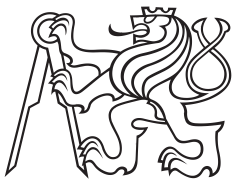
Bachelor's Thesis



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

November 2016
kolarj39@fel.cvut.cz

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kolář** Jméno: **Jakub** Osobní číslo: **434776**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Otevřené elektronické systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Distributed Signal Processing in Radio Communication Networks

Název bakalářské práce anglicky:

Distributed Signal Processing in Radio Communication Networks

Pokyny pro vypracování:

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.

Seznam doporučené literatury:

[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, tutoria 2013

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. Jan Sýkora CSc., katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **27.01.2017** Termín odevzdání bakalářské práce: **26.05.2017**

Platnost zadání bakalářské práce: **27.06.2018**

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta

Acknowledgement / Declaration

TBD.

TBD. In Prague, 26. 5. 2017

.....

Abstrakt / Abstract

TBD.

Klíčová slova: Graf, Laplacián grafu, Lineární konsenzus, Aktualizace s rušením, Odhad parametru

TBD

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

Contents /

/ Figures

Chapter 1
Introduction

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 informations, 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 neighbours 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 informations. Which means, every node can get know also the value that measures its neighbour. This ought to be only a very simple illustration how to transfer a physicaly realisable 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 neighbours 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.

1.1 Outline

A Graph theory provides a very elegant way to represent informations 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 decription of the average consensus algorithm and show some examples with graphically illustrated solution.

In 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 synchronisation or carrier frequency synchronisation. In a very simple case, we can also show how do the nonidealities change the result of algorithm (additive zero-mean noise).

Chapter 2
Graph theory

2.1 Motivation

It is commonly well known [1], that the elements of Graph theory were set by a mathematician Leonhard Euler in 1736. He solved a problem called Seven Bridges of Königsberg.

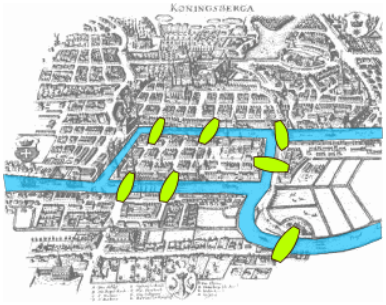


Figure 2.1. View on city Konigsberg with marked bridges [2] .

The problem was formulated like this: River Pregole flows through the city and creates two islands in there. These two islands are connected with the rest of the city by seven bridges (see figure above). The question was, whether it is possible to take a walk through the city in such a way, to pass each bridge exactly once. Euler described this problem as a graph, where the edges represented the seven bridges, and the vertices were the separated parts of the city.

Euler proved, that in this case, it is impossible to pass each bridge only once and showed, that the Eulerian trial (i.e. a trial, that contents every edge exactly once) exists if and only if every vertex of the graph is of an even degree.

Nowadays, a very modern and interesting example of usage of the Graph theory is visualisation 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 paramaters, such as a number of intermediate devices (RIP), round-trip delay or a bandwidth of the connectivity (OSPF). 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 eges with a given cost.)

We also briefly mention the term of Spanning trees. 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. Later, in a 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 synchronised time between server computers. Few of the servers are connected to the reference clocks and next, to them hiearchically connected servers, are averaging neighbour's time to obtain final value of time, that they use and provide to next users.

2.2 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.

2.2.1 Undirected graph

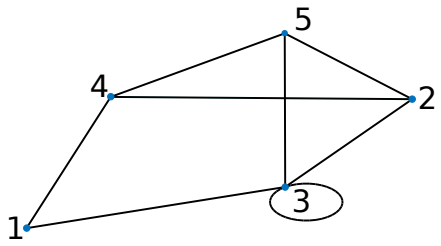


Figure 2.2. Example of a simple undirected graph to demonstrate basic definitions.

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 neigbours $\mathcal{N}_i$ 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 $d_i = |\mathcal{N}_i|$.

2.2.2 Directed graph

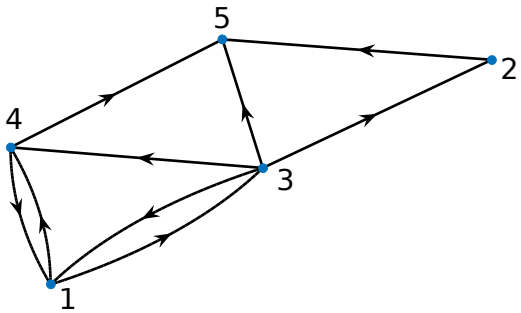


Figure 2.3. Example of a directed graph.

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 neighbours available via oriented edges, $d_i^{IN} = |\mathcal{N}_i|$. Drawing the figure, we distinguish the orientation of edges with arrows.

2.2.3 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:

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

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.

2.2.4 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 nondiagonal elements are equal to 0. 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 $\Delta = \max_i(d_i)$.

2.2.5 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:

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

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 typicaly smaller matrix.

2.3 Laplacian matrix

The structure of folowing section about Laplacian is based mainly on [3]. Supplementary references are added at corresponding paragraphs.

2.3.1 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,

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

Matrix $\boldsymbol{L}(G)$ for a graph with N vertices is $\boldsymbol{L}(G) \in \mathbb{R}^{N \times N}$. From definitions of $\boldsymbol{D}(G)$ and $\boldsymbol{A}(G)$ implies, that loops in graph have no influence on $\boldsymbol{L}(G)$.

To make hold some important results from Linear algebra and Matrix analysis, we will next consider only undirected and loopless graphs. Which means, that the coresponding Adjacency matrix will be symmetric. Having symmetric $\boldsymbol{A}(G)$, the coresponding Laplacian matrix will be also a symmetric matrix.

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

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

Next, we mark $\mu(G, x)$ the characteristical polynom of $\boldsymbol{L}(G)$ defined as

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

Roots of this characteristical polynom 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 $\boldsymbol{L}(G)$.

2.3.2 Basic properties

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

Proof: For example [4] , 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 [5]. 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. [6]

Proof: Let λ be an eigenvalue and v its coresponding eigenvector. Then

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

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

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

We can come to the positive semidefiniteness also using the following quadratic form:

$$\langle \boldsymbol{L}x, x\rangle = \sum_{(u,v)\in E(G)} (x_u - x_v)^2, \tag{2.13}$$

which results will be always non-negative.

2.3.3 Bounds for eigenvalues

Theorem 2.4. *Gershgorin circle theorem* [4]. Consider matrix $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ and $i = 1, 2, ..., N$. Let's denote

$$r_i = \sum_{j=1;\ i\neq j}^{N} = |a_{ij}|, \quad K_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \leq r_i\}. \tag{2.14}$$

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:

$$|\lambda - a_{kk}| = \left|\frac{\sum_{j=1;j\neq k}^{N} a_{kj}x_j}{x_k}\right| \leq \sum_{j=1;j\neq k}^{N}\left|\frac{a_{kj}x_j}{x_k}\right| \leq \sum_{j=1;j\neq k}^{N}|a_{kj}| = r_k. \tag{2.15}$$

Example 2.5. To present Gershgorin theorem practicaly, consider the graph below:

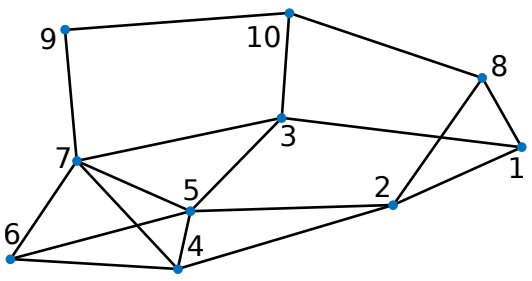


Figure 2.4. Graph to present Gershgorin theorem.

With its Laplacian matrix:

$$\boldsymbol{L} = \begin{pmatrix} 3 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 4 & 0 & -1 & -1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 4 & 0 & -1 & 0 & -1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 5 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & -1 & 5 & 0 & -1 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{pmatrix}. \tag{2.16}$$

And a characteristical 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 $\mu(G,x)$.

Nummerically solving $\mu(G,x) = 0$ we obtain the folowing 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.

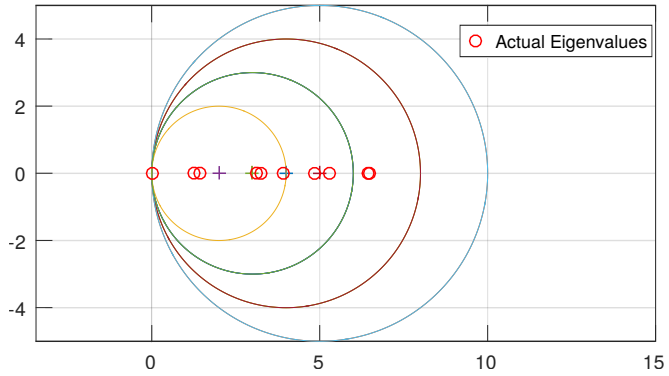


Figure 2.5. Plot of Eigenvalues and according Gershgorin circles.

Theorem 2.6. Let G be a graph with N vertices [3]. Then holds:

- $$\lambda_2 \leq \frac{N}{N-1} \min_i \{d(v_i) | v_i \in V(G)\}, \tag{2.17}$$
- $$\lambda_N \leq \max_i \{d(v_i) + d(v_j) |\, (v_i, v_j) \in E(G)\}, \tag{2.18}$$
- If G is a simple graph, then
$$\lambda_N \leq N, \tag{2.19}$$
- $$\sum_{m=1}^{N} \lambda_m = 2|E(G)| = \sum_{v_i} d(v_i), \tag{2.20}$$
- $$\lambda_N \geq \frac{N}{N-1} \max_i \{d(v_i) | v_i \in V(G)\}. \tag{2.21}$$

These may be found very usefull 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 polynoms with higher degree than 5. Using these bounds, we know where the roots must be, respective where they can not be.

2.3.4 Matrix tree theorem

$\boldsymbol{L}(G)$ may be also refferred to as Kirchhoff matrix due to the following theorem. 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. It is clear, that a spanning tree may be found only for connected graphs.

An (i, j)-cofactor of a matrix is a determinant of a submatrix formed by deleting the i-th row and the j-th column.

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 [7].

Proof: Ommited. Is based on decomposing the Laplacian matrix into product of Incidednce matrix and its transpose and then usage of Cauchy-Binet formula.

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

2.3.5 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 , $\forall i, j \in \{1, 2, ..., N\}$.

Eigen value λ_2 is also called *graph connectivity* [3]. This eigenvalue is probably the most important from the whole spectrum. Holds, that $\lambda_2 > 0$ if and only if the graph is connected. Moreover, the multiplicity of 0 as an eigenvalue of $\boldsymbol{L}(G)$ is the number of connected components.

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 interpratable, in context of the connectivity term, is the following one. Let's consider graph G with N vertices and diameter $diam(G)$. Then holds [3]:

$$diam(G) \geq \frac{4}{N\lambda_2}. \tag{2.22}$$

2.3.6 Operations with disjoint graphs

Very detailed reading about this part of Laplacian topic may be found beside in [3] also in [8]. 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 matrix of a graph consisting of k mutually disjoint sets of vertices will have block diagonal form obtained from matrices $\boldsymbol{L}(G_1), \boldsymbol{L}(G_2), ..., \boldsymbol{L}(G_k)$.

Theorem 2.9. Let G be a graph created as a union of disjoint graphs $G_1, G_2, ..., G_k$. Then holds [3]:

$$\mu(G, x) = \prod_{i=1}^{k} \mu_i(G_i, x). \tag{2.23}$$

Example 2.10.

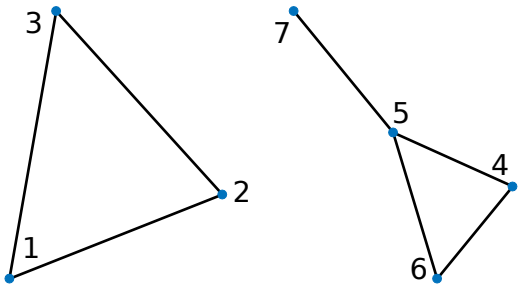


Figure 2.6. Example of a graph with two disconnected components.

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

Laplacian matrix of the whole graph reads:

$$\boldsymbol{L}(G) = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}. \tag{2.24}$$

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

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

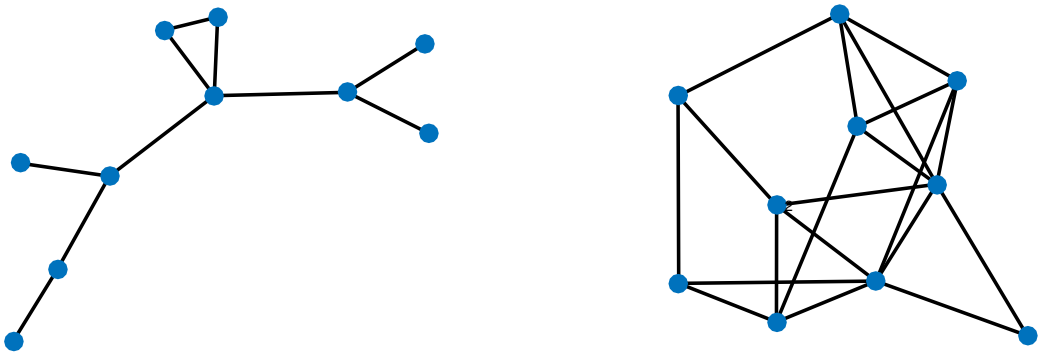


Figure 2.7. Another demonstration of Connectivity eigenvalue meaning.

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.

2.3.7 Laplacian matrix - notes

To demonstrate relation between Laplacian matrix and continuous Laplacian operator Δ, let's consider a differential equation

$$\Delta z(x, y) + \lambda \Delta z(x, y) = 0, \tag{2.25}$$

with initial condition $z(x, y) = 0$ on a simple closed curve Γ in xy-plane. Solution of this task is determined by an infinite sequence of eigenvalues $\lambda_1 \leq \lambda_2 \leq ....$ An aproximation of this solution may be found by placing a grid over the area in plane curved by Γ. The matrix of this finite, discretized problem, comes to be the Laplacian matrix [7].

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 to, for $t \to \infty$, using local communication and computation, in all N vertices of the graph, obtain an average value of all these initial values. Based on a matrix-like description of graph G, our goal will be to construct matrix $\boldsymbol{P}$, whose components $p_{i,j}$ will in fact bear this averaging algorithm, in a formalism of iterative matrix multiplication [9].

In this chapter, subject of our interest will be a linear, discrete-time consensus algorithm. A detailed description of the folowing is in [10], [11] 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 school book [12].

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 sattelites).
- 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

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

where $x(t) = (x_1(t), x_2(t), ..., x_N(t))^T \in \mathbb{R}^N$ and for all values of t, $\boldsymbol{P}(t) \in \mathbb{R}^{N \times N}$ is a *stochastic matrix*, i.e. $p_{i,j}(t) \geq 0$ and $\sum_{j=1}^{N} p_{i,j} = 1, \forall i, j \in 1, 2, ..., N$. Meaning, that all values in each row sum up to 1. The $p_{i,j}$ components are also often reffered to as *weights* [11].

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

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

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 $\sum_{j=1}^{N} p_{i,j} = 1, \forall j \in 1, 2, ..., N$. 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 [10]. For simplicity, we will firtsly 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 $p_{i,j} = 0$ for $j \notin \mathcal{N}_i$ (i.e. i-th node is not in a set of neighbours 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 [13].

Theorem 3.1. Perron-Frobenius theorem [13]. Let $\boldsymbol{P}$ be a primitive non-negative matrix with left and right eigenvectors w and v, respectively, satisfying $\boldsymbol{P}v = v$, $w^T\boldsymbol{P} = w^T$ and $v^Tw = 1$. Then

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

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 *averaging*. 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 $z \in \mathbb{R}^N$, $\overline{z} = \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T z$, we obtain a column vector $\overline{z}$ with all components equal to the average of all N components of the z vector [11].

What we ask about the algorithm is

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

which is for arbitrary vector $x(0)$ 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 *averaging* 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 realisable 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 [14].

3.4 Heuristics based on the Laplacian matrix

Basis material for following section comes mainly from [14].

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, due to his pioneering work in last century [15].

Such a matrix in fact evaluates each edge 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 substraction 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.

The elements of $\boldsymbol{P}$ are

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

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. :

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

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 characteristical 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 immediatly 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. exists the limit.

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

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

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

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 [14]

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

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

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

This is the best possible choice based on the Laplacian matrix. However, very useful is to select the coefficient as stated [16]

$$\alpha_\Delta = \frac{1}{\Delta}, \tag{3.28}$$

which choice depends only on the maximum degree in the graph, asssuming 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, typicaly 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 assymptotical convergence might be found e.g. in [16].

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. It might be clearly seen, 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. Last experiment 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.

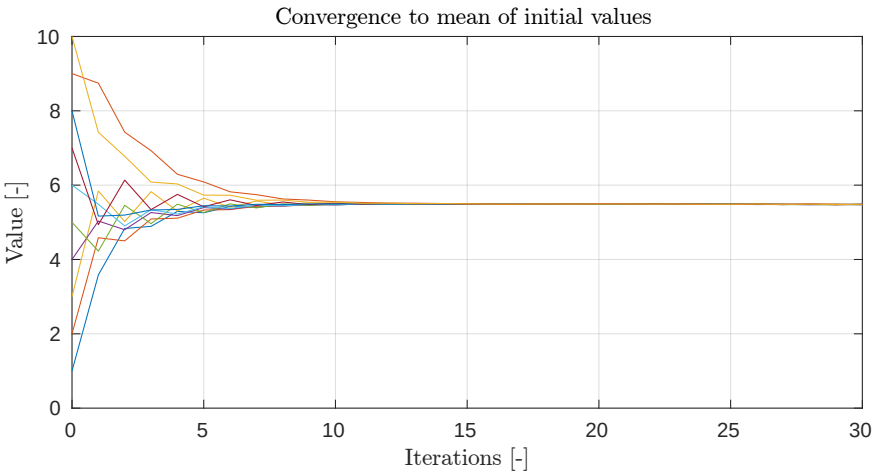


Figure 3.1. Averaging consensus algorithm in graph from Figure 2.2.

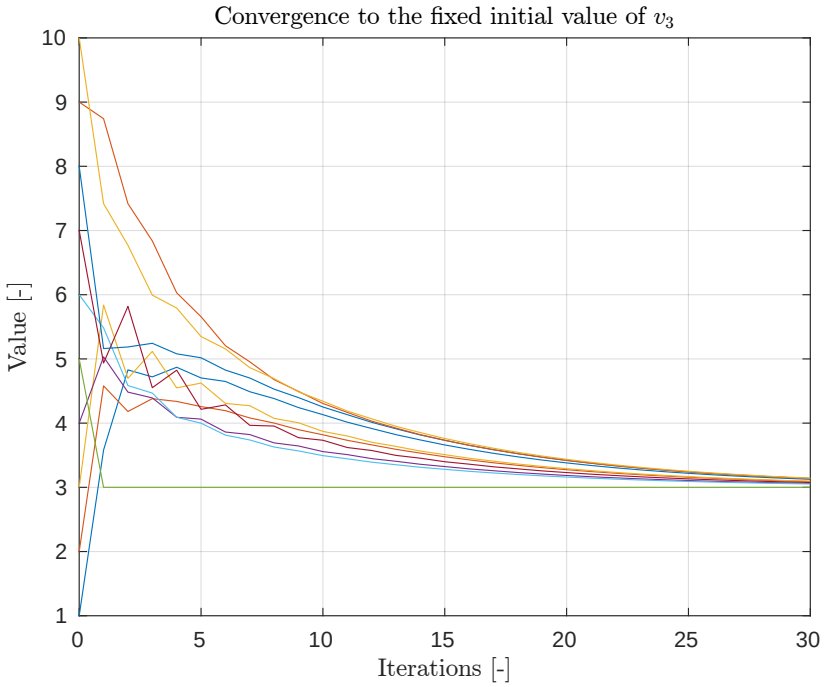


Figure 3.2. Averaging consensus algorithm in graph from Figure 2.2 with fixed value in $v_3, x_{v_3}(t) = 3$.

3.4.1 The Metropolis-Hastings weighting method

As mentioned before, the choice to construct $\boldsymbol{P}$ is not unique. Altough the Laplacian based approach is in the related basic literature probably the most common, we will briefly give some other satisfactory examples. For illustration we provide only one example from [17].

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

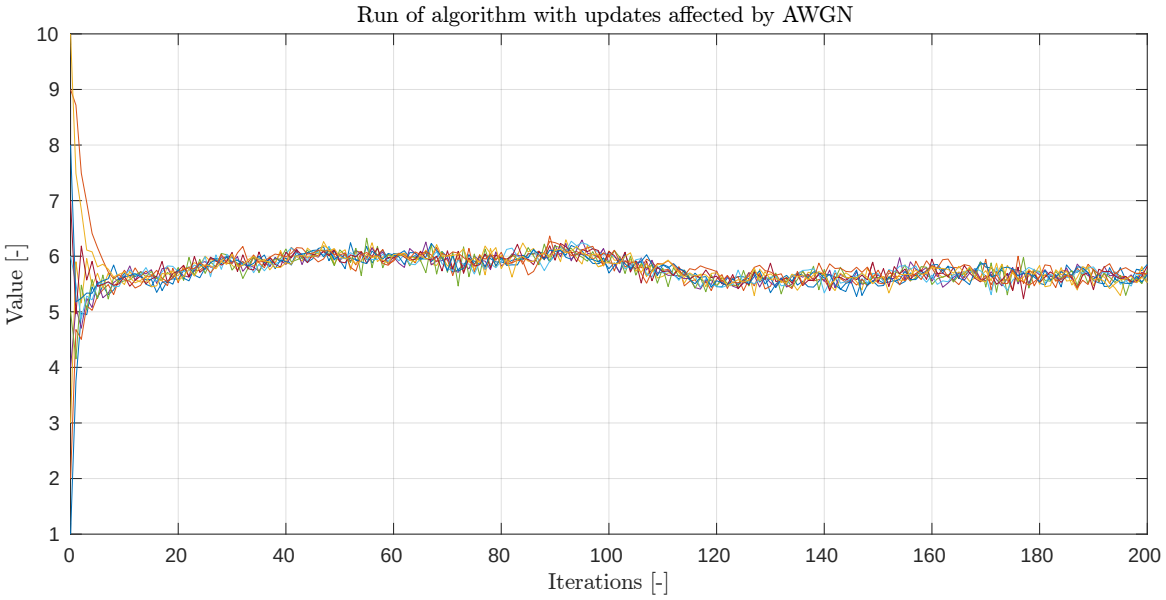


Figure 3.3. Averaging consensus algorithm in graph from Figure 2.2 with noisy updates.

$$p_{i,j}^{MH} = \begin{cases} \frac{1}{1+\max(d_i,\ d_j)} & \text{for } j \in \mathcal{N}_i,\ i \neq j, \\ 1 - \sum_{j \in \mathcal{N}_i} p_{i,j} & \text{if } i = j, \\ 0 & \text{otherwise [17].} \end{cases} \tag{3.29}$$

3.5 Average consensus algorithm with additive noise

Source for this section is mainly [11].

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 $w(t) \in \mathbb{R}^N = (w_1(t), w_2(t), ..., w_N(t))$. In detail, the $w_i(t)$ is a random variable with zero mean and unit variance. Mathematically formulated, the averaging consensus algorithm affected by additive noise will be described as

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

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

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

So using the opearator $\mathbb{E}[.]$, 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

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

that provides an average value of a vector $x(t)$ (i.e. a scalar). Then

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

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

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

and

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

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 $\boldsymbol{P}$. 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.

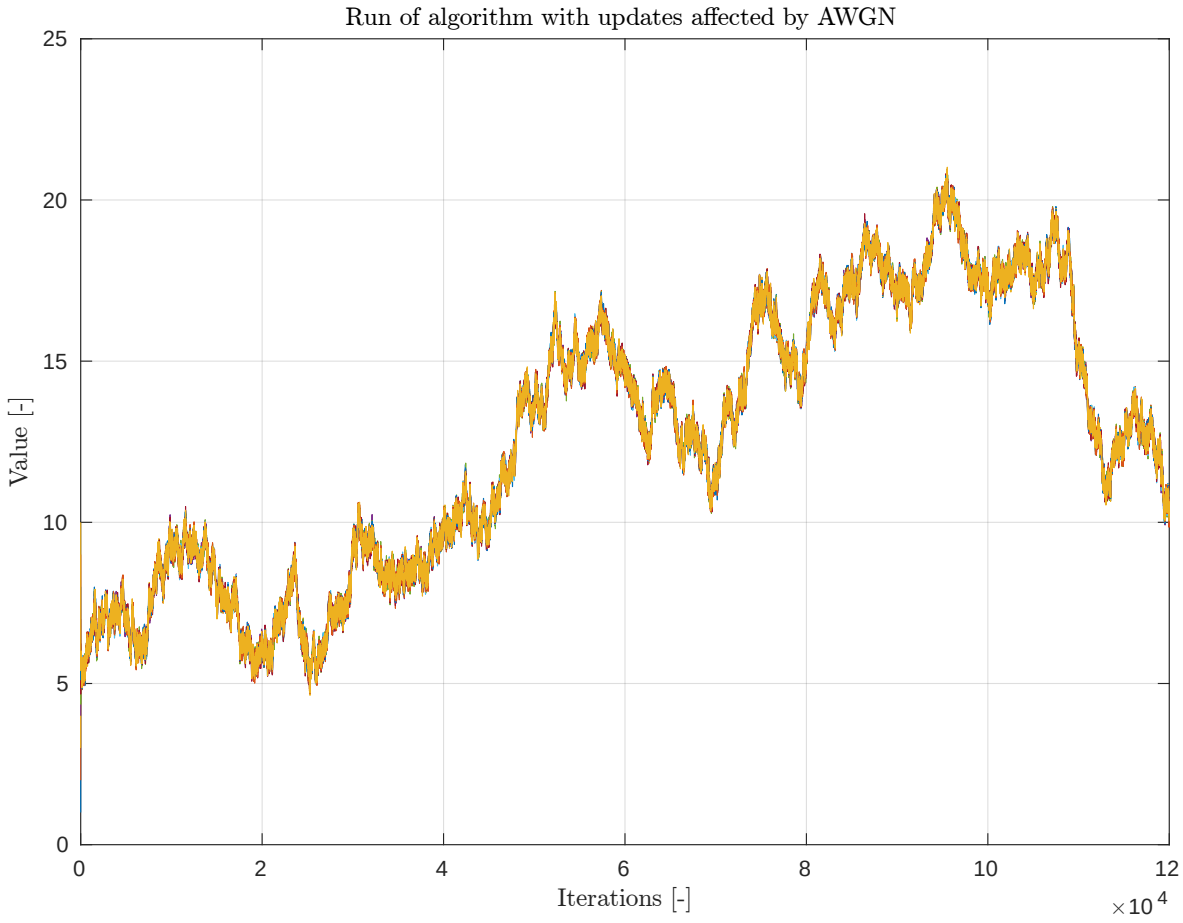


Figure 3.4. More samples to Figure 3.3 show failure of the simplest version of algorithm in presence of noise.

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 convergebecause of increasing variance. In the following section we provide a simple way, according to [18], 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 $\gamma(t+1) < \gamma(t)$. Rouhgly 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 sofisticated methods that take into account the nonidealities of transmision. 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 assumed 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 $x(t) = (x_1(t), x_2(t), ..., x_N(t))^T$. So we keep the taken observations of the nodes, affected by the noise $w(t)$, in vector $x(0) = (x_1(0), x_2(0), ..., x_N(0))^T$. 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

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

where the vector $\zeta(t) = diag\{\boldsymbol{P}(t)\boldsymbol{N}(t)\}$ 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 with Additive noise. The vector $\zeta(t)$ forms a random process and its stochastical 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 [19].

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 [20].

3.6.3 Design of descending step size

There are more approaches to design the step size weigths. In [21] 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, because the conditions in Equations (3.37) and (3.38) are fulfilled, however to avoid an initial divergence of algorithm, coresponding 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 [18] [20] [19].

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 Apendix 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.

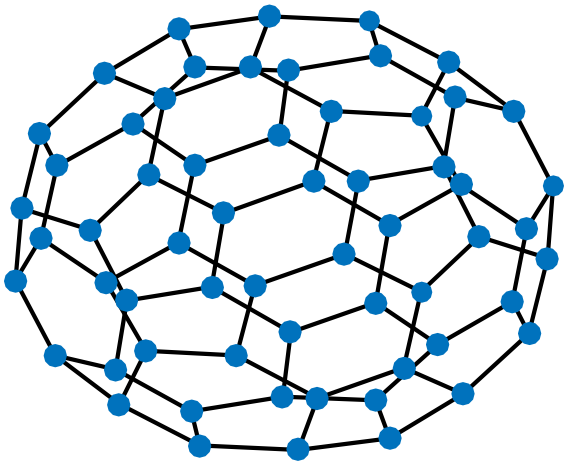


Figure 3.5. Graph used in simulation in Example 3.8.

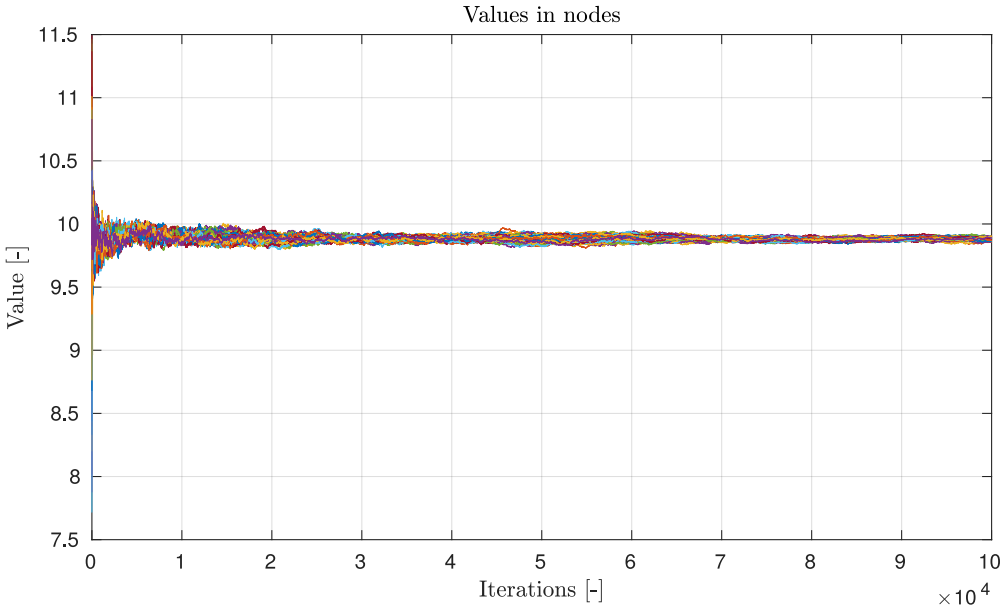


Figure 3.6. The values in nodes of graph in run of 100 000 iterations of the algorithm from Example 3.8

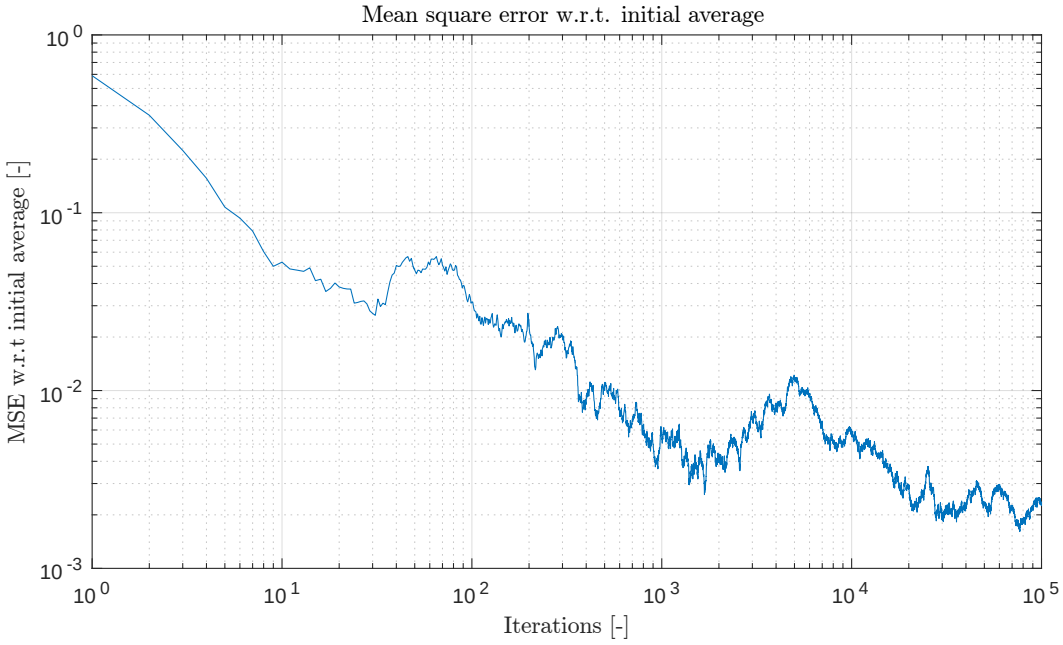


Figure 3.7. Decreasing MSE w.r.t. moving average in nodes of graph in run of the algorithm from Example 3.8. Decrease of this value consequently slows down, because variance of the noise comes to be still more significant.

We can see in Figure 3.6 that such a modification of algorithm works and the values in all nodes converge. In next Figure 3.7 we can see in loglog graph, how the Mean square error of values trully decreases.

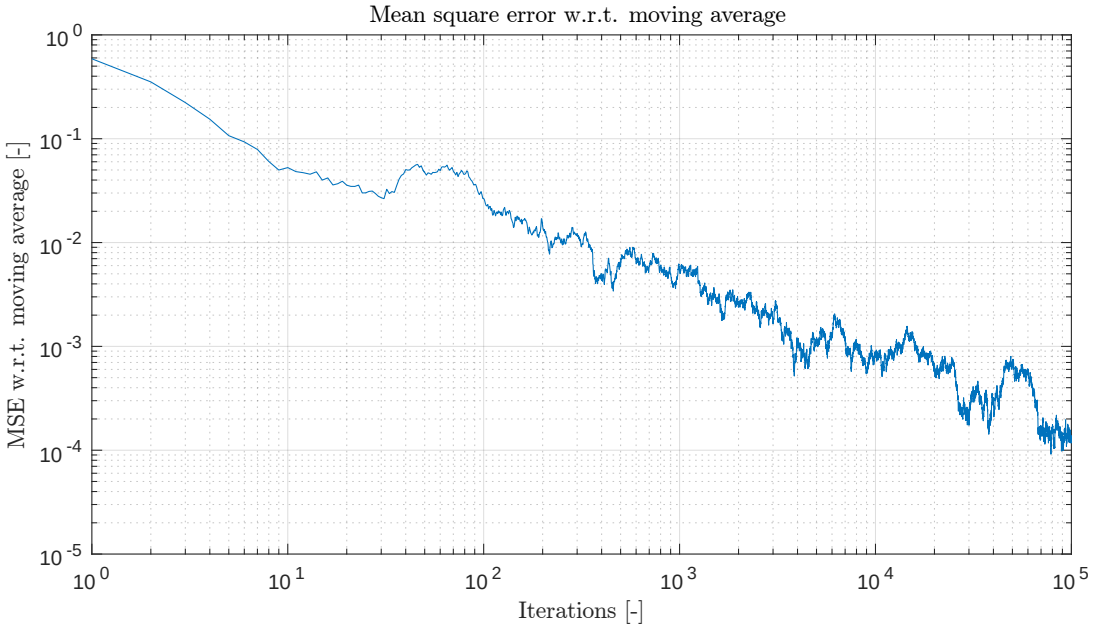


Figure 3.8. Decreasing Mean square error w.r.t. Moving average in nodes of graph in each iteration of run of the algorithm from Example 3.8. We can note, that this value continues to decrease according to fact, that difference between nodes converge to unique value.

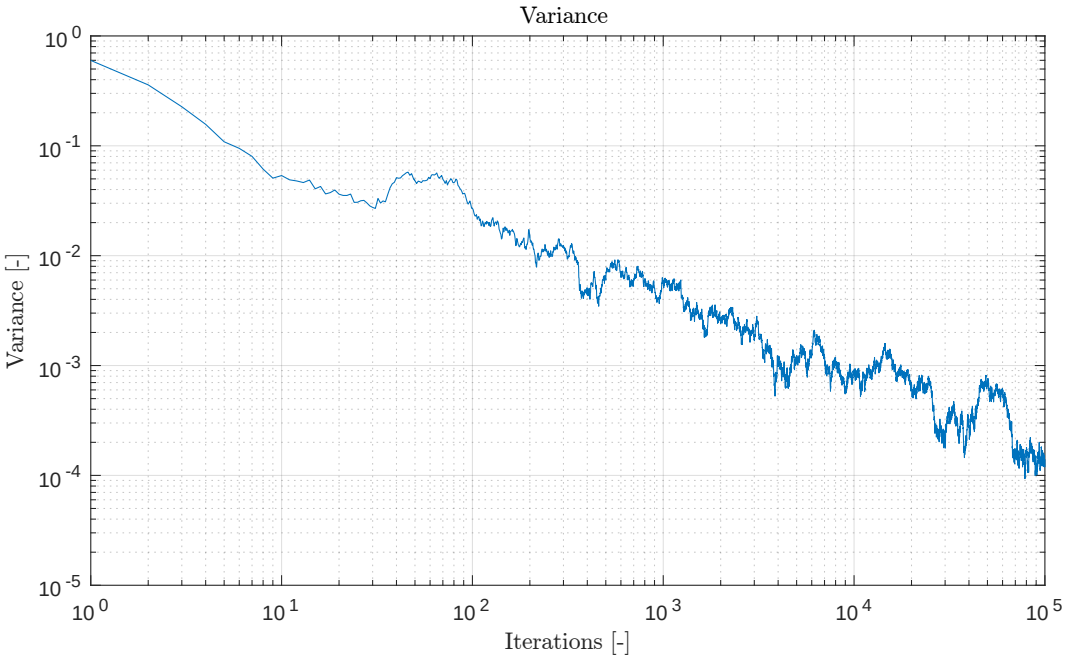


Figure 3.9. Decreasing variance of the values in nodes of graph in run of the algorithm from Example 3.8. We note, that shape of this curve is the same as for Moving MSE from Figure 3.7. This is related to the fact, that for an unbiased estimation MSE is equal to variance.

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.

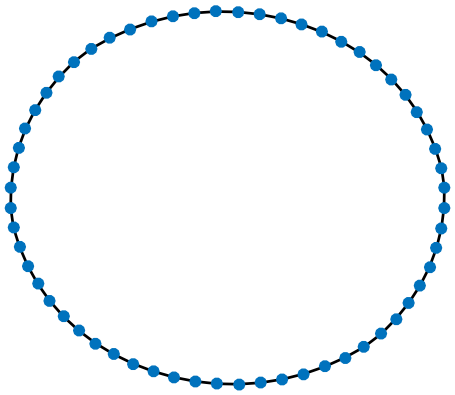


Figure 3.10. 60 nodes ring topology from Example 3.9.

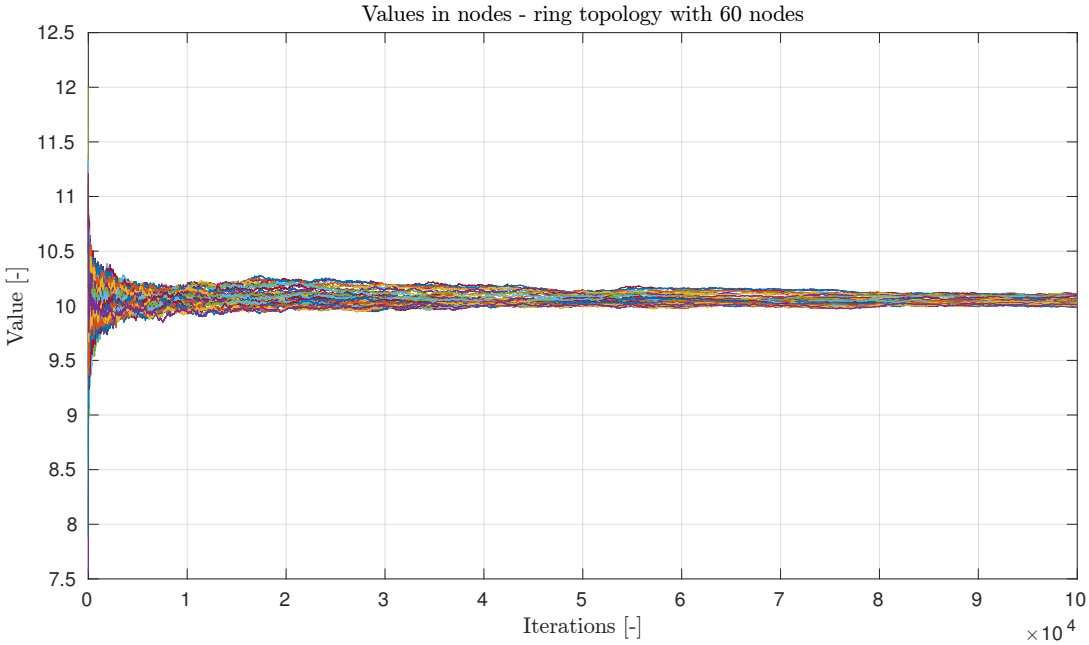


Figure 3.11. Values convergence from Example 3.9.

Let's compare the results from Examples 3.9 and 3.8. Both nodes topologies are in stochasticaly identical environment 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, specificaly 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 analyticaly. The equation to compute Connectivity eigenvalue of Laplacian matrix of N-node ring topology is:

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

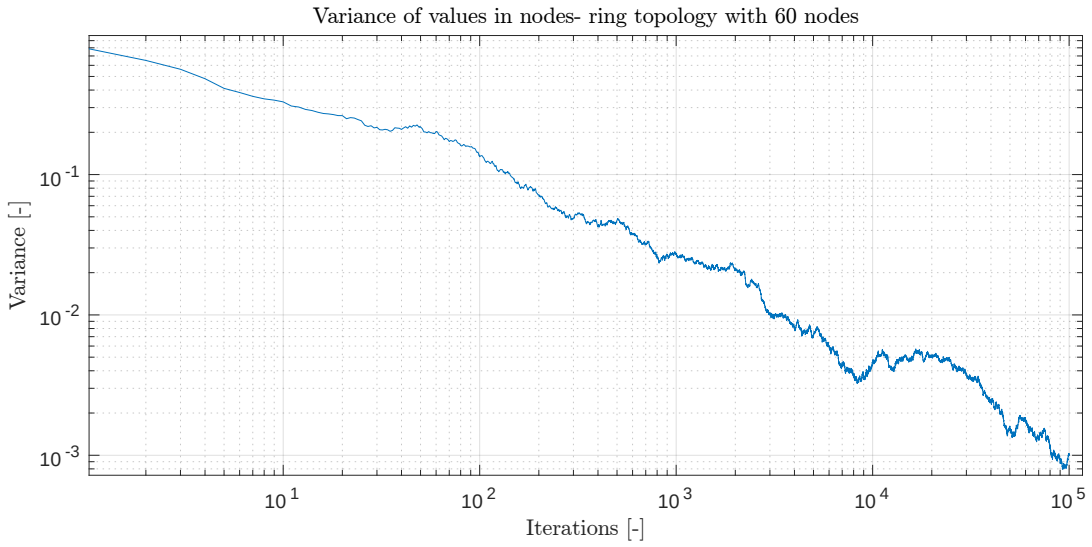


Figure 3.12. Variance running from Example 3.9.

and one can check, that as the number of vertices increases, the connectivity decreases refering to the fact, that the information takes longer time to propagate [22].

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.

- [21] : Here is described the purpose of descending weights during the run of algorithm with noisy updates.
- [19] : This paper is about designing of the weights that minimize MSE during the run of algorithm.
- [23] : Herein were presented two algorithms, that solve problematics 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 [24] where are provided further details.
- [25] : Takes into account quantization noise that takes affect while implementing algorithm in fixed point arithmetics.

Chapter 4
Distributed Estimation in Wireless Sensor Networks

The following chapter should serve as an description of the problematics of distributed estimations in wirelless networks. We will summarize firstly the problematics 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 noniedeal properties of real networks.

This chapter is based on source [26].

4.1 Introduction

Wireless networks are present in a great number of applications of an everyday life. The most commonly given examples are eg. 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 averaging consensus algorithm might be commonly seen in solving a problem of moving in coordinated formations (eg. flights of drons and others unmaned 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 more 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, beacuase 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 [26].

4.2 Overview of Distributed Consensus Estimation

Our general observation model will be

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

where z is our observation, x is the target value, $\boldsymbol{H}$ is an observation matrix and 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 must not have symmetric characteristics, so that e.g. A node can send information to B, however B is not able to A. Hence, a graph describing the WSN inter-nodes communication is then the directed graph.

4.2.1 Consensus-Based Distributed Parameter Estimation

In [26] 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 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 Asymetric communication

Since this model describes situation with asymmetric communication channels a directed graph representation must be used. In our case, a weigted 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 graphps, it is in a litereature common to call a source vertex of an edge *parent* and the destination vertex *child* [26].

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 [26]. 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

[26] solves 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

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

where $\boldsymbol{H_i} \in \mathbb{R}^{J_i \times J}$ is an observation matrix and $w_i(t)$ 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 equiped with the sensors.

The vector nature of θ will consequently bring to the matrix description of the update equations, similiar to the Equation (3.1) Kronecker product. Although such a description is possible, it is very confusing and is reasonable to use probably only in theory. To see this matrix description, we refer to [26].

Definiton 4.1. (Kronecker Product) [27] Given matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and matrix $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, their Kronecker Pruduct $\boldsymbol{C} \in \mathbb{R}^{mp \times nq}$ 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 practise) 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. In each node i is initially known a value $x_i(0)$. $\mathcal{N}_i^S$ and $\mathcal{N}_i^R$ marks the set of SN neighbors and RN neighbors to the $i-$th node, respectively. The update equation of DCUE algorithm states:

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

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; $c_i > 0$ controls impact of $i-$th own measurement 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 , in which h_{ij} is a fading coefficient describing a channel between i and j, and it is a reason why $a_{ij} \neq a_{ji}$ [26].

Next, an update equation for RN nodes reads

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

where γ_{ij} are some nonnegative weighting coefficents [26].

Summarizing the meaning of equations describing the algorithm, that we stated above: Firstly, Equation (4.5) says, that observation of node i, $y_i(t)$, 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 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 recieves 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 characteristcs. 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.

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] Bojan Mohar. *The Laplacian spectrum of graphs*. Wiley, 1991.

[4] Miroslav Dont. *Maticova analyza*. In Prague: Ceska technika, 2011. ISBN "978-8-001-04857-3".

[5] Anne Marsden. *Eigenvalues of the Laplacian and Their Relationship to the Connectedness of a Graph*. 2013.

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

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

[8] Michael William Newman. *The Laplacian Spectrum of Graphs*. 2000.

[9] U.Spagnolini. Distributed signal processing and synchronization, tutorial. 2013,

[10] "Garin. 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" .

[11] 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.

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

[13] 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.

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

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

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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

[25] 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.

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

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

Appendix A
Matlab scripts

A.1 Averaging 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 averaging 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(5)=3; uncomment for convergence to v_3 initial value
    end
end

figure;
plot(discrete_time, running_value(:,:));
ttl1=title( 'Run of algorithm with updates affected by AWGN '  )
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 Averaging 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 averaging 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;
```