(t)] = 1$.) We emphasize, that (3.33) means, the variance of a random variable $a(t)$ increases linearly with time. It's also interesting, that nor (3.32) nor (3.33) do not depend on the structure of matrix $\boldsymbol{P}$ [17]. Let's revisit the Figure 3.3. The increasing variance problem may be seen in Figure 3.4, which is the same setup as in Figure 3.3 but showing more iterations of the algorithm. We can see that there is no convergence and the obtained data are therefore useless.

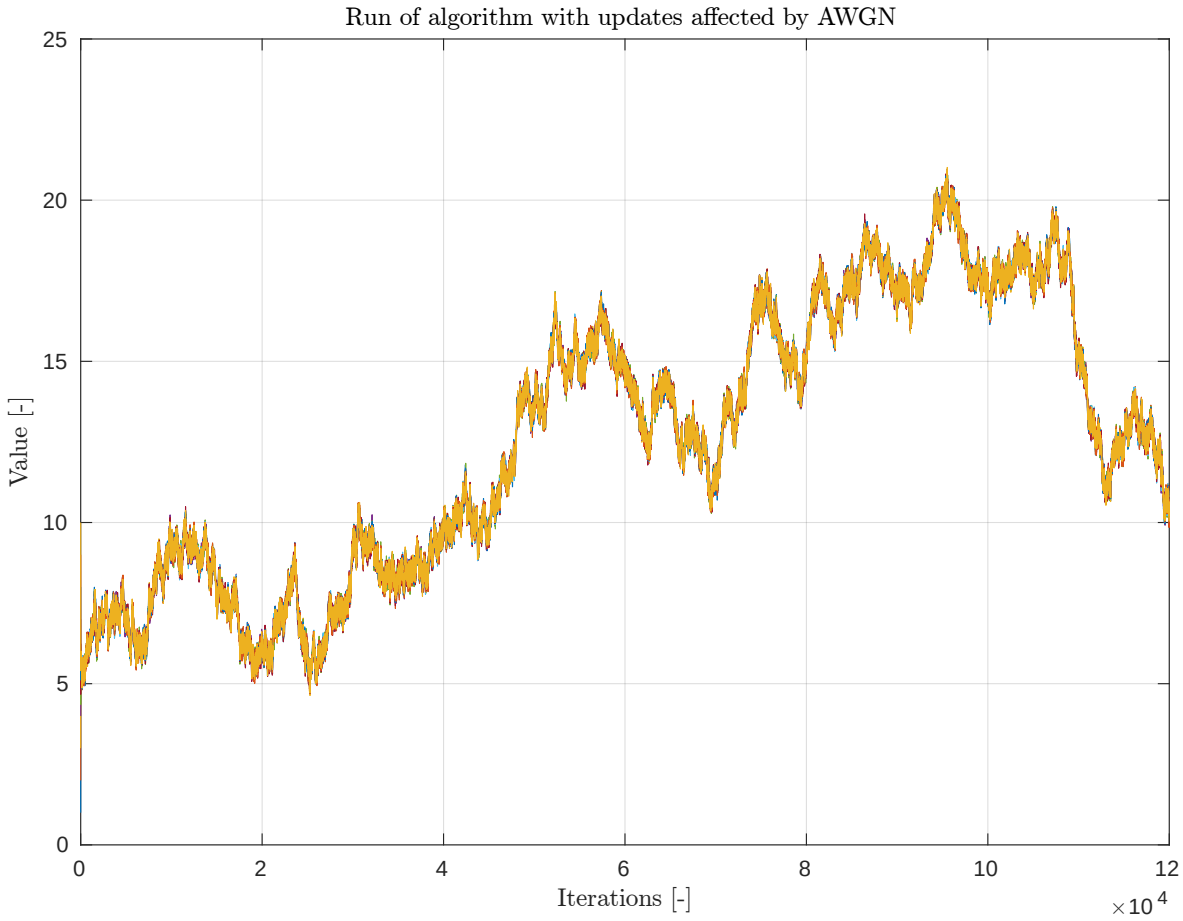


Figure 3.4. More samples to Figure 3.3 show failure of the simplest version of algorithm in presence of noise.

3.6 Mean-square convergence in case of noisy updates and observations

As we have seen above, when taking into account noisy updates, during the run of algorithm, as stated in Equation (3.1), i.e. with *constant* matrix $\boldsymbol{P}$, the algorithm fails

to converge because of increasing variance. In the following section we provide a simple way, according to [24], how to solve this problem. The core idea of the approach is to change the coefficient α from Equation (3.15) to some time variable γ, i.e. $\gamma = \gamma(t)$, and decreasing in time, so that $\gamma(t+1) < \gamma(t)$. Roughly spoken, when properly choosing the sequence $\{\gamma(t)\}_{t=1}^{\infty}$, the effect of updates gradually fades away. This is a basic concept, that is used in many more sophisticated methods that take into account the nonidealities of transmission. Later, we will provide some references.

3.6.1 Model setup

Let's specify the model of the situation first. In the previous parts, we in fact didn't assume anything about the values that were the inputs of the algorithm. The model could have been used as a way to compute an average value of any general inputs.

Now we will follow a model of a situation, where our N nodes observe one fixed-value $\theta \in \mathbb{R}$ in the network with presence of Additive noise w (with zero mean), so that the observation of i-th node, $i = 1, ..., N$, is

$$x_i(0) = \theta + w_i. \tag{3.34}$$

Let's again organize these observations into the vector $\boldsymbol{x}(t) = (x_1(t), x_2(t), ..., x_N(t))^T$. So we keep the taken observations of the nodes, affected by the noise $\boldsymbol{w}(t)$, in vector $\boldsymbol{x}(0) = (x_1(0), x_2(0), ..., x_N(0))^T$. These observations are assumed to be unbiased, uncorrelated and with variance σ^2 and with some finite mean value.

3.6.2 Time-varying weights

Firstly, we change the Perron matrix of the algorithm by adding a scalar weight $\gamma = \gamma(t)$ that will gradually decrease the impact of updates

$$\boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{I} - \gamma(t)\boldsymbol{L}, \tag{3.35}$$

and the impact of noise on the according graph's edge will be integrated to the algorithm via elements of noise matrix $\boldsymbol{N}(t)$, i.e. $n_{ij}(t)$, as

$$\boldsymbol{x}(t+1) = \boldsymbol{P}(t)\boldsymbol{x}(t) + diag\{\boldsymbol{P}(t)\boldsymbol{N}(t)\} = \boldsymbol{P}(t)\boldsymbol{x}(t) + \boldsymbol{m}(t), \tag{3.36}$$

where the vector $\boldsymbol{m}(t) = diag\{\boldsymbol{P}(t)\boldsymbol{N}(t)\}$ consists of the diagonal elements of the product of matrices $(\boldsymbol{P}(t)\boldsymbol{N}(t))$, by which we model the situation when the values of neighbors reach the given node affected with Additive noise. The vector $\boldsymbol{m}(t)$ forms a random process and its stochastic parameters, i.e. variance and mean, will, of course, depend on the weight $\gamma(t)$, that is hidden in $\boldsymbol{P}(t)$ and also on the parameters of matrix $\boldsymbol{N}(t)$ discussed above [25].

For the time-varying weight sequence must hold

$$\sum_{t=0}^{\infty} \gamma(t) = \infty, \tag{3.37}$$

and

$$\sum_{t=0}^{\infty} \gamma^2(t) < \infty, \tag{3.38}$$

in order to reach convergence. The proof may be found in [26].

3.6.3 Approach of descending step size

There are more approaches to design the step size weights. In [27] was verified that a valid choice might be e.g. sequence in form

$$\gamma(t) = \frac{a}{(b+t)^c}, \tag{3.39}$$

where $a > 0, b > 0, c \in (0,5;1]$. Using this sequence will guarantee the convergence for any initial step in sense of decreasing mean square error (MSE), because the conditions in Equations (3.37) and (3.38) are fulfilled, however to avoid an initial divergence of algorithm, corresponding to situation when doesn't hold Condition (3.26), the initial step should be selected to be smaller then $\frac{2}{\Delta}$ for the given graph [24] [26] [25].

Example 3.8. Now we give an example, how the above method works. For this simulation we use a Matlab library graph Bucky with $N = 60$ nodes, each having 3 neighbors. (Its pattern looks like a surface of a soccer ball, see Figure 3.5.) All nodes observe a constant $\theta = 10$ in a zero mean additive noise with variance $\sigma_{observation} = 1$ and the according updates are, traveling through the graph, affected with zero mean additive noise with variance $\sigma_{updates} = 0,1$. Imagine, that the observed value is some physical quantity, e.g. time base, temperature, humidity, etc. with arbitrary scaling factor located e.g. somewhere in the middle of the bucky graph. Since the algorithm is linear, only ratio of the measured values and the variances $\sigma_{observation}$, $\sigma_{updates}$ is meaningful. The script used for the simulation is in Appendix A.2 and it is prepared for a situation $\sigma_{observation} = 10\sigma_{updates}$.

We selected the decreasing sequence $\gamma(t) = \frac{1}{(42+t)^{0,75}}$, which satisfies that $\gamma(0) < \frac{1}{\Delta} = \frac{1}{3}$. To demonstrate the convergence of algorithm, we run 100 000 iterations.

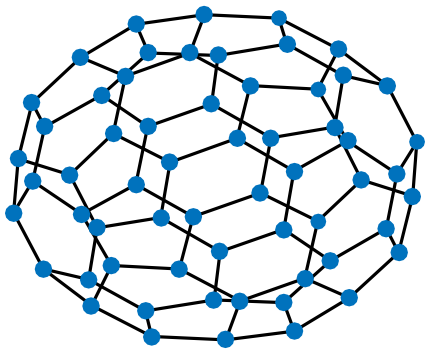


Figure 3.5. Graph used in simulation in Example 3.8.

We can see in Figure 3.6 that such a modification of algorithm works and the values in all nodes converge to one value, altough very slowly. Selecting the sequence $\gamma(t)$ does not have any general rules (or at least we didn't manage to find them) and there is a tradeoff between speed of covergence and to value up to which the MSE decreases. In next Figure 3.8 we can see in loglog graph, how the Mean square error of values truly decreases. In similiar Figure 3.7 we demonstrare another important fact: The MSE of actual values in each decreases according to Figure 3.8, but they do not come to the average of initial values. This shows Figure 3.7. MSE with respect an initial average will stop decreasing at some point. In this sense we have to say, that values in all

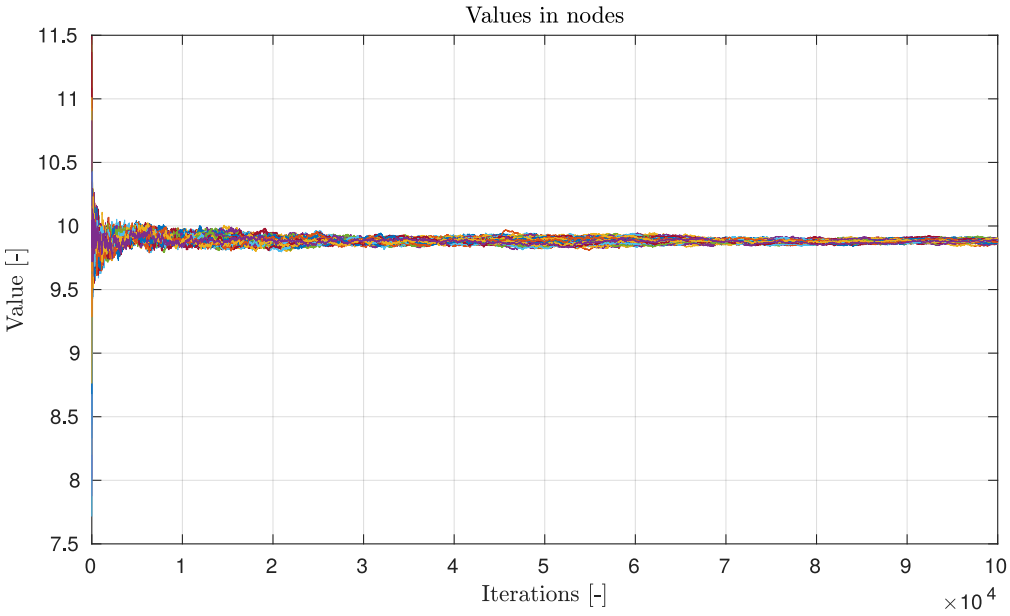


Figure 3.6. The values in nodes of graph in run of 100 000 iterations of the algorithm from Example 3.8. Precious result is, that using the decending step size the algorithm does not diverge.

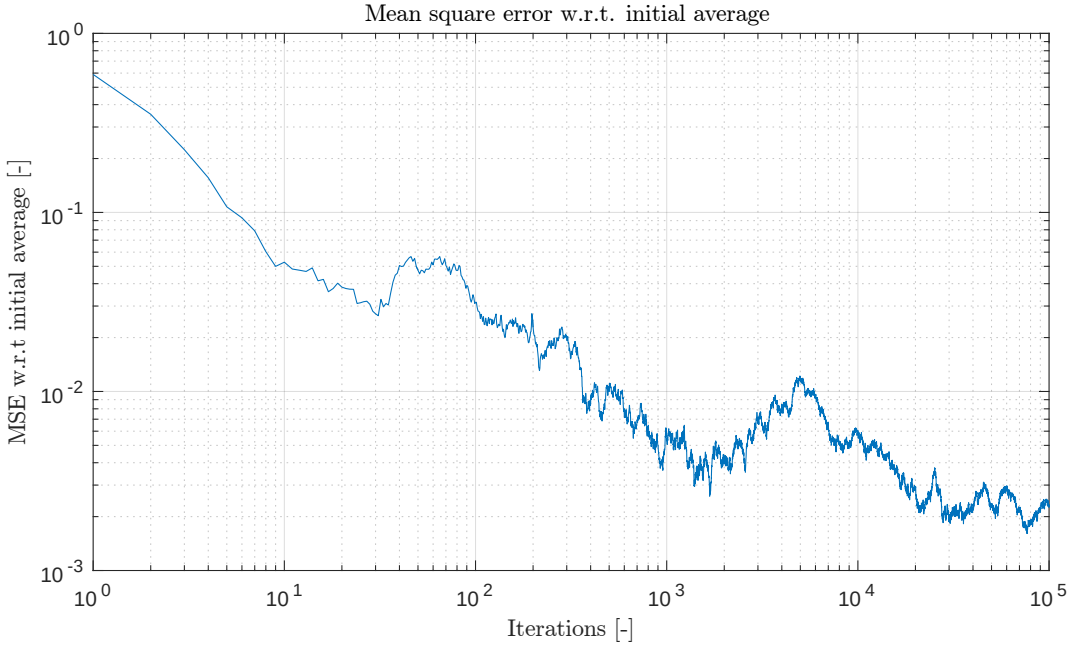


Figure 3.7. Decreasing MSE w.r.t. moving average in nodes of graph in run of the algorithm from Example 3.8. Decrease of this value consequently slows down, because variance of the noise comes to be still more significant w.r.t the small differences in updates.

nodes converge asympototically to one value, but they *do not converge* to the average of *initial values.*

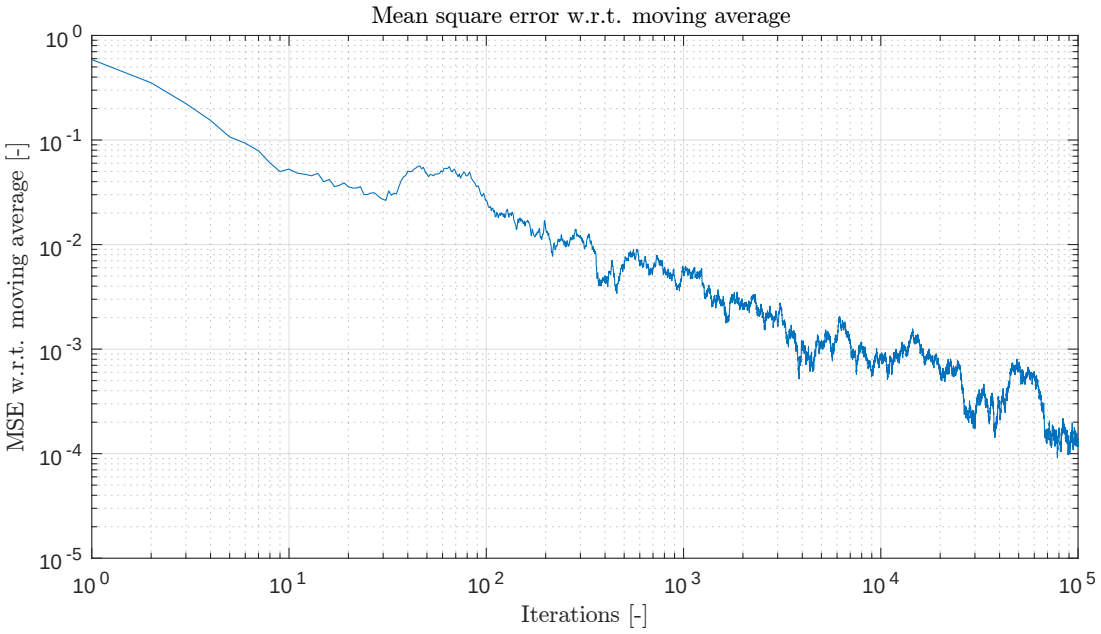


Figure 3.8. Decreasing Mean square error w.r.t. Moving average in nodes of graph in each iteration of run of the algorithm from Example 3.8. We can note, that this value continues to decrease according to fact, that difference between nodes converge to unique value.

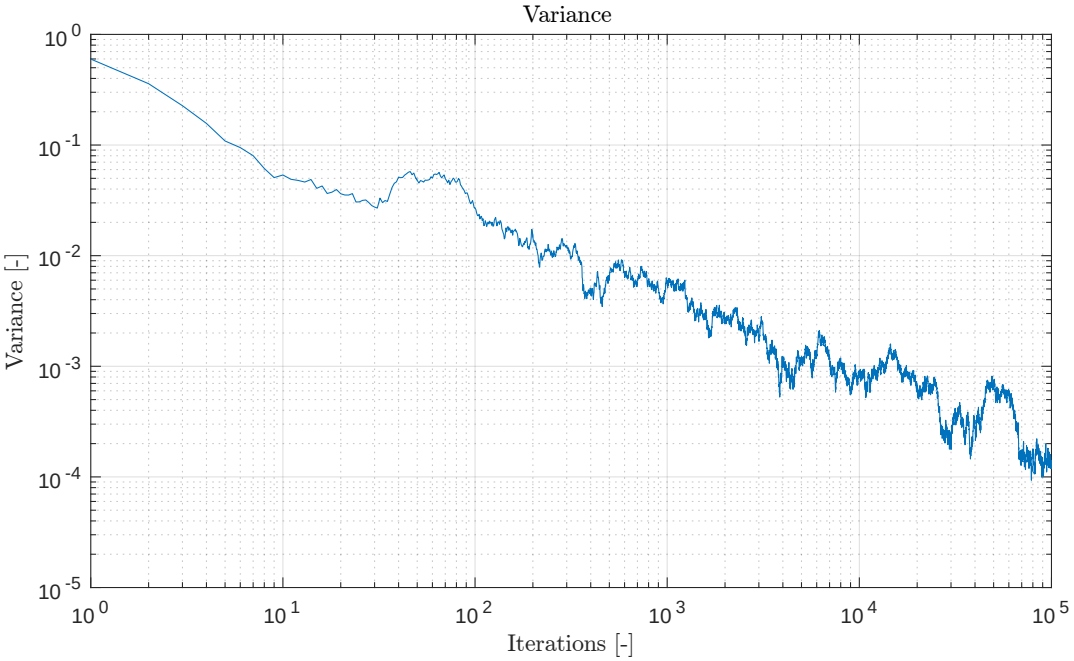


Figure 3.9. Decreasing variance of the values in nodes of graph in run of the algorithm from Example 3.8. We note, that shape of this curve is the same as for Moving MSE from Figure 3.7. This is related to the fact, that for an unbiased estimation MSE is equal to variance.

Example 3.9.

Second example, that we provide here is with the same settings of $\theta, \sigma_{observation}$, $\sigma_{updates}, \gamma(t)$ values as in Example 3.8, but this time with ring topology, again with 60 nodes.

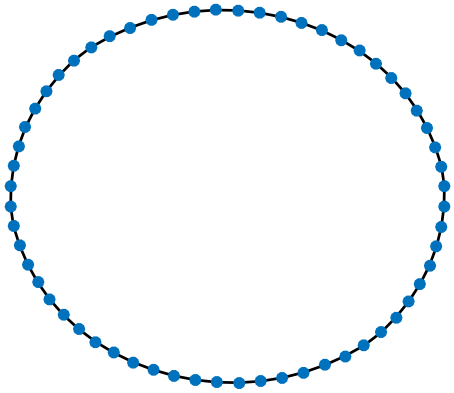


Figure 3.10. 60 nodes ring topology from Example 3.9.

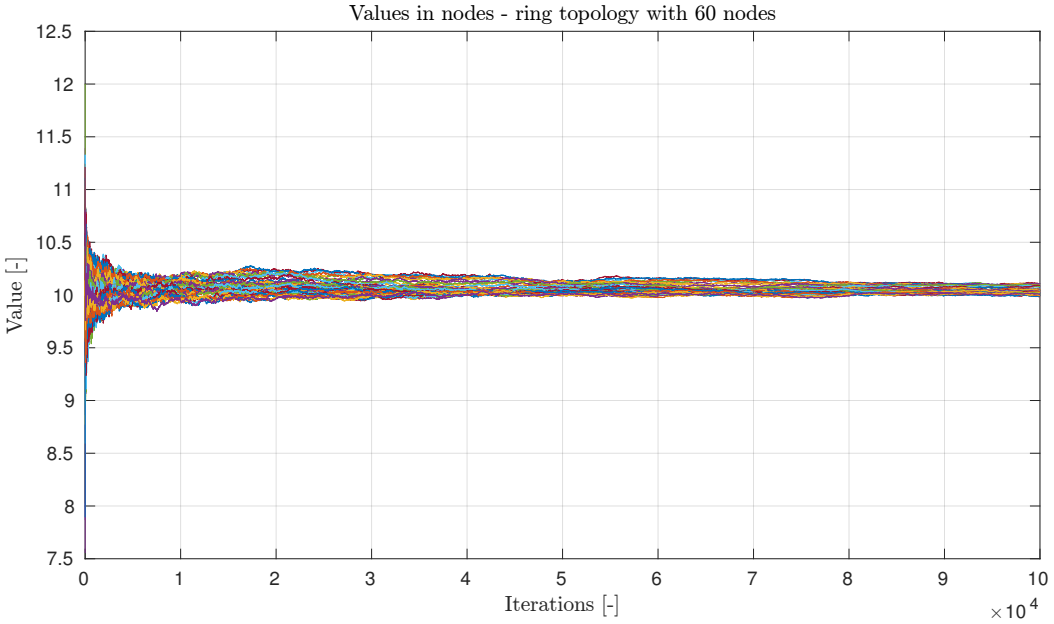


Figure 3.11. Values convergence from Example 3.9.

Let's compare the results from Examples 3.9 and 3.8. Both nodes topologies are in environment with the same stochastical parameters and hence, we can see the Bucky topology converges much faster. Of course, it is because of the connectivity of first graph is much bigger, specifically in Ex. 3.9 $\lambda_2^{bucky} \approx 0,24340$ and $\lambda_2^{ring} \approx 0,01096$.

Note: since Laplacian of ring topology graph depends only on the number of nodes, it is quite interesting, that the connectivity eigenvalue can be computed analytically. The equation to compute Connectivity eigenvalue of Laplacian matrix of N-node ring topology is:

$$\lambda_2 = 2 - 2\cos\left(\frac{2\pi}{N}\right), \tag{3.40}$$

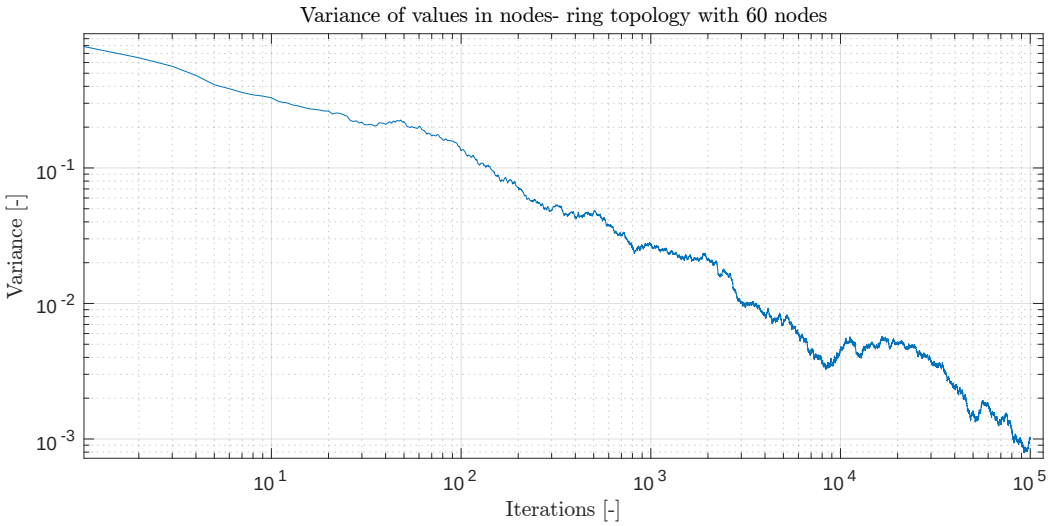


Figure 3.12. Variance running from Example 3.9.

and one can check, that as the number of vertices increases, the connectivity decreases referring to the fact, that the information takes longer time to propagate [28].

3.6.4 Recommendations of literature concerning noisy updates problematic

Here we list some publications, that solve the problematic of noisy updates in a formal way with proofs, which are behind scope of this text.

- [27] : Here is described the purpose of descending weights during the run of algorithm with noisy updates.
- [25] : This paper is about designing of the weights that minimize MSE during the run of algorithm. There may be found also an interesting concept about designing the weights without explicit knowledge of graph Laplacian, which is quite advantageous since in implementation sometimes the exact topology we simply do not know.
- [29] : Herein were presented two algorithms, that solve problematic of noisy updates. The basic idea remains in designing of sufficient descending weights but solution provided here is probably state of the art. The same authors published also [30] where are provided further details.
- [31] : Takes into account quantization noise that takes effect while implementing algorithm e.g. in fixed point arithmetic.

Chapter 4
Distributed Estimation in Wireless Sensor Networks

The following chapter should serve as an description of the problematic of distributed estimations in wireless networks. We will summarize firstly the problematic overview, with focus on the aspects that complicate the estimation process, and then provide description of some basic approaches of implementation of a consensus algorithm, that respects the true non ideal properties of real networks.

This chapter is based on source [32].

4.1 Introduction

Wireless networks are present in a great number of applications of an everyday life. The most commonly given examples are e.g. communication systems (mobile networks, LTE, Wi-Fi), all sorts of measuring devices (technical, medical, industry and other usage). Currently very popular are autonomous cars. In the context of our main topic, the average consensus algorithm might be commonly seen in solving a problem of moving in coordinated formations (e.g. flights of drones and others unmanned vehicles).

Subject of our interest will be now Wirelless Sensor Networks (WSN) with smart nodes, i.e. nodes, that own some computing power and are programmable (i.e. not just an ordinary sensor). By localizing such nodes in some area to create a network, we aim to avoid using any kind of centralized topology. One benefit is, that WSN becomes to be much more robust with respect to a single failure point of a central device. (We do not have any). Naturally, now we are not limited to a fixed topology and the nodes can move. Even more important is, that in a situation, when nodes of WSN aim to estimate some value θ in a distributed manner, it is naturally much faster when the nodes communicate directly. Having nodes, that are battery-supplied, we are also glad to avoid using the limited amount of power without necessity. It seems naturally, that more effective way is to use only direct communication between nodes, that want to obtain the desired estimation, without employing some distant data center.

WSN networks may consist of hundreds of nodes. Its goal is to estimate the value cooperatively and locally, instead of moving the calculation somewhere else, because sometimes it might not be even possible. Typically, in wireless communication the nodes should obtain an overview of the network parameters (network topology, carrier frequency, common time base). Concerning the topology, one node initially knows only its direct neighbors. In digital communication applications is typically necessary, that each node knows the topology of whole network (an Adjacency matrix). This is exactly the case, when we can not avoid a cooperation between nodes. With respect to the previous chapters, we note that the WSN can be for a purpose of distributed consensus algorithm simply coded into a graph with, expressing the possibility of an information exchange between nodes [32].

4.2 Overview of Distributed Consensus Estimation

Our general observation model will be

$$\boldsymbol{z} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{w}, \tag{4.1}$$

where $\boldsymbol{z}$ is our observation, $\boldsymbol{x}$ is the target value, $\boldsymbol{H}$ is an observation matrix and $\boldsymbol{w}$ is the additive noise.

Typical problems in WSN estimation process are:

- Noise in the network. We generally never receive the same value as we send. The observation process is also affected by noise.
- Wireless devices are battery-powered and because of that, the estimation process should be effective and not to waste the power.
- Topology of WSN may generally change.
- The algorithm requires a common clock synchronization between nodes.
- The communication between two nodes generally doesn't have to be symmetric, so that e.g. A node can send information to B, however B is not able answer to A directly. Hence, a graph describing the WSN inter-nodes communication is then the directed graph.

4.2.1 Consensus-Based Distributed Parameter Estimation

In [32] may be found quite general concept of solution to the estimation process in WSN. We will use it to present the main difficulties related to the algorithm design. The authors there use two kinds of sensors: Sensor Nodes (SN), that locally measure the value we want to globally estimate and Relay Nodes (RN), that do not measure anything but only distribute the measure results of SN to other nodes. This concept would have been practically used to save money for many expensive sensors. Such a network we call heterogeneous and hence, all the nodes do not have the same impact on the result. By adding RN into the WSN, we also naturally increase the connectivity of the graph. Typically, we can add cheap RN into the network to enable and/or improve the connectivity of all SN.

Next we briefly list the main difficulties that must be considered using this model.

4.2.2 Asymmetric communication

Since this model describes situation with asymmetric communication channels a directed graph representation must be used. In our case, a weighted graph G is a triplet $G = (V, E, \boldsymbol{A})$, where V is the set of RN and SN, $E \subset V \times V$, and $\boldsymbol{A}$ is a matrix of weights associated to edges E. For elements of $\boldsymbol{A}$ holds

$$a_{ij} > 0 \Leftrightarrow (i, j) \in E. \tag{4.2}$$

Note, this $\boldsymbol{A}$ matrix is a generalization of already before used Adjacency matrix. Considering directed graphs, it is in a literature common to call a source vertex of an edge *parent* and the destination vertex *child* [32].

Assuming, that $i-$th node transmits to the $j-$th at a constant power level P_{T_i} in distance d_{ij}, the communication will be successful if holds the inequality for Signal to Noise Ratio (SNR)

$$\frac{P_{T_i}}{\mathbb{N} d_{ij}^{\eta}} \geq \beta, \tag{4.3}$$

where $\mathbb{N}$ stands for a power level of noise in the channel, η is an exponent expressing the lossy behavior of the channel and β is the minimum SNR value fulfilling the condition for communication [32]. From Equation (4.3) we can determine a maximum distance $d_{ij_{\max}}$ between nodes i, j, that will serve as a threshold for evaluating the communication as possible

$$d_{ij_{\max}} = \sqrt[\eta]{\frac{P_{T_i}}{\mathbb{N}\beta}}. \tag{4.4}$$

4.2.3 Multidimensional observation

In [32] is solved a problem of estimation of a vector $\theta \in \mathbb{R}^J$, whose components are separately measured by SN. The measurement of the desired vector θ is described as

$$\boldsymbol{y_i}(t) = \boldsymbol{H_i}\theta + \boldsymbol{w_i}(t), \forall i \in I_S \tag{4.5}$$

where $\boldsymbol{H_i} \in \mathbb{R}^{J_i \times J}$ is an observation matrix and $\boldsymbol{w_i}(t)$ is white Additive Gaussian noise. The statement $J_i \leq J$ for $i-$th SN means, that it generally provides only limited information about the vector θ, because it can't measure the rest of components. The RN nodes can't measure θ at all, because they are not equipped with the sensors.

The vector nature of θ will consequently bring to the matrix description of the update equations, similar to the Equation (3.1), Kronecker product. Although such a description is possible, it is in my opinion very confusing and is reasonable to use probably only in theory. To see this matrix description, we refer to [32].

Definiton 4.1. (Kronecker Product) [33] Given matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and matrix $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, their Kronecker Product $\boldsymbol{C} \in \mathbb{R}^{mp \times nq}$ is denoted

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B},$$

where $c_{\alpha\beta} = a_{ij}b_{kl}$, using $\alpha = p(i-1)+k$ and $\beta = q(j-1)+l$. To give an intuition, we provide a simple example.

Example 4.2. (Kronecker product practice.) Calculate a Kronecker product $\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{J}$, where

$$\boldsymbol{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \boldsymbol{J} = \begin{pmatrix} j & k \\ l & m \end{pmatrix}.$$

Solution:

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{J} = \begin{pmatrix} aj & ak & bj & bk \\ al & am & bl & bm \\ cj & ck & dj & dk \\ cl & cm & dl & dm \end{pmatrix}.$$

4.2.4 Description of Algorithm DCUE

The DCUE algorithm assumes perfectly synchronized updates. We use the following notation in this subsection: $\boldsymbol{y_i}(t)$ is, as defined in Equation (4.5), an observation of vector θ, determined in the $i-$th node by the Observation matrix $\boldsymbol{H_i}$ in the specific node and affected by the noise vector $\boldsymbol{w_i}(t)$. Next, $\boldsymbol{x_i}(t)$ is a vector containing local estimations of the vector parameter θ in the particular nodes. All the column vectors of local estimations in nodes might be placed into a vector of vectors (i.e. a matrix) $\boldsymbol{X}(t) = (\boldsymbol{x_1}(t),\ \boldsymbol{x_2}(t),\ ..., \boldsymbol{x_N}(t))$. In the case of convergence of the algorithm, as $t \to \infty$, single elements of each raw of this matrix $\boldsymbol{X}(\infty)$ will be equal. In each node i is initially known a value $\boldsymbol{x_i}(0)$ (motivated as the measurement). $\mathcal{N}_i^S$ and $\mathcal{N}_i^R$ mark the set of SN

neighbors and RN neighbors to the $i-$th node, respectively. The update equation of DCUE algorithm states:

$$\boldsymbol{x_i}(t+1) = \boldsymbol{x_i}(t) + \rho(t)\alpha_i \boldsymbol{H_i}^T \left(\boldsymbol{y_i}(t) - \boldsymbol{H_i}\boldsymbol{x_i}(t)\right) + \\ + \frac{\rho(t)}{c_i}\left[\sum_{j\in\mathcal{N}_i^S} a_{ij}(\boldsymbol{x_j}(t) - \boldsymbol{x_i}(t)) + \sum_{j\in\mathcal{N}_i^R} a_{ij}(\boldsymbol{z_j}(t) - \boldsymbol{x_i}(t))\right], \tag{4.6}$$

where $\alpha_i > 0$ controls the update rate of information during the run of algorithm; $\rho(t) > 0$ is a weight controlling an impact of received updates (we preserve the notation of the authors of [32] here: in the previous text, see Subection 3.6.3, this factor $\rho(t)$ was denoted as $\gamma(t)$ i.e. it provides the descending step size); $c_i > 0$ controls impact of $i-$th own measurement (i.e. we use more different observations in different times in particular nodes) and $a_{ij} = \sqrt{\frac{P_{T_j}|h_{ij}|^2}{d_{ij}^{\eta}}}$ represents an amplitude of a signal received by node i from node j (i.e. elements of the Adjacency matrix), in which h_{ij} is a fading coefficient describing a channel between i and j, and it is a reason why $a_{ij} \neq a_{ji}$ [32].

Next, an update equation for RN nodes reads

$$\boldsymbol{z_i}(t) = \sum_{j\in\mathcal{N}_i^S} \gamma_{ij}\boldsymbol{x_j}(t) + \sum_{j\in\mathcal{N}_i^R} \gamma_{ij}\boldsymbol{z_j}(t), \forall i \in I_R, \tag{4.7}$$

where γ_{ij} are some non negative weighting coefficients [32].

Summarizing the meaning of equations describing the algorithm, that we stated above: Firstly, Equation (4.5) says, that observation of node i, $\boldsymbol{y_i}(t)$, in time t are elements of vector θ, according to its appropriate matrix $\boldsymbol{H_i}$, and this observation is affected by the Gaussian noise $\boldsymbol{w_i}$. Next, Equation (4.6) proposes the way how $i-$th SN should update the values to be estimated. It consists of a specific linear combination of values that node i self measures and values that it receives from neighbors SNs and RNs, respectively. Doing this, we take into account distance between neighbors (i.e. in units of length, not number of hops), properties of the channel between nodes, transmitting power and consequently we decide, whether the update can be applied, according to these channel characteristics. Finally, Equation (4.7) is an analogy update equation for i-th RN, whose updates are determined as a linear combination from values of its RNs and SNs neighbors.

Chapter 5
Examples of Usage of Average consensus algorithm in wireless communication

In this chapter we provide few more examples of usage of Average consensus algorithm in wireless digital communication.

5.1 Distributed estimation of the number of deployed nodes

Let's consider a situation, that in some area we placed number of nodes, each marked with its unique identification I, for simplicity we take $I = 1, 2, ..., N$; and these nodes are supposed to exchange information [15]. We want to estimate the overall number of sensors N. We will model this problem as Average consensus algorithm over the set I initialized to the specific vertex. From Figure 5.2 we can read, that in this particular example an average node number $\overline{I}$ is

$$\overline{I} = \frac{1+N}{2} = 50. \tag{5.1}$$

Then we follow to find the number N as

$$N = 2\overline{I} - 1 = 2 \cdot 50 - 1 = 99, \tag{5.2}$$

and 99 is indeed the number used in simulation. Note: a priori none of the nodes didn't know the parameter N, and using simple Equation (5.2) the information about total number of nodes will be obtained in all nodes.

An alternative to this approach is solving this problem in a way, that all nodes send some acknowledgment to a central point, CP computes them and sends the number of nodes back, but using the approach described in our example, we avoid a risk of a central point failure. Matlab script used for simulation is in Appendix A.3.

5.2 Tracking of dynamic target

Let's consider a small graph with only 5 nodes. Four of them, $v_1 - v_4$ are followers and they track a target v_5. In this implementation we expand the previous Example 3.6, where we fixed one vertex value of the algorithm and observed how the rest of vertex values are asymptotically converging to it (see Figure 3.2). We make two changes now. The first is trivial, we run parallel two runs of the algorithm with meaning of moving in plane xy space (see Figure below). The second, the previosly fixed parameter will slowly change (randomly).

In Matlab script in Appendix A.4 we provide an implementation, where the target is initialized in some random position and for a half of the simulation time it has some small dynamics, implemented as moving for some small random distance in each step

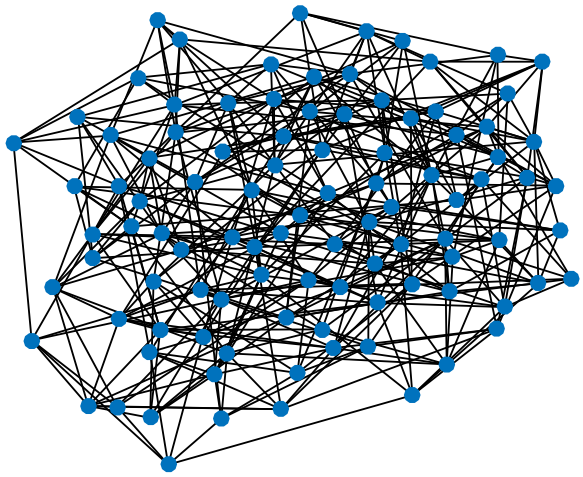


Figure 5.1. Figure of graph on which we demonstrate estimation of overall nodes number using local communication.

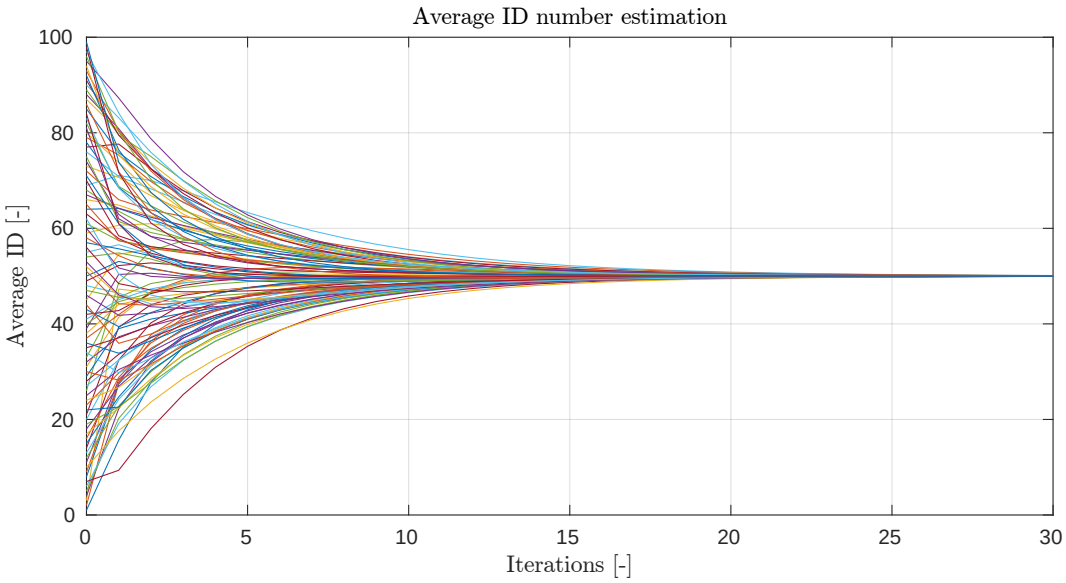
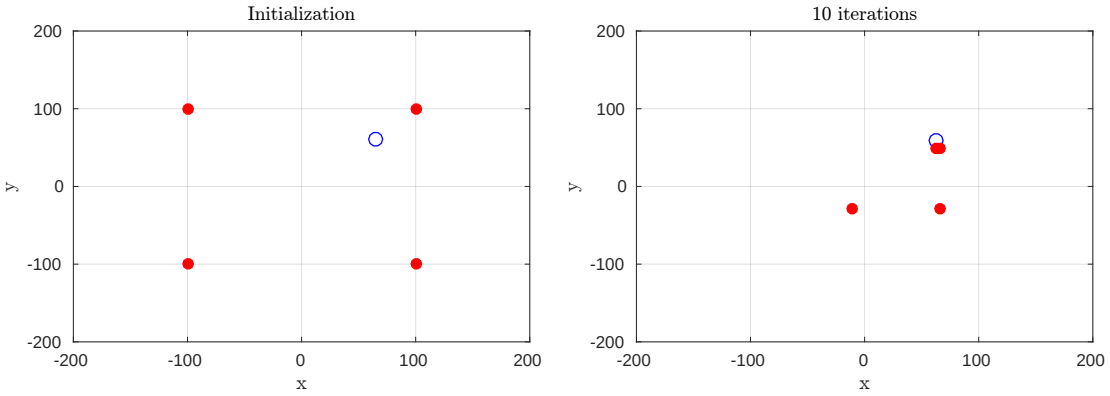


Figure 5.2. Convergence process of estimation to find number of deployed nodes. Using the average consensus algorithm we find the average ID od nodes $\overline{I}$, from which we can easily compute the overall number of deployed nodes.

of iterative algorithm. Goal of the nodes $v_1 - v_4$ is to be as near as possible to v_5, with only some small d distance, that $v_1 - v_4$ hold to prevent collision. In the simulation we had to significantly decrease the choice of α parameter, $\alpha = 0,1 \cdot \alpha^* = \frac{1}{10 \cdot \Delta}$, that controls the choice of Perron matrix $\boldsymbol{P}$, which bears the algorithm. This is justified by fact, that the iterations of algorithm are too fast and we should respect, that the followers are moving with finite velocity. (Consequently, in such a case the animation would not be interesting at all.)

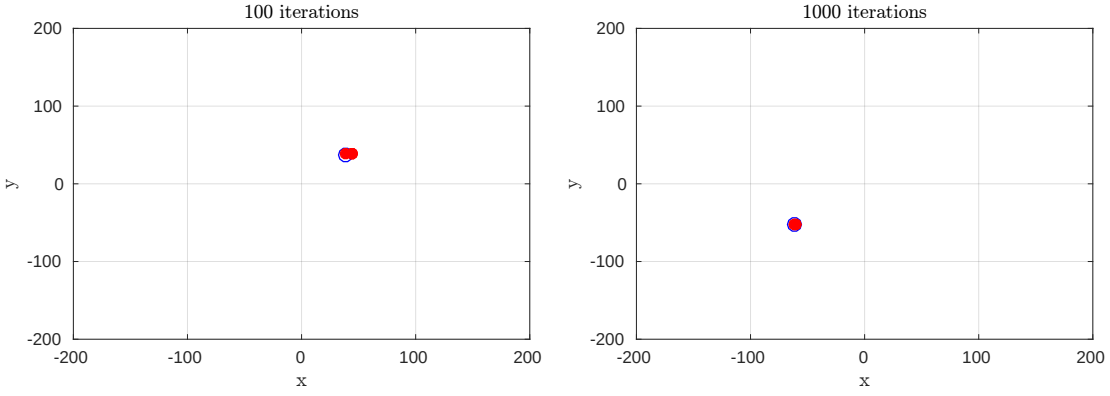
In Figures 5.3 and 5.4 we provide some screens from simulation, that is an output of provided Matlab script in Appendix A.4. Note that as a result $v_1 - v_4$ perfectly tracks v_5. Usage of Average consensus algorithm is quite common in automatic control of e.g. flights of drones, that should keep some given distance between each other; this is referred to as 'Multi-vehicle Formation Control' (or also sometimes less formally

flocking), and for more information and rich references we again recommend e.g. [22]. Average consensus algorithm may be so used to design feedback control system for such unmanned vehicles.



a) b)

Figure 5.3. Initialization and 10th iteration of Target tracking simulation.



a) b)

Figure 5.4. 100th and 1000th iteration of Target tracking simulation.

5.3 Distributed time synchronization of already communicating nodes

In this Example we want to show, that the Average consensus algorithm may be used by perfectly communicating nodes to synchronize to e.g. make a measurement at one moment. *We assume, that there has already been established communication between the nodes.* The problem, that solves this section might have been motivated e.g. like this: Imagine, our N sensors are supposed to measure some quantity, e.g. temperature. Because of typical WSN limitations, such as battery capacity, they are designed to shut down sensors, at moments during which they are not in use, to save power, and we want all the nodes to turn on the sensors and measure at same time instants. Usually, each sensor will keep its time as some big number, i.e. an integer in its memmory, that is being periodically incremented according to ticks of its internal oscillator. E.g. in time provided by GPS satellites the receivers obtain information in form of GPS-weeks and GPS-seconds since January 1980 [34].

A Matlab script, that we used to simulate the problem described above may be found in Appendix A.5. In the simulation we used 30 nodes of an average degree

4. In the Figure 5.5 we present a detail look to initial synchronization. Observation: the algorithm doesn't have any problem with fact, that the values in nodes in each iteration increase, nevertheless we can not speak about something like convergence. The variables, that are inputs of the algorithm still increase and the iterations of the algorithm hold the error in nodes reasonably small. The result, that we desire in this example is, that all the nodes have, within about 30 iterations, the same time base synchronized, which may be used e.g. to turn on the sensors in specific moments.

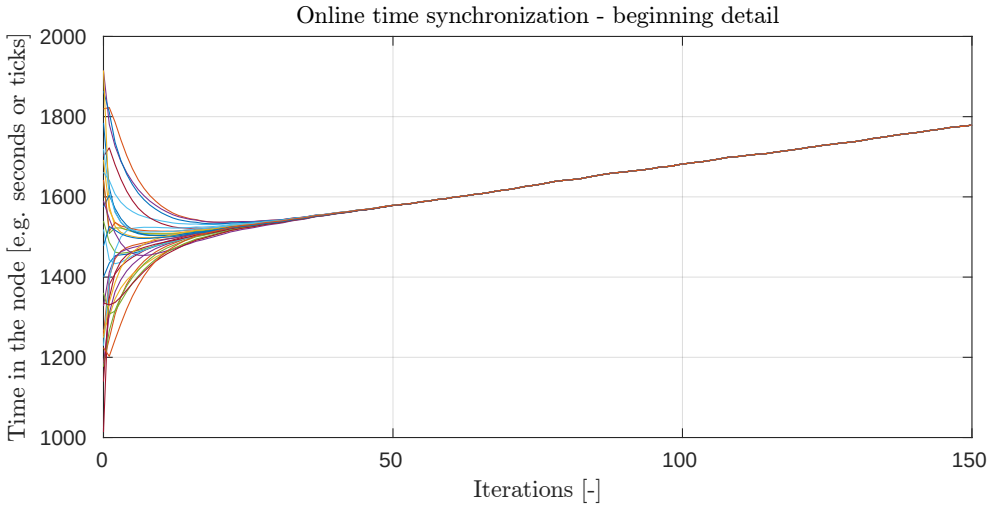


Figure 5.5. Detail view on result of the first 150 iterations of the simulation. Each node was initialized with time base $T = 1000 \pm \delta_{offset}$, where δ_{offset} is random number from Uniform distribution from interval $[0; 1000]$. We can see, that the convergence is reached in about 30 iterations.

5.4 Initial Distributed time base synchronization of nodes

The last example concerns again time base synchronization, but in a different manner, than previously. This time we want to find out, how to use the Average consensus algorithm to synchronize time base, that will serve to rule timing of the communication between nodes, e.g. motivated by purpose of usage a Time Division Multiple Access (TDMA), i.e. situation, when we want to ensure, that transmissions from different nodes will not overlap and cause collisions. Or, as another example, may be given by a situation, when the sensors measure position of a moving object and we want to determine its trajectory: to do so, it is naturally necessary, that the measured data are labeled also with some time stamp to preserve ability to calculate the trajectory.

So we are given N nodes, denoted as $i = 1, 2, ..., N$, and each equipped with its internal oscillator with a (possibly different) period T_i. To clarify the difference between this task and the previous one: in the previous example we expected, that time synchronization from this example was already performed and therefore the nodes could have been exchanging only the variables that symbolized some global time. *Now we do not assume, that there has already been established time-synchronized communication between nodes.* We need, only precise detection of impulses is sufficient, as explained

later. Meaning of the TDMA is explained on the following Figures 5.6 and 5.7, see the labels.

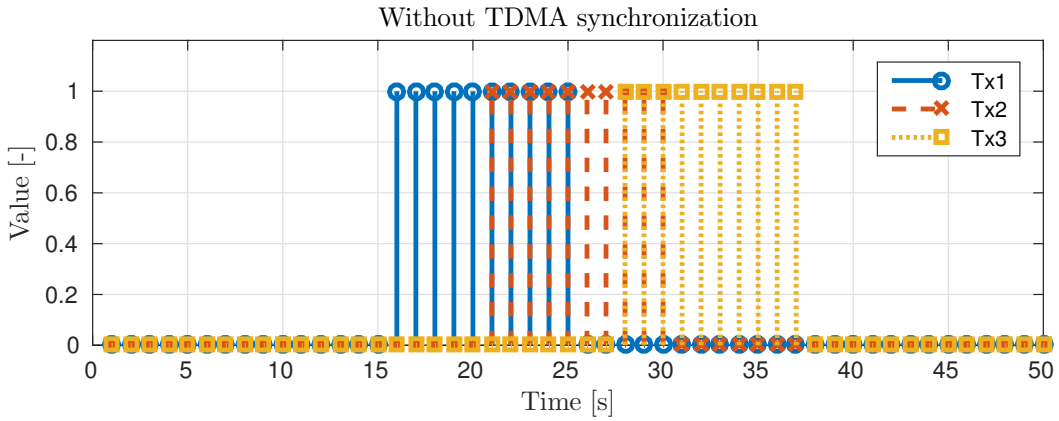


Figure 5.6. Without TDMA synchronization it happens, that the transmissions overlap and the received signal is therefore damaged. (Tx2 damages Tx1 and Tx3.)

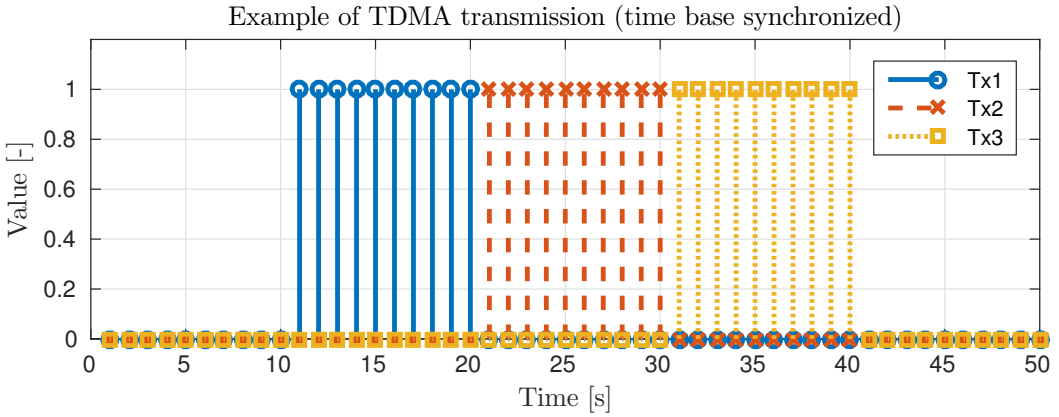


Figure 5.7. TDMA prevents the transmissions to overlap, because each node has designated time slots, when it is allowed to transmit.

A very nice summary of this time synchronization problematic may be found in [35]. We will now present here one possible approach, that may be found there. Solution may be described like this (see Figure 5.8): $i-$th node's internal oscillator ticks on frequency $1/T_i$. Next, e.g. at the end of each period, its antenna sends out an impuls that propagates towards the rest of nodes and reaches its neighbors. These impulses *reveal* the $i-$th node's time base to its neighbors. Simultaneously the $i-$th node receives these impulses from neighbors, so we assume possibility of implementation in full-duplex communication. For simplicity, we now assume, that distances between all nodes are approximately comparable and therefore we neglect time differences caused by different times of propagation of the impulses. Next block is called 'Time difference' block, whose inputs are the time delayed or advanced, impulses received from the neighbors and $i-$th node's own internal time base (see Figure 5.9). Output of this block is denoted as Δt_i and it is a weighted linear combination of these time differences, specified by the weights previously denoted as p_{ij} (i.e. elements of the Perron matrix $\boldsymbol{P}$). We can thus write down the corresponding update equation for this part of loop as:

$$\Delta t_i(n) = \sum_{i=1;i\neq j}^{N} p_{ij}(t_j(n) - t_i(n)). \tag{5.3}$$

Finally as α_i is denoted a filter, that operates over $\Delta t_i(n)$ to determine t_i. The situation is presented on the following Figures. Then we can write the following update Eq.

$$t_i(n+1) = t_i(n) + T_i + \alpha \sum_{i=1;i\neq j}^{N} p_{ij}(t_j(n) - t_i(n)), \tag{5.4}$$

which is very familiar. We can see, that our problem to synchronize time base of nodes using Average consensus algorithm may be solved using Phase-locked loop [35]. The T_i in Eq. (5.4) is an increment of Time in each iteration and the whole Eq. (5.4) should be seen to correspond with Figure 5.8.

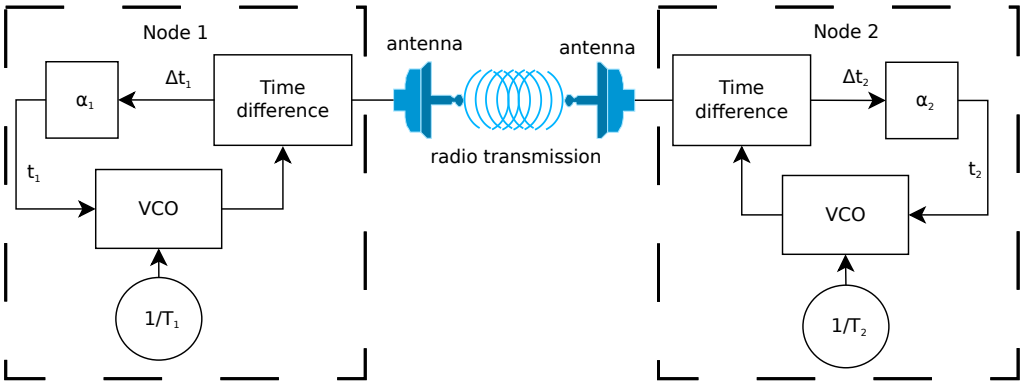


Figure 5.8. Scheme of 2 nodes with Phase-locked loop. Generally VCO in each node ticks with its frequency signed as $1/T_i$. Each node sends out an impulse at moments that the node's clock have e.g. zero value. We neglect the differences in propagating of these impulses (they propagate by speed of light). The Time difference block (TD) then provides input to the filter α_i, that calculates a weighted combination of the offsets and its output goes to the VCO. Also, note, that the input from TD block to antenna is scalar, however from antenna to TD is generally vector of the received neighboring node's impulses. This Figure is according to [35].

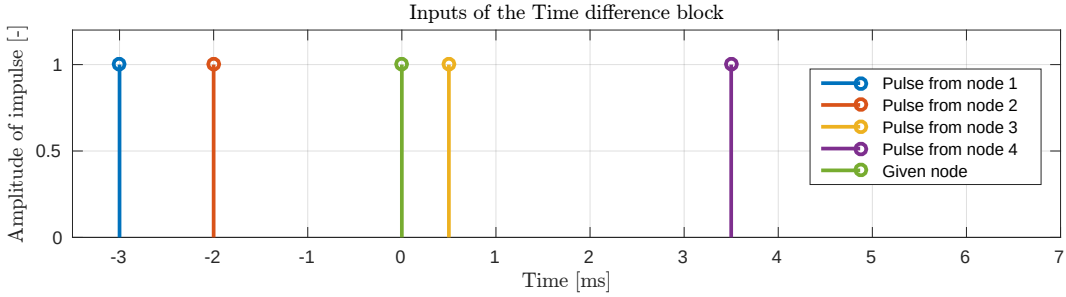


Figure 5.9. Inputs of the Time difference block. The impulses come in to the block with either time delay or advance. The green value located in 0 of horizontal axis corresponds to the value, that the nodes expects to be the correct time base. Asymptotically should the offsets decrease and be approximately zero once the time base is synchronized. Output of the block is Δt_i .

Using the update equation (5.4), we provide result of simulation of this approach.

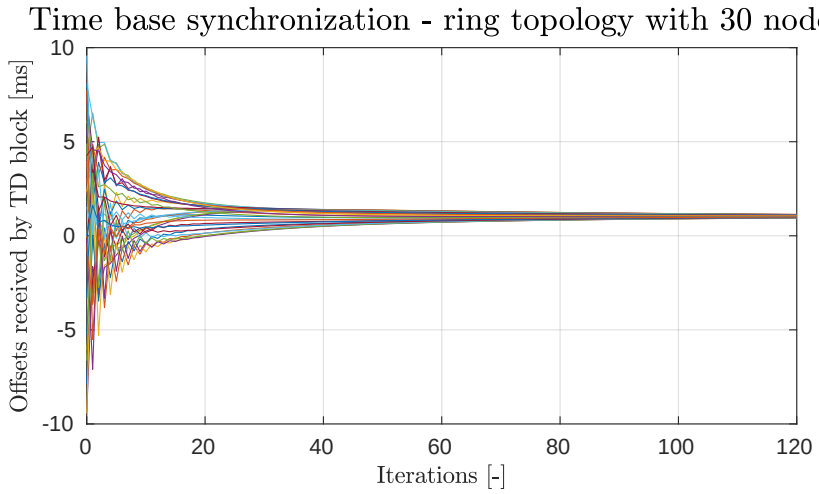


Figure 5.10. Time base synchronization: we use ring topology with 30 nodes and assume, that distances between all neighbors are comparable. Each node initializes the time base with random offset selected from uniform distribution $[-10; 10]$ measured in miliseconds. Each node receives in every iteration 2 impulses, that are shifted from 0 according to the offsets of its neighbors. The Figure shows how the Average consensus algorithm makes the nodes to change the offset w.r.t. the initialized values. After about 80 iterations of the algorithm we can say, that the nodes synchronized mutually their time base.

Chapter 6
Conclusion

Main goal of this Bachelor's thesis was to become acquainted with Linear average consensus algorithm on graph. We provided basic introduction to the Graph theory, including important terms that are common to describe the algorithm, with focus on the Laplacian matrix, i.e. its definition and basic properties. Next, we stated the Linear average consensus algorithm description, as it is common in related literature and also briefly overviewed the core idea concerning the convergence condition. This helped us to understand, why it appears advantageous to design the algorithm using the Laplacian matrix of the graph. We also briefly mentioned the problematic of noisy updates, which is important in wireless digital communication, and showed in particular example, that the algorithm scheme must be modified to preserve the convergence of algorithm. In this noisy-update example we, without proof, used Decreasing step size approach found in relevant mentioned literature. In the text are also mentioned the most common difficulties and pitfalls in context of estimation in wireless networks, including fact, that the mathematical description becomes in general cases quite complex. Finally, we used the knowledge to provide simple examples, implemented as Matlab scripts (in Appendix), that are commented in the last part of the thesis.

Further work on this topic could have focus on the specific approaches to deal with the problematic of noisy updates, which is necessary for ability to use this algorithm in wireless communication. The algorithm itself is general and may be used in many branches.

References

[1] *Seven Bridges of Königsberg*. 2001-.
https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg.

[2] Bogdan Giusca. *The problem of the Seven Bridges of Königsberg*. 2005.
https://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png.

[3] Eric W. Weisstein. *Spanning Tree*.
http://mathworld.wolfram.com/SpanningTree.html.

[4] *Network Time Protocol*.
http://www.ntp.org/ntpfaq/NTP-s-algo.htm.

[5] Eric W. Weisstein. *Adjacency Matrix*.
http://mathworld.wolfram.com/AdjacencyMatrix.html.

[6] Eric W. Weisstein. *Degree Matrix*.
http://mathworld.wolfram.com/DegreeMatrix.html.

[7] Eric W. Weisstein. *Incidence Matrix*.
http://mathworld.wolfram.com/IncidenceMatrix.html.

[8] Bojan Mohar. *The Laplacian spectrum of graphs*. In: *Graph Theory, Combinatorics, and Applications*. Wiley, 1991. 871–898.

[9] Miroslav Dont. *Maticová analýza*. In: Prague, Česká technika, 2011. 91–92. ISBN "978-8-001-04857-3".

[10] Anne Marsden. *Eigenvalues of the Laplacian and Their Relationship to the Connectedness of a Graph*. In: Chicago, 2013. 5 - 6.
http://math.uchicago.edu/~may/REU2013/REUPapers/Marsden.pdf.

[11] Vahid Liaghat. *Spectral Graph Theory and Algorithmic Applications*. 2015.
https://web.stanford.edu/class/msande337/scribe1.pdf.

[12] Eric W. Weisstein. *Cofactor*.
http://mathworld.wolfram.com/Cofactor.html.

[13] R. B. Bapat. *The Laplacian spectrum of a graph*. In: *The Mathematics student*. Dilli: Math Student, 1996. 214 - 223.

[14] M.W. Newman. *The Laplacian Spectrum of Graphs*. University of Manitoba, 2001.
https://mspace.lib.umanitoba.ca/bitstream/handle/1993/2048/MQ57564.pdf?sequence=1.

[15] U.Spagnolini. Distributed signal processing and synchronization, tutoria. 2013,

[16] Federica Garin, and Luca Schenato. *A Survey on Distributed Estimation and Control Applications Using Linear Consensus Algorithms*. In: *Networked Control Systems*. London: Springer London, 2010. 75–107. ISBN 978-0-85729-033-5.
http://dx.doi.org/10.1007/978-0-85729-033-5_3.

[17] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed Average Consensus with Least-mean-square Deviation. *J. Parallel Distrib. Comput.*. 2007, 67 (1), 33–46. DOI 10.1016/j.jpdc.2006.08.010.

[18] Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. 1 edition. Berlin: Springer-Verlag Berlin Heidelberg, 2013. ISBN "978-3-642-38123-2".

[19] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*. 2007, 95 (1), 215-233. DOI 10.1109/JPROC.2006.887293.

[20] Lin Xiao, and Stephen Boyd. Fast Linear Iterations for Distributed Averaging. *Systems and Control Letters*. 2003, 53 65–78.

[21] J. Ding, and A. Zhou. *Nonnegative Matrices, Positive Operators, and Applications*. 2009. ISBN 9789813107434. https://books.google.ie/books?id=FxU8DQAAQBAJ.

[22] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*. 2007, 95 (1), 215-233. DOI 10.1109/JPROC.2006.887293.

[23] Saber Jafarizadeh, and Abbas Jamalipour. *Weight Optimization for Distributed Average Consensus Algorithm in Symmetric, CCS and KCS Star Networks*. 2010. arXiv:1001.4278v3.

[24] C. Mosquera, R. Lopez-Valcarce, and S. K. Jayaweera. *Distributed estimation with noisy exchanges.* In: *2008 IEEE 9th Workshop on Signal Processing Advances in Wireless Communications*. 2008. 236-240.

[25] C. Mosquera, R. Lopez-Valcarce, and S. K. Jayaweera. Stepsize Sequence Design for Distributed Average Consensus. *IEEE Signal Processing Letters*. 2010, 17 (2), 169-172. DOI 10.1109/LSP.2009.2035373.

[26] B. Touri, and A. Nedic. *Distributed consensus over network with noisy links.* In: *2009 12th International Conference on Information Fusion*. 2009. 146-154.

[27] L. Pescosolido, S. Barbarossa, and G. Scutari. *Average consensus algorithms robust against channel noise.* In: *2008 IEEE 9th Workshop on Signal Processing Advances in Wireless Communications*. 2008. 261-265.

[28] *Spectra of complete graphs, stars, and rings*. 2016. https://www.johndcook.com/blog/2016/01/09/spectra-of-complete-graphs-stars-and-rings/.

[29] S. Kar, and J. M. F. Moura. Distributed Consensus Algorithms in Sensor Networks With Imperfect Communication: Link Failures and Channel Noise. *IEEE Transactions on Signal Processing*. 2009, 57 (1), 355-369. DOI 10.1109/TSP.2008.2007111.

[30] Soummya Kar, José M. F. Moura, and Kavita Ramanan. Distributed Parameter Estimation in Sensor Networks: Nonlinear Observation Models and Imperfect Communication. *CoRR*. 2008, abs/0809.0009

[31] B. Li, H. Leung, and C. Seneviratne. *Distributed consensus in noisy wireless sensor networks.* In: *2016 19th International Conference on Information Fusion (FUSION)*. 2016. 1356-1363.

[32] Cailian Chen, Shanying Zhu, Xinping Guan, and Xuemin Shen. *Wireless Sensor Networks : Distributed Consensus Estimation*. 2014 edition. Cham: Springer, 2015. ISBN 9783319123783;3319123785;.

[33] Eric W. Weisstein. *Kronecker Product*. http://mathworld.wolfram.com/KroneckerProduct.html.

[34] NPS. *Time Systems and Dates - GPS Time*. http://www.oc.nps.edu/oc2902w/gps/timsys.html.

[35] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz. Distributed synchronization in wireless networks. *IEEE Signal Processing Magazine*. 2008, 25 (5), 81-97. DOI 10.1109/MSP.2008.926661.

Appendix A
Matlab scripts

A.1 Matlab script: Average consensus algorithm with perfect communication

This is script *perfect_communication_example.m* used in Examples 3.5, 3.6 and 3.7 to present the run of algorithm. In this case we do not use the built-in matlab functions to compute desired values, only to check them.

```
% Run of average consensus algorithm with perfect communication
clear all;
s=[1 1 2 2 3 3 3 4 4 4 5 6 7 8 8 9 9 10  ];%source vertices
t=[2 3 4 5 7 10 5 5 6 7 6 7 5 1 2 10 7 8];%destination vertices
w=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]; %weights of edges
G = graph(s,t,w); %create graph G with parameters s, t, w
iterations = 30;
for i= 1:length(s) %A is adjacency matrix; A(i,j)=1 <=> exists edge (i,j)
   A(s(i),t(i))=1;
   A(t(i),s(i))=1;
end
checkA=isequal(A, adjacency(G)); %check adjacency matrix
M=0; %M is the highest degree of vertex
for i=1:size(A)
    sumRadek=sum(A(i,:))
    if sumRadek>=M
        M=sumRadek;
    end
end
for i = 1:size(A)
    D(i,i)=sum(A(i,:))%D is degree matrix
end
L=D-A; %Laplacian matrix
checkLaplacian=isequal(laplacian(G),L); %check Laplacian
for i= 1:length(s) %Incidence matrix
    E(s(i),i)=1;
    E(t(i),i)=-1;
end
L2=E*transpose(E); %another way to compute Laplacian
checkLaplacian2=isequal(laplacian(G),L2); %check Laplacian
I=L*ones(max(s)); %check that Laplacian rows sum up to 1
D = eig(L,'matrix');%diagonal matrix of eigenvalues of laplacian
L_eig=eig(L);
alpha=2/(L_eig(2)+L_eig(max(s)))
```

```
for i = 1:max(s)
node_initial_value(i)=i; %initialization of values to average
%running_value(i+1,j)=node_value(j);
running_value(1,i)=node_initial_value(i);
running_value_Error(1,i)=-node_initial_value(i)+mean(node_initial_value);
end
for i= 1:max(s)
final_value(i)=mean(node_initial_value); %expected average value
end
node_value=node_initial_value; %inicialization of nodes values
running_value_Error(1,:)=final_value-node_initial_value;
for i=1:iterations+1
discrete_time(i)= i-1;
end
for i = 1:iterations;
    IT=eye([max(s) max(s)])-alpha*L; %iteratation Perron matrix
    node_value=node_value;
    node_value= node_value* IT;
    for j=1:max(s)
            running_value(i+1,j)=node_value(j);
            running_value_Error(i+1,j)=final_value(j)-node_value(j);
    %node_value(3)=3; % uncomment for convergence to v_3 initial value
    end
end

figure;
plot(discrete_time, running_value(:,:));
ttl1=title( 'Run of algorithm '  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Value [-]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;
figure;
grid on
plot(discrete_time, running_value_Error(:,:));
title('shrinking of error');
grid on
figure;
plot(G); %plot graph G
```

A.2 Matlab script: Average consensus algorithm with noisy observation and updates using decreasing step size

This is script *script_noisy_updates_gamma.m* used in Example to present simple way to preserve convergence in case of noisy updates.

```
% Run of average consensus algorithm with noisy
% updates using descending step size scheme gamma=a/(1+b)^c
%% graph definition
clear all; clc;
variance_noise=0.1; %variance of the gaussian noise added to the updates
iterations =5000; %number of iterations of the algorithm
nodes=60; %number of graph nodes
G=graph(bucky); % using matlab default graph bucky
% i.e. 60 nodes with degree 3
%% variables initialization
equiv_noise_vector=zeros(1,nodes);
x_0=zeros(1,nodes);
MSE_ave=zeros(zeros(1,iterations+1));
VAR=zeros(zeros(1,iterations+1));
MSE_act=zeros(zeros(1,iterations+1));
diag_D=zeros(nodes,nodes);
%plot the used graph
figure;
plot(G)
deg=degree(G);
for p=1:nodes
   diag_D(p,p)=deg(p);
end
%observation initialization const 10 + wgn with variance 1
for i=1:nodes     %initialize measurement
    x_0(i)=(10+wgn(1,1,log10(10*variance_noise)));
end
%% variables definition
x_est_run(1,:) = x_0;
MSE_act(1)= sum((x_0-mean(x_0)).^2);
MSE_ave(1)=MSE_act(1);
VAR(1)=var(x_0);
noise_matrix=zeros(nodes);
%Laplacian using library functions
A=adjacency(G); %adjacency matrix
L=laplacian(G); %laplacian matrix
eig_L=eig(L);%eigenvalues of Laplacian
gamma=2/(eig_L(2)+eig_L(nodes));%the best possible coefficient
mean_0=mean(x_0);%value to converge to
%% Iterations of the algorithm
for i=1:iterations
   gamma=1/(i+42)^0.75; %selected descending step size
   P=eye(nodes)- gamma* L; %Perron matrix

   for j=1:nodes %computation of the nise affecting updates exchange
      noise_matrix(j,:)=wgn(nodes,1,10*log10(variance_noise)) ;
```

```
      noise_matrix(j,j)=0 ;
   end
    equiv_noise_matrix= P*noise_matrix;
   for j=1:nodes
     equiv_noise_vector(j)= equiv_noise_matrix(j,j);
   end
   x_est_run(i+1,:)=  P*x_est_run(i,:)'+equiv_noise_vector';
end
%calculation of the run statistics
for i=1:iterations
  MSE_ave(i+1)= sum((x_est_run(i+1,:)-mean(x_0)).^2)/nodes;
  MSE_act(i+1)= sum((x_est_run(i+1,:)-mean(x_est_run(i+1,:))).^2)/nodes;
  VAR(i+1)= var(x_est_run(i+1,:));
end
%%  Plot results
figure;
plot(0:iterations, x_est_run(:,:));
ttl1=title( 'Values in nodes'  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Value [-]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;

figure;
loglog(0:iterations, MSE_ave(:)   );
ttl1=title( 'Mean square error w.r.t. initial average '  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('MSE [-]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;

figure;
loglog(0:iterations, VAR(:));
ttl1=title( 'Variance '  );
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Variance [-]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;
```

A.3 Matlab script: Estimation of nodes number in given area

```
% Script to find number of N sensors based on their ID
%% Initialization
```

```
clear all;
nodes = 99;
iterations = 30;
G = WattsStrogatz(nodes,4,0.5); %generates random graph
L=full(laplacian(G));
degreeMax= max(max(L));
alpha=1/degreeMax;
P=eye(nodes)- alpha* L; %Perron matrix

for i = 1:nodes
x_running(1,i)=i; %initialization vertex with its ID
end

%% iterations of the algorithm
for i = 1:iterations
    x_running(1+i,:)=P*x_running(i,:)';
end

%% plot results
figure;
plot(0:iterations, x_running(:,:));
ttl1=title( 'Average ID number estimation'  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Node ID [-]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;

figure;
h = plot(G);
c = h.EdgeColor;
h.EdgeColor = 'k';
h.LineWidth=1
h.MarkerSize=8;
h.EdgeAlpha=1;
labelnode(h,1:nodes,' ')
```

A.4 Matlab script: Tracking of dynamic target

```
%Tracking of dynamic target script
clear all;
nodes = 5; % 4 followers and 1 target
d=0.01;%distance between followers
r=200; % range of plot axis (xy)
iterations = 1000;
s=[1 1 1 1 2 2 2 3 3 4 ];%graph definition
t=[5 2 3 4 5 3 4 5 4 5];
v1_pos=[100,100]; %initialize followers position
v2_pos=[-100,100];
v3_pos=[100,-100];
```

```
v4_pos=[-100,-100];
v5_init_pos=[100*rand, 100*rand]; %target position is random
G =graph(s,t)%generates  graph
L=full(laplacian(G)); %matlab function for Laplacian
degreeMax= max(max(L));%generate weight that control dynamics
alpha=0.08*1/degreeMax; %coefficient 0.08 found experimentaly
P=eye(nodes)- alpha* L; %Perron matrix
x_pos_vect(1,:)=[v1_pos(1) v2_pos(1) v3_pos(1) v4_pos(1) v5_init_pos(1)];
y_pos_vect(1,:)=[v1_pos(2) v2_pos(2) v3_pos(2) v4_pos(2) v5_init_pos(2)];
v5_pos_x(1)=v5_init_pos(1);
v5_pos_y(1)=v5_init_pos(2);
for i = 1:iterations
 x_pos_vect(i+1,:)=P* x_pos_vect(i,:)';
y_position_vector(i+1,:)=P*y_position_vector(i,:)';
if i<0.5*iterations %for first half of iterations the target moves
y_position_vector(i+1,5)=y_position_vector(i,5)+0.5*rand*(-1^(i));
 x_pos_vect(i+1,5)= x_pos_vect(i,5)+0.5*rand*(-1^(i));
end
% distance between followers
y_position_vector(i+1,1)=y_position_vector(i+1,1)+d;
x_pos_vect(i+1,1)= x_pos_vect(i+1,1)+d;
y_position_vector(i+1,2)=y_position_vector(i+1,1)+d;
x_pos_vect(i+1,2)= x_pos_vect(i+1,5)-d;
y_position_vector(i+1,3)=y_position_vector(i+1,3)-d;
x_pos_vect(i+1,3)= x_pos_vect(i+1,3)+d;
y_position_vector(i+1,4)=y_position_vector(i+1,4)-d;
x_pos_vect(i+1,4)= x_pos_vect(i+1,4)-d;

end
%% plot the animation of mooving point
figure('Position', [100, 100, 450,300])
for i=1:iterations
hold off;
plot(x_pos_vect(i,5),y_pos_vect(i,5),'ob','MarkerSize',8)
hold on;
plot(x_pos_vect(i,1),y_pos_vect(i,1),'or','MarkerFaceColor','r')
hold on;
plot(x_pos_vect(i,2),y_pos_vect(i,2),'or',''MarkerFaceColor','r')
hold on;
plot(x_pos_vect(i,3),y_pos_vect(i,3),'or','MarkerFaceColor','r')
hold on;
plot(x_pos_vect(i,4),y_pos_vect(i,4),'or','MarkerFaceColor','r')
hold on;
grid;
 axis([-r r -r r])
 pause(0.001)
end
```

A.5 Matlab script: (Measurement) Time base synchronization

This is script used to simulate synchronization of Time base, meant to e.g. synchronize the sensors to do the measurement at one common instant. See Section 5.3.

```
% Time base synchronization script
%% Initialization
clear all;
nodes = 30; %number of nodes of graph
iterations = 150; %number of iterations of the algorithm
G = WattsStrogatz(nodes,2,1); %generates random graph
L=full(laplacian(G)); %Laplacian using Matlab function
degreeMax= max(max(L)); %find the greatest degree of graph
alpha=1/degreeMax; %choose parameter to Define Perron matrix
P=eye(nodes)- alpha* L; %create Perron matrix
commonTimeBase=1000; % some big number used to initialize time base
% e.g. in gps time used number of weeks and secs since January'80
for i = 1:nodes %initialize the vertices time base + some random offset
time_running(1,i)=commonTimeBase+1000*rand; %init of time base + offset
end
for i = 1:iterations
    %average consensus iterations
    time_running(1+i,:)=P*time_running(i,:)';
    %each vertex adds to the computed value some addition,...
    %... representing ticks of its internal oscilator
    time_running(1+i,:)= time_running(1+i,:)+1+2*rand;
    % between 300th and 350th we simulate an outage of 10 sensors
    if (i>300)&(i<500)
    time_running(1+i,10:20)= time_running(200,20:30)   ;
    end
end
%% plot results
figure;
plot(0:iterations, time_running(:,:));
ttl1=title( 'Online time synchronization'  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Time in the node [e.g. seconds,ticks]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;
%
figure;
h = plot(G);
c = h.EdgeColor;
h.EdgeColor = 'k';
h.LineWidth=1
h.MarkerSize=8;
h.EdgeAlpha=1;
labelnode(h,1:nodes,' ')
```

A.6 Matlab script: Initial time base synchronization

```
% Initial time base synchronization
%% Initialization
clear all;
nodes = 30; %number of nodes of graph
iterations = 120; %number of iterations of the algorithm
s=(1:1:30); %source and destination vertices of ring graph
t=[30,1:1:29];
G = graph(s,t); %create graph
L=full(laplacian(G)); %Laplacian using Matlab function
Ti=1; %period of the i-th's clock; all nodes have the same inner oscilla-
tor
a=-10;b=10;
offsets= a + (b-a).*rand(nodes,1);%define initial offsets
time_base(1,:)=offsets;%detected by the TD block in PLL
L_eig=eig(L); %eigenvector of Laplacian
alpha=2/(0.5+L_eig(2)+L_eig(max(s)));%choose parameter to Define Perron
matrix
P=eye(nodes)- alpha * L; %create Perron matrix

for i = 1:iterations
    %averaging consensus iterations
    time_base(1+i,:)=P*time_base(i,:)';
end
%% plot results
figure;
plot(0:iterations, time_base(:,:));
ttl1=title( 'Time base synchronization - ring topology with 30 nodes '  )
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Iterations [-]');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('Offsets received by TD block [ms]');
ylbl1=set(ylbl1,'Interpreter','latex');
grid;
```

Appendix B
Content of CD