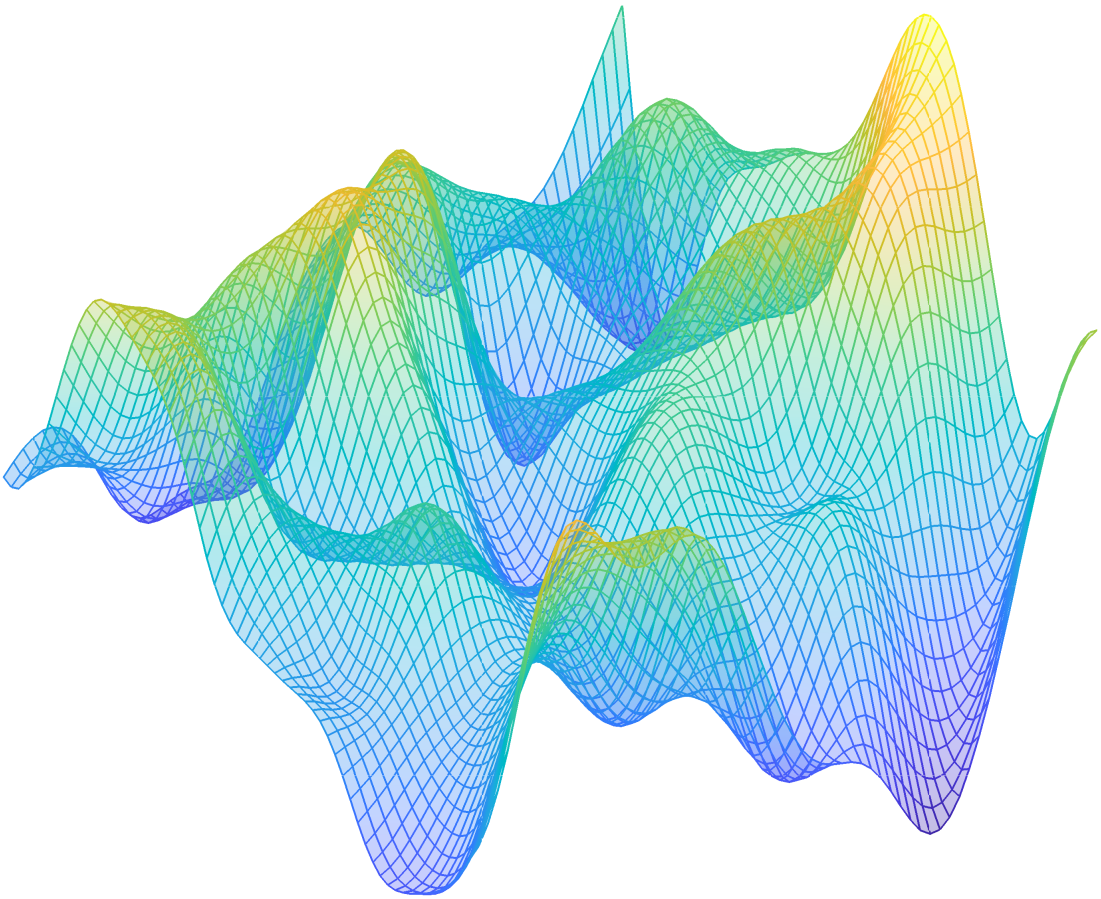
Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-5276179

M.Sc. Thesis

Distributed Gaussian Process for Multi-agent Systems

Peiyuan Zhai B.Sc.
Student ID: 5276179



Faculty of Electrical Engineering, Mathematics and Computer Science **Delft University of Technology**

Distributed Gaussian Process for Multi-agent Systems

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Peiyuan Zhai B.Sc.
from Shenzhen, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Copyright © 2022 Circuits and Systems Group
All rights reserved.

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Distributed Gaussian Process for Multi-agent Systems"** by **Peiyuan Zhai B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 25th August 2022

Advisor:

Dr. Raj Thilak Rajan

Committee Members:

Dr.ir.Richard Heusdens

Dr. Bianca Giovanardi

Abstract

This work is focused on environmental monitoring and learning of unknown field by Gaussian Process (GP) in Multi-agent Systems (MAS). The two main problems are how to develop fully-distributed and robust algorithm to (1). optimize GP hyperparameters, and (2). aggregate GP predictions from agents.

The state-of-the-art distributed GP hyperparameter optimization algorithm is proximal alternated direction method of multipliers (pxADMM), which requires a center station in MAS. Based on pxADMM, two fully-distributed algorithms, pxADMM$_{\text{fd}}$ and pxADMM$^{*}_{\text{fd}}$, are proposed so that the center station is no longer needed. Asynchronous behavior is also introduced into the proposed algorithms to deal with heterogeneous processing time of agents.

Current aggregation methods are classified based on whether datasets are independent. Under independence assumption, PoE and BCM families of methods are distributed by applying discrete time consensus filter (DTCF), which is proposed to be replaced by primal-dual method of multiplier (PDMM) for faster convergence. Without independence assumption, the Nested Pointwise Aggregation of Experts (NPAE) can be distributed by NPAE-JOR in complete graph with high flooding overhead. We propose fully-distributed CON-NPAE in connected graph to eliminate flooding overhead.

Simulation results show that the proposed hyperparameter optimization algorithms are fully-distributed at a cost of 2 to 4.5 times more iterations compared to pxADMM. The fully-dsitributed PoE and BCM based methods are accelerated, and the fully-distributed CON-NPAE makes comparable aggregations as NPAE without flooding overhead. Future work will be focused on the theoretical convergence analysis of pxADMM$^{*}_{\text{fd}}$, the effect of network structure on CON-NPAE and new type of distributed NPAE based on inducing points.

Acknowledgments

This nine-month-long thesis work is not only the final work of my master study in TU Delft, but also a milestone for my academic career. This work is inspired and supported by many people.

First, I would like to show my gratitude to my supervisor *Raj Thilak Rajan* for guiding me through my thesis in the distributed signal processing area. Your guidance helps me to discover an interesting topic for my thesis and keep in the main scope of the project. You have also given many suggestions on good habits of doing research and writing skills, which are valuable for my future career. With your support, I was able to attend SITB2022 conference, which was my first one, and gained memorable experience. What is more, thank you for organizing the weekly meeting of Distributed Autonomous Systems group, from where I have been inspired much to extend my own research.

I would also like to thank *Richard Heusdens* and *Bianca Giovanardi* for being members in my thesis committee.

I want to thank my friends - *Nan, Rui, Zhonggang, Xuzhou, Edoardo and many others* - for all the emotional or technical support, fun time and good restaurants. I would also like to thank *the Circuits and Systems* group for providing very good on-campus offices, inspiring seminars and fun activities.

Finally, I would like to thank my family - dad, mum and brother - for their financial and emotional supports to me, so that I can have a wonderful time during my graduate study.

Peiyuan Zhai B.Sc.
Delft, The Netherlands
25th August 2022

Contents

List of Figures

List of Tables

Acronyms and Nomenclatures

Acronyms

ADMM	Alternated direction method of multipliers
ayncADMM	Asynchronous proximal ADMM
BCM	Bayesian Committee Machine
BMS	Bayesian Model Selection
DTCF	Discrete Time Consensus Filter
GD	Gradient descent
GHRSST	The Group for High Resolution Sea Surface Temperature
GPR	Gaussian Process Regression
GP	Gaussian Process
MAP	Maximize a posterior
MAS	Multi-agent system(s)
MMSE	Minimum Mean Squared Error
MM	Methods of Multipliers
MoE	Mixture of Experts
nGD	Naïve Gradient Descent
NLML	Negative log-marginal likelihood
NPAE	Nested Pointwise Aggregation of Experts
PDMM	Primal-Dual Method of Multipliers
pxADMM	Proximal alternated direction method of multipliers

Nomenclatures

$\forall$	'for all' symbol.
$\in$	'to be a member of' symbol
ν	A parameter controlling the convergence rate of DTCF or JOR.
$\mathbf{1}$	All-ones vector.
$\mathbf{z}$	Auxiliary consensus variable for used in consensus problems.
ξ	Auxiliary variable used in modified PDMM consensus average.
$\mathbf{X}$	Collection of input data points in a dataset.
$\mathbf{y}$	Collection of output data points in a dataset.
$\mathcal{O}(\cdot)$	Complexity (computational, memory, convergence, etc.).
$\mathbf{K}$	Covariance matrix of a Gaussian Process.
$\mathbf{\Sigma}$	Covariance matrix of multi-dimensional radial basis function kernel.
$\mathcal{D}$	Dataset containing input points and corresponding observations.
∇	Difference operator.
$\delta(m, n)$	Dirac delta function equals 1 when input $m = n$, and 0 otherwise.
$\boldsymbol{\theta}$	Hyperparameter set of Gaussian Process.
$\mathbf{I}_N$	Identical matrix of size $N \times N$.
m	Indices for agents owned datasets or variables when used as subscript.
α	Influence weight in rBCM, gPoE and grBCM.
$\langle \mathbf{a}, \mathbf{b} \rangle$	Inner product of vector $\mathbf{a}$ and $\mathbf{b}$.
$\mathbf{x}$	Input vector of a function.
$k(\cdot, \cdot)$	Kernel function.
w	Measurement noise.
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2.
$\mathbf{M}$	Number of agents in multi-agent systems.
$\lvert\cdot\rvert$	Number of elements in a set.
y	Output scalar value of a function.
$\lVert\cdot\rVert_p$	p-norm.
∂	Partial derivative.
$\odot$	Pointwise multiplication.
$\mathcal{A}$	Set of activated agents.
ϵ	Small positive value.
$\mathbf{a}_{[i]}$	The ith entry of vector $\mathbf{a}$.

$\mathbf{A}_{[i,]}$	The ith row of matrix $\mathbf{A}$.
$\mathbf{A}_{[,j]}$	The jth column of matrix $\mathbf{A}$.
$Cov(\mathbf{X}, \mathbf{Y})$	The covariance between two input variables, or every input pairs from two input vectors.
$\mathbf{A}_{[i,j]}$	The entry at position (i, j) of matrix $\mathbf{A}$.
$A \backslash B$	The relative complement of set B respect to set A
t	Time or iteration indices. When used as superscript for a variable, means the specific value of that variable at time instance or iteration t
$\mathcal{N}(m)$	Topological neighborhood of agent m, containing all its neighbors.
$\boldsymbol{\lambda}$	Vector of dual variables.
ω	Weights for GP aggregation.
$diag(\cdot)$	With input a vector, the output is a square diagonal matrix with the elements of input vector on the main diagonal. With input a square matrix, the output is a vector containing the main diagonal elements.

1 Introduction

DISTRIBUTED algorithms in distributed systems are promising solution to many real world applications, e.g., environmental monitoring. In this thesis, the focus is on the algorithm for environmental monitoring in multi-agent systems (MAS). In this chapter, basic introductions and problem statement of the thesis project are introduced in Section 1.1. Some necessary preliminaries are explained in Section 1.2. The datasets and simulation setting are introduced in Section 1.3. Contributions of this work are briefly concluded in Section 1.4. The outline of the thesis report is shown in Section 1.5.

1.1 Background & Problem Statement

1.1.1 Multi-agent systems

Multi-agent system (MAS) consists of identical measurement and computation units that work in distributed manner, where each agent is called an agent. The agents are able to communicate with other agents (or possibly a center station) inside the largest range of communication. Examples of agents include drones, satellites, robots, cars, etc. A specific realization of MAS can also be called a distributed multi-agent network that can be described by graph as explained in Section 1.2.2.

1.1.2 Environmental monitoring

MAS can be deployed for various kinds of applications, e.g., source seeking [3], environmental monitoring [4, 5], Geographical Information Systems modeling [6], signal strength mapping [7], etc. In this work, we specifically focus on the environmental monitoring of unknown spatial fields. As an example, mobile agent network [8] is applied to measure oceanic environmental variables(temperature, flow, biological variable, etc.) so as to better model and understand the ocean on the topic of ecosystems and climate.

The environmental monitoring task learns and reconstructs an unknown field of interest from measurements taken in it. The task can be considered as finding a hidden function describing the underlying environment, which is usually called regression problem. To learn the continuous hidden function based on discrete measurements, a data model defining the behavior of function is firstly assumed and built, after which the exact function is learned from the observed data. Measurement model describing the relationship between hidden function and observed values can also be applied. There are various models for the regression problem, among which the Gaussian Process is chosen.

Gaussian Process (GP) is a non-parametric stochastic process that can be used to describe functions or signals from a probabilistic perspective, in which any finite

collection of input points follows a multi-variate Gaussian distribution. As the data model defined by GP, the unknown environmental field can be learned through finding posterior distribution. As a non-parametric model, GP has the characteristic of high flexibility and robustness, but also has high computational complexity. More details about the GP and motivation of choosing it are introduced in Chapter 2. In MAS, problem arise on how to apply Gaussian Process in a distributed manner, or even fully-distributed situation that does not require assistance from center stations.

1.1.3 Problem statement

In this work, there are two problems to be studied so as to extend and improve Gaussian Process based environmental monitoring in MASs.

1. In MASs, how to train Gaussian Process models, i.e., learning hyperparameters from sampling points in the unknown fields, for environmental monitoring applications in a fully-distributed manner that does not relies on center station.

2. Based on the sampled data points, how does a MAS with distributed Gaussian Process models predicts the unknown field with an compatible accuracy to the full Gaussian Process.

1.2 Preliminaries

1.2.1 Notation

The list of acronyms and nomenclatures are included before the start of this Chapter.

By default, a vector is oriented vertically and denoted by boldface lowercase English or Greek letters, and a matrix is denoted by boldface English or Greek capital letters. A set of variables a_m with $m \in 1, 2, \cdots, M$ can be denoted as $\{a_m\}_{m=1}^{M}$. If the indices for a can be described by a set, e.g., $\mathcal{S}$, then the equivalent notation is $\{a_m\}_{m\in\mathcal{S}}$.

A set of real numbers forming 1 dimensional Euclidean space is denoted as $\mathbb{R}$, for which the D dimensional case is denoted as $\mathbb{R}^D$. The sets for positive and negative real number are respectively denoted as $\mathbb{R}_+$ and $\mathbb{R}_-$.

1.2.2 Graph model

A graph can be used to model the network structure of MAS, and the basics of graph is introduced in this section [9, Ch. 10].

Definition 1.1 (Graph)**.** A Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a non-empty set $\mathcal{V}$ of vertices (or nodes), and a set $\mathcal{E}$ of edges (or links) between pairs of vertices.

The agents in MAS can be regarded as vertices and denoted as $\mathcal{V}$, and the communication links between pairs of agents are the edges denoted as $\mathcal{E}$.

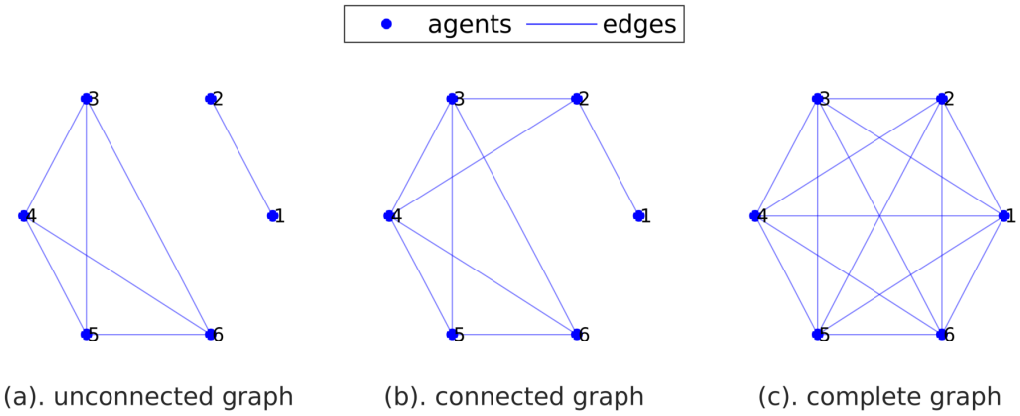


Figure 1.1: Examples of unconnected, connected and complete graphs.

Basic concepts

- **Directed and undirected edge:** An edge connecting a pair of vertices u and v can be denoted as (u, v), indicating that the edge starts from u and ends at v. If the direction matters, then the edge is a directed edge connecting a pair of ordered vertices. If the order does not make a difference, then the edge is undirected. In MAS, an undirected edge represent a two-way communication link between a pair of agents.

- **Directed and undirected graph:** An undirected graph consist of only undirected edges, while a directed graph includes directed edges.

- **Simple graph:** When an edge starts and end at the same vertex, it is called a loop. A simple graph is a graph in which there is not any loop and no two different edges connecting the same pair not vertices.

- **Weighted and unweighted graph:** A graph with different values assigned to the edge is called a weighted graph. A graph in which the edges do not have weights, or only weight 0 and 1, is called unweighted graph. In unweighted graph, 0 and 1 respectively means the disconnection and connection of an edge. In MAS, edges with weights 0 and 1 respectively means the nonexistence and existence of communication links between agents.

- **Complete graph:** A complete graph is a simple undirected graph in which each unique pair of vertices is connected by one edge. A MAS with network structure as complete graph is be called complete network. An example of complete graph with 6 nodes is shown in Figure 1.1c.

- **Connected graph:** A path is a sequence of edges starts from a vertex in the graph and travels from vertex to vertex. The path can end at the same vertex. A connected graph is a graph in which there is always a path between any pair of vertices. A MAS with network structure as connected graph is called connected network. In a connected network, if the information can be passed through agents,

then the information sent from any agent in the network can always be received by any other agents. Examples of unconnected and connected graphs with 6 agents are shown in Figure 1.1a and 1.1b.

- **Neighbors and neighborhood:** For nodes u and v connected by undirected edge (u, v), they are neighbors of each other. For set containing all the neighbors of an agent m, it is called neighborhood of agent m and denoted as $\mathcal{N}(m)$.

Without specific indication, all the networks in this work are undirected, unweighted and connected.

Graph representation

- **Adjacency matrix:** The adjacency matrix $\mathbf{A}$ of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with M vertices is an $M \times M$ matrix with each entry the value 1 or 0. The entry $\mathbf{A}_{[m,n]}$ at position (m, n) is given by

$$\mathbf{A}_{[m,n]} = \begin{cases} 1 & \text{if } (v_m, v_n) \text{ is an edge of } \mathcal{G} \\ 0 & \text{otherwise} \end{cases}, \tag{1.1}$$

  where $v_m, v_n \in \{v_i\}_{i=1}^{M}$ are vertices from $\mathcal{V}$.

- **Degree:** The degree d_m of a vertex $v_m \in \mathcal{V}$ is the number of edges connected to it. Degree vector $\mathbf{d}$ collects the degrees of all the vertices. There is $\mathbf{d} = \mathbf{A1} = \mathbf{1}^T\mathbf{A}$. The degrees can also be organized in a diagonal matrix $\mathbf{D}$ such that $\mathbf{D} = diag(\mathbf{d})$.

- **Laplacian:** The Laplacian matrix of graph is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The Laplacian matrix can be used to check the connectivity of graph. The second smallest eigenvalue of $\mathbf{L}$ can be denoted as λ_2, which is related to the convergence speed of DTCF algorithm introduced in Chapter 4. Also, the graph is disconnected when $\lambda_2 = 0$, which can be used to check the connectivity of generated MAS.

1.3 Datasets and simulations

To construct a dataset, an underlying environmental field is first chosen or generated, based on which the samples are taken through interpolation at points of interest. A dataset $\mathcal{D}$ consists of two parts, the set of input points $\mathbf{X}$ and corresponding output scalar values $\mathbf{y}$ sampled from underlying fields.

The generation of underlying fields is introduced in section 1.3.1, then the simulation and sampling setting are introduced in section 1.3.2.

1.3.1 Underlying field

The datasets provide underlying rectangular 2D environmental fields with range $[r_{x,1}, r_{x,2}] \subset \mathbb{R}$ on x axis and $[r_{y,1}, r_{y,2}] \subset \mathbb{R}$ on y axis, which can also be denoted as $[r_{x,1}, r_{x,2}] \times [r_{y,1}, r_{y,2}] \subset \mathbb{R}^2$.

Artificial 2D Gaussian Process field This is a dataset generated according to 2D spatial Gaussian Process with specified hyperparameters. The details of Gaussian Process are introduced in Chapter 2. A 2D stationary Gaussian Process can be simulated through generating a multivariate normal distribution with circulant embedding method [10], which is a fast algorithm developed to utilize the good properties of Fast Fourier transform (FFT) and multivariate normal distribution. Details of the circulant embedding are out of the scope of this work, and the interested reader is referred to the cited references for detailed descriptions. The code used for circulant embedding methods is adapted from [11]. Figure 1.2 shows four examples of the generated 2D GP fields with the same set of hyperparameters.

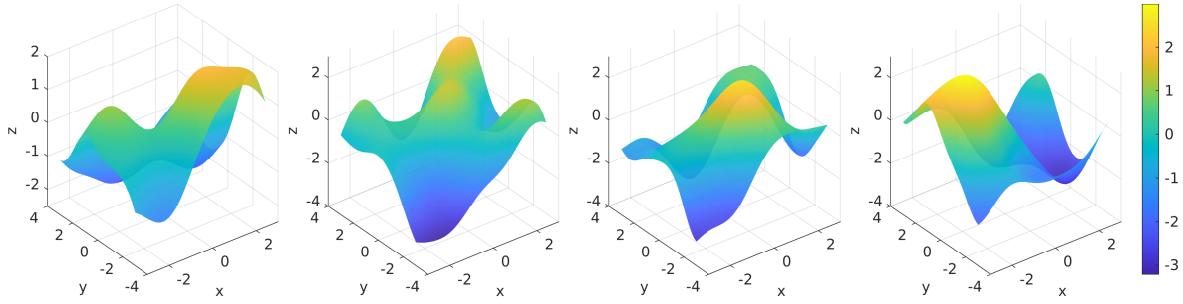


Figure 1.2: Examples of artificially generated 2D GP fields for simulation. The x and y axes span the 2D input space. The z axis span the output space. The magnitudes of the output scalar values are indicated by the colorbar.

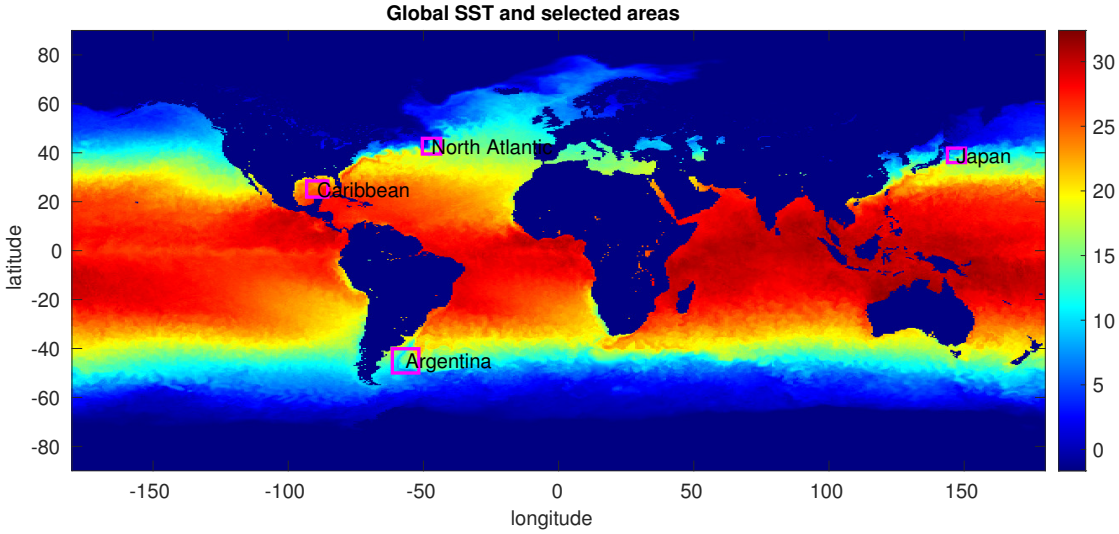
GHRSST dataset The Group for High Resolution Sea Surface Temperature (GHRSST) dataset [12, 13] contains daily global sea surface temperature since 2002. For simplicity, we use Sea Surface Temperature (SST) to indicate the dataset in the following paragraphs. The spatial resolution of the dataset is 0.01 degree in both latitude and longitude. Several regions are copped as 2D field for evaluation of proposed algorithm. Four cropped regions and their positions in the entire map is shown in Fig. 1.3.

1.3.2 Simulation and sampling setting

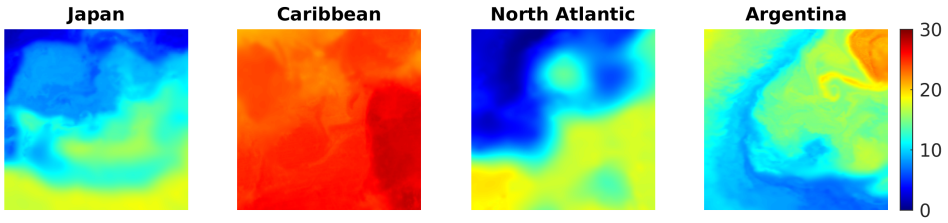
MAS setting MAS can be mathematically represented by an undirected connected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where the number of agents $|\mathcal{V}| = M$. The locations $\mathbf{X}_{\mathcal{V}}$ of agents are uniformly distributed in the range of field. The largest communication distance of an agent is denoted as $r_{max} \in \mathbb{R}_+$, which is identical among the agents. The r_{max} is arbitrarily chosen in a way such that the graph is connected but not complete.

Sampling points and datasets division In MAS, each agent m holds a local dataset $\mathcal{D}_m = \{\mathbf{X}_m, \mathbf{y}_m\}$. Collectively, the set containing all the local datasets is called global dataset and denoted by $\mathcal{D}$.

There are two methods to generate and divide the datasets. The first one is to uniformly sample the points inside the field, which gives the global dataset $\mathcal{D}$. Then



(a) The global SST map and the positions of selected regions



(b) Four selected regions

Figure 1.3: The global SST map and four selected regions. (a): Map of global SST: The map shows data of global SST on 1st April 2022. Temperature values are indicated by different colors shown in colorbar. There is no data in land areas, which are shown in Navy blue. (b): Map of four selected sea regions: The four maps show SST data obtained from sea near Japan, Caribbean sea, North Atlantic and Argentine Sea.

the points are randomly and equally distributed to local datasets without overlap.

The second method firstly generate the local datasets, which are then collectively combined as global dataset. The dataset $\mathcal{D}_m$ for agent m is generated through uniform sampling inside a circle around agent m with radius r_m, which is called the sampling range of agent m. The r_m is assumed to be identical among agents. The points generated outside the region of field are ignored, and makes the exact number of points slightly different among agents. Then the global datasets is formed by the combination of all the local datasets $\mathcal{D} = \{\mathcal{D}_m\}_{m=1}^{M}$.

1.4 Contributions

The main contributions of this thesis project are algorithms proposed to solve the problems stated.

- For hyperparameter optimization of Gaussian Process, two algorithms are pro-

posed. A fully-distributed proximal ADMM algorithm is proposed to extend the current proximal ADMM algorithm. An asynchronous version is also proposed to speed up the convergence. Details of the mentioned algorithms are in Chapter 3.

- For distributed Gaussian Process aggregation, two algorithms are proposed. The current fully-distributed rBCM aggregation algorithm is improved with introduction of PDMM. A fully-distributed NPAE method is proposed so that the amount of data transferred in the network is reduced compared to current NPAE based methods. Details of the mentioned algorithms can be found in Chapter 4.

1.5 Outline

This thesis report is structured as follows.

In Chapter 1, the basic introductions to the background and the problems are given. Necessary preliminaries and notations are also explained.

In Chapter 2, the parametric and non-parametric models for environmental monitoring are compared. It is also motivated to choose the non-parametric model, i.e., Gaussian Process. Gaussian Process for regression problem is also introduced from perspectives of constructing the prior model and predicting based on posterior model.

In Chapter 3, current hyperparameter optimization algorithms, both centralized and distributed, are introduced. Fully-distributed and asynchronous versions of state-of-the-art optimization methods are proposed. Simulations are performed and analyzed.

In Chapter 4, the current algorithms of aggregating local predictions from agents into global predictions are first introduced. Two algorithms are then proposed to improve the current methods.

In Chapter 5, discussions of the thesis project and future works are included.

Figure 1.4 shows the diagram of the structure for this work.

1.6 Conclusion

In this chapter, the following parts are introduced:

- **Background and problem statements**: This work is on the topic of environmental monitoring in multi-agents systems with Gaussian Process.
- **Notations** used in this work, and preliminaries about the **graph model** that is used to model MAS.
- The **datasets** used for simulation, and the simulation settings for constructing MAS and generating sampling points.
- The **outline** of the entire report.

In the next chapter, the motivation of choosing a proper data model, i.e., Gaussian Process, and the basics of Gaussian Process are introduced.

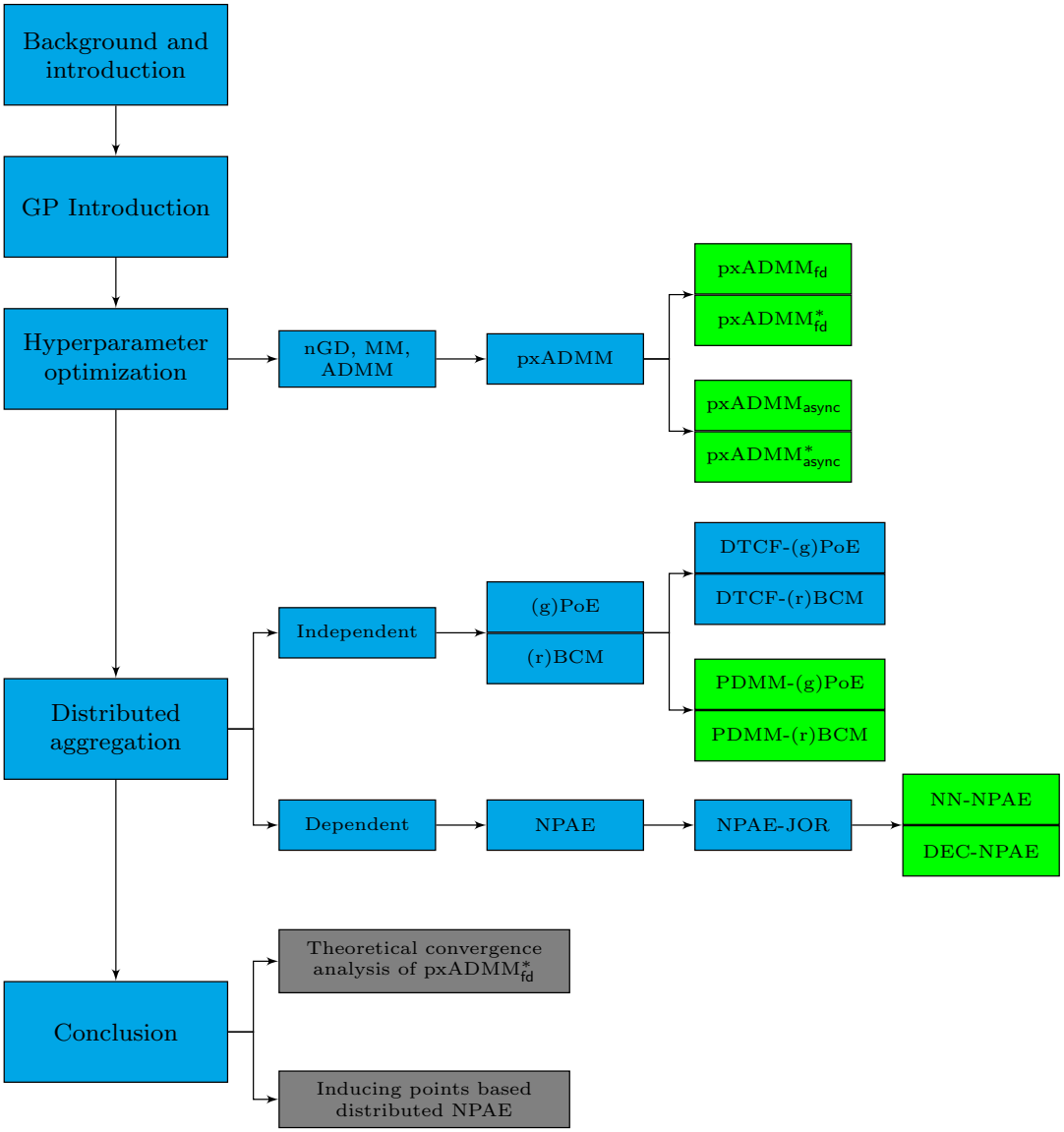


Figure 1.4: Diagram of this work. The green blocks show the contributions of the work. The gray blocks show the future work.

Gaussian Process Regression A Brief Introduction

2

For environmental monitoring, the underlying field can be modeled by Gaussian Process (GP), which is a robust and flexible for unknown signal modeling. In this chapter, more detailed motivation of choosing GP are discussed. Also, the prior and posterior models for GP based regression are introduced. This chapter begins with motivation of choosing Gaussian Process (GP) as the data model in section 2.1, followed by introduction to the basics and properties of GP in section 2.2, including the prior and posterior model. Some 1-D GP simulations are also presented.

2.1 Choosing data model

Let the underlying function of environmental field denoted as $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^D$ is input variable. The task of learning the field is then equivalent to finding corresponding $f(\mathbf{x})$ value under given input x. To describe $f(\mathbf{x})$, various data models are available and can be classified into two groups, parametric and non-parametric model. In this section, an introduction about parametric and non-parametric model is first given, then the Bayesian model and its structure are introduced, which is a class of methods that GP can be classified into.

2.1.1 Parametric and non-parametric model

Depending on whether the output value is directly associated with the input through a set of predetermined parameters or not [14], the data model can be classified as either a parametric or non-parametric model. Their different properties make them suitable for different applications considering the amount of known prior information, computational ability, data available, flexibility, etc.

In many cases, the parametric models contain sets of predefined parameters that directly control the output values under given inputs. For example, a 1-D polynomial model can be noted as $f(x) = \sum_{i=0,1,2,\ldots} w_i x^i$, where parameters $\{w_i\}_{i=0,1,2,\ldots}$ are called weights. The weights directly control the output of the model at given input point. The model is trained by finding the correct set of parameters for the given data, and predictions can then be made by directly calculating the function value at desired input point. Some other typical parametric models include Gaussian Mixture Models and logistic regression. Since the entire model is mainly controlled by few parameters but depends less on the data, a parametric model usually does not require a large dataset for training, which brings a smaller computational intensity. Though strong assumption gives parametric models the above advantages, it also constrains the flexibility of a specific model being applied to other dataset. For example, the performance of linear regression model drops drastically when applied to a dataset with quadratic pattern.

Furthermore, the exact parametric representation for some dataset could be hard to find, either because of the convoluted underlying physics properties or lack of prior knowledge.

On the contrary, non-parametric models do not assume predefined parameters or even an explicit form for the exact underlying function, but rely more on the data to construct a model. Under simple assumptions, non-parametric models automatically explore patterns from the datasets. For example, K-Nearest Neighbors classifier explores similarities among data points based on only one hyperparameter K and a proper distance metric, and provide good classification results in nonlinear dataset. As for GP, it is powerful in nonlinear regression to fit complicated signals that are hard to be parametrically modeled. Another special advantage provided by GP regression is the prediction confidence which can be applied in explore-exploit problem, e.g., maximizing an unknown function with high evaluation cost [15]. Though non-parametric models provides higher flexibility, their data-dependent nature brings higher computational complexity, which is the main drawback for large scale application.

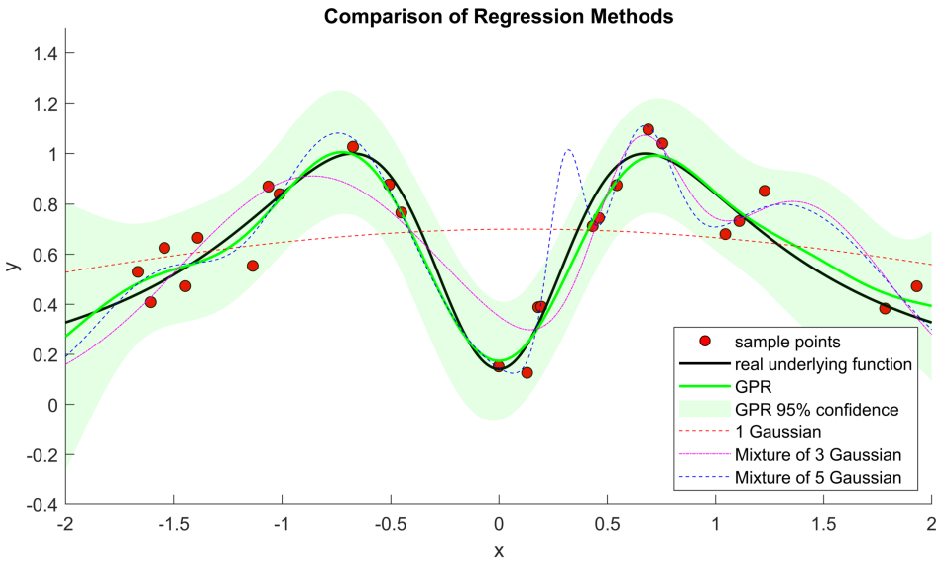


Figure 2.1: The real underlying function shown in black line is generated by $f(x) = \sin(\frac{3}{1+2x^2})$. The red dots are 25 measurement points uniformly distributed across the input region $[-2, 2]$, for which the measurement noise follows $\mathcal{N}(0, 0.1^2)$. The non-parametric GP regressor (GPR) is applied with the most commonly used squared exponential kernel described in Equation (2.7) with hyperparameters $\{\sigma_f, l, \sigma_n\} = \{0.4, 0.4, 0.1\}$. The light green area indicates the 95% confidence interval given by the GPR. Parametric Gaussian Regression with 1,3 and 5 Gaussian components are respectively shown in red, purple and blue dotted lines.

Fig. 2.1 compares examples of parametric and non-parametric model, respectively Gaussian fitting and Gaussian Process Regression (GPR). The non-parametric GPR make prediction by giving posterior distribution, and only need a fixed number of 3 hyperparameters. Parametric Gaussian fitting tries to fit the data points to a summation of multiple weighted Gaussian function, for which the flexibility and number of parameters both increase with the number of Gaussian components. The figure shows

that GPR fit the underlying function better than Gaussian fitting. In fact, Gaussian fitting sometimes gives similar results as GPR, but requires careful selection of Gaussian components number, which requires a good knowledge of the underlying function. On the contrary, GPR requires less information and parameter tuning to reach a even better results.

1. In this work, systems are assumed to be operated in fields with limited prior knowledge, thus the exact underlying physics phenomenons can be hard to decide or parametrized. Under limited prior assumptions, non-parametric models are more flexible and powerful in learning the pattern of the field.

2. As a Bayesian modeling methods, GP builds a prior for the data model and makes posterior prediction based on the data. From a probabilistic view, GP provides the confidence level of prediction, which can be further utilized by MAS as reference for exploring the field. Structure of GP as a Bayesian Model is introduced in section 2.1.2.

3. Without any approximation, GP has high computational and space complexity, respectively $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$, where N is the number of data points. However, under a distributed setting in MAS, the dataset naturally can be divided and assigned to different agents, on which the number of data points to be processed is less. By simply dividing the dataset to M agents, the computational and space complexity can be respectively reduced to $\mathcal{O}(\frac{N^3}{M^2})$ and $\mathcal{O}(\frac{N^2}{M})$, which is already a good start for further optimization.

2.1.2 Bayesian Modeling

Rather than assuming certain values for function, Bayesian modeling is a probabilistic model that regards all the signals involved, either input or output, as random variables. Bayesian modeling relates several parts of the systems based on Bayes' theorem

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}, \tag{2.1}$$

where A, B are the event or random variables, $p(A|B)$ and $p(B|A)$ are respectively posterior and likelihoods of A regarding B. $p(A)$ and $p(B)$ are known as either marginal or prior probability. In the application of non-parametric regression, the structure can also be written with regards to the function, observations and hyperparameters [16] given by

$$\begin{aligned}[Func., Hyperpar.|Observ.] \propto &[Observ.|Func., Hyperpar.] \\ &[Func.|Hyperpar.]\,[Hyperpar.]\,,\end{aligned} \tag{2.2}$$

where $Func.$, $Hyperpar.$ and $Observ.$ represent function, hyperparameters and observations respectively. For regression, a prior for function and hyperparameters is first proposed, after which data points are used for calculating the posterior distribution. The prior and posterior distribution of GP are discussed respectively in section 2.2.1 and 2.2.2. The hyperparameters can also be optimized to best fit the given datasets, which will be discussed in detail in chapter 3.

2.2 Gaussian Process regression

Gaussian Process can be applied to solve either regression or classification problems, both of which can be regarded as approximating function. In regression problem, prior, likelihoods and posterior are all Gaussian, thus making GP for regression more analytically tractable than that for classification, which involves non-Gaussian discrete output [17, Ch.3]. In this section, GP for regression is introduced. The prior data model of GP is first defined, then prediction based on posterior distribution is given.

2.2.1 Prior model

As a Bayesian model, GP assumes a probabilistic model for the underlying function, such that the function is not modeled by a deterministic curve but a random process. The definition of GP is given by the following [17, pp.13].

Definition 2.1 (Gaussian Process)**.** A Gaussian Process is a stochastic process with every entry following Gaussian distribution, of which any finite collection follows a multi-variate Gaussian distribution.

When there is an infinite number of points covering the input region, GP can be used to characterize the distribution of a continuous function, i.e., the underlying function $f(\mathbf{x})$. Similar to single Gaussian distributions, a GP can be characterized by mean function $\mu(\mathbf{x})$ and covariance function $k(\mathbf{x}_m, \mathbf{x}_n)$, and denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(\mu(\mathbf{x}), k(\mathbf{x}_m, \mathbf{x}_n)\right), \tag{2.3}$$

where $\mathbf{x}_m$ and $\mathbf{x}_n$ indicate any pair of input variables. The prior model actually describes an infinite number of possible functions with defined probabilistic properties. Plot (a) in Fig. 2.3 shows some possible realizations of the same GP. The mean function $\mu(\mathbf{x})$ is usually set to zero when knowledge of the underlying function is limited, but can also be set to a specific function to incorporate more prior information. The kernel function $k(\mathbf{x}_m, \mathbf{x}_n)$ describes the correlation of a pair of input variables, for which various choices make GP suitable for function with different properties.

To study GP with noisy dataset, a measurement model is first built as

$$y = f(\mathbf{x}) + w(\mathbf{x}), \tag{2.4}$$

where $w(\mathbf{x}) \sim \mathcal{N}(0, \sigma_n^2)$ is the additive white Gaussian noise (AWGN) with $\sigma_n \in \mathbb{R}$ the noise variance. To train a GP, some measurements are taken in the unknown field under the measurement model. A training set with N data pairs is denoted by $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$ and D is the input dimension. The set of input points is denoted as $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$, and the set of corresponding observed values is $\mathbf{y} = \{y_n\}_{n=1}^N$. Since the number of data points in real application is finite, a zero-mean GP is realized through building a zero-mean multivariate normal distribution, which can be denoted as

$$\mathbf{y} \sim \mathcal{N}(0, \mathbf{K}), \tag{2.5}$$

where $\mathbf{K}$ is covariance matrix generated by

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \cdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \tag{2.6}$$

The kernel function describes the correlation between the random input variables. In the environmental monitoring application, the signals are usually smooth and continuous ones that are suitable for the application of squared exponential kernel, which is also called Radial Basis Function (RBF) kernel. The kernel imposes a smooth relationship such that two input variables are more correlated when they are closer to each other. The RBF kernel is given by

$$k(\mathbf{x}_m, \mathbf{x}_n) = \sigma_f^2 \exp\left[-\frac{1}{2} \cdot (\mathbf{x}_m - \mathbf{x}_n)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_m - \mathbf{x}_n)\right] + \sigma_n^2 \delta(\mathbf{x}_m, \mathbf{x}_n), \tag{2.7}$$

where the prior variance σ_f controls the maximum covariance allowed. For data that varies in a wide range in the output domain, σ_f should be large enough. Fig. 2.2a and 2.2b compares some GP realizations with different σ_f. Characteristic lengths $\boldsymbol{\Sigma} = diag\,(l_1^2, l_2^2, \cdots, l_D^2)$ are related to the effective range of the kernel. A larger l_d in dimension d makes a pair of input points stay correlated in a longer distances. Fig. 2.2a and 2.2c compares examples of GP with different l values. Collectively, prior variance and characteristic lengths can be denoted by $\boldsymbol{\theta} = \{\sigma_f, \boldsymbol{\Sigma}\}$.

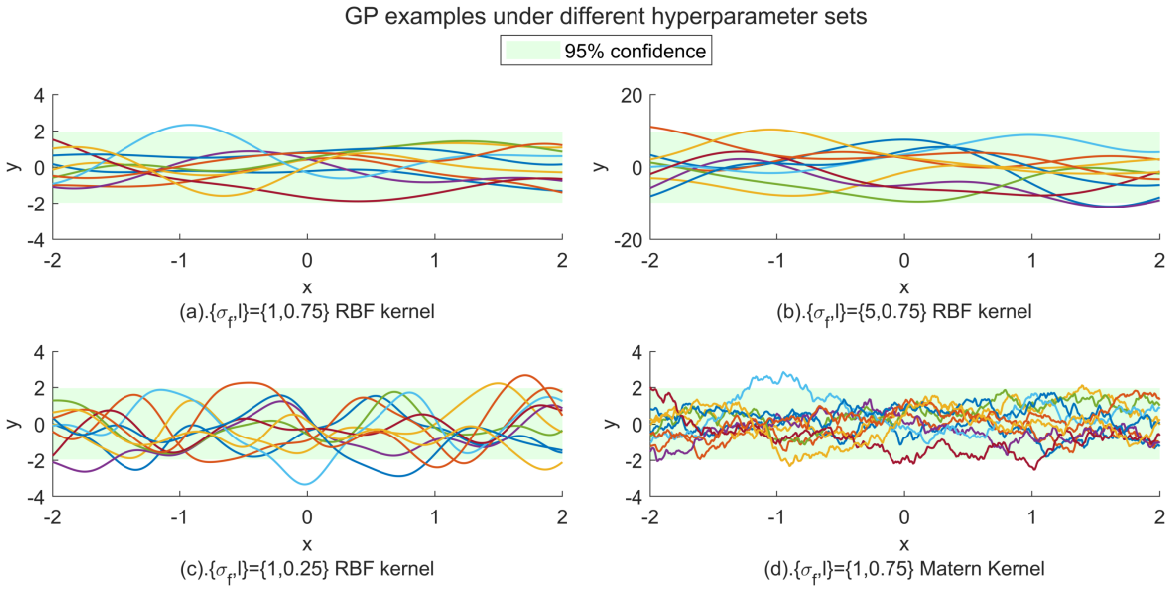


Figure 2.2: Comparison of RBF kernel with different hyperparameter set and Matérn kernel

RBF is actually a special case of a more generalized class of kernel called Matérn kernels, which is argued as a better choice for modeling certain physic phenomenons due to its finite differentiability [18, pp.70]. However, since RBF kernel has already

shown good results in some artificial and real dataset, methods in this work are still developed and tested based on RBF kernel. Fig. 2.2d shows some realizations of GP with Matérn kernel.

2.2.2 Posterior prediction

With the prior model established under hyperparameter set $\boldsymbol{\theta}$, finding the underlying function is equivalent to finding posterior distribution

$$p(f_*|\mathcal{D}, \boldsymbol{\theta}), \tag{2.8}$$

where f_* is the unknown function to be learned. In real application, the regression can be down by recursively performing pointwise prediction of unknown value y_* at $\mathbf{x}_*$ in the region of interest. According to GP definition, the new dataset $\mathcal{D}_* = \{\mathcal{D}, \{\mathbf{x}_*, y_*\}\}$ also follows a multi-variate Gaussian distribution, of which the covariance matrix is an extended matrix based on $\mathbf{K}$ given by

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K} & \mathbf{K}_*^T \\ \mathbf{K}_* & \mathbf{K}_{**} \end{bmatrix}\right) \tag{2.9}$$

, where $\mathbf{K}_* = Cov(y_*, \mathbf{X})$ and $\mathbf{K}_{**} = Cov(y_*, y_*)$ are given by

$$\begin{aligned} \mathbf{K}_* &= \left[k(\mathbf{x}_*, \mathbf{x}_1), k(\mathbf{x}_*, \mathbf{x}_2), \cdots, k(\mathbf{x}_*, \mathbf{x}_N)\right], \\ \mathbf{K}_{**} &= k(\mathbf{x}_*, \mathbf{x}_*). \end{aligned} \tag{2.10}$$

Under Bayesian model, prediction is down by finding the posterior distribution, i.e., the conditional probability $p(y_*|\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})$, or $p(y_*|\mathcal{D})$ for simplicity. According to the definition of GP, this conditional probability also follows the Gaussian distribution [19, pp.337-339], which is given by the posterior

$$p(y_*|\mathcal{D}) \sim \mathcal{N}\left(\mathbf{K}_*\mathbf{K}^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_*\mathbf{K}^{-1}\mathbf{K}_*^T\right). \tag{2.11}$$

With posterior known, the optimal estimation of y_* would then be $\mathbb{E}\{y^*|\mathcal{D}\}$, which is equivalent to an MMSE or MAP estimator. The predictor of posterior mean value is denoted by $\mu(y_*|\mathbf{x}_*, \mathcal{D})$, which can also be denoted as μ in short. Similarly, the posterior variance is denoted by $\sigma^2(y_*|\mathbf{x}_*, \mathcal{D})$, or σ^2 in short. The range of variance is $\sigma^2 \in [0, \sigma_{**}^2]$, where σ_{**}^2 is the prior variance. The variance can be used to indicate the certainty of prediction, for which a minimum value means the most confident prediction, and maximum value indicates totally uncertain prediction.

An example of applying GPR to a toy dataset with zero measurement noise is shown in Fig. 2.3. Some possible process under the prior model without training dataset are shown in Fig. 2.3a, which shows that most function outputs fall in the range of 95% confidence interval. Fig. 2.3b shows the GPR results with 1 known data points. Since the measurement is noiseless, the known point makes the best prediction, the green line, at that point the value of known point. The corresponding prediction variance around the known point is very low, which means high confidence. As predicted points move outside the characteristic length, the best estimation and variance fall back to the prior distribution since no observation is available. Fig. 2.3c and 2.3d shows the GPR

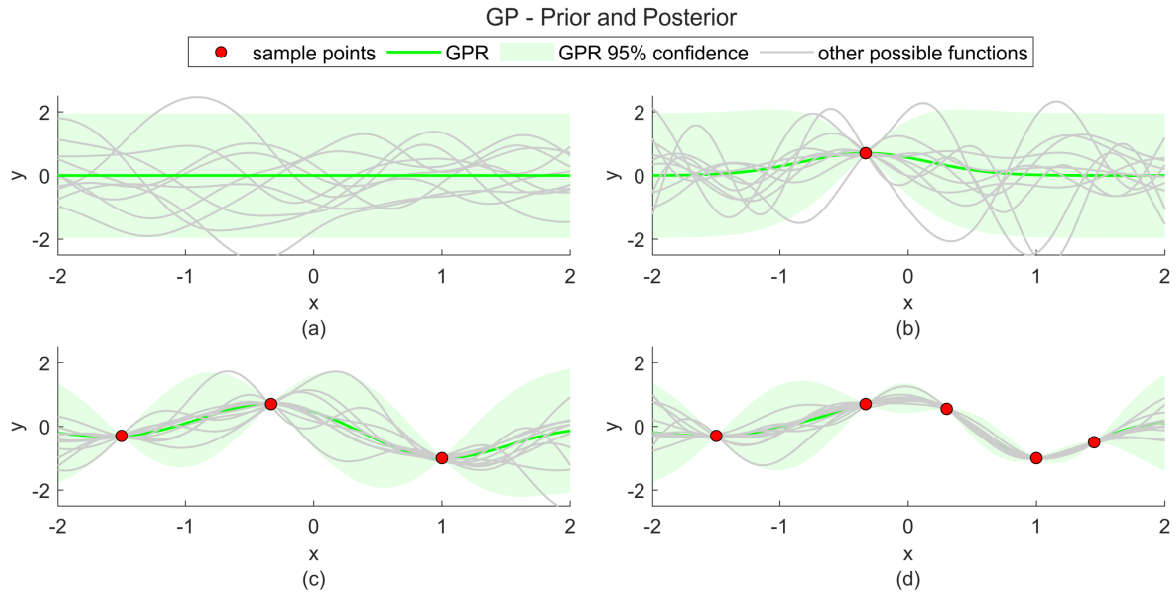


Figure 2.3: 1-D GPR with increasing number of training points under hyperparameter set $\boldsymbol{\theta} = \{\sigma_f, l\} = \{1, 0.5\}$

.

results with more known points added into the training dataset. Similarly, the GPR is more confidence about the prediction at places with known points than the regions without. It should be noticed that, though the green line is the optimal function, GPR actually also allows the possibility of other functions, shown in gray lines. With more and more known points added to the region of interest, the prediction confidence would be increasingly high, revealing the fact that GPR is a data-dependent Machine Learning method.

2.3 Conclusion

In this chapter, the motivation of applying Gaussian Process for distributed environmental monitoring and basics of GP are introduced.

- Motivation: Advantage and disadvantage (section 2.1.1)
  - A GP is a **non-parametric** model that has the advantage of **high robustness, flexibility** and **low reliance** on prior knowledge, which is suitable for the unknown environmental field monitoring task.
  - GP has the disadvantage of **high computational complexity** because of its data-dependent nature. This drawback can be alleviate through distribution in MAS.
- Gaussian Process regression:
  - GPR is based on Bayesian Modelling, which constructs the data model as a **prior model** (i.e., GP), and predicts through **posterior model** based on

observations (Equation (2.2), Figure 2.3).

- The high computational complexity mainly comes from **large matrix inversion**. (Equation (2.11))
- The performance of GP on different datasets is affected by **hyperparameters** (Figure 2.2), which need to be optimized before prediction.

In the next chapter, GP hyperparameter optimization and its distribution are introduced.

Distributed GP Hyperparameter Optimization

3

Although GPR is claimed to be a non-parametric method, this is only true for the data model, which means there is no obvious parameter controlling the data model, e.g., weights for linear model. Hyperparameters still exists in kernel (e.g., the characteristic scalar, largest variance, etc.) , and need to be optimized in training stage of GPR. A proper setting of hyperparameters can largely improve the performance of GPR, while a bad one may result in large error, so careful tuning is important for a good GPR. Finding the best set of hyperparameters for the given dataset is termed as hyperparameter optimization (or model selection).

In section 3.1, some centralized GP hyperparameter optimization algorithms are introduced, including cross-validation and Bayesian model selection. The Bayesian model selection is chosen as the method to be distributed, for which some existing methods are introduced in section 3.2. Some improved algorithms are explained in section 3.3. Finally, simulation results of several selected methods are compared and discussed in section 3.4.

3.1 Centralized GP hyperparameter optimization

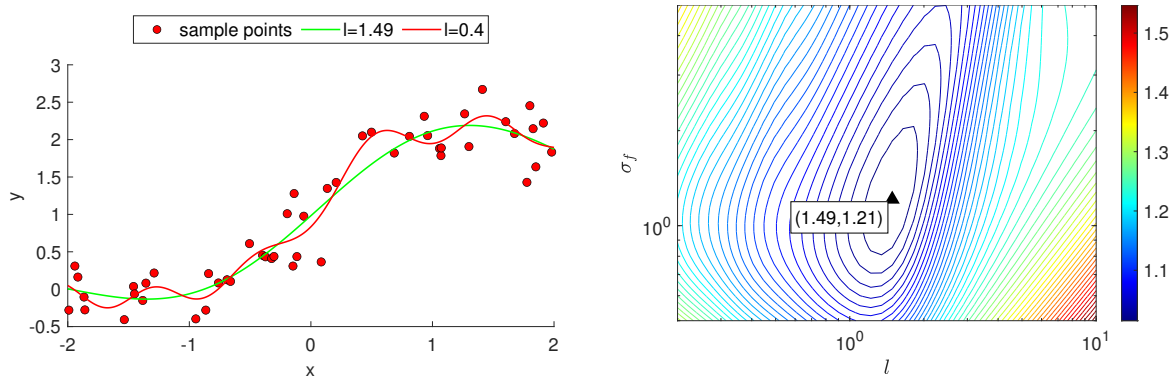
3.1.1 Motivation

As shown in Figure 2.2, the characteristics of GP realizations vary with the change of hyperparameters set or kernel function. From the view of Bayesian Modeling, the hyperparameters affect the prior model for the underlying function, which eventually influence the posterior distribution and the prediction performance. Figure 3.1a shows an example of posterior distribution being affected by the hyperparameters.

The hyperparameters optimization methods tries to find the best set of hyperparameters for the given datasets, which, equivalently speaking, finds the most possible prior data model. Two centralized methods are introduced in [17, Ch. 5], i.e., cross-validation (CV) and Bayesian model selection (BMS). In this section, these methods are briefly introduced, and reasons of choosing BMS is also explained.

3.1.2 Cross-validation

Cross-validation is a widely used methods for evaluating the performance of machine learning algorithms. In the most simple Holdout case, the global dataset $\mathcal{D}$ is divided into two non-overlapping subsets, including a training set $\mathcal{D}_T = \{\mathbf{X}_T, \mathbf{y}_T\}$ and a validation set $\mathcal{D}_V = \{\mathbf{X}_V, \mathbf{y}_V\}$. In the example of GP hyperparameter optimization, a GP model is first trained based on $\mathcal{D}_T$ with covariance matrix $\mathbf{K}_T(\boldsymbol{\theta})$, after which values at input points $\mathbf{X}_V$ are predicted as $\hat{\mathbf{y}}_V$. A cost function $H_{CV}(\mathbf{y}_V, \hat{\mathbf{y}}_V)$ is then applied to measure the differences between the validation output values and predictive means,



(a) Regressions under different l values

(b) Scaled negative Log-likelihood

Figure 3.1: (a). The figure shows two different regression results under two hyperparameter sets with different characteristic lengths l. (b). The figure shows the contour lines of the scaled negative Log-likelihood based on the dataset shown in the left figure. The values of the contours are indicated by the colorbar. The triangle indicates the lowest points among the examined hyperparameters sets, and corresponds to the optimal sets for the given dataset.

for which widely-used choices include squared error [20] and predictive log probability [17, pp. 116]. To find the best hyperparameter set, the cost function is minimized with respect to $\boldsymbol{\theta}$.

In a more robust K-fold CV case, the global dataset is divided into K non-overlapping subsets $\{\mathbf{X}_k, \mathbf{y}_k\}_{k=1}^{K}$. For the k-th iteration of K-fold CV, the validation set is chosen as $\mathcal{D}_{V,k} = \{\mathbf{X}_{V,k}, \mathbf{y}_{V,k}\}$, and corresponding training set $\mathcal{D}_{T,k} = \mathcal{D} \backslash \mathcal{D}_{V,k}$, which is the complementary set of $\mathcal{D}_{V,k}$ with respect to $\mathcal{D}$. With predictive means $\hat{\mathbf{y}}_{V,k}$, the k-th cost function is $H_{CV,k}\left(\mathbf{y}_{V,k}, \hat{\mathbf{y}}_{V,k}\right)$. The global cost function is formulated as $H_{CV}\left(\mathbf{y}_V, \hat{\mathbf{y}_V}\right) = \frac{1}{K}\sum_{k=1}^{K} H_{CV,k}\left(\mathbf{y}_{V,k}, \hat{\mathbf{y}}_{V,k}\right)$ and minimized to find the optimal $\boldsymbol{\theta}$. When $K = N$, the special case of K-fold CV is called Leave-one-out cross-validation (LOO-CV), which brings more robust solution in the cost of higher computational demands.

3.1.3 Bayesian model selection

A hierarchical structure is used for optimizing regression models in [17, Sec. 5.2]. The model describes a probabilistic model for parameter, hyperparameter and model selection with three levels of posterior, likelihood and prior. Model selection based on this structure is called Bayesian model selection. For GP hyperparameter estimation with type of kernel known, the posterior over hyperparameter is

$$p\left(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X}\right) = \frac{p\left(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta}\right)}{p\left(\mathbf{y}|\mathbf{X}\right)}, \tag{3.1}$$

where $p\left(\boldsymbol{\theta}\right)$ is the hyper-prior distribution, $\boldsymbol{\theta}$ is the hyperparameter set, $\mathbf{X}$ contains the input points, and $\mathbf{y}$ contains the output values. The normalizing constant is given by

$$p\left(\mathbf{y}|X\right) = \int p\left(\mathbf{y}|\boldsymbol{\theta}\right) p\left(\boldsymbol{\theta}\right) d\boldsymbol{\theta}. \tag{3.2}$$

To find the best set of hyperparameters given training data, a MAP estimator can be applied to maximize $p(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X})$. Suppose that we do not know prior information about the distribution of $\boldsymbol{\theta}$, the problem can be equivalently solved by an ML estimator maximizing $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$.

Under GP model, marginal likelihood is given by

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = (2\pi)^{-\frac{n}{2}} \cdot |\mathbf{K}(\boldsymbol{\theta})|^{-\frac{1}{2}} \cdot \exp\left(-\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}(\boldsymbol{\theta})\,\mathbf{y}\right), \tag{3.3}$$

where position (i, j) of $\mathbf{K}(\boldsymbol{\theta})$ is calculated by Equation (2.7) under a given set of $\boldsymbol{\theta} = \{\sigma_f, \boldsymbol{\Sigma}\}$. The log-likelihood is then given by $\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}(\boldsymbol{\theta})\,\mathbf{y} - \frac{1}{2}\log|\mathbf{K}(\boldsymbol{\theta})| - \frac{n}{2}\log 2\pi$. Then, the problem can be equivalently formulated as minimizing the negative log-marginal likelihood (NLML) $l(\boldsymbol{\theta})$ as follow

$$\mathcal{P}_0: \quad \min_{\boldsymbol{\theta}} \quad l(\boldsymbol{\theta}) = \mathbf{y}^T\mathbf{K}^{-1}(\boldsymbol{\theta})\,\mathbf{y} + \log|\mathbf{K}(\boldsymbol{\theta})|, \tag{3.4}$$

where the $-\frac{n}{2}\log 2\pi$ term is ignored since an additive constant does not affects the optimization results.

In Figure 3.1b, a scaled version of NLML for dataset shown in Figure 3.1a is visualized through contour lines. Gradient based optimizer can be applied to solve both CV and BMS. Both methods involves the calculation of $\mathbf{K}^{-1}(\boldsymbol{\theta})$ and gradients for all the hyperparameters. The computational complexity for $\mathbf{K}^{-1}(\boldsymbol{\theta})$ is $\mathcal{O}(N^3)$, while computing derivatives are respectively $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$ for LOO-CV and BMS, which indicates that BMS performs slightly better than CV in terms of computational complexity. The CV is argued to have more robust performance under possibly wrong assumption of data model since it directly gives an estimation for the predictive probability [17, pp. 118]. However, it is assumed in this work that the prior data model has been correctly chosen, so the extra robustness provided by CV in model misspecification is unnecessary. Thus, the BMS is preferred for further distribution for its relatively low computational complexity.

3.2 Distributed hyperparameter optimization

Bayesian model selection method is applied to optimize the hyperparameter set, which is equivalent to solving problem $\mathcal{P}_0$ in Equation (3.4). First step of distributing the problem is modify the cost function to a separable form.

Distributed problem formulation The distributed problem is studied with MAS and datasets setting described in section 1.3.2.

Suppose that the NLML can be independently distributed as M local likelihood functions $\{l_1(\boldsymbol{\theta}), l_2(\boldsymbol{\theta}), \cdots, l_M(\boldsymbol{\theta})\}$, where $l_m(\boldsymbol{\theta}) = \mathbf{y}_m^T\mathbf{K}_m(\boldsymbol{\theta})^{-1}\mathbf{y}_m + \log|\mathbf{K}_m(\boldsymbol{\theta})|$. $l_m(\boldsymbol{\theta})$ at agent m is only related to dataset m, i.e. $\{\mathbf{X}_m, \mathbf{y}_m\}$, based on which the local covariance matrix $\mathbf{K}_m(\boldsymbol{\theta})$ is calculated. Also, we assume that the whole network share a same set of hyperparameters. Under these assumptions, $\mathcal{P}_0$ can be approximated by

minimizing the summation of M local likelihood functions

$$\mathcal{P}_1: \quad \min_{\boldsymbol{\theta}} \quad \sum_{m=1}^{M} l_m(\boldsymbol{\theta}). \tag{3.5}$$

Several existing methods can be applied to solve the problem above. The most simple solution for unconstrained minimization problem is naïve gradient descent. After further distribution, problem $\mathcal{P}_1$ can be reformulated as a linearly constrained optimization problem, for which the primal dual method can be applied. Since the cost function is non-convex, the method of multipliers is applied as a special case of proximal point method [2, pp. 23], which increase the robustness for non-convex optimization [21]. Based on alternated direction method of multipliers, a proximal update step is applied to replace exact update [1]. These methods are introduced in detail in the following paragraphs.

3.2.1 Naïve Gradient Descent (nGD) [1]

A centralized gradient descent [22, Cha.9] can be performed by iterating

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \mu \cdot \nabla l(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^t}, \tag{3.6}$$

where μ is step size, and t indicates the iteration number. In the distributed setting, gradient $\nabla l(\boldsymbol{\theta})$ is unknown to a single agent, but can be approximated as $\nabla l(\boldsymbol{\theta}) = \sum_{m=1}^{M} \nabla l_m(\boldsymbol{\theta})$, where $\nabla l_m(\boldsymbol{\theta})$ only depends on data at agent m. Then, a star topology can be applied to the network, in which the agent in the center is called central agent and the rest are worker agents.

For each iteration, the central agent collects gradients $\nabla l_m(\boldsymbol{\theta})$ from worker agents, updates $\boldsymbol{\theta}$ according to Equation (3.6), and sends updated $\boldsymbol{\theta}$ back. After receiving the updated $\boldsymbol{\theta}$, worker agents calculate $\nabla l_m(\boldsymbol{\theta})$ based on the new hyperparameters and start next iteration. At agent m, $\nabla l_m(\boldsymbol{\theta})$ is

$$\begin{aligned} \nabla l_m(\boldsymbol{\theta}) &= \frac{1}{2}\mathbf{y}_m^T \mathbf{K}_m(\boldsymbol{\theta})^{-1} \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \mathbf{K}_m^{-1}(\boldsymbol{\theta}) \mathbf{y}_m - \frac{1}{2} tr\left(\mathbf{K}_m^{-1}(\boldsymbol{\theta}) \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right) \\ &= \frac{1}{2} tr\left(\left(\mathbf{K}_m^{-1}(\boldsymbol{\theta}) \mathbf{y}_m \mathbf{y}_m^T \mathbf{K}_m^{-T}(\boldsymbol{\theta}) - \mathbf{K}_m^{-1}(\boldsymbol{\theta})\right) \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right), \end{aligned} \tag{3.7}$$

where $\frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \sigma_f}, \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial l_1}, \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial l_2}, \cdots, \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial l_D}\right]^T$. With some algebra, the kernel partial derivatives towards σ_f and $\{l_d\}_{d=1}^D$, where D is the input dimension, are given by

$$\begin{aligned} \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial \sigma_f} &= \frac{2}{\sigma_f} \cdot K_m(\boldsymbol{\theta}) \\ \frac{\partial \mathbf{K}_m(\boldsymbol{\theta})}{\partial l_d} &= \sigma_f^2 \cdot dist\left(\mathbf{X}_{m,[d,]}\right) \odot \mathbf{K}_m(\boldsymbol{\theta}), \end{aligned} \tag{3.8}$$

where $dist(\mathbf{X})$ means calculating the covariance matrix regarding all the entries of matrix $\mathbf{X}$. $\mathbf{X}_{[d,]}$ contains only the dth dimension of all the entries, and $\odot$ represents the pointwise multiplication.

Distributed nGD is described in Algorithm 1, where T_{max} is a predefined largest iteration number.

Algorithm 1: nGD for distributed GP hyperparameter optimization [1]

Input: $\{l_m(\cdot)\}_{m=1}^M$, $\mathcal{G}^1$
Output: $\boldsymbol{\theta}$
Initialize: $\boldsymbol{\theta}^0$, $\epsilon > 0$, t, T_{max}, $\mu > 0$
for t=$\{1, 2, \cdots, T_{max}\}$ **do**
  for $m = \{1, 2, \cdots, M\}$ **do**
    $\nabla l_m(\boldsymbol{\theta}^t)$
  end
  for *central agent* **do**
    collects all $\nabla l_m(\boldsymbol{\theta}^t)$ from worker agents.
    $\nabla l(\boldsymbol{\theta}^t) = \sum_{m=1}^{M} \nabla l_m(\boldsymbol{\theta}^t)$. `/*Calculate gradient*/`
    $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \mu \cdot \nabla l(\boldsymbol{\theta}^t)$. `/*Gradient descent*/`
    Broadcast $\boldsymbol{\theta}^{t+1}$ to all worker agents.
  end
  if $\left\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\right\|_2 < \epsilon$ **then**
    Stop iteration.
  end
  $t := t + 1$
end

3.2.2 Alternated direction method of multipliers (ADMM) [2]

In this section, the procedure of developing ADMM for GP hyperparameter optimization problem is shown. The first method is primal-dual method, which is a basic method of solving optimization problems. Considering the con-convex target function, Methods of Multipliers is then introduced. Finally, ADMM is introduced for faster convergence.

Primal-dual method [22] As shown in Equation (3.5), the likelihood function is assumed to be consisted of several parts distributed across the network. Replace $\boldsymbol{\theta}$ with $\boldsymbol{\theta}_m$ for agent m, $l_m(\boldsymbol{\theta}_m)$ is now a local function which does not directly depend on any global variable. To ensure that all agents share the same set of hyperparameters, constraints are added such that $\boldsymbol{\theta}_m = \mathbf{z}$, where $\mathbf{z}$ is a global variable. Problem $\mathcal{P}_1$ can be reformulated as

$$\begin{aligned} \mathcal{P}_2 : \min_{\{\boldsymbol{\theta}_m\}_{m=1}^M, \mathbf{z}} \quad & \sum_{m=1}^{M} l_m(\boldsymbol{\theta}_m) \\ s.t. \quad & \boldsymbol{\theta}_m = \mathbf{z},\ m = 1, 2, \ldots, M, \end{aligned} \tag{3.9}$$

which is a constrained optimization problem.

$\mathcal{P}_2$ minimizes a factorizable target function that can be independently distributed to multiple agents, and simultaneously levies constraints such that specific local variables should finally converge to the same value as their counterparts on other agents. Optimization problems like $\mathcal{P}_2$ are called consensus optimization problem. When the cost function is convex, the problem $\mathcal{P}_2$ can be solved by primal-dual method by constructing dual problem from Lagrangian given by

$$\mathcal{L}\left(\{\boldsymbol{\theta}_m\}_{m=1}^M, \mathbf{z}, \{\boldsymbol{\lambda}_m\}_{m=1}^M\right) = \sum_{m=1}^{M} \left(l_m\left(\boldsymbol{\theta}_m\right) + \langle \boldsymbol{\lambda}_m, \boldsymbol{\theta}_m - \mathbf{z}\rangle\right), \tag{3.10}$$

which, under the knowledge of optimal dual variable values, can be minimized to obtain optimal hyperparameters. Since the dual problem is always concave [22, sec. 5.1.2], the dual ascent methods can be applied to obtain the optimal $\{\boldsymbol{\lambda}_m^*\}_{m=1}^M$. The dual gradient ascent iteration of problem $\mathcal{P}_2$ at time instance t is given by

$$\boldsymbol{\lambda}_m^{t+1} = \boldsymbol{\lambda}_m^t + \mu\left(\boldsymbol{\theta}_m - \mathbf{z}\right), \tag{3.11}$$

where μ is the gradient ascent step size. The primal-dual method apply Lagrangian minimization (primal update) and dual ascent iteratively until convergence to obtain the optimal $\{\boldsymbol{\theta}_m\}_{m=1}^M$.

MM [23] Considering that the $l_m(\boldsymbol{\theta}_m)$ are non-convex functions, the classical primal-dual methods are expected to be not robust. Methods of Multipliers (MM) can be introduced to achieve higher robustness. MM is developed by applying proximal point methods in the optimization of Lagrangian [23], where the new Lagrangian equation is called augmented Lagrangian. For MM, the augmented Lagrangian of $\mathcal{P}_2$ is given by

$$\mathcal{L}\left(\{\boldsymbol{\theta}_m\}_{m=1}^M, \mathbf{z}, \{\boldsymbol{\lambda}_m\}_{m=1}^M\right) = \sum_{m=1}^{M} \left(l_m\left(\boldsymbol{\theta}_m\right) + \langle \boldsymbol{\lambda}_m, \boldsymbol{\theta}_m - \mathbf{z}\rangle + \frac{\rho}{2}\left\|\boldsymbol{\theta}_m - \mathbf{z}\right\|_2^2\right), \tag{3.12}$$

where $\rho \in \mathbb{R}^+$ controls the weight of the quadratic term. The update iterations of MM at time instance t is given by

$$\left(\mathbf{z}^{t+1}, \{\boldsymbol{\theta}_m^{t+1}\}_{m=1}^M\right) = \arg\min \mathcal{L}\left(\{\boldsymbol{\theta}_m^t\}_{m=1}^M, \mathbf{z}^t, \{\boldsymbol{\lambda}_m^t\}_{m=1}^M\right) \tag{3.13a}$$

$$\boldsymbol{\lambda}_m^{t+1} = \boldsymbol{\lambda}_m^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}^{t+1}\right), \tag{3.13b}$$

Although MM provides higher robustness, performing exact joint minimization in Equation (3.13a) for each iteration is unnecessary and time-consuming.

ADMM [2] By restricting the iteration number of joint estimation to be 1, $\mathcal{P}_2$ can be solved by consensus ADMM. Update equation of ADMM at iteration t is given by

$$\mathbf{z}^{t+1} = \arg\min_{\mathbf{z}} \left(\sum_{m=1}^{M} -\left\langle \boldsymbol{\lambda}_m^t, \mathbf{z}\right\rangle + \frac{\rho}{2}\left\|\boldsymbol{\theta}_m^t - \mathbf{z}\right\|_2^2\right) \tag{3.14a}$$

$$\boldsymbol{\theta}_m^{t+1} = \arg\min_{\boldsymbol{\theta}_m} \left(l_m\left(\boldsymbol{\theta}_m\right) + \left\langle \boldsymbol{\lambda}_m^t, \boldsymbol{\theta}_m\right\rangle + \frac{\rho}{2}\left\|\boldsymbol{\theta}_m - \mathbf{z}^{t+1}\right\|_2^2\right) \tag{3.14b}$$

$$\boldsymbol{\lambda}_m^{t+1} = \boldsymbol{\lambda}_m^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}^{t+1}\right), \tag{3.14c}$$

where $\mathbf{z}$ update is equivalent to $\mathbf{z}^{t+1} = \frac{1}{M}\sum_{m=1}^{M}\left(\boldsymbol{\theta}_m^t + \frac{1}{\rho}\boldsymbol{\lambda}_m^t\right)$. More details of performing ADMM are given in Algorithm 2.

Algorithm 2: ADMM for distributed GP hyperparameter optimization [2]

Input: $\{l_m(\cdot)\}_{m=1}^M$, $\mathcal{G}$
Output: $\{\boldsymbol{\theta}_m\}_{m=1}^M$
Initialize: $\{\boldsymbol{\theta}_m^0\}_{m=1}^M$, $\mathbf{z}^0 = \mathbf{1}$, $\{\lambda_m^0\}_{m=1}^M$, T_{max}, $\epsilon > 0$, ρ
for $t = \{1, 2, \cdots, T_{max}\}$ **do**
  for *central agent* **do**
    $\mathbf{z}^{t+1} = \frac{1}{M}\sum_{m=1}^{M}\left(\boldsymbol{\theta}_m^t + \frac{1}{\rho}\boldsymbol{\lambda}_m^t\right)$ `/*Centralized primal z update*/`
    Broadcast $\mathbf{z}^{t+1}$
  end
  for *worker agents* $m = \{1, 2, \cdots, M\}$ **do**
    $\boldsymbol{\theta}_m^{t+1} = \arg\min_{\boldsymbol{\theta}} l_m(\boldsymbol{\theta}) + \langle \boldsymbol{\lambda}_m^t, \boldsymbol{\theta}_m \rangle + \frac{\rho}{2}\left\|\boldsymbol{\theta} - \mathbf{z}^{t+1}\right\|_2^2$ `/*Primal θ update by nGD*/`
    $\boldsymbol{\lambda}_m^{t+1} = \boldsymbol{\lambda}_m^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}^{t+1}\right)$ `/*Dual update*/`
    Send $\boldsymbol{\theta}_m^{t+1}$ and $\boldsymbol{\lambda}_m^{t+1}$ to central agent
  end
  if $\left\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^t\right\|_2 < \epsilon$ **then**
    Stop iteration.
  end
end

Fully-distributed ADMM ($\mathbf{ADMM_{fd}}$) To fully distribute the algorithm over network without central agent, $\mathcal{P}_2$ can be modified as

$$\begin{aligned} \mathcal{P}_3 : \min_{\{\boldsymbol{\theta}_m\}_{m=1}^M, \{\mathbf{z}_{mn}\}_{(m,n)\in\mathcal{E}}} \quad & \sum_{m=1}^{M} l_m(\boldsymbol{\theta}_m) \\ s.t. \quad & \boldsymbol{\theta}_m = \mathbf{z}_{mn},\ \forall (m,n) \in \mathcal{E}, \end{aligned} \tag{3.15}$$

where $\mathbf{z}_{mn}$ is variable associated with edge between agents m and n, and there is $\mathbf{z}_{mn} = \mathbf{z}_{nm}$. Also, recall that $\mathcal{E}$ represents the edges of MAS. Now problem $\mathcal{P}_3$ is an edge-based constrained problem which only involves local variables that are directly associated with a single agent and its neighbors. The augmented Lagrangian now becomes

$$\begin{aligned} &\mathcal{L}\left(\{\boldsymbol{\theta}_m\}_{m=1}^M, \{\mathbf{z}_{mn}, \boldsymbol{\lambda}_{mn}\}_{(m,n)\in\mathcal{E}}\right) = \\ &\sum_{m=1}^{M}\left(l_m(\boldsymbol{\theta}_m) + \sum_{n\in\mathcal{N}(m)}\left(\langle \boldsymbol{\lambda}_{mn}, \boldsymbol{\theta}_m - \mathbf{z}_{mn}\rangle + \frac{\rho}{2}\left\|\boldsymbol{\theta}_m - \mathbf{z}_{mn}\right\|_2^2\right)\right), \end{aligned} \tag{3.16}$$

where $\boldsymbol{\lambda}_{mn}$ are dual variables associated with edge (m, n) stored on agent m. The corresponding update iterations of fully-distributed ADMM is given by

$$\mathbf{z}_{mn}^{t+1} = \frac{1}{2}\left(\rho^{-1}\left(\boldsymbol{\lambda}_{mn}^{t} + \boldsymbol{\lambda}_{nm}^{t}\right) + \boldsymbol{\theta}_{m}^{t} + \boldsymbol{\theta}_{n}^{t}\right) \tag{3.17a}$$

$$\boldsymbol{\theta}_{m}^{t+1} = \arg\min_{\boldsymbol{\theta}_m} \left(l_m\left(\boldsymbol{\theta}_m\right) + \sum_{n\in\mathcal{N}(m)} \left(\left\langle \boldsymbol{\lambda}_{mn}^{t}, \boldsymbol{\theta}_m \right\rangle + \frac{\rho}{2}\left\| \boldsymbol{\theta}_m - \mathbf{z}_{mn}^{t+1} \right\|_2^2 \right) \right) \tag{3.17b}$$

$$\boldsymbol{\lambda}_{mn}^{t+1} = \boldsymbol{\lambda}_{mn}^{t} + \rho\left(\boldsymbol{\theta}_{m}^{t+1} - \mathbf{z}_{mn}^{t+1}\right). \tag{3.17c}$$

in which Equation (3.17b) requires exact update through nGD.

3.2.3 Proximal ADMM (pxADMM)

It can be noticed from Equation (3.3) and (3.7) that $l(\boldsymbol{\theta})$ is a function about $\boldsymbol{\theta}$ that the close form minimum point is hard to find, which results in the time-consuming minimization problems in Equation (3.14b) and (3.17b).

The proximal ADMM (pxADMM) adapts the idea that an accurate update of $\boldsymbol{\theta}_m$ is unnecessary for each iteration, and can be approximately updated to reduce computational complexity. First order approximation to $l_m\left(\boldsymbol{\theta}\right)$ around point $\mathbf{z}$ can be applied as

$$l_m\left(\boldsymbol{\theta}_m\right) \approx l_m\left(\mathbf{z}\right) + \nabla l_m\left(\mathbf{z}\right)\left(\boldsymbol{\theta}_m - \mathbf{z}\right). \tag{3.18}$$

Based on Equation (3.18), primal update of $\boldsymbol{\theta}_m$ is

$$\boldsymbol{\theta}_{m}^{t+1} = \mathbf{z}^{t+1} - \frac{1}{\rho + L}\left(\nabla l_m(\mathbf{z}^{t+1}) + \boldsymbol{\lambda}_{m}^{t}\right). \tag{3.19}$$

To fit the algorithm into network without star topology, the centralized $\mathbf{z}$ update should be distributed. Introduce new variables $\mathbf{z}_m, m \in \{1, 2, \cdots, M\}$ for all agents. Step 4 in Algorithm 2 can be replaced by a local consensus that only update $\mathbf{z}_m$ based on $\mathbf{z}_n, \forall n \in \mathcal{N}(m)$.

3.3 Proposed fully-distributed hyperparameter optimization

3.3.1 Fully-distributed proximal ADMM (pxADMM$_{\text{fd}}$)

In this section, two fully-distributed versions of pxADMM are introduced to solve problem $\mathcal{P}_3$. The first method, named pxADMM$_{\text{fd}}$, is developed based on ADMM$_{\text{fd}}$, of which the convergence have been proved [24]. The second method is adapted based on both ADMM$_{\text{fd}}$ and pxADMM methods. Though the convergence of the method has not been strictly proven, the experimental results shows faster convergence compared to the first method.

pxADMM$_{\text{fd}}$ based on ADMM$_{\text{fd}}$ The pxADMM$_{\text{fd}}$ can be developed based on ADMM$_{\text{fd}}$ through approximating Equation (3.17b). To approximate the target function as shown in Equation 3.18, a center point has to be chosen, around which the

function is approximated. Since a global $\mathbf{z}$ does not exists for fully-distributed setting, simply choosing $\mathbf{z}$ as Equation (3.18) is not plausible.

An alternative choice is using the $\boldsymbol{\theta}_m^t$ as the point. Under this setting, the approximation is given by

$$l_m\left(\boldsymbol{\theta}_m\right) \approx l_m\left(\boldsymbol{\theta}_m^t\right) + \nabla l_m\left(\boldsymbol{\theta}_m^t\right)\left(\boldsymbol{\theta}_m - \boldsymbol{\theta}_m^t\right), \tag{3.20}$$

which gives the $\boldsymbol{\theta}_m$ update as

$$\boldsymbol{\theta}_m^{t+1} = \frac{1}{L + \rho\left|\mathcal{N}(m)\right|}\left(\sum_{n\in\mathcal{N}(m)}\left(\rho\mathbf{z}_{mn} - \boldsymbol{\lambda}_{mn}\right) + L\boldsymbol{\theta}_m^t - \nabla l_m\left(\boldsymbol{\theta}_m^t\right)\right), \tag{3.21}$$

where $L \in \mathbb{R}^+$ is a constant that satisfies $\|\nabla l_m\left(\boldsymbol{\theta}_m\right) - \nabla l_m\left(\boldsymbol{\theta}'_m\right)\| \leq L\|\boldsymbol{\theta}_m - \boldsymbol{\theta}'_m\|$ for all $\boldsymbol{\theta}_m$ and $\boldsymbol{\theta}'_m$. With Equation 3.17b from fully distributed ADMM replaced by Equation 3.21, the pxADMM is extended to the fully-distributed pxADMM$_{\text{fd}}$ method. The fully distributed proximal ADMM algorithm is described in Algorithm 3.

Algorithm 3: pxADMM$_{\text{fd}}$ for fully distributed GP hyperparameter optimization

Input: $\{l_m(\cdot)\}_{m=1}^M$, $\mathcal{G}$
Output: $\{\boldsymbol{\theta}_m\}_{m=1}^M$
Initialize: $\{\boldsymbol{\theta}_m^0\}_{m=1}^M$, $\{\mathbf{z}_m^0\}_{m=1}^M$, $\{\lambda_m^0\}_{m=1}^M$, T_{max}, $\epsilon > 0$, ρ, L
for *iteration* $t = \{1, 2, \cdots, T_{max}\}$ **do**
  for *agent* $m = \{1, 2, \cdots, M\}$ **do**
    for $n \in \mathcal{N}(m)$, *everywhere* **do**
      $\mathbf{z}_{mn}^{t+1} = \frac{1}{2}\left(\rho^{-1}\left(\boldsymbol{\lambda}_{mn}^t + \boldsymbol{\lambda}_{nm}^t\right) + \boldsymbol{\theta}_m^t + \boldsymbol{\theta}_n^t\right)$ `/*Primal` $\mathbf{z}_m$ `update*/`
    end
    $\boldsymbol{\theta}_m^{t+1} = \frac{1}{L+\rho|\mathcal{N}(m)|}\left(\sum_{n\in\mathcal{N}(m)}\left(\rho\mathbf{z}_{mn} - \boldsymbol{\lambda}_{mn}\right) + L\boldsymbol{\theta}_m^t - \nabla l_m\left(\boldsymbol{\theta}_m^t\right)\right)$ `/*Primal` $\boldsymbol{\theta}_m$ `update*/`
    for $n \in \mathcal{N}(m)$ **do**
      $\boldsymbol{\lambda}_{mn}^{t+1} = \boldsymbol{\lambda}_{mn}^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}_{mn}^{t+1}\right)$ `/*Dual update*/`
      Send $\boldsymbol{\theta}_m^{t+1}, \boldsymbol{\lambda}_{mn}^{t+1}, \mathbf{z}_{mn}^{t+1}$ to $\mathcal{N}(m)$
    end
  end
  if $\left\|\boldsymbol{\theta}_m^{t+1} - \boldsymbol{\theta}_m^t\right\|_2 < \epsilon, \forall m \in \{1, 2, \cdots, M\}$ **then**
    Stop iteration.
  end
end

pxADMM$^*_{\text{fd}}$ Considering the problem that a global $\mathbf{z}$ variable does not exist for approximation, another intuitive solution is introducing an auxiliary variable in each agent m as $\boldsymbol{\zeta}_m$, which is used as the center for approximation.

Algorithm 4: pxADMM$^{*}_{\text{fd}}$ for fully distributed GP hyperparameter optimization

Input: $\{l_m(\cdot)\}_{m=1}^M$, $\mathcal{G}$
Output: $\{\boldsymbol{\theta}_m\}_{m=1}^M$
Initialize: $\{\boldsymbol{\theta}_m^0\}_{m=1}^M$, $\{\mathbf{z}_m^0\}_{m=1}^M$, $\{\lambda_m^0\}_{m=1}^M$, T_{max}, $\epsilon > 0$, ρ, L
for *iteration* $t = \{1, 2, \cdots, T_{max}\}$ **do**
  for *agent* $m = \{1, 2, \cdots, M\}$ **do**
    for $n \in \mathcal{N}(m)$, *everywhere* **do**
      $\mathbf{z}_{mn}^{t+1} = \frac{1}{2}\left(\rho^{-1}\left(\boldsymbol{\lambda}_{mn}^t + \boldsymbol{\lambda}_{nm}^t\right) + \boldsymbol{\theta}_m^t + \boldsymbol{\theta}_n^t\right)$ /*Primal $\mathbf{z}_m$ update*/
    end
    $\boldsymbol{\zeta}_m^{t+1} = \frac{1}{1+|\mathcal{N}(m)|}\left(\boldsymbol{\zeta}_m^t + \sum_{n\in\mathcal{N}(m)} \mathbf{z}_{mn}^{t+1}\right)$ /*Auxiliary $\boldsymbol{\zeta}_m$ update*/
    $\boldsymbol{\theta}_m^{t+1} = \boldsymbol{\zeta}^{t+1} - \frac{1}{\rho+L}\left(\nabla l_m(\boldsymbol{\zeta}^{t+1}) + \boldsymbol{\Lambda}_m^t\right)$ /*Primal $\boldsymbol{\theta}_m$ update*/
    for $n \in \mathcal{N}(m)$, *everywhere* **do**
      $\boldsymbol{\lambda}_{mn}^{t+1} = \boldsymbol{\lambda}_{mn}^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}_{mn}^{t+1}\right)$ /*Dual update*/
      Send $\boldsymbol{\theta}_m^{t+1}, \boldsymbol{\lambda}_{mn}^{t+1}, \mathbf{z}_{mn}^{t+1}$ to $\mathcal{N}(m)$
    end
    $\boldsymbol{\Lambda}_m^{t+1} = \frac{1}{1+|\mathcal{N}(m)|}\left(\boldsymbol{\Lambda}_m^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \boldsymbol{\zeta}_m^{t+1}\right) + \sum_{n\in\mathcal{N}(m)} \boldsymbol{\lambda}_{mn}^{t+1}\right)$ /*Auxiliary $\boldsymbol{\Lambda}_m$ update*/
  end
  if $\left\|\boldsymbol{\theta}_m^{t+1} - \boldsymbol{\theta}_m^t\right\|_2 < \epsilon, \forall m \in \{1, 2, \cdots, M\}$ **then**
    Stop iteration.
  end
end

To combine the information from neighbor nodes, the following method of constructing and updating $\boldsymbol{\zeta}_m$ is proposed that

$$\boldsymbol{\zeta}_m^{t+1} = \frac{1}{1+|\mathcal{N}(m)|}\left(\boldsymbol{\zeta}_m^t + \sum_{n\in\mathcal{N}(m)} \mathbf{z}_{mn}^{t+1}\right), \tag{3.22}$$

which is equivalently a special case of distributed consensus filter shown in Equation (4.19). Similarly, auxiliary variable $\boldsymbol{\Lambda}_m$ is proposed to combine $\boldsymbol{\lambda}_{mn}$. The $\boldsymbol{\Lambda}_m$ is maintained as

$$\boldsymbol{\Lambda}_m^{t+1} = \frac{1}{1+|\mathcal{N}(m)|}\left(\boldsymbol{\Lambda}_m^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \boldsymbol{\zeta}_m^{t+1}\right) + \sum_{n\in\mathcal{N}(m)} \boldsymbol{\lambda}_{mn}^{t+1}\right), \tag{3.23}$$

which can also be regarded as a special case based on Equation (4.19).

The update of $\boldsymbol{\theta}_m$ is then adapted based on Equation (3.19) as

$$\boldsymbol{\theta}_m^{t+1} = \boldsymbol{\zeta}^{t+1} - \frac{1}{\rho+L}\left(\nabla l_m(\boldsymbol{\zeta}^{t+1}) + \boldsymbol{\Lambda}_m^t\right). \tag{3.24}$$

Details of the update iterations of pxADMM$^*_{\text{fd}}$ are shown in Algorithm 4. In Section 3.4, it will be shown from the simulation results that the pxADMM$^*_{\text{fd}}$ has a better performance than pxADMM$_{\text{fd}}$ in term of iterations needed for convergence.

3.3.2 Asynchronous proximal ADMM (pxADMM$_{\text{async}}$)

With synchronous algorithm, clock synchronization across the network is required, which is an extra burden for the system, especially in large network or time varying network. Additionally, the agents may varies in their sampling behavior, thus maintain datasets of different size. Since the optimization includes calculation of matrix gradient with $\mathcal{O}(N^2)$ complexity, a small difference may be amplified in terms of computation time for each iteration. If the whole network has to wait for the slowest agent, a lot of time would be wasted. An asynchronous algorithm can alleviate the above issues.

An early version of asynchronous ADMM is proposed by Zhang and Kwok [25]. The algorithm proposed requires only partial synchronization instead of a complete one for each iteration. Two tricks are introduced to keep balance between asynchronous behavior and slow agents update. A star topology is required for the proposed method. Wei and Ozdaglar propose another strategy in [26]. For each iteration, they consider that only part of the constraints and cost functions are involved in the update, while the others are ignored. A similar algorithm is proposed by Iutzeler et al. [27], in which only the data from activated subset is used for update.

In general, asynchronous ADMM can be concluded as partially activated ADMM based on different strategy. If an agent is activated, then it is not performing update of primal or dual variables, but is able to share updates to neighbors. It should be noticed that the agents are assumed to always be able to receive and store information from neighbors. For iteration t in asynchronous ADMM, a subset $\mathcal{A}^t$ of the entire graph is activated. Different asynchronous ADMM algorithms meanly differs in the method of activating the subset. Before introducing fully-distributed asynchronous optimization, $\mathcal{A}^t_m$ is introduced to denote the activated neighbors of agent m in iteration t. The inactivated neighbors are denoted by the complementary set $[\mathcal{A}^t_m]^{\mathcal{C}} = \mathcal{N}(m) \backslash \mathcal{A}^t_m$.

Two strategies are applied to develop asynchronous algorithm, which are known as partial barrier and bounded delay [25].

The partial barrier strategy tries to reduce the waiting time of a fast agent by specifying that an agent only has to wait for a minimum number of $S >= 1$ updates from neighbors. With $S = 1$, the algorithm is equivalent to an edge-based asynchronous algorithm. The maximum value of S on agent m is $|\mathcal{N}(m)|$, which is equivalent to synchronous algorithm.

In contrary to partial barrier that introduce asynchronous behavior, the bounded delay strategy ensures a certain level of synchronous over the general procedure of convergence. It restricts the asynchronous behavior by forcing a pair of agents to communicate with each other at least one time after $\tau > 1$ iterations. The partial barrier assures that every links can be activated occasionally so that information can be exchanged through out the convergence. A special case in which the partial barrier is especially useful is when an agent only has one edge. The agent with only one edge relies heavily on that edge to join the consensus algorithm in the rest of the network. If

the edge is seldom activated, the agent is almost sure to converge to a value nonidentical to the rest of the network. Details of the algorithm are shown in Algorithm 5.

Algorithm 5: Asynchronous proximal ADMM for fully distributed GP hyperparameter optimization

Input: $\{l_m(\cdot)\}_{m=1}^M$, $\mathcal{G}$
Output: $\{\boldsymbol{\theta}_m\}_{m=1}^M$
Initialize: $\{\boldsymbol{\theta}_m^0\}_{m=1}^M$, $\{\mathbf{z}_m^0\}_{m=1}^M$, $\{\lambda_m^0\}_{m=1}^M$, T_{max}, ρ, L
  stop criteria $\epsilon > 0$
for *iteration* $t = \{1, 2, \cdots, T_{max}\}$ **do**
  Select a subset $\mathcal{A}^t$ of nodes
  for *agent* $m \in \mathcal{A}^t$ **do**
    $\mathcal{A}_m^t = \mathcal{N}(m) \cap \mathcal{A}^t$ /*Decide activated neighborhoos*/
    get $\left[\mathcal{A}_m^t\right]^{\mathcal{C}}$ /*Decide inactivate neighborhood*/
    Denote the outdated data from agent $\bar{n} \in \left[\mathcal{A}_m^t\right]^{\mathcal{C}}$ as $\left\{\boldsymbol{\theta}_{\bar{n},old}^t, \boldsymbol{\lambda}_{\bar{n}m,old}^t\right\}$
    for $n \in \mathcal{A}_m^t$ **do**
      $\mathbf{z}_{mn}^{t+1} = \frac{1}{2}\left(\rho^{-1}\left(\boldsymbol{\lambda}_{mn}^t + \boldsymbol{\lambda}_{nm}^t\right) + \boldsymbol{\theta}_m^t + \boldsymbol{\theta}_n^t\right)$ /*Primal $\mathbf{z}_{mn}$ update for active agents*/
    end
    for $\bar{n} \in \left[\mathcal{A}_m^t\right]^{\mathcal{C}}$ **do**
      $\mathbf{z}_{m\bar{n}}^{t+1} = \frac{1}{2}\left(\rho^{-1}\left(\boldsymbol{\lambda}_{m\bar{n}}^t + \boldsymbol{\lambda}_{\bar{n}m,old}^t\right) + \boldsymbol{\theta}_m^t + \boldsymbol{\theta}_{\bar{n},old}^t\right)$ /*Primal $\mathbf{z}_{m\bar{n}}$ update for inactive agents*/
    end
    $\boldsymbol{\theta}_m^{t+1} =$
    $\frac{1}{L+\rho|\mathcal{N}(m)|}\left(\sum_{n\in\mathcal{N}(m)}(\rho\mathbf{z}_{mn} - \boldsymbol{\lambda}_{mn}) + \sum_{\bar{n}\in\mathcal{N}(m)}(\rho\mathbf{z}_{m\bar{n}} - \boldsymbol{\lambda}_{m\bar{n}}) + L\boldsymbol{\theta}_m^t - \nabla l_m\left(\boldsymbol{\theta}_m^t\right)\right)$
    /*Primal $\boldsymbol{\theta}_m$ update*/
    for $n \in \mathcal{A}_m^t$ **do**
      $\boldsymbol{\lambda}_{mn}^{t+1} = \boldsymbol{\lambda}_{mn}^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}_{mn}^{t+1}\right)$ /*Dual update for active agents*/
      Send $\boldsymbol{\theta}_m^{t+1}, \boldsymbol{\lambda}_{mn}^{t+1}, \mathbf{z}_{mn}^{t+1}$ to $\mathcal{A}_m^t$
    end
    for $\bar{n} \in \left[\mathcal{A}_m^t\right]^{\mathcal{C}}$ **do**
      $\boldsymbol{\lambda}_{m\bar{n}}^{t+1} = \boldsymbol{\lambda}_{m\bar{n}}^t + \rho\left(\boldsymbol{\theta}_m^{t+1} - \mathbf{z}_{m\bar{n}}^{t+1}\right)$ /*Dual update for inactive agents*/
      Send $\boldsymbol{\theta}_m^{t+1}, \boldsymbol{\lambda}_{m\bar{n}}^{t+1}, \mathbf{z}_{m\bar{n}}^{t+1}$ to $\left[\mathcal{A}_m^t\right]^{\mathcal{C}}$
    end
  end
  if $\left\|\boldsymbol{\theta}_m^{t+1} - \boldsymbol{\theta}_m^t\right\|_2 < \epsilon, \forall m \in \{1, 2, \cdots, M\}$ **then**
    Stop iteration.
  end
end

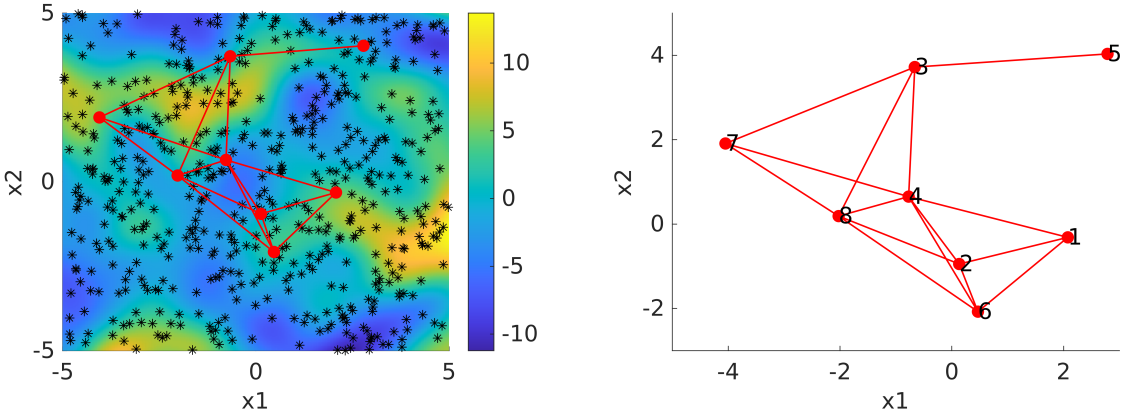
3.4 Simulation&Discussion

3.4.1 Artificial dataset

The following simulation results are carried out in the 2D GP fields randomly generated based on hyperparameter set $\boldsymbol{\theta} = \{\sigma_f = 5, l_1 = 1, l_2 = 1\}$ in the range $[-5, 5] \times [-5, 5]$. The sampling points are sampled under an zero-mean AWGN with $\sigma_n = \sqrt{0.1}$.

There are in total $M = 8$ agents randomly generated inside the field. Each agent owns a local dataset of size $N_m = 70$ that is randomly divided from the entire dataset. The stop criteria $\epsilon = 1 \times 10^{-6}$. Figure 3.2 shows MAS topology and underlying scalar field.

The proposed hyperparameter optimization algorithms, i.e., pxADMM$_{\text{fd}}$, pxADMM$^*_{\text{fd}}$ and their asynchronous versions, will be simulated with the aforementioned setting and compared with existing algorithms, i.e., nGD, ADMM, ADMM$_{fd}$ and pxADMM. It should be noticed that, for simplicity, the results of fully-distributed methods in the figures are plotted by only the results from agent with code 1 in the network.



(a) Topology, sampling points and background

(b) Topology with agent codes

Figure 3.2: (a). The figure shows MAS and underlying scalar field generated by 2D stationary GP. The field is shown as background with magnitudes indicated by color according to the colorbar. The red dots shows the positions of agents. The communication links are plotted as red lines connecting pairs of agents. The black asterisks scattered around the field shows the position of sampling points. (b). The figure shows the topology structure of MAS, in which the numbers indicate the code of agents from 1 to M, i.e., 8.

nGD, ADMM, ADMM$_{\text{fd}}$, pxADMM The convergence behavior of nGD, ADMM and ADMM$_{\text{fd}}$ are respectively shown in Figure A.1, A.2 and A.3 in Appendix A.1. For nGD, the step size is $\mu = 1 \times 10^{-5}$, and the maximum iteration number is $T_{max} = 11000$. For ADMM, the maximum iteration is $T_{max} = 1500$, and the maximum iteration number for solving local $\boldsymbol{\theta}_m$ minimization problem is $T_{max,inner} = 50$. The hyperparameter optimization results are listed in Table 3.2.

Synchronous pxADMM$_{\text{fd}}$ and pxADMM$^*_{\text{fd}}$ The parameter values and initialization of variables are shown in Table 3.1.

Table 3.1: Parameters & Variables of pxADMM

Parameters & variables	Values	Only used in
ρ	$8 \times 70 \times 0.3$	
L	$8 \times 70 \times 0.7$	
T_{max}	11000	
$\boldsymbol{\theta}_m^0 = \{\sigma_f^0, \mathbf{l}^0, \sigma_n^0\}$	$\{3, [2,2]^T, 1\}$	
ϵ	1×10^{-5}	
$\boldsymbol{\lambda}_m^0$	$[1,1,1,1]^T$	
$\mathbf{z}_m^0$	$\boldsymbol{\theta}_m^0$	pxADMM
$\mathbf{z}_{mn}^0, \mathbf{z}_{nm}^0, \boldsymbol{\lambda}_{mn}^0, \boldsymbol{\lambda}_{nm}^0$	$[0,0,0,0]^T$	pxADMM$_{\text{fd}}$, pxADMM$^*_{\text{fd}}$
S_m	$\min(3, \lvert\mathcal{N}(m)\rvert)$	pxADMM$_{\text{async}}$, pxADMM$^*_{\text{async}}$
τ	10	pxADMM$_{\text{async}}$, pxADMM$^*_{\text{async}}$

The simulation results of centralized pxADMM, pxADMM$_{\text{fd}}$ and pxADMM$^*_{\text{fd}}$ are respectively shown in Figure A.4, A.5 and A.6 in Appendix A.1. The hyperparameter optimization results are listed in Table 3.2.

Table 3.2: Results of hyperparameters optimization

Methods	σ_f	$\mathbf{l}$	σ_n
nGD	4.2320	[0.9871;0.9509]	0.3690
ADMM	4.2652	[0.9901;0.9541]	0.3691
ADMM$_{\text{fd}}$	4.1996	[0.9842;0.9478]	0.3689
pxADMM	4.2650	[0.9901;0.9541]	0.3691
pxADMM$_{\text{fd}}$	4.2572	[0.9886;0.9462]	0.3822
pxADMM$^*_{\text{fd}}$	4.2625	[0.9898;0.9538]	0.3691
pxADMM$_{\text{async}}$	4.2574	[0.9886;0.9462]	0.3822
pxADMM$^*_{\text{async}}$	4.2631	[0.9899,0.9539]	0.3691

Based on the results in the figures and table, the proposed synchronized fully-distributed pxADMM methods pxADMM$_{\text{fd}}$ and pxADMM$^*_{\text{fd}}$ converge to the expected hyperparameters in similar behaviors as the centralized pxADMM.

As shown in Figure A.5, the convergence of pxADMM$_{\text{fd}}$ is reached around 1.6×10^3 iterations. The number of iterations needed is larger than that of pxADMM, which is expected considering that the fully-distributed version does not have a global $\mathbf{z}$ variable that plays a strong role in helping agents reach consensus. Since the auxiliary variables $\mathbf{z}_{mn}$ are spread on the edges across the network, more iterations are needed so that an agent can be influenced from the agent on the far side of the network.

Results in Figure A.6 shows that pxADMM$^*_{\text{fd}}$ converges after approximately 6×10^2

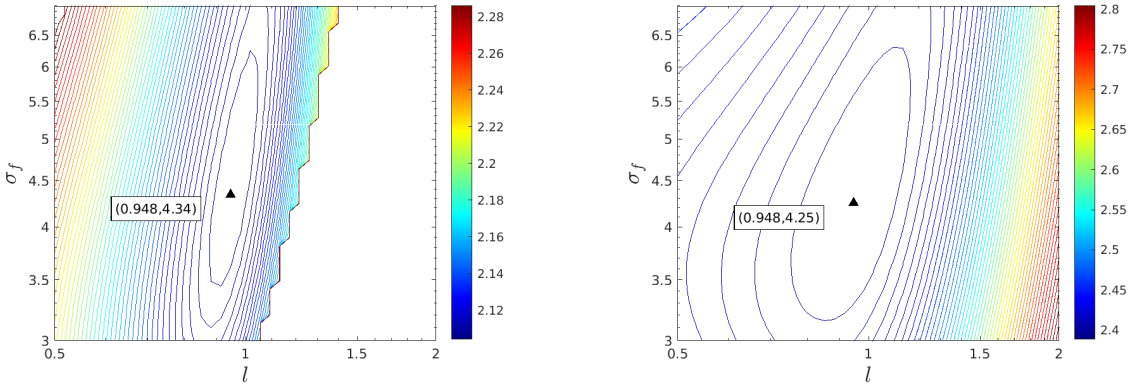
iterations, which is much less compared to the pxADMM$_{\text{fd}}$. The smaller number of iterations needed makes the method more acceptable than the pxADMM$_{\text{fd}}$ and ADMM$_{\text{fd}}$ in MAS, where the communication burden is rather high.

The fast convergence of pxADMM$^{*}_{\text{fd}}$ probably comes from the auxiliary variables introduced in Equation 3.22 and 3.23. As explained in Section 3.3.1, by maintaining auxiliary variables $\boldsymbol{\zeta}_m$ and $\boldsymbol{\Lambda}_m$, a DTCF process is equivalently applied that calculate the average of respectively $\mathbf{z}_{mn}$ and $\boldsymbol{\lambda}_{mn}$ variables, which can possibly pull the network further to consensus. Though the current results shows convergence, future works on mathematical analysis can be down to prove the convergence.

It can be noticed that, for all the methods examined, the signal variance σ_f converge to values around 4 instead of the real value 5. To the best of our knowledge, there is currently no analytical reason for this. Possible reason comes from the randomness of the datasets that the range of the generated data can not reflect the real variance.

To show that the aforementioned discrepancy between converged value and real value does not come from the error in simulation, the scaled NLML values under different setting of σ_f and l_1 are shown in Figure 3.3. The NLML shown in Figure 3.3b shows the NLML calculated by summation of local NLMLs, in which the optimal σ_f is close to the converged values of the examined methods.

The Figure 3.3a shows the scaled NLML values calculated based on the global dataset. The ranges of the examined σ_f and l_1 values are the same as those in Figure 3.3b, but the contours are obviously different. Also, the optimal σ_f value is different from that shown in Figure 3.3b.



(a) Based on NLML of global dataset

(b) Based on summation of NLML of local datasets

Figure 3.3: The figures shows the NLML for the given datasets. The NLML values are scaled for better visualization. The axes are in log scale. The black triangle shows the minimum points. (a) shows the scaled NLML directly calculated based on the entire global dataset, that is $l(\boldsymbol{\theta})$. The empty areas at the upper right and lower left corners are NLML values with infinite $log(det(\mathbf{K}))$ values, which is a phenomenon that happens when the dataset is large and the hyperparameters examined are far from the real values. (b) shows the scaled NLML calculated based on the summation of the NLML of local datasets, that is $\sum_{m=1}^{M} l_m(\boldsymbol{\theta})$.

Another experimental results on the σ_f is that the convergence is more stable with a smaller than real initial value than a large initial value. When the optimization of σ_f starts from large value, e.g., 10 when the real value is 5, the convergence results frequently change among many values. Also, the σ_f converges more slowly than the other hyperparameters. A possible reason for the above phenomenons is that the σ_f indicates the ability of GP for capturing the signal changes. When the σ_f is larger than enough, there is no strong 'force' that pulls the variance to optimal. The Figure 3.3a shows the scaled negative Log-likelihood of the given datasets. It can be found that the slope on the σ_f direction is small when σ_f is large, which results in a smaller gradients. This supports the experimental results that σ_f converges slower with smaller gradients. The results suggest a strategy that the optimization for σ_f should starts from a small value, which can be approximated before optimization based on a small subset of the datasets.

Asynchronous pxADMM$_{\text{async}}$ and pxADMM$^*_{\text{async}}$ The Figure A.7 shows the convergence results of asynchronous pxADMM$_{\text{async}}$. The asynchronous behavior can be obviously seen as shown in Figure A.7b, in which the step size curves vary among agents. The iteration numbers needed for convergence range from approximately 5×10^2 to 7×10^2.

The Figure 3.4 shows the comparison among the pxADMM methods, i.e., centralized pxADMM, fully-distributed pxADMM and asynchronous pxADMM, of agent 1. The curves for asynchronous methods ignore the inactivated iterations recorded. It can be found that the asynchronous algorithms converge in general same patterns as their synchronous versions at linear rates. Also, the activated iterations used are almost the same. The largest difference is that the convergence of asynchronous algorithms oscillate more than corresponding synchronous versions. In terms of number of iterations used, the pxADMM$^*_{\text{fd}}$ uses approximately 2 times iterations than pxADMM to reach convergence, while the pxADMM$_{\text{fd}}$ uses approximately 4.5 times more iterations.

The Figure 3.5 shows the comparison of all the methods mentioned above, in which all the axes are in logarithmic scale. It can be found that all the methods finally lead to a convergence. The centralized pxADMM, as shown in yellow solid line, converges the fastest. The GD converges slower than all the pxADMM based methods. The centralized ADMM converges in a much slower speed compared to GD or pxADMM based methods, because a large amount of iterations are used to optimize the problem in Equation 3.14b. The fully-distributed ADMM requires more iterations than centralized ADMM to reach convergence. The result shows that the proximal update strategy alleviates the computational complexity of ADMM.

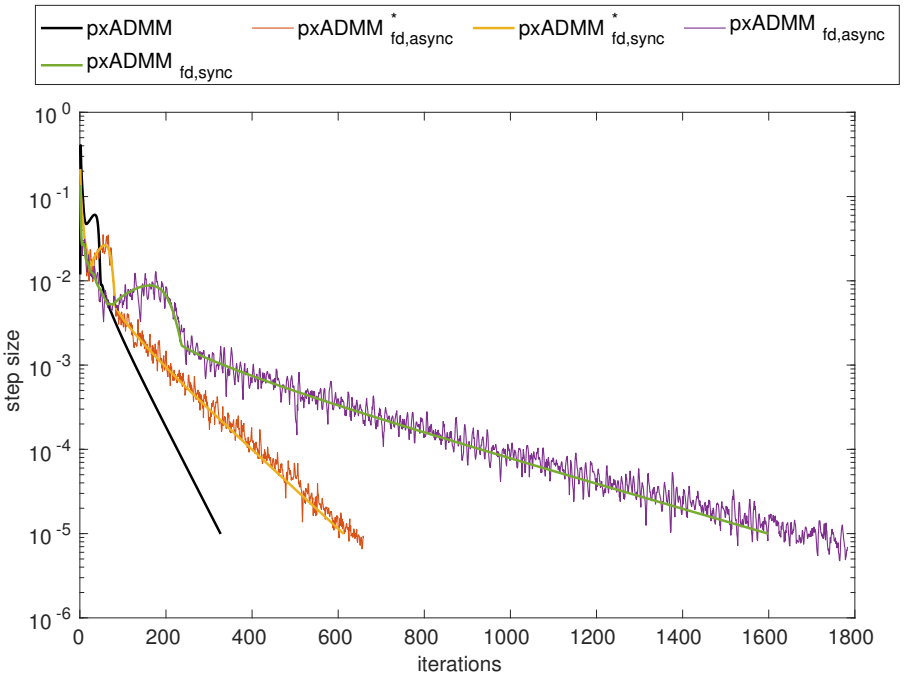


Figure 3.4: Comparison of different versions of pxADMM methods.

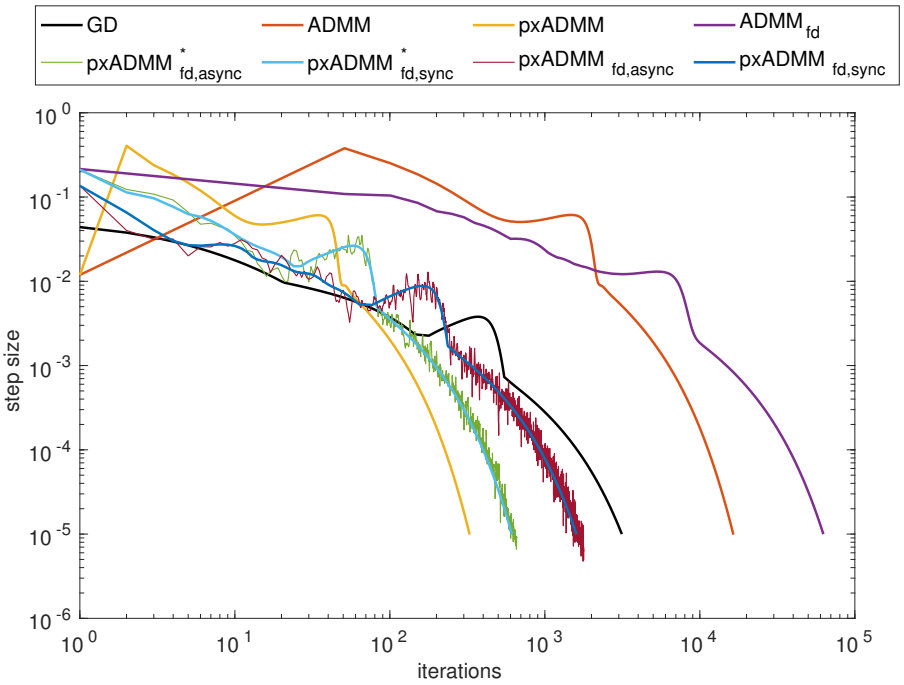


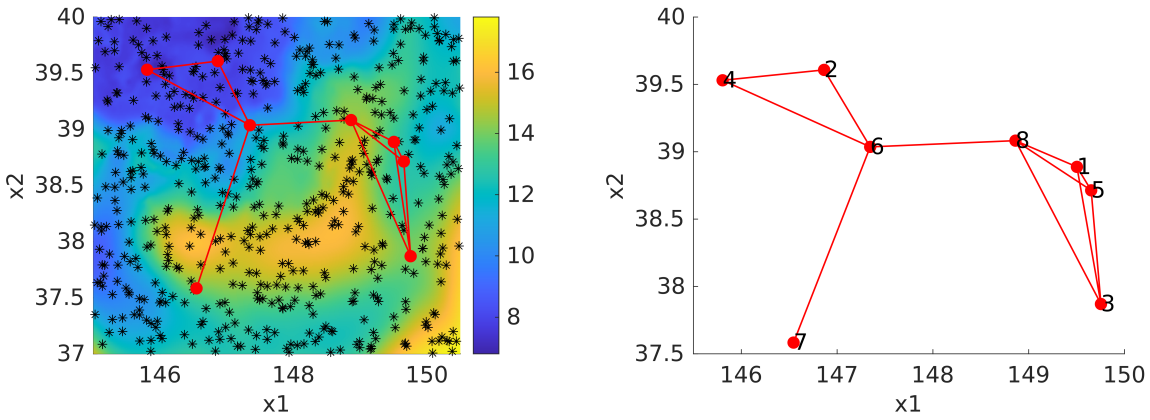
Figure 3.5: Step size comparison of hyperparameter optimization methods with both x and y axes in log scale. The curves of ADMM and ADMM_{fd} shows the step size regarding the inner iterations based on nGD.

3.4.2 Real dataset: GHRSST

The following simulations are carried out in the reconstructed 2D field from the GHRSST dataset. An example is selected for demonstration. The ranges of the field are respectively $[145.0, 150.5]$ and $[37.0, 40.0]$ on longitude and latitude directions (or referred as x and y axes). Since the real dataset used is not generated according to known GP, the real hyperparameter set is unknown. The performance of optimization methods will be analyzed through their convergence curves and comparisons with each other.

Table 3.3: Parameters & Variables of pxADMM for GHRSST

Parameters & variables	Values	Only used in
ρ	$8 \times 70 \times 0.3$	
L	$8 \times 70 \times 0.7$	
T_{max}	15000	
$\boldsymbol{\theta}_m^0 = \{\sigma_f^0, \mathbf{l}^0, \sigma_n^0\}$	$\{6, [2,2]^T, 0.5\}$	
ϵ	1×10^{-5}	
$\boldsymbol{\lambda}_m^0$	$[1,1,1,1]^T$	
$\mathbf{z}_m^0$	$\boldsymbol{\theta}_m^0$	pxADMM
$\mathbf{z}_{mn}^0, \mathbf{z}_{nm}^0, \boldsymbol{\lambda}_{mn}^0, \boldsymbol{\lambda}_{nm}^0$	$[0,0,0,0]^T$	$\text{pxADMM}_{\text{fd}}$, $\text{pxADMM}_{\text{fd}}^*$
S_m	$\min(3, \lvert\mathcal{N}(m)\rvert)$	$\text{pxADMM}_{\text{async}}$, $\text{pxADMM}_{\text{async}}^*$
τ	10	$\text{pxADMM}_{\text{async}}$, $\text{pxADMM}_{\text{async}}^*$



(a) Topology, sampling points and background

(b) Topology with agent codes

Figure 3.6: (a). The figure shows MAS and underlying GHRSST field. The field is shown as background with magnitudes indicated by color according to the colorbar. The red dots shows the positions of agents. The communication links are plotted as red lines connecting pairs of agents. The black asterisks scattered around the field shows the position of sampling points. (b). The figure shows the topology structure of MAS, in which the numbers indicate the code of agents from 1 to M, i.e., 8.

nGD, ADMM, ADMM$_{\text{fd}}$, pxADMM The convergence behavior of nGD, ADMM and ADMM$_{\text{fd}}$ are respectively shown in Figure A.9, A.10 and A.11 in Appendix A.1. Some of the optimization settings are listed in Table 3.3. For nGD, the step size is $\mu = 5 \times 10^{-5}$. For ADMM, the maximum iteration is $T_{max} = 2000$, and the maximum iteration number for solving local $\boldsymbol{\theta}_m$ minimization problem is $T_{max,inner} = 50$.

Based on comparison among the methods from the Table 3.4 and figures, it can be found that the nGD and ADMM$_{\text{fd}}$ methods have not converged after reaching the maximum iteration number.

Synchronous pxADMM$_{\text{fd}}$ and pxADMM$^*_{\text{fd}}$ The simulation results of centralized pxADMM, pxADMM$_{\text{fd}}$ and pxADMM$^*_{\text{fd}}$ are respectively shown in Figure A.12, A.13 and A.14 in Appendix A.1.

Asynchronous pxADMM$_{\text{async}}$ and pxADMM$^*_{\text{async}}$ Figure A.15 shows the convergence curves of pxADMM$_{\text{async}}$, in which the number of iterations range from approximately 5×10^3 to 5.5×10^3.

Figure A.16 shows the convergence curves of pxADMM$^*_{\text{async}}$. The number of iterations ranges from approximately 3×10^3 to 4×10^3.

Results of the optimized hyperparameters are shown in Table 3.4.

Table 3.4: Results of hyperparameters optimization with real dataset

Methods	σ_f	$\mathbf{l}$	σ_n
nGD	8.2247	[1.6009;1.2245]	0.7015
ADMM	8.4463	[1.6100;1.2290]	0.7005
ADMM$_{\text{fd}}$	7.8172	[1.5847;1.2167]	0.7038
pxADMM	8.4503	[1.6102;1.2291]	0.7005
pxADMM$_{\text{fd}}$	8.4619	[1.5731;1.2113]	0.6831
pxADMM$^*_{\text{fd}}$	8.4266	[1.6092;1.2286]	0.7006
pxADMM$_{\text{async}}$	8.4640	[1.5731;1.2114]	0.6831
pxADMM$^*_{\text{async}}$	8.3870	[0.6076,1.2277]	0.7007

3.5 Conclusion

In this chapter, GP hyperparameter optimization is motivated and introduced. The current centralized and distributed hyperparameter optimization methods are introduced. Then, new methods are proposed for optimization in MAS.

- Motivation: The GP prior model should be fitted to the training dataset for a good model that gives accurate prediction, thus requiring the hyperparameter optimization.
- Centralized optimization is performed based on Bayesian model selection methods by minimizing negative Log-likelihood. The problem can be solved by nGD.
- For problem $\mathcal{P}_2$, **ADMM** and **pxADMM** can be applied with assistance of center station.
- $\mathcal{P}_3$ can be solved by $\mathbf{ADMM_{fd}}$: slow convergence and high computational complexity.
- Proposed synchronized methods
  - $\mathbf{pxADMM_{fd}}$: Convergence proved, slower convergence than the centralized version
  - $\mathbf{pxADMM^{*}_{fd}}$: Convergence not mathematically proved, but experimental results show convergence. Faster convergence than $\mathbf{pxADMM_{fd}}$
- Asynchronous methods $\mathbf{pxADMM_{async}}$ and $\mathbf{pxADMM^{*}_{async}}$ based on two strategies
  - **Partial barrier** introduces asynchronous behavior by allowing fast agents entering new iterations with partial updates from neighbors.
  - **Bounded delay** restricts that a link is activated at least one time in certain number of iterations, which acts as soft synchronization over the network that assures similar convergence speed across the network.

4 Distributed GP Aggregation

With a well trained Gaussian Process prior model, predictions in the field of interest are to be made for further applications. The GP built on the full dataset is called full GP, which is accurate but computational intensive. In distributed systems, the dataset is divided into several smaller local datasets, which requires some modification to GP prediction process.

In this chapter, methods of aggregating different GP predictions from agents in the network are introduced. This chapter starts from introducing the background of distributed GP in Section 4.1. The GP aggregation problem is also defined in section 4.1, where some current methods are also introduced. Current works that try to distribute the aggregation process are introduced in Section 4.2. Some improved GP aggregation algorithms are proposed in section 4.3. Simulation results of several selected methods are compared and discussed in section 4.4.

4.1 GP aggregation

4.1.1 Background

Early efforts of separating a full GP or combining several GP were mainly motivated by two needs, less computational intensity and higher flexibility.

Though GP is already a flexible model compared to most parametric models, it sometimes fails to represent special structures with a single set of hyperparameters. A simple case can be unstable noise variance, which varies across the input space, thus making a single choice of kernel variance inadequate. At times, GP also fails to detect discontinuity, which requires kernel with short characteristic length to represent.

As explained in section 2.1.1, GP is a data dependent model with high computational complexity of $\mathcal{O}(N^3)$, which mainly comes from the matrix inversion in Equation (2.11). By distributing the datasets into M local datasets, the computation can be distributed parallelly with reduced complexity $\mathcal{O}(\frac{N^3}{M^2})$.

To solve the above problems, the Mixture of Experts (MoE) model is applied to Gaussian Process in [28]. In the proposed algorithm, there are multiple experts with different GP hyperparameters responsible for different parts of the function characteristics. A gating network assigns incoming data points to appropriate expert and makes final predictions based on prediction provided by experts. The system tracks different characteristics better than a single GP, and also speeds up the prediction because of the smaller datasets at the agents. Further improvement to this algorithm is proposed in [29], which brings more robustness against incomplete data by modifying the gating system, and [30], which further speeds up the training and inference through linear approximation. Another similar structure is proposed in [31], where the mean value prediction of GP is approximated by a matrix-vector multiplication (MVM) problem.

Then, the MVM can be accelerated through grouping the data points in a multiresolution tree structure, i.e., the KD-tree that recursively performs binary partition of dataset.

All algorithms introduced above divide the full datasets into smaller subsets to speed-up the computation and introduce more flexibility. Though these centralized algorithms are mainly developed to increase flexibility, the MoE structure has been shown to be a promising methods for distribution.

4.1.2 GP Aggregation problem

For a full GP with complete dataset $\mathcal{D}$, prediction for the value y_* at a point of interest $\mathbf{x}_*$ can be calculated by finding mean value in predictive distribution. Recall GP predictive distribution from Equation (2.11) as follow

$$p(y_*|\mathcal{D}) \sim \mathcal{N}\left(\mathbf{K}_*\mathbf{K}^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{K}_*\mathbf{K}^{-1}\mathbf{K}_*^T\right). \tag{4.1}$$

Also, recall from Section 2.2.2 that the predictor for posterior mean and variance and be respectively represented by $\mu(y_*|\mathbf{x}_*, \mathcal{D})$ and $\sigma^2(y_*|\mathbf{x}_*, \mathcal{D})$, or μ and σ^2 for simplicity.

The aggregation of GP can be interpreted from two perspective. The first view is combining predictors. In a distributed system described by graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, predictions are first independently given by each agent m from local predictors $\mu_m(y_*|\mathbf{x}_*, \mathcal{D}_m)$ and $\sigma_m^2(y_*|\mathbf{x}_*, \mathcal{D}_m)$ (or μ_m and σ_m^2 for simplicity), and then combined into global predictors $\mu_{\mathcal{G}}(y_*|\mathbf{x}_*, \{\mathcal{D}_m\}_{m=1}^M)$ and $\sigma_{\mathcal{G}}^2(y_*|\mathbf{x}_*, \{\mathcal{D}_m\}_{m=1}^M)$ (or $\mu_{\mathcal{G}}$ and $\sigma_{\mathcal{G}}^2$ for simplicity).

From a Bayesian modeling perspective, the full GP prediction corresponds to finding posterior distribution $p(y_*|\mathcal{D})$. GP aggregation is equivalent to finding a relationship that connects local posterior predictions $p_m(y_*|\mathcal{D}_m)$ with $p(y_*|\mathcal{D})$.

There are two different kinds of MoE methods developed based on assumption that whether the local datasets are independent or nested. Fig.4.1 shows an example of dividing full covariance matrix of a global dataset into 6 local covariance matrices, which are shown in colored boxes. In Fig. 4.1b, the areas outside the boxes record the cross-correlation among local datasets, which are unknown in the distributed setting. The independent assumption assumes that the unknown region are filled with zero values, which means that the local datasets are independent with each other. On the contrary, nested assumptions tries to discover information in the unknown area, so that the correlation among local datasets can also be take into consideration during prediction.

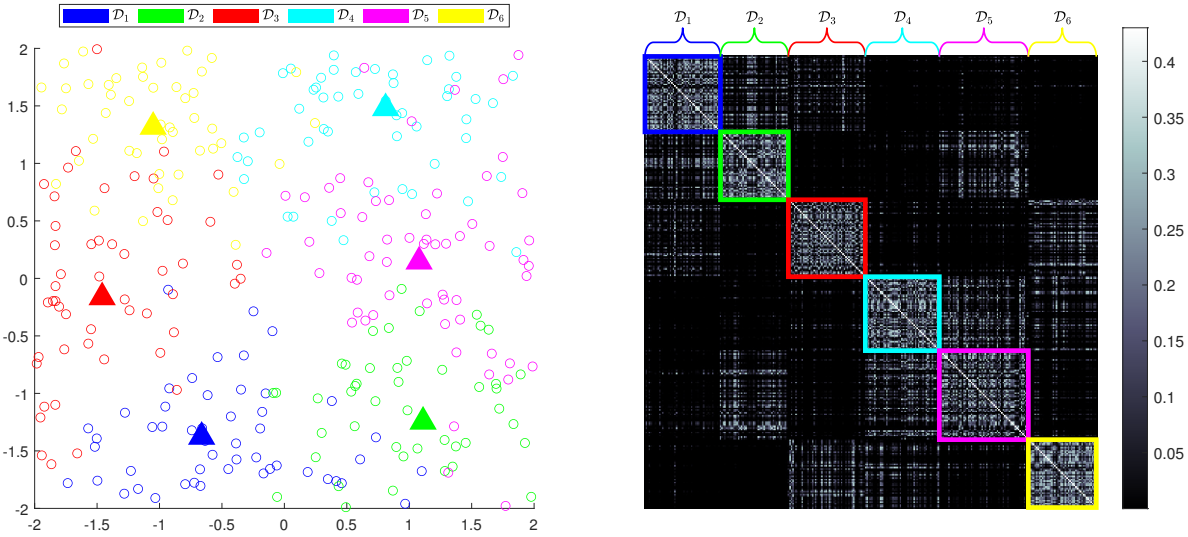
4.1.3 Independent aggregation

Given two datasets $\mathcal{D}_m$ and $\mathcal{D}_n$, Bayes' rule indicates that

$$p(y_*|\mathcal{D}_m, \mathcal{D}_n) = \frac{p(y_*)p(\mathcal{D}_m|y_*)p(\mathcal{D}_n|\mathcal{D}_m, y_*)}{p(\mathcal{D}_m, \mathcal{D}_n)}. \tag{4.2}$$

Under independence assumption [32], $\mathcal{D}_m$ and $\mathcal{D}_n$ are conditionally independent to each other such that

$$p(\mathcal{D}_n|\mathcal{D}_m, y_*) \approx p(\mathcal{D}_n|y_*), \tag{4.3}$$



(a) Local dataset partition

(b) Full and divided covariance matrix

Figure 4.1: (a) The plot shows a global dataset containing 300 sampling data points in circles. Different color represent different partitions of data points. The triangles are the position of agents that hold the local datasets. (b) The covariance matrix of the full datasets. The colored frame contains the corresponding local datasets at agents.

which can be noted as $\mathcal{D}_m \perp\!\!\!\perp \mathcal{D}_n$. Based on Equation (4.3), the posterior prediction with two joint dataset in Equation (4.2) in can be factorized as multiple independent predictions based on local datasets, so that

$$\begin{aligned} p(y_*|\mathcal{D}_m, \mathcal{D}_n) &\approx \frac{1}{p(\mathcal{D}_m, \mathcal{D}_n)} \cdot p(y_*)p(\mathcal{D}_m|y_*)p(\mathcal{D}_n|y_*) \\ &= \frac{p(\mathcal{D}_m), p(\mathcal{D}_n)}{p(\mathcal{D}_m, \mathcal{D}_n)} \cdot \frac{p(y_*|\mathcal{D}_m)p(y_*|\mathcal{D}_n)}{p(y_*)} \\ &\propto \frac{p(y_*|\mathcal{D}_m)p(y_*|\mathcal{D}_n)}{p(y_*)}. \end{aligned} \tag{4.4}$$

By recursively applying conditional independence to all the local datasets, it can be shown that

$$p(y_*|\mathcal{D}) \propto \frac{\prod_{m=1}^{M} p(y_*|\mathcal{D}_m)}{p^{M-1}(y_*)}, \tag{4.5}$$

where local posterior distributions are calculated according to Equation (2.11), and the mean and standard deviation are noted as μ_m and σ_m for agent m.

PoE family Product of Experts (PoE) was first proposed in [33] to combine multiple latent-variable models, and applied to GP aggregation in [34].

For PoE, Equation (4.5) is applied without considering prior distribution $p(y_*)$,

which is equivalent to the assumption that

$$p(y_*|\mathcal{D}) = \prod_{m=1}^{M} p(y_*|\mathcal{D}_m). \tag{4.6}$$

Because each of the local posterior distributions is a Gaussian distribution, the mean and variance values of the aggregated posterior distribution at input $\mathbf{x}_*$ are given by

$$\mu_{PoE} = \sigma_{PoE}^2 \sum_{m=1}^{M} \sigma_m^{-2} \mu_m, \tag{4.7a}$$

$$\sigma_{PoE}^2 = \frac{1}{\sum_{m=1}^{M} \sigma_m^{-2}} \tag{4.7b}$$

Intuitively, the PoE combines the aggregations of agents based on their confidence about their own predictions. If an agent is uncertain about its prediction, a large posterior variance makes the weight of its predicted mean values smaller during aggregation, which finally makes the global predictor trust more on agents with high confidence levels.

Since PoE does not consider prior distribution, the variance predictions are not consistent with prior assumption. For PoE, when all agents give uncertain predictions such that $\sigma_m = \sigma_{**}$, the global posterior variance is $\frac{1}{M}\sigma_{**}^2 < \sigma_{**}^2$, which indicates an over-confident prediction. The over-confidence problem of PoE indicates that the predictive variances are not always reliable as weights for aggregation.

To alleviate the problem of PoE, the generalized Product of Expert (gPoE) proposed in [35] further introduces another set of variables $\{\alpha_m \in \mathbb{R}_+\}_{m=1}^{M}$ controlling the influence of experts, which can also be referred as expert importance. The relationship between full posterior and local posteriors can be easily derived as

$$p(y_*|\mathcal{D}) = \prod_{m=1}^{M} p^{\alpha_m}\left(y_*|\mathcal{D}_m\right), \tag{4.8}$$

where the α_m introduces more freedom when weighting predictions from agents. Under gPoE model, the posterior mean and variance values are given by

$$\mu_{gPoE} = \sigma_{gPoE}^2 \sum_{m=1}^{M} \alpha_m \sigma_m^{-2} \mu_m. \tag{4.9a}$$

$$\sigma_{gPoE}^2 = \frac{1}{\sum_{m=1}^{M} \alpha_m \sigma_m^{-2}} \tag{4.9b}$$

For gPoE, choosing non-negative α_m such that $\sum_{m=1}^{M} \alpha_m = 1$ assures that the posterior variance falls back to prior σ_{**}^2 in highly uncertain areas, but it also results in too conservative prediction in areas with data points.

Suppose that the local posterior mean and variance values are known before aggregation, the PoE and gPoE methods have computational complexity of $\mathcal{O}(M)$

Table 4.1: Independent aggregation methods

Method	α_m	σ_{**}^{-2}
PoE	1	0
gPoE	$\sum_{m=1}^{M} \alpha_m = 1, \alpha_m > 0$	0
BCM	1	$\sigma_f^2 + \sigma_n^2$
rBCM	$\frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_m^2))$ [36]	$\sigma_f^2 + \sigma_n^2$

BCM family The Bayesian Committee Machine (BCM) [32] considers prior distribution in Equation (4.5). The posterior mean and variance values are then derived as

$$\mu_{BCM} = \sigma_{BCM}^2 \sum_{m=1}^{M} \sigma_m^{-2} \mu_m, \tag{4.10a}$$

$$\sigma_{BCM}^2 = \frac{1}{\sum_{m=1}^{M} \sigma_m^{-2} - (M-1)\sigma_{**}^{-2}} \tag{4.10b}$$

When GP and measurement noise are assumed to be stationary across the points, the σ_{**}^2 is usually given by $\sigma_f^2 + \sigma_n^2$. By introducing the prior information, the aggregated posterior variance falls back to prior variance σ_{**}^2 when all agents are uncertain about their prediction with $\sigma_m = \sigma_{**}$, thus avoids the over-confidence problem of the PoE.

Similar as how gPoE is developed, the robust Bayesian Committee Machine (rBCM) introduces a set of influence weights $\{\alpha_m \in \mathbb{R}_+\}_{m=1}^M$ to further adjust the influence of agents. The Bayesian model in Equation (4.4) becomes

$$\begin{aligned} p(y_*|\mathcal{D}) &\propto p(y_*) \prod_{m=1}^{M} p^{\alpha_m}(\mathcal{D}_m|f_*) \\ &= \frac{\prod_{m=1}^{M} p^{\alpha_m}(f_*|\mathcal{D}_m)}{f_*^{-1+\sum_{m=1}^{M}\alpha_m}}. \end{aligned} \tag{4.11}$$

The posterior mean and variance are then given by

$$\mu_{rBCM} = \sigma_{rBCM}^2 \sum_{m=1}^{M} \alpha_m \sigma_m^{-2} \mu_m, \tag{4.12a}$$

$$\sigma_{rBCM}^2 = \frac{1}{\sum_{m=1}^{M} \alpha_m \sigma_m^{-2} - \left(\sum_{m=1}^{M} \alpha_m - 1\right) \sigma_{**}^{-2}} \tag{4.12b}$$

where $\alpha_m > 0$ but not necessarily $\sum_{m=1}^{M} \alpha_m = 1$. The $\left(\sum_{m=1}^{M} \alpha_m - 1\right) \sigma_{**}^{-2}$ acts as a correction term that automatically pulls the posterior variance back to the prior. By comparing the aggregators, it can be found that the PoE, gPoE and BCM aggregators can all be generated by adjusting the parameters from rBCM aggregators. Table 4.1 shows how rBCM can be degenerated into PoE, gPoE and BCM.

4.1.4 Nested aggregation

Though independent MoE models alleviate the computation intensity, they are based on the assumption that the datasets are (conditionally) independent, which is usually not the case. As shown in Fig. 4.1, correlation between local datasets should also be considered for better approximation and consistency to full GP. Several works are proposed under nested datasets assumption, including the generalized robust BCM (grBCM) proposed by Liu et al. [37] and Nested Pointwise Aggregation of Experts (NPAE) by Rullière et al. [38]. These methods are introduced in this section.

Generalized robust BCM The grBCM shares similar aggregation formula as those from PoE and BCM families, but make changes in the structure of network and content of datasets for better aggregation. Based on the local datasets $\{\mathcal{D}_1, \cdots, \mathcal{D}_M\}$, a global expert dataset $\mathcal{D}_c$ is generated through randomly extracting points from local datasets, which makes the $\mathcal{D}_c$ contain points spreading throughout the entire region of interest. The predictive distribution of y_* based on $\mathcal{D}_c$ is given by $p(y_*|\mathcal{D}_c) \sim \mathcal{N}(\mu_c, \sigma_c^2)$. The agent (or central computing unit) maintaining the $\mathcal{D}_c$ can communicate directly with all the rest agents, which makes the $\mathcal{D}_c$ accessible by other agents. For each agent m, an enhanced dataset $\mathcal{D}_{+m} = \{\mathcal{D}_m, \mathcal{D}_c\}$ is constructed, based on which the enhanced predictive distribution $p_{+m}(y_*|\mathcal{D}_{+m}) \sim \mathcal{N}(\mu_{+m}, \sigma_{+m}^2)$ can be derived. With further proposing the conditional independence $\mathcal{D}_m \perp\!\!\!\perp \mathcal{D}_n|\mathcal{D}_c$, the predictive distribution of grBCM is proposed as

$$p(y_*|\mathcal{D}) = \frac{\prod_{m=1}^{M} p_{+m}^{\alpha_m}(y_*|\mathcal{D}_{+m})}{p^{-1+\sum_{m=1}^{M}\alpha_m}(y_*|\mathcal{D}_c)}, \tag{4.13}$$

where the $\alpha_m = \frac{1}{2}\left(\log \sigma_c^2 - \log \sigma_{+m}^2\right)$.

The grBCM provides consistent aggregations and falls back to the prior model at the costs of a slight increase of computational complexity. The drawback of applying and improving grBCM in this work is that grBCM requires a network structure with central computing unit, which is usually not applicable for fully-distributed MAS. Thus, the grBCM is not included in the simulation and comparison of aggregation methods.

NPAE For a point of interest $\mathbf{x}_*$, the NPAE considers a linear aggregation of local predictions in the form of $\mu_{NPAE} = \sum_{m=1}^{M} \omega_m \mu_m$, where $\{\omega_m \in \mathbb{R}\}_{m=1}^{M}$ weight the predictions from agents. The exact form of ω can be derived by finding the Best Linear Unbiased Predictor (BLUP). Let the vector collecting all local predictions be noted as

$\mathcal{M} = [\mu_1, \cdots, \mu_M]^T$, and the weight vector $[\omega_1, \cdots, \omega_M]^T$ as $\boldsymbol{\omega}$, the BLUP is derived by minimizing the mean squared error $e_{\mathcal{M}} = E\left\{(y_* - \boldsymbol{\omega}^T\mathcal{M})^2\right\}$, which gives the optimal weight vector $\hat{\omega} = \mathbf{K}_{\mathcal{M}}^{-1}\mathbf{k}_{\mathcal{M}}$. The $\mathbf{k}_{\mathcal{M}}$ is the covariance between local predictions and value to be predicted, which is given by

$$\begin{aligned}\mathbf{k}_{\mathcal{M}} &= Cov(\mathcal{M}, y_*) \\ &= Cov([\mu_1, \mu_2, \cdots, \mu_M]^T, y_*) \\ &= Cov\left(\left[\mathbf{K}_{*,1}\mathbf{K}_1^{-1}\mathbf{y}_1, \mathbf{K}_{*,2}\mathbf{K}_2^{-1}\mathbf{y}_2, \cdots, \mathbf{K}_{*,M}\mathbf{K}_M^{-1}\mathbf{y}_M\right]^T, y_*\right) \\ &= \left[\mathbf{K}_{*,1}\mathbf{K}_1^{-1}\mathbf{K}_{*,1}^T, \mathbf{K}_{*,2}\mathbf{K}_2^{-1}\mathbf{K}_{*,2}^T, \cdots, \mathbf{K}_{*,M}\mathbf{K}_M^{-1}\mathbf{K}_{*,M}^T\right]^T,\end{aligned} \tag{4.14}$$

where $\mathbf{K}_{*,m} = Cov(y_*, \mathbf{y}_m)$. It should be noticed that the mth entry of $\mathbf{k}_{\mathcal{M}}$ can be locally computed by agent m.

Similarly, the $\mathbf{K}_{\mathcal{M}} = Cov(\mathcal{M}, \mathcal{M})$ is given by

$$\begin{bmatrix} \mathbf{K}_{*,1}\mathbf{K}_1^{-1}\mathbf{K}_{1,1}\mathbf{K}_1^{-T}\mathbf{K}_{*,1}^T & \mathbf{K}_{*,1}\mathbf{K}_1^{-1}\mathbf{K}_{1,2}\mathbf{K}_2^{-T}\mathbf{K}_{*,2}^T & \cdots & \mathbf{K}_{*,1}\mathbf{K}_1^{-1}\mathbf{K}_{1,M}\mathbf{K}_M^{-T}\mathbf{K}_{*,M}^T \\ \mathbf{K}_{*,2}\mathbf{K}_2^{-1}\mathbf{K}_{2,1}\mathbf{K}_1^{-T}\mathbf{K}_{*,1}^T & \mathbf{K}_{*,2}\mathbf{K}_2^{-1}\mathbf{K}_{2,2}\mathbf{K}_2^{-T}\mathbf{K}_{*,2}^T & \cdots & \mathbf{K}_{*,2}\mathbf{K}_2^{-1}\mathbf{K}_{2,M}\mathbf{K}_M^{-T}\mathbf{K}_{*,M}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}_{*,M}\mathbf{K}_M^{-1}\mathbf{K}_{M,1}\mathbf{K}_1^{-T}\mathbf{K}_{*,1}^T & \mathbf{K}_{*,M}\mathbf{K}_M^{-1}\mathbf{K}_{M,2}\mathbf{K}_2^{-T}\mathbf{K}_{*,2}^T & \cdots & \mathbf{K}_{*,M}\mathbf{K}_M^{-1}\mathbf{K}_{M,M}\mathbf{K}_M^{-T}\mathbf{K}_{*,M}^T \end{bmatrix}, \tag{4.15}$$

where $\{\mathbf{K}_{m,n} = Cov(\mathbf{y}_m, \mathbf{y}_n)\}_{m,n\in\{1,2,\cdots,M\}}$ are the cross covariance between datasets m and n. The size of vector $\mathbf{k}_{\mathcal{M}}$ is $M \times 1$, and size of matrix $\mathbf{K}_{\mathcal{M}}$ is $M \times M$.

Substituting Equation (4.14) and (4.15) into linear predictor and error $e_{\mathcal{M}}$, the pointwise global predictor proposed by NPAE is then given by

$$\mu_{NPAE} = \mathbf{k}_{\mathcal{M}}^T\mathbf{K}_{\mathcal{M}}^{-1}\mathcal{M}, \tag{4.16a}$$

$$\sigma_{NPAE}^2 = \mathbf{K}_{**} - \mathbf{k}_{\mathcal{M}}^T\mathbf{K}_{\mathcal{M}}^{-1}\mathbf{k}_{\mathcal{M}}. \tag{4.16b}$$

It is shown in [39] that the NPAE is an asymptotically consistent method such that error $e_{\mathcal{M}}$ converge to 0 as the number of observations increase to infinity. Although NPAE provides more consistent aggregation, the improvement comes at the cost of higher computational complexity due to calculating cross covariance matrices and inverse of $\mathbf{K}_{\mathcal{M}}$.

4.1.5 Other aggregation methods

Another distributed GP for spatial-temporal environmental monitoring is studied in [40]. Sparsity is introduced to reduce the computation cost. The predictive distribution is sparsely approximated with projected process approximation. Covariance matrix is also approximated with a specially chosen kernel function. For the distributed prediction, a node only need to receive data and position information from its neighbors in communication range.

A sparsely approximated GP by Karhunen-Loève expansion is distributed in [41]. The proposed method considers a network consisting of agents with limited communication range. Agents measure their own environmental data, and collaboratively

maintain a global GP model. To reduce computation cost, Karhunen-Loève expansion is applied to factorize the kernel into weighted sum of orthogonal eigenfunctions, among which only a few most important ones are kept, thus reducing the dimension of data space. Then a modified prediction stage is applied with the new kernel. The measured data for prediction are not obviously exchanged between agents, but implicitly spread across the network through consensus algorithms on latent variables.

Due to time constraints, further simulation and improvement of methods based on sparsity, subspace, etc., are not included in this work.

4.2 Distributed GP Aggregation

The methods aforementioned provide parallelable aggregation methods based on predictions with local datasets, but all require centralized computing structure, which is not favored in many distributed multi-agent systems. In this section, the state-of-the-art distributed methods are introduced and discussed.

4.2.1 Fully-distributed PoE and BCM families

The PoE and BCM families of methods can be fully distributed in a network through consensus. Since all the methods can be obtained through adjusting the rBCM, the distributed algorithm is explained based on the rBCM. The posterior prediction from Equation (4.12) can be modified as the following

$$\sigma^2_{rBCM} = \frac{1}{\sigma_{**}^{-2} + \sum_{m=1}^{M} \gamma_{\sigma,m}} \tag{4.17a}$$

$$\mu_{rBCM} = \sigma^2_{rBCM} \sum_{m=1}^{M} \gamma_{\mu,m}, \tag{4.17b}$$

where $\gamma_{\sigma,m} = \alpha_m(\sigma_m^{-2} - \sigma_{**}^{-2})$ and $\gamma_{\mu,m} = \alpha_m \sigma_m^{-2} \mu_m$. The $\gamma_{\sigma,m}$ and $\gamma_{\mu,m}$ are totally local variables that can be computed independently by each agent, so that the global mean and variance can be obtained based on the consensus values of $\bar{\gamma}_{\sigma,m} = \frac{1}{M} \sum_{m=1}^{M} \gamma_{\sigma,m}$ and $\bar{\gamma}_{\mu,m} = \frac{1}{M} \sum_{m=1}^{M} \gamma_{\mu,m}$. The consensus average problem is formulated as follow

$$\begin{aligned} \mathcal{P}_4 : \min_{\{\gamma_m\}_{m=1}^M} \quad & \sum_{m=1}^{M} \left\| \gamma_m - \gamma_m^0 \right\|_2^2 \\ s.t. \quad & \gamma_m = \gamma_n, \quad \forall (m,n) \in E \end{aligned} \tag{4.18}$$

The discrete-time consensus filter (DTCF) [42] is currently applied to solve $\mathcal{P}_4$ [43]. For agent m, the consensus iteration at time instance t is given by

$$\gamma_m^{t+1} = \gamma_m^t + \nu \sum_{n \in \mathcal{N}(m)} \left(\gamma_n^t - \gamma_m^t \right), \tag{4.19}$$

where $\nu \in (0, \frac{1}{\Delta})$ and Δ is the largest degree in the network. The DTCF follows a linear convergence rate that is determined by the $1 - \nu\lambda_2$ [42]. The DTCF based fully distributed rBCM algorithm is denoted by DTCF-rBCM as shown in Algorithm 6. Similarly, the other methods in PoE and BCM families can be denoted as DTCF-PoE, DTCF-gPoE and DTCF-BCM.

Algorithm 6: Fully distributed DTCF based rBCM algorithm

Input: $\mathbf{x}_*$, σ_{**}, $\{\boldsymbol{\theta}_m\}_{m=1}^M$, $\{\mathbf{X}_m, \mathbf{y}_m\}_{m=1}^M$
Output: μ_{rBCM}, σ^2_{rBCM}
Initialize: T_{max}, $\nu \in (0, \frac{1}{\Delta})$
for *agent* $m = \{1, 2, \cdots, M\}$ **do**
  $[\mu_m, \sigma_m^2] = GPR(\mathbf{x}_*, \mathbf{X}_m, \mathbf{y}_m, \boldsymbol{\theta}_m)$ /*Local GP Prediction*/
  generate α_m
  $\gamma^0_{\mu,m} = \alpha_m \sigma_m^{-2} \mu_m$ /*Initialize $\gamma_{\mu,m}$*/
  $\gamma^0_{\sigma,m} = \alpha_m(\sigma_m^{-2} - \sigma_{**}^2)$ /*Initialize $\gamma_{\sigma,m}$*/
end
for *iteration* $t = \{0, 1, \cdots, T_{max}\}$ **do**
  for *agent* $m = \{1, 2, \cdots, M\}$ **do**
    Collects $\gamma^t_{\mu,n}$ and $\gamma^t_{\sigma,n}$, $n \in \mathcal{N}(m)$
    $\gamma^{t+1}_{\mu,m} = \gamma^t_{\mu,m} + \nu \sum_{n \in \mathcal{N}(m)} \left(\gamma^t_{\mu,n} - \gamma^t_{\mu,m}\right)$ /*Consensus $\gamma_{\mu,m}$ update*/
    $\gamma^{t+1}_{\sigma,m} = \gamma^t_{\sigma,m} + \nu \sum_{n \in \mathcal{N}(m)} \left(\gamma^t_{\sigma,n} - \gamma^t_{\mu,m}\right)$ /*Consensus $\gamma_{\sigma,m}$ update*/
  end
end
$\sigma^2_{rBCM} = 1/(\sigma_{**}^{-2} + M\gamma^{T_{max}+1}_{\sigma,m})$
$\mu_{rBCM} = \sigma^2_{rBCM,m} M \gamma^{T_{max}+1}_{\mu,m}$ /*Result can be obtained from any agents*/

4.2.2 Distributed NPAE

The distribution of NPAE is achieved through solving linear algebraic problems by distributed algorithms. In Equation (4.16), $\mathbf{K}_{\mathcal{M}}^{-1}\mathcal{M}$ and $\mathbf{K}_{\mathcal{M}}^{-1}\mathbf{k}_{\mathcal{M}}$ can respectively be regarded as the solution to linear algebraic problems

$$\mathbf{K}_{\mathcal{M}}\mathbf{q}_{\mu} = \mathcal{M}, \tag{4.20a}$$
$$\mathbf{K}_{\mathcal{M}}\mathbf{q}_{\sigma^2} = \mathbf{k}_{\mathcal{M}}, \tag{4.20b}$$

where $\mathbf{q}_{\mu} \in \mathbb{R}^M$ and $\mathbf{q}_{\sigma^2}$ are unknown. If the problems can be solved across the network such that each agent m holds the result $\mathbf{q}_{\mu,[m]}$ and $\mathbf{q}_{\sigma^2,[m]}$, then the NPAE predictions can be obtained by calculating $\mathbf{k}_{\mathcal{M}}^T\mathbf{q}_{\mu}$ and $\mathbf{k}_{\mathcal{M}}^T\mathbf{q}_{\sigma^2}$ in distributed manner.

Two methods based on different network structures are proposed to fully distribute NPAE in [43]. The first method NPAE-JOR, based on Jacobi over relaxation (JOR) [44, Ch. 2.4] algorithm, is proposed for complete network, in which each agent can directly communicate with any other agents. It is also assumed that each agent m has known

$\mathbf{K}_{\mathcal{M},[m,]}$, $\mathbf{k}_{\mathcal{M},[m]}$ and μ_m. For problem $\mathbf{Aq} = \mathbf{b}$, the JOR iteration at time instance t is given by

$$\mathbf{q}_{[m]}^{t+1} = (1-\nu)\mathbf{q}_{[m]}^{t} + \frac{\nu}{\mathbf{A}_{[m,m]}} \left(\mathbf{b}_{[m]} - \sum_{n \in \mathcal{N}(m)} \mathbf{A}_{[m,n]} \mathbf{q}_{[n]}^{t} \right), \tag{4.21}$$

where $\nu \in \left(0, \frac{2}{M}\right)$. By applying JOR, $\mathbf{q}_{\mu,[m]}$ and $\mathbf{q}_{\sigma^2,[m]}$ are known for agent m, which means $\mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\mu,[m]}$ and $\mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\sigma^2,[m]}$ can be calculated. The average values of $\mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\mu,[m]}$ and $\mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\sigma^2,[m]}$ can be obtained through consensus average algorithm as $\bar{\gamma}_{JOR,\mu}$ and $\bar{\gamma}_{JOR,\sigma^2}$. Since $\mathbf{k}_{\mathcal{M}}^T\mathbf{q}_{\mu} = \sum_{m=1}^{M} \mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\mu,[m]}$ and $\mathbf{k}_{\mathcal{M}}^T\mathbf{q}_{\sigma^2} = \sum_{m=1}^{M} \mathbf{k}_{\mathcal{M},[m]}\mathbf{q}_{\sigma^2,[m]}$, the NPAE prediction is finally given by

$$\mu_{NPAE} = M\bar{\gamma}_{JOR,\mu}, \tag{4.22a}$$

$$\sigma^2_{NPAE} = \mathbf{K}_{**} - M\bar{\gamma}_{JOR,\sigma^2}. \tag{4.22b}$$

The second method NPAE-DALE is proposed for connected but not necessarily complete networks. The method proposes similar setting as NPAE-JOR, but applies a different algorithm called Distributed Algorithm for solving Linear Equations (DALE) [45] to solve the linear algebraic problems. For problem $\mathbf{Aq} = \mathbf{b}$, the DALE iteration at time instance t is given by

$$\mathbf{q}_{[m]}^{t+1} = \mathbf{P}_1\mathbf{b}_{[m]} + \frac{1}{|\mathcal{N}(m)|}\mathbf{P}_2 \sum_{n \in \mathcal{N}(m)} \mathbf{q}_{[n]}^{t}, \tag{4.23}$$

where $\mathbf{P}_1 = \mathbf{A}_{[m,]}^T \left(\mathbf{A}_{[m,]}\mathbf{A}_{[m,]}^T\right)^{-1}$ and $\mathbf{P}_2 = \mathbf{I}_M - \mathbf{A}_{[m,]}^T \left(\mathbf{A}_{[m,]}\mathbf{A}_{[m,]}^T\right)^{-1} \mathbf{A}_{[m,]}$. After knowing $\mathbf{q}_{\mu}$ and $\mathbf{q}_{\sigma^2}$, the $\bar{\gamma}_{DALE,\mu}$ and $\bar{\gamma}_{DALE,\sigma^2}$ can be obtained through consensus algorithm, which are then substituted into Equation (4.22) for NPAE prediction.

The NPAE-JOR requires a complete network, which is usually not the case in MAS. Though NPAE-DALE alleviates the requirement to a connected network, the network still needs to be initialized through flooding necessary data points across the entire network so that the $\mathbf{K}_{\mathcal{M},[m,]}$ becomes locally available, which is costly in term of bandwidth, time and memory.

4.3 Proposed distributed aggregation methods

In this section, two groups of algorithms are proposed. The first group of algorithms are proposed to achieve faster convergence for fully-distributed independent aggregation methods, i.e., PoE and BCM families. The second group of methods are proposed to extend NPAE to fully-distributed situations.

4.3.1 PDMM-PoE/BCM

As shown in section 4.2.1, the distributed PoE and BCM families can be realized through solving consensus problem $\mathcal{P}_4$. It should be noticed that the target function

of $\mathcal{P}_4$ is closed, proper and convex, for which a Primal-Dual Method of Multipliers (PDMM) can be applied for fast linear convergence rate of $\mathcal{O}(\frac{1}{t})$ [46].

By applying PDMM, the tth update iteration for consensus average problem at agent m is given by

$$\gamma_m^{t+1} = \frac{1}{1+cd_m}\left(\gamma_m^0 + \sum_{n\in\mathcal{N}(m)}\left(c\gamma_n^t + \lambda_{n|m}^t\right)\right), \tag{4.24a}$$

$$\lambda_{m|n}^{t+1} = -\lambda_{n|m}^t + c\left(\gamma_m^{t+1} - \gamma_n^t\right), \tag{4.24b}$$

where $\lambda_{m|n}$ and $\lambda_{n|m}$ are dual variables associated with edge (m,n) respectively stored in agent m and n. $c>0$ is a penalty parameter from the Lagrangian, and is related to the convergence rate. The c is usually a value smaller than 1 for consensus average, and an optimal choice is 0.303 [47]. γ_m^0 are the initial values. For each iteration on each link, the standard PDMM requires transmission of four values [1], which is more than DTCF that only transmit two values, γ from both agents. However, by introducing auxiliary variables $\xi_{m|n} = c\gamma_m + \lambda_{m|n}$ and $\xi_{n|m} = c\gamma_n + \lambda_{n|m}$ on agent m and n respectively, only two auxiliary variables need to be transmitted. With the auxiliary variable, Equation 4.24b can be simplified as

$$\begin{aligned}\lambda_{m|n}^{t+1} + c\gamma_m^{t+1} &= -\left(\lambda_{n|m}^{t+1} + c\gamma_n^{t+1}\right) + 2c\gamma_m^{t+1}\\ \xi_{m|n}^{t+1} &= -\xi_{n|m}^t + 2c\gamma_m^{t+1}.\end{aligned} \tag{4.25}$$

The update iteration is now

$$\gamma_m^{t+1} = \frac{1}{1+cd_m}\left(\gamma_m^0 + \sum_{n\in\mathcal{N}(m)}\xi_{n|m}^t\right), \tag{4.26a}$$

$$\xi_{m|n}^{t+1} = -\xi_{n|m}^t + 2c\gamma_m^{t+1}, \tag{4.26b}$$

where $\xi_{m|n}$ can be initialized by zero. By replacing the DTCF in fully-distributed PoE and BCM with PDMM, the proposed methods are denoted as PDMM-PoE(gPoE) and PDMM-BCM(rBCM). The fully distributed PDMM based rBCM is shown in Algorithm 7.

4.3.2 LOC-NPAE and CON-NPAE

According to Equation 4.16, the $\mathbf{k}_\mathcal{M}$ and $\mathbf{K}_\mathcal{M}$ have to be constructed based on all the datasets from the agents, which must be flooded through the network. In this section, the Local NPAE (LOC-NPAE) method is proposed to avoid flooding in the entire network. LOC-NPAE alleviates the requirement for complete network through abandoning certain amount of cross-correlation information among local datasets. From Fig. 4.1, it can be observed that the cross-correlation matrix is generally darker for those pairs of agents with larger distance between them, which means that the corresponding

[1] $\{\gamma_m, \lambda_m\}$ and $\{\gamma_n, \lambda_n\}$ from both agents on the sides of edge (m,n).

Algorithm 7: Fully distributed PDMM based rBCM algorithm

Input: $\mathbf{x}_*$, σ_{**}, $\{\boldsymbol{\theta}_m\}_{m=1}^M$, $\{\mathbf{X}_m, \mathbf{y}_m\}_{m=1}^M$, $\{d_m\}_{m=1}^M$

Output: μ_{rBCM}, σ^2_{rBCM}

Initialize: T_{max}, $c \in (0,1)$, $\{\xi^0_{\mu,m|n}\}_{m\in\mathcal{V},n\in\mathcal{N}(m)}$=0

for *agent* $m = \{1, 2, \cdots, M\}$ **do**

  $[\mu_m, \sigma_m^2] = GPR(\mathbf{x}_*, \mathbf{X}_m, \mathbf{y}_m, \boldsymbol{\theta}_m)$ /*Local GP Prediction*/

  generate α_m

  $\gamma^0_{\mu,m} = \alpha_m \sigma_m^{-2} \mu_m$ /*Initialize $\gamma_{\mu,m}$*/

  $\gamma^0_{\sigma,m} = \alpha_m(\sigma_m^{-2} - \sigma_{**}^2)$ /*Initialize $\gamma_{\sigma,m}$*/

end

for *iteration* $t = \{0, 1, \cdots, T_{max}\}$ **do**

  for *agent* $m = \{1, 2, \cdots, M\}$ **do**

    Collects $\xi^t_{\mu,n|m}$ and $\xi^t_{\sigma,n|m}$, $n \in \mathcal{N}(m)$

    $\gamma^{t+1}_{\mu,m} = \frac{1}{1+cd_m}\left(\gamma^0_{\mu,m} + \sum_{n\in\mathcal{N}(m)} \xi^t_{\mu,n|m}\right)$

    $\xi^{t+1}_{\mu,m|n} = -\xi^t_{\mu,n|m} + 2c\gamma^{t+1}_{\mu,m}$,

    $\gamma^{t+1}_{\sigma,m} = \frac{1}{1+cd_m}\left(\gamma^0_{\sigma,m} + \sum_{n\in\mathcal{N}(m)} \xi^t_{\sigma,n|m}\right)$

    $\xi^{t+1}_{\sigma,m|n} = -\xi^t_{\sigma,n|m} + 2c\gamma^{t+1}_{\sigma,m}$,

  end

end

$\sigma^2_{rBCM} = 1/(\sigma_{**}^{-2} + M\gamma^{T_{max}+1}_{\sigma,m})$

$\mu_{rBCM} = \sigma^2_{rBCM,m} M\gamma^{T_{max}+1}_{\mu,m}$ /*Result can be obtained from any agents*/

local datasets are less correlated. For example, the dataset $\mathcal{D}_5$ is more correlated to $\mathcal{D}_4$ and $\mathcal{D}_2$ than $\mathcal{D}_1, \mathcal{D}_3$ and $\mathcal{D}_6$. Since the neighborhood $\mathcal{N}(m)$ of an agent m contains the $|\mathcal{N}(m)|$ closest agents to it, an assumption can be made that $\mathcal{N}(m)$ holds the $|\mathcal{N}(m)|$ most correlated datasets with that of agent m. Based on the assumption, it is proposed that the full NPAE aggregator based on the global datasets can be separated into M local NPAE aggregators maintained by every agents. For agent m, the local NPAE only involves $\{\mathcal{D}_m, \{\mathcal{D}_n\}_{n\in\mathcal{N}(m)}\}$, which alleviates the overhead of flooding datasets across the entire network.

For agent m, LOC-NPAE is equivalent to perform NPAE according to Equation (4.16) on a subgraph including agents $\{m, \mathcal{N}(m)\}$, where the edges are inherited from the full graph. The subgraph with agent m as the center unit is denoted as $\mathcal{G}_m = \{\mathcal{V}_m, \mathcal{E}_m\}$. Each agent m has to obtain $\mathbf{k}_{\mathcal{M},m}$ and $\mathbf{K}_{\mathcal{M},m}$ based on $\{\mathcal{D}_m, \{\mathcal{D}_n\}_{n\in\mathcal{N}(m)}\}$. As shown in Figure 4.2, there are three types of subgraphs that have an effect on the strategy of exchanging data in LOC-NPAE. The first type is shown in the lower left plot, where the selected agent 1 has only one neighbor. According to Equation (4.15), the $\mathbf{K}_{\mathbf{M},1}$ of size 2×2 can be obtained based on $\mathcal{D}_1$ and $\mathcal{D}_2$, which is directly accessible from agent 2.

The second type with agent 2 as the center unit is shown in the lower middle plot

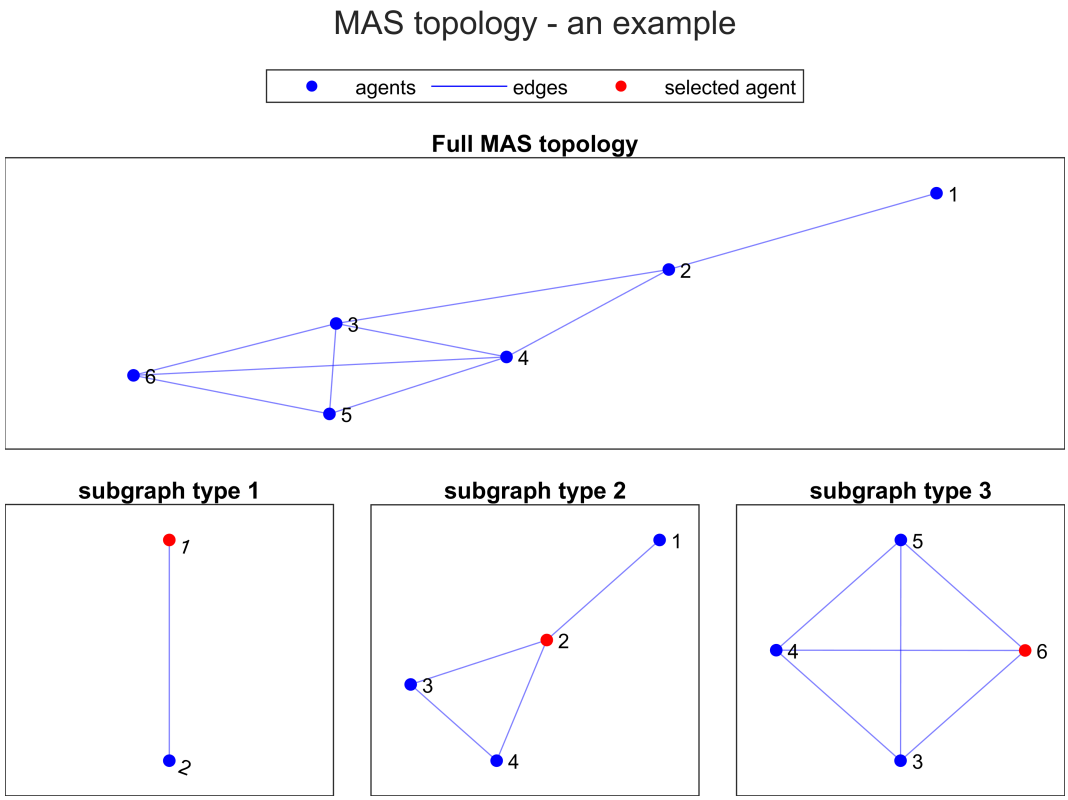


Figure 4.2: The upper figure shows the full topology of a MAS with 6 agents shown in clue dots. The lower figures shows three different kinds of subgraphs that LOC-NPAE operates on. In the lower plots, the red dots represent the agents selected as the local computing center, in which the LOC-NPAE is computed.

in Figure 4.2, where more than one neighbors of agent 2 exist. The agent 2 has to first prepare the $\mathbf{K}_{\mathcal{M},2}$ values that are associated with edge $(2,1),(2,3)$ and $(2,4)$ after receiving $\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4$. Then, the values associated with agent pairs $\{1,3\},\{1,4\}$ and $\{3,4\}$ has to be obtained. Because that there are no links between agents pairs $(1,3)$ and $(1,4)$, the corresponding values have to be calculated in agent 2. However, since an edge exists between agent 3 and 4, the value $\mathbf{K}_{*,3}\mathbf{K}_3^{-1}\mathbf{K}_{3,4}\mathbf{K}_4^{-T}\mathbf{K}_{*,4}^T$ can be prepared by either agent 3 or 4 before transmitting to agent 2, which reduces the computational complexity on agent 2.

The third type is a special case shown in lower right plot in Figure 4.2, where the local subgraph of agent 6 is actually a complete graph. For a complete subgraph, every values in $\mathbf{K}_{\mathcal{M},6}$ can be prepared by corresponding agent pairs, and then sent directly to agent 6.

Based on the NPAE in the aforementioned three types of subgraphs, the following strategy can be applied in LOC-NPAE. First, $\mathbf{K}_{*,m}$ and $\mathbf{K}_m^{-1}$ are calculated locally at every agent m, based on which value $\mathbf{K}_{*,m}\mathbf{K}_m^{-1}\mathbf{K}_{*,m}^T$ is locally available. Then, for agent pair (m,n), one of the agent, m for instance, collects the $\mathcal{D}_n$, $\mathbf{K}_{*,n}$ and $\mathbf{K}_n^{-1}$ from agent n and calculate $\mathbf{K}_{*,m}\mathbf{K}_m^{-1}\mathbf{K}_{m,n}\mathbf{K}_n^{-T}\mathbf{K}_{*,n}^T$, which is then also sent to agent n. If an agent pair (n_1,n_2) in $\mathcal{N}(m)$ has a link in between, the value $\mathbf{K}_{*,n_1}\mathbf{K}_{n_1}^{-1}\mathbf{K}_{n_1,n_2}\mathbf{K}_{n_2}^{-T}\mathbf{K}_{*,n_2}^T$ should

be directly available for agent m by collecting from either agent n_1 or n_2. For an agent pair (n_1', n_2') in $\mathcal{N}(m)$ without link, value $\mathbf{K}_{*,n_1'}\mathbf{K}_{n_1'}^{-1}\mathbf{K}_{n_1',n_2'}\mathbf{K}_{n_2'}^{-T}\mathbf{K}_{*,n_2'}^{T}$ is calculated by agent m. Finally, the local predictions (μ_n, σ_n^2) are collected from neighbors. Now, the $\mathbf{K}_{\mathcal{M},m}$ and $\mathbf{k}_{\mathcal{M},m}$ should be ready for agent m to perform LOC-NPAE.

The results of LOC-NPAE from agent m is denoted as $(\mu_{LN,m}, \sigma_{LN,m}^2)$. It should be noted that the LOC-NPAE on agent m only give confident prediction in the region covered by $\{\mathcal{D}_m, \{\mathcal{D}_n\}_{n\in\mathcal{N}(m)}\}$, but is less confident outside the region. To develop CON-NPAE, DEC-rBCM is applied after LOC-NPAE, so that the agents in the network finally obtain identical predictions about the entire region of interest.

Comparison of current methods and proposed methods are listed in Table 4.2.

Algorithm 8: Fully-distributed LOC-NPAE CON-NPAE

Input: $\mathbf{x}_*$, σ_{**}, $\{\boldsymbol{\theta}_m\}_{m=1}^M$, $\{\mathbf{X}_m, \mathbf{y}_m\}_{m=1}^M$, $\{d_m\}_{m=1}^M$
Output: μ_{rBCM}, σ_{rBCM}^2
Initialize: $c \in (0,1)$, $\{\xi_{\mu,m|n}^0\}_{m\in\mathcal{V},n\in\mathcal{N}(m)}$=0
for *agents* $m = \{1, 2, \cdots, M\}$ **do**
    $[\mu_m, \sigma_m^2] = GPR(\mathbf{x}_*, \mathbf{X}_m, \mathbf{y}_m, \boldsymbol{\theta}_m)$ `/*Local GP Prediction*/`
end
for *agents on edge* $(u, v) \in \mathcal{E}$ **do**
    exchange $\mathcal{D}_u$ and $\mathcal{D}_v$
    exchange $\mathbf{K}_u^{-1}$ and $\mathbf{K}_v^{-1}$
    exchange $\{\mu_u, \sigma_u^2\}$ and $\{\mu_v, \sigma_v^2\}$
    calculate $\mathbf{K}_{u,v} = Cov(\mathbf{X}_u, \mathbf{X}_v)$
end
for *agents* $m = \{1, 2, \cdots, M\}$ **do**
    for *each pair of agents* (u, v) *in* $\mathcal{N}(m)$ **do**
        if $\mathbf{K}_{u,v}$ *available at agent* u *or* v **then**
            Receive $\mathbf{K}_{u,v}$ from agent u or v
        else
            Calculate $\mathbf{K}_{u,v}$ in agent m
        end
    end
    Calculate $\mathbf{k}_{\mathcal{M},m}$ and $\mathbf{K}_{\mathcal{M},m}$
end
for *agent* $m = \{1, 2, \cdots, M\}$ **do**
    Calculate $\mu_{LOC-NPAE,m}$ and $\sigma_{LOC-NPAE,m}^2$ based on Equation (4.16).
end
Calculate $\mu_{CON-NPAE}$ and $\sigma_{CON-NPAE}^2$ from rBCM_{fd} (Algorithm 6 or 7) based on $\{\mu_{LOC-NPAE,m}\}_{m=1}^M$ and $\{\sigma_{LOC-NPAE,m}^2\}_{m=1}^M$

Table 4.2: Models for GP experts aggregation.

Methods	Author (Year)	Independent datasets	Expert importance	Prior knowledge	Fully-distributed
PoE	Ng and Deisenroth (2014) [34]	✓	✗	✗	✗
gPoE	Cao and Fleet (2014) [35]	✓	✓	✗	✗
DTCF-PoE	Kontoudis and Stilwell (2022) [43]	✓	✗	✗	✓
DTCF-gPoE	Kontoudis and Stilwell (2022) [43]	✓	✓	✗	✓
PDMM-PoE	proposed	✓	✗	✗	✓
PDMM-gPoE	proposed	✓	✓	✗	✓
BCM	Tresp (2000) [32]	✓	✗	✓	✗
rBCM	Deisenroth and Ng (2015) [36]	✓	✓	✓	✗
grBCM	Liu et al. (2018) [37]	✗	✓	✓	✗
DTCF-BCM	Kontoudis and Stilwell (2022) [43]	✓	✗	✓	✓
DTCF-rBCM	Kontoudis and Stilwell (2022) [43]	✓	✓	✓	✓
PDMM-BCM	proposed	✓	✗	✗	✓
PDMM-rBCM	proposed	✓	✓	✗	✓
NPAE	Rullière et al. (2018) [38]	✗	✗	✓	✗
NPAE-JOR	Kontoudis and Stilwell (2022) [43]	✗	✗	✓	Only in complete graph
CON-NPAE	proposed	partly	✓	✓	✓

4.4 Simulation

4.4.1 Quality assessment

Under the presumption that the hyperparameters of GP model have been optimized, it is assumed that the real underlying field can be recovered by the full GPR. To assess the performance of methods, the aggregation results are compared to the results given

by full GPR through root-mean-square error (RMSE) given by

$$e_{RMSE} = \frac{1}{N_{exp}} \sum_{p=1}^{N_{exp}} \sqrt{\frac{1}{M} \sum_{m=1}^{M} \frac{1}{N_*} \sum_{n=1}^{N_*} (y_{*,m,n} - y_{*,full,n})^2}, \tag{4.27}$$

where N_{exp} is the number of repeated simulations under different GP fields, N_* is the number of values to be predicted, $y_{*,m,n}$ is the prediction of nth points by agent m, and $y_{*,full,n}$ is corresponding prediction given by the full GPR.

4.4.2 Artificial dataset

To test the performance of aforementioned aggregation methods, the simulation is repeated for $N_{exp} = 3$ times with different underlying fields. For each time, a global training dataset $\mathcal{D}$ is sampled from a field generated by stationary 2D Gaussian Process with hyperparameters $\{\sigma_f, l_1, l_2\} = \{5, 1, 1\}$ and measurement error $\sigma_n = \sqrt{0.1}$. The span of the field is $[-5, 5] \times [-5, 5]$. The points to be predicted are evenly spread over the entire field on a 100×100 grid. There are totally the fixed number of 1600 sampling points in the datasets, which are divided into M partially overlapping local datasets. There are 6 M values examined, respectively 2, 4, 8, 12 and 16, which means the corresponding local datasets sizes are respectively 800, 400, 200, 130 and 100.

The fully-distributed gPoE, BCM and rBCM are examined based on both DTCF and PDMM algorithms, which are applied with a fixed number of 20 iterations. For DTCF, as the convergence requires that $\nu \leq \frac{1}{\Delta}$, the step size is set to $\nu = 0.9\frac{1}{\Delta} < \frac{1}{\Delta}$. The penalty parameter of PDMM is set to $c = 0.35$, which is an experimental value for consensus average problem in randomly generated graph.

The NPAE-JOR is simulated with a maximum 400 iterations with $\nu = 0.9\frac{2}{M}$, which is smaller than the theoretical maximum $\nu = \frac{2}{M}$ to achieve convergence. The NPAE-DALE does converge for unknown reasons, so the results are not included. Parameters used in the methods are listed in Table 4.3. It should be noticed that the defined parameter values are also applied in simulation of real dataset.

As for the CON-NPAE, their is no parameters to be controlled. The CON-NPAE is simulated by concatenating NN-NPAE and fully-distributed rBCM, where the rBCM adapts the setting as shown in Table 4.3. The Figure 4.3 shows the RMSE of predictive means and variances of examined methods with DTCF algorithm. The Figure 4.4 shows the RMSE of predictive means and variances of examined methods with PDMM algorithm. In Figure 4.5, the RMSE of PDMM and DTCF based algorithms are compared.

PDMM based methods From Figure 4.5, it can be found that the PDMM based methods start to perform better than DTCF based methods with the agent number larger than 4. When the number of agents is smaller or equal to 4, MASs are very small, in which the iteration numbers needed for the convergence of PDMM and DTCF are small. As the number of agents increases, the network becomes more complicated, where more iterations are required for convergence. In this situation, the PDMM has increasingly larger advantage over DTCF in terms of smaller error under the same

Table 4.3: Parameters & variables of GP aggregation

Parameters & variables	Values	Comment
N_{exp}	3	Number of simulation
N_*	10000	Points to be predicted
$\|\mathcal{D}\|$	1600	Size of training dataset
M	$\{2, 4, 8, 12, 16\}$	Number of agents
$\{\sigma_f, l_1, l_2, \sigma_n\}$	$\{5, 1, 1, \sqrt{0.1}\}$	Hyperparameters and noise variance
ν	$0.9\frac{1}{\Delta}$	Used in DTCF based fully-distributed PoE and BCM families
c	0.35	Used in PDMM based fully-distributed PoE and BCM families
ν	$0.9\frac{2}{M}$	Used in NPAE-JOR

number of iterations. The Figure A.17 shows an example of the consensus error curves of DTCF and PDMM.

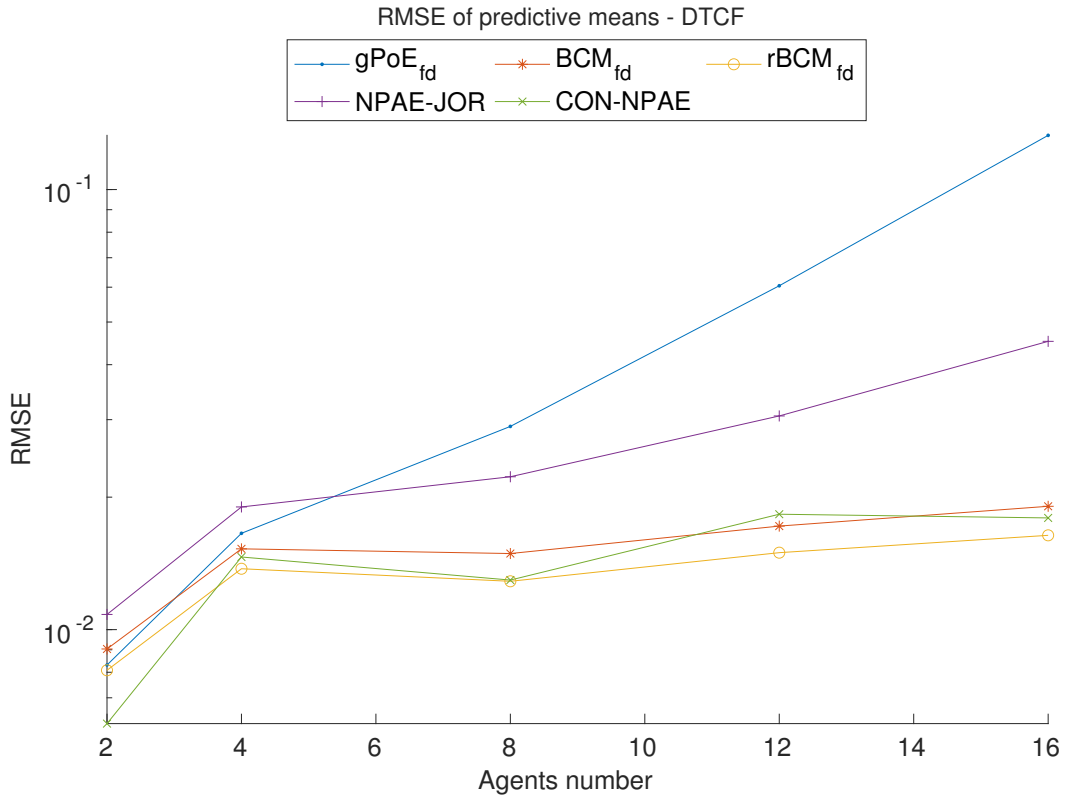
It should also be noticed that the DTCF relies on the right choice of ν value to converge, which requires that the agents have the knowledge of the global topology of MAS. As for the PDMM, there are experimental values of penalty parameters $c \in (0.3, 0.5)$ for consensus average problem. Thus, the PDMM based methods are better choices when the agents are assumed to have no knowledge of the global network structure.

NPAE-JOR and CON-NPAE As shown in Figure 4.3, the NPAE family of methods do not achieve the expected performance. For the NPAE-JOR method, the RMSE is always higher than those of PDMM or DTCF based PoE and BCM families of methods. A possible reason is that the choice of ν value is not optimal, so that the JOR algorithm does not converge to the right $\mathbf{q}$ values in Equation (4.20). The results indicate that the application of NPAE-JOR requires fine tune of parameters, which may be tricky. Further work on automatically optimizing the choice of ν could be done to improve the performance, but is out of the topic of environmental field learning.

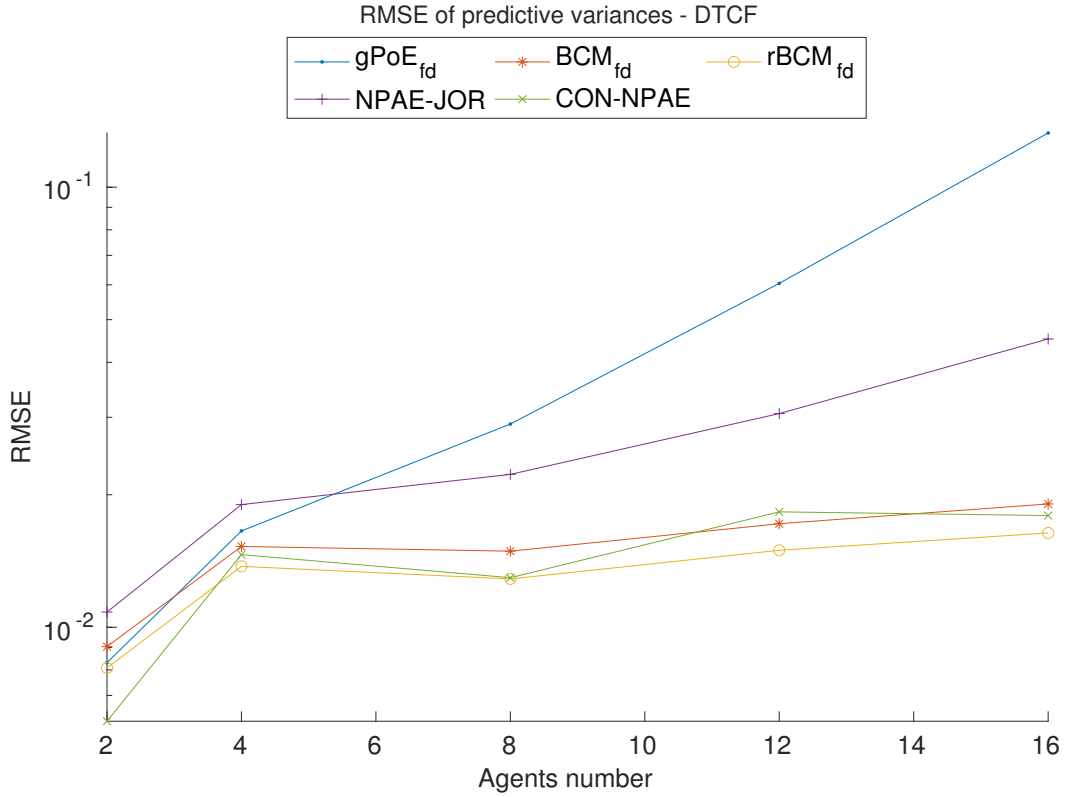
As for the CON-NPAE, the results from Figure 4.3 and 4.4 show that the performance of CON-NPAE is constrained by the performance of rBCM_{fd} when the number of agents is larger than 2, which is far from expectation. To analyze the reason, it should be noticed that the proportion of the proportion of local datasets size in neighborhood to global dataset generally becomes smaller as the number of agents increases, which means that $\frac{|\{\mathcal{D}_m, \{\mathcal{D}_n\}_{n\in\mathcal{N}(m)}\}|}{|\mathcal{D}|}$ reduces. An agent performing NN-NPAE can utilize the information from its own and neighbors' datasets. When the number of agents is small, the local neighborhood already contains the most proportion of the global dataset, which means that the NN-NPAE can utilize almost all of global information. Thus, with the cross-correlation considered, the NN-NPAE and CON-NPAE have better performance than BCM family of methods as expected. Specifically, these two methods are just NPAE when number of agents is 2. However, when the number of agents increase,

the local neighborhood contains smaller proportion of data points compared to the total number of data points. Under this situation, even if the NN-NPAE can utilize the local neighborhood datasets $\{\mathcal{D}_m, \{\mathcal{D}_n\}_{n\in\mathcal{N}(m)}\}$ better than independence assumption based methods, the CON-NPAE performance is still dominated by the rBCM_{fd}.

Though NPAE based methods are expected to have theoretically better performance than independence assumption based methods, the improvement comes at a cost of much higher computational complexity. Considering that the computational, memory and communication abilities are usually restricted for MAS, the current NPAE family of methods may be not have obvious advantage over independence assumption based methods.

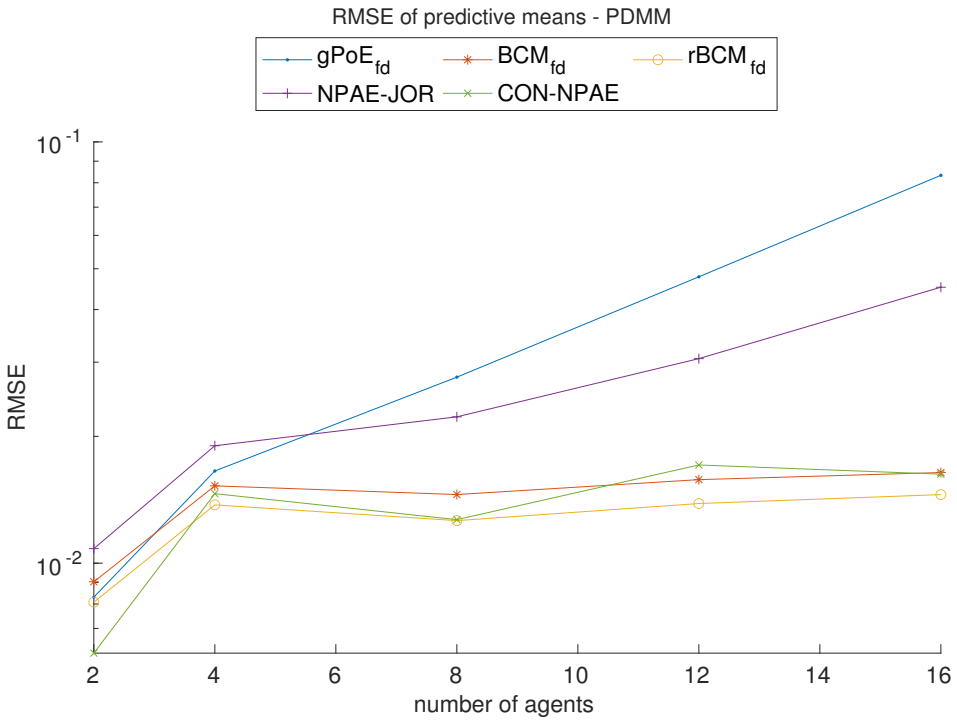


(a) RMSE of predictive means.

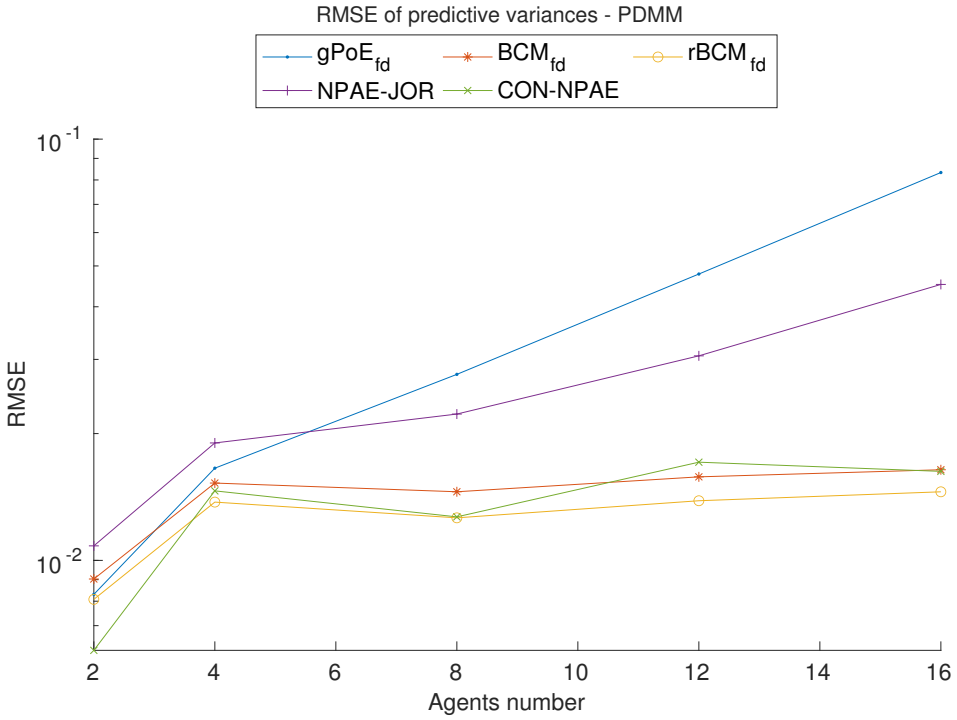


(b) RMSE of predictive variances.

Figure 4.3: The figure shows the RMSE comparison of the simulated methods. The consensus average algorithm applied is DTCF. (a). The RMSE comparison of the predictive means. (b). The RMSE comparison of the predictive variances.

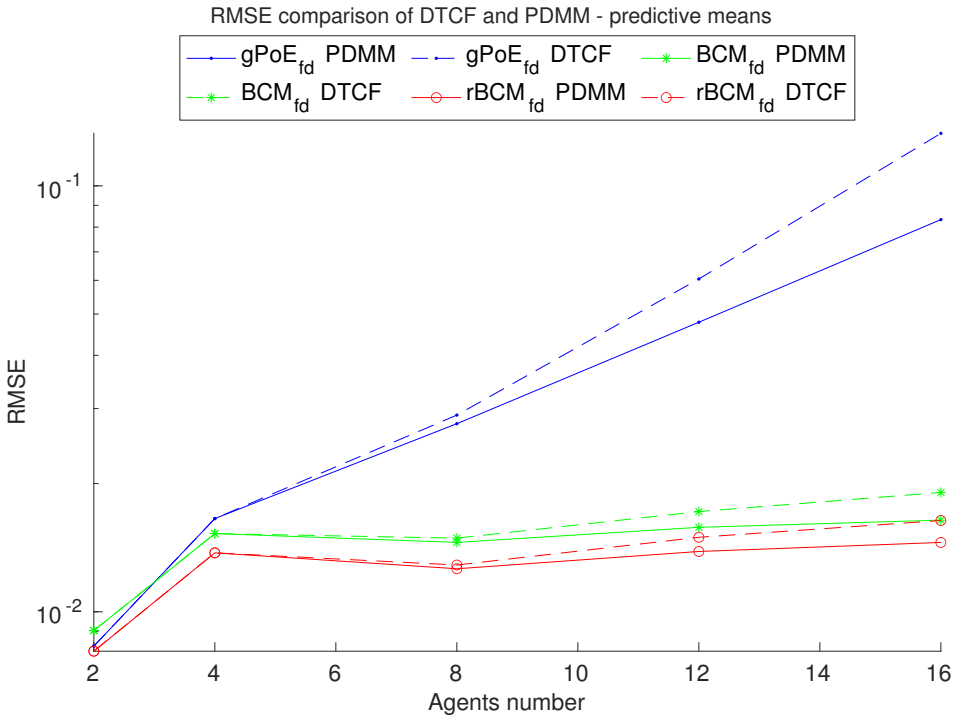


(a) RMSE of predictive means.

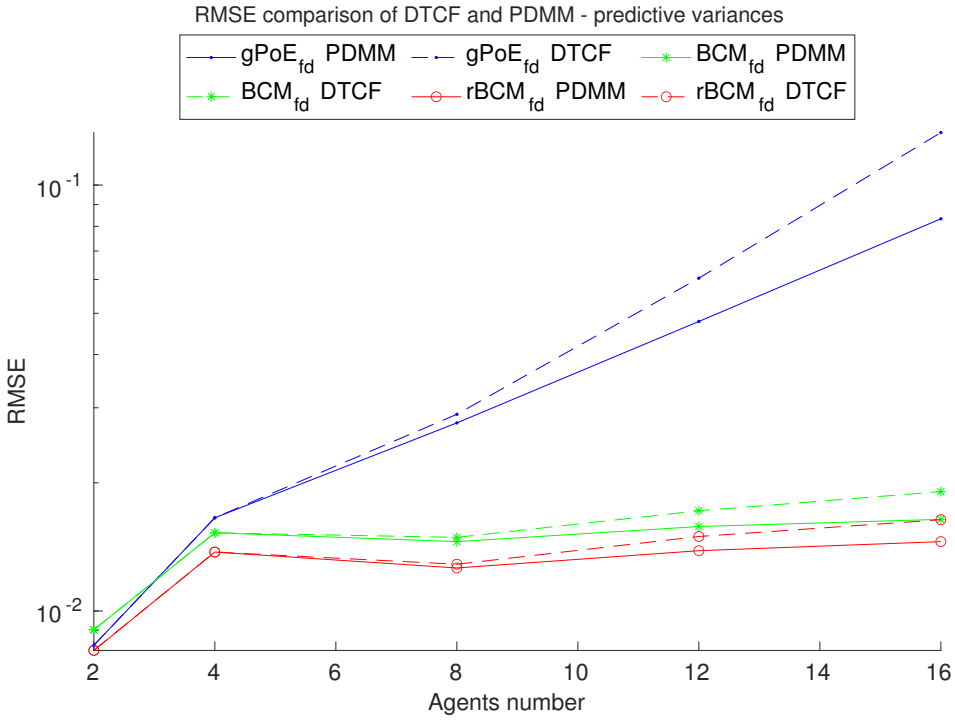


(b) RMSE of predictive variances.

Figure 4.4: The figure shows the RMSE comparison of the simulated methods. The consensus average algorithm applied is PDMM. (a). The RMSE comparison of the predictive means. (b). The RMSE comparison of the predictive variances.



(a) RMSE of predictive means.



(b) RMSE of predictive variances.

Figure 4.5: The figure shows the RMSE comparison DTCF and PDMM based methods. The methods applied include both DTCF and PDMM version of PoE, gPoE, BCM and rBCM

4.4.3 Real dataset: GHRSST

The methods are also simulated in a field cropped from the GHRSST datasets. Due to time limit, only one subset of the real dataset is selected for simulation. The ranges of the field are respectively $[145.0, 150.5]$ and $[37.0, 40.0]$ on longitude and latitude directions (or referred as x and y axes). The field is shown in the background in Figure 3.6a. It should be noticed that the graph topology in the simulations for real datasets are different from those applied to artificial datasets.

Table 4.4: Parameters & variables of GP aggregation

Parameters & variables	Values	Comment
N_{exp}	1	Number of simulation
N_*	10000	Points to be predicted
$\|\mathcal{D}\|$	1600	Size of training dataset
M	$\{2, 4, 8, 12, 16\}$	Number of agents
$\{\sigma_f, l_1, l_2, \sigma_n\}$	$\{8.5, 1.6, 1.2, 0.7\}$	Hyperparameters and noise variance trained by pxADMM$_{fd}$
ν	$0.9\frac{1}{\Delta}$	Used in DTCF based fully-distributed PoE and BCM families
c	0.35	Used in PDMM based fully-distributed PoE and BCM families
ν	$0.9\frac{2}{M}$	Used in NPAE-JOR

The Figure 4.6 shows the RMSE of predictive means and variances of examined methods. In Figure 4.7, the RMSE of PDMM and DTCF based algorithms are compared.

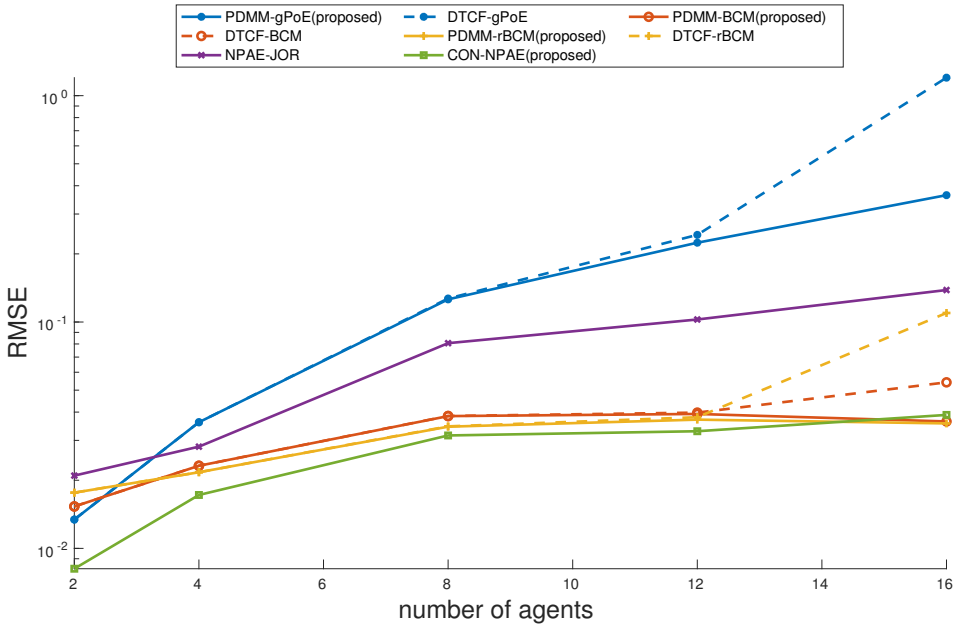
PDMM based methods As shown in Figure 4.7, the PDMM start to have obviously better performance than DTCF when the number of agent is larger than 12, which is different from the results shown in Figure 4.5. A reason contributing to the difference is that the connectivity of graphs are high for the numbers of agents 2, 4, 8 and 12, which is illustrated in Figure A.19. The result shows that PDMM and DTCF based aggregations are likely to give similar results in MAS with good connectivity, but PDMM performs better when the network is not highly connected.

NPAE-JOR and CON-NPAE From the Figure 4.6, it can be found that the errors of NPAE-JOR are sometimes lower than errors of DTCF-gPoE, DTCF-BCM or DTCF-rBCM. However, NPAE-JOR still have generally larger errors than the DTCF-BCM and DTCF-rBCM methods, which indicates that the JOR algorithm applied here is still not stable with non-optimal choice of ν parameter.

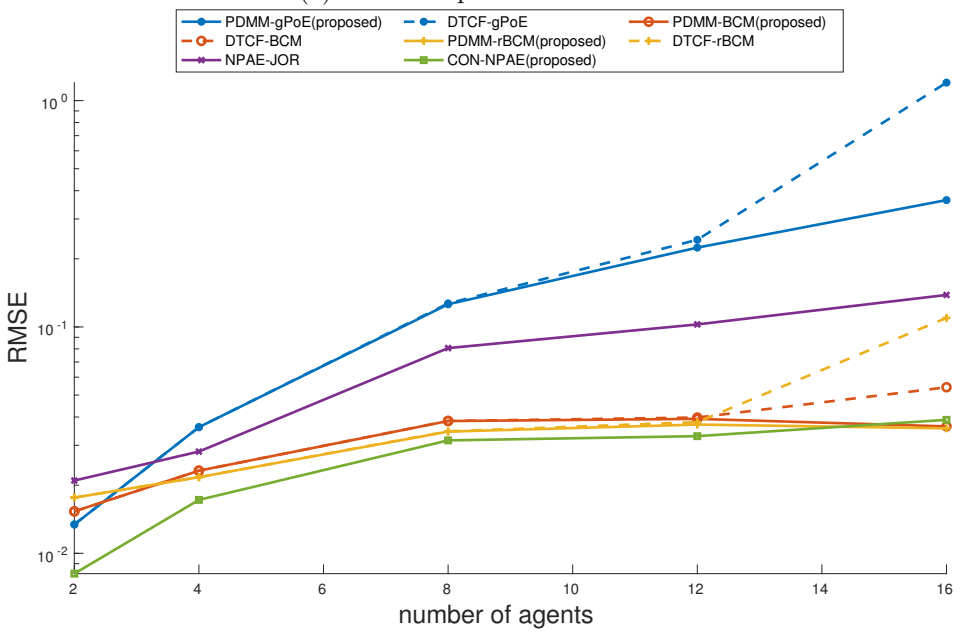
As for the CON-NPAE, the method performs better than the other methods, except when the number of agents is 16 that it has same performance as DTCF-rBCM. The

reason is the same as that analyzed in Section 4.4.2, that is the CON-NPAE can utilize lower proportion of global dataset, which counteract the performance improvement of NPAE.

The results of CON-NPAE from both artificial and real datasets indicate that the performance of CON-NPAE is associated with the graph structure, especially with the graph connectivity, which could be further studied in future works.

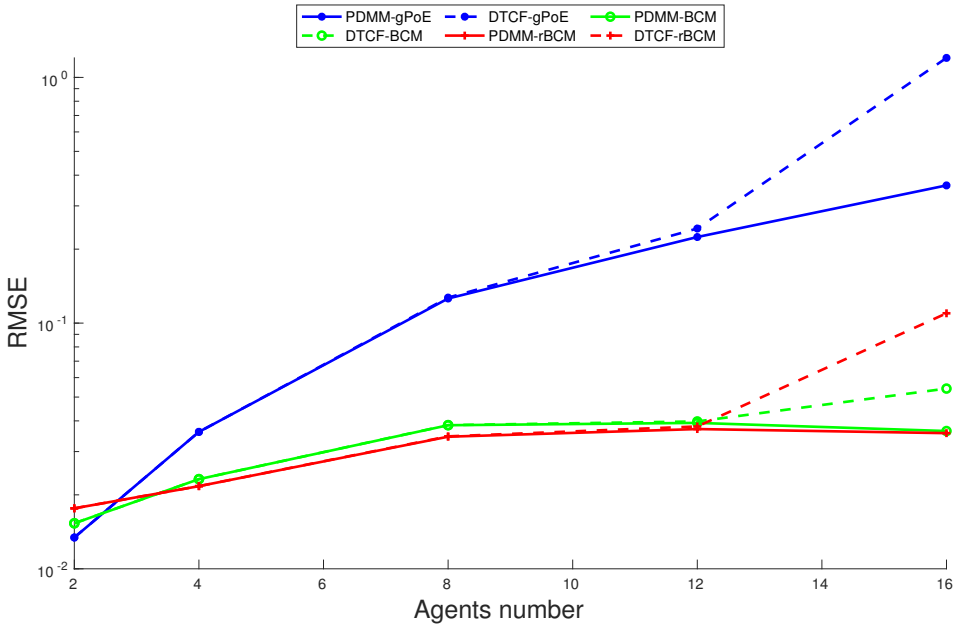


(a) RMSE of predictive means.

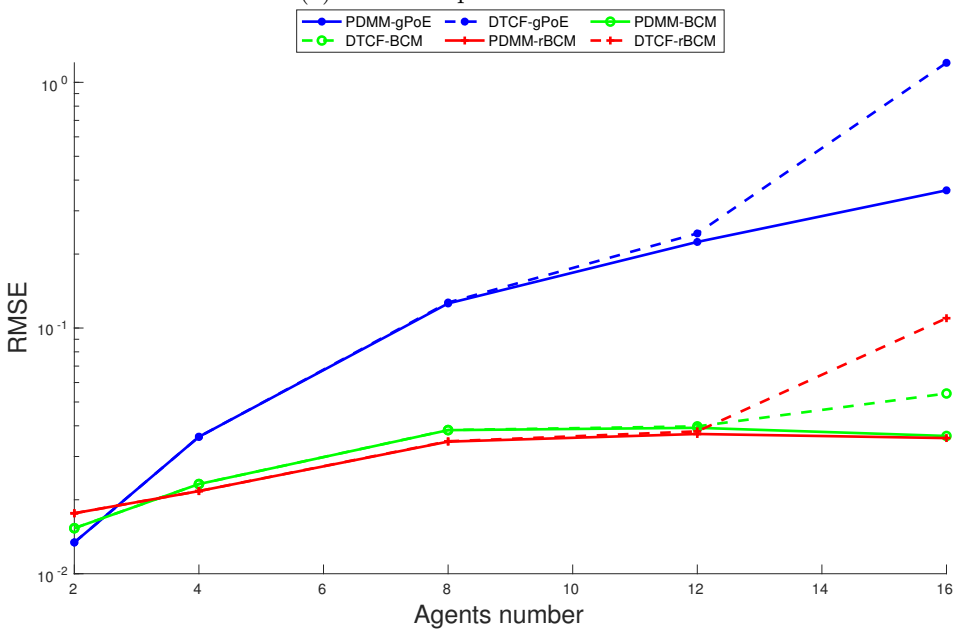


(b) RMSE of predictive variances.

Figure 4.6: The figure shows the RMSE comparison of the simulated methods. The consensus average algorithm applied is DTCF. (a). The RMSE comparison of the predictive means. (b). The RMSE comparison of the predictive variances.



(a) RMSE of predictive means.



(b) RMSE of predictive variances.

Figure 4.7: The figure shows the RMSE comparison DTCF and PDMM based methods. The methods applied include both DTCF and PDMM version of PoE, gPoE, BCM and rBCM

4.5 Conclusion

1. Current methods:

   - (g)PoE/(r)BCM methods with main advantage of low computational complexity. Their drawbacks are mainly inconsistent prediction regards to the real underlying function or full GPR results. Their distribution are realized through assistance of consensus average algorithm DTCF.
   - NPAE method with main advantage of providing consistent predictions. The drawback meanly lies in 1). the large computational overhead due to the flooding of necessary data points across the network, 2). and also the complete network requirement. The NPAE is distributed through reformulating the problem into distributed linear algebraic problems that can be solved by distributed solvers, e.g., JOR or DALE. The drawback of the distributed NPAE lies in the 1). large flooding overhead, 2). and parameters to be tuned in JOR.

2. Main contributions:

   - Proposing PDMM based DEC-(g)PoE/(r)BCM methods, which decrease the number of iterations needed for aggregation.
   - Proposing LOC-NPAE and CON-NPAE methods, which alleviate the flooding overhead of classical NPAE, NPAE-JOR and NPAE-DALE. The proposed methods are expected to alleviate both computational and memory complexity compared to previous NPAE based methods. However, the simulation results shows that the performance of CON-NPAE can be constrained by the DTCF-rBCM, especially when the number of agents increase. Because, as the number of agents increase, the contribution of NPAE to the aggregation is relatively reduced, while the DTCF-rBCM becomes dominant.

5 Discussion and Conclusion

In this chapter, the contributions of this thesis are further discussed. Possible future works are also introduced.

5.1 Conclusion - Hyperparameter Optimization

5.1.1 Research problem

In Chapter 3, the fully-distributed Gaussian Process hyperparameters optimization problem is introduced and discussed. By optimizing the hyperparameters, Gaussian Process can be optimized to best fit the given training datasets in terms of Likelihood in Equation (3.3). The optimization problem is formulated as problem $\mathcal{P}_0$ in Equation (3.4), which is restated as follow

$$\mathcal{P}_0 : \quad \min_{\boldsymbol{\theta}} \quad l(\boldsymbol{\theta}) = \mathbf{y}^T \mathbf{K}^{-1}(\boldsymbol{\theta})\, \mathbf{y} + \log |\mathbf{K}(\boldsymbol{\theta})| \,. \tag{5.1}$$

Recall the problem stated in Chapter 1 about hyperparameter optimization

- In a MAS, how to train Gaussian Process models, i.e., learning hyperparameters from sampling points in the unknown fields, for environmental monitoring applications in a fully-distributed manner that does not relies on center station.

5.1.2 Current methods

To solve the problem in MAS, the target function can be divided through dividing dataset into local datasets on agents, which results in problem $\mathcal{P}_1$ in Equation (3.5). By further restricting GP to be stationary across the field, the optimization problem can be formulated as $\mathcal{P}_2$ in Equation (3.9) by adding consensus constraints on hyperparameters. Further modification to edge-based constraints gives fully-distributed version as $\mathcal{P}_3$ in Equation (3.15).

Classical methods applied to the aforementioned problem includes nGD and ADMM (centralized and distributed). A state-of-the-art solution to the problem is pxADMM as introduced in Section 3.2.3, which adapts proximal hyperparameters update so that exact optimization of a sophisticated target function is avoided.

5.1.3 Proposed methods

Based on the pxADMM and fully-distributed ADMM, two algorithms are proposed to extend pxADMM to fully-distributed algorithms.

- The first one successfully extends the pxADMM so that the optimization can be applied in fully-distributed MAS. As shown by the simulation results in Figure A.5 and 3.4, the pxADMM$_{\text{fd}}$ converges to the preset hyperparameters, though much slower than the centralized pxADMM. It has been proven [24] that the convergence can be achieved as long as ρ and L in Equation (3.21) are large enough, where ρ controls the weight of quadratic term in augmented Lagrangian, and L should be larger than the gradient of $l(\boldsymbol{\theta})$ everywhere.
- The second methods pxADMM$^*_{\text{fd}}$ is also based on ADMM$_{\text{fd}}$, but adapts hyperparameters update directly from pxADMM, where the global $\mathbf{z}$ and local $\boldsymbol{\lambda}_m$ are replaced by their local counterparts. Though the convergence of this method has not been mathematically proven, experimental results shows convergence under large enough ρ and L values. The method also converge faster than pxADMM$_{\text{fd}}$, and has equivalent convergence speed as centralized pxADMM.

In a synchronized network, fast agents has to wait for slow agents, which slower down the processing speed in the network for each iteration. Also, synchronization in large MAS is itself a burden. To solve the problem, asynchronous versions of fully-distributed pxADMM are developed based on two strategies.

- The partial barrier specifies that an agent do not need to receive all updates from neighbors, which introduces asynchronous behavior.
- The bounded delay ensures that a link between two agents is always activated after certain number of iterations.

5.1.4 Conclusion

The proposed algorithms pxADMM$_{\text{fd}}$, pxADMM$^*_{\text{fd}}$ are fully-distributed ones that can be fitted into MAS. The simulation results in both artificial and real datasets show that the algorithms can stably learn hyperparameter sets based on limited sampling points from the unknown environmental field. With a proper choice of hyperparameters, the GP can well model the scalar field in environmental monitoring task.

The proposed asynchronous algorithms pxADMM$_{\text{async}}$, pxADMM$^*_{\text{async}}$ converge to the same value under the same number of update iterations as the synchronous algorithms. The asynchronous behavior can provide MAS with higher robustness in environmental monitoring task.

5.2 Conclusion - Distributed Aggregation

5.2.1 Research problem

In Chapter 4, the fully-distributed Gaussian Process prediction aggregation problem is introduced and discussed. Recall the problem statement from Chapter 1.

- Based on the sampled data points, how does a MAS with distributed Gaussian Process models predicts the unknown field with an compatible accuracy as the full Gaussian Process.

According to the statement, the goal of GP aggregation is to recover the underlying environmental field based on local predictions in MAS, such that the quality of prediction is comparable with full GPR. At the same time, the computational complexity should also be considered.

5.2.2 Current methods

The current efforts of solving the problem can be classified into two groups based on whether the method makes independent assumption on the local datasets. Under independence assumption, the posterior distribution can be totally factorized into product of local posterior distributions as explained in Section 4.1.3. Based on the assumption, the PoE, gPoE, BCM and rBCM have been proposed. On the contrary, the other group of methods, including NPAE and grBCM, consider the cross-correlations among local datasets.

As shown in Section 4.2.1, the PoE and BCM families of methods can be distributed through solving distributed consensus average problem. The current algorithm applied is DTCF, of which the convergence speed is affected by the graph connectivity.

The NPAE method can be distributed through solving linear algebraic problems. The current solutions include JOR in complete network and DALE for connected network. Even though DALE is claimed to extend the algorithm to connected graph, the local datasets still need to be flooded through the network before aggregation, which is a time-consuming and computational expensive process.

5.2.3 Proposed methods

To reduce the iterations needed for fully-distributed PoE and BCM methods, PDMM is introduced to solve the consensus average problem. PDMM has been proved to converge for the distributed consensus average problem at a rate of $\mathcal{O}(\frac{1}{t})$, where the t represents the number of iterations.

Based on the simulation results from Chapter 4, the PDMM based PoE and BCM families of methods have shown improved performance compared to their DTCF versions, i.e., fewer iterations are needed to achieve the same error. Thus, the PDMM based aggregation methods are suitable for application on MAS, where the computational and communication abilities are usually constrained.

To extend the NPAE method to fully-distributed aggregation, the NN-NPAE and CON-NPAE methods are proposed. The assumption of NN-NPAE is that the local neighborhood contains the most relevant datasets to the selected agents. Based on the assumption, the NN-NPAE on a single agent only perform NPAE based on the datasets from local neighborhood. In order to let the local agents reconstruct the global field, the rBCM_{fd} is concatenated with NN-NPAE to provide global aggregation.

As for the performance, the CON-NPAE only performs better than independence assumption based methods when the network is small. The reason is that, though NN-NPAE provides the agents with good aggregations around local areas, more information are provided by the rBCM_{fd}, which limits the general performance of CON-NPAE method.

Also, it is found from the simulation results that the current method of NPAE-JOR and NPAE-DALE do not show high robustness of convergence, but requires fine tune of parameters, which could be the drawback of applying them in MAS.

5.2.4 Conclusion

The proposed PDMM based PoE and BCM families of methods require smaller number of iterations for the GP aggregation task than their DTCF based counterparts. The PDMM based algorithms reduce the computational complexity of the environmental monitoring task based on GP. The algorithms are also favored in terms of the communication and computation energy saved in MAS.

The proposed NN-NPAE and CON-NPAE extend the NPAE methods to fully-distributed situation, such that flooding variables across MAS is not required, which reduces the computational complexity compared with the current NPAE-JOR algorithm. In small network with high connectivity, CON-NPAE outperforms the PoE and BCM families of methods, but has equal or worse performance in large network with low connectivity. Future work can be focused on related issue.

5.3 Future work

5.3.1 Distributed hyperparameter optimization

- As discussed in Section 3.3.1 and Section 3.4, the pxADMM$^*_{\text{fd}}$ has shown to converge in the simulations, but not been proved analytically. Future research could be down on the convergence analysis of pxADMM$^*_{\text{fd}}$.
- The Cross-Validation based methods are not focused in this thesis. However, it would also be interesting to study the CV based GP hyperparameter optimization in MAS.

5.3.2 Distributed GP aggregation

- As discussed in Section 4.1.4, the centralized NPAE have been proven to be consistent GP aggregation methods. However, simulations from Section 4.4 shows the distribution of NPAE comes at a cost of high computational complexity and unstable convergence. Another potential solution is the NPAE based on inducing points methods [48]. The idea is that only small proportion of the most representative points from local datasets are selected to be exchanged among agents, which could reduce the computational, memory and especially communication complexity.
- Based on the simulation results, it can be noticed that the graph structure plays a role in the aggregation performance of different methods, which, however, is not extensively studied in this thesis. The influence of graph structure on GP aggregation performance could be a meaningful topic as future research.

Bibliography

[1] A. Xie, F. Yin, Y. Xu, B. Ai, T. Chen, and S. Cui, "Distributed Gaussian Processes Hyperparameter Optimization for Big Data Using Proximal ADMM," *IEEE Signal Processing Letters*, vol. 26, no. 8, pp. 1197–1201, 2019.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.

[3] N. A. Atanasov, J. Le Ny, and G. J. Pappas, "Distributed algorithms for stochastic source seeking with mobile robot networks," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, vol. 137, no. 3, pp. 1–9, 2015.

[4] J. Choi, S. Oh, and R. Horowitz, "Distributed learning and cooperative control for multi-agent systems," *Automatica*, vol. 45, no. 12, pp. 2802–2814, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.automatica.2009.09.025

[5] K. Tiwari, V. Honore, S. Jeong, N. Y. Chong, and M. P. Deisenroth, "Resource-constrained decentralized active sensing for multi-robot systems using distributed Gaussian processes," *International Conference on Control, Automation and Systems*, vol. 0, no. Iccas, pp. 13–18, 2016.

[6] A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand, "Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets," *Journal of the American Statistical Association*, vol. 111, no. 514, pp. 800–812, 2016. [Online]. Available: http://dx.doi.org/10.1080/01621459.2015.1044091

[7] F. Yin and F. Gunnarsson, "Distributed Recursive Gaussian Processes for RSS Map Applied to Target Tracking," *IEEE Journal on Selected Topics in Signal Processing*, vol. 11, no. 3, pp. 492–503, 2017.

[8] N. E. Leonard, D. A. Paley, F. Lekien, R. Sepulchre, D. M. Fratantoni, and R. E. Davis, "Collective motion, sensor networks, and ocean sampling," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 48–74, 2007.

[9] K. H. Rosen, *Discrete Mathematics and its applications*, 2019. [Online]. Available: https://academic.oup.com/teamat/article-lookup/doi/10.1093/teamat/hrq007

[10] A. T. A. Wood and G. Chan, "Simulation of Stationary Gaussian Processes in $[0, 1]^d$," *Journal of Computational and Graphical Statistics*, vol. 3, no. 4, p. 409, dec 1994. [Online]. Available: https://www.jstor.org/stable/1390903?origin=crossref

[11] Z. Botev, "Circulant Embedding method for generating stationary Gaussian field," 2016. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/38880-circulant-embedding-method-for-generating-stationary-gaussian-field

[12] NASA/JPL, "GHRSST Level 4 MUR Global Foundation Sea Surface Temperature Analysis (v4.1)," 2015. [Online]. Available: http://podaac.jpl.nasa.gov/dataset/MUR-JPL-L4-GLOB-v4.1

[13] T. M. Chin, J. Vazquez-Cuervo, and E. M. Armstrong, "A multi-scale high-resolution analysis of global sea surface temperature," *Remote Sensing of Environment*, vol. 200, no. December 2016, pp. 154–169, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.rse.2017.07.029

[14] S. Theodoridis, "Chapter 3 - Learning in Parametric Modeling: Basic Concepts and Directions," in *Machine Learning (Second Edition)*, second edi ed., S. Theodoridis, Ed. Academic Press, 2020, pp. 67–120. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012818803300012X

[15] E. Contal, D. Buffoni, A. Robicquet, and N. Vayatis, "Parallel Gaussian Process Optimization with Pure Exploration," *Springer*, pp. 225–240, 2013. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-40988-2_15%0Ahttp://www.ecmlpkdd2013.org/wp-content/uploads/2013/07/460.pdf

[16] P. Congdon, *Applied Bayesian Modelling*, ser. Wiley Series in Probability and Statistics. Chichester, UK: John Wiley & Sons, Ltd, mar 2003, vol. 59. [Online]. Available: http://doi.wiley.com/10.1002/0470867159

[17] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*. the MIT Press, 2006. [Online]. Available: www.GaussianProcess.org/gpml

[18] M. L. Stein, *Interpolation of Spatial Data*, ser. Springer Series in Statistics. New York, NY: Springer New York, 1999. [Online]. Available: http://link.springer.com/10.1007/978-1-4612-1494-6

[19] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. USA: Prentice-Hall, Inc., 1993.

[20] L. Xu, F. Yin, J. Zhang, Z.-Q. Luo, and S. Cui, "A General $\mathcal{O}(n^2)$ Hyper-Parameter Optimization for Gaussian Process Regression with Cross-Validation and Non-linearly Constrained ADMM," 2019. [Online]. Available: http://arxiv.org/abs/1906.02387

[21] A. Kaplan and R. Tichatschke, "Proximal Point Methods and Nonconvex Optimization," *Journal of Global Optimization*, vol. 13, no. 4, pp. 389–406, 1998.

[22] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2009.

[23] R. T. Rockafellar, "Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming." *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.

[24] M. Hong, Z. Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[25] R. Zhang and J. T. Kwok, "Asynchronous distributed ADMM for consensus optimization," *31st International Conference on Machine Learning, ICML 2014*, vol. 5, no. 2, pp. 3689–3697, 2014.

[26] E. Wei and A. Ozdaglar, "On the O(1/k) Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers," *2013 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013 - Proceedings*, pp. 551–554, jul 2013. [Online]. Available: http://arxiv.org/abs/1307.8254

[27] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," *Proceedings of the IEEE Conference on Decision and Control*, pp. 3671–3676, 2013.

[28] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of Gaussian process experts," *Advances in Neural Information Processing Systems*, 2002.

[29] E. Meeds and S. Osindero, "An alternative infinite mixture of Gaussian Process experts," *Advances in Neural Information Processing Systems*, pp. 883–890, 2005.

[30] C. Yuan and C. Neubauer, "Variational mixture of Gaussian process experts," *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, pp. 1897–1904, 2009.

[31] Y. Shen, A. Y. Ng, and M. Seeger, "Fast Gaussian process regression using kd-trees," *Advances in Neural Information Processing Systems*, pp. 1225–1232, 2005.

[32] V. Tresp, "A Bayesian Committee Machine," *Neural Computation*, vol. 12, no. 11, pp. 2719–2741, nov 2000. [Online]. Available: https://direct.mit.edu/neco/article/12/11/2719-2741/6426

[33] G. E. Hinton, "Training Products of Experts by Minimizing Contrastive Divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, aug 2002. [Online]. Available: https://direct.mit.edu/neco/article/14/8/1771-1800/6687

[34] J. W. Ng and M. P. Deisenroth, "Hierarchical Mixture-of-Experts Model for Large-Scale Gaussian Process Regression," 2014. [Online]. Available: http://arxiv.org/abs/1412.3078

[35] Y. Cao and D. J. Fleet, "Generalized Product of Experts for Automatic and Principled Fusion of Gaussian Process Predictions," no. ii, pp. 1–5, 2014. [Online]. Available: http://arxiv.org/abs/1410.7827

[36] M. P. Deisenroth and J. W. Ng, "Distributed Gaussian processes," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, 2015, pp. 1481–1490.

[37] H. Liu, J. Cai, Y. Wang, and Y.-S. S. Ong, "Generalized robust Bayesian committee machine for large-scale Gaussian process regression," in *International Conference on Machine Learning*. PMLR, jun 2018, pp. 3131–3140. [Online]. Available: http://arxiv.org/abs/1806.00720

[38] D. Rullière, N. Durrande, F. Bachoc, and C. Chevalier, "Nested Kriging predictions for datasets with a large number of observations," *Statistics and Computing*, vol. 28, no. 4, pp. 849–867, jul 2018. [Online]. Available: http://link.springer.com/10.1007/s11222-017-9766-2

[39] F. Bachoc, N. Durrande, D. Rullière, and C. Chevalier, "Properties and Comparison of Some Kriging Sub-model Aggregation Methods," *Mathematical Geosciences*, no. 20, 2022.

[40] D. Gu and H. Hu, "Spatial gaussian process regression with mobile sensor networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1279–1290, 2012.

[41] G. Pillonetto, L. Schenato, and D. Varagnolo, "Distributed multi-agent Gaussian regression via finite-dimensional approximations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2098–2111, 2018.

[42] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, jan 2007. [Online]. Available: http://ieeexplore.ieee.org/document/5485032/http://ieeexplore.ieee.org/document/4118472/

[43] G. P. Kontoudis and D. J. Stilwell, "Fully Decentralized, Scalable Gaussian Processes for Multi-Agent Federated Learning," 2022. [Online]. Available: http://arxiv.org/abs/2203.02865

[44] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, 2003. [Online]. Available: papers2://publication/uuid/491F0F0A-5B39-4B26-BF18-611A76CF0FDE

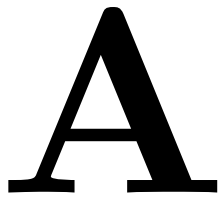
[45] X. Wang, S. Mou, and D. Sun, "Improvement of a Distributed Algorithm for Solving Linear Equations," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 4, pp. 3113–3117, 2017.

[46] G. Zhang and R. Heusdens, "Distributed Optimization Using the Primal-Dual Method of Multipliers," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 1, pp. 173–187, mar 2018. [Online]. Available: http://ieeexplore.ieee.org/document/7859410/

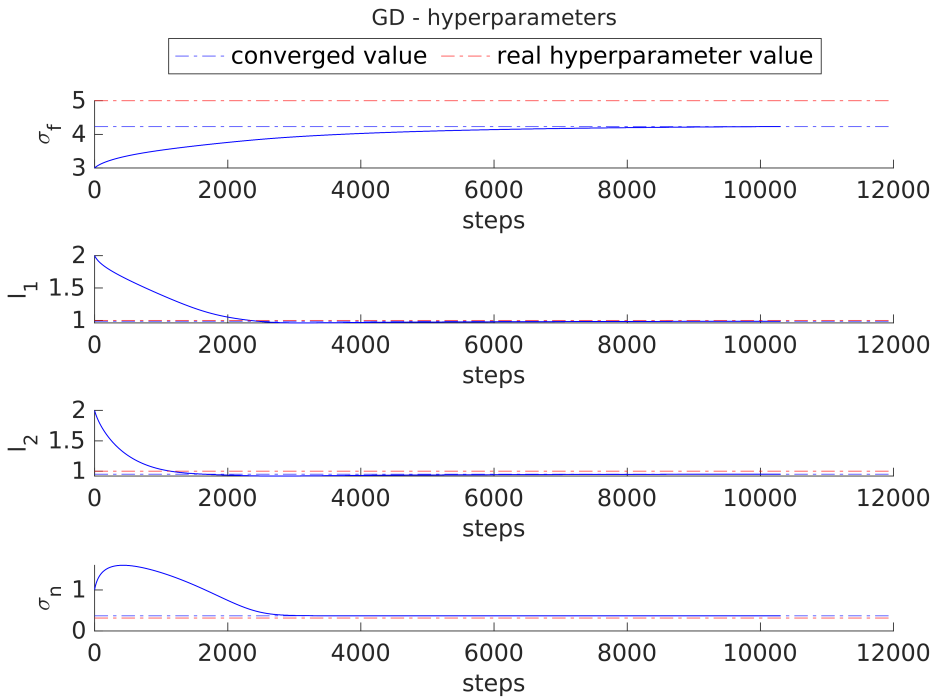
[47] T. Sherson, R. Heusdens, and W. B. Kleijn, "On the relationship between pdmm and a distributed admm variant," in *2017 Symposium on Information Theory and Signal Processing in the Benelux*, p. 201.

[48] M. K. Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," *JMLR Workshop and Conference Proceedings*, vol. 5, pp. 567–574, 2009.

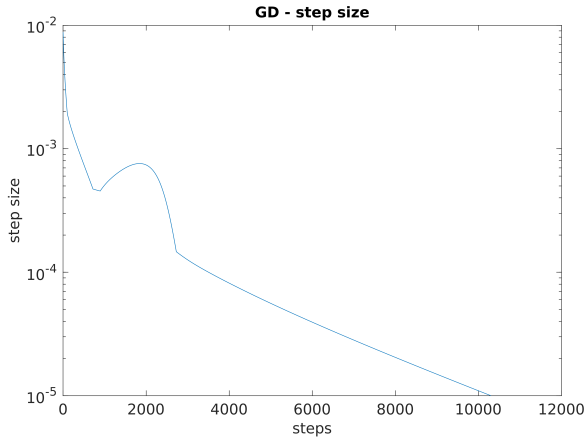
Appendix



A.1 Figures of Hyperparameter Optimization

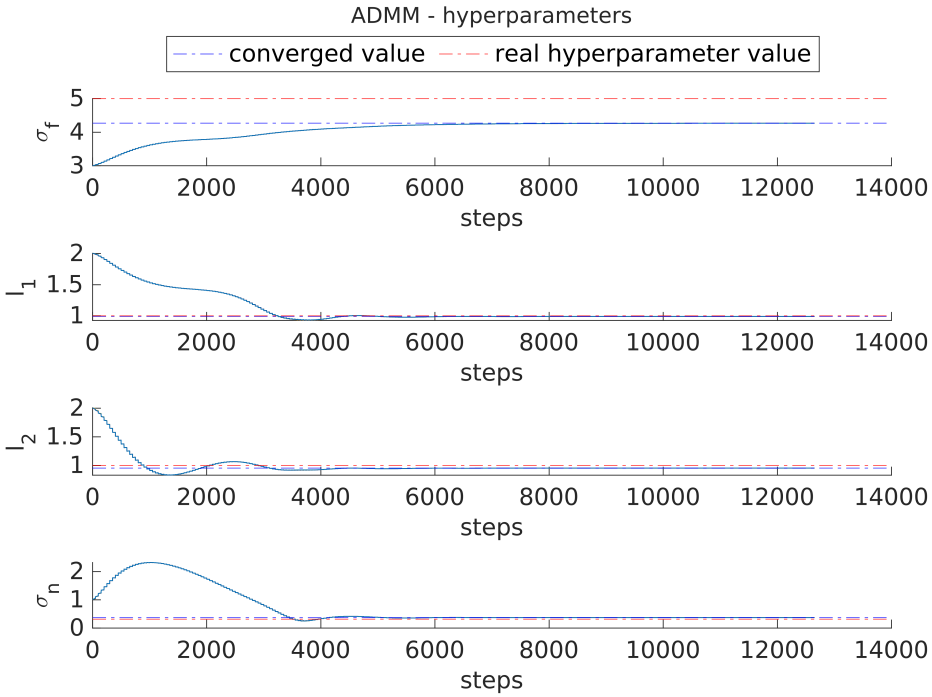


(a) hyperparameters

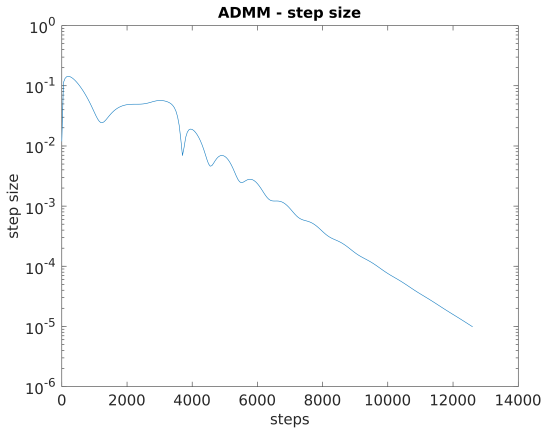


(b) step size

Figure A.1: Change of hyperparameters and step size in term of iteration number for nGD

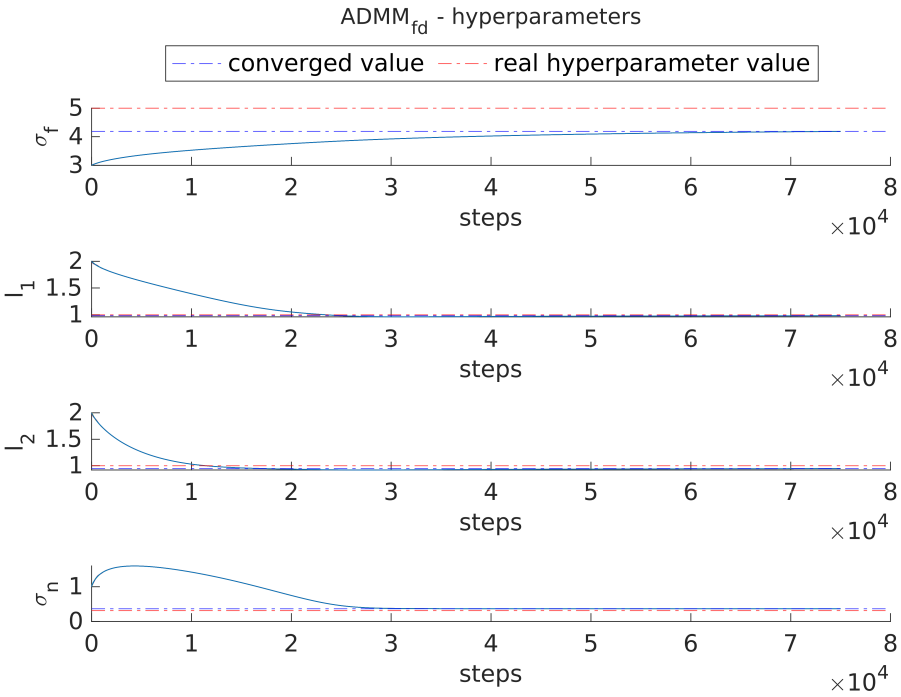


(a) hyperparameters

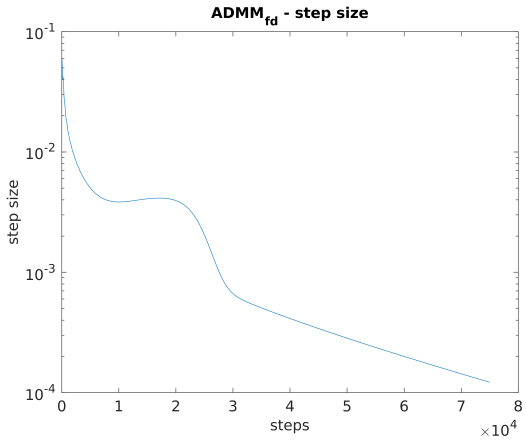


(b) step size

Figure A.2: Change of hyperparameters and step size in term of iteration number for centralized ADMM

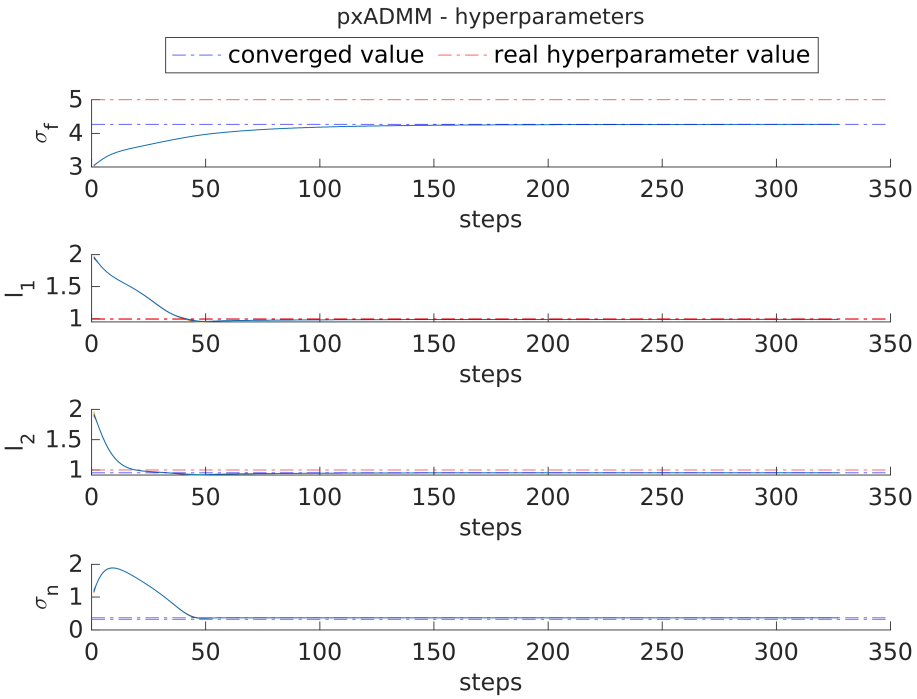


(a) hyperparameters

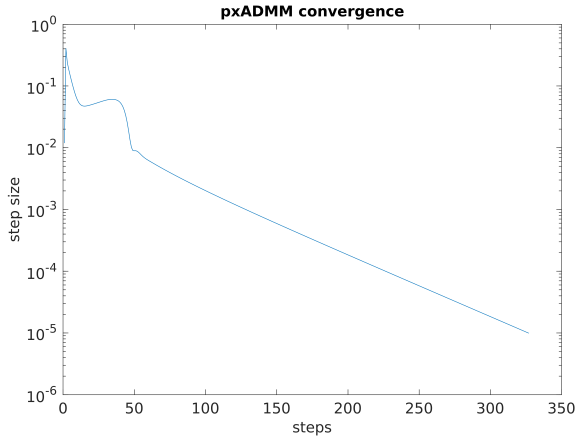


(b) step size

Figure A.3: Change of hyperparameters and step size in term of iteration number for fully-distributed ADMM

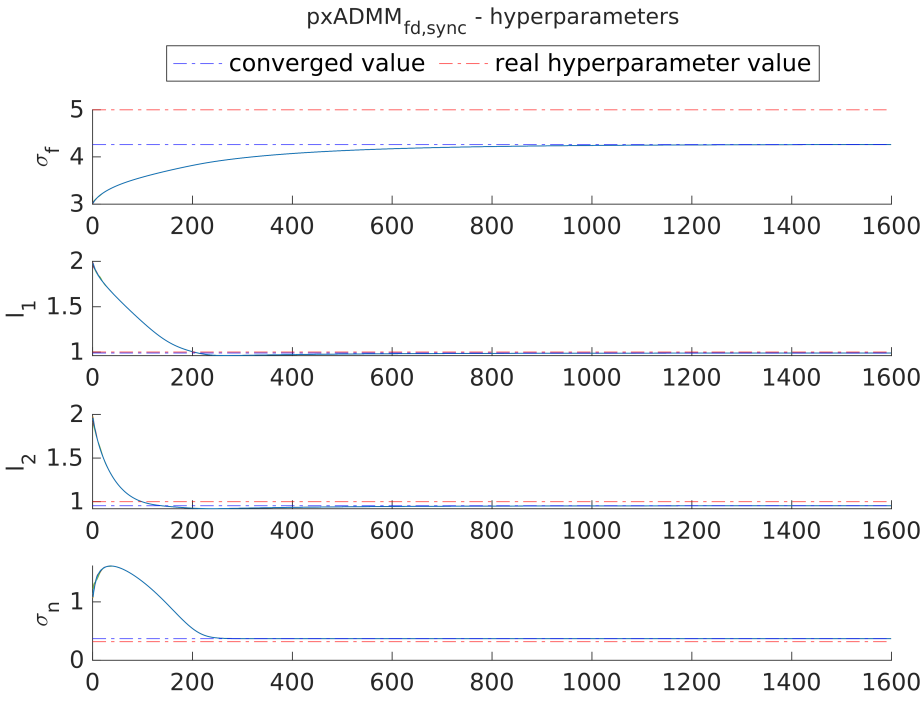


(a) hyperparameters

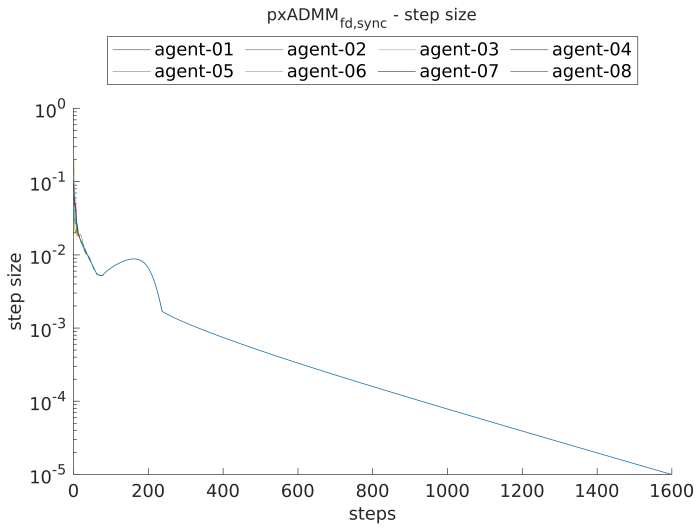


(b) step size

Figure A.4: Change of hyperparameters and step size in terms of iteration number for centralized pxADMM

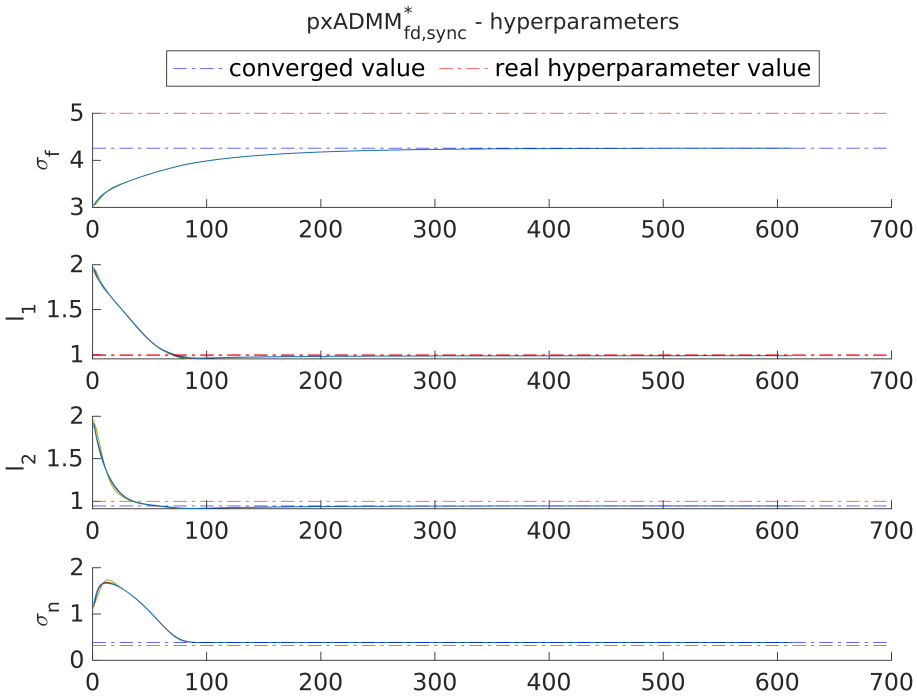


(a) hyperparameters

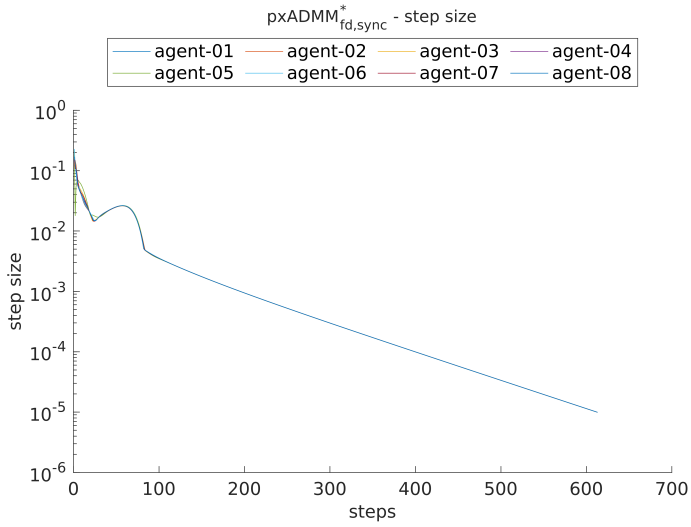


(b) step size

Figure A.5: Change of hyperparameters and step size in terms of iteration number for $\text{pxADMM}_{\text{fd}}$

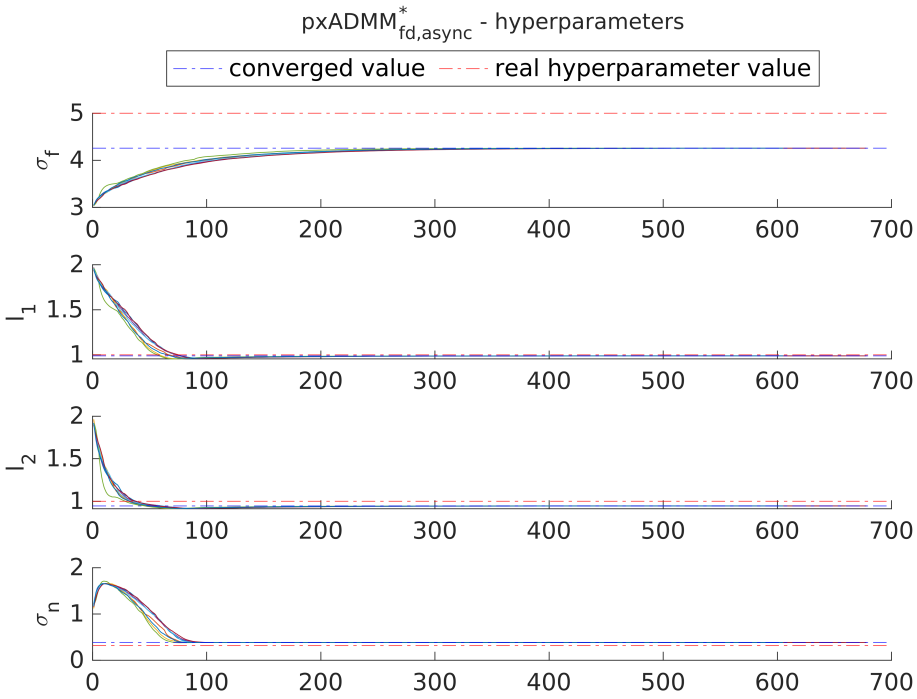


(a) hyperparameters

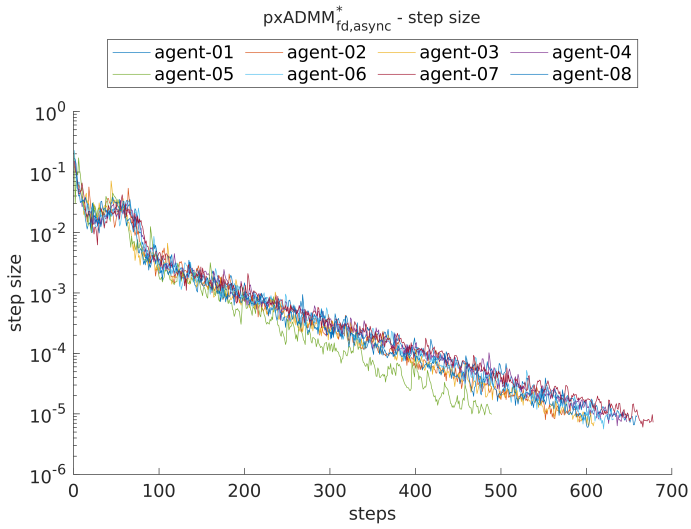


(b) step size

Figure A.6: Change of hyperparameters and step size in terms of iteration number for pxADMM$^*_{\text{fd}}$

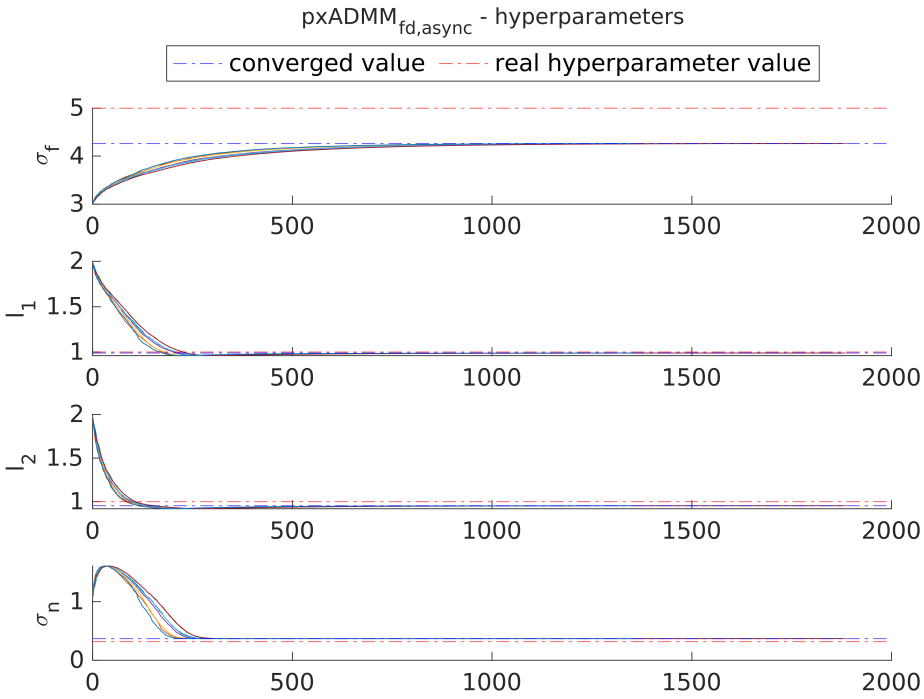


(a) hyperparameters

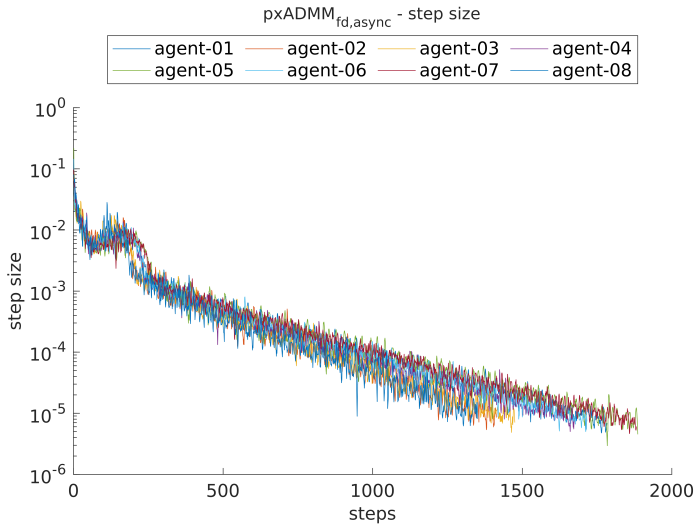


(b) step size

Figure A.7: Change of hyperparameters and step size in terms of iteration number for $\text{pxADMM}_{\text{async}}$

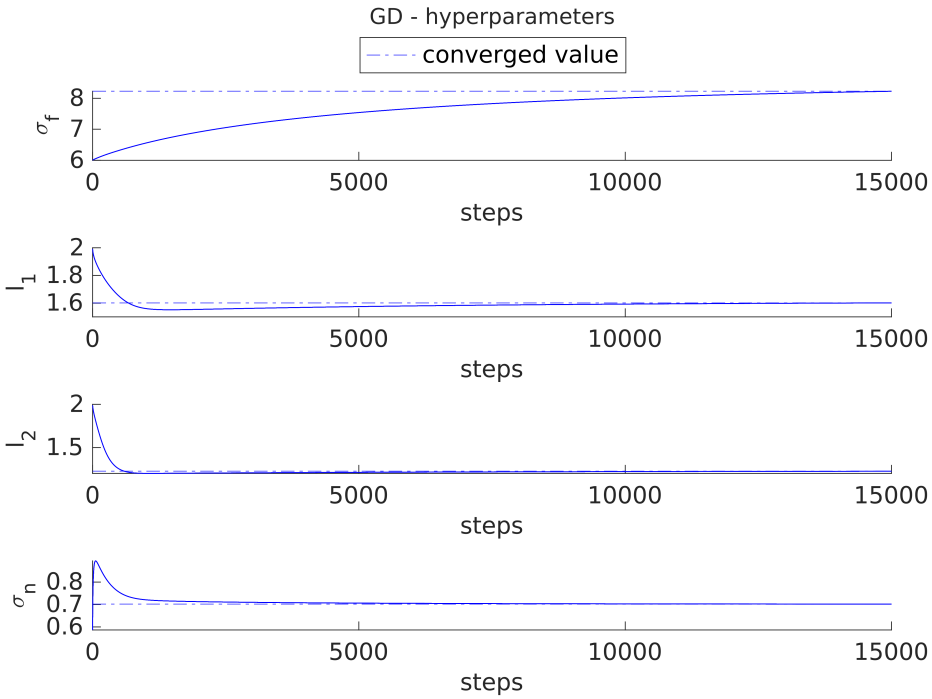


(a) hyperparameters

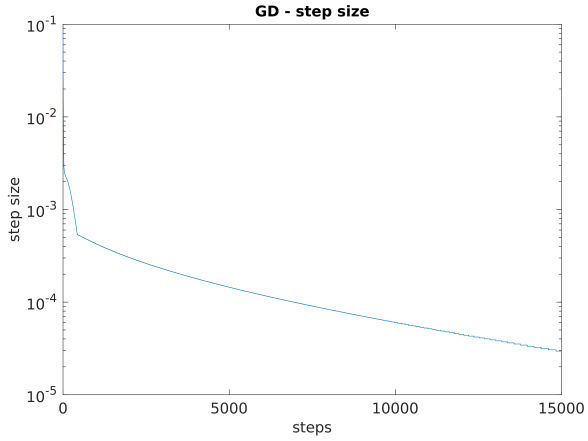


(b) step size

Figure A.8: Change of hyperparameters and step size in terms of iteration number for pxADMM$^{*}_{\text{async}}$

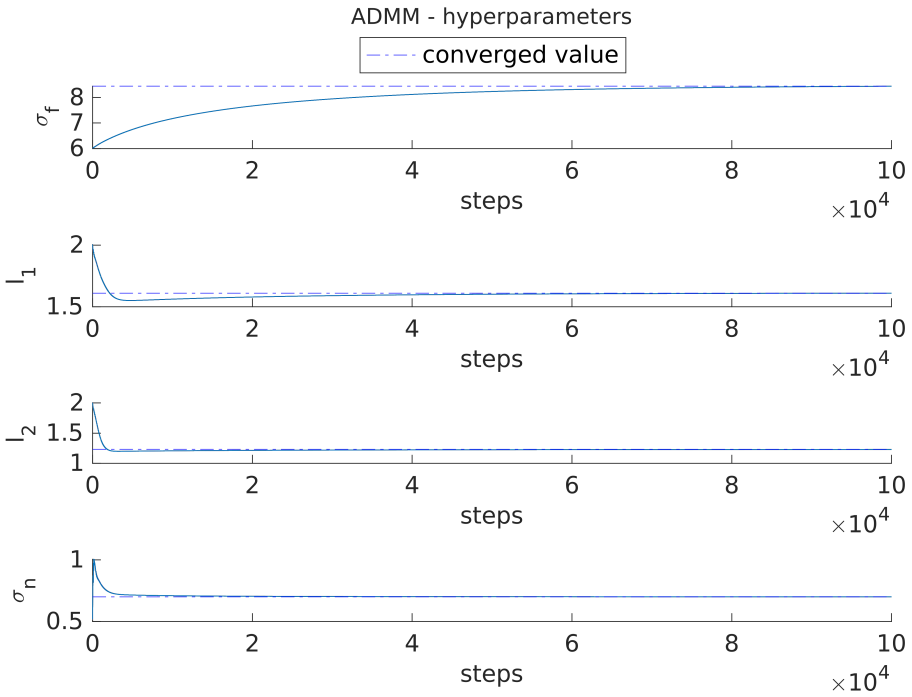


(a) hyperparameters

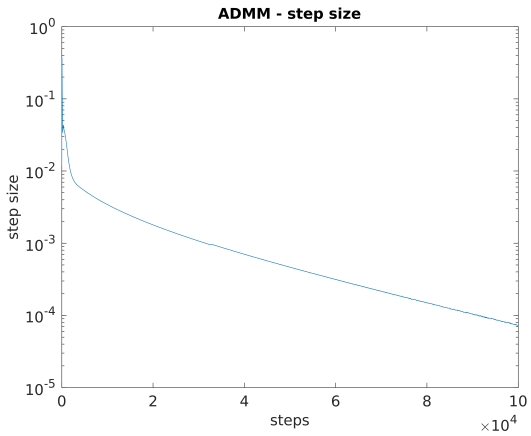


(b) step size

Figure A.9: Change of hyperparameters and step size in term of iteration number for nGD

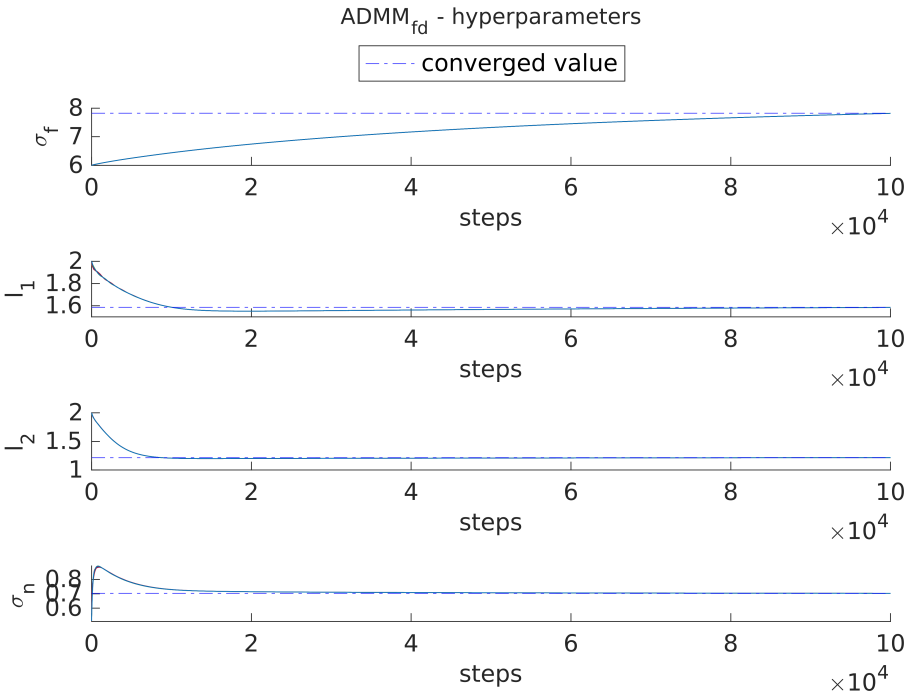


(a) hyperparameters

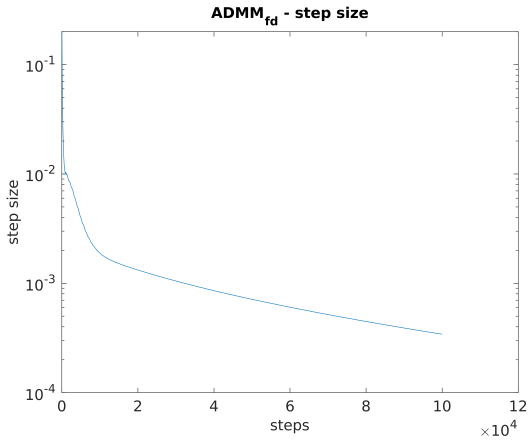


(b) step size

Figure A.10: Change of hyperparameters and step size in term of iteration number for centralized ADMM

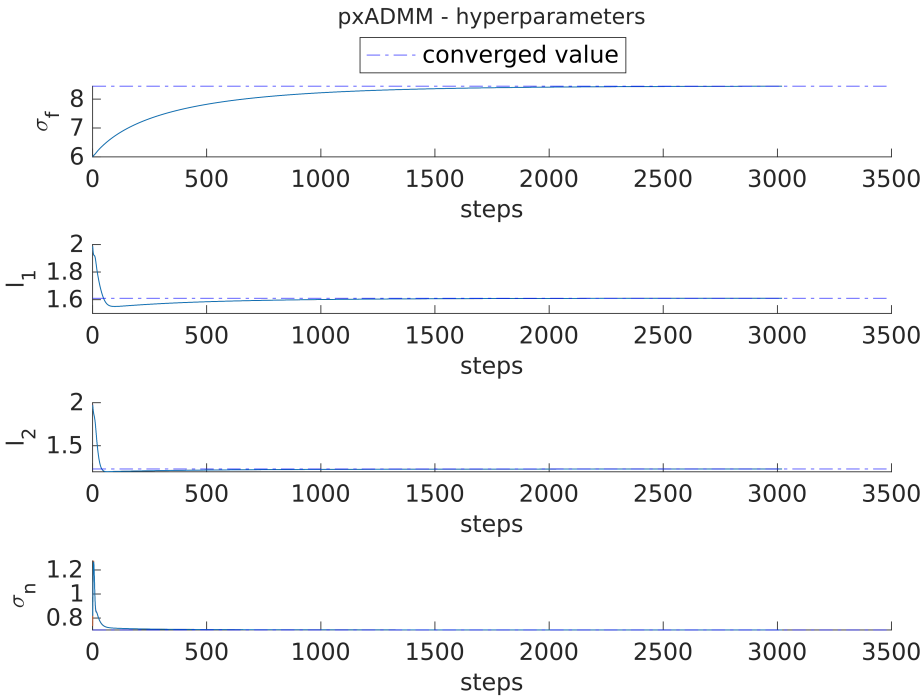


(a) hyperparameters

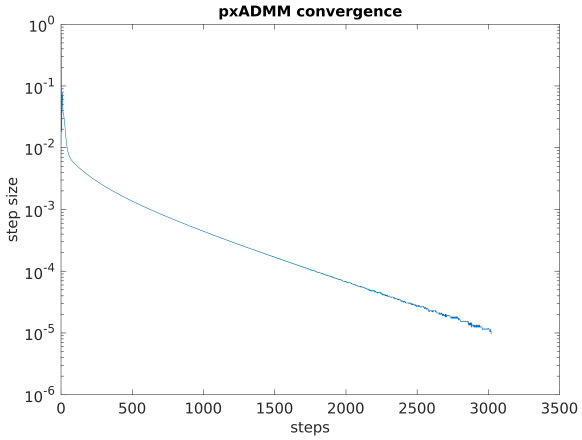


(b) step size

Figure A.11: Change of hyperparameters and step size in term of iteration number for fully-distributed ADMM

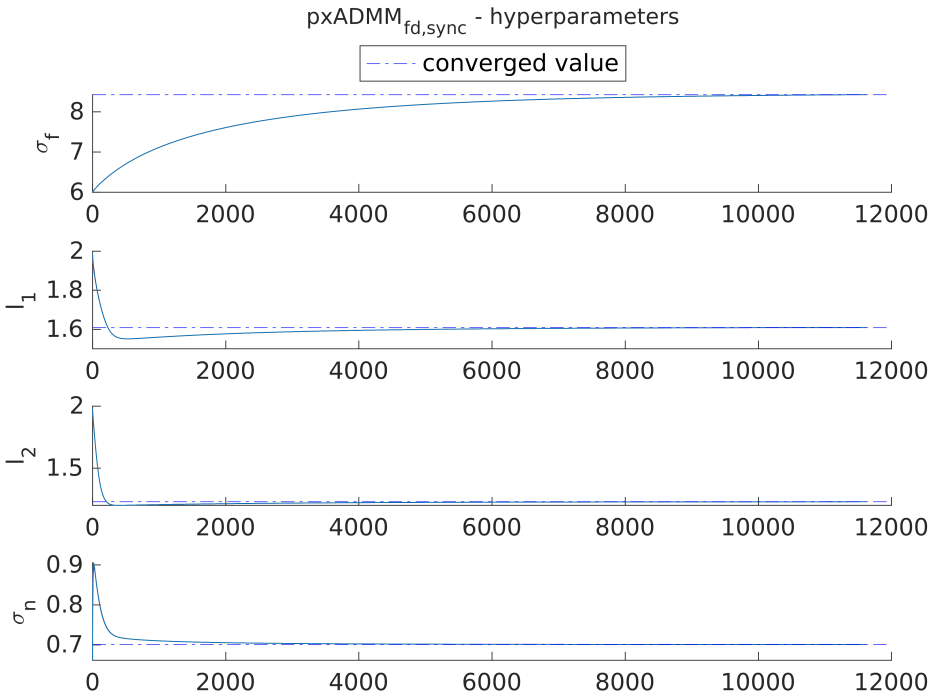


(a) hyperparameters

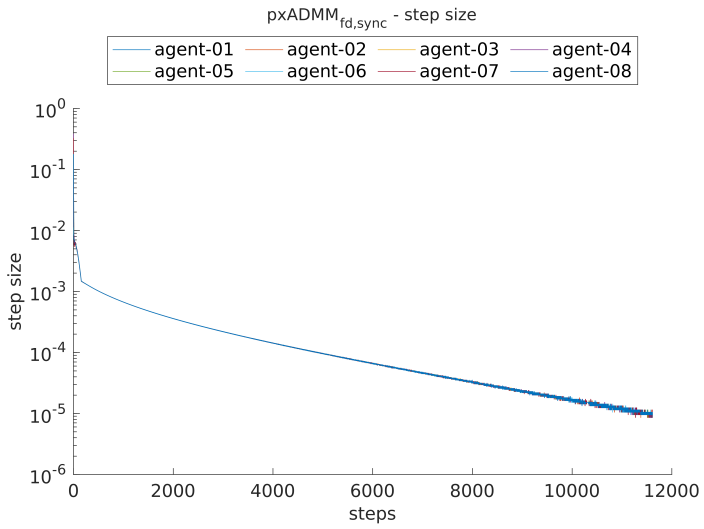


(b) step size

Figure A.12: Change of hyperparameters and step size in terms of iteration number for centralized pxADMM

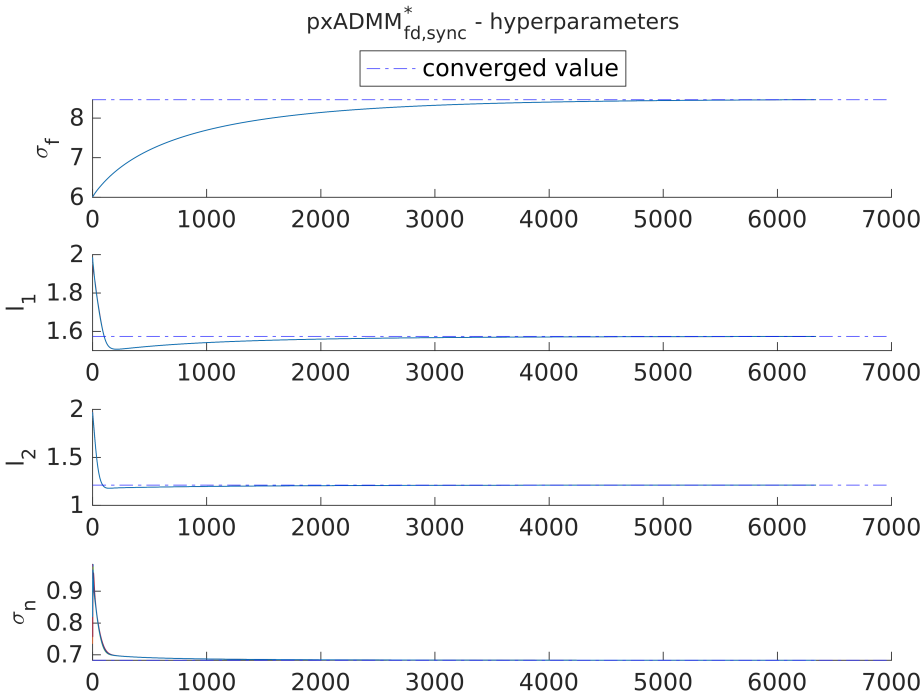


(a) hyperparameters

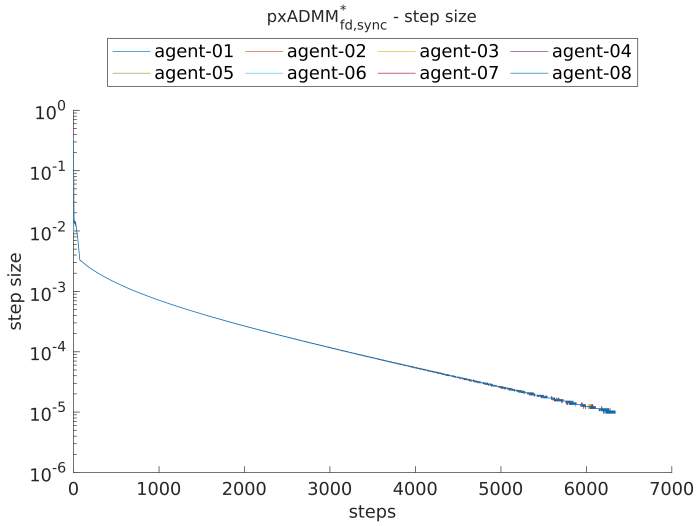


(b) step size

Figure A.13: Change of hyperparameters and step size in terms of iteration number for $\text{pxADMM}_{\text{fd}}$

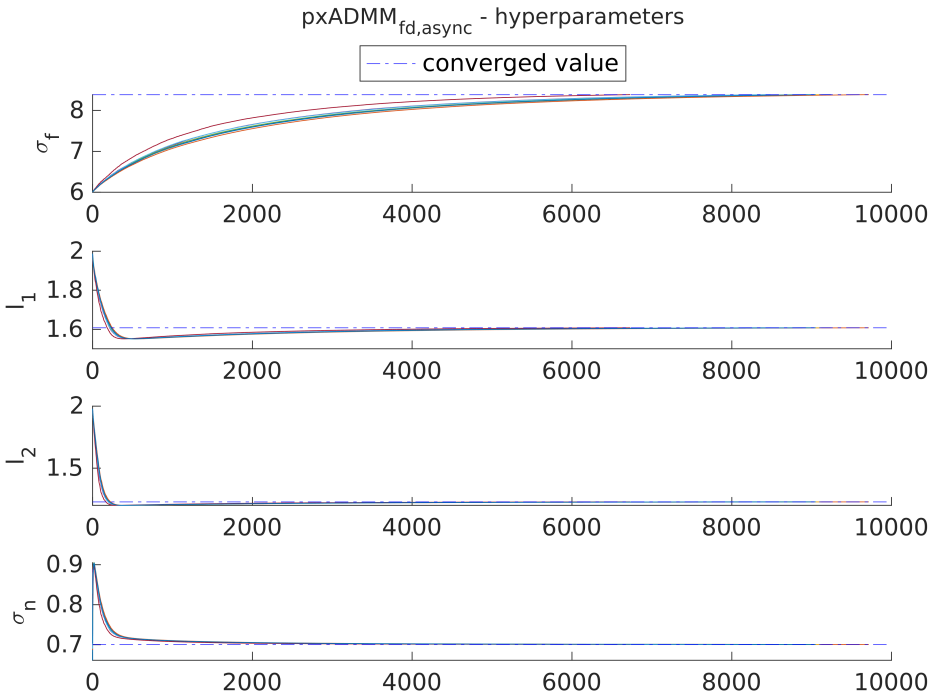


(a) hyperparameters

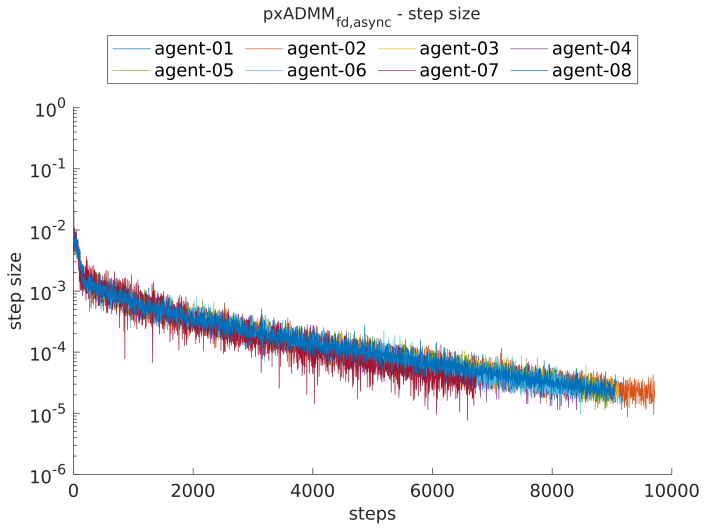


(b) step size

Figure A.14: Change of hyperparameters and step size in terms of iteration number for pxADMM$^*_{\text{fd}}$

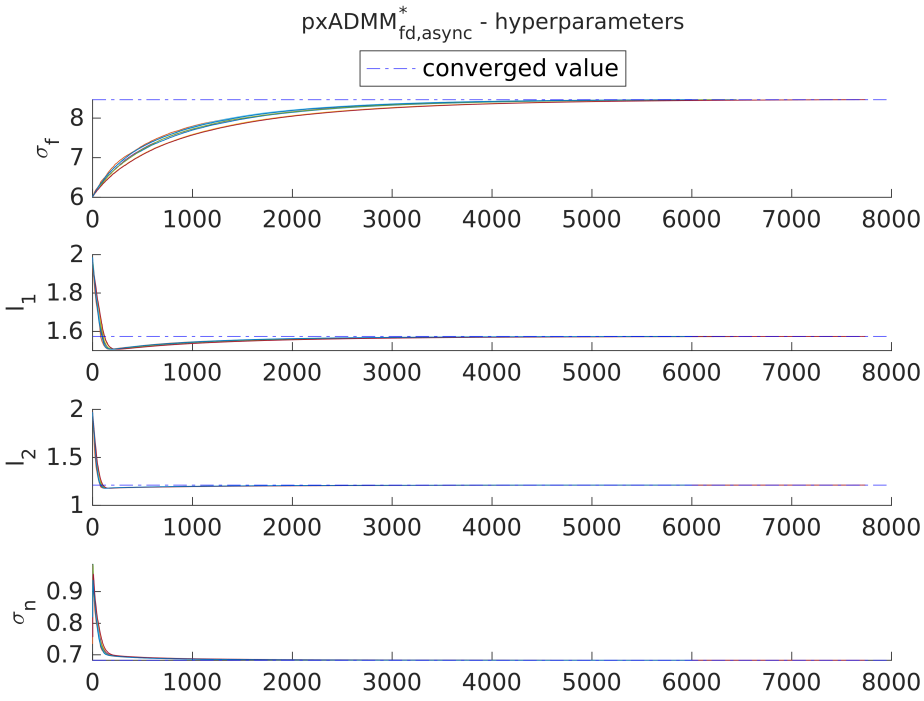


(a) hyperparameters

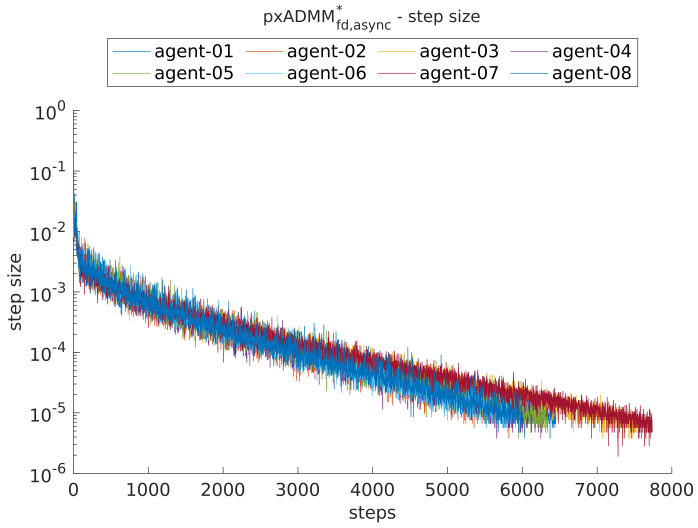


(b) step size

Figure A.15: Change of hyperparameters and step size in terms of iteration number for $\text{pxADMM}_{\text{async}}$



(a) hyperparameters



(b) step size

Figure A.16: Change of hyperparameters and step size in terms of iteration number for $\text{pxADMM}^*_{\text{async}}$

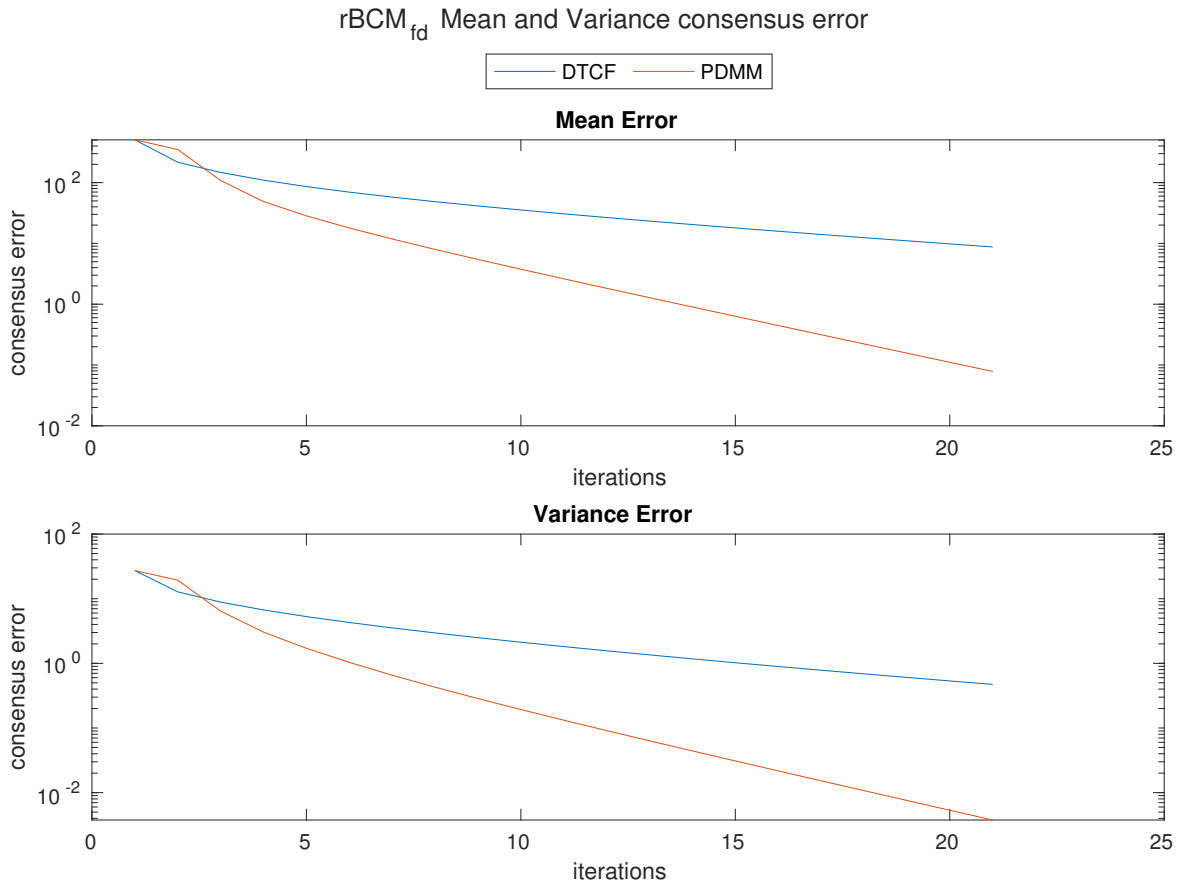


Figure A.17: The figure shows an example of the consensus error - iteration curves of PDMM and DTCF methods. It can be found that the PDMM converges faster than the DTCF methods in terms of the consensus error.

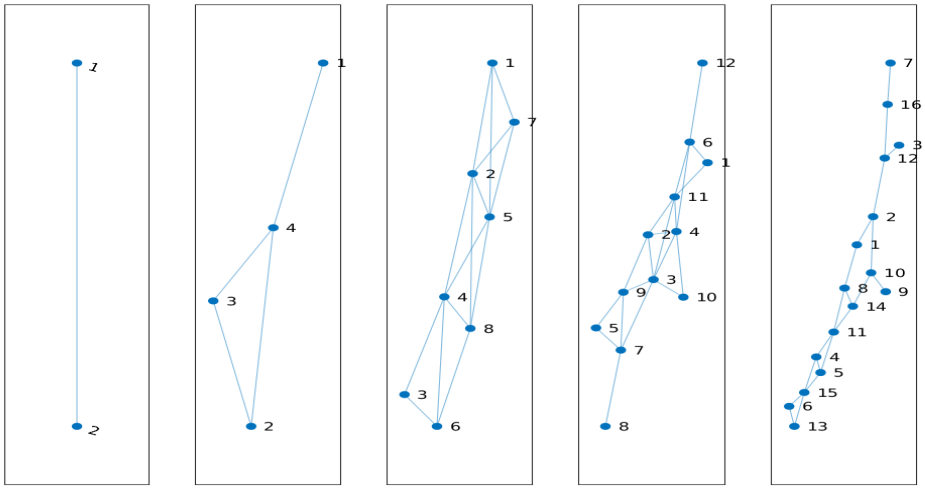


Figure A.18: Topology of MAS under different number of agents in artificial dataset simulation. The numbers of agents from left to right are respectively 2, 4, 8, 12 and 16. It can be found that the topology of network with 16 agents looks less connected than the other.

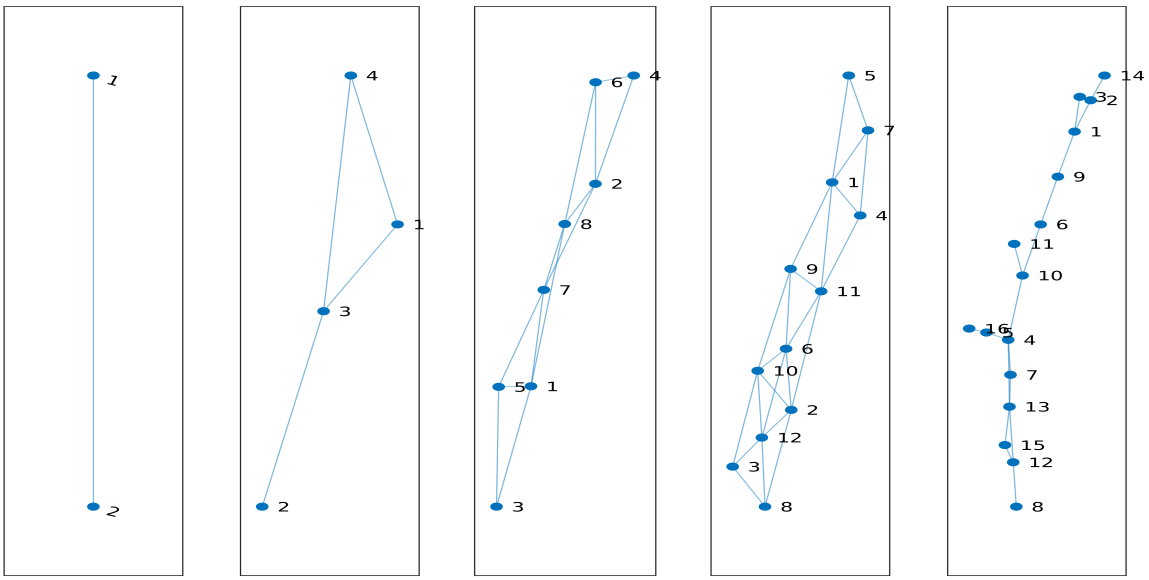


Figure A.19: Topology of MAS under different number of agents in real dataset simulation. The numbers of agents from left to right are respectively 2, 4, 8, 12 and 16. It can be found that the topology of network with 16 agents looks like a tree structure, which is a less connected structure than the first 4 graphs on the left.