Multi-agent System Distributed Sensor Fusion Algorithms

Shaondip Bhattacharya

Specialization : Cybernetics and Robotics
Thesis Supervisor : Kristian Hengster-Movric, Ph.D

Czech Technical University

A thesis submitted for the degree of

Master of Science

June 2017

I dedicate this work to my parents. There are a few more people other than my parents without whom it would not have been possible for me to complete my thesis. I thank you Shiva, Debajyoti, Santanu, Hana, Mondar, Jakub, Rock, Matko for being there whenever I needed you. I thank my advisor, Dr. Kristian Hengster-Movric from the bottom of my heart for his untiring guidance and his patience with me. I take this opportunity to thank my parents again.

I declare that I wrote the presented thesis on my own and that I cited all the used information sources in compliance with the methodical instructions about the ethical principles for writing an academic thesis.

Abstract

The concept of consensus filters for sensor fusion is not an entirely new proposition but one with an internally implemented Bayesian fusion is. This work documents a novel state update algorithm for sensor fusion which works using the principle of Bayesian fusion of data with variance implemented on a single integrator consensus algorithm. Comparative demonstrations of how consensus over a pinning network is reached is presented along with a weighted Bayesian Luenberger type observer and a 'Consensus on estimates' algorithm. This type of a filter is something that is novel and has not been encountered in previous literature related to this topic to the best of our knowledge.

In this work we also extend the proof for a distributed Luenberger type observer design to include the case where the network being considered is a strongly connected digraph.

Contents

1 An Over-view of Sensor Fusion Techniques **1**

1.1 Ergodicity . . . 3

1.2 Bayesian Inference . . . 3

1.3 Bayesian Fusion . . . 4

2 A Study on the Consensus of Multi-Agent Systems through Co-operation over a Network **5**

2.1 Introduction . . . 5

2.2 Mathematical Preliminaries . . . 6

2.2.1 Description of framework . . . 6

2.2.2 Gershgorin's disc theorem . . . 7

2.2.3 The general consensus problem in continuous time . . . 7

2.2.4 Consensus in discrete time . . . 8

2.3 Luenberger Observer on a Directed Graph . . . 9

3 Consensus Based Sensor Fusion **13**

3.1 Bayesian Sensor Fusion . . . 13

3.1.1 Definition of variables . . . 14

3.1.2 The state update algorithms . . . 15

3.2 Noisy Input . . . 17

3.2.1 A constant theoretical measurement . . . 17

3.2.2 A sinusoidal theoretical measurement . . . 17

3.3 Simulation . . . 18

3.3.1 Visualization . . . 19

3.4 Low Pass Distributed Consensus Filter . . . 27

3.5 Simulation . . . 29

3.5.1 Visualization . . . 29

3.6 Luenberger Observer . . . 35

3.7 Simulation 36
3.7.1 Visualization 37
3.8 Conclusion 40

4 Statistical Observations **41**
4.1 Bayesian Consensus Fusion on a Strongly Connected Digraph (where sensors are not communicating to each other) 41
4.2 Recursive vs Non Recursive/Process Variance 42
4.2.1 Discussion on the results 45
4.2.2 Coupling 46
4.2.3 Noise output 52
4.2.4 General discussion 54
4.3 Generalized Consensus on Estimates for n Noisy Sensors in a Communication Network 55
4.4 Process Variance 55
4.4.1 Discussion on results 57
4.4.2 Coupling 57
4.4.3 Noise output 62
4.4.4 General discussion 64
4.5 Luenberger Observer on a Strongly Connected Digraph (where all the nodes are sensors) 65
4.6 Process Variance 66
4.6.1 Discussion on results 66
4.6.2 Coupling 67
4.6.3 Noise output 69
4.6.4 General discussion 71

5 Conclusion and Future Work **72**

Bibliography **73**

List of Figures

1.1 Generalized sensor fusion flowchart 2

3.1 Constant measurement with noise 17
3.2 Sinusoidal measurement with noise 18
3.3 Non recursive state update on SD set 1 19
3.4 Recursive state update on SD set 1 20
3.5 Non recursive state update on SD set 2 20
3.6 Recursive state update on SD set 2 21
3.7 Continuous time state update on SD 1 22
3.8 Continuous time state update on SD 2 23
3.9 Slow sinusoid response for non recursive 24
3.10 Fast sinusoid response for non recursive 24
3.11 Slow sinusoid response for recursive 25
3.12 Fast sinusoid response for recursive 26
3.13 Slow sinusoid response for continuous time 26
3.14 Fast sinusoid response for continuous 27
3.15 Discrete time consensus on estimates for SD 1 29
3.16 Discrete time consensus on estimates for SD 2 30
3.17 Continuous time consensus on estimates for SD 1 30
3.18 Continuous time consensus on estimates for SD 2 31
3.19 Slow sinusoid response for discrete time consensus on estimates 32
3.20 Fast sinusoid response for discrete time consensus on estimates 33
3.21 Slow sinusoid response for continuous time consensus on estimates 34
3.22 Fast sinusoid response for continuous time consensus on estimates 35
3.23 Luenberger observer on SD 1 37
3.24 Luenberger observer on SD 2 38
3.25 Slow sinusoid response of luenberger observer 39
3.26 Fast sinusoid response for luenberger observer 40

4.1 Recursive discrete time bayesian on low SD 43
4.2 Non recursive discrete time bayesian on low SD 44
4.3 Continuous time bayesian state update on low SD 45
4.4 Crosscovariance on DT recursive 47
4.5 Crosscorrelation on DT recursive 47
4.6 Crosscovariance on DT non recursive 50
4.7 Crosscorrelation on DT non recursive 50
4.8 Crosscovariance on CT state update 51
4.9 Crosscorrelation on CT state update 51
4.10 Normplot for non-sensing node 1 52
4.11 Normplot for sensing node 4 53
4.12 Crosscovariance for noise output at node 1 53
4.13 Crosscorrelation for noise output at node 4 54
4.14 Discrete time consensus on estimates on low SD 56
4.15 Continuous time consensus on estimates on low SD 57
4.16 Crosscorrelation on DT consensus on estimates 58
4.17 Crosscovariance on DT consensus on estimates 59
4.18 Crosscorrelation on CT consensus on estimates 60
4.19 Crosscovariance on CT consensus on estimates 61
4.20 Normplot for node 4 62
4.21 Crosscovariance for noise on Consensus on estimates 63
4.22 Crosscorrelation for noise on Consensus on estimates 64
4.23 Bayesian state update crosscorrelation with sinusoidal input 65
4.24 Consensus on estimates crosscorrelation with sinusoidal input 65
4.25 Crosscorrelation on Luenberger 67
4.26 Crosscovariance on Luenberger 68
4.27 Luenberger noise normplot for node 4 69
4.28 Luenberger noise crosscovariance for node 4 70
4.29 Luenberger noise crosscorrelation for node 4 71

Chapter 1

An Over-view of Sensor Fusion Techniques

A core problem in navigating through our different environments lies in the understanding of how we perceive this external world. Information is generally integrated from multiple sources (senses) to enable this awareness, by employing different sensing techniques. The rise of the information age, coupled with improvements in technology has enabled the use of sensors in a variety of applications. Leveraging this influx of data from multiple sensors by fusing them then becomes an important step in obtaining a reliable understanding of the environment.

Sensors are devices that detect, or measure a certain physical property such as range, angle, pressure etc. Any such sensor then comes with an inherent uncertainty, in that the sensor model (the physical relationships between the sensed input and the actual state of the sensed environment) can at best be an approximation of the real world. Factoring in the uncertainty and noise present in the environment, we see that no single sensor can be used by itself to provide a reliable approximation. This is where the use of multiple sensors comes into play. Fusing the information from multiple sensors provides one with data that is much richer and more accurate than otherwise. Multiple sensors can be used to measure the same physical quantities, to provide redundancy in case of sensor failure.

Thus, we see that multi-sensor fusion allows one to minimize the uncertainty inherent in a sensing system, thereby making it more reliable.

The problem then raises the question of how best to fuse such diverse information. Fig 1.1 provides us with a flowchart of how such a process might work. The first step involves an understanding of the nature of input measurements, the larger context/environment, and the sensor limitations. Obtaining a probabilistic understanding of the measurement uncertainty in the sensor also features here.

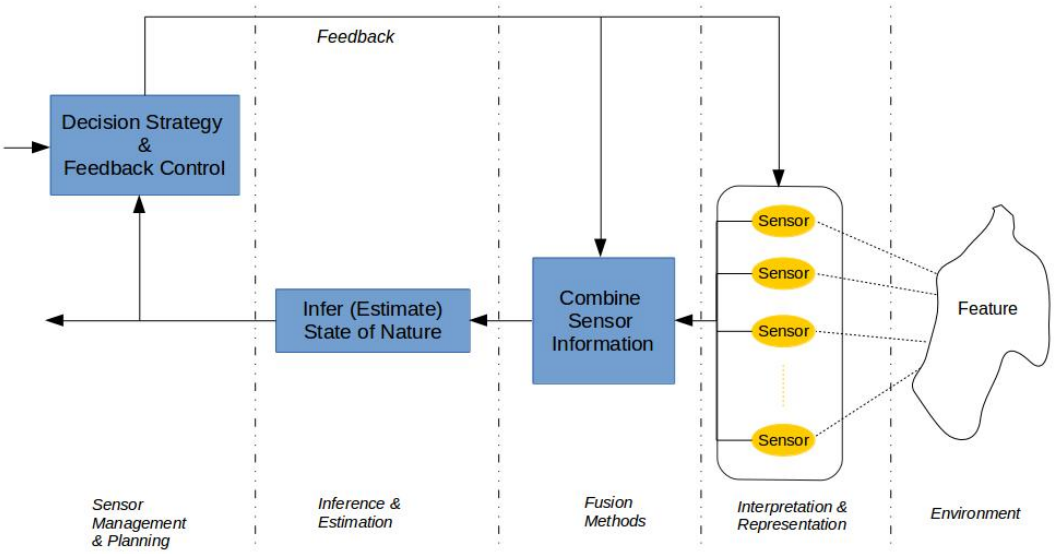


Figure 1.1: Generalized sensor fusion flowchart

Next is the issue of fusing sensed data in a coherent manner, thereby obtaining an estimation of the state of the environment being sensed. Care is taken to account for sensor uncertainties in this step.

In case one is faced with multiple sensor configurations, a third step is required to choose the multi-sensor system that makes best use of the sensors.

Data fusion, in general, encompasses a vast number of topics, ranging from physical sensor modelling to signal processing and filtering and estimation.

Different methods have been developed with the goal of fusing information, and a common approach here is to make use of weighting techniques. The inference problem, wherein one determines the state of the system, is well known, with solutions reported in the form of Bayesian estimation [4], Least Squares Estimation and Kalman Filters [2]. Another variable in the kind of fusion method used concerns the architecture of the sensor system, leading to the use of either centralized, or hierarchical configurations.

1.1 Ergodicity

A stochastic process is said to be **ergodic** if its statistical properties can be determined by one randomly chosen sufficiently long finite sample of the process.

An ergodic process may have various statistics, for example a wide sense stationary process $X(t)$ has constant mean

$$\mu_X = E[X(t)] \tag{1.1}$$

and an autocovariance

$$r_X(\tau) = E[(X(t) - \mu_X)(X(t+\tau) - \mu_X)] \tag{1.2}$$

The notion of ergodicity also applies to discrete-time random processes.
if

$$\hat{\mu}_X = \frac{1}{N}\sum_{n=1}^{N} X[n] \tag{1.3}$$

then the process is said to be ergodic in mean. In this work, all random processes have been assumed to be ergodic unless stated otherwise.

1.2 Bayesian Inference

Bayesian inference [7] is a statistical model which updates the probability of a hypothesis as more information becomes available. It is given as:

$$P(H|E) = P(E|H)P(H)/P(E) \tag{1.4}$$

Here, $P(H)$ is the probability estimate for the hypothesis H when the evidence E is yet to be observed.
$P(H|E)$ is the probability estimate for the hypothesis H when the evidence E is observed.
$P(E)$ is referred to as the 'model evidence' which is the same for all hypotheses, hence it does not affect the model.

1.3 Bayesian Fusion

If x_1,x_2,x_3.... are measurements of the same event with a standard deviation σ_1,σ_2,σ_3.... yielding a fused

$$\hat{x} = [x_1/(\sigma_1)^2 + x_2/(\sigma_2)^2 + x_3/(\sigma_3)^2 + ...]/[1/(\sigma_1)^2 + 1/(\sigma_2)^2 + 1/(\sigma_3)^2 + ...]$$

This is the formulation for Bayesian fusion.

Chapter 2

A Study on the Consensus of Multi-Agent Systems through Co-operation over a Network

2.1 Introduction

The basic ideas and philosophy of cooperative control systems have been inspired by naturally occurring phenomena in biological systems, physics, and chemistry. One of the first attempts to characterize the interaction of connected dynamical systems was by Alan Turing in his paper 'The Chemical Basis of Morphogenesis'[22] where he considers a *reaction diffusion system* which is a ring of n cells each designated by an index i with a ring radius r. Within the cells there are two chemicals referred to as *morphogens* by Alan Turing. The morphogens interconvert with each other and also diffuse around the ring under some specific set of rules. If the concentrations of these two morphogens are c_1 and c_2 then depending on i each cell has a particular 2-tuple $(c_1(i,t), c_2(i,t))$ associated with itself and was calculated by Turing. Such a spatial diffusion of molecules can lead to an ordered spatial pattern of chemical concentrations as documented by [5]. The article [12] documents how a unified framework to systematically formulate the reaction-diffusion system as an interconnected multi-agent dynamical system can be achieved. Hence, this seminal work by Turing is arguably how the study of biologically inspired dynamic interconnected multi agent systems may be understood to have started.

In 1987, Craig Reynolds developed an *artificial life program* called *Boid* [19] which is a short form for "bird-oid" objects. The program simulates naturally occurring flocking in living beings (like birds) by the means of *emergent behavior*. The complexity of Boids arise from the interaction of individual agents (the boids, in this case) adhering to a set of simple

rules. Individual motions in a flock are the result of the balance of two opposing behaviors: a desire to stay close to the group and a desire to avoid collisions with other individuals. Reynolds formulated three rules in his program for the boids to follow. According to him, the desired behaviors that lead to simulated flocking of boids are:
1. Collision Avoidance: avoid collisions with nearby flockmates
2. Velocity Matching: attempt to match velocity with nearby flockmates
3. Flock Centering: attempt to stay close to nearby flockmates
Boids was very successful in modeling the phenomena of flocking. Its most notable uses have been in the field of computer generated graphics with the likes of Stanley and Stella in: Breaking the Ice (1987) and Batman Returns (1992). Boids has also been used to automatically program multi channel Internet based radio stations [3], for time varying data visualization [13] and has been extensively used in swarm robotics [11] [20].

It is understood [8] that the rules such as the one developed by Reynolds, depend on the awareness of each individual of his neighbor. To model the behaviors and interactions of dynamical systems that are interconnected by the links of a communication network, the communication network is modeled as a graph with directed edges corresponding to the allowed flow of information between the systems. The systems are modeled as the nodes in the graph and are sometimes called agents.

In the next section we will familiarize ourselves with the mathematical frameworks for the study of interconnected dynamical systems reaching consensus. In the context of such a system the word "consensus" [15] refers to reaching an agreement regarding a certain quantity of interest that depends on the state of all agents. The phrase 'consensus algorithm' refers to the protocols of interaction (for the purpose of information exchange) between an agent and all of its neighbors on the network.

2.2 Mathematical Preliminaries

2.2.1 Description of framework

For all the discussions that are about to follow, We will be using the following general definitions,
A Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$
The label for each node is $v_i \in \mathcal{V}$ where i$\in\{1, 2, 3...n\}$ where n is the number of nodes in the graph.

The degree of node v_i=deg(v_i) is the number of its neighbours $|N_i|$ where N_i=$\{j : ij \in \mathcal{E}\}$
The degree matrix is an $n \times n$ matrix denoted as Δ where Δ=$\{\Delta_{ij}\}$, where

$$\Delta_{ij} = \begin{cases} \deg(v_i) \text{ for } i = j \\ 0 \text{ for } i \neq j \end{cases}$$

if the adjacency matrix for $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is denoted by A then A=$[a_{ij}]$ where a_{ij}=1 if $(j, i) \in \mathcal{E}$ and $a_{ij} = 0$ otherwise.
Then, the Laplacian matrix of the graph is denoted as L and defined to be L=Δ-A

2.2.2 Gershgorin's disc theorem

Let

$$A = [a_{ij}] \in \mathbb{R}^{n \times n} \tag{2.1}$$

and

$$r_i = \sum_{j=1}^{n} |a_{ij}|, i \neq j \tag{2.2}$$

That is, r_i is the sum of absolute values of all entries in the row i of the adjacency matrix but the diagonal element. It is a measure of how well connected node i is with its neighbours.

Gershgorin's Disc Theorem states that all eigenvalues of A are located in the union of n disks,

$$G(A) : \bigcup_{i=1}^{n} G_i(A) \tag{2.3}$$

where

$$G_i(A) = \{z \in \mathcal{C} : |z - a_{ii}| \leq r_i\} \tag{2.4}$$

2.2.3 The general consensus problem in continuous time

A simple consensus algorithm to reach an agreement regarding the state of n integrator agents with dynamics $\dot{x_i} = u_i$ can be expressed as following [16]

$$\dot{x_i}(t) = \sum_{j \in N_i} e_{ij}(x_j(t) - x_i(t)) + b_i(t) \tag{2.5}$$

$$x_i(0) = z_i \in \mathbb{R}$$

The collective dynamics of the group of agents following the algorithm 3.1 can be written as

$$\dot{x} = -Lx \tag{2.6}$$

where L is the graph laplacian of the network.
Now since L=Δ-A, L has zero row sum.
Hence, 0 is an eigenvalue of L with associated eigenvector $\mathbf{1} \triangleq [1, 1, 1, ...]^T$
Also from Gershgorin's Disc Theorem, all non zero eigenvectors of L are positive for undirected graphs or have positive real parts for directed graphs.
And, span of the eigenvector **1** (since it has an associated eigenvalue of zero) is contained in the kernel of L or the null space of L.
It follows that if zero is a simple eigenvalue of L then $x(t) \longrightarrow \bar{x}\mathbf{1}$ where $\bar{x}$ is a scalar constant. This implies that $|x_i(t) - x_j(t)| \longrightarrow 0$ as $t \longrightarrow \infty$
For convergence analysis, we try to show that zero is a simple eigenvakue of L.

If a directed graph is strongly connected then zero is a simple eigenvalue of L. However this is not the necessary condition according to [9] [6] [18]. Zero is a simple eigenvalue of L if and only if the associated directed graph contains a rooted spanning tree.

2.2.4 Consensus in discrete time

An iterative form [15] of the consensus algorithm can be stated as follows in discrete time:

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} e_{ij}(x_j(t) - x_i(t)) \tag{2.7}$$

The discrete time collective dynamics of the network can thus be written as:

$$x(k+1) = Px(k) \tag{2.8}$$

Where P=$I - \epsilon L$ and $\epsilon > 0$ is the step size. In general, $P = exp(-\epsilon L)$ where P is referred to as the *Perron* matrix of a graph G with parameter ϵ.

2.3 Luenberger Observer on a Directed Graph

Let us consider a continuous time linear system which is a representative of the state trajectory

$$\dot{x} = A(t)x + B(t)w \tag{2.9}$$

being sensed by a sensor modeled by

$$z = H(t)x + v \tag{2.10}$$

Then the continuous time Kalman filter can be modeled as the following set of equations

$$\dot{\hat{x}} = A\hat{x} + K(z - H\hat{x}) \tag{2.11}$$

$$K = PH^T R^{-1} \tag{2.12}$$

$$\dot{P} = AP + PA^T + BQB^T - PH^T R^{-1} HP \tag{2.13}$$

Here, $\dot{\hat{x}}$ =central estimate associated with data z. P is the estimation error covariance matrix, H is the observation matrix and Q and R are the process and measurement noise covariance matrices respectively.

Lemma 1[14]. Let $\eta = x - \hat{x}$ denote the estimation error in the model described above.Then, $\dot{\eta} = (A - KH)\eta + w_e$ where w_e is the input noise to the system and $w_e = Bw + Kv$. Without noise, the error dynamics is a stable linear system with a Lyapunov function

$$V(\eta) = \eta^T P(t)^{-1} \eta \tag{2.14}$$

differentiating, we see :

$$\dot{V} = \dot{\eta}^T P^{-1}\eta + \eta^T P^{-1}\dot{\eta} - \eta^T P^{-1}\dot{P}P^{-1}$$

$$= \eta^T[(A-KH)^T P^{-1} + P^{-1}(A-KH)]\eta - \eta^T[P^{-1}A + A^T P^{-1} + P^{-1}BQB^T P^{-1} - H^T R^{-1} H]\eta$$

$$= \eta^T[H^T R^{-1} H + P^{-1}BQB^T P^{-1}]\eta < 0$$

for all

$$\eta \neq 0$$

Thus, $\eta = 0$ is globally asymptotically stable and $V(\eta)$ is a Lyapunov function for the linear system formed by the error dynamics.

Consider a network of n sensors with the following sensing model:

$$z_i(t) = H_i(t)x + v_i \tag{2.15}$$

with

$$E[v_i(t)v_i(s)^T] = R_i\delta(t-s) \tag{2.16}$$

Assuming that the pair(A,H) with

$$H = col(H_1, H_2...H_n) \tag{2.17}$$

is observable, then we propose (extending on proposition 2 on undirected graphs in [14]):
Proposition 1. For a sensor network that is represented by a strongly connected digraph with an edge weight being represented by e_{ij} and has a sensing model equivalent to (7), Then if each node applies the following distributed estimation algorithm

$$\dot{\hat{x_i}} = A\hat{x_i} + K_i(z_i - H_i\hat{x_i}) + \gamma P_i \sum_{j\in N_i} e_{ij}(\hat{x_j} - \hat{x_i}) \tag{2.18}$$

$$K_i = P_i H_i^T R_i^{-1}, \gamma > 0$$

$$\dot{P_i} = AP_i + P_iA^T + BQB^T - K_iR_iK_i^T$$

with a Kalman consensus estimator and initial conditions $P_i(0) = P_0$ and $\dot{\hat{x_i}} = x(0)$, then the collective dynamics of the estimation errors $\eta_i = x - \hat{x_i}$ (without noise) is a stable linear system with a Lyapunov function (see lemma 6 of [23])

$$V(\eta) = \sum_{i=1}^{N} p_i\eta^T P(t)^{-1}\eta \tag{2.19}$$

where $p = [p_1, p_2, ..p_i.., p_N]^T$ is the left eigenvector of the graph laplacian L associated with eigenvalue λ=0 (i.e.,p is the first left eigenvector of L) and $p^T L = 0$ and $p_i > 0 \; \forall i \in N$. Also, asymptotically all estimators agree i.e. $\dot{\hat{x_1}} = \dot{\hat{x_2}} = ...\dot{\hat{x_n}} = x$

Proof: Defining vectors

$$\eta = col(\eta_1, \eta_2 ... \eta_n)$$

$\hat{x} = col(\hat{x_1}, \hat{x_2}..\hat{x_n})$
we note that $\dot{\hat{x}}_j - \dot{\hat{x}}_i = \eta_i - \eta_j$ and hence the estimator dynamics can be written as

$$\dot{\hat{x}}_i = A\hat{x}_i + K_i(z_i - H_i\hat{x}_i) - \gamma P_i \sum_{j \in N_i} e_{ij}(\eta_j - \eta_i) \tag{2.20}$$

which gives us the following error dynamics for the ith node

$$\dot{\eta}_i = (A - K_i H_i)\eta_i + \gamma P_i \sum_{j \in N_i} e_{ij}(\eta_j - \eta_i) \tag{2.21}$$

denoting $F_i = A - K_i H_i$ we write,

$$\dot{\eta}_i = F_i \eta_i + \gamma P_i \sum_{j \in N_i} e_{ij}(\eta_j - \eta_i) \tag{2.22}$$

Now,

$$\dot{V} = \sum_i p_i(\eta_i^T P_i^{-1} \dot{\eta}_i + \dot{\eta_i}^T P_i^{-1} \eta_i - \eta_i^T P_i^{-1} \dot{P}_i P_i^{-1} \eta_i) + \dot{p_i}(\eta_i^T P_i^{-1} \eta_i) \tag{2.23}$$

since $\dot{p_i} = 0$,

$$\dot{V} = \sum_i p_i(\eta_i^T P_i^{-1} \dot{\eta}_i + \dot{\eta_i}^T P_i^{-1} \eta_i - \eta_i^T P_i^{-1} \dot{P}_i P_i^{-1} \eta_i) \tag{2.24}$$

and

$$\eta_i^T P_i^{-1} \dot{\eta}_i = \eta_i^T P_i^{-1} F_i \eta_i + \gamma \eta_i^T \sum_{j \in N_i} e_{ij}(\eta_j - \eta_i) \tag{2.25}$$

Adding the terms in $\dot{V}$ and using lemma 1 gives us,

$$\dot{V} = -\sum_i p_i(\eta_i^T[H_i^T R_i^{-1} H_i^T + P_i^{-1} B Q B^T P^{-1}]\eta_i) + \gamma \sum_{i,j} p_i e_{ij}(\eta_j - \eta_i)^2 \tag{2.26}$$

$$\dot{V} = -\eta^T \Lambda \eta - \gamma \sum_{i,j} p_i e_{ij}(\eta_j - \eta_i)^2 \leq 0 \tag{2.27}$$

Here Λ is a positive definite block diagonal matrix with diagonal blocks

$$p_i(H_i^T R_i^{-1} H_i^T + P_i^{-1} B Q B^T P^{-1})$$

Also, $\dot{V} = 0$ implies that all η_i's are equal and $\eta = 0$. Hence $\hat{x_i} = x$ $\forall$i as t $\to \infty$
Later, we demonstrate an implementation of the same in MATLAB and present the numerical results in the final section.

Chapter 3

Consensus Based Sensor Fusion

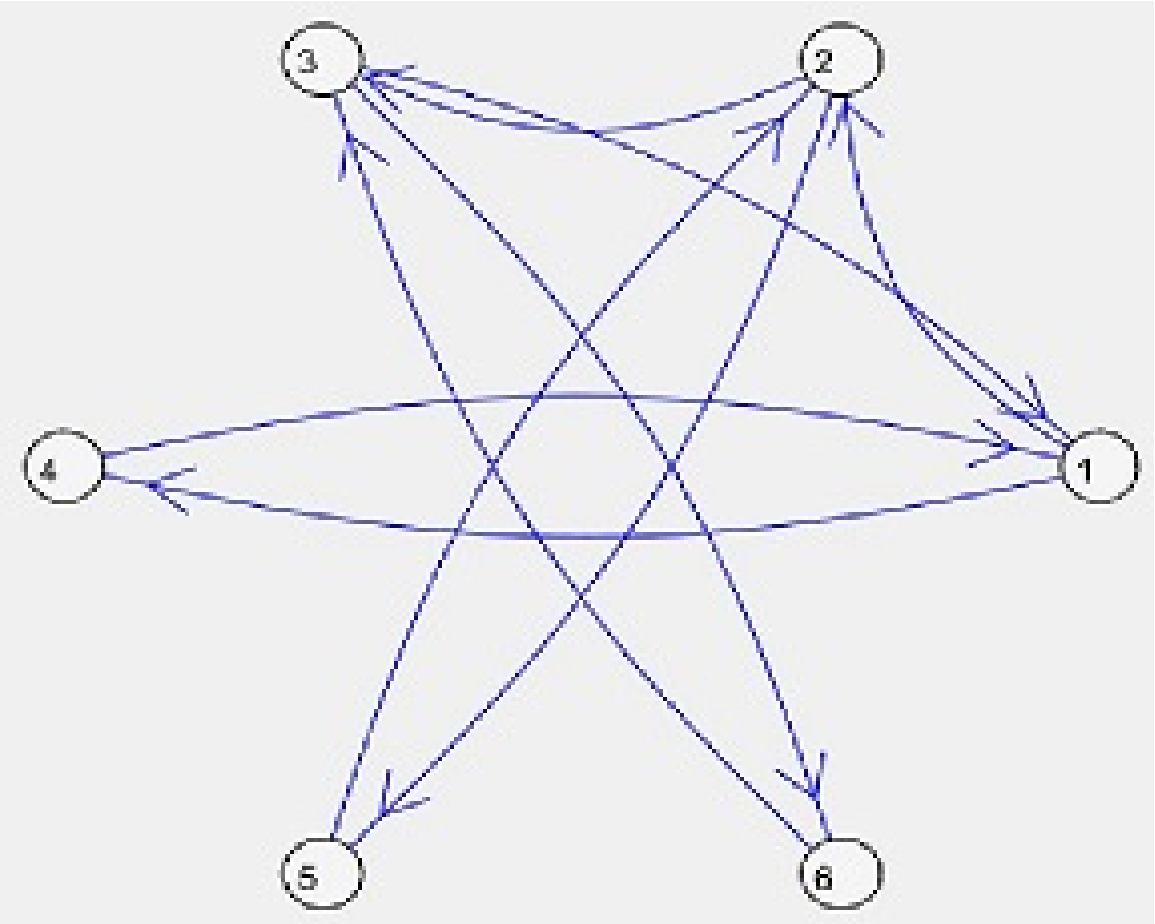
3.1 Bayesian Sensor Fusion

Here we propose a set of algorithms which may be termed as consensus filters with internally implemented Bayesian fusion. It is something that is novel and has not been encountered in previous literature related to this topic to the best of the author's knowledge. The objective in this section is to construct the Bayesian sensor fusion algorithms on a particular network structure which essentially consists of a strongly connected graph of nodes which cannot sense, but can update their states according to the trajectory defined by the sensors which are pinned on to them. The non-sensing nodes also communicate with the other non sensing nodes as well as the sensing nodes. We consider the multiagent system to have state dynamics represented by

$$\dot{x}(t) = u$$

where x is the state at a node and u is the input at that node. For the purpose of demonstration, a ring network with sensors(leaders) pinned on each node has been used. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the graph representing the network. Let the adjacency matrix for the said graph be,
$\mathcal{A} = [a_{ij}]$ where a_{ij}=1 if$(j, i) \in \mathcal{E}$ and $a_{ij} = 0$ otherwise. Below we present the adjacency matrix that we used for the simulation along with a visual representation of the graph $\mathcal{G}$.

$$\mathcal{A} = \begin{matrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$



Structure of the communication network

3.1.1 Definition of variables

Here we list all the variables that have been used in this work

$\mathbb{N}$= Set of node labels of non sensing nodes

$\mathbb{S}$= Set of node labels of sensing nodes

The indices used for non−sensing nodes are [i,j,l]

The indices used for sensing nodes are [j_0,k_0]

$\xi_{j_0}(k) = x_0 + \delta_{j_0}$ where x_0 is the correct measurement and δ_{j_0} is a zero mean Gaussian noise i.e $\langle \delta_{j_0} \rangle = 0$

e_{ij} is the set of edges which connect the non sensing nodes

e_{ij}=a_{ij} where i and j $\in \mathbb{N}$

g_{ij_0} is the set of pinning gains

g_{ij_0}=a_{ij} where i $\in \mathbb{S}$ and j $\in \mathbb{N}$

$h_{j_0 l}$ is the set of feedback gains from non sensing node to the sensing nodes

$h_{j_0 l}$=a_{ij} where i $\in \mathbb{S}$ and j $\in \mathbb{N}$

d_i is the diagonal matrix of the row sums of the adjacency matrix of the graph consisting solely of the non-sensing nodes, $[d_i]$=$[a_{ij}]$ where i and j $\in \mathbb{N}$

For our particular digraph, we choose $\mathbb{N}$={1,2,3} and

$\mathbb{S}$={4,5,6}

3.1.2 The state update algorithms

Upon focusing our attention on the non sensing nodes, it is noted that under the combined influence of the non-sensing nodes and the sensing nodes if the state of the non-sensing node is denoted as x_i, then the continuous time state dynamics for a particular time index i can be written down as,

$$\dot{x_i} = \sum_j e_{ij}(x_j - x_i) + \{\sum_{j_0} g_{ij_0}(x_{j_0} - x_i)/\sigma_{j_0}^2\}/(\sum_{j_0} g_{ij_0}/\sigma_{j_0}^2) \tag{3.1}$$

or upon expansion

$$\dot{x_i} = -d_i x_i + \sum_j e_{ij} x_j + \{\sum_{j_0} g_{ij_0} x_{j_0}/\sigma_{j_0}^2/\sum_{j_0} g_{ij_0}/\sigma_{j_0}^2\} - x_i \tag{3.2}$$

Now, at steady state $\dot{x_i} = 0$ therefore we can come to the conclusion that at steady state

$$x_i = [\sum_j e_{ij} x_j + \{\sum_{j_0} g_{ij_0} x_{j_0}/\sigma_{j_0}^2/\sum_{j_0} g_{ij_0}/\sigma_{j_0}^2\}]/(1 + d_i) \tag{3.3}$$

And the discrete time state update algorithm can be written down as

$$x_i(k+1) = [x_i(k) + \sum_j e_{ij} x_j + \{\sum_{j_0} g_{ij_0} x_{j_0}/\sigma_{j_0}^2/\sum_{j_0} g_{ij_0}/\sigma_{j_0}^2\}]/(2 + d_i) \tag{3.4}$$

in recursive form.

For the sake of exposition, the discrete time version may also be written as,

$$x_i(k+1) = [x_i(k) + M + N]/(2 + d_i) \tag{3.5}$$

Here M=$\sum_j e_{ij} x_j(k)$ and is the average state of all the neighboring non-sensing nodes multiplied by the in-degree and N=$\sum_{j_0} g_{ij_0} x_{j_0}(k)/\sigma_{j_0}^2/\sum_{j_0} g_{ij_0}/\sigma_{j_0}^2$ which is the Bayesian fused sensor reading from all the sensing nodes connected to x_i.

Intuitively too, it is evident that the (k+1)th state of a non-sensing node is influenced by 3 factors, namely its own previous state, the average influence of its non-sensing neighbors weighted by their degree of connectivity to the node in consideration and the fused sensor reading from all the sensing nodes connected to it.

Now we take a look at the sensing nodes. The update equation for the sensing nodes can have multiple forms. We write the continuous time version as follows:
We can write it as

$$\dot{x} = [1/(\sigma_{j_0}^2) + (\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)]^{-1}[(\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)(\sum_{l} h_{j_0 l})^{-1}(\sum_{l} h_{j_0 l}(x_l - x_{j0})) + (\xi_{j_0}/\sigma_{j_0}^2)] \tag{3.6}$$

$$\begin{aligned} x_{j_0}(k+1) =& [(\xi_{j_0}(k)/\sigma_{j_0}^2) + (\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)(\sum_{l} h_{j_0 l} x_l(k) \\ & /\sum_{l} h_{j_0 l})/\{(1/\sigma_{j_0}^2) + \sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2\}] \end{aligned} \tag{3.7}$$

or as

$$x_{j_0}(k+1) = (P+Q)/R$$

where P=$\xi_{j_0}(k)/\sigma_{j_0}^2$
is the noisy measurement at the node x_{j_0} divided by the variance at the node.
Q=$(\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)(\sum_l h_{j_0 l} x_l(k)/\sum_l h_{j_0 l})$ is the average state of all non-sensing nodes attached to the sensing node multiplied by the total network variance (excluding the node x_{j_0})
And R=$(1/\sigma_{j_0}^2) + \sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2$ which is the total network variance. This is essentially the standard Bayesian fusion equation which can be applied upon assuming a scenario where x_1,x_2,x_3.... are a measurement of the same event with a standard deviation σ_1,σ_2,σ_3.... yielding a fused

$$\hat{x} = [x_1/(\sigma_1)^2 + x_2/(\sigma_2)^2 + x_3/(\sigma_3)^2 + ...]/[1/(\sigma_1)^2 + 1/(\sigma_2)^2 + 1/(\sigma_3)^2 + ...]$$

Now we can transform this same equation to a recursive form by taking the average of the current fused reading and the posterior fused reading, that is we can write

$$x_{j_0}(k+1) = 1/2[(P+Q)/R + x_{j_0}(k)]$$

that is,

$$x_{j_0}(k+1) = 1/2[(\xi_{j_0}(k)/\sigma_{j_0}^2) + (\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)(\sum_{l} h_{j_0 l} x_l(k) / \sum_{l} h_{j_0 l})/\{(1/\sigma_{j_0}^2) + \sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2\} + x_{j_0}(k)] \tag{3.8}$$

3.2 Noisy Input

The following two general kind of noisy input signals have been used for the demonstration of tracking performance for all algorithms in this chapter:

3.2.1 A constant theoretical measurement

This kind of noisy measurement is of the form of
$\xi_{j_0}(k) = x_0 + \delta_{j_0}$ where x_0 is the constant and δ_{j_0} is a zero mean Gaussian noise i.e $\langle \delta_{j_0} \rangle = 0$

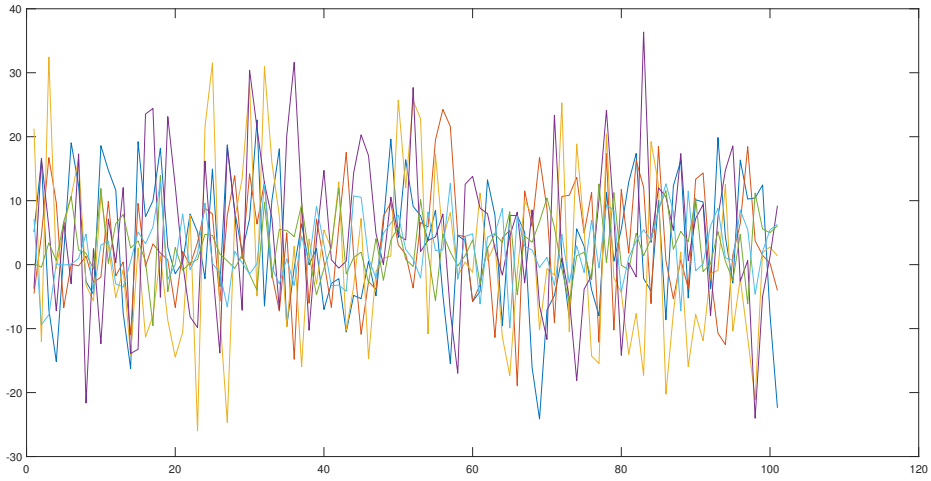


Figure 3.1: Constant measurement with noise

3.2.2 A sinusoidal theoretical measurement

This kind of noisy measurement is of the form of
$\xi_{j_0}(k) = x(t) + \delta_{j_0}$ where $x(t)$ is a sinusoid and δ_{j_0} is a zero mean Gaussian noise i.e $\langle \delta_{j_0} \rangle = 0$

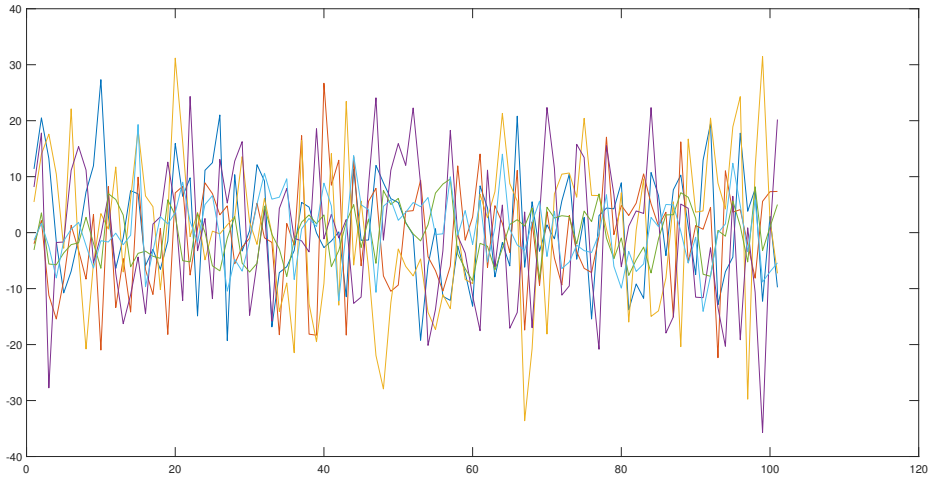


Figure 3.2: Sinusoidal measurement with noise

3.3 Simulation

Coding the above state update algorithms was done in MATLAB 2016B. The results shown here are for all the variations of the state update algorithms that have been mentioned above.

Starting with the actual theoretical measurement value of 3 the state update algorithm for the network was run for 100 iterative steps.

The following arbitrary value sets were the initial conditions:
Set of initial states x=[0.1,0.2,0.31,3.2,4.2,1.2]
Initial noisy measurements ξ =[0,0,0,3.7243,4.7243,2.7243]
Standard deviations at a node σ=[0,0,0,0.3,0.21,0.004]

later it was changed to σ=[0,0,0,0.2,0.4,0.3]
and it was seen that the algorithm consistently 'distrusts' the sensing node with the maximum standard deviation the most, and subsequently the rate of convergence is greatest at the noisiest node and so on. In the 1st case, the node number 4 with an initial state of 3.2 was the noisiest with a standard deviation of 0.3. And in the 2nd case, the node number 5 with an initial state of 4.2 was the noisiest with a standard deviation of 0.4.

After this, we demonstrate the tracking performance of such algorithms where the tracking signal is a Sinusoid. All the other conditions remain the same. For the purpose of demonstration we choose the second set of standard deviation values.

3.3.1 Visualization

Fig 3.3 is for the first variation of the state update algorithm (non recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.7) on the first set of standard deviations.

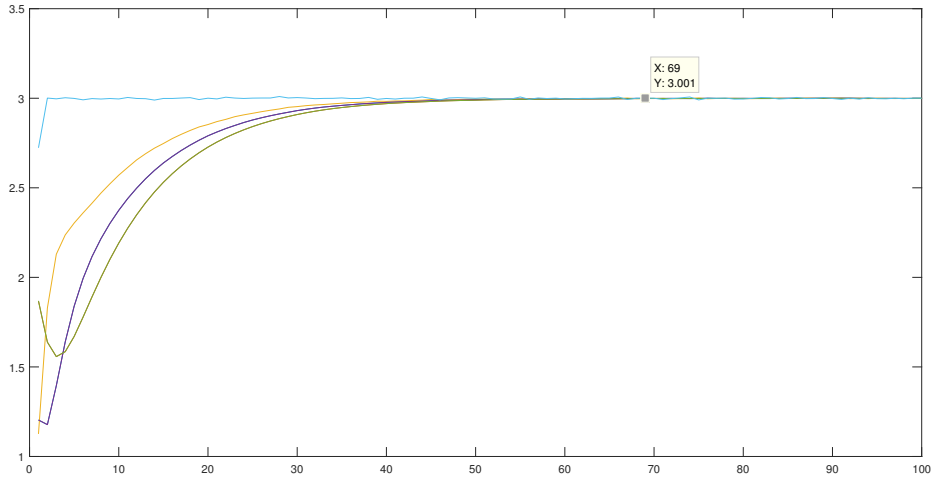


Figure 3.3: Non recursive state update on SD set 1

Fig 3.4 is for the second variation of the state update algorithm (recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.8) on the first set of standard deviations

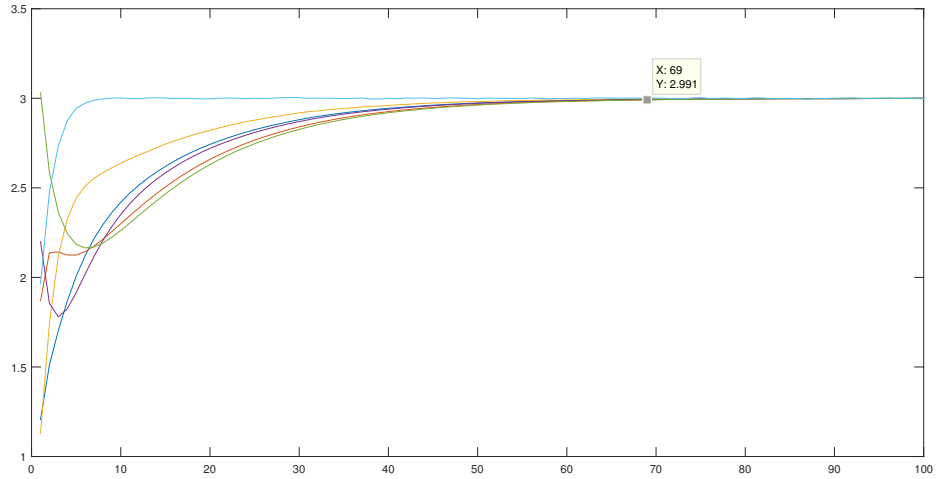


Figure 3.4: Recursive state update on SD set 1

Fig 3.5 is for the first variation of the state update algorithm (non recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.7) on the second set of standard deviations

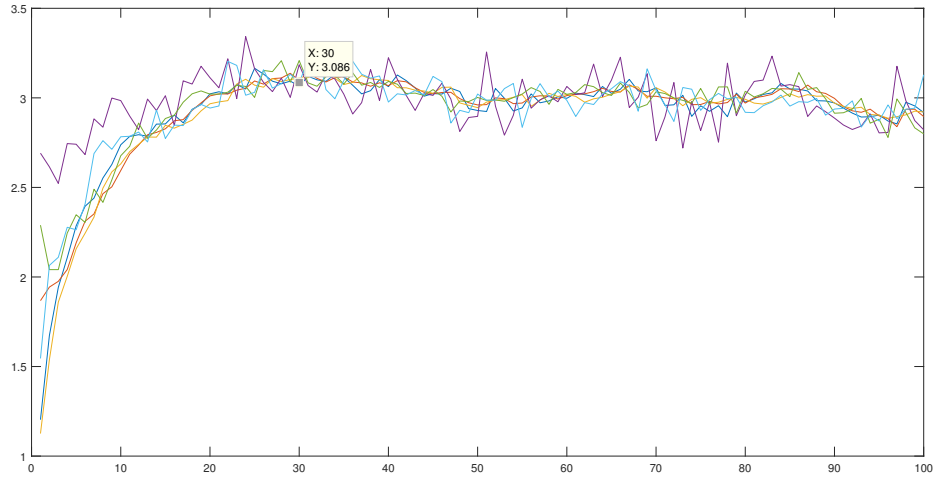


Figure 3.5: Non recursive state update on SD set 2

Fig 3.6 is for the second variation of the state update algorithm (recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.8) on the

second set of standard deviations

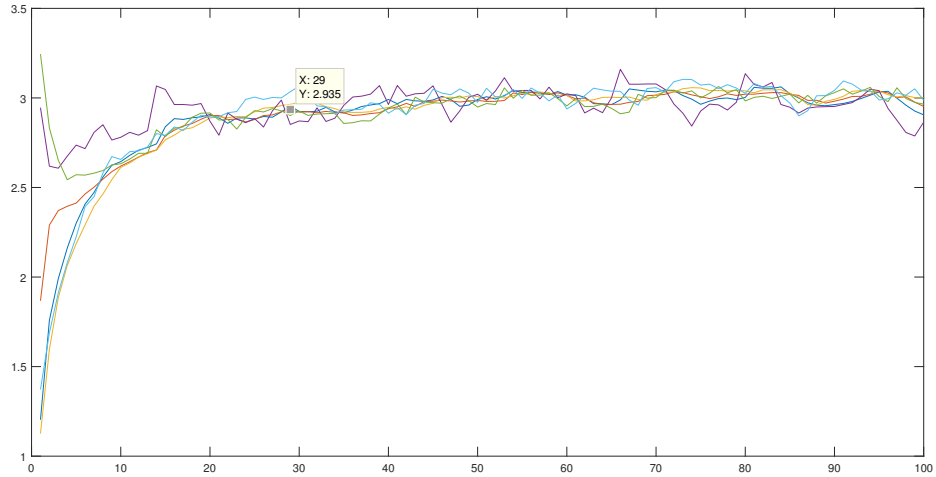


Figure 3.6: Recursive state update on SD set 2

Fig 3.7 is for the continuous time state update algorithm, which is the pair of update equations numbered (3.2) and (3.6) on the first set of standard deviations

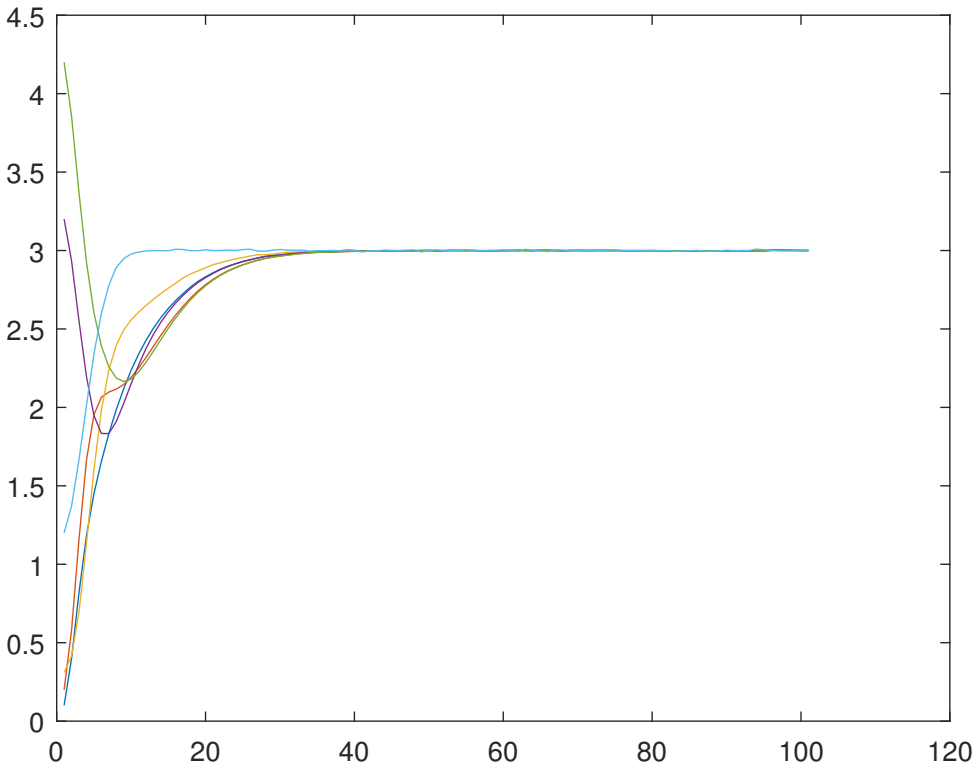


Figure 3.7: Continuous time state update on SD 1

Fig 3.8 is for the continuous time state update algorithm, which is the pair of update equations numbered (3.2) and (3.6) on the second set of standard deviations

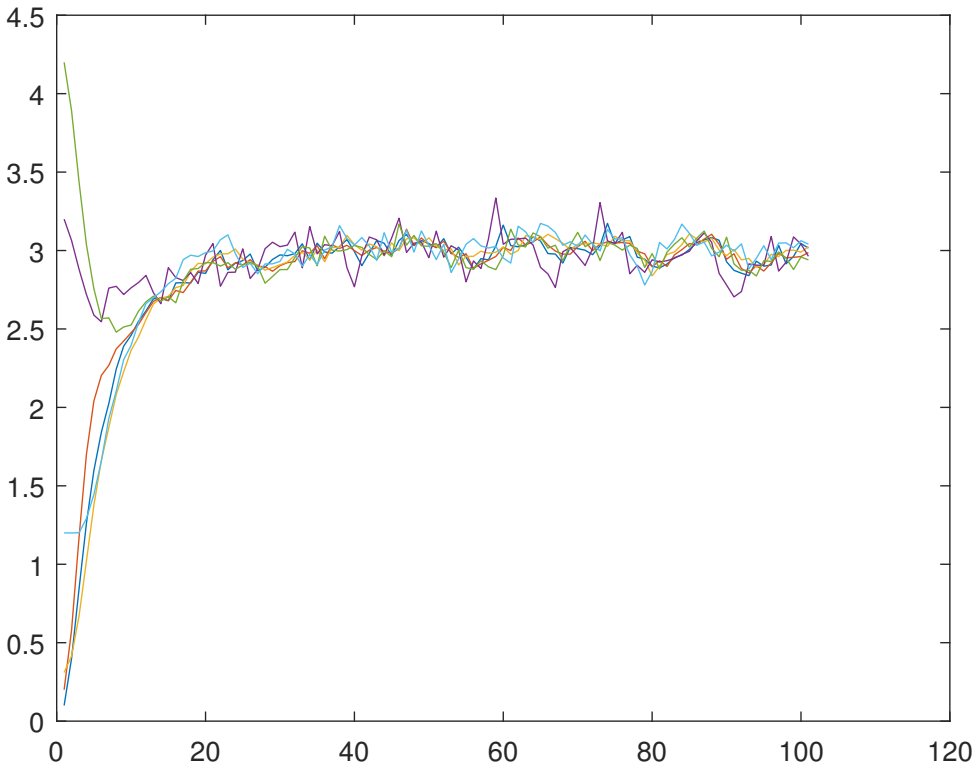


Figure 3.8: Continuous time state update on SD 2

Fig 3.9 is a demonstration for the sinusoidal tracking performance of the first variation of the discrete time state update algorithm (non recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.7) on the second set of standard deviations. The signal that has been used here as an input is a slowly varying sinusoid.

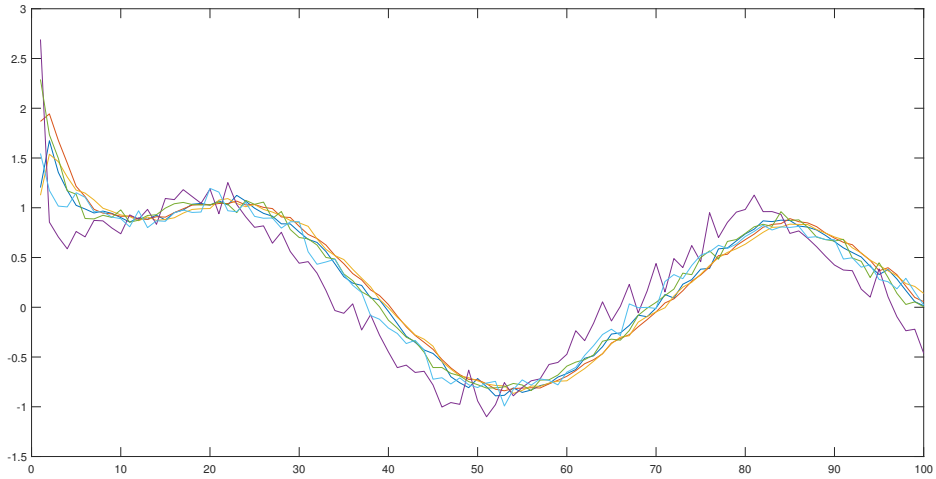


Figure 3.9: Slow sinusoid response for non recursive

Fig 3.10 is a demonstration for the sinusoidal tracking performance of the first variation of the discrete time state update algorithm (non recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.7) on the second set of standard deviations. The signal that has been used here as an input is a fast varying sinusoid.

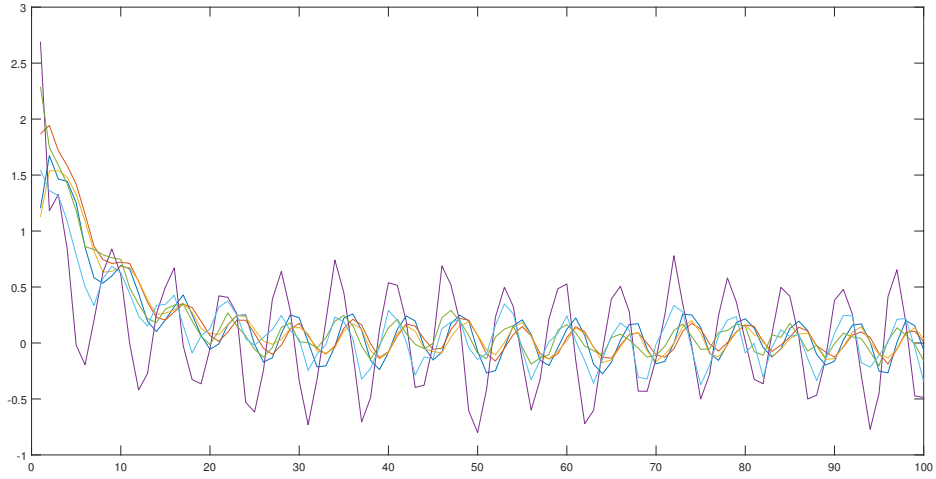


Figure 3.10: Fast sinusoid response for non recursive

Fig 3.11 is a demonstration for the sinusoidal tracking performance of the second varia-

tion of the discrete time state update algorithm (recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.8) on the second set of standard deviations. The signal that has been used here as an input is a slowly varying sinusoid.

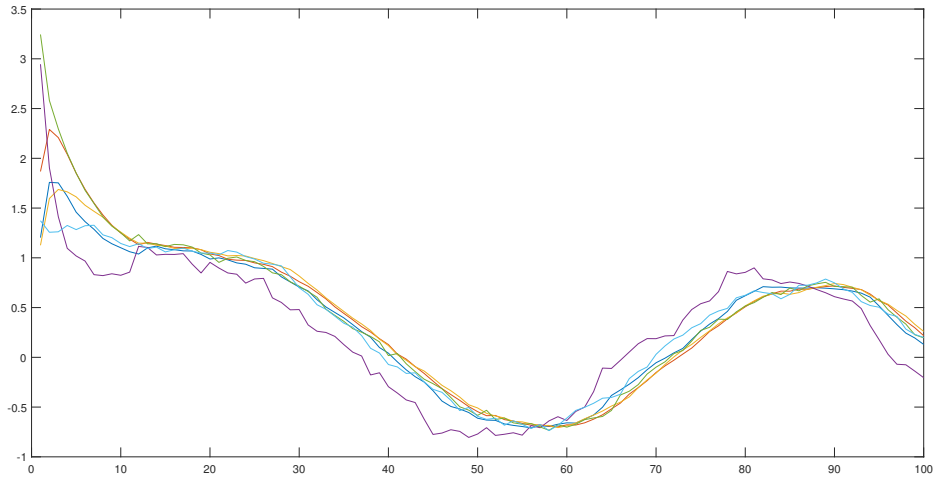


Figure 3.11: Slow sinusoid response for recursive

Fig 3.12 is a demonstration for the sinusoidal tracking performance of the second variation of the discrete time state update algorithm (recursive update scheme for sensor nodes), which is the pair of update equations numbered (3.4) and (3.8) on the second set of standard deviations. The signal that has been used here as an input is a fast varying sinusoid.

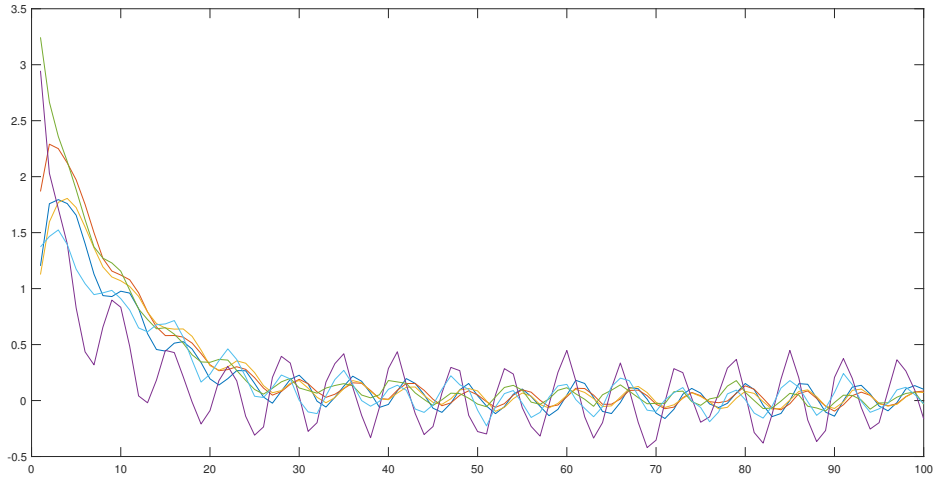


Figure 3.12: Fast sinusoid response for recursive

Fig 3.13 is a demonstration for the sinusoidal tracking performance of the continuous time state update algorithm, which is the pair of update equations numbered (3.2) and (3.6) on the second set of standard deviations. The signal that has been used here as an input is a slowly varying sinusoid.

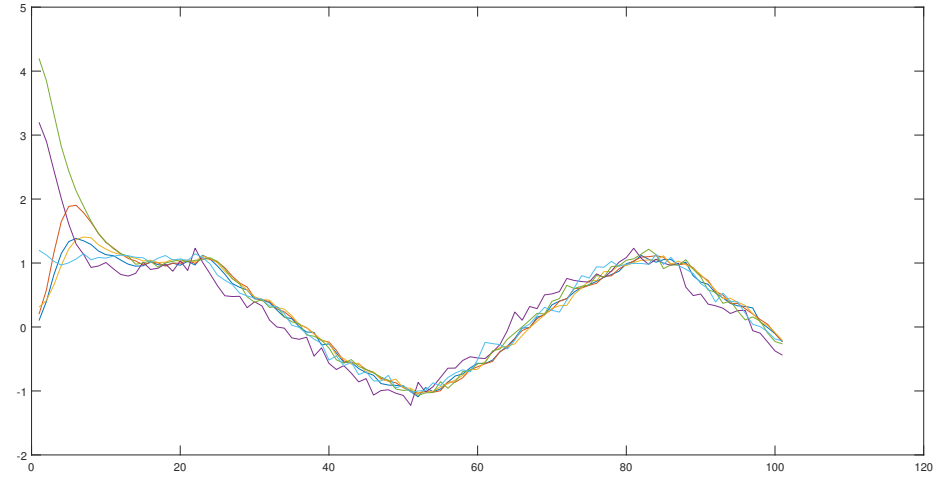


Figure 3.13: Slow sinusoid response for continuous time

Fig 3.14 is a demonstration for the sinusoidal tracking performance of the continuous

time state update algorithm, which is the pair of update equations numbered (3.2) and (3.6) on the second set of standard deviations. The signal that has been used here as an input is a fast varying sinusoid.

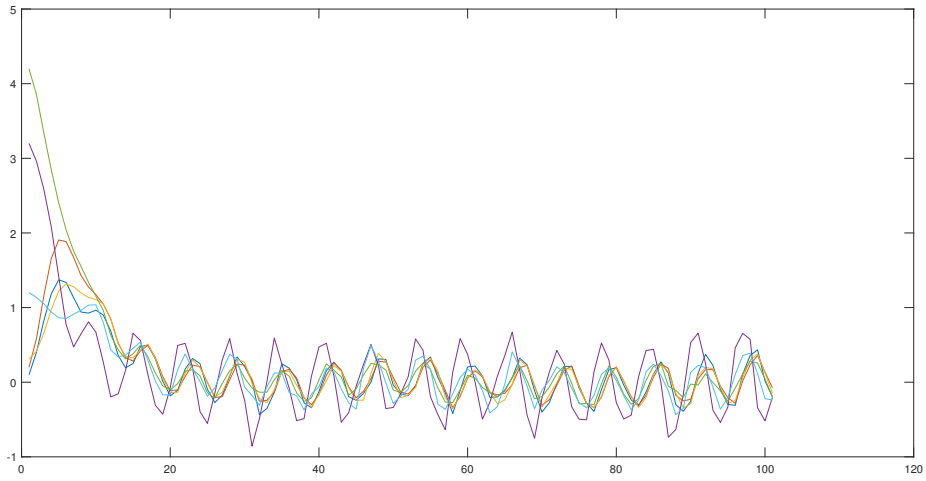


Figure 3.14: Fast sinusoid response for continuous

3.4 Low Pass Distributed Consensus Filter

Here we represent an example of consensus on estimates where, only state estimates are averaged out to reach the consensus state.

This particular section is with reference to the paper [17], where the authors have introduced an average consensus based distributed low-pass filter that makes the nodes of a sensor network track the average of n noisy sensor measurements. The sensing model of the network is the same as the previous analysis, i.e. :
$\xi_{j_0}(k) = x_0 + \delta_{j_0}$ where x_0 is the correct measurement and δ_{j_0} is a zero mean Gaussian noise i.e $\langle \delta_{j_0} \rangle = 0$

All mathematical notations and network configurations remain the same as have already been discussed in the beginning of this chapter. The only two changes are
in the definition of the set of sensor nodes which we choose for this particular algorithm as, $\mathbb{N}$={1,2,3,4,5,6}
and in the definition of indices used in describing the sensing nodes. In this case the indices used for sensing nodes are $[i,j]$.

The authors have proposed the following dynamic consensus algorithm for this purpose where each sensor node has been treated as an agent and as a result of this, the existing results of single integrator dynamics in multiagent systems can be directly applied to the distributed filters plus one extra consensus term to reflect the measurement features. (we formulate it in this particular way knowing that there are no self loops in the digraph):

$$\dot{x_i} = \sum_{\forall j \in \mathbb{N}} e_{ij}(x_i - x_j) + \sum_{\forall j \in \mathbb{N}} e_{ij}(\xi_i - x_j) \tag{3.9}$$

and in discrete time:

$$x_i(k+1) = x_i(k) + \delta[\sum_{\forall j \in \mathbb{N}} e_{ij}(x_i - x_j) + \sum_{\forall j \in \mathbb{N}} e_{ij}(\xi_i - x_j)] \tag{3.10}$$

with δ being the stepsize, which must be chosen with care to ensure the stability which in turn can be dependant on the structure of the network graph. The necessary and sufficient condition [21] for stability under arbitrary interconnection of the sensors is given as $\delta d_{max} < 1$, where d_{max} is the maximum node degree of the network. Hence, one natural choice for δ is $\delta = (1/1 + d_{max})$

The authors prove two important related results, namely:
Lemma:

$$\dot{x} = -(I_n + \Delta + L)x + (I_n + \mathcal{A})u \tag{3.11}$$

that is the filter is essentially an LTI system with a proper MIMO transfer function with strictly negative poles, given by

$$H(s) = [sI_n + (I_n + \Delta + L)]^{-1}(I_n + \mathcal{A}) \tag{3.12}$$

Since this kind of an algorithm does not require the information of local error covariance matrix or that of the local probability density functions, it has been commonly used for various consensus filter designs. For example in [1], the authors have designed a two-step consensus filtering strategy where the first step is a standard kalman estimation of state (which is local) and the second step is the consensus on those estimates. another example can be found in [10] where the authors employ a similar strategy computing a local state estimate using a Luenberger type observer.

3.5 Simulation

The above mentioned dynamic consensus algorithm on estimates was run for 100 iterations to track a theoretical measurement value of 3 and we used the same set of initial values for the demonstration of its consensus performance, namely:
Set of initial states x=[0.1,0.2,0.31,3.2,4.2,1.2]
Initial noisy measurements ξ =[0,0,0,3.7243,4.7243,2.7243]
Standard deviations at a node σ=[0,0,0,0.3,0.21,0.004]

and later it was changed to σ=[0,0,0,0.2,0.4,0.3]

We also demonstrate the tracking performance of this algorithm using a sinusoidal input for the second set of standard deviations.

3.5.1 Visualization

Fig 3.15 is for the discrete time dynamic consensus algorithm on estimates, which is the update equation numbered (3.10) on the first set of standard deviations.

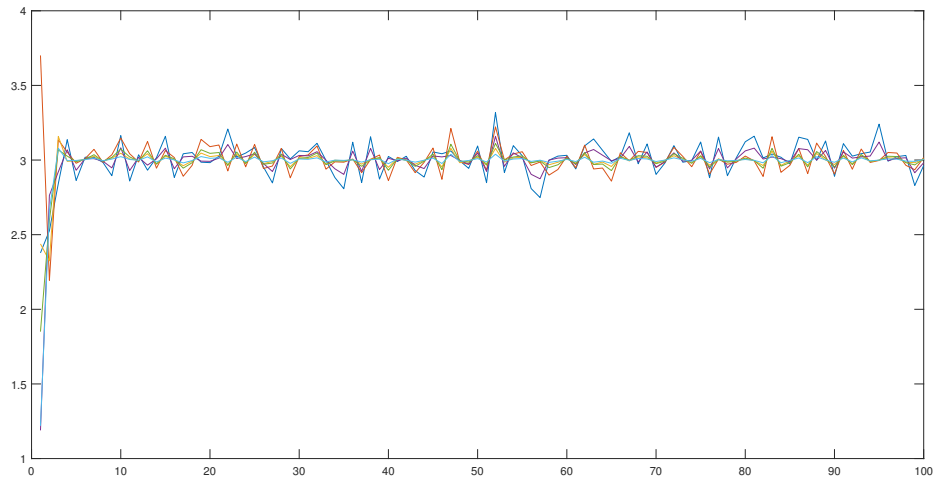


Figure 3.15: Discrete time consensus on estimates for SD 1

Fig 3.16 is for the discrete time dynamic consensus algorithm on estimates, which is the update equation numbered (3.10) on the second set of standard deviations.

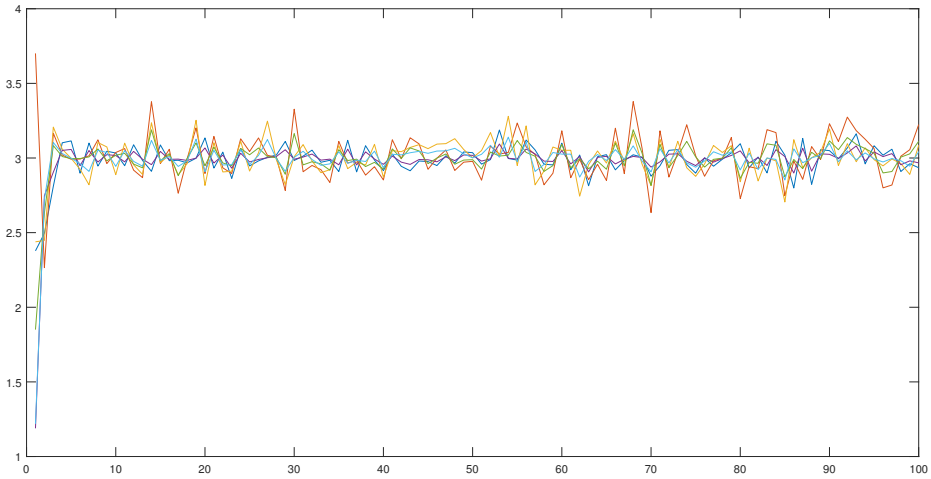


Figure 3.16: Discrete time consensus on estimates for SD 2

Fig 3.17 is for the continuous time dynamic consensus algorithm on estimates, which is the update equation numbered (3.9) on the first set of standard deviations.

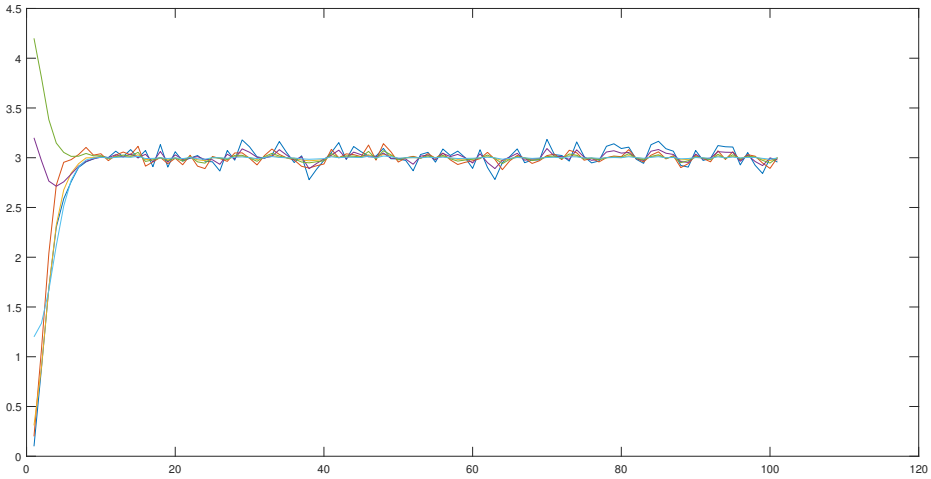


Figure 3.17: Continuous time consensus on estimates for SD 1

Fig 3.18 is for the continuous time dynamic consensus algorithm on estimates, which is the update equation numbered (3.9) on the second set of standard deviations.

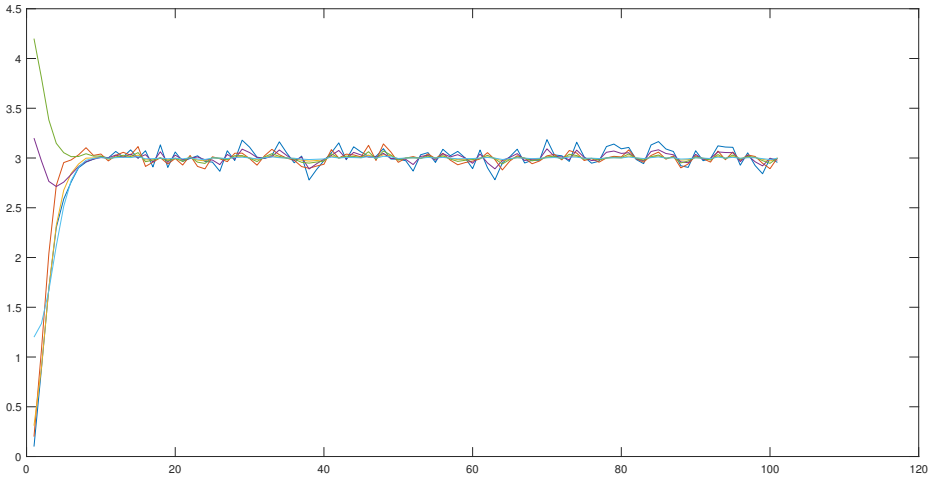


Figure 3.18: Continuous time consensus on estimates for SD 2

Fig 3.19 is a demonstration for the sinusoidal tracking performance of the discrete time dynamic consensus algorithm on estimates, which is the which is the update equation numbered (3.10) on the second set of standard deviations. The signal that has been used here as an input is a slow varying sinusoid.

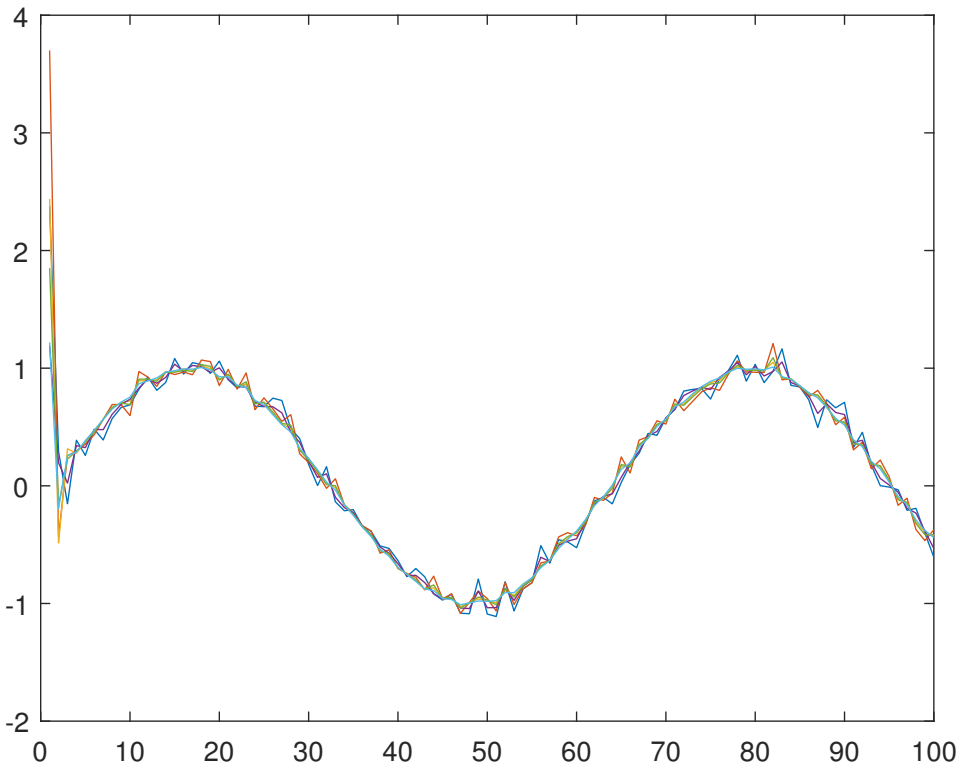


Figure 3.19: Slow sinusoid response for discrete time consensus on estimates

Fig 3.20 is a demonstration for the sinusoidal tracking performance of the discrete time dynamic consensus algorithm on estimates, which is the which is the update equation numbered (3.10) on the second set of standard deviations. The signal that has been used here as an input is a fast varying sinusoid.

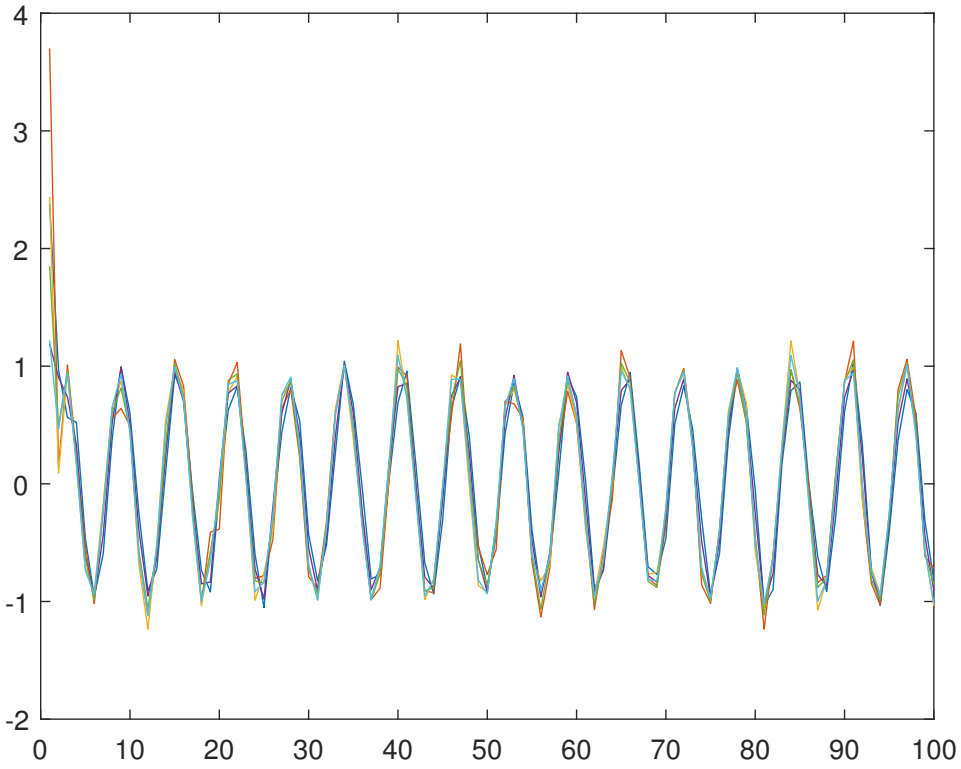


Figure 3.20: Fast sinusoid response for discrete time consensus on estimates

Fig 3.21 is a demonstration for the sinusoidal tracking performance of the continuous time dynamic consensus algorithm on estimates, which is the which is the update equation numbered (3.9) on the second set of standard deviations. The signal that has been used here as an input is a slow varying sinusoid.

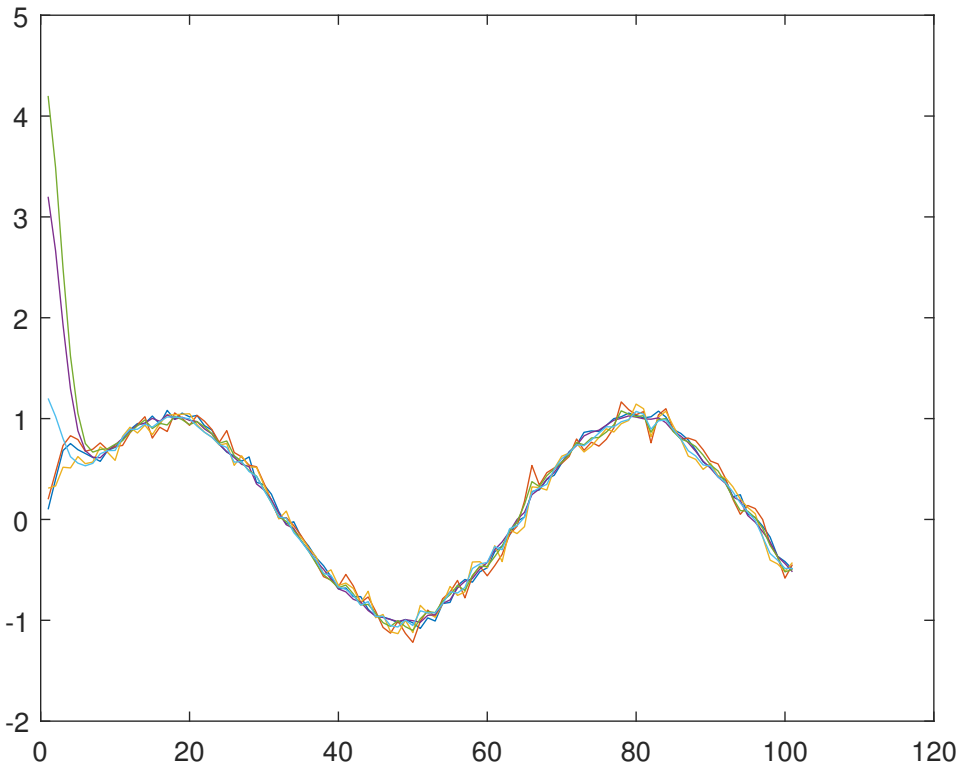


Figure 3.21: Slow sinusoid response for continuous time consensus on estimates

Fig 3.22 is a demonstration for the sinusoidal tracking performance of the continuous time dynamic consensus algorithm on estimates, which is the which is the update equation numbered (3.9) on the second set of standard deviations. The signal that has been used here as an input is a fast varying sinusoid.

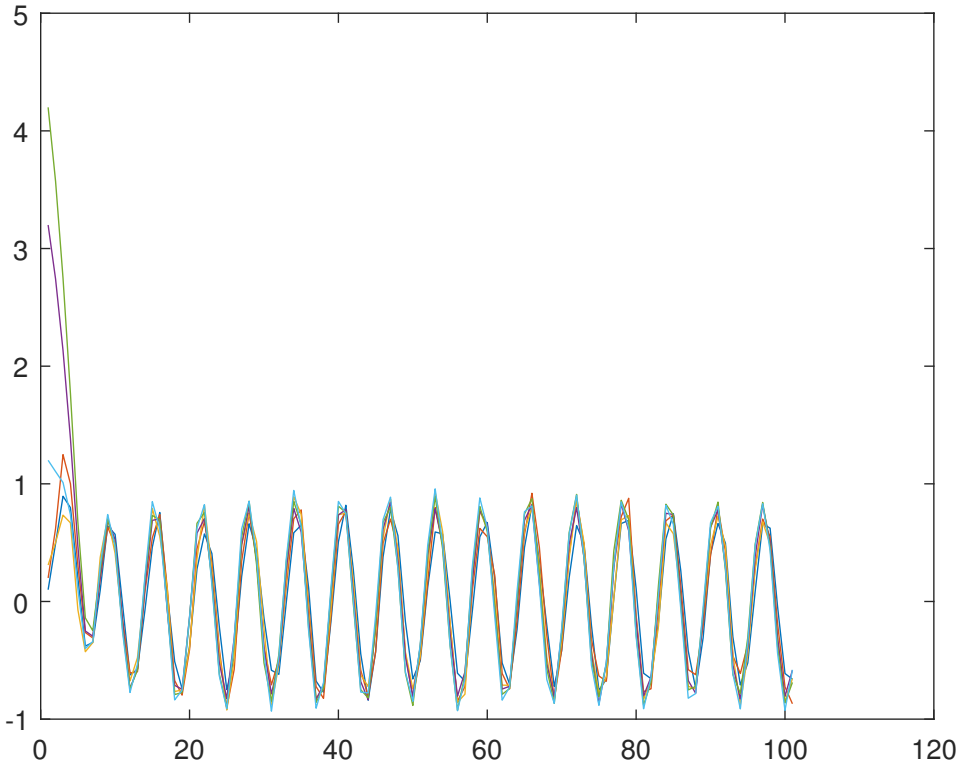


Figure 3.22: Fast sinusoid response for continuous time consensus on estimates

3.6 Luenberger Observer

In this section we modify the distributed Luenberger observer that was proposed in [14] which we proved to be a valid consensus algorithm for a directed network graph as well and try to implement the Bayesian fusion by the straightforward method of modifying the edge weights so as to reflect the Bayesian decision making process. The Luenberger observer is of the form

$$\dot{\hat{x}_i} = A\hat{x_i} + K_i(z_i - H_i\hat{x_i}) + \gamma P_i \sum_{j \in N_i} e_{ij}(\hat{x_j} - \hat{x_i}) \tag{3.13}$$

$$K_i = P_i H_i^T R_i^{-1}, \gamma > 0$$

$$\dot{P}_i = AP_i + P_iA^T + BQB^T - K_iR_iK_i^T$$

with a Kalman consensus estimator and initial conditions $P_i(0) = P_0$ and $\dot{\hat{x}}_i = x(0)$
In order to appropriately weigh the edges for a Bayesian Consensus without affecting the condition that the graph is strongly connected, we make the following changes:

$$K_i \longrightarrow K_i(1 + 1/\sigma_i^2)$$

We multiply the local state update gain K_i with $(1 + 1/\sigma_i^2)$ where σ_i^2 is the variance of measurement at node i

$$e_ij \longrightarrow [(\sum_{k\neq i} 1/\sigma_k^2)/(\sigma_i^2 + \sum_{k\neq i} 1/\sigma_k^2)]e_{ij}$$

We multiply the $e_{ij} \in 0, 1$ with $(\sum_{k\neq i} 1/\sigma_k^2)/(\sigma_i^2 + \sum_{k\neq i} 1/\sigma_k^2)$

Over here, we see that the Bayesian weights have been chosen in such a manner so as to make the algorithm trust the local update term more than the consensus term when the variance of measurement at the node being considered is lower compared to the network variance, at the same time if the network variance of measurement is lower compared to the variance of measurement at the node of consideration then the algorithm trusts the network more, and the measurement at the sensor less.

3.7 Simulation

The Luenberger observer described above was run on a 6-node digraph network configuration (following the proof) which is the same as the one that was chosen for the Bayesian consensus filter in the beginning of this chapter. The only change being, all nodes are now considered as sensor nodes. For the first two simulations, we are tracking a process of the form
$z = x_0 + \delta$ where x_0 is a constant and δ is a zero mean Gaussian noise with a covariance R. Hence, $\dot{x} = 0$

and this gives us the following set of starting values,
$A = [0]$, $B = [0]$,$Q = [0]$,$H = [1]$,$R = \sigma^2$,$P_0 = 1$ and a random initial state
$x(0)$=$[90.1, 8.6, 2.1, 1.9, 3.8, 19]$

We observe the convergence of the observer with a constant measurement value of 3 at first and then we try a sinusoidally varying measurement value. We choose the set of initial states x=[90.1, 8.6, 2.1, 1.9, 3.8, 19]
Initial noisy measurements ξ =[1.8,2.1,0.3.7243,4.7243,2.7243]
Standard deviations at a node σ=[11.4, 3.8, 4.6, 4.3, 8.2,13.4]

and later it was changed to σ=[10,13,14,4.5,8.5,8]

3.7.1 Visualization

Fig 3.23 is a demonstration of convergence for the Luenberger observer on the first set of standard deviations. We note that the rate of convergence to the theoretical measurement value is extremely slow.

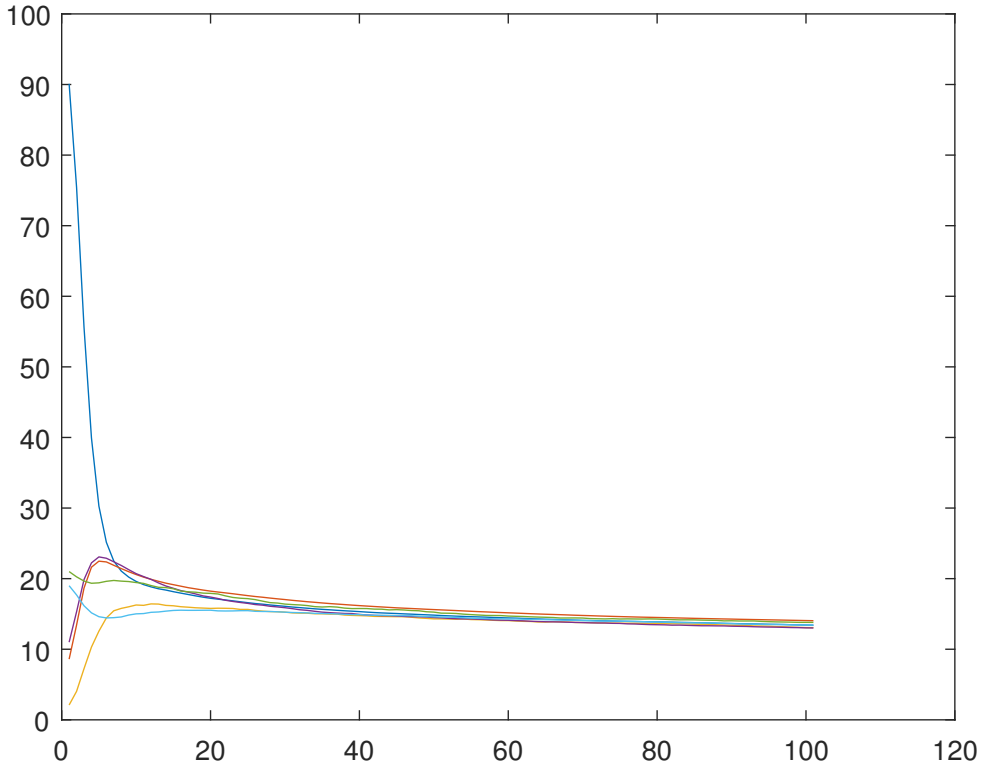


Figure 3.23: Luenberger observer on SD 1

Fig 3.24 is a demonstration of convergence for the Luenberger observer on the second set of standard deviations. We note that the rate of convergence to the theoretical measurement value is extremely slow.

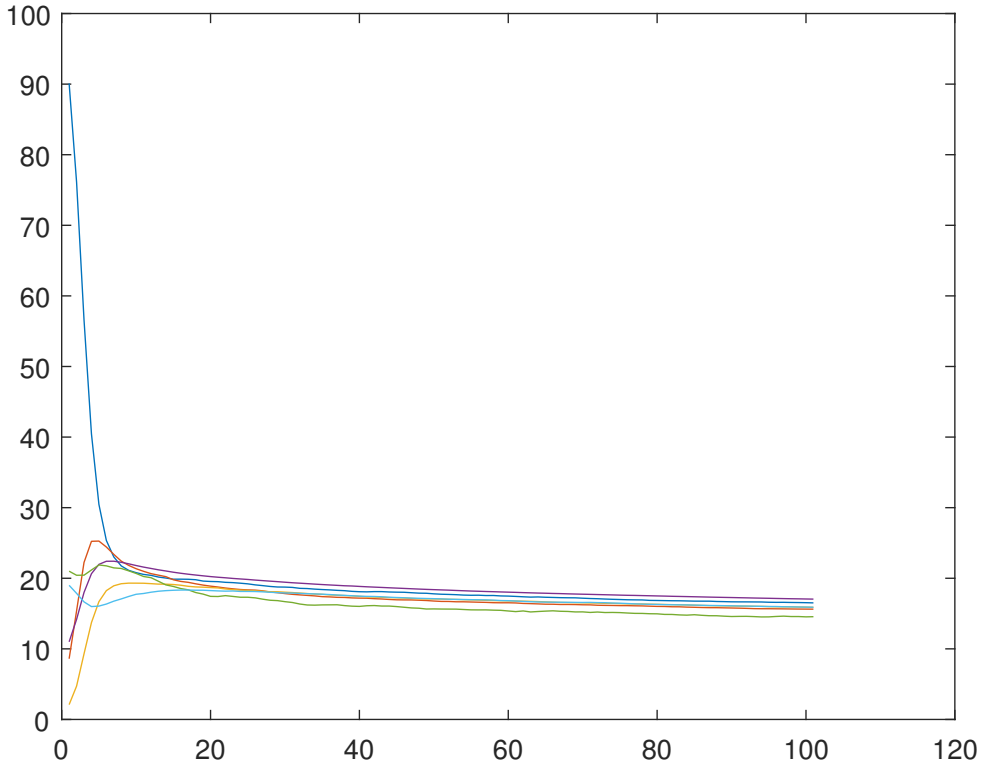


Figure 3.24: Luenberger observer on SD 2

Fig 3.25 is a demonstration of the behaviour of the Luenberger observer on the second set of standard deviations for a theoretical measurement value which is a slowly varying sinusoid. We note that the Luenberger observer by itself possesses little to zero tracking ability for a varying measurement signal.

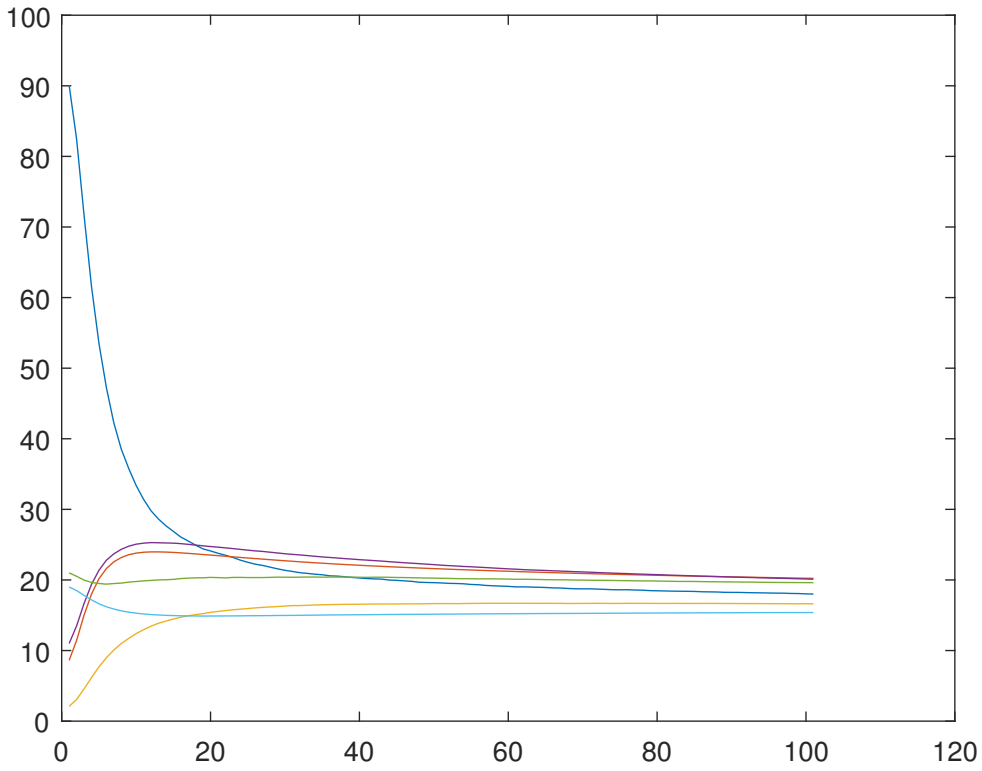


Figure 3.25: Slow sinusoid response of luenberger observer

Fig 3.26 is a demonstration of the behaviour of the Luenberger observer on the second set of standard deviations for a theoretical measurement value which is a fast varying sinusoid. We note that the Luenberger observer by itself possesses little to zero tracking ability for a varying measurement signal.

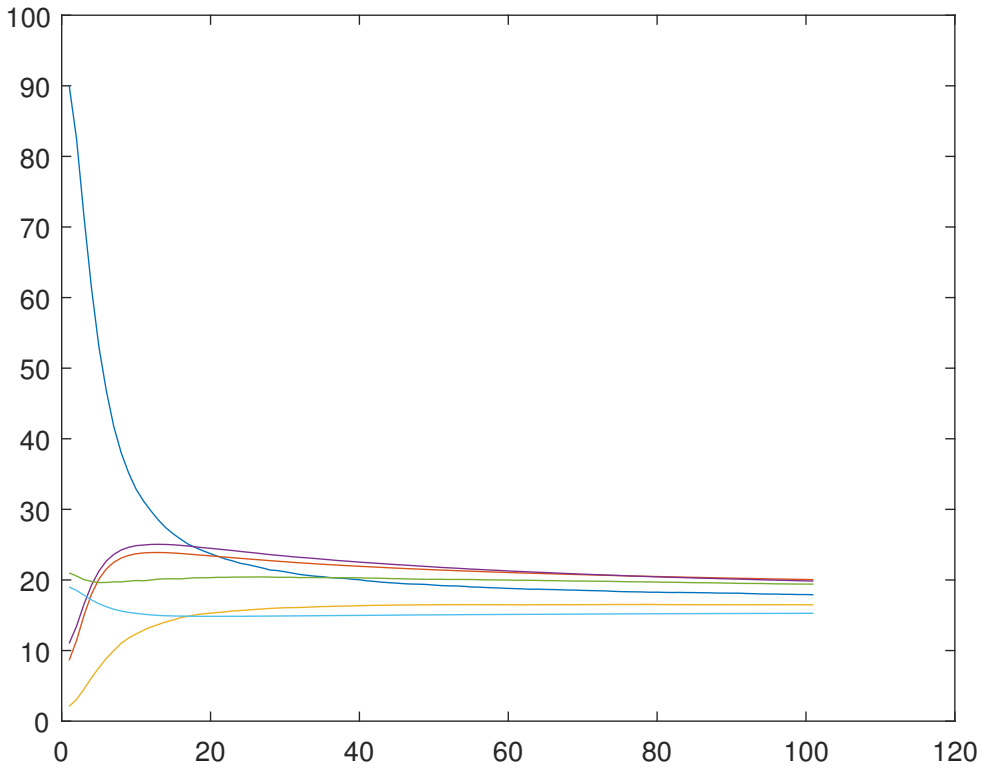


Figure 3.26: Fast sinusoid response for luenberger observer

3.8 Conclusion

In this chapter we see the relative consensus performance of consensus algorithms based various strategies, namely: Bayesian Consensus, Consensus on estimates and a Luenberger estimator. We see that the performance of the first two strategies have been noticeably better than the last one in the test cases that we have considered. In the next section we go into a deeper analysis of the results.

Chapter 4

Statistical Observations

4.1 Bayesian Consensus Fusion on a Strongly Connected Digraph (where sensors are not communicating to each other)

Here we observe the statistics of the outputs for the Bayesian consensus algorithms. The algorithms that are to be tested in this section are the following:

Algorithm set A (Discrete time, non-recursive sensor state update):

$$x_i(k+1) = [x_i(k) + \sum_j e_{ij}x_j + \{\sum_{j_0} g_{ij_0}x_{j_0}/\sigma^2_{j_0}/\sum_{j_0} g_{ij_0}/\sigma^2_{j_0}\}]/(2+d_i) \tag{4.1}$$

$$\begin{aligned} x_{j_0}(k+1) =&[(\xi_{j_0}(k)/\sigma^2_{j_0}) + (\sum_{k_0\neq j_0} 1/\sigma^2_{k_0})(\sum_l h_{j_0 l}x_l(k) \\ &/\sum_l h_{j_0 l})/\{(1/\sigma^2_{j_0}) + \sum_{k_0\neq j_0} 1/\sigma^2_{k_0}\}] \end{aligned} \tag{4.2}$$

Algorithm set B (Discrete time, recursive sensor state update):

$$x_i(k+1) = [x_i(k) + \sum_j e_{ij}x_j + \{\sum_{j_0} g_{ij_0}x_{j_0}/\sigma^2_{j_0}/\sum_{j_0} g_{ij_0}/\sigma^2_{j_0}\}]/(2+d_i) \tag{4.3}$$

$$\begin{aligned} x_{j_0}(k+1) =&1/2[(\xi_{j_0}(k)/\sigma^2_{j_0}) + (\sum_{k_0\neq j_0} 1/\sigma^2_{k_0})(\sum_l h_{j_0 l}x_l(k) \\ &/\sum_l h_{j_0 l})/\{(1/\sigma^2_{j_0}) + \sum_{k_0\neq j_0} 1/\sigma^2_{k_0}\} + x_{j_0}(k)] \end{aligned} \tag{4.4}$$

Algorithm set C (Continuous time):

$$\dot{x_i} = -d_i x_i + \sum_j e_{ij} x_j + \{\sum_{j_0} g_{ij_0} x_{j_0}/\sigma_{j_0}^2 / \sum_{j_0} g_{ij_0}/\sigma_{j_0}^2\} - x_i \tag{4.5}$$

$$\dot{x} = [1/(\sigma_{j_0}^2) + (\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)]^{-1}[(\sum_{k_0 \neq j_0} 1/\sigma_{k_0}^2)(\sum_l h_{j_0 l})^{-1}(\sum_l h_{j_0 l}(x_l - x_{j0})) + (\xi_{j_0}/\sigma_{j_0}^2)] \tag{4.6}$$

4.2 Recursive vs Non Recursive/Process Variance

To make a comparison between the three sets of sensor state update algorithms, the variances at different nodes (after the transients approximately died down) has been compared.

- Using the recursive sensor state update algorithm (A) for a set of large standard deviations (i.e generally noisy sensors) like σ=[0,0,0,0.2,0.4,0.3] the variances at the nodes 1 to 6 were observed to be
  σ^2=[0.0015,0.0014,0.0013,0.0059,0.0026,0.0027]
  A visual example of such an evolution of states is in Fig.3.6
  Using the recursive sensor state update algorithm for a set of very low standard deviations (i.e very precise sensors) like σ=[0,0,0,0.002,0.004,0.003] the variances at the nodes 1 to 6 were observed to be
  σ^2=[0.1033,0.1518,0.1597,0.0227,0.1410,0.1093] *1.0e-03
  A visual example of such an evolution of states is in Fig.4.1

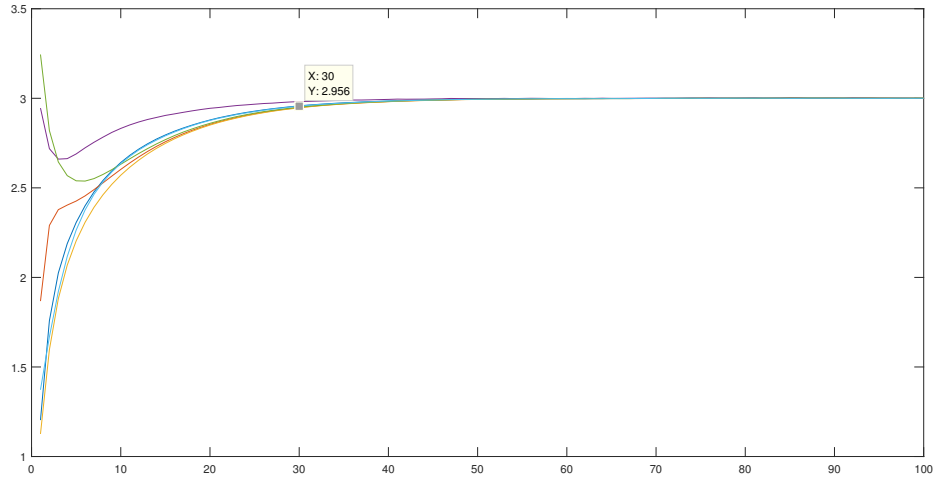


Figure 4.1: Recursive discrete time bayesian on low SD

- Using the non recursive sensor state update algorithm for a set of high standard deviations (i.e generally noisy sensors) like σ=[0,0,0,0.2,0.4,0.3] the variances at the nodes 1 to 6 were observed to be
  σ^2=[0.0030,0.0035,0.0025,0.0117,0.0080,0.0069]
  A visual example of such an evolution of states is in Fig.3.5
  Using the non recursive sensor state update algorithm for a set of very low standard deviations (i.e very precise sensors) like σ=[0,0,0,0.002,0.004,0.003] the variances at the nodes 1 to 6 were observed to be
  σ^2=[0.1468,0.2080,0.2120,0.0288,0.1606,0.1226] *1.0e-04
  A visual example of such an evolution of states is in Fig.4.2

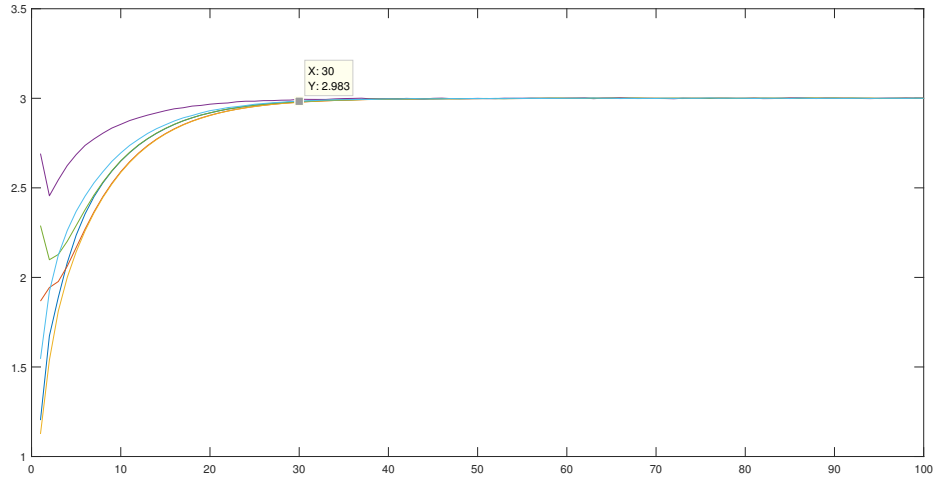


Figure 4.2: Non recursive discrete time bayesian on low SD

Using the continuous time state update algorithm for a set of high standard deviations (i.e generally noisy sensors) like σ=[0,0,0,0.2,0.4,0.3] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.0053,0.0030,0.0038,0.0158,0.0046,0.0073]
A visual example of such an evolution of states is in Fig.3.7 and 3.8
Using the continuous time state update algorithm for a set of very low standard deviations (i.e very precise sensors) like σ=[0,0,0,0.002,0.004,0.003] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.1362,0.1704,0.1461,0.1537,0.1668,0.1071] *1.0e-05
A visual example of such an evolution of states is in Fig.4.3

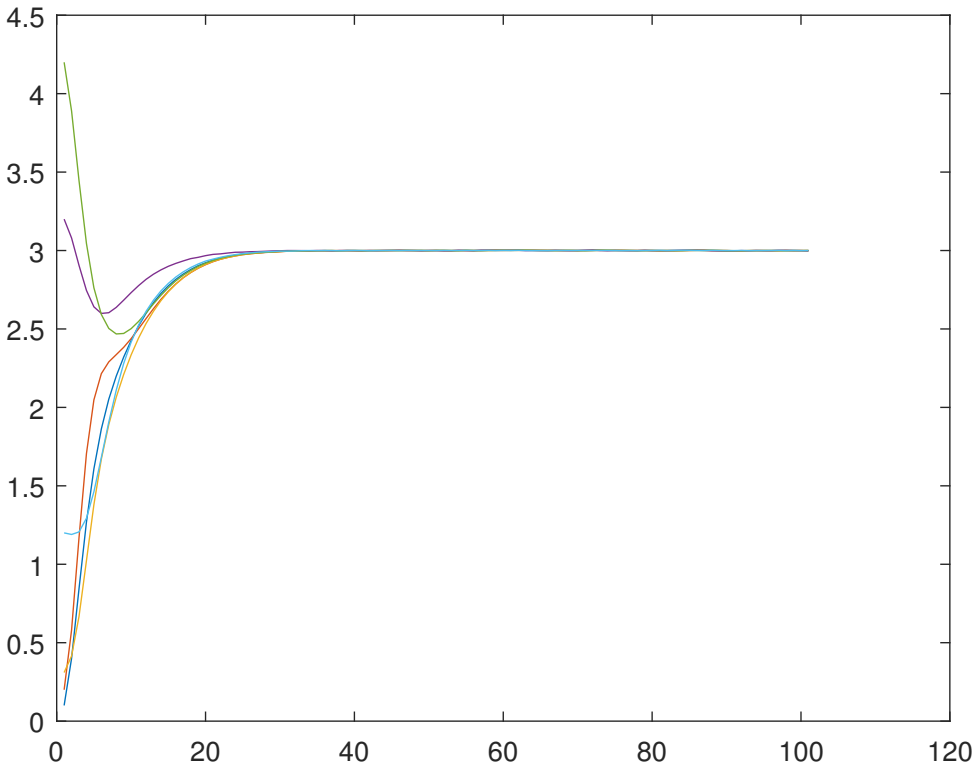


Figure 4.3: Continuous time bayesian state update on low SD

4.2.1 Discussion on the results

- What can be seen from the above set of observations is that when dealing with noisy sensors, the recursive state update algorithm set B for the sensing nodes is clearly superior in case of the nodes 4 through 6 i.e the sensing nodes where the variances are almost half of what can be observed while using the non recursive state update algorithm set A. It is also clearly superior when its interaction with the non-sensing state update algorithm is considered, which can be seen as an almost two fold increase in variance in the nodes 1 through 3 i.e the non-sensing nodes, when the non-recursive algorithm is applied.

- Interestingly however, the non recursive sensor state update algorithm outperforms the recursive sensor state update algorithm as can be seen by a difference in variances between the two which is almost a couple of orders of magnitude when the sensors are extremely precise.

- As expected, the process variance of the continuous time state update algorithm set C is very similar to that of the non-recursive discrete time state update algorithm set A.

4.2.2 Coupling

As a way to visualize the coupling between the states for both the recursive and the non recursive versions of the state update algorithm, we use the MATLAB functions *xcorr()*, which returns a matrix whose columns contain the autocorrelation and cross-correlation sequences for all combinations of the columns of the input matrix and *xcov()*, which returns a matrix whose columns contain the autocovariance and cross-covariance sequences for all combinations of the columns of the input matrix. In this case, an input matrix that contains as its columns the sequence of state updates for each node of the graph for a set of standard deviations σ=[0,0,0,0.2,0.4,0.3] was used. Only the last 70 iterations were used (out of the 100 iterations for which the code was run) for the purpose of disregarding the transient states. Each such column has one autocorrelation/autocovariance sequence and 5 cross-correlation/cross-covariance sequences. Therefore the output matrices have 36 columns and 139 rows.
Fig.4.4 and Fig.4.5 are the plots for the outputs of the *xcov()* function and the *xcorr()* respectively when the recursive state update algorithm is used. It can be seen that the covariance and correlation becomes the strongest while approaching a lag of zero, that is the end of the sequence.

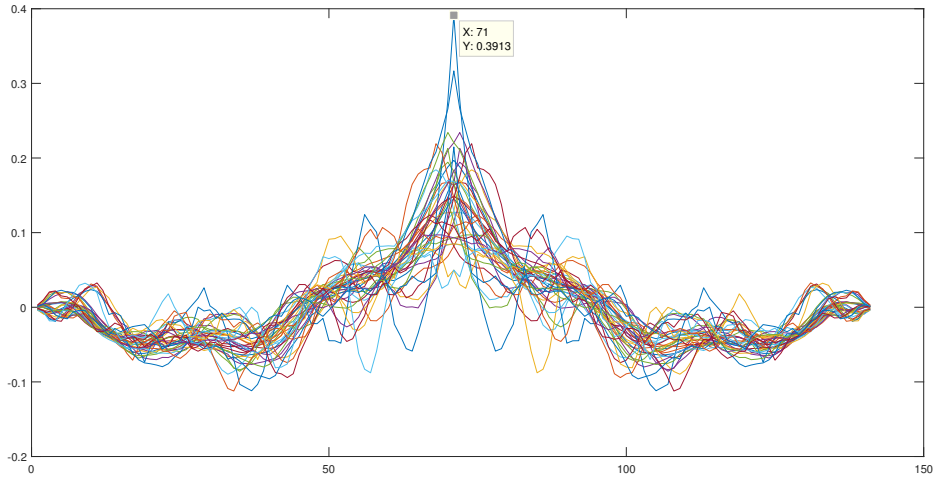


Figure 4.4: Crosscovariance on DT recursive

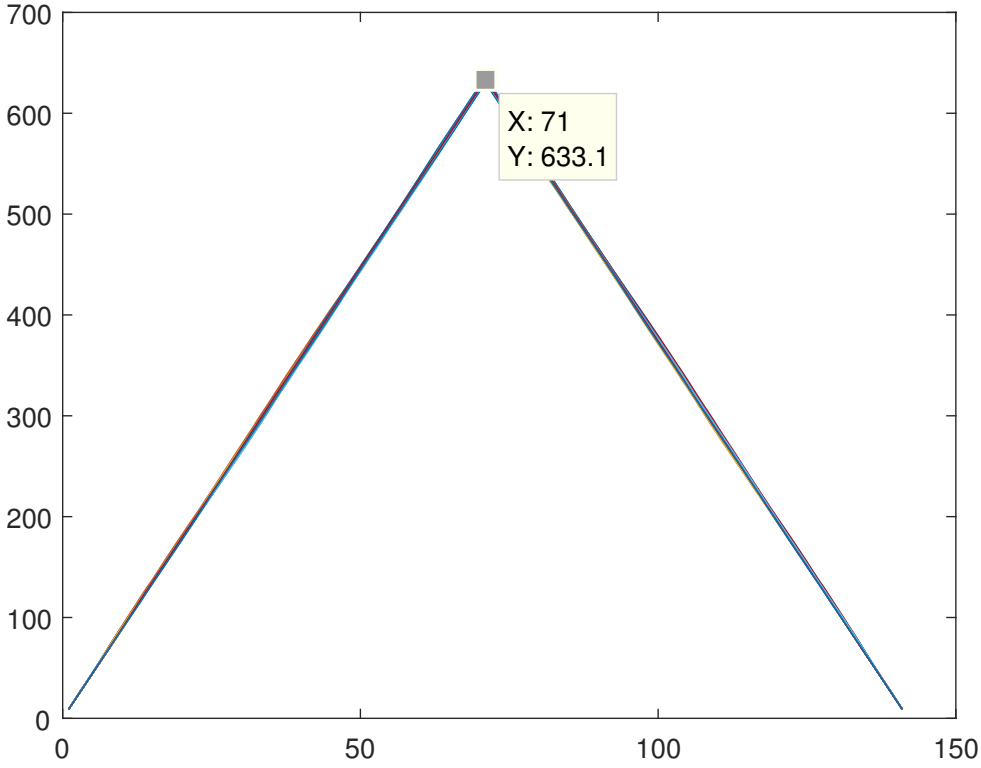


Figure 4.5: Crosscorrelation on DT recursive

Fig.4.6 and Fig.4.7 are the plots for the outputs of the *xcov()* function and the *xcorr()* respectively when the non-recursive state update algorithm is used.
It can be observed here that the non recursive algorithm gives rise to 'less similar' state update sequences when compared to the state update sequences of the recursive version.
Here too it is seen that the covariance and correlation becomes the strongest while approaching a zero lag, which is the end of the sequence.

Fig.4.8 and Fig.4.9 are the plots for the outputs of the *xcov()* function and the *xcorr()* respectively when the continuous time state update algorithm is used.
It can be observed here that it has coupling characteristics very similar to the non-recursive discrete time state update algorithm
For two time series x_t and y_t we can calculate the true cross-covariance as

$$\sigma_{xy}(T) = 1/(N-1)\sum_{t=1}^{N}(x_{t-T} - \bar{x})(y_t - \bar{y})$$

where $\bar{x}$=mean of time series x and $\bar{y}$=mean of time series y
at zero lag,

$$\sigma_{xy}(0) = 1/(N-1)\sum_{t=1}^{N}(x_t - \bar{x})(y_t - \bar{y})$$

and when there is autocovariance,

$$\sigma_{xx}(0) = 1/(N-1)\sum_{t=1}^{N}(x_t - \bar{x})^2$$

which is the variance of the time series.
In MATLAB the *xcov()* function computes raw covariance with no normalization. That is, to get the true covariance the output must be divided by N-1. For expository purpose, the 70^{th} row of the cross covariance matrix is taken (which corresponds to the zero lag) and is arranged into a 6x6 matrix (by using the *reshape()* function in MATLAB) whose elements are arranged in the following way where [A,B,C,D,E,F] correspond to the columns containing the states of nodes 1-6 for any given number of iterations for any version of the algorithm (the pair of update equations numbered (3.4) and (3.8) has been used for this particular exposition)

$$xcov_0 = \begin{bmatrix} AA & AB & AC & AD & AE & AF \\ BA & BB & BC & BD & BE & BF \\ CA & CB & CC & CD & CE & CF \\ DA & DB & DC & DD & DE & DF \\ EA & EB & EC & ED & EE & EF \\ FA & FB & FC & FD & FE & FF \end{bmatrix}$$

Here, the letter sequence XY where X and Y $\in [A, B, C, D, E, F]$ corresponds to the expression

$$\sigma_{xy}(0) = \sum_{t=1}^{N} (x_t - \bar{x})(y_t - \bar{y})$$

where the symbols have their usual meaning.

The letter sequence XX where X $\in [A, B, C, D, E, F]$ corresponds to the expression

$$\sigma_{xx}(0) = \sum_{t=1}^{N} (x_t - \bar{x})^2$$

Which indicates towards the conclusion that upon normalizing the $xcov_0$ matrix by dividing it by N-1 (where N=70), the diagonal elements of the resulting matrix will be equal to the variance of the process at the corresponding nodes.

Such a matrix aids in clearly visualizing the coupling of the sensing nodes with the non sensing nodes even when the network structure is unavailable. The position of the maximally correlated elements coincide with the position of the connected elements in the adjacency matrix. One such matrix obtained is presented below,

$$Nxcov_0 = \begin{bmatrix} 7 & 4 & 2 & 5 & 1 & 0 \\ 4 & 5 & 2 & 4 & 4 & 1 \\ 2 & 2 & 4 & 1 & 1 & 3 \\ 5 & 4 & 1 & 40 & 1 & -1 \\ 1 & 4 & 1 & 1 & 13 & 0 \\ 0 & 1 & 3 & -1 & 0 & 14 \end{bmatrix} * 10^{-4}$$

where the variance at the nodes, obtained using the *var()* function is

$$\sigma^2 = [0.0007, 0.0005, 0.0004, 0.0040, 0.0013, 0.0014]$$

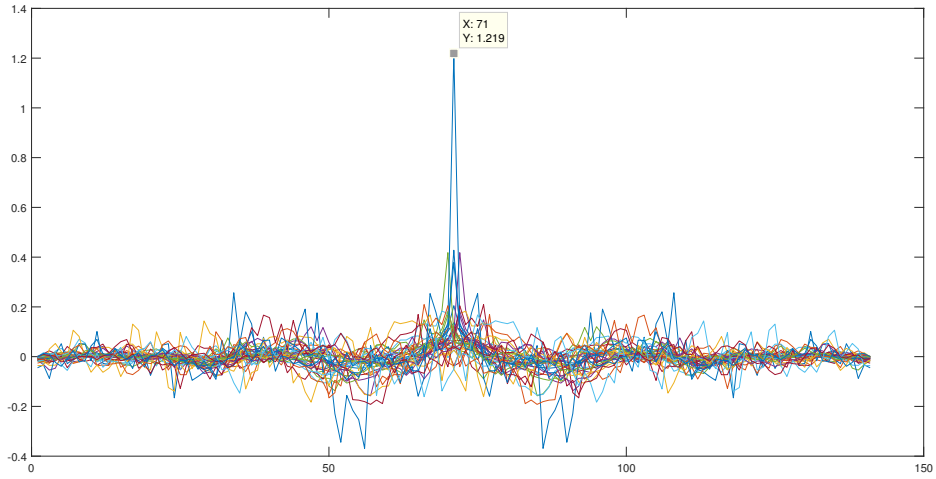


Figure 4.6: Crosscovariance on DT non recursive

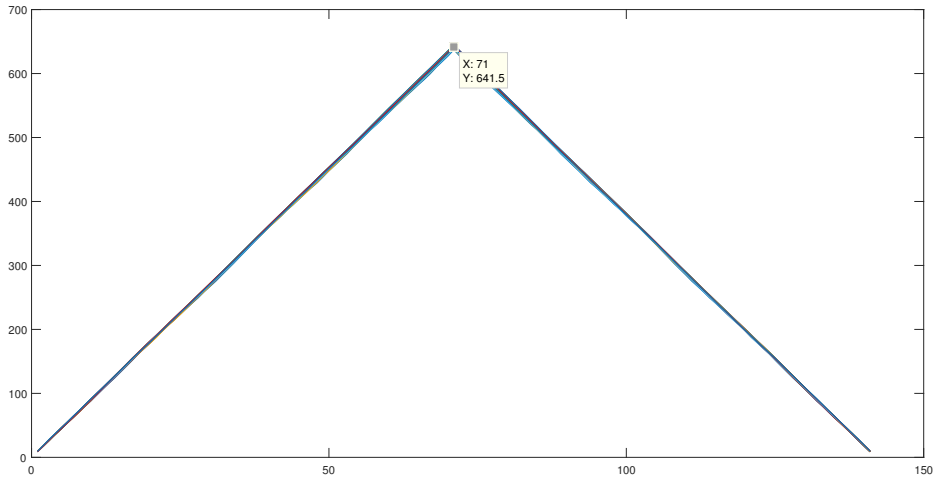


Figure 4.7: Crosscorrelation on DT non recursive

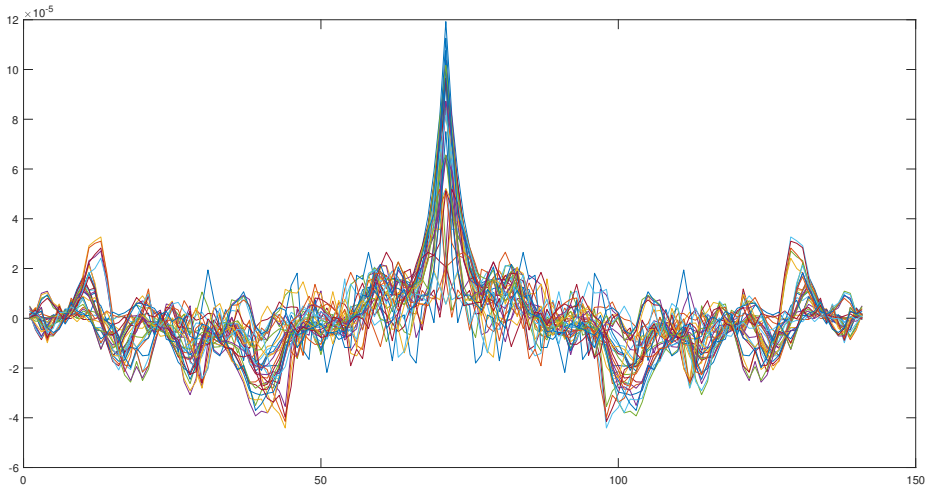


Figure 4.8: Crosscovariance on CT state update

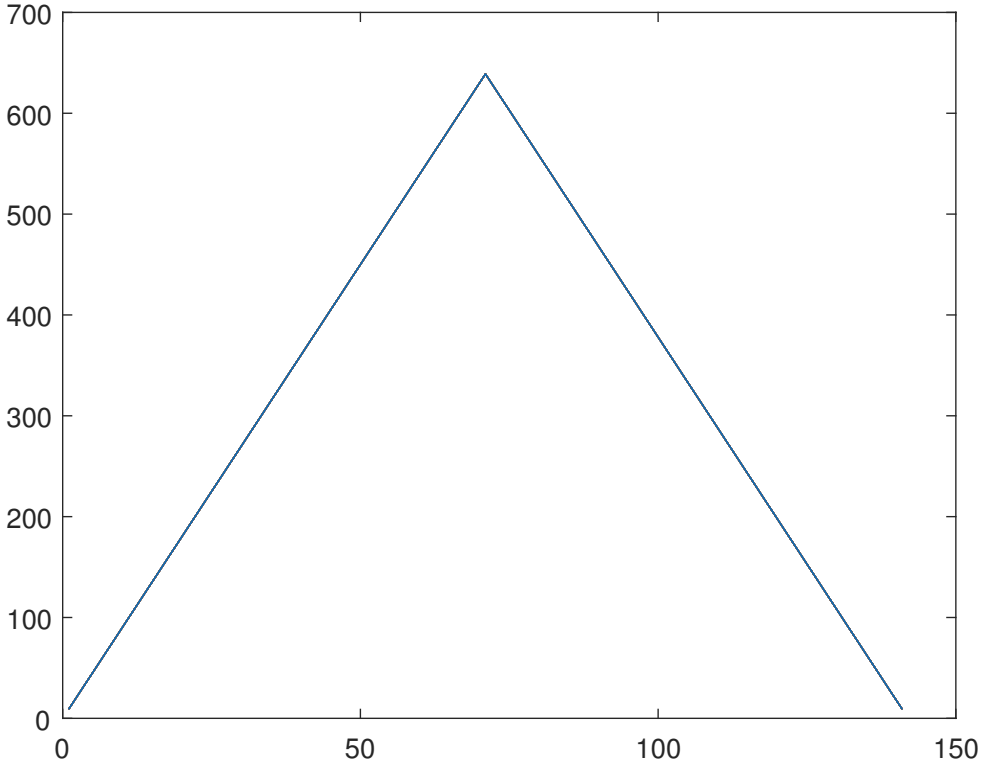


Figure 4.9: Crosscorrelation on CT state update

4.2.3 Noise output

- As a last step to visualize the random noise generation at the output, for each node the state values for a very noisy sensor were subtracted from the state values of a very precise sensor for a very large number of steps (1000 in this case)to get the noise and then a check was run to see if the noise is indeed Gaussian. For this purpose the MATLAB command *normplot()* was used and as a sample the results for node 1 (Fig.4.10) and node 4 (Fig.4.11) are presented indicating that the noise indeed is Gaussian for both the sensing and the non sensing nodes

- Another interesting observation is obtained when the cross-correlation and cross-covariance matrices of the Gaussian noise matrix obtained in the previous step is evaluated using MATLAB

  Fig.4.12 and Fig.4.13 shows that the output Gaussian noise has a degree of cross-covariance and cross-correlation (in that order) with peaks at a lag of zero as well. It is consistent with the fact that when white noise is an input to a linear system, the output is a correlated Gaussian noise whose autocorrelation sequence is not a Dirac delta function.

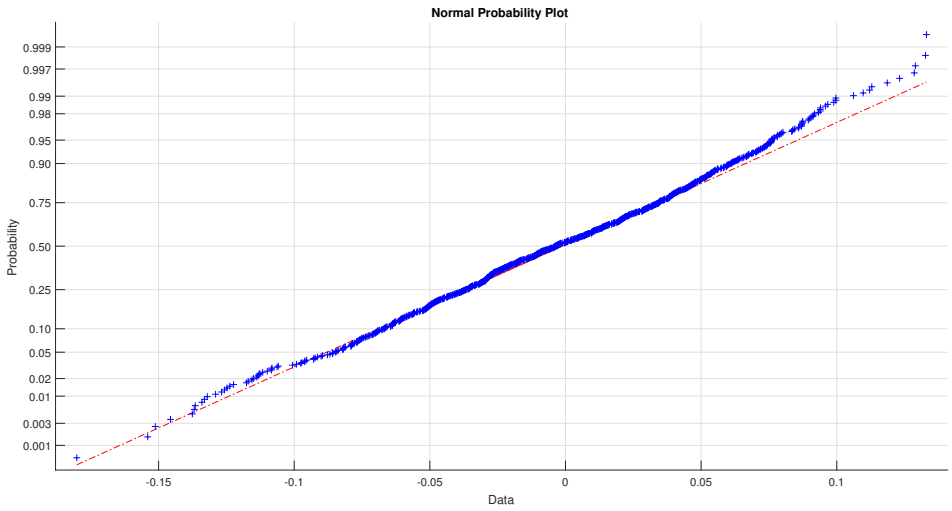


Figure 4.10: Normplot for non-sensing node 1

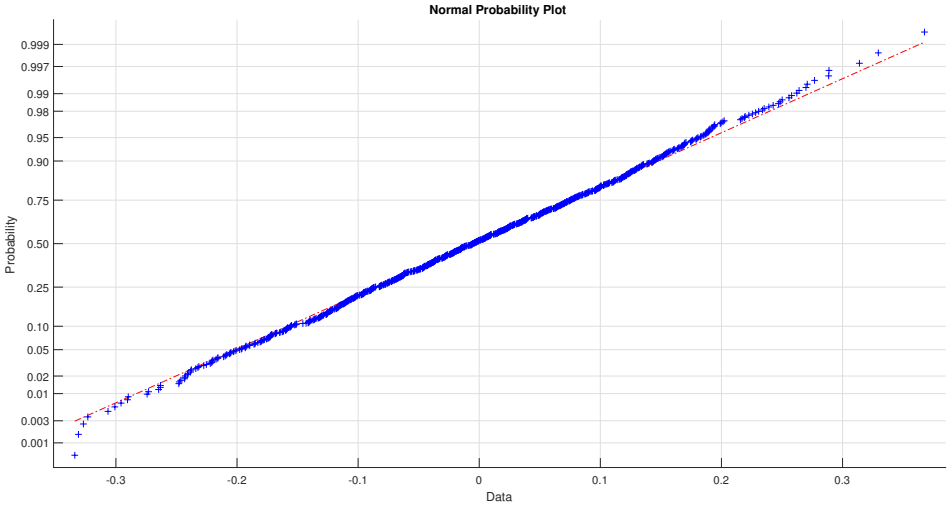


Figure 4.11: Normplot for sensing node 4

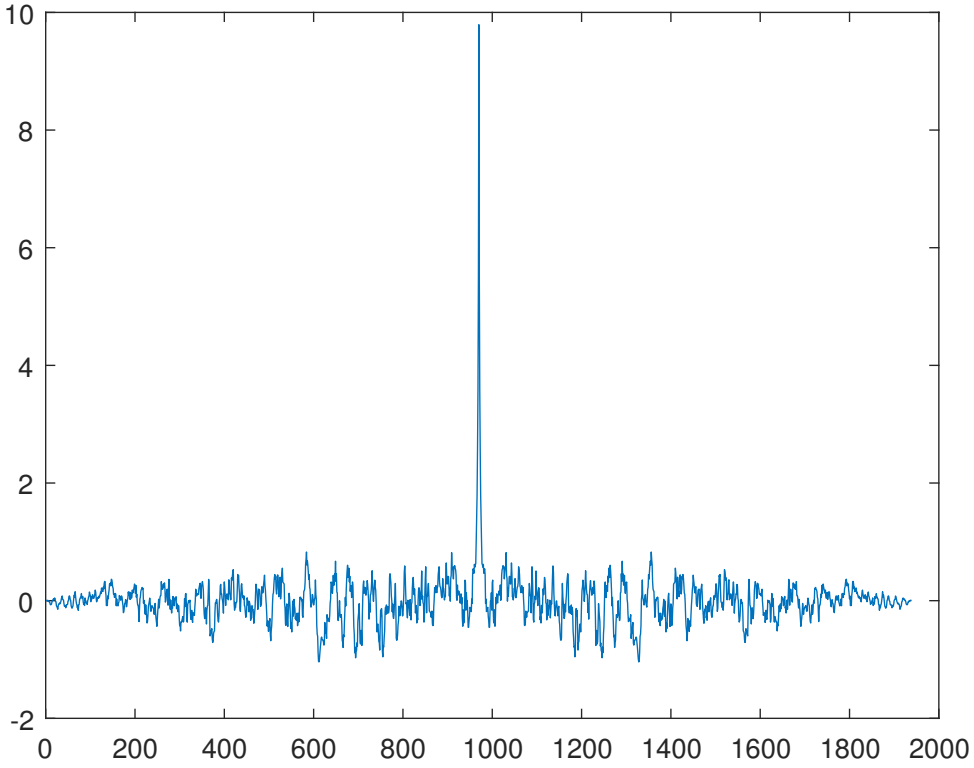


Figure 4.12: Crosscovariance for noise output at node 1

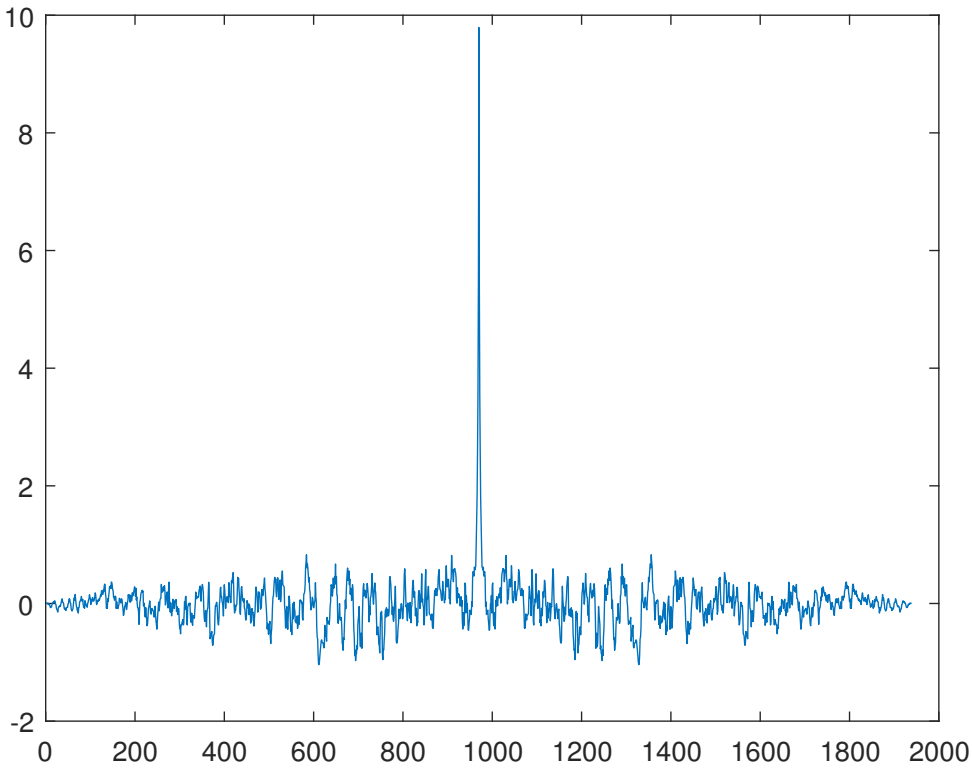


Figure 4.13: Crosscorrelation for noise output at node 4

-

4.2.4 General discussion

In general for the above set of observations and analysis, we can conclude that the recursive version of the sensor state update algorithm performs better for noisy sensors where as it performs much worse but still excellent in case of very precise sensors. As future work, a further addition to the simulation code itself can be an implementation of Prim's algorithm to find the root of the spanning tree given the adjacency matrix. In case of a large adjacency matrix, this will become necessary for determining the possible position of the pinning nodes.

4.3 Generalized Consensus on Estimates for n Noisy Sensors in a Communication Network

In this section we observe the output statistics for the consensus algorithm on estimates. The algorithms that are to be tested in this section are the following:

Algorithm set D (Discrete time):

$$x_i(k+1) = x_i(k) + \delta[\sum_{\forall j \in \mathbb{N}} e_{ij}(x_i - x_j) + \sum_{\forall j \in \mathbb{N}} e_{ij}(\xi_i - x_j)] \tag{4.7}$$

Algorithm set E (Continuous time):

$$\dot{x_i} = \sum_{\forall j \in \mathbb{N}} e_{ij}(x_i - x_j) + \sum_{\forall j \in \mathbb{N}} e_{ij}(\xi_i - x_j) \tag{4.8}$$

4.4 Process Variance

We observe the process variance of the two sets of algorithms in this section in the same way as it was observed in section 4.2 earlier in the chapter.

Running algorithm set D on a set of large standard deviations(i.e generally noisy sensors) like σ=[0,0,0,0.2,0.4,0.3] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.0060,0.0177,0.0157,0.0013,0.0044,0.0039]
A visual example of such an evolution of states is in Fig.3.16
Using the algorithm set D for a set of very low standard deviations (i.e very precise sensors) like σ=[0,0,0,0.002,0.004,0.003] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.0641,0.1781,0.1173,0.0160,0.0445,0.0293] *1.0e-05
A visual example of such an evolution of states is in Fig.4.14

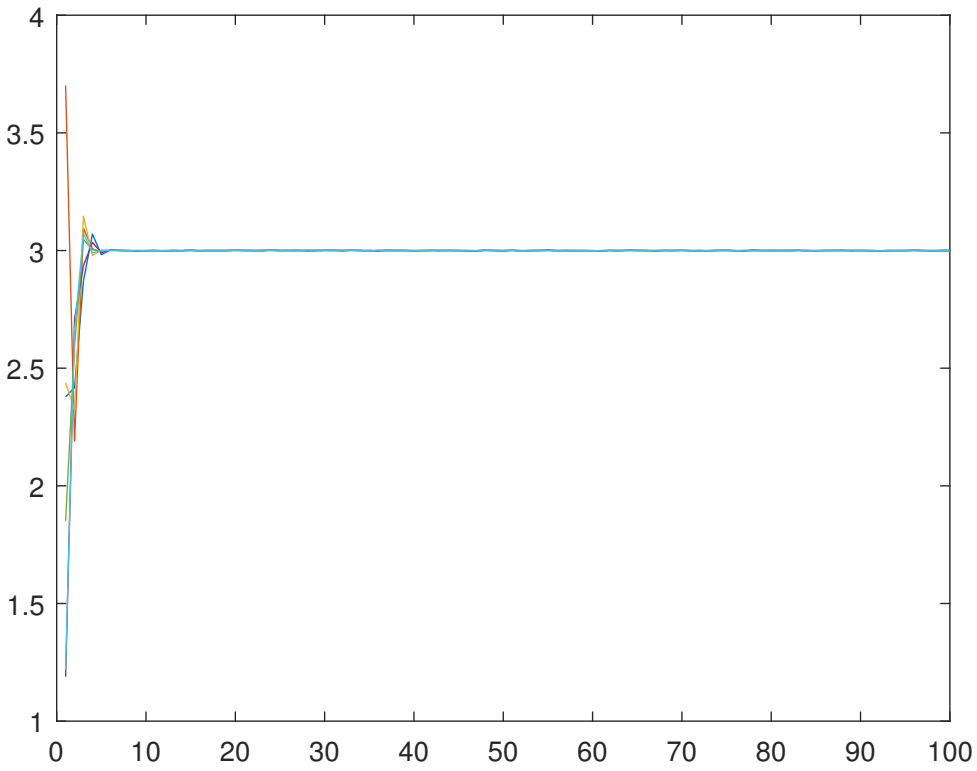


Figure 4.14: Discrete time consensus on estimates on low SD

Running algorithm set E on a set of large standard deviations(i.e generally noisy sensors) like σ=[0,0,0,0.2,0.4,0.3] the variances at the nodes 1 to 6 were observed to be σ^2=[0.0034,0.0074,0.0056,0.0009,0.0019,0.0014]
A visual example of such an evolution of states is in Fig.3.18
Using the algorithm set E for a set of very low standard deviations (i.e very precise sensors) like σ=[0,0,0,0.002,0.004,0.003] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.23611,0.3945,0.1950,0.0589,0.0985,0.0487] *1.0e-05
A visual example of such an evolution of states is in Fig.4.15

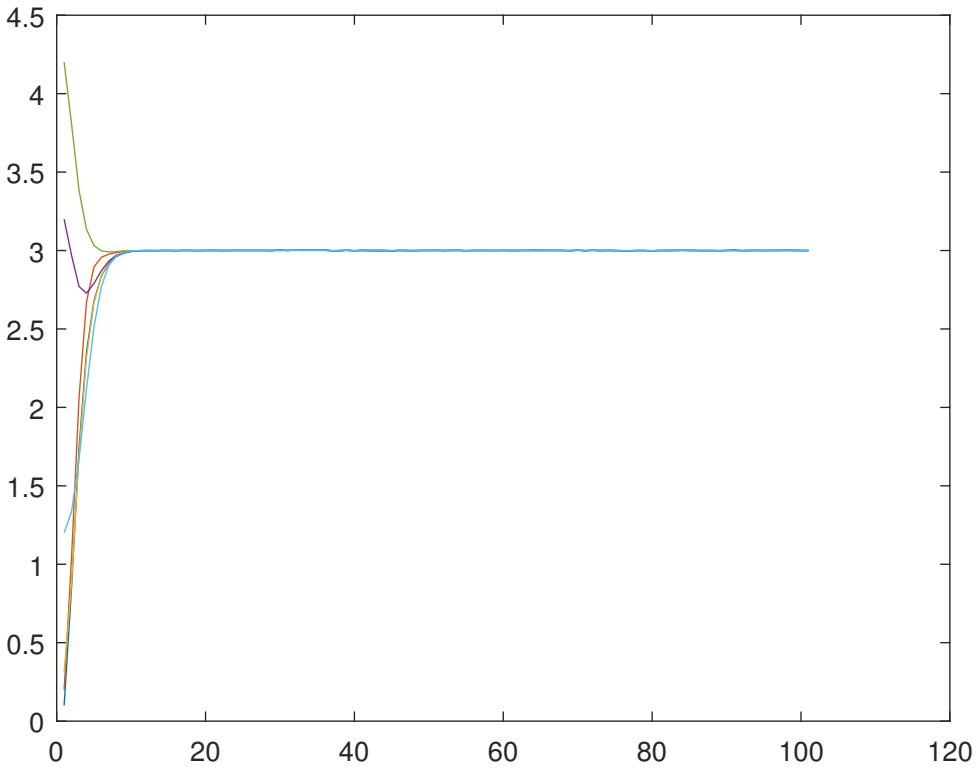


Figure 4.15: Continuous time consensus on estimates on low SD

4.4.1 Discussion on results

From what can be inferred from the above process variances, the performance of the consensus on estimates is a little bit inferior for noisy sensor data, when compared to the Bayesian consensus algorithm for the same set of sensor variances. However, its calculation does not require the knowledge of sensor variances and for this very fact it remains more easily implementable. For sensors with less noise, the consensus algorithm on estimates outperforms the Bayesian state update algorithm by 1-2 orders of magnitude.

4.4.2 Coupling

In order to visualize the coupling between the states for both the algorithms, we again use the *xcorr* and *xcov* functions in MATLAB in exactly the same way as has been described in 4.2.2.

Fig 4.16 and Fig 4.17 are the plots for the outputs of the *xcov()* function and the *xcorr()* function respectively when the Algorithm set D is used.

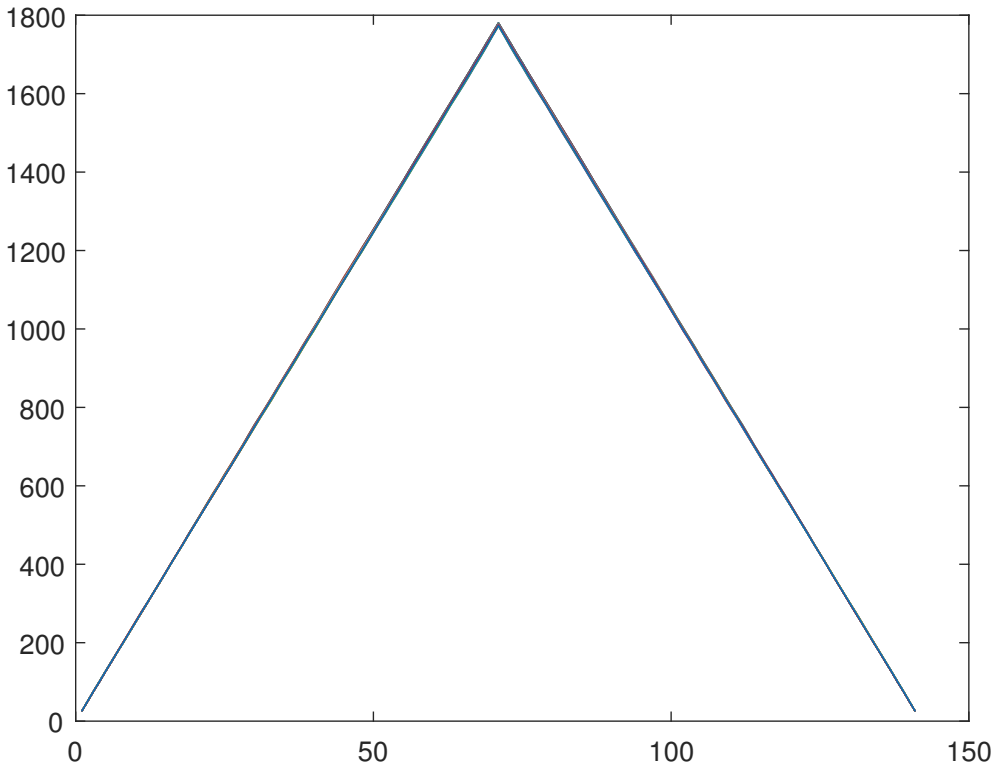


Figure 4.16: Crosscorrelation on DT consensus on estimates

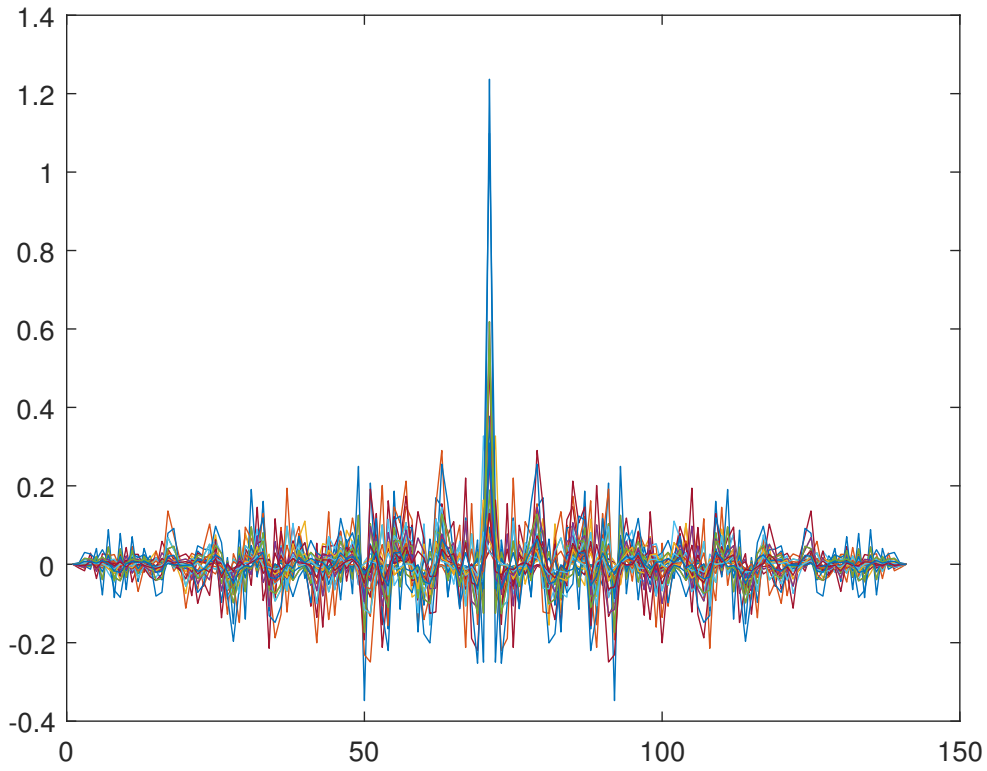


Figure 4.17: Crosscovariance on DT consensus on estimates

Fig 4.18 and Fig 4.19 are the plots for the outputs of the *xcov()* function and the *xcorr()* function respectively when the Algorithm set E is used.

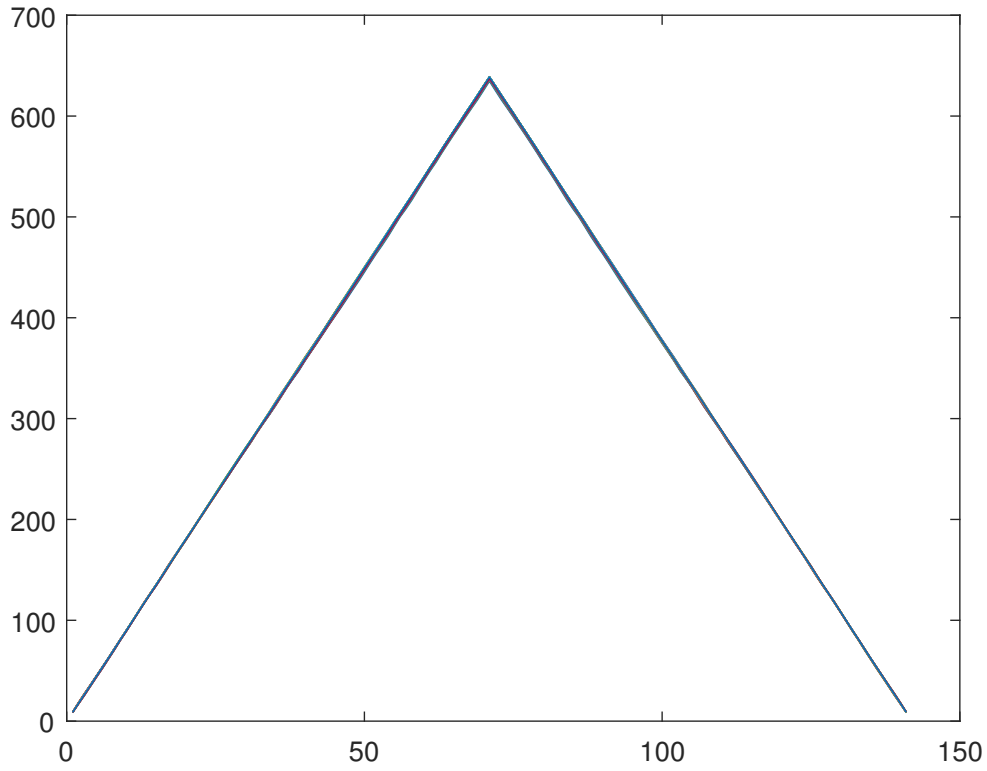


Figure 4.18: Crosscorrelation on CT consensus on estimates

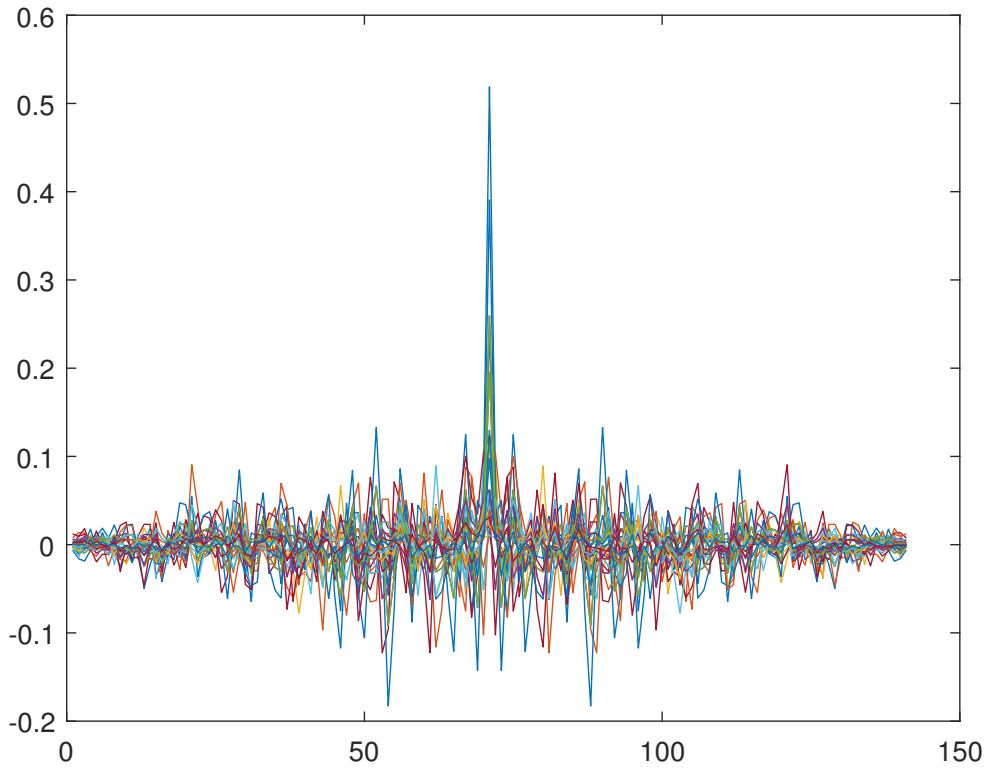


Figure 4.19: Crosscovariance on CT consensus on estimates

Using the update algorithm D, we get the matrix

$$xcov_0 = \begin{bmatrix} AA & AB & AC & AD & AE & AF \\ BA & BB & BC & BD & BE & BF \\ CA & CB & CC & CD & CE & CF \\ DA & DB & DC & DD & DE & DF \\ EA & EB & EC & ED & EE & EF \\ FA & FB & FC & FD & FE & FF \end{bmatrix}$$

and from that the normalized cross covariance matrix

$$Nxcov_0 = \begin{bmatrix} .0448 & .0319 & .0108 & .0224 & .0159 & .0054 \\ .0319 & .2162 & .0734 & .0159 & .1081 & .0367 \\ .0108 & .0734 & .1132 & .0054 & .0367 & .0566 \\ .0224 & .0159 & .0054 & .0112 & .0080 & .0027 \\ .0159 & .1081 & .0367 & .0080 & .0541 & .0184 \\ .0054 & .0367 & .0566 & .0027 & .0184 & .0283 \end{bmatrix} * 10^{-5}$$

where the process variance at the nodes, obtained using the *var()* function is

$$\sigma^2 = [0.0448, 0.2162, 0.1132, 0.0112, 0.0541, 0.0283] * 10^{-5}$$

4.4.3 Noise output

To visualize the noise characteristics at each node, we subtract the state values for a very noisy sensor from the state values of a very precise sensor for a very large number of steps (1000 in this case). We use the *normplot()* command to demonstrate the Gaussian nature of the noise. Fig. 4.20 is the normplot for node 4 in the network.

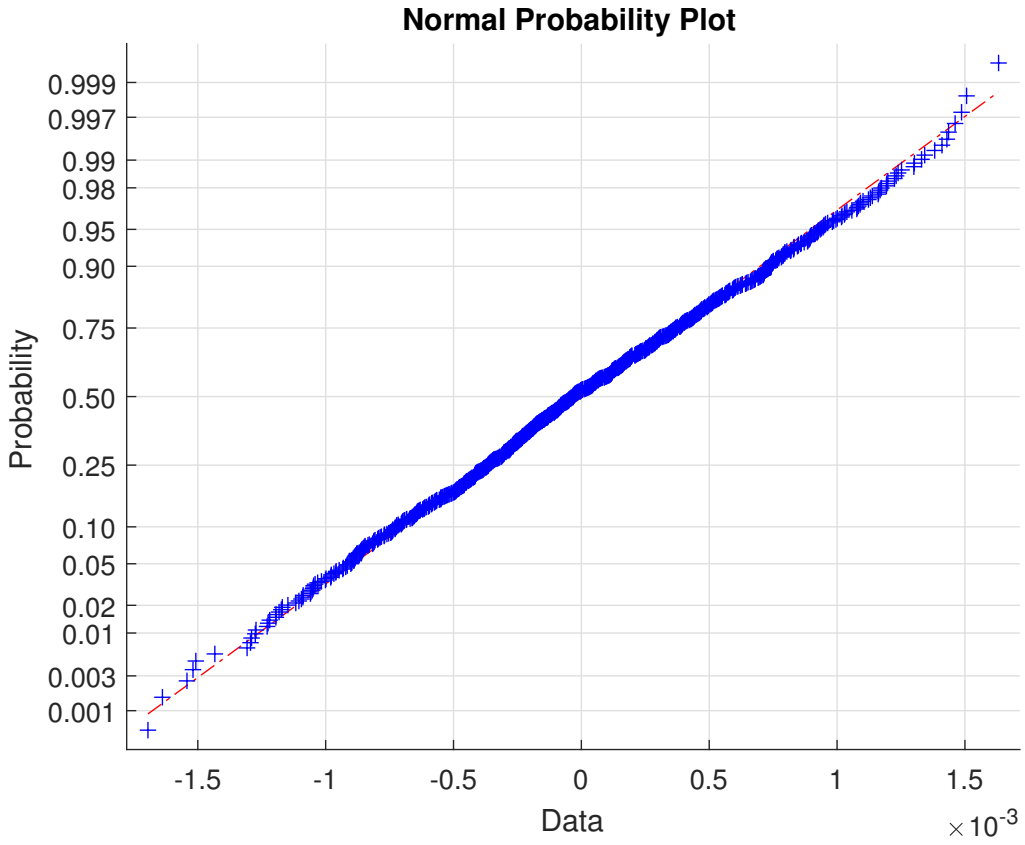


Figure 4.20: Normplot for node 4

The same Gaussian noise that is at the output of node 4, is shown to have a degree of cross covariance, as shown in Fig. 4.21 and cross correlation, as shown in Fig 4.22 - which is not simply a Dirac delta function at the zero lag position. In other words it is a coloured noise having a Gaussian distribution,which is consistent with the fact that the system that is in consideration is a linear system.

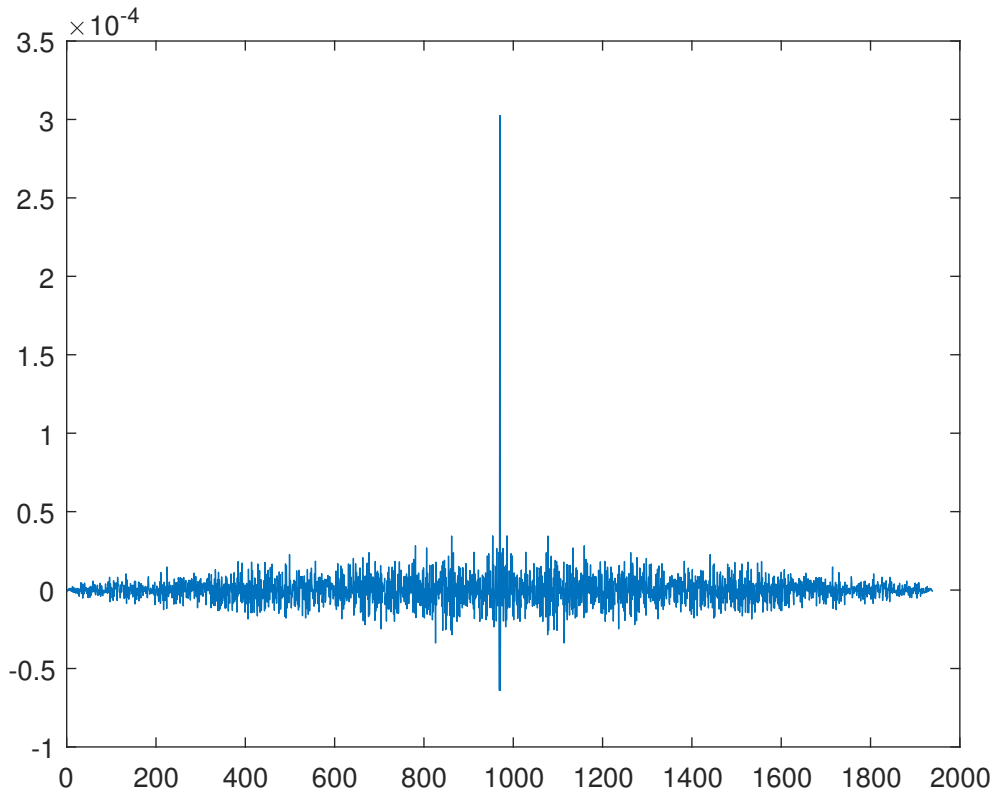


Figure 4.21: Crosscovariance for noise on Consensus on estimates

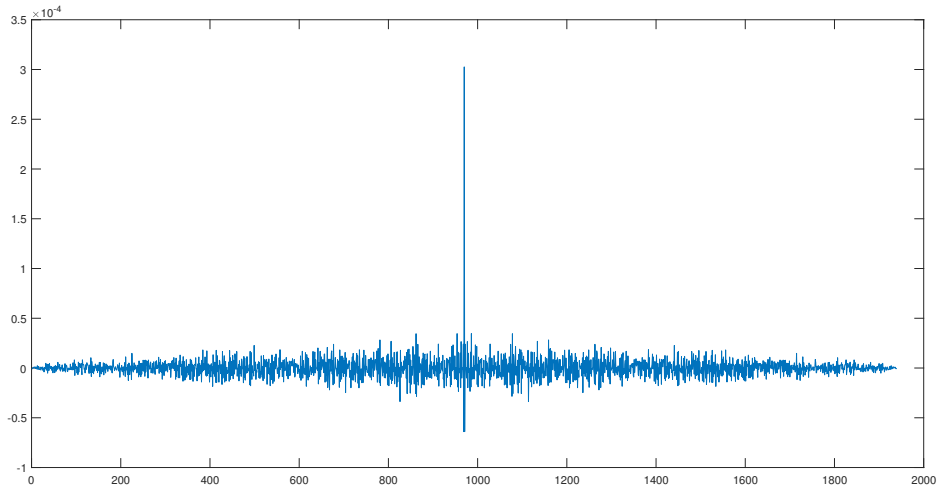


Figure 4.22: Crosscorrelation for noise on Consensus on estimates

4.4.4 General discussion

for the above set of observations and analysis, we can conclude that in general for noisy sensors, the Bayesian state update algorithm performs better than the consensus algorithm on estimates. However when the task of tracking a fast varying signal is considered, the consensus algorithm on estimates perform better than the Bayesian state update algorithm. Below in fig 4.23 we exhibit the cross correlation between a high frequency sinusoid and the sensor node outputs for the Bayesian state update algorithm and in fig 4.24 we exhibit the same for the consensus algorithm on estimates.

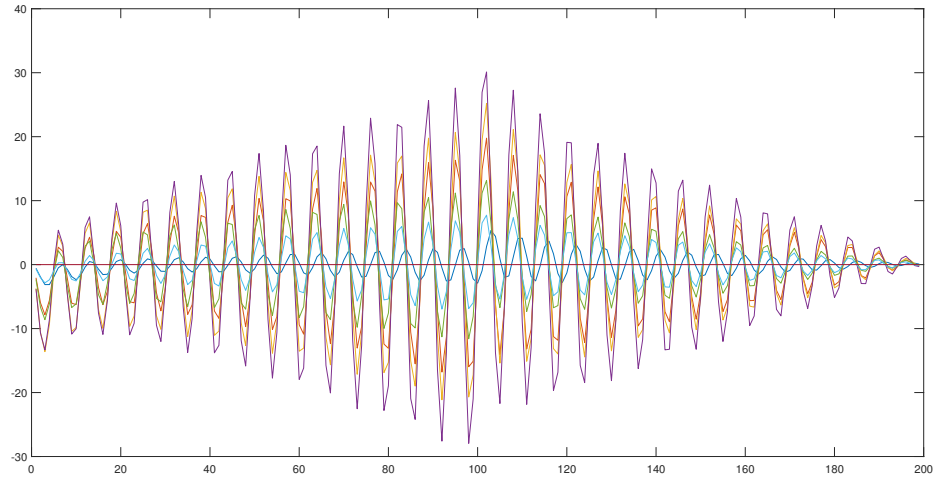


Figure 4.23: Bayesian state update crosscorrelation with sinusoidal input

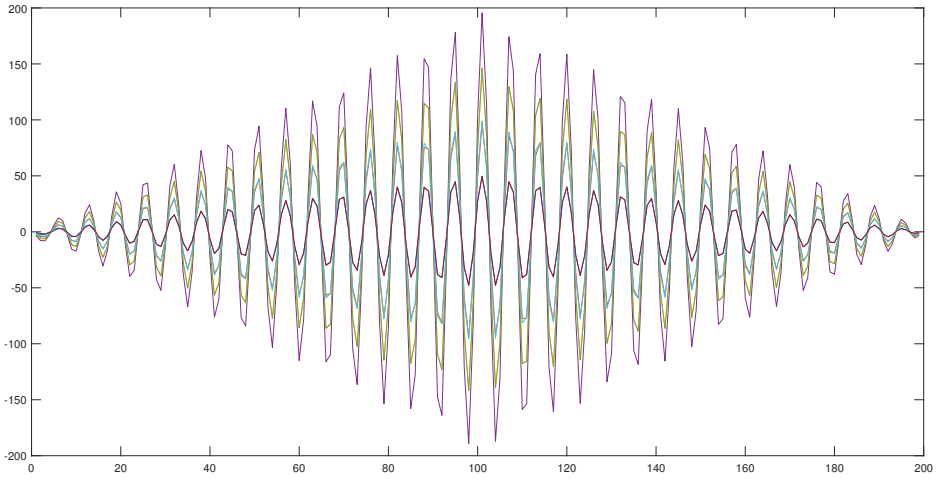


Figure 4.24: Consensus on estimates crosscorrelation with sinusoidal input

4.5 Luenberger Observer on a Strongly Connected Digraph (where all the nodes are sensors)

Here we observe the statistics of the outputs for the Luenberger observer that was discussed in 4.6. It is given as:

$$\dot{\hat{x}_i} = A\hat{x_i} + K_i(z_i - H_i\hat{x_i}) + \gamma P_i \sum_{j \in N_i} e_{ij}(\hat{x_j} - \hat{x_i}) \tag{4.9}$$

$$K_i = P_i H_i^T R_i^{-1}, \gamma > 0$$

$$\dot{P_i} = AP_i + P_i A^T + BQB^T - K_i R_i K_i^T$$

with a Kalman consensus estimator and initial conditions $P_i(0) = P_0$ and $\dot{\hat{x}_i} = x(0)$

with a modified Bayesian weighing of the injection gain K_i and the edge weight e_{ij}

$$K_i \longrightarrow K_i(1 + 1/\sigma_i^2)$$

$$e_i j \longrightarrow [(\sum_{k \neq i} 1/\sigma_k^2)/(\sigma_i^2 + \sum_{k \neq i} 1/\sigma_k^2)]e_{ij}$$

4.6 Process Variance

We observe the process variance of the Luenberger observer algorithm in this section in the same way as it was observed in section 4.2 and 4.4 earlier in the chapter.

Running the Luenberger observer algorithm on a set of large standard deviations(i.e generally noisy sensors) like σ=[11.4, 3.8,4.6,4.3,8.2,13.4] the variances at the nodes 1 to 6 were observed to be
σ^2=[0.2159,0.2252,0.2089,0.2301,0.2412,0.2110]
A visual example of such an evolution of states is in Fig.3.23
Using the Luenberger observer for a set of very low standard deviations does not work very well in this case because of the design of the Bayesian weighings which tends towards a singularity error with very low values of sensor variances.

4.6.1 Discussion on results

It is evident from the above set of observations that when dealing with noisy sensors, The currently implemented version of the Luenberger observer does not perform satisfactorily.

The reason behind this could be the lack of an optimal design approach to fine tune the observer coefficients. In the original Olfati Saber paper [14] as well, the convergence was never satisfactory. A redesigning of the Bayesian weights may be necessary as well to get rid of the singularity errors for sensors with low variance in measurements.

4.6.2 Coupling

In order to visualize the coupling between the states for the algorithm, we again use the *xcorr* and *xcov* functions in MATLAB in exactly the same way as has been described in 4.2.2 and 4.4.2.
Fig 4.25 and Fig 4.26 are the plots for the outputs of the *xcov()* function and the *xcorr()* function respectively when the Luenberger observer is used.

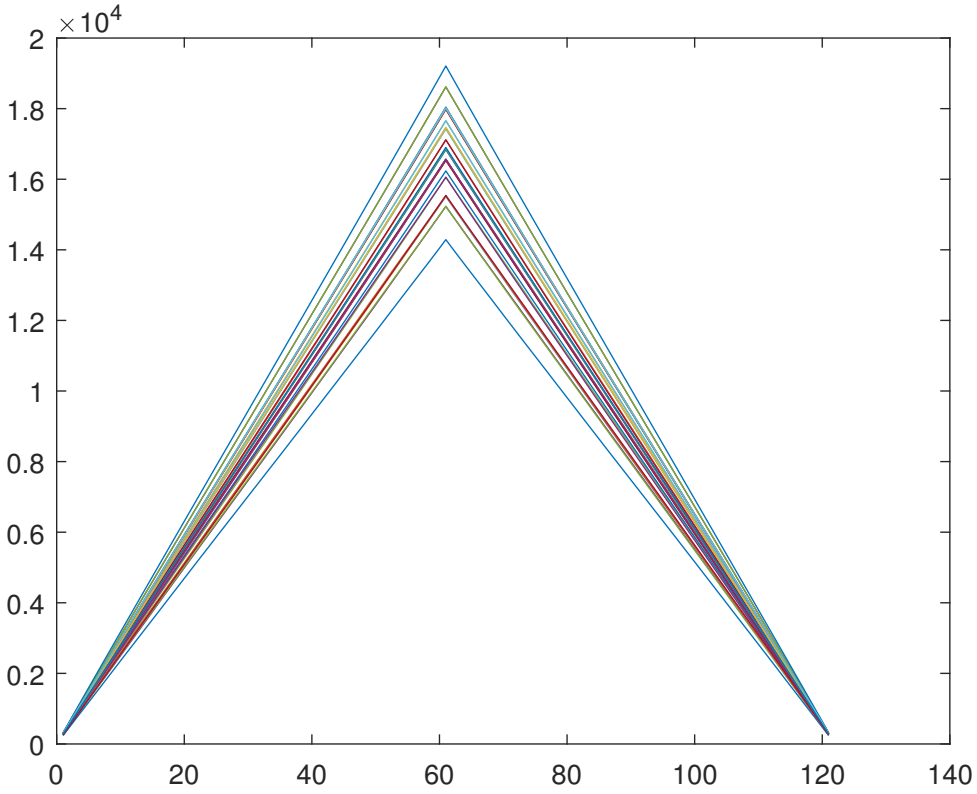


Figure 4.25: Crosscorrelation on Luenberger

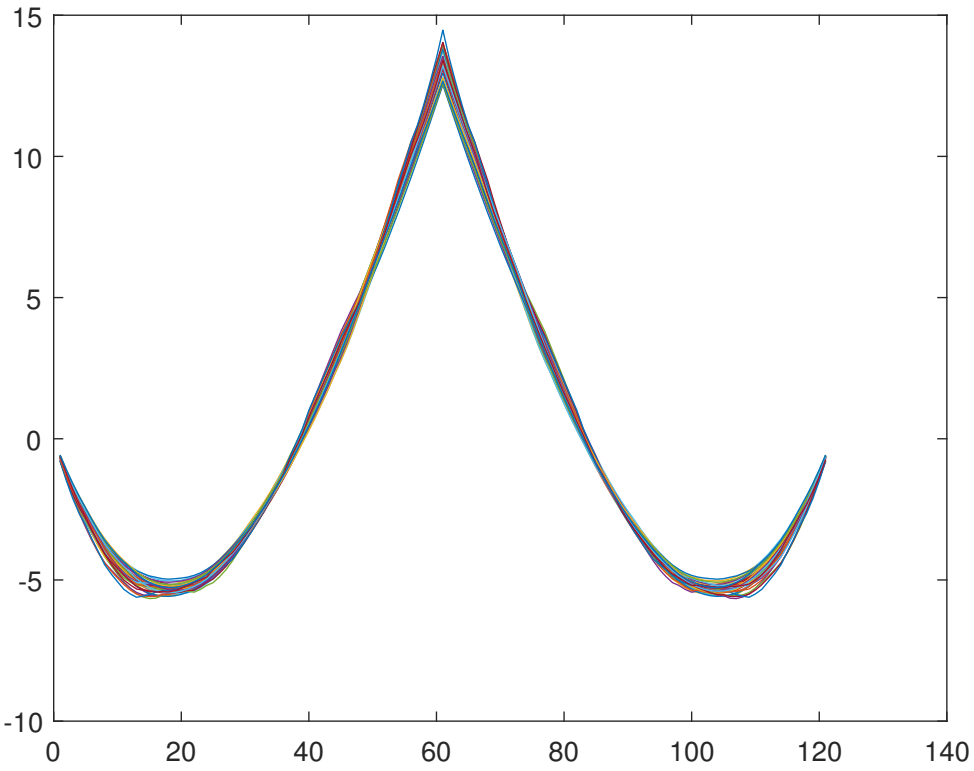


Figure 4.26: Crosscovariance on Luenberger

Using the Luenberger observer, we get the normalized cross covariance matrix

$$Nxcov_0 = \begin{bmatrix} .3299 & .3731 & .3242 & .3462 & .4356 & .2985 \\ .3731 & .4256 & .3696 & .3935 & .4982 & .3397 \\ .3242 & .3696 & .3221 & .3423 & .4331 & .2959 \\ .3462 & .3935 & .3423 & .3652 & .4610 & .3151 \\ .4356 & .4982 & .4331 & .4610 & .5894 & .3980 \\ .2985 & .3397 & .2959 & .3151 & .3980 & .2726 \end{bmatrix}$$

where the process variance at the nodes, obtained using the *var()* function is

$$\sigma^2 = [0.3299, 0.4256, 0.3221, 0.3652, 0.5894, 0.2726]$$

4.6.3 Noise output

We use the *normplot()* command to demonstrate the Gaussian nature of the noise. Fig. 4.26 is the normplot for node 4 in the network.

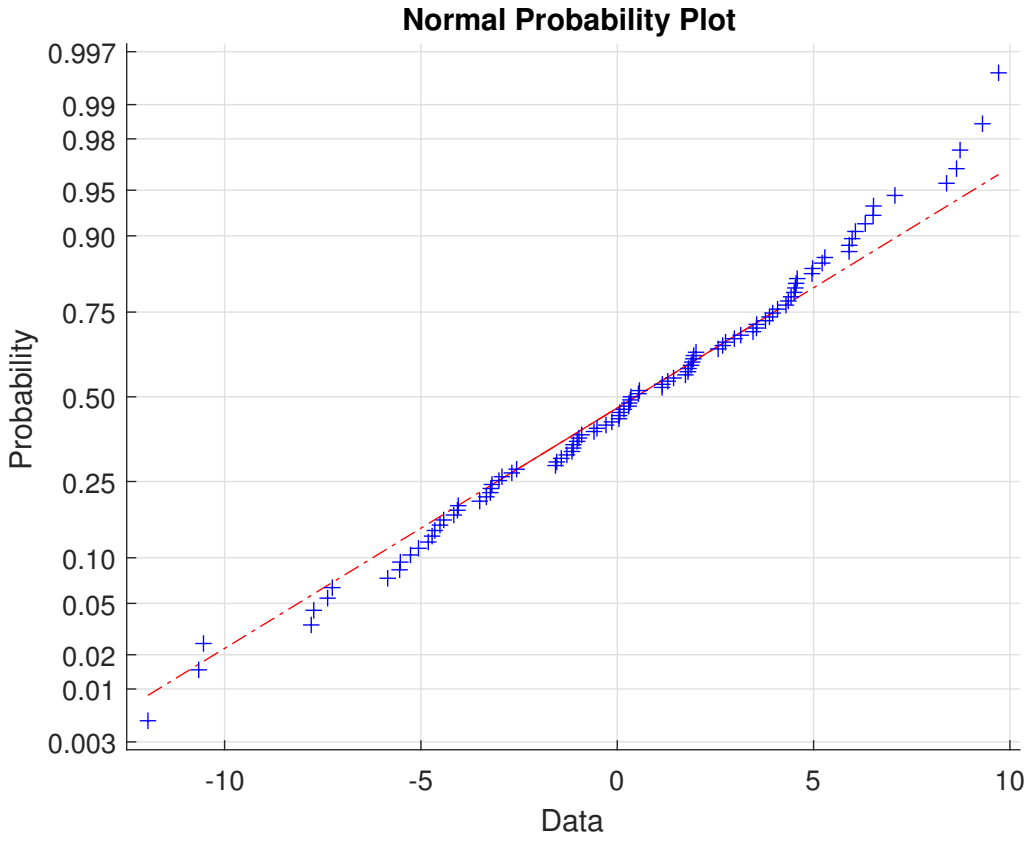


Figure 4.27: Luenberger noise normplot for node 4

The same Gaussian noise that is at the output of node 4, is shown to have a degree of cross covariance, as shown in Fig. 4.27 and cross correlation, as shown in Fig 4.28 - which is a coloured noise having a Gaussian distribution, which is consistent with the fact that the system that is in consideration is a linear system.

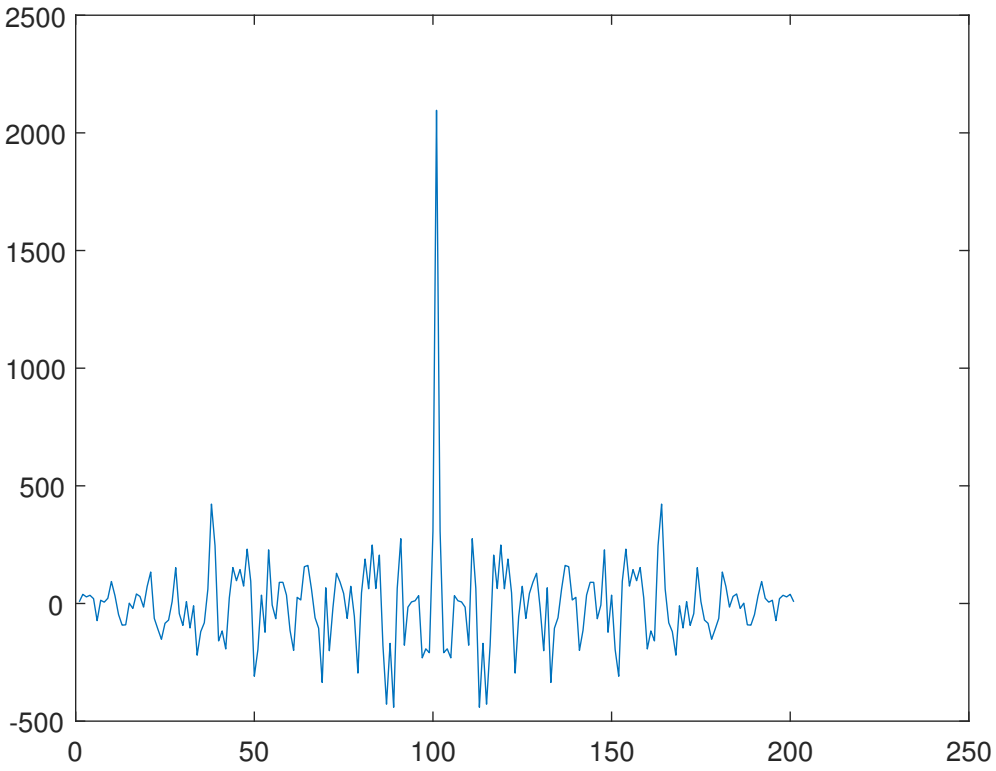


Figure 4.28: Luenberger noise crosscovariance for node 4

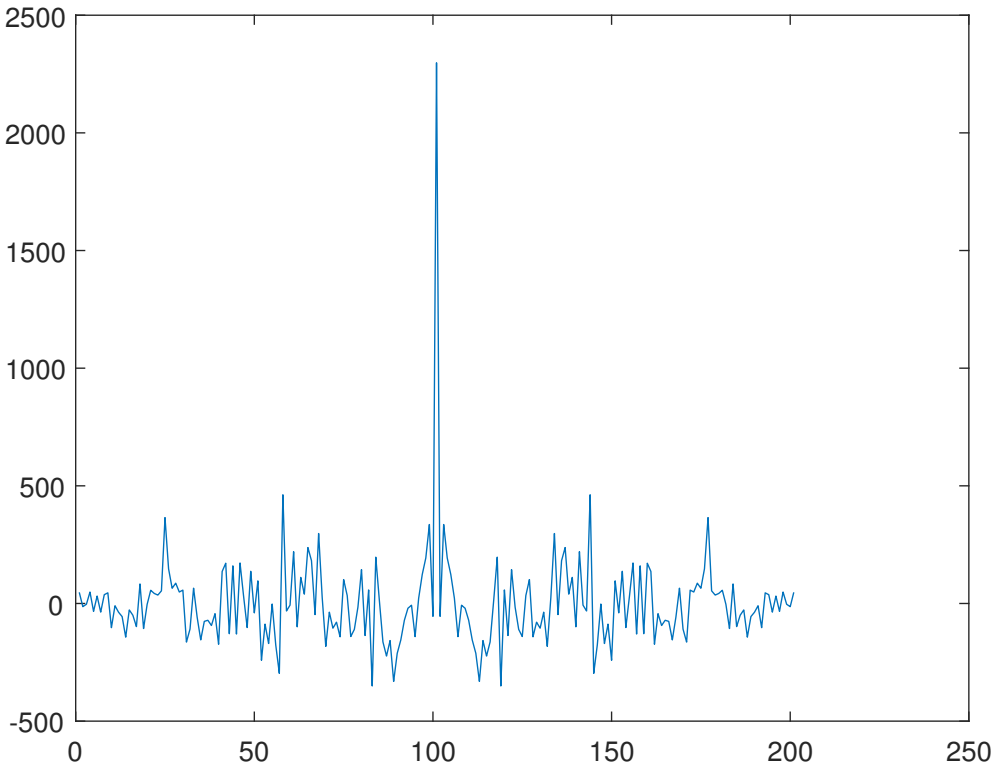


Figure 4.29: Luenberger noise crosscorrelation for node 4

4.6.4 General discussion

for the above set of observations and analysis, we can conclude that in general for noisy sensors, the Luenberger state estimator is not ideally suited without an optimized design. This remains an area where opportunity for future work remains in case of a Bayesian weighted distributed Luenberger observer design.

Chapter 5

Conclusion and Future Work

In this work, we were able to show that the single integrator Bayesian Consensus algorithm performs satisfactorily for tracking a steady/slowly varying measurement value. Infact it was shown to be performing better than the already established Consensus on estimates algorithm. However, The tracking performance of the Bayesian Consensus algorithm is inferior to that of the Consensus on estimates filter.

We also observe that the un-optimized performance of the Luenberger type observer is not comparable to that of the Bayesian or the Consensus on estimates algorithm. In its present form it is virtually useless, it however has promising prospects in the sense that the accounted for variances at the sensor nodes (through Bayesian weighing) may lead to a robust observer once optimized.

For future work an extension of the present work to cater for non linear dynamics may be considered. Also, Hardware based implementations of the algorithms may be another avenue of interest. However, the most immediate improvement would be the optimization of the Luenberger observer design.

Bibliography

[1] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri. Distributed kalman filtering based on consensus strategies. *IEEE Journal on Selected Areas in Communications*, 26(4):622–633, May 2008.

[2] Thomas E. Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Multi-target tracking using joint probabilistic data association. *Proceedings of the IEEE Conference on Decision and Control*, 2:807–812, 1980.

[3] Jesús Ibáñez, Antonio F. Gómez-Skarmeta, and Josep Blat. Dj-boids: Emergent collective behavior as multichannel radio station programming. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pages 248–250, New York, NY, USA, 2003. ACM.

[4] Robert E. Kass and Adrian E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

[5] Shigeru Kondo and Takashi Miura. Reaction-diffusion model as a framework for understanding biological pattern formation. *science*, 329(5999):1616–1620, 2010.

[6] Gerardo Lafferriere, Alan Williams, J Caughman, and JJP Veerman. Decentralized control of vehicle formations. *Systems & control letters*, 54(9):899–910, 2005.

[7] A. Leon-Garcia. *Probability, Statistics, and Random Processes for Electrical Engineering*. Pearson/Prentice Hall, 2008.

[8] Frank L. Lewis, Hongwei Zhang, Kristian Hengster-Movric, and Abhijit Das. *Cooperative Control of Multi-Agent Systems: Optimal and Adaptive Design Approaches*. Springer Publishing Company, Incorporated, 2014.

[9] Zhiyun Lin, Bruce Francis, and Manfredi Maggiore. Necessary and sufficient graphical conditions for formation control of unicycles. *IEEE Transactions on automatic control*, 50(1):121–127, 2005.

[10] Ion Matei and John S. Baras. Consensus-based linear distributed filtering. *Automatica*, 48(8):1776 – 1782, 2012.

[11] H. Min and Z. Wang. Design and analysis of group escape behavior for distributed autonomous mobile robots. In *2011 IEEE International Conference on Robotics and Automation*, pages 6128–6135, May 2011.

[12] Hiroki Miyazako, Yutaka Hori, and Shinji Hara. Turing instability in reaction-diffusion systems with a single diffuser: Characterization based on root locus. *CoRR*, abs/1309.0111, 2013.

[13] A. V. Moere. Time-varying data visualization using information flocking boids. In *IEEE Symposium on Information Visualization*, pages 97–104, 2004.

[14] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *2007 46th IEEE Conference on Decision and Control*, pages 5492–5498, Dec 2007.

[15] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

[16] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, Sept 2004.

[17] R. Olfati-Saber and J. S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 6698–6703, Dec 2005.

[18] Wei Ren, Randal Beard, and Timothy McLain. Coordination variables and consensus building in multiple vehicle systems. *Cooperative control*, pages 439–442, 2005.

[19] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM.

[20] M. Saska, V. Vonásek, T. Krajník, and L. Preucil. Coordination and navigation of heterogeneous uavs-ugvs teams localized by a hawk-eye approach. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2166–2171, Oct 2012.

[21] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. Approximate distributed kalman filtering in sensor networks with quantifiable performance. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 133–139, April 2005.

[22] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 237(641):37–72, 1952.

[23] Hongwei Zhang, Frank L Lewis, and Zhihua Qu. Lyapunov, adaptive, and optimal design techniques for cooperative systems on directed communication graphs. *IEEE Transactions on Industrial Electronics*, 2012.