Thesis for the Degree of Doctor of Philosophy

PERFORMANCE ORIENTED PREDICTION BASED ON IMPROVED GAUSSIAN PROCESS REGRESSION

Bui Dinh Mao

Department of Computer Science and Engineering
Graduate School
Kyung Hee University
Republic of Korea

February 2018

PERFORMANCE ORIENTED PREDICTION BASED ON IMPROVED GAUSSIAN PROCESS REGRESSION

Bui Dinh Mao

Department of Computer Science and Engineering
Graduate School
Kyung Hee University
Republic of Korea

February 2018

Performance oriented prediction based on improved Gaussian process regression

by

Bui Dinh Mao

Supervised by

Professor Sungyoung Lee

Submitted to the Department of Computer Science and Engineering and the Faculty of Graduate School of Kyung Hee University in partial fulfillment of the requirements of the degree of Doctor of Philosophy

Dissertation Committee:

Professor Eui-Nam Huh

Professor Tae-Seong Kim

Professor Jeong-Gil Ko

Professor Sungyoung Lee

Professor Young-Koo Lee

Firstly, I would like to acknowledge my advisor, Professor Sungyoung Lee, for his patient advice and guidance over years.

Secondly, I would like to thank my dear family, especially my parents, my wife and my little daughter, for standing by me in the darkest hours.

Abstract

In many years, Gaussian process was popularly utilized in many research areas such as signal processing, data communications and image processing, etc. The main benefit of Gaussian process is to accurately model characteristics of the target system by engaging probabilistic methodology. Unlike other techniques which try to determine all of the parameters of system model, Gaussian process adapts these parameters to reflect the actual underlying model. Because of that, this approach can be explicitly addressed as a non-parametric methodology. Moreover, as a comparison to other well-known methods, Gaussian process regression (GPR) possesses much better performance in terms of precision and versatility. Due to these benefits, GPR should be considered as a promising solution for performing the prediction. However, this technique does have a critical drawback, which is the limitation in processing rate. Theoretically, most of the complexity of GPR focus on the expensive computation ($O(n^3)$) and the corresponding storage space ($O(n^2)$) when dealing with n data points. The majority of burden mostly originates from inverting the covariance matrix and computing the log determinant. Obviously, when the data grows up, the problem becomes even more serious. This problem indeed degrades the performance in term of processing speed, and consequently deteriorates the overall efficiency of prediction.

In order to solve the aforementioned limitation, a number of research have been done over these years. Some of them focus on applying mathematical methods to reduce the complexity. Others spend their efforts on improving optimization techniques. However, these methodologies encounter the reliability issue as well as the comprehensiveness. Also, a limited theoretical improvement, which has been achieved, is still unsatisfied to solve the realistic problems. In this thesis, with the purpose of significantly enhancing Gaussian process regression, we would like to propose a performance-oriented solution for prediction based on Gaussian process. Thoroughly,

the proposed methodology solves the complexity problem in GPR by incorporating the latest mathematical modeling, optimization and parallel computing together. As a result, the final solution is able to drive the complexity to an affordable expense while still preserves an acceptable accuracy. In order to verify the proposed idea, some experiments and potential applications are also introduced in the performance evaluation to prove the effectiveness of the solution.

Table of Contents

Abstract i

Table of Contents iii

List of Figures vi

List of Tables viii

Chapter 1 Introduction 1

1.1 Research domain 1

1.2 Motivation 3

1.3 Problem statement 4

1.4 Key contributions 5

1.4.1 Analysis of Gaussian process regression 5

1.4.2 Complexity reduction in hyper-parameters learning phase 5

1.4.3 Introduction of 'divide and conquer' idea in training phase 6

1.5 Thesis organization 6

Chapter 2 Related works 8

Chapter 3 Domain analysis and potential problems 11

3.1 Domain analysis 11

3.1.1 Characteristics of data 11

3.1.2 Spectral representation 14

3.1.3 Complexity of modeling 16

3.2 Potential problems 21
3.2.1 Regression in communications 21
3.2.2 Small-scale problem 22
3.2.3 Large-scale problem 24

Chapter 4 Gaussian process regression **30**
4.1 Bayesian learning 30
4.1.1 Probabilistic background 30
4.1.2 Learning procedure 35
4.2 Gaussian process regression 43
4.2.1 Original form 43
4.2.2 Practical form 45
4.2.3 Kernel functions 46
4.2.4 Mean function 57
4.2.5 Prediction 58

Chapter 5 Improved Gaussian process regression **60**
5.1 Hyper-parameters learning phase 60
5.1.1 Finding hyper-parameters 60
5.1.2 Convergence law of log determinant 62
5.1.3 Properties of convergence law 64
5.1.4 Technique of improvement 65
5.2 Training phase 78
5.2.1 Solving linear system 78
5.2.2 Divide stage 80
5.2.3 Conquer stage 82
5.2.4 Parallelism engagement 84

Chapter 6 Performance evaluation **87**
6.1 Bench-marking experiments 87
6.1.1 Empirical experiment 87

6.1.2 Spectral experiment 91
6.2 Potential applications 95
6.2.1 Regression in communications 95
6.2.2 Small-scale prediction 99
6.2.3 Large-scale prediction 105

Chapter 7 Conclusions and future work **113**
7.1 Conclusions 113
7.2 Future work 114

Bibliography **115**

A List of publications **124**
A.1 First author 124
A.1.1 SCI journals 124
A.1.2 Conferences 125
A.1.3 Patents 126
A.2 Co-author 126
A.2.1 Journals 126
A.2.2 Conferences 126

List of Figures

3.1 Abstraction of processing tasks based on Markov model. 12
3.2 Example of CPU monitoring statistics in time domain. 15
3.3 Equivalent statistics in frequency domain. 16
3.4 Corresponding power spectral density of statistics. 17
3.5 Model selection and corresponding complexity by marginal likelihood. 18
3.6 Synchronous DS-CDMA system. 21
3.7 The work flow of small scale prediction in CPU multicore scheduler. 23
3.8 A typical utilization of virtualized computing cluster. 25
3.9 Architecture of energy efficiency management (E2M) system. 27
3.10 Flowchart of energy efficiency management (E2M) system. 29

4.1 Bayesian networks including Q as defined in (4.18). 37
4.2 Bayesian network for repeated trials X_i. 40
4.3 Bayesian network for an original GP. 44

5.1 Empirical data produced by the complex underlying combined functions. 69
5.2 Power spectral density of complex empirical data. 70
5.3 Logic of hyper-parameters learning phase. 76
5.4 Direction of calculating 'local' expansion step **S2W**. 83
5.5 MapReduce-oriented implementation of parallel Fast Gauss Transform. 84

6.1 Accuracy and processing speed evaluation. 88
6.2 Empirical data produced by the complex underlying combined functions. 92

6.3 Prediction based on empirical data. 93
6.4 Estimated power spectral density of complex empirical data. 94
6.5 Accuracy in the relationship with the size of dataset. 96
6.6 Time complexity enhancement of the proposed method. 97
6.7 Bit-error-rate at each signal-to-noise ratio. 98
6.8 Processing rate between two systems (lower is better). 100
6.9 Power evaluation on proposed method over one hour running time. 101
6.10 Computational speed evaluation on each phase of prediction (lower is better). . . 102
6.11 Overall processing rate of prediction techniques (lower is better). 103
6.12 Prediction result including mean and variance of proposed method. 104
6.13 Evaluating active physical servers on Google traces. 107
6.14 Evaluating power consumption on Google traces (lower is better). 108
6.15 Power consumption vs average latency. 109
6.16 Prediction in memory utilization benchmark. 110
6.17 Prediction evaluation of the proposed method in Google traces experiment. . . . 111

List of Tables

5.1 Computation cost of proposed method compared with others 61
5.2 Transformation of objective function 75

6.1 Google cluster's organization and configuration 105
6.2 Summary of Google traces' characteristics 105
6.3 Equivalent energy consumption exchanging scheme 107

Chapter 1

Introduction

1.1 Research domain

In most of computing system, decision making process is considered to be one of the most critical parts, which directly affects performance, reliability and effectiveness of the system. In fact, before making any decision, the related information should be collected in advance. After that, an optimization step is usually performed to produce a suitable decision with regards to some predefined objective functions. If the delay occurs in the step of collecting data, the final decision might be degraded. Due to this reason, prediction are often incorporated into decision making process to overcome this obstacle. With the help of prediction, the usefulness of the input information is enhanced, which subsequently improves the accuracy of making decision. Traditionally, the prediction is mostly done by using regression techniques, which analyzes the underlying relationships between the events that came out in the previous moments and the latest one. The acquired relationships are then engaged to estimate possible value of potential event, which may happen in the next moment. Indeed, when the regression techniques are well built and suitably reflected the aforementioned relationships, the futuristic information can be precisely anticipated with regard to the real data. Statistically, there are a number of regression techniques. Depending on the prior knowledge of the investigated relationships, one regression technique might perform better than others. This prior knowledge is usually encapsulated into parameters of the regression function. It is worth noting that the more detail the parameters are described, the better result the regression function can produce. However, in real world, there are very rare situation when the parameters are well defined in advance due to the lack of knowledge on every factors which can affect the regression model. Regularly, only the statistics of the past events exist as the input data. Because of this fact, there is a critical need for a regression technique that can adapt itself

to the provided data. This kind of technique is categorized as non-parametric regression. Briefly, non-parametric regression does not predetermine the member parameters. These parameters are actually constructed with regard to the information retrieved from the statistical data. In this thesis, one non-parametric regression technique, namely Gaussian process regression, is investigated and improved to cope with the requirement of performance oriented prediction.

Additionally, there is an important reason to choose the Gaussian process regression the prediction method in distributed systems. That reason is the high accuracy of this technique among others [1]. In comparison to many well known regression techniques (linear regression, k-nearest neighbor, multivariate adaptive regression splines, multilayer perceptron ensembles trained with early stopping, multilayer perceptron-Markov chain Monte Carlo), the regression from Gaussian process family can be considered as the most accuracy technique in both benchmarks for noise-free and noisy data. Due to this attractive feature, Gaussian process regression describes itself as a high potential technique to engage the prediction that enhances decision making in distributed system.

Based on the principles of Bayesian learning, Gaussian process has built up a useful probabilistic and non-parametric solution to work with complex data. In fact, GP methods effectively reflect the relationship among data points regarding the time manner. In other words, GP methods adapt the member parameters to describe the hidden function which can potentially interpolates predictive data. Basically, the predictive result of GP methods can be simplified into twofold. The first part is the mean value, which is considered as the most likely anticipated value at the target moment. The remaining part is the variance, which measures the confidence of prediction. Because of the flexibility and high capability of adaptation, GP is an appropriate solution for processing various kinds of data, even with noisy or corrupted data. Unfortunately, there is one remarkable drawback, namely the high complexity in terms of computation and storage, which prevents GP methods to be integrated into real world systems. Theoretically, modeling by using GP encounters high complexity for time ($O(n^3)$) and space ($O(n^2)$) when performing on n data points. With the size increment of the dataset, this problem even gets extremely unacceptable. Due to this uncomfortable reason, the applications based on GP methods are awfully limited. To overcome this issue, a novel methodology is proposed in this thesis to reduce the computational

complexity, while still maintains an analogously acceptable accuracy level. Hopefully, this significant improvement would help to innovate the development of solution based on GP.

1.2 Motivation

As mentioned previously, the target environment of the research is including but not limited to distributed computing systems. Nowadays, distributed systems are widely utilized due to their attractive features. Some of the features can be listed as flexible load balance, high performance, improved robustness as well as the capability to implement advance strategies to achieve better power consumption, or resource utilization. In order to enhance these features, usually a monitoring system such as Ganglia is used to provide the overall perspective. This kind of information is crucial to orchestrate the operation of the system, reconfigure the components, or apply the predefined scenarios. Due to this fact, any delay in collecting monitoring data may degrade the effectiveness of the strategy. Unfortunately, delay is the sensitive factor that intensively occurs in monitoring system. The main reason for this issue is that the monitoring system is unable to collect the statistics in real-time manner. Only periodic tasks are possible to retrieve the data from target computing objects. Obviously, this mechanism makes the desire data for decision making algorithms obsolete. Apparently, there is a critical need for fast data prediction over the big monitoring statistics to improve the usefulness of the data.

In order to conduct an appropriate prediction, it is mandatory to investigate the nature of monitoring data in the target computing system. Most of statistics in distributed system are related to arrival rate of tasks coming to computing nodes. These tasks can be any universal requests from clients to servers, or between servers themselves. For instance, a web request from any regular web browser to web server can be considered as an incoming task. When the web server prepares the result that would be sent to the client, it also needs to connect to another servers such as database servers, content servers, or Hadoop master node. This intra-connections inside web system are obviously taken into account as tasks. Theoretically, the distribution of incoming tasks follows the Poisson distribution. According to the relationship between probabilistic distributions, when arrival rate increases, the Poisson distribution might converges to the Gaussian distribution. It is worth mentioning that the Gaussian distribution over time establishes the Gaussian process.

Because of this fact, using Gaussian process regression (GPR) to conduct the prediction on the monitoring statistics is perfectly appropriate. Connecting to the claim in previous paragraph, the GPR needs to be fast enough to fulfill the requirement of processing big dataset.

Besides, due to the fact that GPR are popularly utilized in many research fields such as communications, image processing, signal processing, robotics, thermal control, atmospheric modeling, etc. Any enhancement to GPR might find the extreme importance to innovate the development of corresponding applications in many different research areas. Obviously, this one of the reasons that strongly motivates the research to propose a significant advancement for Gaussian process regression.

1.3 Problem statement

Gaussian processes are taken into account as powerful methodologies to conduct non-parametric prediction. Unfortunately, there is an unavoidable issue that limits the capability of this regression family. This problem is known as the high complexity issue. Due to this problem, the processing speed of Gaussian processes is deteriorated hundreds of times in comparison to other techniques [1], especially when the size of dataset increases. Basically, the complexity comes from both computation and storage, which costs $O(n^3)$ and $O(n^2)$ [2], respectively. The reason for this drawback is twofold:

- The first complexity comes from covariance matrix inverse. In mathematics, matrix inverse is always one of the most expensive tasks. For example, assume that there is a square matrix $n \times n$. At the beginning, when the first row has length n, it takes n operations to zero out any entry in the first column (one division, and $n - 1$ multiply-subtracts to find the new entries along the row containing that entry. To get the first column of zeroes therefore takes $n(n - 1)$ operations. In the next column, we need $(n - 1)(n - 2)$ operations to get the second column zeroed out. In the third column, we need $(n - 2)(n - 3)$ operations and so on. The sum of all of these operations eventually goes to $O(n^3)$.

- The remaining complexity is related to calculate the log determinant for the covariance matrix. This task is mandatory for the optimization steps in finding the appropriate value of

the hyper-parameters. Obviously, this complexity is also inescapable.

The above drawback exists in both hyper-parameters learning phase as well as training phase of Gaussian process regression. However, the discrepancies in operation of these two phases leads to the divergence in how we propose the corresponding solutions to each phase. Clearly, solving the aforementioned obstacles is an interesting objective. Any success in reducing the complexity can either innovate the development of useful applications based on Gaussian process regression, or improve the current Gaussian processes family approaches.

1.4 Key contributions

The main contributions of this thesis is described as below:

1.4.1 Analysis of Gaussian process regression

In this thesis, it is worth noting that the Gaussian process regression (GPR) is targeted to solve the problem of prediction with an enhancement in performance, especially to improve the monitoring statistics of distributed system. Due to this reason, it is more productive to describe GPR as a predictive component inside a decision making application. In other words, GPR component is engaged to enhance the input, which is the statistical information. This predictive input is subsequently send to the application to create appropriate decisions with regard to any predetermined objectives. Moreover, it is worth mentioning that: although the proposed GPR is designed to work with statistics in computer system, it can also be slightly modified to apply to other areas such as signal processing or communications,...etc without any problem. In order to effectively build an appropriate GPR model based on prior knowledge of the system, an adequate analysis of the domain and the GPR itself is compulsory.

1.4.2 Complexity reduction in hyper-parameters learning phase

In this part of contribution, the speed of hyper-parameters learning phase is improved by proposing a complexity reduction method. In nature, the hyper-parameters are used to define the covariance functions or kernel functions, which represent the characteristics of the target uncertainty model.

These kernel functions again define the covariance matrix, which is a square matrix that indicates the amplitude of correlations between entries of dataset. In order to interfere the predicted data in training phase, the hyper-parameters are needed to be computed first. As stated previously, finding hyper-parameters related to inverting the covariance matrix and calculating the log-determinant, which is very expensive. To solve this burden, an suitable cooperation of fast Fourier transform, convergence law of log determinant and stochastic gradient descent is proposed to significantly cut down the processing time.

1.4.3 Introduction of 'divide and conquer' idea in training phase

After having the hyper-parameters, the prediction process still needs to perform training phase to retrieve the desired predictive output. Unfortunately, this process is also related to matrix-vector product, which expenses $O(n^2)$ for computational complexity. To reduce this cost, an idea of "divide and conquer" approach is attached with parallelism to directly mitigate the training time.

1.5 Thesis organization

The dissertation is organized into chapters as follows:

- **Chapter 1: Introduction**. Chapter 1 briefly introduces the research work for Gaussian process prediction in dealing with large dataset. It is worth noting the potential applying environment can be but not limited to the distributed computing systems. This chapter also presents the problems of Gaussian process regression, the goals to solve these problems, and finally the objectives achieved in this research work.

- **Chapter 2: Related works**. Some interesting research works, which are related to the proposed method, would be introduced. In addition, the further discuss and significance comparisons on pros and cons of each related work are also included in this chapter.

- **Chapter 3: Domain analysis and potential problems**. This chapter provides the detail of the research domain, the characteristic of the target system, and the complexity analysis when applying the regression. Besides, some potential problems, which can be applied the

idea of engaging GPR to improve the predictive system statistics, are also discussed. The content of discussion mostly focuses the objectives, the possible designs and the internal working mechanisms of corresponding solutions. It is worth noting that the role of this part is to explain how GPR is actually utilized in action.

- **Chapter 4: Gaussian process regression**. In this chapter, we present a fundamental foundation of how to perform Gaussian process prediction under the Bayesian inference approach. Primarily, the relationship between probability theory, Bayesian learning and Gaussian process would be investigated. Based on this background, the formulation of Gaussian process regression is shown in detail. Also, the correlations between the covariance functions, covariance matrix and hyper-parameters are explained.

- **Chapter 5: Improved Gaussian process regression**. This chapter presents the technical details of proposed methodology to perform the complexity reduction for Gaussian process regression. Basically, this process comprises the improvements to two major phases of GPR, namely the hyper-parameters learning phase and training phase. As stated previously, different kinds of techniques are formulated corresponding to the objectives in each phase.

- **Chapter 6: Performance evaluation**. In this chapters, two kinds of experiments are conducted to evaluate the performance of proposed method in term of accuracy, computation time and effectiveness. While the first part of experiments measures the regular metrics as well as shows the relationship of regression between time and frequency domains; the potential applications in the second part are used to demonstrate how the improved GPR actually works in real-life situations.

- **Chapter 7: Conclusions and future work**. This chapter concludes the thesis and also provides future directions in this research area.

Chapter 2

Related works

For a long time, a huge number of applications have implemented Gaussian process as a successful solution for many harsh problems. In supervised learning area, many famous applications using this method can be classified into classification and regression, especially for dealing with time-series event. Improving the effectiveness of Gaussian process is also synonymous with improving the corresponding applications. Because of this fact, many effort has been made to clarify the role and the relationship between internal components such as covariance matrix, mean function, kernel functions and hyper-parameters. Based on the extracted knowledge, some interesting enhancements have been proposed to tackle the complexity issue of Gaussian process. These research works are summarized as follows.

In order to solve the problem of disordered time series prediction, an advance GPR [3] has been suggested for modeling purpose. In the center of this approach, the knowledge, which is acquired continuously, is gradually used to reconcile the hyper-parameters to accommodate the prediction. In the other words, GPR in online sparse fashion has been utilized to consecutively process the streaming data and adapt the hyper-parameters. In fact, doing research on extending the prediction to online processing area is innovating. However, the mentioned technique is not advanced enough to solve the problem of high complexity. Actually, the conjugate gradient (CG) and the Cholesky decomposition are coupled for solving the matrix inverse and optimizing the negative marginal log likelihood. In detail, CG is known to be a conventional optimization technique to solve linear system. Also, Cholesky decomposition is well-known to be a popular method to break down the matrix without inverting. Unfortunately, the computational complexity is only lessen a little bit to $O(n^3/6)$, which is not enough to improve the overall performance.

Discussing online hyper-parameters learning is fascinating. One other approach can be listed

is "Bayesian non-parametric adaptive control using Gaussian processes" [4]. In this research, the authors try to solve the complexity problem by proposing an advanced optimization method based on stochastic gradient ascent (SGA) technique. For more information, SGA is known to be empirically appropriate for online learning procedure. Besides, the irrelevant data-point elimination technique, namely the uncorrelated data cleaning mechanism, is introduced to shrink the dataset. Because of that, the computational complexity is dropped to $O(n^2)$ when dealing with n training points. This magnitude of complexity is quite potential in comparison to the above technique.

In order to handle large dataset, an interesting research based on GPR [5] has been conducted. The nature of this research is to use stochastic projection technique to compact the characteristics of data in term of dimension. For more information, the aforementioned technique is introduced to approximate the covariance matrix. Theoretically, the main idea is to reduce the rank of this matrix to simplify the computation. In fact, this idea really improves the matrix inverse. As a result, the computational complexity is dropped from $O(n^3)$ to $O(mn^2)$, where n is the size of dataset and m is the best approximation for the rank of corresponding covariance matrix. Although the result is not so impressive in comparison to other methods, the proposed method has shown the possibility in clustering and partially solving the large dataset, which can lead to a lot of further developments.

Another effort to process large dataset, namely "Gaussian processes for big data" [6], can be described as a good contribution to the field. In this research, the authors focused on proposing a specific Gaussian process model based on stochastic variational inference (SVI) method. The interesting point of SVI approach is the independent complexity regardless the dataset. By using this technique, the proposed method can perform on very large dataset and achieve an impressive performance. Unfortunately, as mentioned above, SVI technique only endeavors properly in a specific system, where the observational and latent variables can be globally factorized. With general systems, it would be a burden to add these kinds of variables. Moreover, due to the fact that this method is input-independent, when the size of dataset increases, the precision might be significantly degraded.

Some researchers are interested in solving the complexity problem in training phase only rather than dealing with the hyper-parameters learning phase. The first approach can be listed as "fast

Gaussian process regression using kd-trees" [7]. This approach initially builds a binary tree from the scratch. After that, the dataset is and recursively partitioned and assigned to corresponding leaves of the built tree. Besides, the cached information of weighted sum of each leaf is also used to enhance the calculation procedure. By engaging this idea, the proposed method can reduce the complexity of training phase to $O(nlogn)$, which can be seen as a significant improvement. Nonetheless, this improvement cannot apply to the hyper-parameters learning phase because of differences in objective functions. Due to this reason, the proposed approach is not comprehensive.

In contrast with above kernel density method, the improved fast Gauss transform (IFGT) [8] is another approach to reduce the complexity in training phase. In nature, this technique is based on the fast multipole method [9] to cluster the data and approximate the final result. By flexible adopting the precision level during the approximation, IFGT can much improve the original technique when dealing with large dataset. Furthermore, IFGT approach also possesses its own partitioning mechanism to compete with kd-tree. In nature, IFGT slices the domain by clustering into k-levels. Subsequently, the calculation of conjugacy in each level is performed instead of conducting matrix-vector products in the conventional default approach. In the next step, the sum of contribution in each level is propagated to the higher level to compute the final result. By applying partitioning mechanism in such different manner, IFGT approach can also significantly drive the complexity of training phase to $O(n)$, which even better than kd-tree approach. However, IFGT method still has drawback that limits this technique in the training phase only. Indeed, IFGT technique relies on fast Gauss transform, which is an advanced version of fast multipole method.

Go through the related works, despite the fact that there are a number of efforts to tackle the complexity problem of GPR, the enhancement of performance in term of processing speed is still limited, especially in hyper-parameters learning phase. Moreover, due to the fact that large datasets tend to be skyrocketed quantitatively and qualitatively, there is a critical need to develop a less complex and acceptable accuracy prediction method based on GPR. The success in proposing this technique not only encourage the development of more efficient and reliable applications in distributed environment, but also improve the current solutions of GPR-family in many research fields.

Chapter 3

Domain analysis and potential problems

3.1 Domain analysis

3.1.1 Characteristics of data

Due to the fact that the research domain of this thesis focuses on improving the monitoring data collected from distributed systems, it is more productive to incorporate the analysis on the aforementioned data as well as the corresponding potential issues. Thus, the goal of this chapter is to deeply study the characteristic of the system statistics and how we can use it to model the desired problems. After this rigorous analysis, some potential applications are introduced to solve the modeled problems. The description of these applications includes the design, the internal mechanism and, the detail explanation. Obviously, this information is useful to provide an intuitive perspective of Gaussian process regression implementation to unlock the real-world obstacles.

One interesting type of monitoring data can be listed as the utilization. Statistically, there are many kinds of system utilization such as CPU utilization, memory utilization, network traffic utilization or throughput utilization, etc. In order to investigate the characteristic of the data, the CPU utilization is chosen as a typical case. Other utilization can also be explained in a similar manner. In other words, most of the inference can be calculated by using the CPU utilization, which is retrieved from monitoring statistics. In essence, the data source is periodic, mostly noise-free, and involves twofold: the first factor that affects the utilization is the assigned tasks. This factor can be measured based on the task's arrival rate. The second factor is the processing rate of CPU cores, so-called the service rate. According to queuing theory, these factors can be modeled *via* the $M/M/s$ Markov model (Figure 3.1). Theoretically, Markov model is a famous stochastic model for modeling the event. In detail, the first M represents the arrival rate of tasks, the second

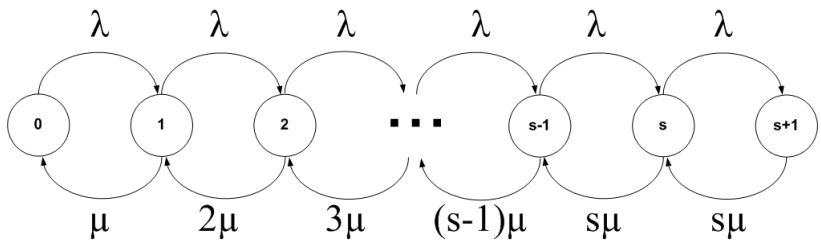


Figure 3.1: Abstraction of processing tasks based on Markov model.

M stands for the service rate, and the last term-s represents the number of service providing units, which are the CPU cores in the processor. Regularly, a normal CPU might consist of s cores. Each core has an identical service rate μ. This service rate represents the processing capability of that core to serve the incoming tasks. Note that the incoming tasks have the rate of arrival λ, repeatedly arrive at CPU cores to be processed. By model the system using these parameters, finally we can describe the utilization of CPU (denoted by ρ) by the following equation:

$$\rho = \frac{\lambda}{s\mu}. \tag{3.1}$$

Theoretically, the inter-arrival time of tasks as well as the service time of cores are considered to be exponential. Franky speaking, these properties are very analogous to the arrival rates in communications area. Therefor, the counting process based on the aforementioned properties can also be considered as a Poisson process. Based on this fact, using stochastic regression technique to deal with the uncertainty of arrival tasks is closely suitable [10].

Assume that with a given set of CPU utilization until time t, the prediction on CPU utilization at time $t+1$ is requested. In this case, the given data can be considered as the input of anticipation procedure, and the desired output is the predictive CPU utilization. In order to achieve the target of prediction, the correlation between input and output is needed to be reformulated. It means that the hidden function, that produces output from input, is function of interest. To reveal this function, there are usually two approaches to be considered. The first approach is restriction bias approach. And the second one is preference bias approach. In the former approach, the classes of candidate are restricted to some potential functions. Meanwhile, the latter approach assigns every possible functions a unique probability. A function with the highest likelihood can be consider as

the most suitable candidate to match the previous hidden function. Each of these methods also has pros and cons. In the first approach, the set of considered functions is limited. Hence, the speed of testing candidate is short and simple. However, the result quality of this approach tends to be poor if the pool of chosen functions does not possess any candidate that matches the hidden function. Moreover, if the implementer decides to breakthrough this drawback by expanding the pool of function to accommodate to the training data. This action might lead to the risk of over-fitting. In that case, even though the trained algorithm might work well with the trained dataset, the accuracy definitely declines drastically when running in empirical situations. Contrary to the restriction bias, the preference bias is obviously more flexible in term of establishing the pool of function. All possible functions are performed to calculate the probability of matching. Therefore, there would be a high chance that the matched function is found. Unfortunately, this advanced feature is also a major reason to degrade the system performance. Apparently, there are a huge number of potential functions to perform. Checking all every functions in a boundary amount of time is clearly impossible. Due to this fact, it is essential to construct another method, which can efficiently do the job within an acceptable sense in terms of accuracy and processing speed.

In order to build an appropriate method, more information of target system and the objective function is needed. Probably, the number of tasks over each period of time establishes a time series function [11]. In other words, a function $y(t)$ with regard to the time t is a stochastic function of the time variable t. To model this relationship, usually the regression technique is utilized to acknowledge the correlation between the current arrival rate and the futuristic one. Following is the step-by-step explanation on how we build the appropriate solution. Firstly, we should denote the time series function as follows:

$$y(t) = f(t) + \varepsilon, \tag{3.2}$$

in which, $f(t)$ denotes an unknown function that generates the empirical data, ε is a small Gaussian noise that is added to the result. Due to the prior knowledge that the data in monitoring statistics is noise-free, the process ε can be ignored for more convenient reasoning. It is worth noting that adding the noise into the data would make no problem because Gaussian process regression (GPR), which is the major technique used to create the solution, is highly adaptive. Hence, temporarily

removing noise does not affect the generality of the proposal. Considering one observation as a pair of input-output, the target is to measure the probability of Y at given time t. To achieve this target, there is a proper clue to follow. In theory, the probability inference can be done through the derivation of the Bayes' rule [12]. This technique is also named as the Bayesian approach [13] [14], which has the form as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.3}$$

As stated previously, the probability density of counting process for the arrival task can be seen as Poisson process, which is described by:

$$f(k, \lambda) = P(Y = k) = \frac{\lambda^k \exp(-\lambda)}{k!}. \tag{3.4}$$

In central limit theorem [12], the system model that follows the Poisson distribution might tend to converge to Gaussian distribution if the arrival rate increases (and yet the arrival rate of incoming tasks in distributed system increases very quickly). For more information, the Gaussian distribution over time forms up the Gaussian process. Therefore, it is reasonable to incorporate the Gaussian process (GP) method into the Bayesian inference [14] [15]. In comparison to other regression techniques, Gaussian process framework is taken into account as the most appropriate to handle the prediction issues. The advanced feature of the Gaussian process can be listed as non-parametric, robustness and high tractability [1] [16]. More information, Gaussian process framework also possesses the capability of interpolating the predictive value without explicit determination of exact hidden function.

3.1.2 Spectral representation

There is a relationship between the regular data (especially the monitoring statistics) in the time domain and the corresponding representation in the frequency domain. In fact, once a dataset can be described as a set of 'signal' with regard to time dependence, that dataset can be transformed to the frequency domain without problem. In a number of real-world applications, it is natural to consider the values of output as the amplitudes of signals. For instance, the daily stock prices,

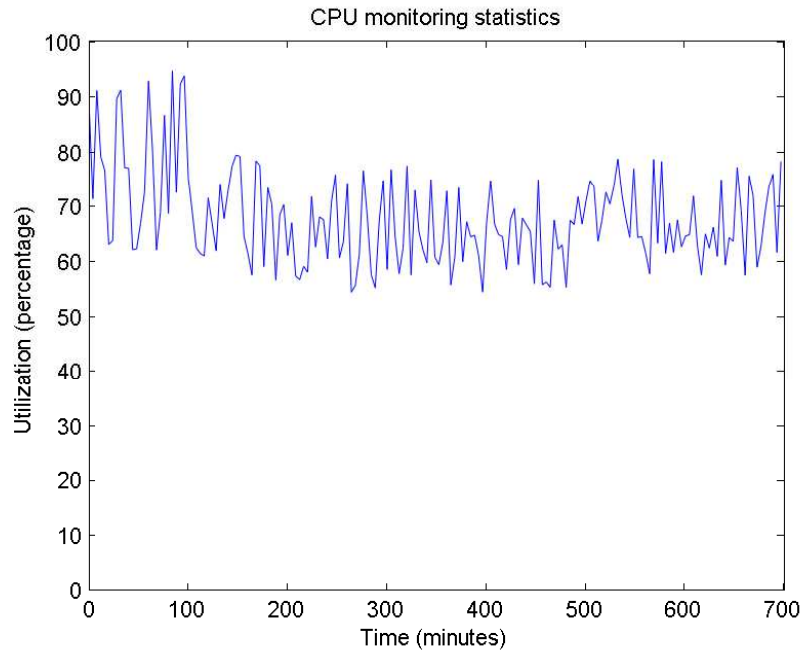


Figure 3.2: Example of CPU monitoring statistics in time domain.

the hourly temperature, or the monthly average atmospheric CO_2 in some specific places can be exhibited as signal. In Figure 3.2, there is an example of CPU monitoring statistics, whose feature of utilization can be seen as amplitude of the signal.

In order to convert data between the time and the frequency domain, the main technique is to use the Fourier transform and the inverse version. By using this technique, a 'sparse' representation of the signal, which encapsulates most of the important information, is retrieved. Analytically, the 'sparseness' is necessary for structural discovery of the signal. The Figure 3.3 shows the equivalent statistics of above CPU utilization but in frequency domain. In this figure, the horizontal axis stands for the considered frequencies, the vertical axis stands for the magnitude of corresponding frequencies. As a side note, we care less about the phase of frequencies, because this property does not contain much useful information. After having this equivalent statistics, we can also calculate the distribution of overall power over the considered frequencies. This distribution is one of the most important prior knowledge, which helps to define the establishment of regression model.

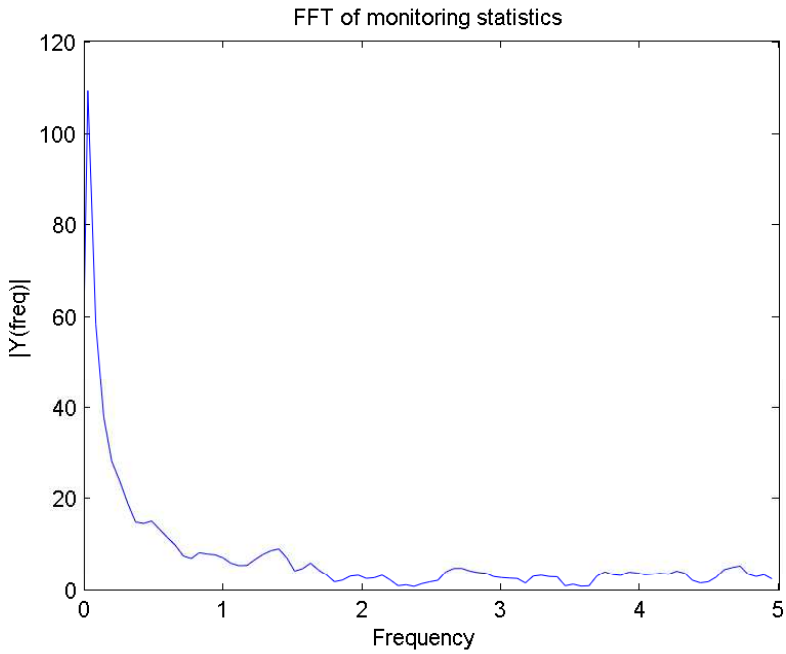


Figure 3.3: Equivalent statistics in frequency domain.

3.1.3 Complexity of modeling

Assume that we have a model M_i that reflects the function $f_i(x, w)$ with prior probability $p(\boldsymbol{w})$ on the parameter $\boldsymbol{w}$. As a consequence, the marginal likelihood of M_i is as follows:

$$p(\boldsymbol{y}|M_i, X) = \int p(\boldsymbol{y}|f_i(x, \boldsymbol{w}))p(\boldsymbol{w})d\boldsymbol{w}. \tag{3.5}$$

The probability in (3.5) is usually known as the marginal likelihood. This probability reproduces a given dataset $\mathbf{y}$ when does the sampling on the set of parameters $\boldsymbol{w}$. These samples are then conditioned on the underlying function that consists of the parameters (for example, $p(\boldsymbol{y}|f_i(x; \boldsymbol{w}))$). By definition, the marginal likelihood is regular probability distribution over $\boldsymbol{y}$. In simple models, when the marginal likelihood is considered to be high on some parts of dataset, it must distribute low probability to the remaining dataset. This property is to satisfy the requirement of normalization. This kind of simple models is described in Figure 3.5. In contrast, a model is evaluated to be complex when it spreads the distribution over the whole dataset, but do not give extreme probability to any particular parts of the dataset.

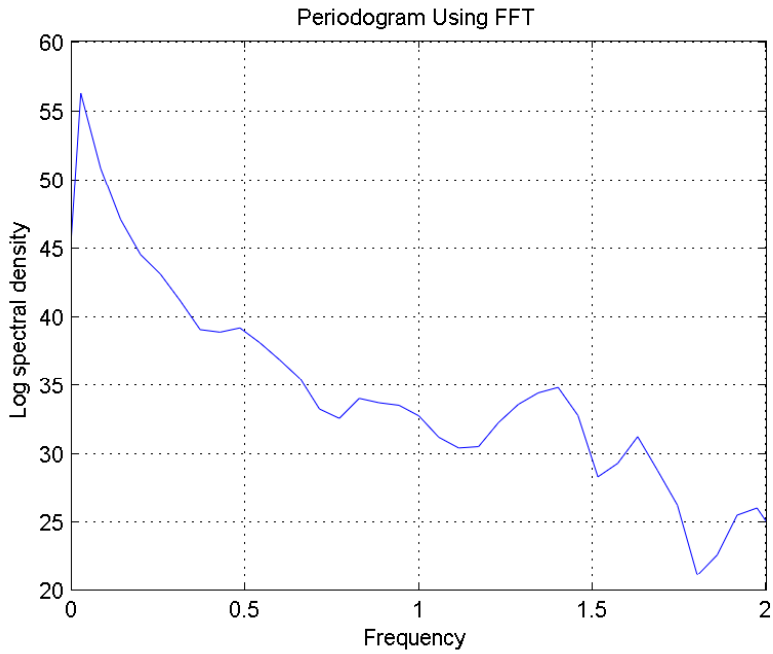


Figure 3.4: Corresponding power spectral density of statistics.

Take a careful look at Figure 3.5, the common sense to define the complexity is based on heuristic observation of the model evidence. However, this method is not always reasonable. Considering a parametric model which has high probability of evidence staying on the investigated parts of dataset. This model of course possesses low probability of evidence in the remaining parts. According to the definition above, we can classify this model as a simple model. However, by conducting the investigation in this way, there would be a great chance that we are over-fitting the model. Contrary this case, in a non-parametric model, instead of focusing on one typical part of data, we equally distribute the possibility to the whole dataset. This method leads to a very high complexity of the model. Even though the new model can earn rich information from data, it cannot help to learn anything. It means that the prediction using this model is independent of the data. Moreover, in our opinion, performing the model selection over simple, reasonable or complex model does not reflect well the underlying function because these corresponding order of complexity is quite limited. This opinion can be easily proved by simple example. Taking into account a polynomials of different orders. Assume that the observations $y(x)$ follow

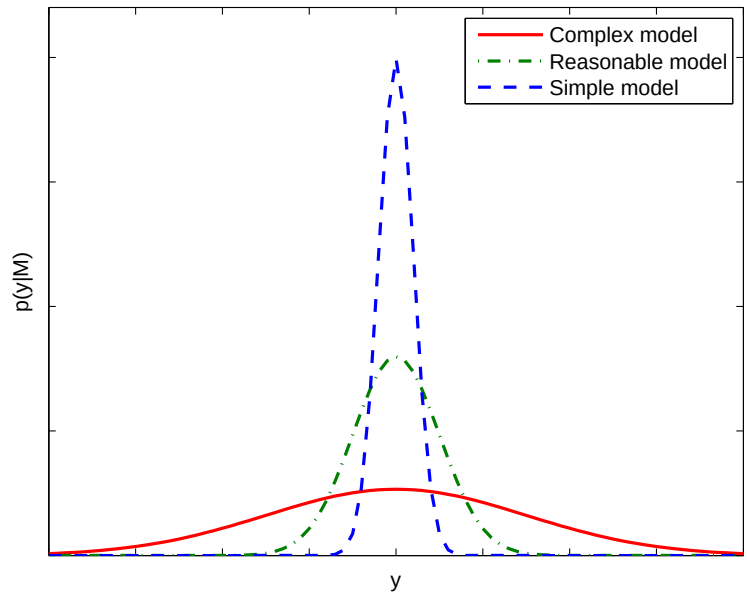


Figure 3.5: Model selection and corresponding complexity by marginal likelihood.

$p(y(x)|f(x)) = \mathcal{N}(y(x); f(x); \sigma^2)$ as shown below:

$$
\begin{aligned}
f_0(x) &= k_0, \\
f_1(x) &= k_0 + k_1 x, \\
f_2(x) &= k_0 + k_1 x + k_2 x^2, \\
&\cdots \\
f_i(x) &= k_0 + k_1 x + k_2 x^2 + \cdots + k_i x^i.
\end{aligned}
\tag{3.6}
$$

Suppose that the actual underlying function behind the observed data does not follow any of above class of model. In reality, this fact is very popular in the practical cases. For example, the data can come from a step function, which is unable to represented at all inputs $x \in \mathbf{R}$ by any finite order polynomial. Preliminary methods in machine learning books such as [17] usually choose an isotropic prior over all parameters $p(\boldsymbol{k}) = \mathcal{N}(0; \sigma_f^2 I)$, and analytically integrate away the parameters $\boldsymbol{a}$. The purpose is to compute the evidence (marginal likelihood) $p(y|M_i)$ of the data $\boldsymbol{y}$ for each model order i. However, because the real data might be out of the class of model, any

finite order polynomial of models might not describe exactly the data. Certainly, the higher order polynomials can describe better the data, but not close enough. This characteristic comes from the fact that a polynomial of order i comprises all polynomials of order less than i. It means that higher order polynomial model comprises all the accuracy of lower orders, and of course be more accurate. Due to this fact, it would be waste of time to compute huge amount of polynomial of different orders, and use them to compute the prediction. It should be more appropriate to engage big enough order of polynomial, which is still computationally feasible. Nevertheless, using such kind of polynomial of high order might result the model to be low evidence as described in Figure 3.5. For instance, the "Occam's hill", which represents this direction, is simply an example of bad prior. Furthermore, if the variance on coefficient k_i is scaled with i, e.g. $p(k_i) = \mathcal{N}(0; i^{-\gamma}I)$. The model might learn the scaling γ from the data. This procedure is done by maximizing the likelihood as a function of γ. Consequently, the marginal likelihood is basically discrepant as a function of model order.

There is another approach to achieve a similar result of integrating away γ, which is sampling techniques. As stated previously, using infinite order model might describe better the real data, which is generated by a hidden process. Harmonizing with increasing the order, scaling up the γ has the same meaning of increasing the complexity of the model. To do that, one possible experiment is to plot samples from a high order polynomial, then varies the parameter γ, to understand how γ makes an impact on the distribution over functions. In fact, our interest focuses on the functions that are used to model the data, not the parameters in a parametric model. Based on this reasoning, it is natural to directly consider functions, rather than the parameters. This idea is completely compatible with the nature of GP framework, which is mentioned in the subsequent sections. The procedure of Gaussian process framework reflects the prior beliefs to use as high order polynomial as possible to model the data. After that, the Bayesian inference is performed in that model. The detail of this approach will be discussed later in this thesis. Basically, the non-parametric models in Bayesian fashion give us a flexible tool to work with infinite order models without over-fitting. Indeed, these models allow us to exactly reflect our beliefs.

The remaining question is that how can we deal with the complexity? According to Figure 3.5, a non-parametric model, which is a complex model, can extract big amount of information

but learn nothing from the data. This assumption conflicts with our target of using non-parametric model to do the prediction. Therefore, one possible solution is to couple the complexity to the information that can be learned. By this solution, the appropriate approach is to adopt the highest complex model as long as it is still computationally feasible as discussed above. Practically, the research in [18] proposes to utilize the mutual information, which obtained by the process of prediction as well as extracted from empirical data. It is worth noting that the prediction is performed by a model to establish the "information capacity" of the model as:

Theorem 3.1.1 *The capacity of a model M_i is obtained as the mutual information between the empirical data $\boldsymbol{y}$ (at N locations X) and the process of prediction performed by the model $\boldsymbol{y}_*$ (at locations X_*)*

$$I_{i,N} = \sum_{\boldsymbol{y},\boldsymbol{y}_*} p(\boldsymbol{y}, \boldsymbol{y}_*|M_i) \log \frac{p(\boldsymbol{y}, \boldsymbol{y}_*|M_i)}{p(\boldsymbol{y}|M_i)p(\boldsymbol{y} * |M_i)} \tag{3.7}$$

If this information capacity is conditioned on the empirical data $\boldsymbol{y}$, and evaluated at a continuous set of test points $\boldsymbol{y}_*$, then (3.7) can be stated as:

$$I_{i,N} = p(\boldsymbol{y}) \int p(\boldsymbol{y}_*|\boldsymbol{y}) \log \frac{p(\boldsymbol{y}_*)|\boldsymbol{y})}{p(\boldsymbol{y}_*)} d\boldsymbol{y}_*. \tag{3.8}$$

Using Theorem 3.1.1 as new definition of complexity, then the more complex the models are, the more chance that we can learn from the empirical data. Furthermore, the complexity should be defined in more fine-grained fashion. The model is considered to be complex in which term? Intuitively, a mature enough definition of complexity should consider rates of learning as well as a metric that stands for mutual information. All of these perspectives would be explained in detail in the next chapter. At this stage, it is more fruitful to take a look at some potential problems that we solve by using prediction technique. These potential problems might motivate and give us some instructions of how to understand and setup suitable models.

3.2 Potential problems

In this sections, one communications problem and two realistic applications of Gaussian process regression are introduced to showcase how to model and design the predictive solutions to solve problems in real-world. The problems are carefully chosen from communications area and energy efficiency in computer systems. This selection reflects the main target of the research in this thesis, which not only focuses on improving the distributed system performance, but also be able to extend to other regression issues. Moreover, it is worth noting that the guidance is presented step by step excluding the prediction part, which would be explained in detail in the next chapters.

3.2.1 Regression in communications

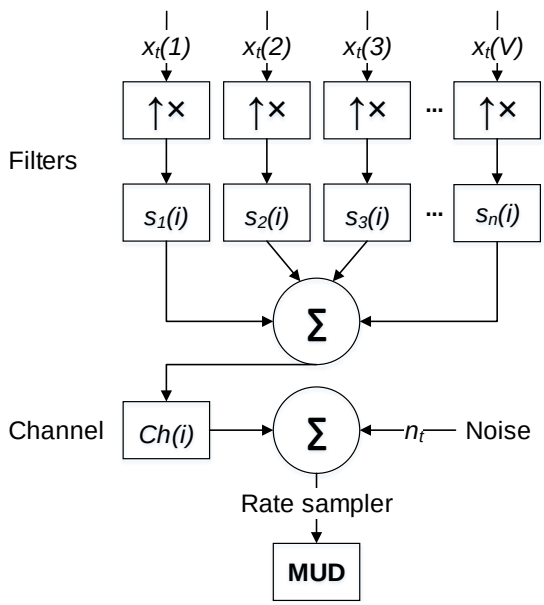


Figure 3.6: Synchronous DS-CDMA system.

The direct sequence code division multiple access (DS-CDMA) system distinguishes users by signals over the channel, as depicted in Figure 3.6. Unfortunately, the interference between the signals happens even with a small number of users and can be recognized as the multiple access interference (MAI). This noisy issue critically increases the bit error rate (BER) under the near/far effect. To alleviate this issue, the multiuser detection [19] (MUD) technique has been developed

to reduce the interference. The known optimal solution for MUD can be retrieved *via* minimizing the mean square error (MMSE) estimation [20]. Nevertheless, doing this estimation immensely costs the computational resources. In order to solve this problem, many approaches have been proposed. Among these approaches, Gaussian process regression (GPR) is considered as the most promising method in terms of flexibility and accuracy [21].

Practically, GPR is widely used in many research fields such as data communication, networking and signal processing. Due to this fact, it is appropriate to apply our proposed method to enhance the multiuser detection. The detail explanation of how to extend the method to this research field can be found in the original paper [22].

3.2.2 Small-scale problem

In the first potential issue, we deal with a small scale of predictive problem, namely energy efficiency in low level of CPU. To do that, we develop a CPU multicore scheduler [23] using prediction technique to save the power consumption. The designed work flow for the corresponding application is described in Figure 3.7. Particularly, the target of this scheduler is to mitigate the power expenditure of processing cores in CPU. Assume that there are many cores on a CPU. In contemplation of achieving energy efficiency, the conventional idea is to stack the working processes on a minimum set of CPU cores and deactivate the remaining idle cores for saving power. Firstly, the source and the destination cores for extracting and inserting the processes, respectively, need to be determined in advance. Noting that the mentioned extraction and insertion procedures are parts of process migration instruction. These procedures crave the predictive information of CPU cores to anticipate the futuristic utilization. Primarily, the information is collected in every heartbeat of the monitoring component. Without this information, the successive construction of process migration might not work properly because of obsolete data. As designed, this information is interpolated in the utilization predictor and plays an crucial role in building the application. Also, we also would like to denote the CPU process that is being considered to migrate as the target process. For reducing the side effect of migrating processes between CPU cores, the application should be attached into background daemon to reduce the delay of making decision.

After finishing the prediction on futuristic utilization of every cores, the utilization predictor

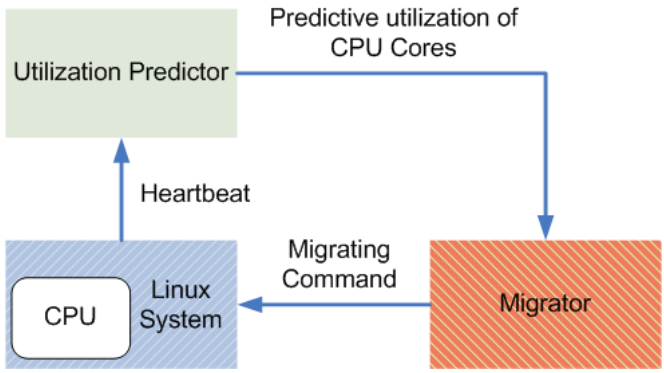


Figure 3.7: The work flow of small scale prediction in CPU multicore scheduler.

sends the desired information to the migrator to build the CPU instructions. Indeed, the migrator employs this information to decide which core is the likely candidate for target processes extraction. In regular situation, the lowest utilized CPU core is mostly preferred to be the source core. For choosing the destination core, there would be a set of important conditions that must be satisfied. Depended on the queuing theory, these conditions are as follows. Firstly, the source and the destination cores should not be the same. Otherwise, no migration can be performed. Secondly, the destination core should not be blocked. As a side note, one core is considered to be blocked when it is too much bus with current tasks and be unable to server another process. Lastly, in case there are more than one suitable destination core matched the aforementioned conditions, the core with higher prediction of utilization is chosen.

Literally, among above conditions, measuring the the blocking rate of CPU cores is the most crucial factor. Presume that new arrival task comes to an arbitrary core. That task has the ability to 'estimate' the average amount of tasks, which currently occupy the processing cores. As discussed before, the arrival tasks counting process is considered to be the Poisson process. Therefore, applying the theory of Poisson arrival see time averages (PASTA) [24] to 'estimate' the average number of running processes is possible. According to this theory, the average arrival time is known to be equivalent to the expectation of waiting time in the CPU cores. Therefore, it would be right time to engage the Pollaczek-Khintchine formula to calculated the expected waiting time

$\mathbf{E}_j$ of CPU core j:

$$\mathbf{E}_j(W) = \frac{\lambda_\tau {}^1\!/\!_{\mu^2}}{2(1 - \lambda_\tau {}^1\!/\!_{\mu})}, \tag{3.9}$$

in which, λ_τ is the arrival rate of incoming task at period τ, μ has been previously denoted as the cores' service rate, and W is the expected queuing delay of CPU. By evaluating the average arrival time, we can estimate the blocking rate of CPU cores as follows:

$$\mathbf{BR}_j = \frac{\mathbf{E}_j(W)}{\mu}. \tag{3.10}$$

After blocking rate evaluation, the remaining job of the migrator focuses on creating the migrating instruction. In the worst case when no handy CPU core for promoting to be the destination, the migrating instruction is postponed until the existence of low-blocking rate core appears. Last but not least, if there is the scenario that all reachable cores are blocked, the stand-by cores would be reactivated to serve the incoming tasks.

3.2.3 Large-scale problem

The methodology of this work [25] can be considered as the expanded effort of the above study. In this research, the problem of interest focuses on reducing the power consumption on a larger scale, the cloud computing domain. So as to obtain this target, the prediction has to combine with the optimization method to solve two following issues. Firstly, it is a must to obtain an effective reduction scheme for power consumption. The remaining issue is to find the possible boundary solution to equate energy savings with an acceptable performance. It has not escaped our notice that the term 'acceptable performance' concerns two conditions. Therein, keeping an acceptable performance, which is documented explicitly in the service level agreement (SLA), is the first condition. In case when this condition is violated, the second condition is in charge of minimizing the penalty cost on the system performance. If the energy efficient orchestrator can firmly hold the conditions, we can say that an acceptable performance has been achieved.

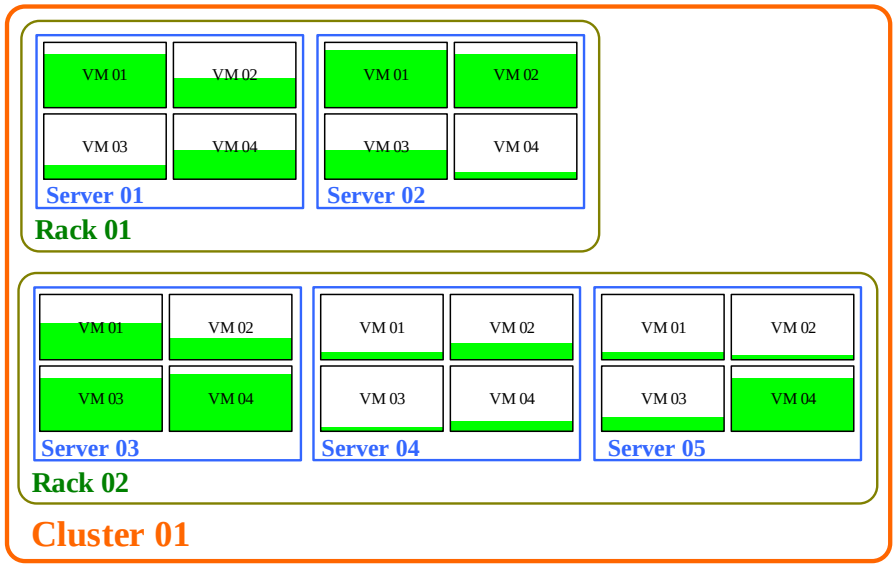


Figure 3.8: A typical utilization of virtualized computing cluster.

Problem description

In the perspective of system organization, cloud computing is nothing special but a bunch of clusters as seen in Figure 3.8. The only difference is that instead of using physical machines to provide services, many virtual machines (VMs) are dynamically allocated on top of the physical infrastructure. As described in this figure, physical machines (PMs) play the role of the container for hosting virtual machines. The cloud orchestrator indeed keeps the highest control and organization. For the computational convenience, the infrastructure works under the assumption that the homogeneous system is actually implemented. This means an identical configuration of hardware and software is equipped with every physical machine. This assumption does not detract the generality because the heterogeneous system can also be modeled as the homogeneous one by compensating with extra weighted parameters.

To lessen the power expenditure of cloud system, the idea is to stack the virtual machines into an optimized pool of physical servers. After that, the action of shutting down the remaining idle machines is effectuated. This idea relies on the fact that a physical server even in idle state consumes the energy up to 60% [26] [27] [28] compared with the same machine in peak performance.

Note that powering up a machine only burns 23.9% [29]. Besides, energy consumption in the cluster is not only related to the power to maintain the running computers, but also the power to operate the cooling system. Consequently, proactively turning off the possible physical servers might lead to better benefit, even this action would consume an extra power to turn these machine on afterward. Based on this fact, the energy efficiency management (E2M) system is formulated as a demonstration for the large-scale potential problem. This system is described in Figure 3.9. The working of this system mainly focuses on optimally scheduling as well as reallocating the VMs over the physical containers. Finally, the operation of idle physical facilities is discontinued to reduce the power consumption. The functionality of internal components in E2M is described below:

- Ganglia [30] component plays the role of the monitoring unit, which collects resources utilization from both physical and virtual machines. Ganglia is a well-known open-source program for collecting purpose. It is simple to implement but robust and effective to get information from most of the computing facilities. At regular intervals, Ganglia reliably hand over the required parameters to the next stage-the prediction step.

- The predictor is developed based on the idea of modeling the processing of arrival requests, is accountable to inference the futuristic data of the desired monitoring period. Subsequently, the predictive information of VMs and PMs is utilized in the energy optimizer.

- The energy optimizer takes the responsibility to create a near-optimized solution. This solution has the objective of achieving power savings scheme while maintaining the aforementioned 'acceptable performance'. For more information, the energy optimizer would decide the appropriate number of active physical machines to host the VMs requests. Moreover, This component judges which physical machines are suitable to stack the virtual machines. All of these decisions are encapsulated in the energy decision, which is then sent to the cloud orchestrator for execution.

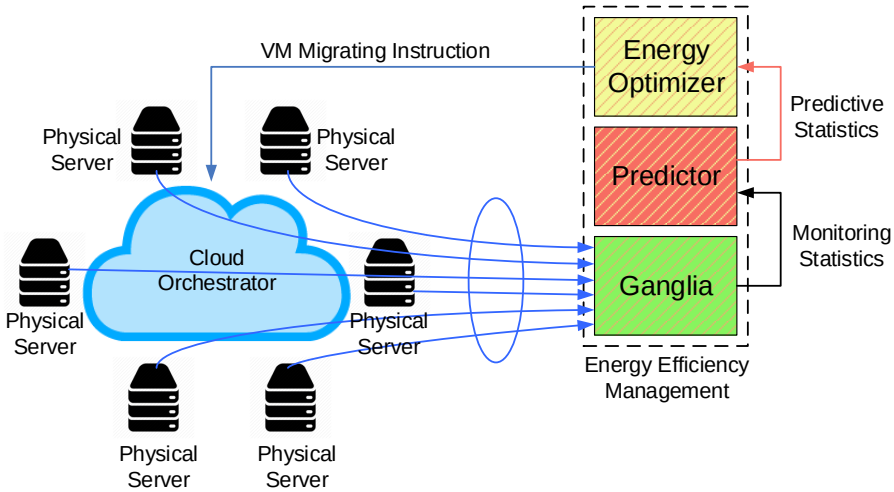


Figure 3.9: Architecture of energy efficiency management (E2M) system.

Energy decision

The information of migrating mechanism should be explained first to understand the working of VM migration. In short, the energy decision mainly relies on the effectiveness of creating VM migrating instruction. Primarily, migrating VMs definitely causes some delays due to the context switching as well as resource re-locating operations. In order to cut down the unavoidable duration, live migration can be a good solution. In essence, this technology can help to partially preserve the seamless behavior of the services and relieve the pain of system downtime. By using live migration, only the memory of VMs is actually sent over the network. In this situation, the migrated VM undergoes a little moment of transient state. After that, the VM might be up and running as usual. As a reference, one research [31] shows that live migration can lessen the downtime into less than 100 ms, which can be accepted in most of the realistic situations. One more benefit, because there would be no need to migrate the storage of VMs, this technology also helps to reduce the network contention, which indirectly reduces the processing overhead. For system performance preservation purpose, sometimes we reactivate the suspended PMs or even keep a controllable small set of under-performance PMs to reduce the overhead on resource utilization. This preservation factor is included as a weight parameter in modeling the energy consumption. Continue to discuss the migration mechanism for VMs, the corresponding migrating

decision is created in the sub-component of the energy optimizer, namely the power management component. The detail organization of the energy optimizer can be found in Figure 3.10.

For creating the VMs migrating instruction, the energy optimizer needs to calculate the optimal quantity of PMs based on the predictive information of utilization of underlying infrastructure. Relying on this instruction, the cloud orchestrator would adjust the current number of PMs. In order to change the amount of PMs, it is compulsory to select the source as well as the destination PMs to withdraw and restore the collected VMs. For choosing the source PMs, usually, we choose the PMs with the slightest performance in terms of CPU and memory utilization. This choice is actually affected by the purpose of resource deallocation on low utilization PMs. After that, when the chosen PMs have been fully released, they can be deactivated to save the energy. Choosing the source is not a burden, but selecting the destination should be carefully considered. Based on the awareness of designing the operating system, many conditions must be matched to pick up a destination. The first condition is the duplication of the source and the destination. Obviously, the two parts of target PMs must not be the same node. The second condition is that the destination PMs should not be too much busy. Otherwise, this designated PM would be unable to accept more VMs. These PMs can be marked as blocked PMs and apparently not be a feasible candidate for the destination. The last condition is considered only when there are many possible PMs to be chosen. If that circumstance turns up, the PM possesses lowest blocking rate but highest predictive utilization is decided to be the destination. The apparatus of blocking rate calculation can be found in our original paper [25].

Energy optimization

Coming to this step, the energy optimizer conducts the optimization for power consumption, after receiving enough predictive statistics. As said previously, a minimum-but-feasible number of PMs is required as the output of this stage. Note that the output is subsequently used to construct the instruction for VM migration. Primarily, there are two sub-components in the energy optimizer, namely power management and cluster optimizer. The power management observes the resource pool and incorporates the energy decision that has been made from the cluster optimizer. The final decision can be referred to as the instruction for VM migration. This instruction is sent to the

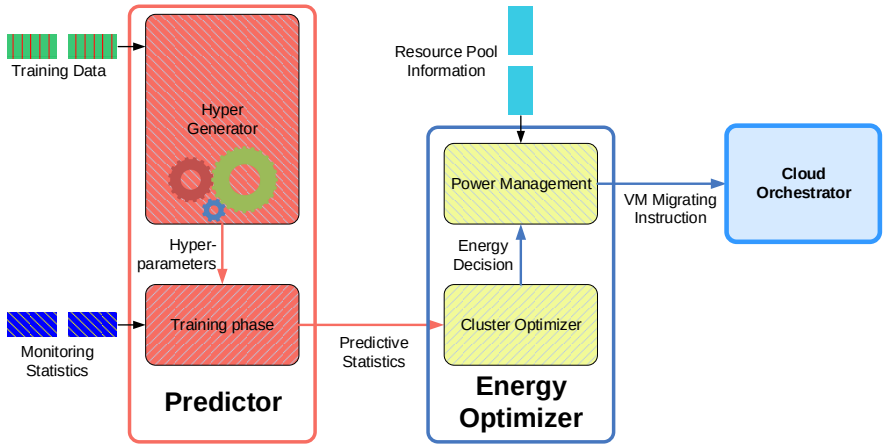


Figure 3.10: Flowchart of energy efficiency management (E2M) system.

cloud orchestrator to actuate.

Chapter 4

Gaussian process regression

4.1 Bayesian learning

4.1.1 Probabilistic background

This section discusses the construction of Bayesian learning. When do the regression, the procedure is basically related to how the reasoner should infer under the uncertain situation. Apparently, the theoretical background for the interference of any similar problem is probability theory [32]. It is worth mentioning that the Bayesian approach is the major technique to investigate in dealing with the uncertainty. Whereby, the probability $P(A \mid I)$ stands for the degree of confidence to state the existence possibility of A given that I is aware of being the truth. Usually, I stands for the prior knowledge of the available information. In other words, this notation represents the cumulative and related information, which the reasoner knows in advance to be useful for the process of inference. Obviously, the term I has another role of conditional probability to calculate the likelihood of A. Also, assume that the reasoning procedure has been given an I amount of background information, a unique probability might assign the confidence to the appearance of A. In that sense, $P(A \mid I)$ determinedly represents a unique value of probability without any redundant definition. It has the same meaning that when providing any reasoner with the exact knowledge I, the same probability of confidence of statement for A would be derived. Thus, a conclusion can be extracted from this fact is that: the result of Bayesian learning mostly relies on the provided information.

Moreover, the term $P(A \mid I)$ also can be seen as a measurement of how I might lead to A [33]. This manner shows the fact that the statement $I \Rightarrow A$ can happen in probability. For example, assume that A is the statement that $x > 0$, I stands for the prior knowledge that $x = 1$. In this

case, $P(A \mid I) = 1$ is an acceptance of the aforementioned hypothesis. However, if I is changed to $x = -1$, obviously $P(A \mid I) = 0$ is a rejection. These situations are described as the extreme cases. Regularly, the probability $P(A \mid I)$, in reality, stays between these extremes. Based on this probability, the reasoner would decide whether it is enough to mark the case as accepted or rejected. Clearly, the probability theory is not only the solid background to build up the logical reasoning, but also the reliable method to evaluate the possibility of any proposition. This can be seen as an extension for widening the traditional logic. In fact, we use this logical extension daily. For instance, when facing a door locked without an appropriate key, we usually wonder 'where is the key now?'. Naturally, there are always the assignments of probability to the potential places for leaving the key. Due to this reason, the Bayesian theory is common sense described not in human language, but mathematical language.

Before discussing Bayesian theory, it is necessary to introduce some probabilistic laws, which are the most relative. It has not escaped from out notice that the probability is assigned as a real number from the range of the extremes $[0, 1]$:

$$P(\neg A \mid I) + P(A \mid I) = 1 \tag{4.1}$$

$$P(A, B \mid I) = P(A \mid I)P(B \mid A, I) = P(B \mid I)P(A \mid B, I), \tag{4.2}$$

in which, $P(A, B \mid I)$ is known as the probability of the logical conjunctive proposition $P(A \wedge B \mid I)$, $\neg A$ means the negative A (e.g A is true, $\neg A$ is false and vice versa). Critically, the conjunctive proposition and negative proposition are the most important set of operations that construct most of the reasoning laws. Note that the propositions (4.1) and (4.2) are well-known enough for having their own labels, namely the sum rule and the product rule, respectively. Furthermore, the product rule (4.1) is used to establish the Bayesian theorem, which is depicted as follows:

$$P(B \mid A, I) = \frac{P(A \mid B, I)P(B \mid I)}{P(A \mid I)}. \tag{4.3}$$

In this equation (4.3), the prior knowledge for the term B is represented *via* $P(B \mid I)$. This knowledge actually reflects the belief of the reasoner about B before doing any investigation on the term A. $P(A \mid B, I)$ is described as the likelihood of A conditioned on the existence of B, I. $P(A \mid I)$ is the evidence for A; This evidence or the normalization constant can be rewritten as $P(A \mid I) = P(A \mid B, I)P(B \mid I) + P(A \mid \neg B, I)P(\neg B \mid I)$. This rewritten term can describe better our previous statement in terms of prior knowledge and likelihood. Finally, the term $P(B \mid A, I)$ stands for the posterior probability of B based on the observation of A, I. Actually, this term updates the aforementioned belief of the reasoner about B after the investigation of A. It means that with the help of Bayes' theorem, the reasoner can update the probabilities with the help of new information. In fact, Bayes' theorem is a powerful theory to reason and readjust the decision based on the latest and updated knowledge.

It is right time to firstly discuss the notation of variables. As a side note, capital letters are used for uncertain propositional variables. Let say X is this kind of variable, and x in small letter stands for one arbitrary value from the sample space S_X. For example, X can be referred to as a symbolization for a ball, S_X is 'the surface of the ball' and x is a single point on that 'surface'. We also have the statement that the set $X = x; \forall x$ is collectively exhaustive and mutually exclusive. It means that one and only one of the value x is true. Subsequently, the probabilistic distribution $P(X = \cdot \mid I)$ over x should be denoted. For the purpose of convenience, we can omit the term X to simplify the above notation to $P(x \mid I)$. In case we would like to mention the distribution, the reduced term $P(\cdot \mid I)$ is used. By using the probabilistic laws from (4.1) and (4.2), the normalized condition can be stated as follows:

$$\sum_x P(X = x \mid I) = 1. \tag{4.4}$$

Besides, the marginal probability of X can be calculated over the sum of all possible values of Y on the following joint probability

$$P(X = x \mid I) = \sum_y p(X = x, Y = y \mid I). \tag{4.5}$$

Whenever the probability density function (pdf) needs to be mentioned, we might also use the small letter to denote the desired term. For example, $p(X = x \mid I)$ denotes the pdf for continuous random variable X. Similarly, X can be omitted for the purpose of convenience. Following equation gives the definition of the desired pdf as:

$$p(X = x \mid I) \triangleq \lim_{\delta x \to 0} \frac{P(x \leq X < x + \delta x \mid I)}{\delta x} \tag{4.6}$$

Note that this limit is in general a non-trivial operation [32]. Because of that, ignoring this definition and keeping using the probabilistic laws as if applying them to continuous random variables can result the error. Nonetheless, in case there is a restriction of using finite, normalized pdfs only, the pdfs can be engaged mostly equal as the probabilities [34]. Due to this fact, for infinitesimal dx, a more versatile notation can be employed as:

$$p(X = x \mid I)dx \triangleq P(x \leq X < x + dx \mid I) \tag{4.7}$$

In dealing with continuous random variables, the sums in (4.4) and (4.5) can be replaced with integrals to forms up the corresponding pdfs:

$$1 = \int p(X = x \mid I)dx \tag{4.8}$$

$$p(X = x \mid I) = \int p(X = x, Y = y \mid I)dy \tag{4.9}$$

Unfortunately, there still would be the existence of some small risks. Obviously, (4.6)) implies that $p(X = x \mid I)$ does not remain unchanged over time. It means that even we provide the reasoner the same information I, the probability that we receive later might be different for each trial. On the left hand side of (4.6), there is a dimensionless probability. Meanwhile, on the right hand side, dx and x have the same kind of unit. Therefore, $p(X = x \mid I)$ must possess the unit, which is in the inverse form of the unit of x. Clearly, it is wrong in employing dimensionless function as a pdf for unit-existed variables.

Also, it is natural to make a transformation of x as such: $x \to y = f(x)$. When this circumstance turns up, the probability mass around x and y should be warranted, which subsequently results in the following equation:

$$\begin{aligned} P(x \leq X < x + dx \mid I) &= P(u \leq Y < y + dy \mid I) \\ p(X = x \mid I)dx &= p(Y = y \mid I)dy \\ p(X = x \mid I) &= p(Y = y \mid I)\left|\frac{\partial y}{\partial x}\right| \end{aligned} \tag{4.10}$$

By this way, the pdf is actually scaled by Jacobian transform when changing the variables. Besides, there is another way to achieve the same result by using the wrapped function $y = f(x')$ in (4.9):

$$\begin{aligned} p(X = x \mid I) &= \int p(X = x \mid Y = y, I)p(Y = y \mid I)dy \\ &= \int \delta(x - f^{-1}(y))p(Y = y \mid I)dy \\ &= \int \delta(x - x')p(Y = f(x') \mid I)\left|\frac{\partial y}{\partial x'}\right|dx' \\ &= p(Y = f(x) \mid I)\left|\frac{\partial y}{\partial x}\right| \end{aligned} \tag{4.11}$$

in which, $\delta(x-a)$ is a *Dirac* delta density in x centered at a. For more example, when transforming variable by using wrapped function as $y = \log x$, the pdf of y might has this form: $p(Y = y(x) \mid I)$. This pdf of y corresponds to a $p(X = x \mid I)$ with form $1/x$.

Assume that the extremum x^* of the pdf of a random variable X has been found in this form: $dp(X = x \mid I)/dx\big|_{x=x^*} = 0$. Subsequently, the wrapped function $y = f(x)$ is used as a transforma-

tion, the pdf of Y is as follows:

$$\begin{aligned} p(Y = y \mid I) &= p(X = x \mid I)\frac{dx}{dy} \\ \frac{dp(Y = y \mid I)}{dy} &= p(X = x \mid I)\frac{d^2x}{dy^2} + \frac{dp(X = x \mid I)}{dx}\left(\frac{dx}{dy}\right)^2 \\ \frac{dp(Y = y \mid I)}{dy} &= p(X = x \mid I)\frac{1}{f''(x)} + \frac{dp(X = x \mid I)}{dx}\left(\frac{1}{f'(x)}\right)^2, \\ \left.\frac{dp(Y = y \mid I)}{dy}\right|_{y=f(x^*)} &= p(X = x^* \mid I)\frac{1}{f''(x^*)} \neq 0 \end{aligned} \tag{4.12}$$

Apparently, there is no evidence supports the claim that the equivalent point of Y: $y = f(x^*)$ is also the extremum of the pdf of Y.

4.1.2 Learning procedure

Investigate a simple probability model, namely Bernoulli. Assume that we have a bag of ball and define X as the color of each ball drawn from the bag. Also, we define two types of information which are as follows:

- I_a stands for the information that 'the aforementioned bag encloses two million balls. The color of each ball can be red or white'.
- I_b stands for the information that 'the aforementioned bag encloses an exactly equal amount of red and white balls. The total number of balls is two million.'

At the time we draw the ball, we are unaware of the possibility that we might draw red ball or white ball. In other words, two above information does not give a determined knowledge to distinguish $X = Red$ and $X = White$. Because of that, we assign equally the probability to both cases as below:

$$P(X = \text{Red} \mid I_a) = \frac{1}{2} \tag{4.13}$$

$$P(X = \text{Red} \mid I_b) = \frac{1}{2} \tag{4.14}$$

Nevertheless, there is an inevitable truth that it seems I_b in some senses provides more useful information than I_a does. Obviously, the attitude of the reasoner might be different when receiving both information at the same time. Emotionally, the reasoner is encouraged to believe in the statement (4.14) more than in (4.13). We can call this circumstance as the 'confidence' of defining a statement. And by incorporate this confidence to the belief, the way that the reasoner treats the new information is also different in each case. For example, we provide the reasoner more information $J_d =$'100 red balls have been drawn consecutively'. When learning this information, naturally in the case of I_a, the reasoner might have a thought in his mind that there is a high chance that the bag might enclose only red balls, or the ratio of red balls is much higher than the white balls. However, in the case of I_b, the reasoner is relentless to believe that the quantity of each kind of balls stays the same; and the probability of drawing red ball in the next trial is still unchanged whatsoever. Following is the possibly updated probabilities of (4.13) and (4.14):

$$P(X = \text{Red} \mid I_a, J_d) \gg \frac{1}{2} \tag{4.15}$$

$$P(X = \text{Red} \mid I_b, J_d) \simeq \frac{1}{2}, \tag{4.16}$$

Obviously, (4.14) given I_b is more sustainable to the effect of information J_d than the (4.13) given I_a. Due to this situation, it is reasonable to conclude that the degree of confidence and the corresponding probability might change when given some additional information J. In order to clarify this fact, we declare a variable Q for each outcome of random variable X in the light of additional information I and J as follows:

$$Q(x) \triangleq P(X = x \mid I, J). \tag{4.17}$$

It is worth noting that the variable Q is a set of collectively exclusive and mutually exhaustive propositions. This variable is also treated as others. For example, the proposition $Q = q$ stands for another proposition related to X: $P(X = x \mid I, J) = q(x)$ ($P(X = \cdot \mid I, J)$ in the reduced

form). This proposition is defined exclusively by I and J with an assumption of the distinct form $q(\cdot)$. This conditional probability is depicted in $4.1a$. Since Q can be seen as a unique probability with assumption of I and J, we describe this node as a determined node based on I and J.

$$P(Q = q \mid I, J) = \delta(q` P(X = \cdot \mid I, J)). \tag{4.18}$$

Certainly, I and J still possess some uncertainties. This fact leads to a conclusion that Q also has the uncertainty which are only connected to I and J. Assume that I is a known information, then Q is the point that keeps all the uncertainty for J. For more information on this point, especially the relationship between the probability Q and the information J, are described on [35]. As a side note, Q actually casts no direct effect on X. Intuitively, when the information of I and J is known, Q does not provide any new information to the reasoner. In fact, together with the awareness of I and J, Q gives us exactly the confidence and belief about X. Therefore, learning Q does not make any sense about X if we all know I and J before hand. However, consider the case that

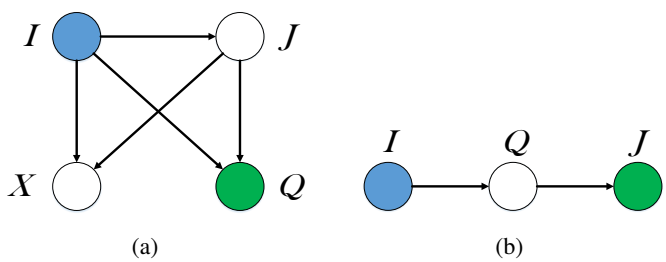


Figure 4.1: Bayesian networks including Q as defined in (4.18).

the reasoner does not have any clue on J. This situation can be seen as the most popular case in reasoning: the reasoner is lacking of the needed information. In regular sense, the suitable solution

in this case is to drop this variable by marginalization.

$$\begin{aligned} p(x, q \mid I) &= \int p(x \mid I, j)p(q \mid I, j)p(j \mid I)dj \\ &= q(x) \int p(q \mid I, j)p(j \mid I)dj \\ &= q(x) \int p(q \mid I, j)\frac{p(I, j)}{p(I)}dj \\ &= \frac{q(x)}{p(I)} \int p(q \mid I, j)p(I, j) \\ &= \frac{q(x)}{p(I)}p(qI) \\ &= q(x)p(q \mid I) \end{aligned} \tag{4.19}$$

As shown in (4.19), the result of the marginalization only concerns with Q and I. It means that the variable I can only affect the belief on X *via* Q even though Q does not have any influence on X. One advantage of introducing the new wrapped variable Q is that we can engage the pdf $p(Q = q|I)$ as a replacement for the requirement of investigating the probability mass around J. Because J is an information source that the reasoner has to learn to construct the belief on X, this variable can represent a huge and complex amount of data in terms of size and feature, which subsequently results in high dimensional data. It is obviously impossible to specify this kind of probability distribution. Fortunately, we substitute the variable J with Q, which is a wrapped function of X. All that we have to do is to investigate the probability distribution around X. In other word, the uncertainty of J is encapsulated into the simpler Q.

Discussing the effect of Q is important. Primarily, the conditional probability $p(q|I)$ in (4.19) can be considered as the prior knowledge. Observe that only the first moment (mean) of this distribution is actually can have an effect on X.

$$P(X = \text{Red} \mid I) = \int_0^1 qp(q|I)dq \tag{4.20}$$

For the previous trial of drawing the ball, mostly the feature of interest of drawn ball is only the color. Other features such as the color distribution of ten first drawn balls, the number of continuous same color drawn balls or the gravity that affects the drawn balls,...etc are not considered.

Because of that, only the mean of $p(q|I)$ is valuable. Any higher moments of $p(q|I)$ might be interesting only when other factors are taken into account for learning. In order to clarify the role of Q, one other knowledge should be stated as I_c = 'the containing bag consists of only white balls or only red balls, but not both of them'. Another aspect of this statement is that I_a, I_b and I_c possess the same probability of occurrence, which is $1/2$, as shown in (4.13) and (4.14). As long as the target of reasoning focuses on X, three statements have the same chance and intuitively are unable to distinguish. However, just as the last time we treated the information which is provided by I_b, whenever the reasoner receives new information, it would affect the belief of reasoning the next drawn balls. In other words, the variable Q with the outcome q, change the probability between the range $[0, 1]$, which subsequently reflects the belief on X, whenever receives new data. However, by observing the case of I_b, the reasoner may be relentless keep the probability of $1/2$ regardless the content of new information. This fact has the same meaning that based on the kind of initial information, the new data might be valuable or might be not worth to learn. In the case of I_c, right at the time we draw the first ball, immediately the probability of Q is established as the extremum because only one ball can reveal all the unchanged truth about what is inside the bag.

In fact, the influence of updated data can be more than that. We can investigate this influence by taking into account a new information K. It is not escaped from our notice that we observe the effect of new information K, but still stay close to the potential of learning the information J subsequently. In order to measure the impact of K, it is necessary to calculate the first moment of Q, which is defined in (4.18), given the information K as follows:

$$P(x \mid I, K) = \int q(x)p(q|I, K)dq \quad = \frac{\int q(x)p(q \mid I)p(K \mid q)dq}{\int p(q \mid I)p(K \mid q, I)dq}. \tag{4.21}$$

Follow the guidance of Bayes' theorem, the prior distribution $p(q \mid I)$ can be updated to provide the knowledge of the posterior distribution $p(q \mid I, K)$ on the belief. This procedure is also applied to the variable J, which also stands for the new information. Get back to the bag of ball that we are interested in. Assume that we draw the ball from the bag continuously with replacement, as shown in Figure 4.2, the observation on the drawn balls can provide more knowledge to update the belief. For the notation, variable K might be used to denote the observations to draw N balls. It is worth mentioning that there is an assumption that the learned information I stays the same

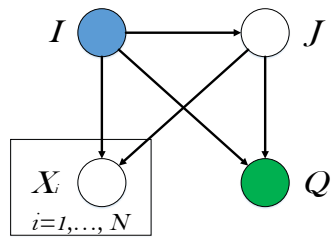


Figure 4.2: Bayesian network for repeated trials X_i.

in every repeated trials. This means there would be no discrimination that the information I can affect to each trial. Also in the Figure 4.2, we still keep the representation of variable J as the hidden information that directly affects the variable Q. Note that the effect of J happens in every trial identically. Last but not least, with the equipment of information I and J, the chance to draw ball in the trial is known to be independent and identically distributed. Then, we have the following probability to define the chance that we draw n balls:

$$P(x_i; i = 1, ..., N \mid I, J) = \prod_{i=1}^{N} P(x_i \mid I, J) \tag{4.22}$$

in which, x_i represents the color of the i^{th} drawn balls, $p(X_i = \text{Red} \mid I, J) = q$ is the probability that we draw the ball with the desired color at time i^{th}. As a side note, the probability of drawing the ball red is similar, given the same information I and J. At this moment, it is necessary to depict the proposition that represents the change we draw exactly n 'red' balls in n trials by using the following definition:

$$\mathbf{X}_l = \begin{cases} X_i = \text{Red} & i = l_1, ..., l_n \\ X_i = \text{White} & i = l_{n+1}, ..., l_N \end{cases} \tag{4.23}$$

This expression looks very similar to definition of the Binomial distribution which can be presented as:

$$P(x_i \mid I, J) = q^n (1 - q)^{N-n} \tag{4.24}$$

It is worth noting that each possibility of drawing red ball is not only independent but also has the identical probability $P(x_i \mid I)$, given the same information I that we know. This knowledge so far is what we understand about the experiment. Anyway, each time the reasoner observes the ball that is drawn, the probability of X is not identical as$P(x_2 \mid I) \neq P(x_2 \mid I, x_1)$. The reason for this issue is that the reasoner has incrementally learned about the configuration of the bag. Because of that, the reasoner possesses more information of the possibility of the next draw. Moreover, with the existence of J, which is presumed to be also identical in each draw, the reasoner has enough ingredient to learn the impact of these observations. This fact explains how we construct the probability in (4.24).

One more feature that needs to be mentioned, the order of the trial is really not important. Whatever the order that we draw the balls, probability $P(\mathbf{X}_l \mid I)$ is unchanged. The only thing that is important in this trial is the composition of the drawn balls out of the bag. In other words, the valuable information to the reasoner is the separately cumulative numbers of red balls and white balls that are drawn. This is called the exchangeable feature. Mathematically, this fact is encapsulated in the below equation, based on (4.24):

$$\begin{aligned} P(x_l \mid I) &= \iint P(\mathbf{x}_l \mid I, j)p(q \mid I, j)p(j \mid I)djdq \\ &= \int q^n(1-q)^{N-n}p(q \mid I)dq \end{aligned} \tag{4.25}$$

Furthermore, there is a theorem which was derived by De Finetti [36] (the proof for this theorem can be found in the research of Heath and Sudderth [37]) comments that for any infinite set of trials, which in that set the exchangeable belief distribution exists, there would be a variable Q with corresponding prior $p(q \mid I)$), has the same role as the namesake variable in (4.25). Remember that we denote Q previously just for the reason of aliasing the content of J, when this variable affects the distribution of variable X. Thus, De Finetti's theorem might imply that in case the belief still has the exchangeable feature even under an infinite number of the experiments, there would be an existence of a set of J, which gives us the knowledge to assign all the trials with independent identical distribution (i.i.d).

Certainly, in a realistic situation, there would be numerous factors that affect the trials, rather than only simple variable J (which represents only the interior of the bag), which again assumed

to has some impacts on the color of the drawn balls. The guidance of trial is only about how to approach the bag and likely draw the balls inside this bag. This trial actually does not possess many things that we can practically study. Obviously, the information is very limited to the color of the subject, not the characteristics of the bag (shape, structure,...). Moreover, the marginalization of hidden information J also drops the important information of composition of the balls. It means that the trials are now unbiased: red balls and white balls might be drawn by the same probability. Given the fact that all these information are keep away from the reasoner, the best thing that the reasoner can do is to eradicate the uncertainty. In other words, the reasoner can only encapsulate the hidden information in the variable Q.

Apparently, if the reasoner is able to learn some of the hidden information, then the trial is no more exchangeable. Understanding the distribution of consecutive drawn balls until a specific time might reveal more accurate information about next ball would be drawn. From this point of view, the final decision might be much more precise and obviously more valuable to do the prediction.

To make clear in this claim, we re-evaluate (4.21) with regard to Figure 4.2. Denote $K = \mathbf{X}_l$, then we have

$$\begin{aligned} P(x \mid I, \mathbf{x}_l) &= \frac{1}{p(\mathbf{x}_l \mid I)} \iint p(j \mid I) P(x \mid I, j) P(\mathbf{x}_l \mid I, j) p(q \mid I, j) dq dj \\ &= \frac{\int q^{n+1}(1-q)^{N-n} p(q \mid I) dq}{\int q^n (1-q)^{N-n} p(q \mid I) dq}. \end{aligned} \tag{4.26}$$

It is worth noting that (4.26) would be identical if we consider the variable K only as the observation for the numbers of balls discriminating by the colors. Thus, the binomial coefficients would be canceled from the equation. For example, consider the case that the diffuse $p(q \mid I) = 1$. This case means there would be no information of the ratio between white and red balls. In this case, the Beta function can be utilized to achieve

$$P(x \mid I, \mathbf{x}_l) = \frac{n+1}{N+2}. \tag{4.27}$$

This is exactly the Laplace's rule of succession, which can be explained in more detail in [32].

When conducting the repeated trial, there is a feeling that we can use the variable Q as a probability to describe the other probability X. This viewpoint makes the reasoner considers

the variable Q as a 'physical' probability rather than just an alias for the hidden information J. However, it is worth mentioning that this probability is just the interpretation of the belief. And this belief relies on the hypothesis that the reasoner is aware of the knowledge, which is provided by J. Due to the lack of the real information underneath (which is unknown), this variable gives us the impression that it is identical to every trial. Because of this reason, the reasoner would never forget that Q is no more than an alias for J. This definition makes the variable Q useful. In this way, Q, as a semantic replacement for J, can interact with other variables. This interaction can be clarified in detail when more reasoners are added to the system, make it multiagent system. Subsequently, it is natural that one particular reasoner might have the desire to know what is the belief of others.

4.2 Gaussian process regression

4.2.1 Original form

After utilizing Bayesian inference as for the learning framework, it is the right time to discuss the prediction model. As stated in the motivation of this research, Gaussian process (GP) is chosen due to the property of high accuracy in doing regression. Primarily, the collaboration of Gaussian process and Bayesian learning is really close as described in [2]. Generally, Gaussian process possesses a robust capability to conduct the Bayesian inference over functions. When evaluating potential hidden functions, there is an effective way that we can encapsulate all possible functions into vectors. In this way, we can consider a huge number of functions with convenience. However, let's consider the regular case that an infinite number of necessary functions are adopted as input to the model. Common sense is that there would be an infinite number of corresponding output functions. Handling these couples of numerous functions is obviously unfeasible. Because of this reason, we use the Gaussian process to define a probability distribution over functions. In this manner, the defined Gaussian process becomes a prior distribution for aforementioned functions.

Moreover, there would be a shortage if not mention one other important feature of the Gaussian process: the flexibility, which is a very useful aspect. Imagine that we engage GP to define the probability distribution over functions as stated above. Technically, any subset of these func-

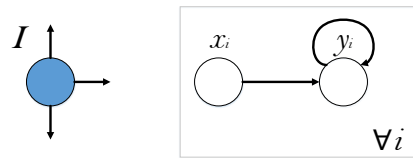


Figure 4.3: Bayesian network for an original GP.

tions also establishes a multivariate Gaussian. Furthermore, the marginalization and the crossover conditioned probability of any subset are Gaussian. This feature put an immense power in our hand to investigate or omit any arbitrary subset at the highest degree of versatile.

As an illustration for above claim, suppose that we have a hidden function $y(x)$. Because this is a hidden function, none of the input or output of the function is considered to be deterministic. Clearly, we have no choice than to consider these quantities as random variables. At each epoch x_i of time x, the function $y(x_i)$ appears with different instance. This observation reminds us the definition of random process. Instead of the random variable, which assigns a number to each outcome x_i of variable x in a sample space $\mathbf{X}$, a random process assigns a sample function to each outcome x_i of variable x. It means that if we define $Y(x) = y(x)$, $Y(x)$ would be a random process. It is necessary to note that we are considering the time series prediction technique. Because of that, we assign a specific role for variable x as the quantity of time. In general, x can be anything, not only for expressing the point of time. To be compatible with our previous statement, we can also depict y as the vector of output functions and t as the vector of input functions. Considering a typical Bayesian network, which is illustrated in Figure 4.3, we might use this network to describe the relationship between defined variables. As claimed before, the function $y(x)$ is originated from the random process $Y(x)$. What if $Y(x)$ is Gaussian process. Whereby, this GP can be defined as follows:

$$p(\mathbf{y} \mid \mathbf{x}, \mu, \mathbf{K}, I) \triangleq \mathcal{N}(\mathbf{y}, \mu, \mathbf{K}). \tag{4.28}$$

As discussed above, this GP is enduring with parameter μ standing for the mean and parameter $\mathbf{K}$ standing for the covariance matrix regardless any subset of y and x.

4.2.2 Practical form

In most of the problems that we encounter until now, which motivate us to propose this enhancement, the object of the prediction is more specific. Hence, the target is to improve the time series monitoring statistics of the computer system. Naturally, Bayesian learning and Gaussian process regression are employed as the inference framework and probability model, respectively. Because the input data for this model is the time series information, curve-fitting is preferred over function mapping for the mapping approach. It is important to note that the curve-fitting is more flexible with regard to the time series data and non-stationary model. Due to the fact that our problem is very specific as stated, a distinct and more useful Gaussian process with regard to the time-series data should be re-defined as below.

Given the input as a set of sampling location $\boldsymbol{x} = [x_1, x_2, x_3, \cdots x_n]$, the random output is denoted by: $\boldsymbol{y} = [y_1, y_2, y_3, \cdots y_n]$, this set of $\boldsymbol{y}$ stands for a joint Gaussian distribution of incoming desired information with regard to the time order. This set over the time constraint actually forms up the Gaussian process:

$$f(y|x) \sim \mathcal{GP}\big(m(x), k(x, x')\big) \tag{4.29}$$

with

$$m(x) = \mathbb{E}\big(f(x)\big) \tag{4.30}$$

$$k(x, x') = \mathbb{E}\bigg(\big(f(x) - m(x)\big)\big(f(x') - m(x')\big)\bigg) \tag{4.31}$$

in which, $m(x)$ is the mean function, evaluated at the location x variable, and $k(x, x')$ is the covariance function, also known as the kernel function [38]. By definition, the kernel function is a positive-definite function, used to define the prior knowledge of the underlying relationship. Basically, the kernel function is only a mandatory requirement when there is a lack of finite dimensional form of the feature space. Otherwise, it can be dropped by directly calculating the sample.

However, this feature space dimension is frequently infinite, which means that the kernel function cannot be directly calculated. For this reason, the kernel function technique is often chosen to tackle the Gaussian process regression. In addition, the kernel function comprises some special parameters that specify its own shape. These parameters are referred to as hyper-parameters. Because the input data comes to the Predictor as a set of locations, the kernel should be engaged in matrix form.

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{pmatrix} \tag{4.32}$$

4.2.3 Kernel functions

In the center of every Gaussian process model, the covariance function controls all the power of modeling. By definition, the covariance function k directly defines the covariance between random functions, which are measured at any arbitrary input points: $k(x, x') = cov(f(x), f(x'))$. Indeed, the covariance function, or the kernel, encapsulates the prior beliefs, which reflects the potential of target functions that might closely model the data. Whereby, when selecting a covariance function, we actually select the characteristic of the potential solution. Should the corresponding functions be periodic, smooth, linear or polynomial, etc? All of these properties can be defined *via* describing the covariance function. In other words, instead of modifying or re-defining the algorithm to adapt to the fluctuation of the model, we can simply achieve the same result by adjusting the covariance function. Moreover, before predicting or inferring posterior distributions over empirical data, we can also incorporate our prior knowledge of the data such as the periodicity, rate of fluctuation and the smoothness...

As a side note, the terminologies such as "kernel", "covariance kernel", and "covariance function" are used interchangeably with the same meaning. Generally, the definition of a kernel stands for a function that maps any pair of inputs into $\mathbf{R}$. The typical example of covariance function is the kernel of Gaussian process. For example, $k(x, x')$ is a valid covariance function of a Gaussian

process. Moreover, the matrix $\mathbf{K}$ with elements $k_{ij} = k(x_i, x_j)$ must be positive semi-definite ($z^\top \mathbf{K} z \geq 0$ for all $z \in \mathbf{R}^N$). This mandatory properties is strictly needed due to the fact that Gaussian distribution requires the covariance matrix to be positive semi-definite. In more detail, the strict requirement of positive semi-definite means that the covariance function actually coincides to an inner product in some feature space [17]. Note that the requirement of positive semi-definite is quite easy to check directly rather than looking for the underlying inner product. Considering a popular linear model, which is depicted below:

$$f(x) = \boldsymbol{w}^\top \boldsymbol{\phi}(x) \boldsymbol{w} \sim \mathcal{N}(0, \Sigma_w). \tag{4.33}$$

This model coincides to a Gaussian process with following covariance kernel:

$$k(x, x') = \boldsymbol{\phi}(x)^\top \Sigma_w \boldsymbol{\phi}(x'). \tag{4.34}$$

One of the most popular covariance kernels that we are interested in is the squared exponential (SE) kernel:

$$k_{SE}(x, x') = \sigma_f \exp(\frac{||x - x'||^2}{l^2}). \tag{4.35}$$

Not only we can use the predefined covariance functions, but also we can define our own covariance functions by using some simple techniques, which are introduced below. Note that these techniques are from [17] and [39]. By engaging these techniques, we can build new valid covariance functions from existing covariance functions. Assume that $k_1(x, x')$ and $k_2(x, x')$ are

valid kernels. Subsequently, the following kernels are also valid:

$$
\begin{aligned}
k(x, x') &= h(x)k_1(x, x')h(x'), \\
k(x, x') &= r(k_1(x, x')), \\
k(x, x') &= \exp(k_1(x, x')), \\
k(x, x') &= k_1(x, x') + k_2(x, x'), \\
k(x, x') &= k_1(x, x')k_2(x, x'), \\
k(x, x') &= k_3(\boldsymbol{\phi}(x), \boldsymbol{\phi}(x')), \\
k(x, x') &= x^\top A x', \\
k(x, x') &= k_i(x_i, x_i') + k_j(x_j, x_j'), \\
k(x, x') &= k_i(x_i, x_i')k_j(x_j, x_j'),
\end{aligned}
\tag{4.36}
$$

in which, h is any function, r is a polynomial with non-negative coefficients, $\boldsymbol{\phi}(x)$ is a function from x to $\mathbf{R}^M$, k_3 is a valid covariance function in $\mathbf{R}^M$, A is a symmetric positive definite matrix, x_i and x_j are not necessarily disjoint variables with $x = (x_i, x_j)^\top$, and k_i and k_j are valid covariance functions in their respective spaces.

In the incoming sections, some popular covariance functions such as dot product, squared exponential, rational quadratic, neural network, Gibbs, Matérn, and periodic kernels are introduced after including the definition of stationary kernels.

Stationary kernels

A kernel is considered to be stationary when that kernel is invariant under translations of the input space. It means that stationary kernel only depends on the lag between the inputs, not on the absolute values of the inputs. In other word, any kernel, which is only the function of $\tau = x - x'$ with x and x' are any arbitrary inputs, is stationary kernel. In particular, the distance kernels, so called isotropic kernels, are clear representatives of stationary kernels.

The stationary assumption is very useful in incorporating the prior beliefs. Because of this reason, we concentrate on introducing the stationary kernels such as squared exponential, rational quadratic, and Matérn kernels,... It is worth noting that these kernels are not only stationary, but

also isotropic. In reality, learning the specific stochastic process by using covariance function is pretty hard, if the kernel is assumed to produce equal probability for every distance of inputs. Such kind of assumption is practically unrealistic and not reflects well the prior beliefs.

Following discussion is about the relationship of stationary kernel between time domain and frequency domain. Theoretically, the kernel can be calculated as an integral using Bochner's theorem [40]:

Theorem 4.2.1 *(Bochner) A complex-valued function k on $\mathbf{R}^P$ is the covariance function of a weakly stationary mean square continuous complex-valued random process on $\mathbf{R}^P$ if and only if it can be represented as*

$$k(\tau) = \int_{\mathbf{R}^p} \exp^{2i\pi\omega\tau} \psi(d\omega), \tag{4.37}$$

where ψ is a positive finite measure.

Furthermore, the function ψ can be expressed as a density $S(\omega)$ in frequency domain, then S is exactly the power spectral density of kernel k. Mutually, k and S are Fourier duals [41] as shown below:

$$k(\tau) = \int S(s) \exp^{2i\pi\omega^\top\tau} d\omega, \tag{4.38}$$

$$S(\omega) = \int k(\tau) \exp^{-2i\pi\omega^\top\tau} d\tau, \tag{4.39}$$

According to this relationship, it is reasonable that the power spectral density can help to reveal the properties of corresponding stationary kernel. Analytically, the power spectral density usually provides more information than the kernel itself. Basically, when we apply the Fourier transform on a stationary kernel, the output power spectral density would reveal the power distribution on a specific range of frequencies. For example, if the power tends to distribute to the range of high frequencies, then the power spectral density might have the shape of heavy tailed distribution. In contrast, the Gaussian white noise, which corresponds to an SE kernel with length-scale $l \to 0$,

might have a uniform (flat) power spectrum.

It is worth noting that a stochastic process with a heavy tailed distribution of power spectrum might consists more errors. This result comes from the fact that the power spectrum of this process distributes more power over high frequencies. The Ornstein-Uhlenbeck process can be a typical example for this kind of processes. Contrary to this point, a random process with power spectrum focuses on low frequencies will appear to be less erratic or noisy. Obviously, by investigating the kernel function in frequency domain, we can learn many hidden features that we cannot see them in time domain.

For more practical instances, Let consider the data on a regular 1D input grid of N points. The empirical power spectrum $\hat{S}(\omega)$ can be defined as follows:

$$\hat{S}(\omega_m) = \frac{|\tilde{y}(\omega_m)|^2}{N},\ m = 0, ..., N-1, \tag{4.40}$$

in which, $\tilde{y}(\omega_m)$ is the m^{th} element of the discrete Fourier transform (DFT) of the data vector $\boldsymbol{y}$. The empirical power spectral density is determined for the range of frequencies $\omega_m = 0, f_s/N, 2f_s/N, \cdots, f_s/2$, where f_s is the sampling rate of the data. By applying the Nyquist frequency of $0.5f_s$, signals are aliased back to lower frequencies.In other word, the Fourier transform on kernel function can be seen as the low-pass filter in frequency domain.

Dot product kernel

Considering linear functions $f(x) = ax + b$, in which, $a \sim \mathcal{N}(0; \alpha)$ and $b \sim \mathcal{N}(0, \beta)$. In theory, the random functions with this form can be seen as a Gaussian process with mean and covariance functions as $m(x) = 0$ and $k(x, x') = \alpha^2 x \cdot x' + \beta$, respectively. By using this Gaussian process to model the empirical data, the linear functions output can be extrapolated or interpolated accurately. This kernel function $k(x, x')$ is called as dot product covariance function. As a side note, this kernel is non-stationary because it does not depend on the lag between data points. Generally, the dot product kernel can be extended to the polynomial kernel, $k(x, x') = (x \cdot x' + \sigma_0^2)^p$. Using this extended version equals to using polynomial basis functions, which is popular in linear regression. For example, if $p = 2, \sigma_0^2 = 0$, it means that the dimension of x is 2D, then $k(x, x') = \boldsymbol{\phi}(x)\boldsymbol{\phi}(x')$ and $\boldsymbol{\phi}(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$. According to the rule of establishing new covariance kernels, these

kernels are valid because they are setup as inner products of basis functions.

Squared exponential kernel

The squared exponential (SE) kernel is considered as the most popularly used in the machine learning areas based on kernel [2]. This kernel is also named as Gaussian kernel, radial basis kernel (RBF). In order to derive the SE kernel, an easy-to-understand approach is to derive in the weight space:

$$\begin{aligned} f(x) &= \sum_{i=1}^{J} w_i \phi_i(x), \\ w_i &\sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right), \\ \phi_i(x) &= \exp\left(-\frac{(x-c_i)^2}{2l^2}\right). \end{aligned} \tag{4.41}$$

(4.41) is the definition of a regression model based on radial basis function, which is centered at the mean c_i. From (4.34), the kernel of this Gaussian process is defined as:

$$k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^{J} \phi_i(x)\phi_i(x'). \tag{4.42}$$

From this equation, it is intuitive that the whole radial basis weight space of (4.41) is encoded as a distribution over functions with kernel in (4.42). (4.42) can be seen as a Riemann sum if aliasing $c_{i+1} - c_i = \Delta c = {}^1/_J$:

$$k(x, x') = \lim_{J\to\infty} \frac{\sigma^2}{J} \sum_{i=1}^{J} \phi_i(x)\phi_i(x') = \int_{c_0}^{c_\infty} \phi_c(x)\phi_c(x')dc. \tag{4.43}$$

In case that $c_0 = 1$ and $c_1 = 1$, The infinite basis functions can be spread through the whole real line with distance $\Delta c \to 0$ separated:

$$\begin{aligned} k(x, x') &= \int_{-\infty}^{\infty} \exp(-\frac{x-c}{2l^2}) \exp(-\frac{x'-c}{2l^2}) dc \\ &= \sqrt{\pi} l \sigma^2 \exp(-\frac{(x-x')^2}{2(\sqrt{2}l)^2}) \end{aligned} \tag{4.44}$$

Due to this fact, the squared exponential kernel of GP, which is derived as:

$$k_{SE}(x, x') = \sigma_f^2 \exp(-\frac{||x-x'||^2}{2l_f^2}), \tag{4.45}$$

equals to an RBF model with infinite basis functions spreading across the whole real line. Obviously, the prediction can be done by using a kernel within a set of finite computational resources.

One noticeble property of SE kernel is that this kernel is infinitely differentiable. Besides, the SE kernel engaged inside a Gaussian process possesses the behavior of approximation. It means that given enough data to a Gaussian process equipped with SE kernel, the desired approximation on any function would be well-done [42] [43]. Moreover, due to the infinitely differentiable feature, a Gaussian process equipped with SE kernel can model any continuous function to within any small ϵ band [42].

Last but not least, the relationship of SE kernel between time domain and frequency domain should be addressed. By conducting the Fourier transform on the SE kernel, we retrieve the corresponding power spectral density for inputs $x \in \mathbf{R}^P$, which is in the form of: $\mathcal{F}_{SE}(\omega) = (2\pi l_f^2)^{P/2} \exp(-2\pi^2 l_f^2 \omega^2)$. The short comment of this power spectrum is that the power would be distributed most of the support on low frequencies due to the low-pass filter's behavior, as discussed in the previous section.

Rational quadratic kernel

The formulation of rational quadratic kernel is to improve the weakness of squared exponential kernel. In nature, the SE kernel tends to model the data by using only one specific length-scale hyper-parameter. Nevertheless, the variety in underlying functions might drive the empirical data in very discrepant ways and different scales. As an example, the research in [44] shows that

The squared exponential kernel assumes that the data are only varying at one particular length-scale. In reality, however, different mechanisms underlying the data could be varying on different scales. For example, the research in [44] found that the variance of the results on the indices might encapsulate patterns that fluctuate in different scales. Because of this reason, modeling this kind of problem can be improved if the sum of reasonable number of SE kernels is engaged. Each SE kernel cane be setup with different length-scales to model data better.

Unfortunately, the exact fluctuation of the scales are unknown depending on data only. In this sense, the feasible choice is to take care as many as possible scales. The rational quadratic (RQ) kernel goes by this way. This kernel is actually a scale mixture of SE kernels with various length-scales. Theoretically, by aliasing the general SE kernel as function of $r = ||x - x'||$, the scale mixture of these SE kernels is written as below:

$$k(r) = \int \exp(-\frac{r^2}{2l^2}) p(l) dl, \tag{4.46}$$

in which, $p(l)$ is a probability distribution over length-scales l. In case of putting a Gamma distribution on inverse squared length-scales, $\gamma = l^{-2}$, $g(\gamma|\alpha, \beta) \propto \gamma^{\alpha-1} \exp(-\alpha\gamma/\beta)$, with $\beta^{-1} = l^2$, the rational quadratic kernel is derived as follows:

$$\int_0^\infty k(r|\gamma) g(\gamma|\alpha, \beta) d\gamma = (1 + \frac{r^2}{2\alpha l'^2})^{-\alpha}. \tag{4.47}$$

In summary, the rational quadratic kernel is defined to model data varying over a diverged set of length-scales. When $\alpha \to \infty$, this kernel might converge to the SE kernel. Note that Gamma density function is not the only choice to describe $p(l)$. Other functions can be used, which then lead to other kinds of covariance functions.

Neural network kernel

Currently, Gaussian process is innovated by the development of neural network kernel in the area of machine learning. Since Bayesian models is known not to over-fit [45], it is possible to engage the models that have higher capability to depict the complex stochastic processes existing in reality. Even in case that the model is really complex, there should be a chance in which some parts of

the data could be encoded enough information to improve the performance. Thus, the research in [45] intends to build up large models with regard to this philosophy. In this research, a Bayesian neural network can converge to a Gaussian process by using a neural network kernel. Originally, this approach is inspired by [2] to discover more about the capability of models based on Gaussian process.

According to [45] and [2], it is possible to consider a neural network with one hidden layer:

$$f(x) = b + \sum_{i=1}^{J} v_i h(x, \boldsymbol{u}_i), \tag{4.48}$$

in which, v_i are the hidden to output weights, h is the transfer function with any bounded hidden unit, $\boldsymbol{u}_i$ are the input to hidden weights, and J is the number of hidden units. The bias b and the hidden to output weights v_i can be expressed by the distributions with independent zero mean, and variances σ_b^2 and σ_v^2/J, respectively. Analogously, the distribution of weights for each hidden unit $\boldsymbol{u}_i$ can be independent and identical.

Taking first order and second order moments of $f(x)$ in (4.48), then encapsulating all weights into the vector $\boldsymbol{w}$, we have:

$$\mathbb{E}_{\boldsymbol{w}}[f(x)] = 0, \tag{4.49}$$

$$\begin{aligned} cov[f(x), f(x')] = \mathbb{E}_{\boldsymbol{w}}[f(x)f(x')] &= \sigma_b^2 + \frac{1}{J}\sum_{i=1}^{J} \sigma_v^2 \mathbb{E}_{\boldsymbol{u}}[h_i(x, \boldsymbol{u}_i)h_i(x', \boldsymbol{u}_i)] \\ &= \sigma_b^2 + \sigma_v^2 \mathbb{E}_{\boldsymbol{u}}[h(x, \boldsymbol{u})h(x', \boldsymbol{u})]. \end{aligned} \tag{4.50}$$

The second line of (4.50) is derived since each of the $\boldsymbol{u}_i$ are identically distributed. Note that the sum in the first line is over J i.i.d. random variables (RVs). Moreover, all moments also have explicit boundaries. If b follows the Gaussian distribution, obviously the central limit theorem can be applied to indicate that when $J \to \infty$, any set of function $f(x1), \cdots, f(x_N)$ might have a joint Gaussian distribution. Therefore, the neural network is now transformed to a Gaussian process with kernel given by the second line of (4.50), with $J^{-0.5}$ is the rate of convergence.

In case the transfer function is selected as $h(x, \boldsymbol{u}) = erf(u_0 + \sum_{j=1}^{P} u_j x_j)$, where $erf(z) = {}^2/_\pi \int_0^z \exp -t^2 dt$, $u \sim \mathcal{N}(0; \Sigma)$, then explicit form of neural network kernel can be obtained as:

$$k_{NN}(x, x') = \frac{2}{\pi} sin(\frac{2\tilde{x}^\top \Sigma \tilde{x}'}{\sqrt{(1 + 2\tilde{x}^\top \Sigma \tilde{x})(1 + 2\tilde{x}'^\top \Sigma \tilde{x}')}}), \tag{4.51}$$

in which, $x \in \mathbf{R}$ and $\tilde{x} = (1, x^\top)^\top$.

As shown in [2], samples from a Gaussian process equipped with this kernel are superpositions of the functions $erf(u_0 + \boldsymbol{u}^\top x)$, which is consistently converged to a constant for large value x. In that sense, the kernel might become a non-stationary kernel. Last but not least, the squared exponential kernel can also be reformulated by applying an infinite neural network with transfer function $h(x, \boldsymbol{u}) = \exp(-||x - \boldsymbol{u}||^2/\sigma_l^2$ and $u \sim \mathcal{N}(0\sigma_u^2 I)$. This procedure will generalize the new form of SE kernel.

Gibbs kernel

In some particular purposes, a non-stationary kernel function, which consists of an input dependent length-scale $l(x)$, is preferred to the common stationary kernels. In principle, the desired kernel can not be achieved by simply substituting l in the SE kernel of (4.45) with any function $l(x)$. This action can not produce a valid kernel. The reason is that to make sure the $k(x, x')$ to be a valid kernel, the covariance matrix $\mathbf{K}$ with elements $k(x, x')$ must be positive semi-definite. This requirement exists based on the fact that the covariance matrix in a Gaussian distribution must be positive semi-definite. In order to satisfy this requirement,the research in [46] derived a valid kernel with an input dependent length-scale as such:

$$k(x, x') = \prod_{p=1}^{P} (\frac{2 l_p(x) l_p(x')}{l_p^2(x) + l_p^2(x')})^{1/2} \exp(-\sum_{p=1}^{P} \frac{(x_p - x'_p)^2}{l_p^2(x) + l_p^2(x')}), \tag{4.52}$$

in which, x_p is the p^{th} component of x.

Periodic kernel

Unlike the Gibbs kernel, there is another technique to create a valid non-stationary kernel. This technique is to map the inputs *via* a non-linear function $\boldsymbol{v}(x)$. After that, a stationary kernel is adopted in $\boldsymbol{v}$-space. [39] uses this transformation in a various manners to derive a stationary periodic kernel. The authors engaged the transformation $\boldsymbol{v} = (cos(x), sin(x))$, and then uses the SE kernel (4.45) in $\boldsymbol{v}$-space to retrieve:

$$k(x, x') = \exp(-\frac{2sin^2(x - x'/2)}{l^2}). \tag{4.53}$$

It is necessary to point out that the aforementioned periodic kernel actually keep behavior as periodic functions, this kernel is strictly positive, which is very familiar after investigating many stationary kernels above. When using this kernel, the expectation is to model the underlying periodic functions, like a sinusoid, to have negative covariances. The reason for this claim is that the peaks are anti-correlated with troughs. Usually, this kernel is coupled with other kernel to model the perspective of periodicity. For instance, the authors in [2] uses periodic kernel combined with SE and RQ kernels to model the oscillatory environment activity's data with a repeated trend.

Matérn kernel

Similar to the SE kernel, the Matérn kernel seems to be the second most popular kernel. The authors in [47] reasonably claim that in some cases, the smoothness property of the SE kernel is somehow unrealistic for modeling physical processes. Also in this research, the Matérn kernel is recommended as an appropriate substitute. Similar to SE kernel, the Matérn kernel can also be derived by modeling a power spectral density $S(\omega)$ as a t-distribution. This distribution certainly possesses an analytic solution. In frequency domain, the heavy tails of a t-distribution indicates that the corresponding power spectrum distributes more power to the high frequencies. Unlike SE kernel, the Matérn kernel are not infinitely differentiable.

Following is the general form of the Matérn kernel:

$$k_{Matérn}(x, x') = \frac{2^{1-\nu}}{\Gamma(nu)}(\frac{\sqrt{2\nu}|x - x'|}{l})^{\nu} K_{\nu}(\frac{\sqrt{2\nu}|x - x'|}{l}), \tag{4.54}$$

in which, K_ν is a modified Bessel function [48]. In case of one dimension, when $\nu + 1/2 = p$ for some natural number p, the corresponding GP equipped with Matérn kernel is a continuous time AR(p) process. By assigning $\nu = 1$, the Matérn kernel becomes the Ornstein-Uhlenbeck (OU) kernel:

$$k(x, x') = \exp(-\frac{x - x'}{l}), \tag{4.55}$$

This form of (4.55) is known to be the kernel of Ornstein-Uhlenbeck process [49]. This kernel introduces to the model the velocity of a particle following Brownian motion.

4.2.4 Mean function

It is a common sense in doing Gaussian process prediction to ignore the mean function. Usually, we make an assumption of zero mean by removing the empirical mean from the model. Nevertheless, considering the mean gives us an extra tool to encapsulate more prior beliefs into the modeling. Besides, it is also useful for getting important information from the dataset.

As an example, in the real problem, it is not easy to apply a general GP and Bayesian inference methods. Actually, the combination of these techniques does not particularly model any physical problem that might exist. All we have to do is to deal with the output data. Due to this reason, it would be more sensible to utilize parametric model for a specific system. After that, the parameter estimation is conducted to get the appropriate values for unknown parameters. This approach looks more attractive than the non-parametric models. Unfortunately, any parametric technique is going to be corrupted because of putting too much subjective into the model. Contrary to parametric techniques, the non-parametric approaches are much better in including the uncertainty to reasonably model the data. Clearly, the predictive result getting from this kind of modeling methods is better in terms of accuracy. Furthermore, the non-parametric approach such as Gaussian process can even reuse the aforementioned parametric techniques as the mean function. The remaining thing is to vary the prediction around the mean function by using the hyper-parameter of signal variance (σ_f^2 in the SE kernel of (4.45)). The amount for varying the value can be learned from the empirical data as a hyper-parameter. It is worth indicating that the unknown parameters of mean function can also be considered as the regular hyper-parameters in the marginal likelihood, which

can also be learned in the subsequent optimization step.

4.2.5 Prediction

In the next step, we evaluate the posterior distribution of the Gaussian process. Assuming that the incoming value of the input data is (x_*, y_*), the joint distribution of the training output is y, and the test output is y_*, as below:

$$p\left(\begin{bmatrix} y \\ y_* \end{bmatrix}\right) = \mathcal{GP}\left(\begin{bmatrix} m(x) \\ m(x_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(x,x') & \mathbf{K}(x,x_*) \\ \mathbf{K}(x_*,x) & \mathbf{K}(x_*,x_*) \end{bmatrix}\right) \tag{4.56}$$

in which, $\mathbf{K}(x_*, x_*) = k(x_*, x_*)$, $\mathbf{K}(x, x_*)$ is the column vector made from $k(x_1, x_*), k(x_2, x_*) \cdots, k(x_n, x_*)$. In addition, $\mathbf{K}(x_*, x) = \mathbf{K}(x, x_*)^\top$ can be retrieved by taking the transposition of $\mathbf{K}(x, x_*)$. In the next step, we can evaluate the posterior over y_*. But before that, the mean m_* and covariance C_* of this distribution can be estimated as below:

$$m_* = m(x_*) + \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}(y - m(x)) \tag{4.57}$$

$$C_* = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{4.58}$$

Then the probability distribution of predicted value is defined as:

$$p(y_*) \sim \mathcal{GP}(m_*, C_*) \tag{4.59}$$

We can achieve the best estimation value for the predictive output y_* by calculating the mean of probability distribution in (4.59):

$$\overline{y}_* = \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}y. \tag{4.60}$$

Additionally, we can also estimate the confident region of the predictive output by computing the

distribution's variance in (4.59):

$$var(y_*) = \mathbf{K}(x_*, x_*) - \mathbf{K}(x_*, x)\mathbf{K}(x, x')^{-1}\mathbf{K}(x, x_*) \tag{4.61}$$

Chapter 5

Improved Gaussian process regression

In this section, we would like to introduce our idea and proposal to improve the practical Gaussian process regression (GPR). This improvement is actuated by reducing the complexity in both phases of GPR: the hyper-parameters learning phase (HPLP) and the training phase. In HPLP, we would like to propose our improved technique based on the cooperation of fast Fourier transform, convergence law of log determinant and stochastic gradient descent. The role and benefit of each component would be discussed in detail in the next section. In the training phase, the 'divide and conquer' idea is introduced to partition the domain and locally compute the result. The parallelism, which is motivated by the *MapReduce* model, is also engaged to boost up the computation.

5.1 Hyper-parameters learning phase

5.1.1 Finding hyper-parameters

The proposed model invokes a set of hyper-parameter $\theta = [\sigma_f, l]$ which exists in covariance and mean function. Theoretically, these hyper-parameters are supposed to be evaluated through the marginalization process. This process is named as hyper-parameters learning phase (HPLP). By using the Bayes' theorem, the equation (4.59), which is in charge of probability distribution of predicted value, can be rewritten as:

$$p(y_*|y) = \frac{\int p(y_*|y,\theta)p(y|\theta)p(\theta)\mathrm{d}\theta}{\int p(y|\theta)p(\theta)\mathrm{d}\theta}. \tag{5.1}$$

In this equation, the marginal likelihood $p(y) = \int p(y|\theta)p(\theta)\mathrm{d}\theta$ is the main point of interest. Theoretically, the maximum a posteriori (MAP) estimation of θ can be obtained when $p(\theta|y)$

Table 5.1: Computation cost of proposed method compared with others

	Direct method	eGPR [3]	nGPR [15]	bGPR [6]	Proposed method
Hyper-parameters learning phase	$O(n^3)$	$O(n^2/6)$	$O(n^2)$	$O(nm^2)$	$O(nlogn)$
Training phase	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(m+n)$

reaches its maximum [50]. From the first moment of inference process, the prior $p(\theta)$ must be aware as per the hyper-parameters to reflect the domain knowledge. Based on the input data, this task is not an obstacle. In addition, according to the Bayes' theorem, the probability $p(\theta|y)$ is known to be proportional to $p(y|\theta)$. Therefore, the optimization step only involves maximizing the $\log p(y|\theta)$ or minimizing the negative $\log p(y|\theta)$ [51], which is described below:

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\log|\mathbf{K}| + \frac{n}{2}\log(2\pi). \tag{5.2}$$

The partial derivative of this negative marginal log likelihood with regard to each hyper-parameter is known as:

$$-\frac{\partial}{\partial\theta_i}\log p(y|\theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_i}\mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}tr(\mathbf{K}^{-1}\frac{\partial\mathbf{K}}{\partial\theta_i}). \tag{5.3}$$

As previously stated, estimating the set θ of hyper-parameters can be achieved by minimizing the negative marginal log likelihood. In other words, the set θ is evaluated by taking first-order partial derivatives of the negative marginal log likelihood and setting them to zero. This is the easy and trivial method to construct the learning scheme for hyper-parameters. Nonetheless, solving these derivative equations drives the whole computation into the state of worst-case execution time. The reason is high complexity in calculating the matrix inverse and the partial derivatives for every hyper-parameters. Therefore, it is necessary to determine another approach for this learning phase.

Instead of putting effort into minimizing negative marginal log-likelihood, this heavy-load-job can be done faster by approximating an adjacent value of this term [52]. In (5.2), the dominant computation focuses on two terms: the data-fit [2], which is denoted by $\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y}$; and the log-determinant (so-called the complexity penalty) $\log|\mathbf{K}|$. Before going further, the log-determinant should be studied to reveal the possibility of compacting the equation (5.2). As a side note, the

compactness of the corresponding equation not only affects the further derivations but also possesses role in partially reducing the complexity [53]. In the next section, the convergence law of log determinant, which is originated in [54], is engaged to compact the aforementioned equation.

5.1.2 Convergence law of log determinant

Central limit theorem

This section discusses the mathematical background for compacting the negative marginal log likelihood in (5.2). As a common problem, finding the determinant of the covariance matrix is a mandatory task not only in the area of regression, but also to build the hypothesis tests in multivariate statistics [55] [56]. In order to do that, a thorough understanding of the log determinant of sample covariance matrix is a requirement. Denote the dimension of the data as p, n is the size of dataset. It is important noting that the theorems in this section are extracted from the research for high-dimensional dataset [54] where $p \leq n$. Obviously, this research is also appropriate to utilized in our case, which the dimension of the dataset is only temporal-spatial ($p = 2$). Because of this fact, we would like to derive most of the theorems for using in case of $p = 2$.

Firstly, the central limit theorem for the log determinant of sample covariance matrix should be clarified. As a side note, the target of study is the covariance matrix of Gaussian distributions. Denote $X = X_1, ..., X_{n+1}$ as a set of random variable from the Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. Note that the elements of the set X are mutually independent. Based on this set, the sample covariance matrix can be built as follows:

$$\hat{\mathbf{K}} = \frac{1}{n}\sum_{k=1}^{n+1}(X_k - \overline{X})(X_k - \overline{X})^\top. \tag{5.4}$$

Subsequently, a central limit theorem can be established for the log determinant of $\hat{\mathbf{K}}$. In case when the size of dateset increases to large enough $\lim_{n\to\infty} {}^2\!/_n = 0$, the central limit theorem is that:

$$\frac{\log\det\hat{\mathbf{K}} - \sum_{k=1}^{2}\log(1-\frac{k}{n}) - \log\det\mathbf{K}}{\sqrt{-2\log(1-\frac{2}{n})}} \xrightarrow{L} \mathcal{N}(0,1) \text{ as } n \to \infty. \tag{5.5}$$

Primarily, the above result can be seen as the central limit theorem for log determinant for any arbitrary matrix with independent identical Gaussian distribution (*i.i.d*) entries. Apparently, this result is also possible to apply to the aforementioned covariance matrix. More information of this setup can be found on [57] [58]. Afterwards, an optimal estimation of the log determinant for covariance matrix is considered. This estimator would be constructed and investigated the corresponding properties. As a supplementary component, the upper bound for the mean squared error is also studied. Moreover, in order to complete the convergence law of log determinant, the convergence rate is also evaluated *via* proposing a minimax lower bound by using Cramer-Rao theory of information inequality. These properties also show the optimal aspect of the proposed estimator and the consistency of the estimation.

Limiting law of log determinant

Given a covariance matrix $\hat{\mathbf{K}}$ for a particular dataset, the limiting distribution of log determinant needs to be considered as a premise to construct the central limit theorem for $\log\det\hat{\mathbf{K}}$. Assume the size of dataset is n, we define a bias correction τ_n as

$$\tau_n := \sum_{k=1}^{2}\left(\psi\left(\frac{n-k+1}{2} - \log\left(\frac{n}{2}\right)\right)\right), \tag{5.6}$$

in which, $\psi(x) = \frac{\partial}{\partial z}\log\Gamma(z)|_{z=x}$ is the Di-gamma function ($\Gamma(z)$ is the gamma function), and define a constant σ_n by

$$\sigma_n = \sqrt{\left(\sum_{k=1}^{2}\frac{2}{n-k+1}\right)}. \tag{5.7}$$

The central limit theorem for $\log det\hat{\mathbf{K}}$ can be stated as follows:

Theorem 5.1.1 *(Asymptotic Distribution): Denote an i.i.d set $X = X_1, ..., X_{n+1}$ with $X_i \sim \mathcal{N}(\mu, \sigma), i = 1, ..., n+1$. Assume that $n \to \infty$, the log determinant of covariance matrix $\hat{\mathbf{K}}$ is ensured to satisfy*

$$\frac{\log\det\hat{\mathbf{K}} - \tau_n - \log\det\mathbf{K}}{\sigma_n} \xrightarrow{L} \mathcal{N}(0,1) \text{ as } n \to \infty. \tag{5.8}$$

in which, the bias correction τ_n and the constant σ_n are given in (5.6) and (5.7), respectively. It is worth mentioning that the assumption in Theorem 5.1.1 is generally weak. For instance, the existence of the limit: $\lim_{n\to\infty} 2/n = 0$ is not a mandatory condition for the correct of this theorem. In the case of temporal-spatial dimension, the log determinant of $\hat{\mathbf{K}}$ satisfies

$$\frac{\log\det\hat{\mathbf{K}} - \frac{3}{n} - \log\det\mathbf{K}}{\sqrt{\frac{4}{n}}} \xrightarrow{L} \mathcal{N}(0,1) \text{ as } n \to \infty. \tag{5.9}$$

5.1.3 Properties of convergence law

Estimation of log determinant

As discussed above, finding the log determinant of covariance matrix can be seen as one of the most important tasks in various area of research. Due to this fact, it is a need for deriving an optimal estimation for the log determinant of Gaussian distributions. Besides, the corresponding minimax upper and lower bounds also need to be investigated. Based on the original research [54], a claim can be stated that the final results can be evaluated as really accurately asymptotic minimaxity.

Assume that our data set can be described as an *i.i.d* set $X = X_1, ..., X_{n+1}$ with $X_i \sim \mathcal{N}(\mu, \sigma), i = 1, ..., n+1$. By utilizing the central limit theorem for $\log\det\hat{\mathbf{K}}$ which is proposed in Theorem 5.1.1, the estimator for log determinant $L = \log\det\mathbf{K}$ of covariance matrix $\mathbf{K}$ can be built as:

$$\hat{L} = \log\det\hat{\mathbf{K}} - \tau_n. \tag{5.10}$$

Upper bound

Theorem 5.1.2 *(Upper Bound): Denote the estimator $\hat{L}$ as being defined in (5.10). Afterwards, the risk of $\hat{L}$ is ensured to satisfy*

$$\mathbb{E}\left(\hat{L} - \log\det\mathbf{K}\right)^2 \leq -2\log(1 - \frac{2}{n}) + \frac{20}{3n}\frac{1}{n-2}. \tag{5.11}$$

The proof of this theorem can be found in the original paper [54]. By connecting to Theorem

5.1.1, we come to a conclusion as:

$$\mathbb{E}\left(\hat{L} - \log\det\mathbf{K}\right)^2 \sim \sigma_n^2 = \sum_{k=1}^{2} \frac{2}{n-k+1} \leq -2\log(1-\frac{2}{n}), \tag{5.12}$$

which claims the dominant term in (5.11). The higher order term on the right hand side of (5.11) can be executed explicitly by engaging the Taylor expansion.

Rate of convergence

As stated previously, Theorem 5.1.2 allows us to define an upper bound for the risk of estimator $\hat{L}$. Moreover, it is possible to look for the optimal rate of convergence for estimating $\log\det\mathbf{K}$. This possibility is actuated by achieving a minimax lower bound, which can be found by engaging the Cramer–Rao information inequality as follows:

Theorem 5.1.3 *(Information Bound): Assume there is an i.i.d set* $X = X_1, ..., X_{n+1}$ *such that* $X_i \sim \mathcal{N}(\mu, \sigma), i = 1, ..., n+1$. *Afterwards, the minimax risk for* $\log\det\mathbf{K}$ *estimation is ensured to satisfy*

$$\inf_{\delta}\sup_{\mathbf{K}} \mathbb{E}(\delta - \log\det\mathbf{K})^2 \geq \frac{4}{n}. \tag{5.13}$$

It is worth noting that the infimum is calculated over all measurable estimators δ. Meanwhile, the supremum is calculated by evaluating all the possible positive definite covariance matrix $\mathbf{K}$. One conclusion can be extracted from the Theorem 5.1.3 is that that the estimator $\hat{L}$ can achieve an asymptotically accurate minimax.

5.1.4 Technique of improvement

Because the interval time for collecting samples is non-overlapped, periodic and identical, these random samples are independent and identically distributed (i.i.d.). It means that the dimension can be reduced from the spatial-temporal to the spatial only [59]. Therefore, the definition of sample covariance matrix from (5.4) and the bias correction from (5.6) can be reused. After determining these two important values, the log determinant $L = \log|\mathbf{K}|$ calculating on the covariance

matrix $\mathbf{K}$ can be estimated as:

$$\hat{L} = \log|\hat{\mathbf{K}}| - \tau_n. \tag{5.14}$$

This estimation is exactly compatible with the log determinant estimator, which is described in (5.10). In addition, the upper bound of this estimation is also evaluated according to the convergence law of log determinant as follows:

$$\mathbb{E}\left(\hat{L} - L\right)^2 \sim \sigma_n^2 = \sum_{k=1}^{2} \frac{2}{n-k+1} \leq -2\log(1-\frac{2}{n}). \tag{5.15}$$

Another aspect that needs to be mentioned is that within the above condition of upper bound, the estimation in (5.14) is proven in previous section to converge into a fixed term in the spatial dimension. For the derivations on high dimensional space, the evidences are described in detail in [54]. Due to this fact, the converged estimation can be adopted as a replacement for the original log determinant. Consequently, the equation (5.2) is simplified to:

$$-\log p(y|\theta) = \frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y} + \frac{1}{2}\hat{L} + \frac{n}{2}\log(2\pi). \tag{5.16}$$

in which, in addition to the constant $\frac{n}{2}\log(2\pi)$, when the process runs for enough number of cases guaranteed by 5.15, the term $\hat{L}$ converges to a fixed term. Therefore, it makes sense to state that the partial derivative of negative marginal log likelihood in the equation (5.3) is mutually simplified by dropping these constants:

$$-\frac{\partial}{\partial\theta_i}\log p(y|\theta) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial\theta_i}\mathbf{K}^{-1}\mathbf{y}. \tag{5.17}$$

Interestingly, this derivative indicates that the negative marginal log likelihood should only involve minimizing the following reduced negative marginal log likelihood estimation (rMLL):

$$-\log p(y|\theta)_{rMLL} = \frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y}. \tag{5.18}$$

Traditionally, dealing with this task concerns inversing the covariance matrix $\mathbf{K}$. This matrix

operation normally costs $O(n^3)$, which is very computationally expensive. Nonetheless, this effect can be mitigated by applying an appropriate solution. Motivated by the application in the stochastic frontier model [50] and Kriging problem [60], the fast Fourier transform (FFT) is a promising tool for mitigating this complexity. As previously mentioned, the kernel function is a positive-definite function. Thus, FFT makes it possible to transform this kernel in order to bring the computation from the spatial-temporal domain (or previously reduced spatial domain) into the frequency domain. After that, the most expensive task is not the matrix inverse, but rather calculating the power spectrum (the quantity shows how much of the signal is at the frequency ω), which only costs $O(nlogn)$. This cost is much less expensive and can be computed faster than the aforementioned traditional approach.

Mixture problem

In order to achieve this advantage, first the squared exponential kernel $k_{SE}(x, x')$ in the equation (4.35) needs to be rewritten in Fourier transform representation [61] as shown below:

$$\mathcal{F}_{SE}(\omega) = \sigma_s^2 \exp(\frac{-2\pi(\omega - \omega_0)}{l_s^2}), \tag{5.19}$$

in which, ω is the frequency representation of the time location x, ω_0 is the specific frequency with the peak power, σ_s^2 is the power scaling factor, l_s is the overall bandwidth in the frequency domain. To accelerate the optimization procedure, the uniform fast Fourier transform (UFFT) can be applied due to the identical interval of sampling.

Essentially, it is compulsory to investigate the relationship of hyper-parameters between time domain and frequency domain. In the time domain, the set of hyper-parameters consists of $[\sigma_f^2, l_f]$ as seen in (4.45), which stand for signal variance and length-scale, respectively. By definition, σ_f^2 is actually a scaling factor. This hyper-parameter determines the variation of function $f(x)$ from the mean. Small value of σ_f^2 characterize functions that stay close to mean value, larger values allow more variation. In short, the signal variance σ_f^2 controls the amplitude of signal. In the other hand, length-scale l_f describes how smooth the function $f(x)$ is. Small length-scale value means that function values can change quickly, large values characterize functions that change only slowly. Length-scale also determines how far we can reliably extrapolate from the training

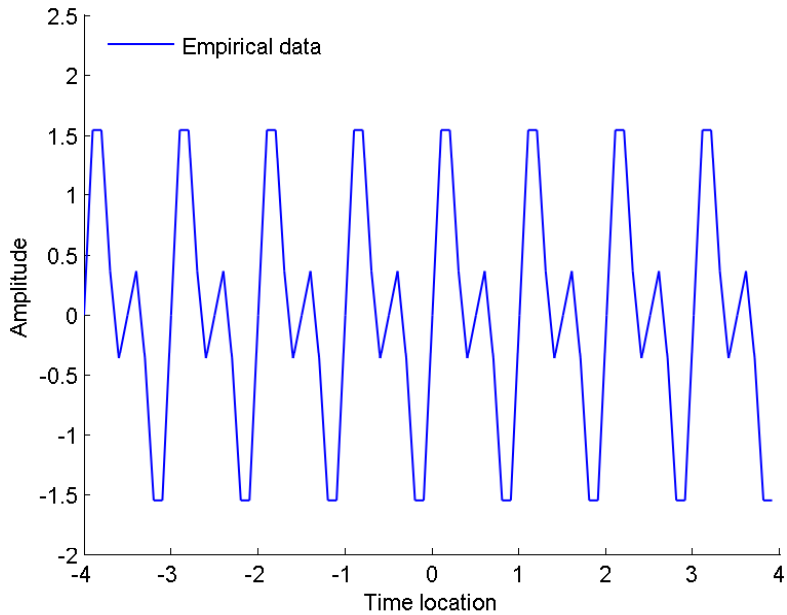
data. In the frequency domain, the corresponding hyper-parameters are $[\sigma_s^2, l_s]$ as shown above. The relationship between these hyper-parameters is as follows:

$$\begin{aligned} \sigma_s^2 &= \frac{\sqrt{2\pi}\sigma_f^2}{l_s}, \\ l_s &= \frac{1}{l_f}. \end{aligned} \tag{5.20}$$

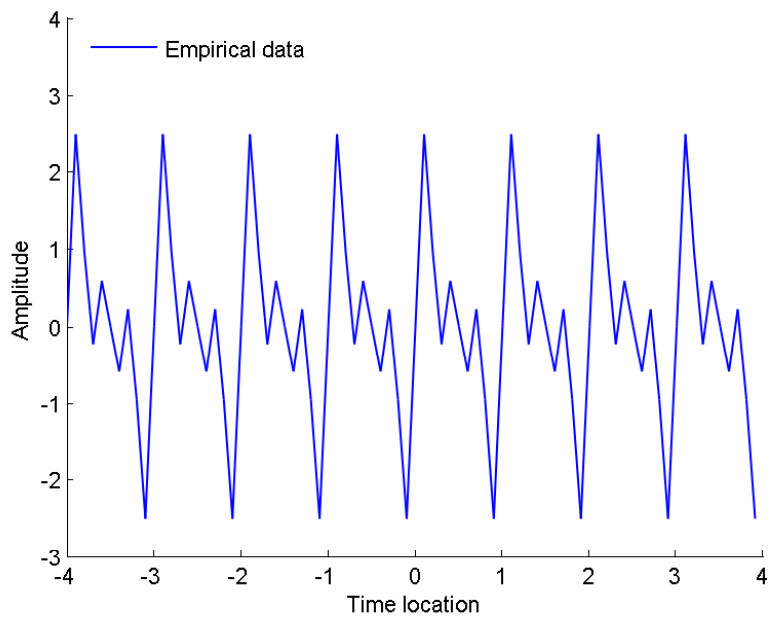
In (5.20), it is intuitive to see that the bandwidth l_s is exactly the inverse length-scale of l_f. This length-scale makes the Fourier transform of Gaussian kernel act like a low-pass filter. Basically, the bandwidth l_s determines the bandwidth or the range of the considered frequencies. This hyper-parameter shows the power correlation between each frequency and the peak one - ω_0 (the frequency with the peak power). It means that the frequencies closer to the peak would receive more power than the frequencies that stay far. As a side note, within a fixed overall power, smaller bandwidth results in stronger power that each frequency inside that bandwidth might receive, vice versa. Meanwhile, the role of σ_f^2 and σ_s^2 is pretty similar, which is the scaling factor. In frequency domain, variances σ_s^2 scale the power that each signal might receive with regard to the peak powers.

This equivalent kernel in (5.19) can work properly if the empirical dataset represents signals according to the model of DC power source (only one frequency with peak power) as shown in Figure 3.4. Note that we can treat the underlying function in signal processing fashion - as the power source for creating the signal. In this figure, the power spectral density (PSD) explains the distribution of power, which is proportional to the magnitude of frequencies described in Figure 3.3. Note that the magnitude of frequencies can be retrieved by conducting the Fourier transform on the monitoring data. The transformation would bring the corresponding statistics, which is depicted in Figure 3.2, from the time domain to the frequency domain. Indeed, the model of DC power actually maps perfectly with most of monitoring statistics that we deal with. However, there is a chance that the underlying functions might follow the model of multiple power sources. Let consider the case of following complex underlying functions:

$$f(\boldsymbol{x}) = \sin(2\pi x) + \sin(4\pi x) + \epsilon, \tag{5.21}$$

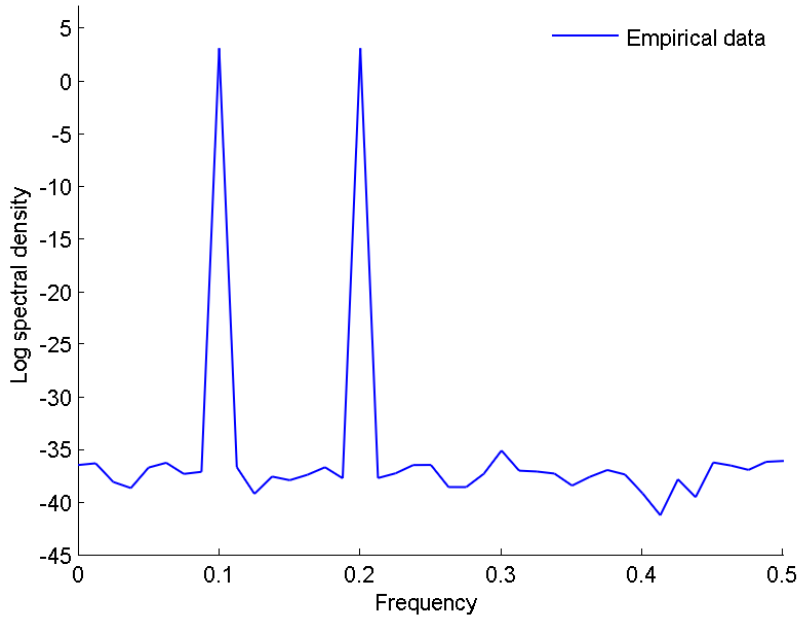


(a) Two sin functions

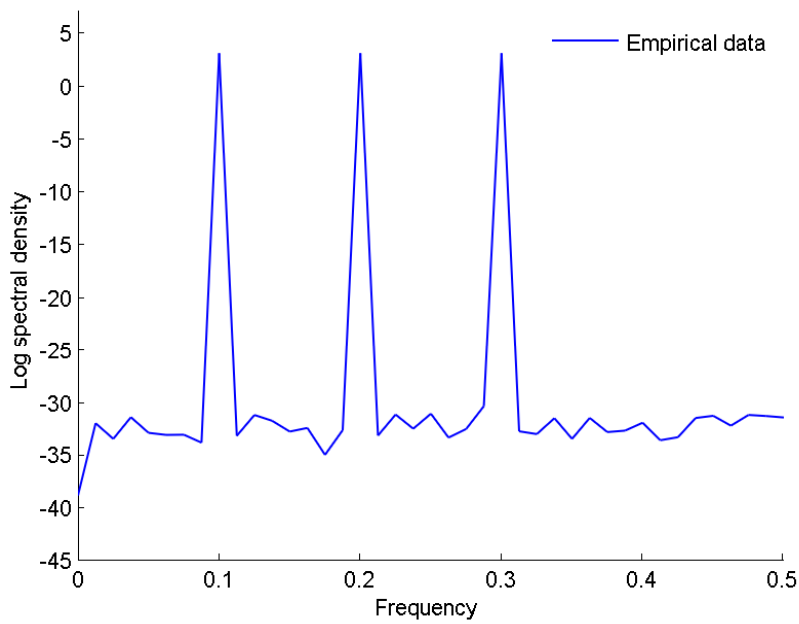


(b) Three sin functions

Figure 5.1: Empirical data produced by the complex underlying combined functions.



(a) Two sin functions



(b) Three sin functions

Figure 5.2: Power spectral density of complex empirical data.

$$f(\boldsymbol{x}) = \sin(2\pi x) + \sin(4\pi x) + \sin(6\pi x) + \epsilon, \tag{5.22}$$

The complex function in (5.21) is a combination of two sin functions plus a small white Gaussian noise ϵ. The data, which is produced by this complex function, is shown in Figure 5.1a. Investigating the data by taking Fourier transform on the mentioned complex data, the retrieved power spectral density reveals that there are two power sources, which focus most of their power on frequencies 1Hz and 2Hz, as shown in Figure 5.2a. Moreover, the complex function can be extended to three sin functions in (5.22), with corresponding data and PSD shown in Figure 5.1b and 5.2b, respectively. It is intuitive to see that there are three power sources according to the number of peak frequencies (focus on 1Hz, 2Hz, and 3Hz). In these cases, if using only the SE kernel and the corresponding PSD in (5.19), the model might not cope with the empirical data properly. Indeed, the original problem has been extended to Gaussian mixture problem. It means that we have to optimize the equivalent hyper-parameters to relatively fit the predictive mixture spectral density to the empirical PSD in the frequency domain, instead of extrapolating or interpolating the predictive data in time domain. Because the original problem has been changed, the corresponding kernels should be adapted to properly model the new data. In order to fulfill this requirement, we have to extend the original kernel to the mixture Gaussian kernel, which has the following PSD form in the frequency domain:

$$\mathcal{F}_{SE}(\omega) = \sum_{j=1}^{J} \sigma_{sj}^2 \exp(\frac{-2\pi(\omega - \omega_j)}{l_{sj}^2}) \tag{5.23}$$

in which, ω_j is the frequencies with peak power. Actually, J number of ω_j decides the number of components would be used to formulate the final mixture PSD. The power variance σ_{sj} and the bandwidth l_{sj} are the hyper-parameters of each component, respectively. In order to exchange the hyper-parameters between domains, the transformation in (5.20) can be reused. As a side note,

the PSD mentioned above is equivalent to the mixture Gaussian kernel in the time domain:

$$k(x, x') = \sum_{j=1}^{J} \sigma_{fj}^2 \exp(-\frac{||x - x'||^2}{2l_{fj}^2}), \tag{5.24}$$

By using the extended mixture kernel and the corresponding PSD, we can model the complex data and then conduct the regression. The result of this experiment is shown later in the chapter of performance evaluation.

Objective transformation

The next step is to derive the objective function of (5.18) to the frequency domain by using Fourier transform [62]:

$$\mathcal{F}_{rMLL}(\theta) = \mathcal{F}\big(- \log p(y|\theta)_{rMLL}\big) = \frac{1}{2N} \hat{\mathbf{y}}^\top \widehat{\mathbf{K}^{-1} * \mathbf{y}}_\circ, \tag{5.25}$$

in which, the hat sign from $\hat{\mathbf{y}}$ denotes a Fourier transform of $\mathbf{y}$ and $\mathbf{y}_\circ$ denotes the data vector in the periodic nature of discrete Fourier transform (DFT) [63]. For example, given a sequence of data which consists of 128 points. The DFT views these points to be a single period of an infinitely long periodic signal. This means that the left side of the acquired signal is connected to the right side of a duplicate signal. Likewise, the right side of the acquired signal is connected to the left side of an identical period. This can also be thought of as the right side of the acquired signal wrapping around and connecting to its left side. In this view, sample 127 occurs next to sample 0, just as sample 43 occurs next to sample 44. This is referred to as being circular and is identical to viewing the signal as being periodic.

In order to do the discrete Fourier transform in (5.25), a supportive component related to the inverse matrix $\mathbf{K}^{-1}$ should be derived in advance. In common sense, to solve the problem of inverting covariance matrix, the conventional approach tends to decompose the matrix $\mathbf{K}$. In our approach, because we intend to solve the objective function in the frequency domain, the aforementioned supportive component should be also developed in this domain. Let's make an

observation as follows:

$$\mathbf{K}\mathbf{K}^{-1} = \mathbf{I}, \tag{5.26}$$

in which, $\mathbf{I}$ is the identity matrix. Equation (5.26) defines the matrix $\mathbf{K}^{-1}$ as the inverse of matrix $\mathbf{K}$. This definition is a short-hand notation for the following summation:

$$\sum_j k_{ij} k'_{jm} = \delta_{im}, \tag{5.27}$$

in which, k_{ij} and k'_{jm} are the elements of $\mathbf{K}$ and $\mathbf{K}^{-1}$, respectively, and δ_{im} is Kronecker delta. At this moment, we use three continuous variables that correspond to each of the three indices [64], say:

$$\begin{aligned} i &\to x, \\ j &\to y, \\ m &\to z. \end{aligned} \tag{5.28}$$

Then, the matrix elements in (5.27) correspond to following definitions in continuous manner [65]

$$\begin{aligned} k_{ij} &\to k(x-y), \\ k'_{jm} &\to k'(y-z), \\ \delta_{im} &\to \delta(x-z). \end{aligned} \tag{5.29}$$

Subsequently, the summation over j becomes the integration over y in the continuous space as below:

$$\int_{\mathbb{R}} k(x-y)k'(y-z)dy = \delta(x-z). \tag{5.30}$$

Substitute $u = y - z$, then $du = dy$. It is worth noting that the limits do not change as u still

covers all of $\mathbb{R}$. Moreover, we also have $x - y = x - z - u$, so (5.30) becomes

$$\int_{\mathbb{R}} k(x - z - u)k'(u)du = \delta(x - z). \tag{5.31}$$

Once again, substitute $h = x - z$ and we have:

$$\int_{\mathbb{R}} k(h - u)k'(u)du = \delta(h). \tag{5.32}$$

It is clear that the left hand side of (5.32) is the definition of linear convolution of k and k'. Due to this fact, (5.32) is reduced to:

$$\{k * k'\}(h) = \delta(h), \tag{5.33}$$

in which, h is taking over the role of i, j, k; and $*$ denotes the convolution operator. We transform (5.33) by using Fourier transform. As a side note, according to the convolution theorem [66], the convolution $*$ is now replaced by the scalar multiplication in the frequency domain as follows:

$$\hat{k}(\omega)\hat{k'}(\omega) = 1. \tag{5.34}$$

Furthermore, equation (5.34) can be rewritten to:

$$\hat{k'}(\omega) = \frac{1}{\hat{k}(\omega)}. \tag{5.35}$$

in which, $\hat{k}(\omega)$ is exactly the equivalent squared exponential kernel as shown in (5.19). Subsequently, (5.35) becomes:

$$\hat{k'}(\omega) = \frac{1}{\mathcal{F}_{SE}(\omega)}. \tag{5.36}$$

Now, it is time to take a look at the Fourier transform of the objective function in (5.25). By applying the convolution theorem to the last term, we can adopt (5.25) in the frequency domain as

Table 5.2: Transformation of objective function

	Time domain	Frequency domain
Discrete	$-\frac{1}{2}\boldsymbol{y}^\top \mathbf{K}^{-1}\boldsymbol{y}$ $\mathbf{K}\mathbf{K}^{-1} = \mathbf{I}$	$-\frac{1}{2N}\sum_n \hat{k}'(\omega_n)\hat{y}_n^2$ $\hat{k}(\omega_n)\ \hat{k}'(\omega_n) = 1$
Continuous	$-\frac{1}{2}\int \overline{y}(h)\{k' * y\}(h)dh$ $\{k\ k'\}(h) = \delta(h)$	$-\int \hat{k}'(\omega)\hat{y}^2(\omega)d\omega$ $\hat{k}(\omega)\ \hat{k}'(\omega) = 1$

shown below:

$$\mathcal{F}_{rMLL}(\theta) = -\frac{1}{2}\int \hat{k}'(\omega)\hat{y}^2(\omega)d\omega. \tag{5.37}$$

As a side note, one properties of discrete Fourier transform (DFT), which is the Parseval's theorem [67], states that the power contained in a signal $\mathbf{y}$ is the same as the power contained in the spectrum $\hat{\mathbf{y}}$. This statement can be expressed in mathematical form as below:

$$\mathbf{y}^2 = \frac{1}{N}\hat{\mathbf{y}}^2. \tag{5.38}$$

By replacing (5.36) to (5.37), we have the continuous form of rMLL:

$$\mathcal{F}_{rMLL}(\theta) = -\frac{1}{2}\int \frac{\hat{y}^2(\omega)}{\mathcal{F}_{SE}(\omega)}d\omega. \tag{5.39}$$

We can get the discrete form of rMLL by discretizing (5.39). Finally, we retrieve the desired version of rMLL, which is as follows:

$$\mathcal{F}_{rMLL}(\theta) = -\frac{1}{2N}\sum_n \frac{\hat{y}_n^2}{\mathcal{F}_{SE}(\omega_n)}. \tag{5.40}$$

The summary of transformation of objective function can be found at Table (5.2). With this form of the equation (5.40), it is no longer expensive to determine the set of hyper-parameters by

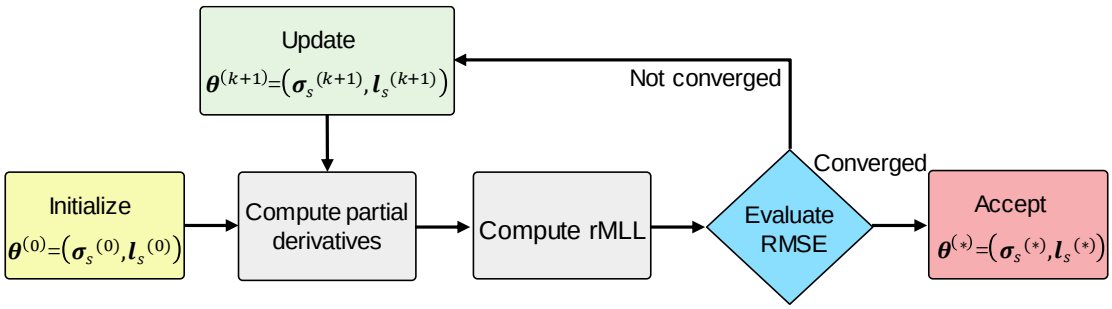


Figure 5.3: Logic of hyper-parameters learning phase.

using gradient-based optimizing techniques. In this research, we choose the Stochastic Gradient Descent (SGD) for the optimization step. The main reason is that this technique is more suitable for the large dataset, faster than other gradient techniques, and critically less sensitive to the local minima [68]. To integrate SGD into hyper-parameters learning phase, the partial derivatives of the equation (5.40) with regard to each hyper-parameter are required. These partial derivatives can be calculated as:

$$\frac{\partial}{\partial l_{sj}}\mathcal{F}_{rMLL} = -\frac{1}{2N}\sum_{n}^{N}\hat{y}_n^2 \exp\left(\frac{-2\pi^2(\omega_n-\omega_j)^2}{l_{sj}^2}\right)\left(\frac{4\pi^{5/2}(\omega_n-\omega_j)^2 - l_{sj}^2}{l_{sj}\sigma_{sj}^2}\right), \tag{5.41}$$

and

$$\frac{\partial}{\partial \sigma_{sj}}\mathcal{F}_{rMLL} = -\frac{1}{2N}\sum_{n}^{N}2^{5/4}\pi^{1/4}\hat{y}_n^2\frac{\exp\left(\frac{-2\pi^2(\omega_n-\omega_j)^2}{l_{sj}^2}\right)}{\sigma_{sj}^3\sqrt{l_{sj}}}. \tag{5.42}$$

After getting the necessary partial derivatives, an updating scheme is issued to update the hyper-parameters. This scheme is as follows.

$$\boldsymbol{l}_s^{(k)} \leftarrow \boldsymbol{l}_s^{(k-1)} + \alpha(k)\frac{\partial}{\partial \boldsymbol{l}_s^{(k-1)}}\mathcal{F}_{rMLL}, \tag{5.43}$$

$$\boldsymbol{\sigma}_s^{(k)} \leftarrow \boldsymbol{\sigma}_s^{(k-1)} + \alpha(k)\frac{\partial}{\partial \boldsymbol{\sigma}_s^{(k-1)}}\mathcal{F}_{rMLL}, \tag{5.44}$$

Algorithm 1: Hyper-parameters learning phase

Data: Utilization Statistic of CPU Core. This is the latest history of utilization of each core with regard to time step

Result: Hyper-parameters array $\boldsymbol{\theta}^{(*)} = [\boldsymbol{l}_s^{(*)}, \boldsymbol{\sigma}_s^{(*)}]$

Initialize value for $\boldsymbol{\theta}^{(0)} = [\boldsymbol{l}_s^0, \boldsymbol{\sigma}_s^0], \omega_{[j]}, \epsilon_{RSME}$;

/* Fast Fourier Transform of input data */

$\hat{y}$= ufft1d1(y);

for *k=1* **to** *sizeof($\hat{y}$)* **do**

/* step_size is equivalent to α in the equation (5.43) and (5.44) */

step_size=decay_function(k);

n=random(1,*sizeof*($\hat{y}$));

/* partial derivative of $\mathcal{F}_{rMLL}$ w.r.t $\boldsymbol{l}_s$ */

$\nabla \boldsymbol{l}_s =$ partial_$\boldsymbol{l}_s(\hat{y}_{[n]}, \omega_{[n]}, \omega_{[j]}, \boldsymbol{l}_s^{(k-1)}, \boldsymbol{\sigma}_s^{(k-1)})$;

/* partial derivative of $\mathcal{F}_{rMLL}$ w.r.t $\boldsymbol{\sigma}_s$ */

$\nabla \boldsymbol{\sigma}_s =$ partial_$\boldsymbol{\sigma}_s(\hat{y}_{[n]}, \omega_{[n]}, \omega_{[j]}, \boldsymbol{l}_s^{(k-1)}, \boldsymbol{\sigma}_s^{(k-1)})$;

/* update hyper-parameters */

$\boldsymbol{l}_s^{(k)} = \boldsymbol{l}_s^{(k-1)}$+step_size*$\nabla \boldsymbol{l}_s$;

$\boldsymbol{\sigma}_s^{(k)} = \boldsymbol{\sigma}_s^{(k-1)}$+step_size*$\nabla \boldsymbol{\sigma}_s$;

Compute $\mathcal{F}_{rMLL}^{(k)}(\boldsymbol{\theta}^{(k)})$;

Compute $RMSE^{(k)} = RMSE(\mathcal{F}_{rMLL}^{(k)})$;

if *($RMSE^{(k)} \leq \epsilon_{RSME}$)* **then**

break();

end

end

return $\boldsymbol{\theta}^{(*)} = [\boldsymbol{l}_s^{(*)}, \boldsymbol{\sigma}_s^{(*)}]$;

in which, $\alpha(k)$ is the decay function with regard to the k^{th} iteration. We opt to use the decay function instead of the exact line search or backtracking line search. It is mainly due to the performance issue. For the ease of calculation, a Robbins-Monroe sequence [69] is employed to construct the decay function $\alpha(k) = 1/(k+1)$. In fact, the Robbins-Monroe sequence is popularly used, since it is sufficient to ensure the convergence of the optimization algorithm [70], especially in the SGD method.

To govern the number of iteration for the optimization algorithm (in this case, SGD), an error function is defined based on the Root Mean Square Error (RMSE) method to measure the convergence. It is important to note that the RMSE method is stricter than the frequently-used

Mean Square Error (MSE) method. By using this error function, the error gap between the current iteration value and the previous one can be evaluated as follows:

$$RMSE = \sqrt{\frac{(\sum_{i=1}^{n} \mathcal{F}_i^{(k)} - \mathcal{F}_i^{(k-1)})^2}{n}}, \tag{5.45}$$

in which, $\mathcal{F}_i^{(k)}$ and $\mathcal{F}_i^{(k-1)}$ respectively stand for the k^{th} and $k-1^{th}$ iteration values of rMLL at the target location i. Theoretically, RMSE threshold is limited to 10^{-11} which produces an adjacent solution to the real one. The purpose of this optimization procedure is to conduct all the steps in the periodic domain. It means that the optimization can be done with no matrix inverse. Additionally, the vector of dual weight $\mathbf{y}_0 \approx \mathbf{\Phi} * \mathbf{y}_\circ$ can also be easily estimated in the spectral domain. In the end of this hyper-parameters learning phase, the set of hyper-parameters is ready for the training phase. Intuitively, the hyper-parameters learning algorithm is also described in Algorithm 1 and Figure 5.3.

5.2 Training phase

5.2.1 Solving linear system

In the training phase, most of the computation involves determining the mean value in the equation (4.60). In this equation, once the kernel matrix is known, the matrix inverse becomes the main problem. In fact, dealing with the matrix inverse is one of the most intensive computing tasks in optimization. Although the Cholesky decomposition is usually employed to avoid doing the matrix inversion directly, the computational complexity is still $O(n^3/6)$, where n is the number of training point of the dataset. In addition, $O(n^2)$ is also taken into account for matrix storage. This issue is a significant bottleneck for the system. Because of that, it is necessary to search for a more suitable technique, which is feasible to calculate the prediction. The procedure to establish such kind of method is introduced in this section.

First of all, the equation (4.60) needs to be rewritten as:

$$\overline{y}_* = \mathbf{K}(x_*, x)\xi, \tag{5.46}$$

Algorithm 2: Conjugate gradient algorithm for solving linear system

Data: vector y, covariance matrix $\mathbf{K}$
Result: ξ_* such that $y = \mathbf{K}\xi_*$

$\xi_0 = 0$, $r_0 = y - \mathbf{K}\xi_0$;
$j = 0$;
While ($r_j > 0$) **do**
/* Step 1: parameters establishment */
    if($j == 0$)
        $p_j = r_0$;
        $\beta = 0$;
    else
        $\beta = \frac{||r_j||_2^2}{||r_{j-1}||_2^2}$;
        $p_j = r_j + \beta p_{j-1}$;
/* Step 2: core computation */
    $\alpha = \frac{||r_j||_2^2}{p_j^\top \mathbf{K} p_j}$;
    $\xi_{j+1} = \xi_j + \alpha p_j$;
    $r_{j+1} = r_j + \alpha \mathbf{K} p_j$;
    $j = j + 1$;
$\xi_* = \xi_j$;
return ξ_*

in which

$$\xi = \mathbf{K}(x, x')^{-1} y. \tag{5.47}$$

Multiplying both sides with $K(x, x')$

$$y = \mathbf{K}(x, x')\xi \tag{5.48}$$

The parameter of interest is ξ can be found through solving (5.48). Observe that (5.48) is nothing more than just a linear system. More important, the matrix $\mathbf{K}(x, x')$ is known to be symmetric and positive definite, the conjugate gradient [71] iterative method is engaged to solve this linear problem. The step-by-step guidance of the corresponding solution can be found in Algorithm 2. In this solution, a starting point is chosen arbitrarily and a series of steps are created to converge upon the approximation ξ_i which is adjacent to the real one ξ. The rate of convergence

to the best solution of this process can be given by the inequality below:

$$\frac{\|\xi - \xi_i\|_{\mathbf{K}}}{\|\xi - \xi_0\|_{\mathbf{K}}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2i} \tag{5.49}$$

in which, the constant $\kappa = \lambda_{max}/\lambda_{min}$ is the ratio of the largest to the smallest eigenvalue of matrix $\mathbf{K}$, and the $\mathbf{K} - norm$ is calculated as $\|z\|_{\mathbf{K}} = z^T\mathbf{K}z$ with z is any arbitrary vector. The tolerance parameter ζ is also given such that $0 < \zeta < 1$. This parameter is the upper bound for the practical conjugate-Gradient scheme.

$$\frac{\|y - \mathbf{K}\xi_i\|_2}{\|y - \mathbf{K}\xi_0\|_2} \leq \zeta \tag{5.50}$$

in which, at the end of the i^{th} iteration, $\|y - \mathbf{K}\xi_i\|_2$ is obtained as the residual in the Euclidean norm. Regularly, the starting point of the iteration process is ξ_0. Then, the relative error is also obtained as shown below:

$$\frac{\|y - \mathbf{K}\xi_i\|_2}{\|y - \mathbf{K}\xi_0\|_2} \leq \sqrt{\kappa}\frac{\|\xi - \xi_i\|_{\mathbf{K}}}{\|\xi - \xi_0\|_{\mathbf{K}}} \leq 2\sqrt{\kappa}\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2i} \tag{5.51}$$

In fact, a number of iteration is done to achieve the given tolerance parameter ζ. This number can be evaluated as shown below.

$$i \geq \frac{\ln\left(\frac{2\sqrt{\kappa}}{\zeta}\right)}{2\ln\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)} \tag{5.52}$$

5.2.2 Divide stage

The complexity of the conjugate gradient method is $O(in^2)$, in which i is the number of iteration, n is the number of training point of dataset. The storage cost is $O(n)$, due to the fact that the matrix-vector product is able to do the calculation without retaining the whole matrix. The cost of computation and storage in this method is less than the aforementioned direct matrix operation. However, it still does not satisfy the computational intensity of the large-scale system. In fact, the system usually works on an enormous number of dataset. In this case, the quadratic complexity

algorithm could deteriorate the overall performance rapidly. Therefore, the conjugate gradient needs to be coupled with the improved fast Gauss transform (IFGT) method [72] to achieve even faster calculation. The IFGT technique is actually derived from the fast Gauss transform (FGT) [73] [74] [75], which is an $\epsilon - exact$ approximate algorithm. According to the Algorithm 2, in the j^{th} step of the conjugate gradient, most of the computation focuses on finding a matrix-vector product $\mathbf{K}p_j$. Theoretically, the product $\mathbf{K}p_j$ can be transformed by utilizing discrete Gauss tranform (DGT) as seen in (5.53):

$$\mathbf{K}p_j \triangleq G(x_j) = \sum_{i=1}^{N} q_i \exp(-\frac{\|x_j - x_i\|^2}{2l^2}) \tag{5.53}$$

in which, with m is the number of target point and n is the number of source point, x_j is the target point with $\{x_j \in \mathbb{R}^2\}_{j=1,...,m}$, q_i is the source weight with $\{q_i \in \mathbb{R}\}_{i=1,...,n}$, x_i is the source point with $\{x_i \in \mathbb{R}^2\}_{i=1,...,n}$, and l is the bandwidth with $\{l \in \mathbf{R}^+\}$. Technically, the DGT method expands the j^{th} conjugacy (the A-orthogonal multiplication [71]) $G(x_j)$ into a plane-wave expansion of the previous squared exponential kernel (the Matérn kernel might be expanded similarly). In fast Gauss transform, this Gaussian-type expansion can be calculated approximately by using the fast Fourier transform [74] [76]:

$$\mathbf{K}p_j \triangleq G(x_j) \approx \sum_{|\alpha| \leq p} \mathcal{F}(\alpha) w_\alpha \exp(\frac{i\alpha L(x_j - x_i)}{\sqrt{2}pl}) \tag{5.54}$$

with $\mathcal{F}(\alpha)$ and w_α are given by

$$\mathcal{F}(\alpha) = \frac{1}{2^3\sqrt{\pi}} \exp(-\frac{L^2|\alpha|^2}{4p^2}) \tag{5.55}$$

$$w_\alpha = \left(\frac{L}{p}\right)^2 \sum_{y \in U} f(y) \exp(\frac{i\alpha L(c^U - y)}{\sqrt{2}pl}) \tag{5.56}$$

in which, $\alpha = (\alpha_1, ...\alpha_d)$ is the multi-dimensional index which stands for a d-tuple of non-negative integers (in this context, $d = 2$), p is the number of plane-wave coefficient required per dimension

to obtain the desired precision ϵ and L is the truncation error term (the detail configurations of p and L can be found in [76]).

Assume that the domain Ω of interest is a unit square $[0,1]^2$ because of the spatial-temporal dimension of the domain (if a value stays out of the range, shifting and re-scaling have to be performed), by partitioning Ω into uniform squares U of size $\sqrt{2}l$ as the beginning of FGT technique.

5.2.3 Conquer stage

In theory, FGT might compute the desired result in three steps: **S2W**, **W2L** and **L2T**. Before explaining these terms, the definition of 'interaction list' should be firstly addressed. This list is denoted by $\mathcal{I}[U]$ which describes a specific set of neighbor for U. Basically, this set supports the kernel at the center of U. Firstly, FGT starts with the **S2W** step. This step sequentially calculates the equation (5.54) for each square U. Subsequently, the plane-wave expansion, which is created in **S2W**, propagates to all of the elements V of $\mathcal{I}[U]$ as a 'local' expansion. Intuitively, the visualization of propagation steps can be found in Figure 5.4. In this figure, the 'local' step **S2W** is computed straightly in green layer. After that, the interaction list $\mathcal{I}[U]$ is checked to calculate other layers. This calculation is conducted by the propagation following the light magnitude of blue color. The step **W2L** plays its role by modifying the specified expansion as shown below:

$$w_\alpha^* = w_\alpha \exp(\frac{i\alpha L(c^V - c^U)}{\sqrt{2}pl}) \tag{5.57}$$

In the last step **L2T**, the conjugacy $G(x_j)$ is computed at x_j using the 'local' expansion from the box containing it:

$$\mathbf{K}p_j \triangleq G(x_j) = \sum_{|\alpha| \leq p} \mathcal{F}(\alpha) w_\alpha^* \exp(\frac{i\alpha L(x - c^V)}{\sqrt{2}pl}) \tag{5.58}$$

For acceleration purposes, the $sweeping$ algorithm in [77] is implemented with the FGT method. As previously mentioned, FGT helps reduce the computational costs to $O(mn)$ with m is the number of the target point and n is the number of source point. However, this result suffers a decreasing in accuracy due to the ϵ parameter which also critically influences the p and

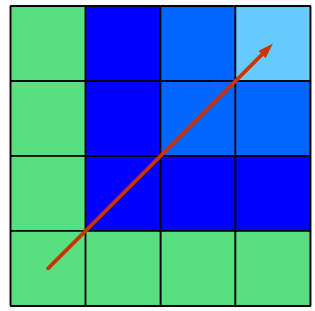


Figure 5.4: Direction of calculating 'local' expansion step **S2W**.

L parameters. To overcome this drawback, the IFGT proposes a strategy to adaptively select the ϵ parameter without the loss of accuracy. This strategy is based on an improvement to the Krylov subspace method [78] for the symmetric positive definite matrix. In this research, ϵ is chosen using the following inequality:

$$\epsilon_i \leq \frac{\delta}{n} \frac{\|y - K\xi_0\|}{\|\check{r}_{i-1}\|} \tag{5.59}$$

in which, δ is the bound determined by the subtraction of the i^{th} iteration's residual to the corresponding residual of the approximate matrix-vector product: $\|\check{r}_i - r_i\| \leq \delta$, and n is the number of training data points. With this enhancement, the complexity in training phase now drops to $O(m+n)$ which is the linear complexity.

There has been an issue related to whether it is possible to apply the IFGT coupling with the conjugate gradient (hereinafter, IFGT-CG) in hyper-parameters learning phase. As shown above, when doing the conjugate gradient iteration to solve the linear system in the equation (5.48), it requires the matrix-vector multiplication (MVM) which costs $O(in^2)$ where i is the number of iterations and n is the number of training points of dataset. Together, the IFGT-CG reduces this complexity to only $O(m+n)$. Due to this reason, it sounds suitable to engage this combination to the hyper-parameters learning phase to achieve better reaction rate. Unfortunately, the answer is 'yes' for the conjugate gradient and 'no' for the IFGT. The reason is that IFGT, which is derived from the FGT technique, works properly only if the objective function can be represented in the potential form [79] (far field or near field [9]). This strict requirement is impossible for rMLL in the equation (5.18) as well as the partial derivative in the equation (5.17). To make this point more

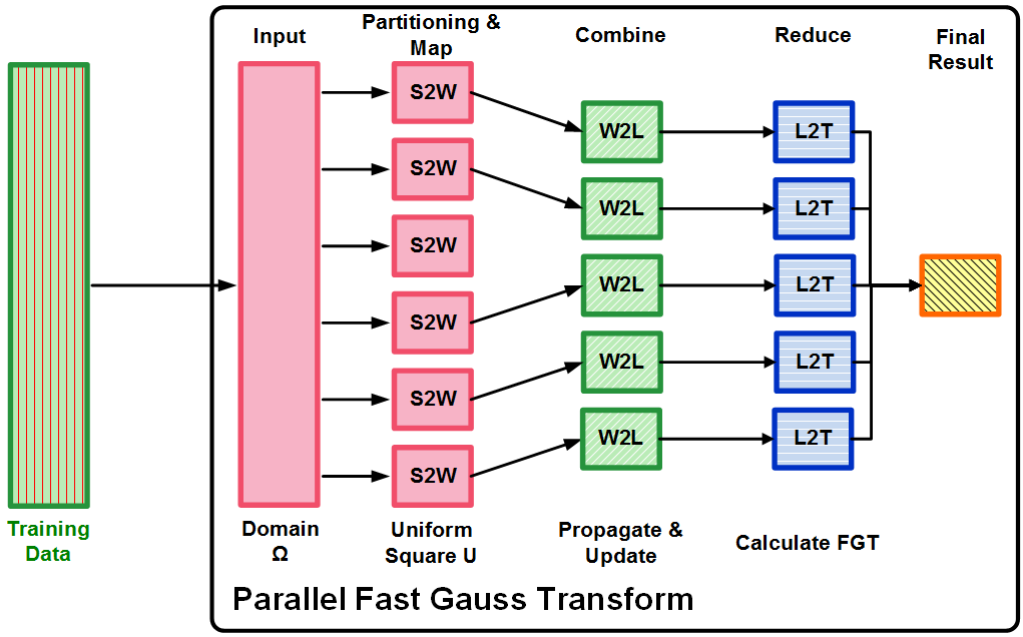


Figure 5.5: MapReduce-oriented implementation of parallel Fast Gauss Transform.

clear, assume that $F(x)$ is a scalar or vector field. For a fixed target point y, depending on the location of source point with a predefined range r, the field $F(x-y)$ for x inside the range r is called *near field*. For x out of the range r, this field is defined as *far field*. The problem is that there is no way to transform the objective functions in equations (5.17) and (5.18) to the Gaussian-type potentials. It means that minimizing rMLL is problematic when using the IFGT. In this case, the standalone conjugate gradient solves the hyper-parameters learning at the computational complexity of $O(n^2)$ with n is the number of training point of the dataset. This complexity is worse than the $O(nlogn)$ of the proposed technique introduced in the previous section.

5.2.4 Parallelism engagement

Although the optimization procedure benefits immensely from IFGT and conjugate gradient, the training phase can be further improved by implementing the parallelism. While maintaining the notion of dynamically choosing the precision parameter ϵ, the FGT operation can be adjusted to enable the parallel computing. As a consequence, the IFGT, which engages the FGT, also shows the improved performance. When examining the structure of the FGT method, it seems natural

to parallelize three steps (**S2W**, **W2L** and **L2T**) of calculating conjugacy to take advantage of the efficiency of concurrent computing. Since the dividing step and **S2W** can be grouped up. It is suitable to assign this combined step to *Map* phase. Similar to **W2L** and **S2L**, which can be assigned to *Combine* phase and *Reduce* phase, respectively. After assignment, it is possible to start the parallelism.

Technically, unlike the parallel method introduced in [77], the mechanism of parallelism here is to invoke the idea of MapReduce method to take advantage of a robust and flexible parallel implementation. This method can be implemented straightforwardly by using any parallel technique resulting in another kind of parallel fast Gauss transform (PFGT) which accelerates the training phase (Figure 5.5). As described above, **S2W** can be considered as the *Map* phase, **W2L** and **L2T** are mapped to *Combine* and *Reduce* phases. Prior to this, in the *Map* phase, a grid of separated U squares is partitioned. Then, these squares are distributed to separate computing node as the regular input. Depending on the data, the tasks for the Fourier transform of each square U are created. These tasks follow the equation (5.54) and can be controlled by the a specific component, which mimics the role of task tracker of the MapReduce framework. After the completion of *Map* phase, the outcome of each task is propagated to all other the members in the interaction list of each square U as a 'local' expansion. This propagated value is modified afterwards in *Combine* phase. Subsequently, *Reduce* phase receives the updated data from the *Combine* phase and creates the task for the **L2T** step. By executing these **L2T** tasks during *Reduce* phase, the results can be achieved at a much faster rate in comparison with the original FGT method.

Although it is hard to analyze the complexity of the MapReduce-oriented operation, the overall complexity of PFGT performing on n source points and m target points can be roughly estimated but not exactly as $O(m + n/n_p)$. The reason for this estimation is that the computing tasks with the same complexity are now divided and simultaneously processed at n_p computing facilities. However, it is worth noting that the main complexity of the training phase does not change mathematically, even the processing speed is improved. The comparison of complexity between the proposed method and the others can be found in Table 5.1. Finally, the implementation of PFGT is described in Algorithm 3.

Algorithm 3: MapReduce Implementation of Fast Gauss Transform

Data: Training data
Result: $G(x_j)$ over the square U
Partition the domain Ω into squares U of size $\sqrt{2}l$;
Calculate $G(x_j)$ on each square U;
`/* Execute the S2W step */`
Distribute squares U to n Map job;
On Map **phase:**
    Execute the 'local' expansion;
    Propagate the contribution to the interaction list;
`/* Execute the W2L step */`
On $Combine$ **phase:**
    Execute the modification;
`/* Execute the L2T step */`
On $Reduce$ **phase:**
    Calculate the conjugacy $G(x_j)$;
return $G(x_j)$;

Chapter 6

Performance evaluation

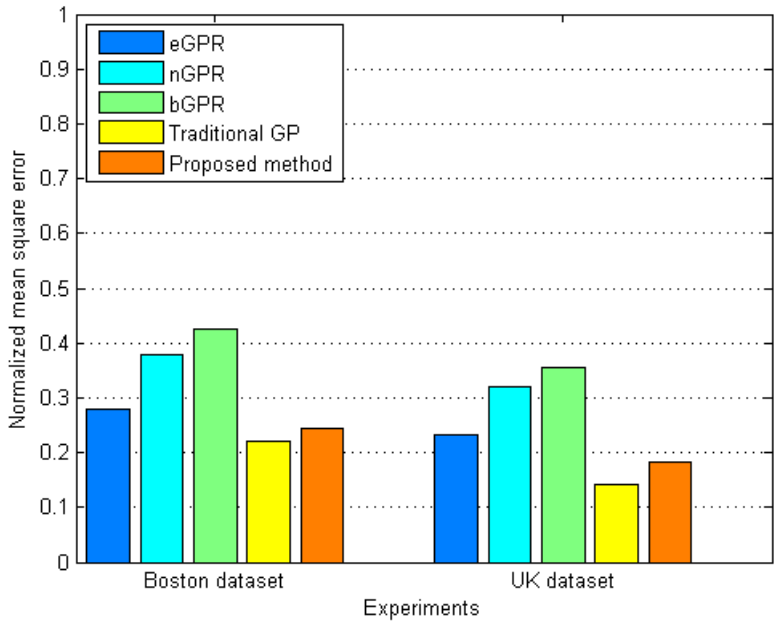
In this chapter, two groups of experiments would be conducted. The first group, namely bench-marking experiments, includes the benchmarks on the empirical dataset and spectral experiment. This group of experiments shows the significant improvement of the proposed method over some other well-known improved Gaussian process regression approaches. Especially in the spectral experiment, the proposed method is extended to solve the issue of the signal processing area. This extension shows the fact that not only the prediction-oriented applications enjoy the benefit, but also other Gaussian process regression problems can be solved by applying the proposed methodology. In the second group of experiments, three applications are developed using the proposed method to support the decision making in solving some real-world problems. Hence, the applications cover two topics: regression in communications area and energy efficiency in computing systems. Once again, we can clearly see the extension of the proposed method to the related areas. For more information, the problem analysis and the implementation logic of these applications can be found in chapter 3 and the corresponding papers.

6.1 Bench-marking experiments

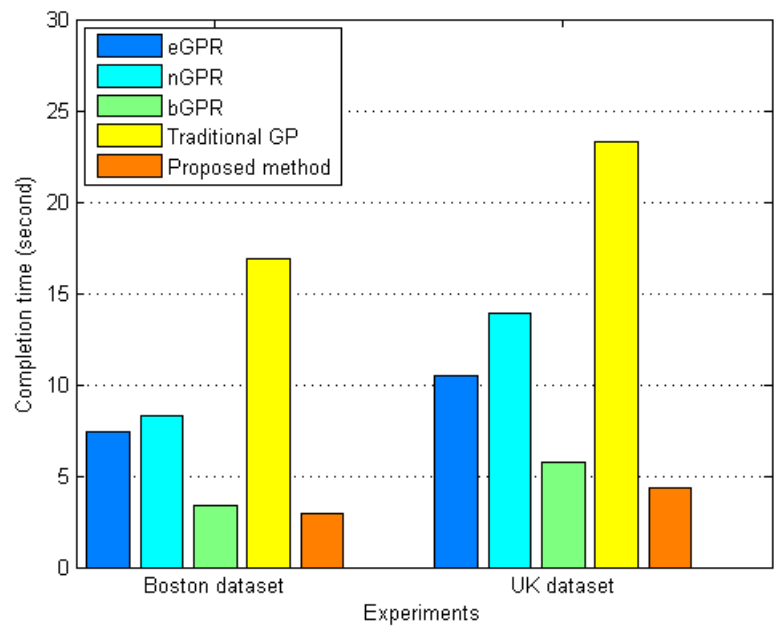
6.1.1 Empirical experiment

Experiment design

Two datasets are used in this experiment: the Boston housing dataset [80] and the UK land registry price paid dataset [81]. The former dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. It was obtained from the StatLib archive and has been used extensively throughout the literature to benchmark algorithms. The name for



(a) Normalized mean square error benchmark (lower is better)



(b) Completion time benchmark (lower is better)

Figure 6.1: Accuracy and processing speed evaluation.

this dataset is simply Boston. It has two prototasks: nox, in which the nitrous oxide level is to be predicted; and price, in which the median value of a home is to be predicted. In the experiment, we choose to predict only the median value of a home. The dataset is small in size with only 506 cases. In contrast, the latter dataset is quite large. We downloaded the monthly price paid data for the period February to October 2012 in the UK, which covers England and Wales and filtered for apartments. This resulted in a data set with 75,000 entries, which we cross-referenced against a postcode database to get latitude and longitude, on which we regressed the normalized logarithm of the apartment prices. Randomly selecting 8,000 data points as a test set. In this subset, we divide the data into tenfold, in which each fold consists of 800 data points. Our target is to model the changing cost of apartments. The results of experiment across tenfold are eventually averaged. The benchmark for both datasets is conducted on a single system equipped with Intel Xeon E7-2870 2.4 GHz, 16 GB of RAM. It is worth noting that we intend to choose different size of the dataset to evaluate whether there would be any divergence in the final result.

Implementation

Beside the implementation of the proposed method, we choose four more algorithms including the classic GP-MAP for comparison purpose. These algorithms are as follows:

- eGPR: the method which is developed from the evolving Gaussian process for predicting chaotic time series [3]. This method use the combination of Cholesky decomposition and conjugate gradient to update and minimize the negative marginal log likelihood.
- nGPR: derived from the research of 'Bayesian nonparametric adaptive control using Gaussian processes.' [15]. The methodology is mainly the entropy optimization combined with Kullback–Leibler divergence maximization
- bGPR: this method is based on stochastic variational inference, which is an input-independent technique. The detail can be found in the original research, namely 'Gaussian processes for big data' [6].
- Traditional GP: the classic and traditional Gaussian process regression (GP-MAP) [1]. In this method, no enhancement is implemented.

Metrics

Five methods including the proposed method are performed on two datasets. The experiment is supposed to measure two metrics: the accuracy of the prediction and the speed of calculating the result. The accuracy is measured by the value of normalized mean square error (NMSE), which is according to below equation:

$$NMSE = 1 - \frac{||y_i - y_i^*||_2}{||y_i - \overline{y}||}, \tag{6.1}$$

in which, y_i^* is the predictive value for the test sample $i = 1, ..., n$ and y_i is the actual test value. The training data mean is denoted by $\overline{y}$. Reasonably, the lower mean square error a method reflects, the more accurate that method achieves. As stated previously, the average result is computed for the second dataset after benchmarking on tenfold of selected data.

Result

In Figure 6.1a, considering the benchmark on the Boston dataset, the traditional GP-MAP can be seen as the most accurate technique among the investigated methods. It is understandable due to the fact that there is no degradation in the precision of constructing the hyper-parameters. The proposed method, which sacrifices a little accuracy to achieve better performance, obtains the second place, but still outperforms other techniques according to the convergence law of log determinant. The same circumstance happens with the UK dataset. In overall, the accuracy of the proposed method is basically degraded from 2% to 7% compared with GP-MAP with regard to each dataset. It is worth noting that the accuracy also increases but not so fast when more data is added to the training.

In Figure 6.1b, although the traditional GP-MAP achieves the highest accuracy in the previous measurement, it suffers an extremely bad computational speed due to matrix inverse and log determinant calculation. Contrary to a slight degradation in the test of accuracy, the proposed method can cut down the processing time more than 82% compared with the traditional GP-MAP. Our method also has better performance in comparison to the remaining methods. Besides, it is noticeable that bGPR can achieve very analogous performance to the proposed method in term

of speed. This can be seen as the result of data-independent technique, which is used in bGPR. However, this technique also drastically degrades the accuracy in comparison to others.

6.1.2 Spectral experiment

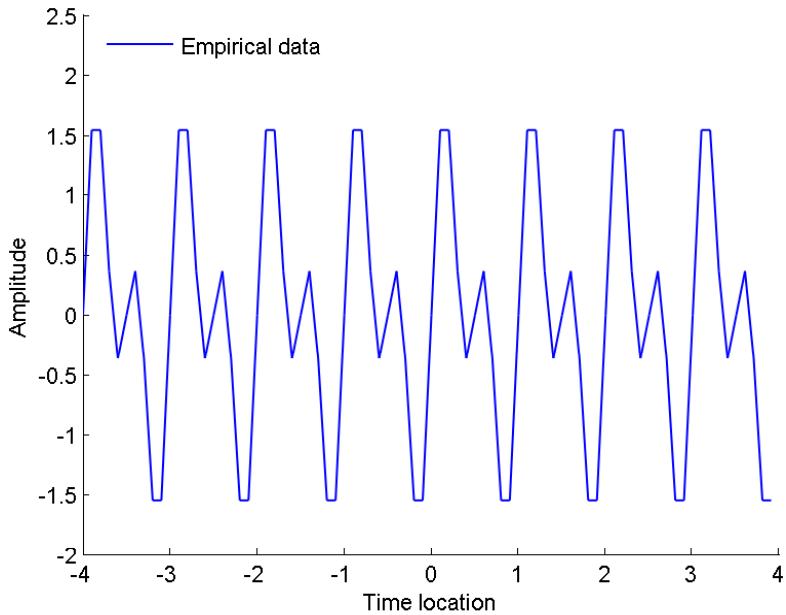
Experimental design

In this experiment, it is necessary to remind that we conduct the regression based on the complex data, which are generated by combined sin functions in (5.21) and (5.22). Intuitively, the empirical or training data of these functions are described in Figure 6.2a and 6.2b, respectively. The goal is to predict the futuristic region where the time location is in the range of $[4, 5]$.

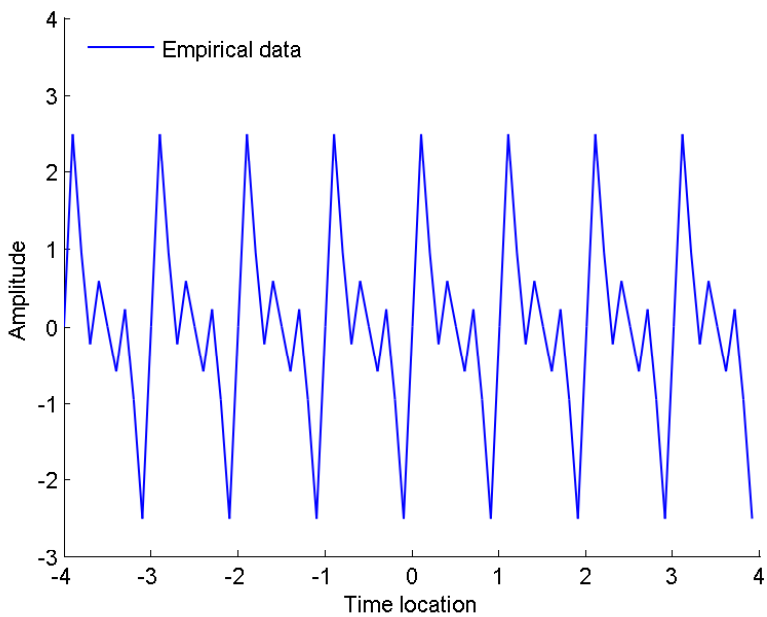
Result

By using our Gaussian mixture kernel, we can do the desired regression as shown in Figure 6.3a and 6.3b for two and three combined sin functions, respectively. In this experiment, it is obvious that the predictive signals (in red dashed line) at the desired locations are not only successfully extrapolated, but also achieved high accuracy compared with the test data (in green line). As discussed previously, the prediction in the time domain is equivalent to the power spectral density estimation in the frequency domain. This idea is clearly proven in Figure 6.4a and 6.4b. Whereby, the estimated power spectral densities, which are produced by the corresponding PSDs (in red line), fit quite well with the empirical PSDs (in blue line). Of course, there would be some discrepancies between two PSDs because of the small noise added to the data and the prediction, but in general, the Gaussian mixture kernel is really effective to deal with the complex underlying kernel functions.

Last but not least, the proposed mixture kernel obviously can be extended easily to deal with any number of combined functions. Contrary to this convenience, the regular Gaussian kernel, unfortunately, cannot express correctly the complex data. Hence, there is needless to compare the proposed method with other techniques (which are based on regular Gaussian kernel) in the spectral experiment.

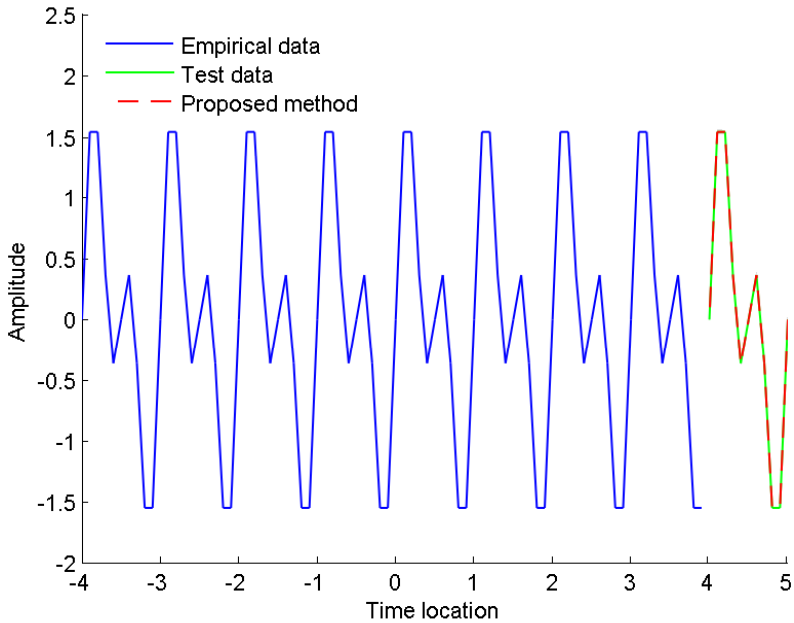


(a) Two sin functions

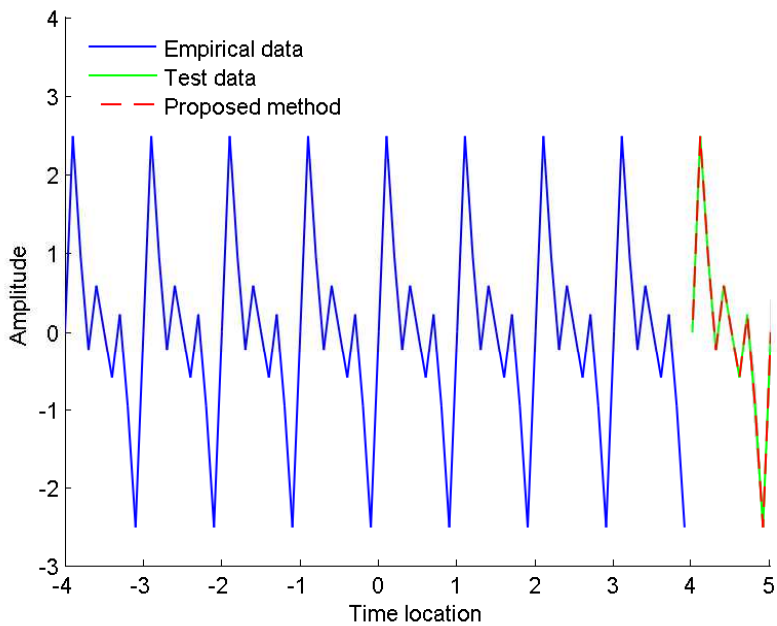


(b) Three sin functions

Figure 6.2: Empirical data produced by the complex underlying combined functions.

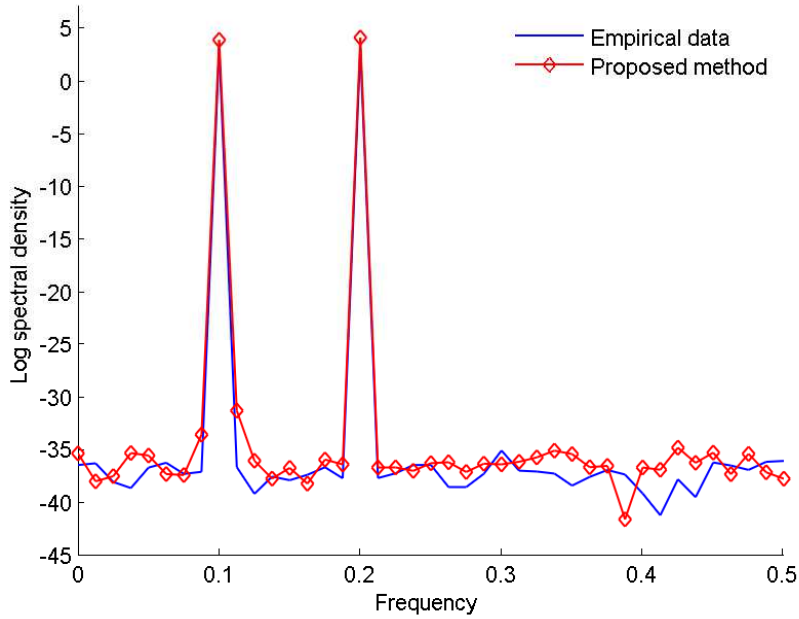


(a) Two sin functions

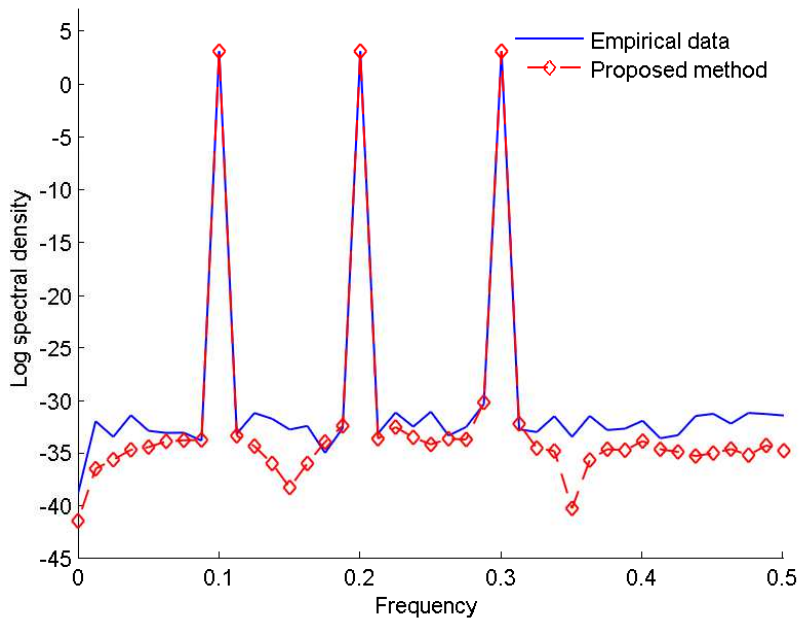


(b) Three sin functions

Figure 6.3: Prediction based on empirical data.



(a) Two sin functions



(b) Three sin functions

Figure 6.4: Estimated power spectral density of complex empirical data.

6.2 Potential applications

6.2.1 Regression in communications

Experiment design

In the experiments, we plan to evaluate some typical metrics in communications, namely the bit error rate (BER) and the signal to noise ratios (SNRs). The target system is of synchronous DS-CDMA type with 8 users spreading by the Gold sequences. Particularly, these binary sequences are generated with the length of 31. The powers of all users are equal with SNR = 4dB. The channel model is as follows:

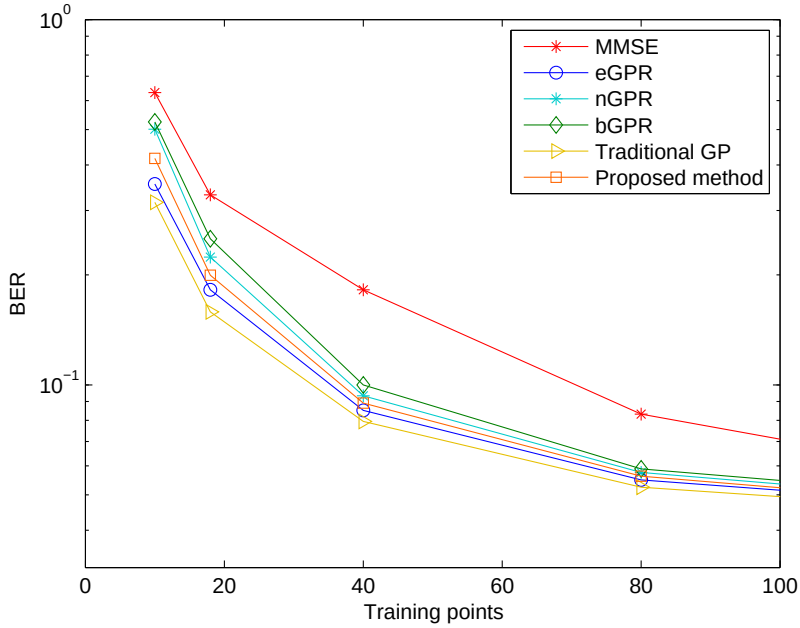
$$H(z) = 0.4 + 0.9z^{-1} + 0.4z^{-2}. \tag{6.2}$$

For evaluation purpose, the MMSE estimation, eGPR, nGPR, bGPR, the traditional Gaussian process regression (traditional GP), and the proposed method are performed and compared together. Initially, the hyper-parameters θ_1 and θ_2 of GPR-family are set to 0. By the end of the learning process, the values of θ_1 and θ_2 are updated to 0.6782329 and 6.782329, respectively. Note that the MSE threshold is limited to $\epsilon = 0.1$. With this threshold, the hyper-parameters need around 4 to 5 iterations to reach the above values.

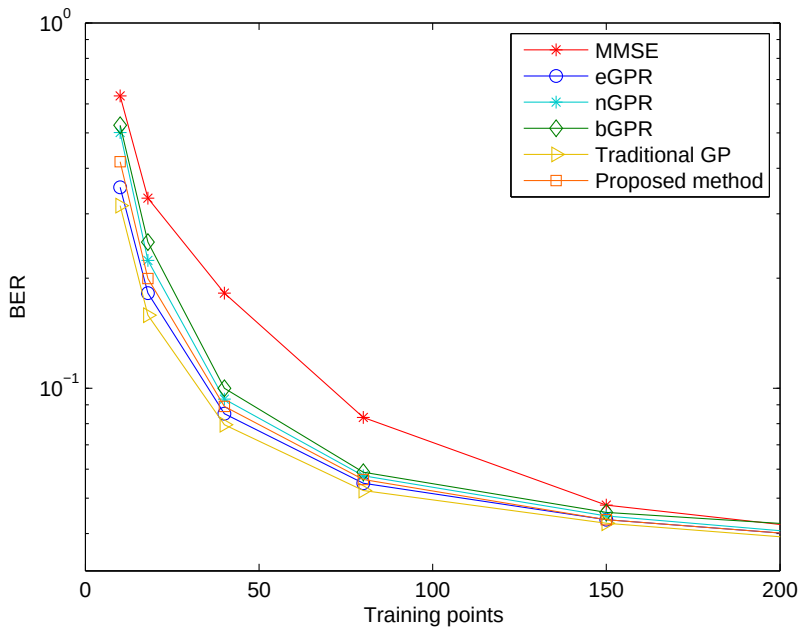
Result

In Figure 6.5a and 6.5b, a series of experiments are conducted in ascending order of the size of the training dataset. For each experiment, the BER is computed for 10^6 bits. In these experiments, the result of GP-family is very close together. It is worth mentioning that the GPR-family outperforms the MMSE estimation in terms of BER performance, especially when the size of training dataset is small. When the number of training points increases gradually, the results of all approaches come close together. Make a comparison between approaches in GP-family, it seems the eGPR scores better accuracy when the number of training point is small. When the size of dataset increases more than 100, the proposed method tends to converge and overcome the eGPR in term of accuracy.

In Figure 6.7a and 6.7b, the relationship between the BER and the SNRs is depicted *via* the



(a) On 100 data points.



(b) On 200 data points.

Figure 6.5: Accuracy in the relationship with the size of dataset.

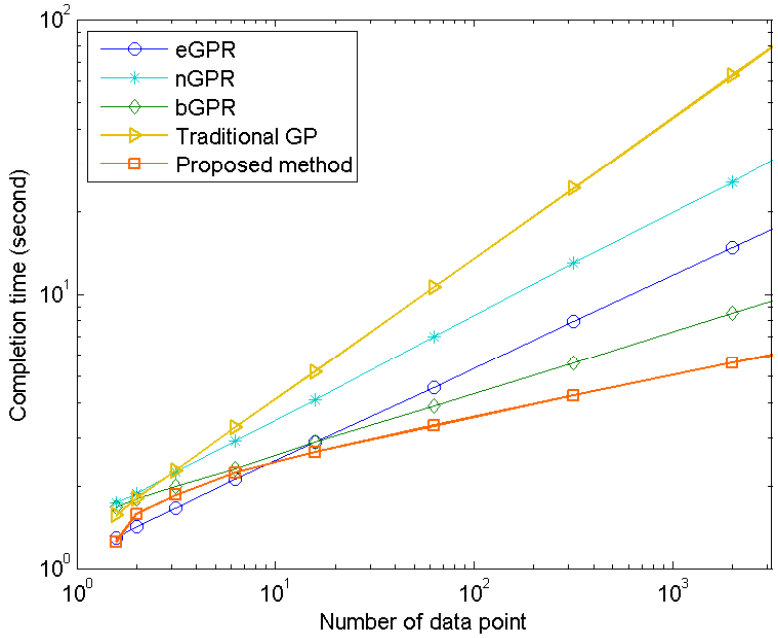
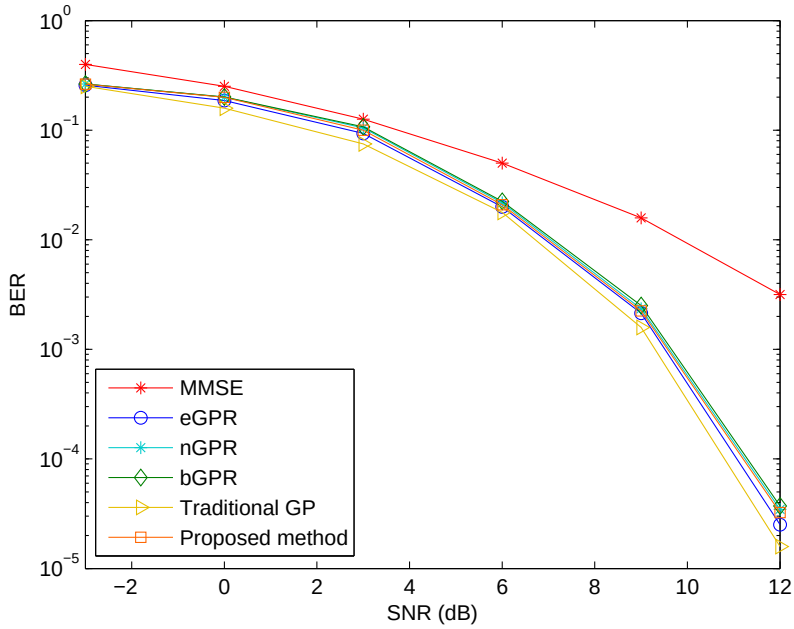


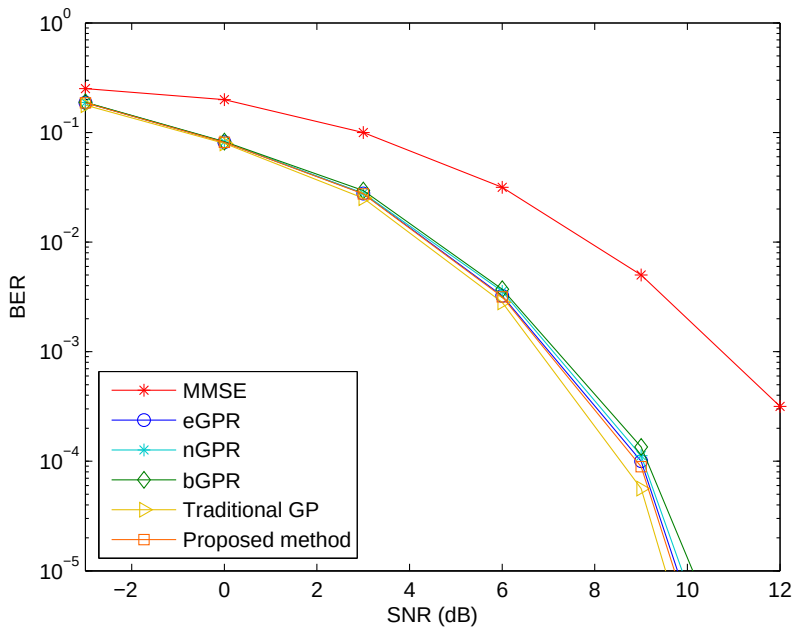
Figure 6.6: Time complexity enhancement of the proposed method.

tests on 100 and 200 training points, respectively. In the test on 100 points, the eGPR is slightly more accurate than the proposed method, just like the previous experiment. The reason is also analogous: the proposed method needs some time to converge before achieving the better result. In the test on 200 points, the proposed method clearly converges and outperform other techniques excluding the traditional GP, which is the most accurate in all of the experiments. Based on these tests, a conclusion can be made that the proposed method provides a very close result to the traditional GP when increasing the size of the dataset. Only a small gap exists between the GPR approaches due to the error in the hyper-parameters approximation. Even in that case, the result of GP-family is still much better than the MMSE estimation.

Connecting with the runtime, when training size skyrockets, the processing rate of the proposed method outperforms the remaining approaches as a consequence of lower complexity level. In order to make the conclusion more reasonable, the benchmark of completion time is conducted on a larger dataset with more than 3000 training points. This dataset is a subset of Google traces dataset [82]. The simulation is implemented in Python on a minimal CentOS system with no algorithmic change. Critically, the significant improvement of the proposed method can be seen in



(a) On 100 data points.



(b) On 200 data points.

Figure 6.7: Bit-error-rate at each signal-to-noise ratio.

Figure 6.6.

6.2.2 Small-scale prediction

Experimental design

In the first evaluation of potential applications, our experiment is aimed to investigate the performance of the proposed application in terms of energy efficiency and execution time when orchestrating the processes inside a regular CPU (Intel Core i7-3770, 3.40GHz). In the initial experiments, the workload was generated *via* the CPU intensive benchmark for one hour to determine the energy savings. In this test, in order to more easily control the number and the intensiveness of the workload, a benchmark software, namely *stress*-1.0.4, was used to simulate the incoming processes. Otherwise, in the second experiment, ten bunches of ten concurrent jobs (totally one hundred instances of *gzip* command on 256KB of test data) were pushed into the system to test the execution time. To aggregate the results, the *powerstat*-0.01 and the *sysstat*-9.0.4 were used to log the power consumption and workload statistics, respectively. Note that the underlying OS is the popular CentOS 6.5.

Implementation

The predictive utilization of each core was anticipated using a combination of *Python* script and *C++* library. *Python* was chosen because of the light-weight feature in comparison with *Matlab* [83]. In fact, *Python* possesses a small core of commands equipped with all the functionality that researcher would require. Besides, *Python* interpreter is free and available for all operating systems. In this combination, in addition to the hyper-parameter estimation source code implemented directly in *Python* for the ease of environmental parameters tuning, the core of Parallel IFGT was implemented in *C++* and wrapped by $ctypes$ as a library for compatibly running with the *Python*. The main reason for this particular implementation is related to the performance. In addition to the aforementioned implementation of the proposed method, three other algorithms, namely the original IFGT coupling with Conjugate Gradient (in short, IFGT), the pure Conjugate Gradient (in short, CG) and the Direct method (the Gauss-Jordan elimination) were also applied for matrix inversion in the hyper-parameters learning phase and training phase, mainly for pur-

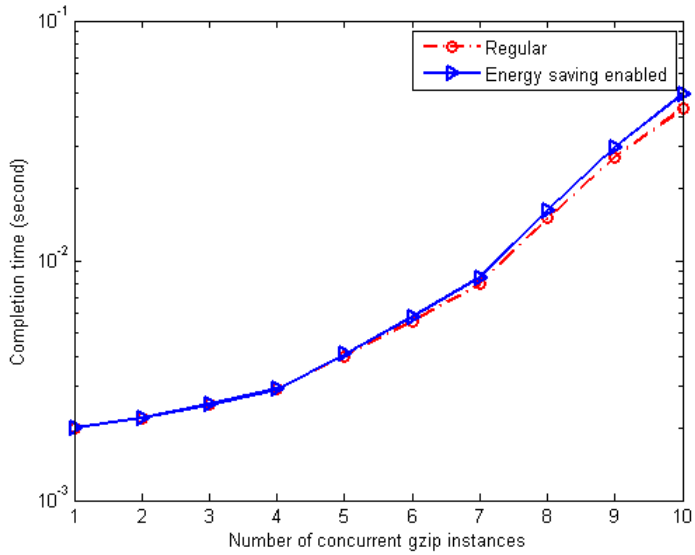


Figure 6.8: Processing rate between two systems (lower is better).

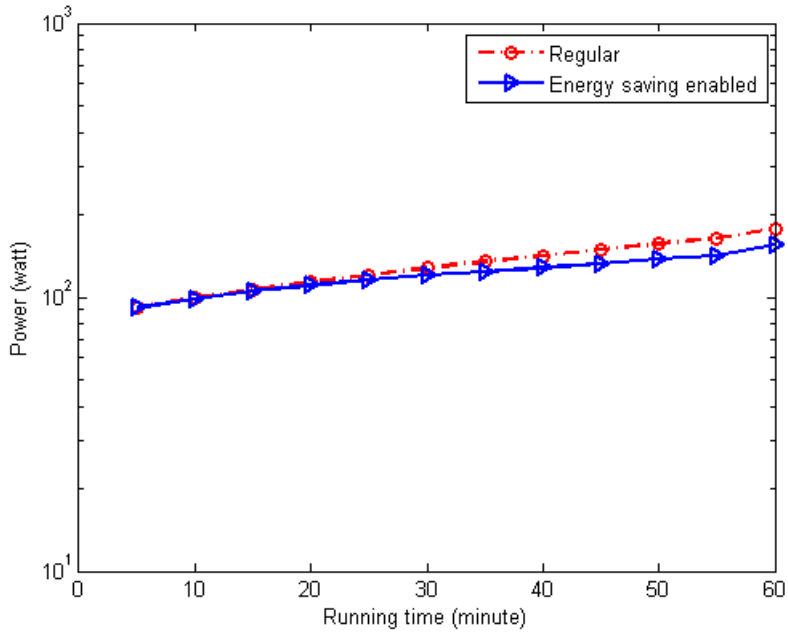
poses of comparison.

Metrics

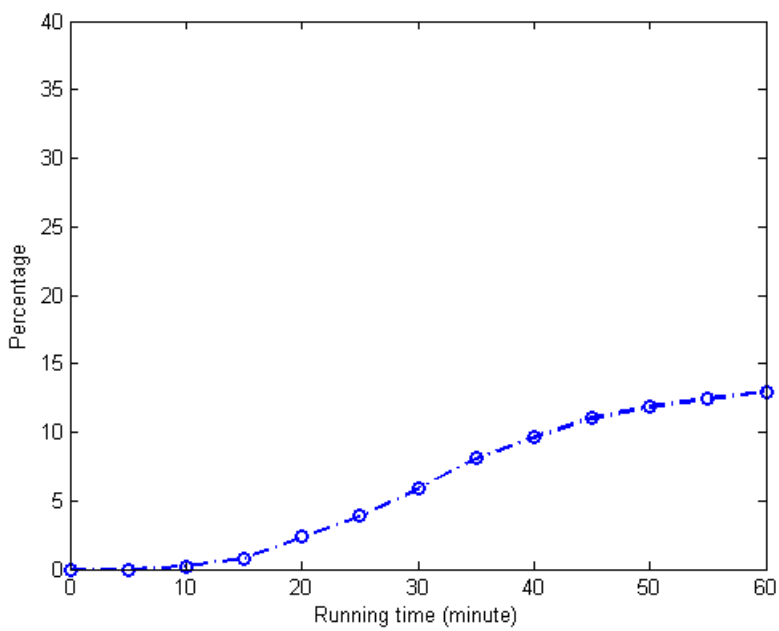
The proposed architecture was measured on two levels: the algorithm level and the application level. In the algorithm level, the metric of interest was the completion time of prediction. In the application level, as previously mentioned, the energy efficiency and execution time were the metrics of interest. If the application were able to save the energy consumption, as well as maintain an acceptable execution time, the energy efficiency of the CPU would be significantly improved.

Result

Application level - Execution time evaluation: in the *gzip* experiment, the system engaging the energy saving application was slightly slower. In comparison with the regular system, the energy saving enabled system takes longer running time to finish the equivalent number of task. This extra amount of completion time is measured so as from 2% to 14%. In essence, this delay time comes from both predicting the utilization as well as migrating the processes and be considered as *context switching* cost [84]. In the worst case, despite increasing by 14%, the time gap between

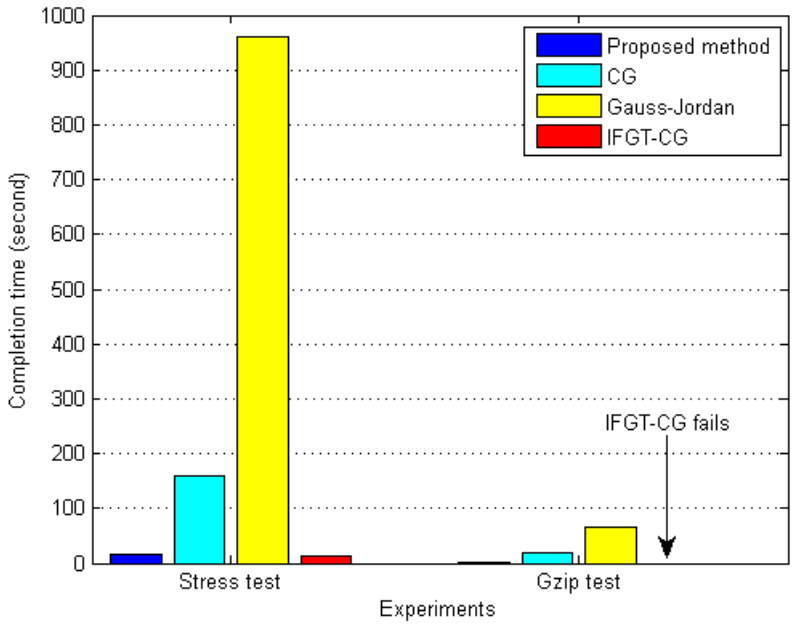


(a) Power consumption (lower is better)

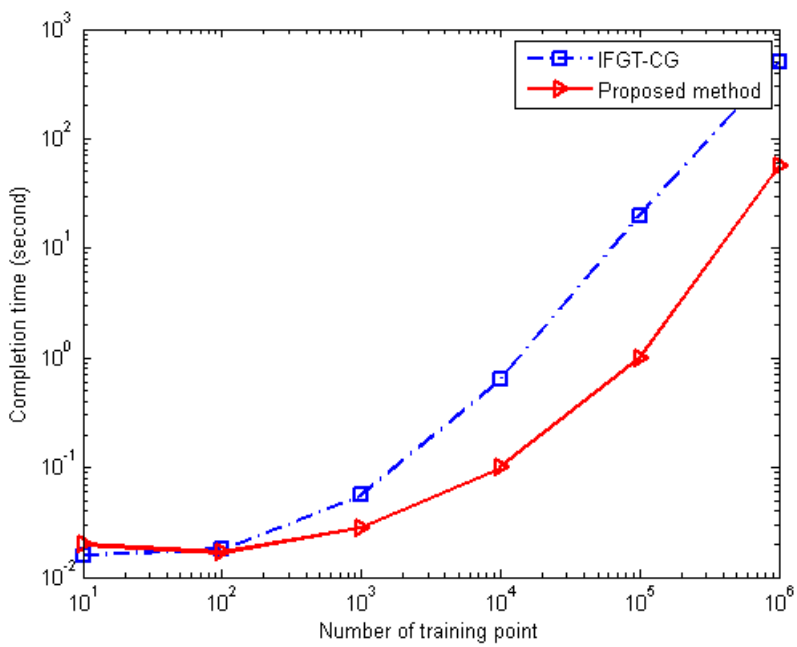


(b) Power saving

Figure 6.9: Power evaluation on proposed method over one hour running time.



(a) Hyper-parameters learning phase



(b) Training phase

Figure 6.10: Computational speed evaluation on each phase of prediction (lower is better).

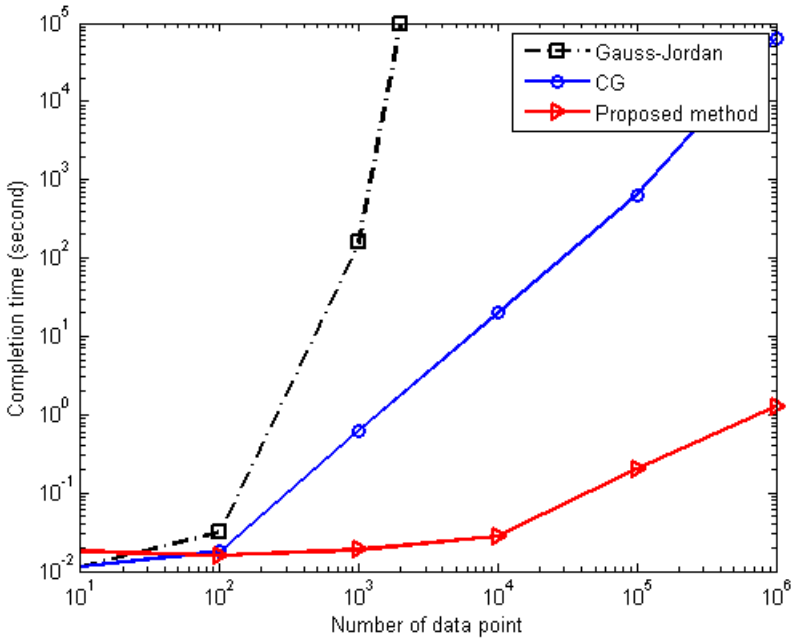


Figure 6.11: Overall processing rate of prediction techniques (lower is better).

two systems was just $6.43 * 10^{-3}$ seconds which is infinitesimal and acceptable (Figure 6.8).

Application level - Energy efficiency evaluation: take a look at the Figure 6.9a, the target systems began with idle status. This status expenses 91.49 watts for keeping alive. The stress test was performed for 60 minutes on each of the target systems. By the end of the experiment, the energy saving enabled system consumed 154.93 watts, while the regular system paid around 177.96 watts. As a consequence, the energy saving application could reduce an energy amount of 23.03 watts (12.94%) (Figure 6.9b). In the hardware perspective, 12,94% energy saving is a significant improvement.

Algorithm level - Prediction performance: as seen in Figure 6.10a, within the same error bound ($\epsilon = 10^{-11}$) and the same training dataset (around 10^3 points), the proposed application took 17 seconds to finish estimating the hyper-parameters on the stress test, whereas the CG and Gauss-Jordan elimination cost 160 seconds and 960 seconds, respectively. For a different training dataset (100 target points in *gzip* test) was used, the proposed application needed approximately 1.7 seconds to finish estimating the hyper-parameters, while the CG and Gauss-Jordan elimination cost 20 seconds and 66 seconds, respectively. In particular, for this small test, the original IFGT

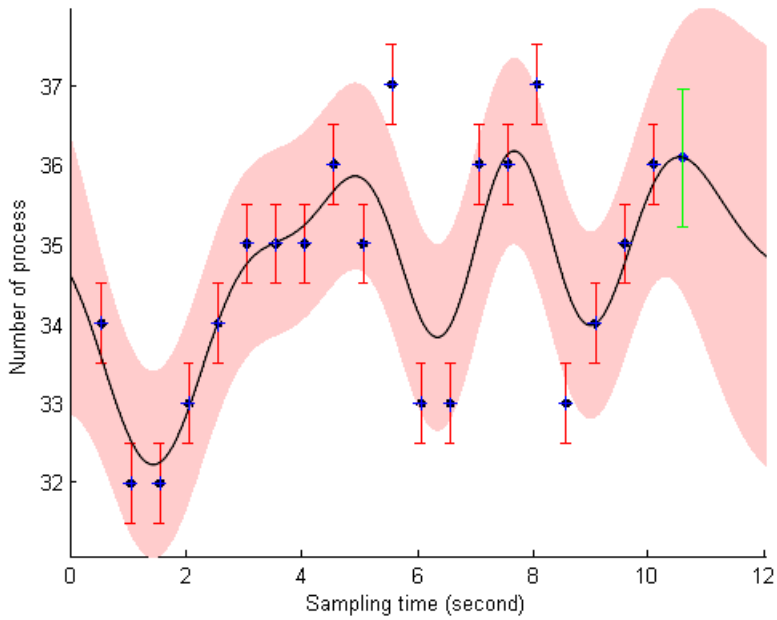


Figure 6.12: Prediction result including mean and variance of proposed method.

algorithm tolerated more failure in the computation. This is predominant due to the difficulties inherent in applying the Gaussian-type potential for maximum likelihood estimation, which has been discussed in detail in [85]. For this reason, this algorithm was excluded from the hyper-parameters learning estimation procedure. The proposed application still coped well with the same learning data. Thus, the hyper-parameters estimated by the proposed application were shared for the IFGT algorithm so as to continue conducting a performance evaluation in the training phase test. Consequently, in the training phase, as well as the overall prediction evaluation, which is described in Figures 6.10b and 6.11, when the number of training point increases, the proposed application continues to significantly outperform the other methods in terms of the reaction rate. Finally, for the reliability measurement, because the proposed application also partially relies on the IFGT, which defines the precision of $\epsilon = 10^{-11}$ in advance, the accuracy requirement is always satisfied. For an accuracy benchmark of 20 consecutive testing points in the stress experiment, the mean prediction was able to adapt well to the testing data, with a 95% confidence maintained by the variance (Figure 6.12).

Table 6.1: Google cluster's organization and configuration

Quantity of nodes	Category	CPU	RAM
1	1	0.50	0.06
3	3	1.00	0.50
5	1	0.50	0.97
5	1	0.50	0.03
52	1	0.50	0.12
126	2	0.25	0.25
795	3	1.00	1.00
1001	1	0.50	0.75
3863	1	0.50	0.25
6732	1	0.50	0.50

Table 6.2: Summary of Google traces' characteristics

Time span	# of PMs	#VM requests	Trace size	# of users
29 days	12583	>25M	>39GB	925

6.2.3 Large-scale prediction

Experiment design

In this experiment, the proposed prediction method is engaged in the predictor to provide the predictive analysis for the energy optimizer. It is worth to remind the purpose of energy optimizer is to produce the near-optimal number of PMs. Based on this number, the cloud orchestrator would execute the migration strategy to condense the VMs, then turn off the idle PMs to save the energy. More information of the internal energy saving mechanism can be found on the potential problems of chapter 3, or from our original publication [25]. The test-bed for performance evaluation is a cluster of 16 homogeneous servers. For the detail configuration, an Intel Xeon E7-2870 2.4Ghz and 12GB of RAM are geared towards the purposed of hosting up to 8 VMs in each serves. With this equipment, the infrastructure can host up to 128 VMs at maximum to conduct the experiment.

There are two kinds of experiments in the evaluation. In the first experiment, Google traces [82] are used to simulate the workload for training the energy efficiency management system. Additionally, the datasets comprise the system statistics from more than 12,000 servers over a duration of 29 days. However, only a set of 6732 machines is chosen to satisfy the assumption of the homogeneous system. In this set, we also extract randomly 2.26 GB from 39 GB of compressed data for the experiment. The chosen dataset consists of many parts. Each part represents one day of monitoring statistics. For the ease of calculation, the length of measurement is scaled to 60 seconds. This length is configured as the sampling window. Moreover, some important properties of Google traces are described in Table 6.2.

In addition to the above experiment, *Montage* workflow [86] [87] is utilized to evaluate proposed energy efficiency management (E2M) system in realistic case. Primarily, *Montage* workflow can be seen as a workload creator to make the stress-test more realistic. This software is developed by NASA to assemble the flexible image transport system (FITS) into custom mosaics [88]. To do the experiment, we choose only one set of *Montage* work-flow, which consists of 10429 tasks.

Implementation

Firstly, the default first-fit algorithm is chosen for cloud's scheduling task. This setup is regularly configured in many popular cloud orchestrators such as Nimbus, OpenNebula, and OpenStack. For the evaluation, we implement four energy saving schemes as shown below:

- The default schemes: all of the PMs are activated all the time. No power savings is acquired at all.
- The greedy first fit decreasing (FFD) scheme [89]: the VMs are sorted into the queue by descending order in term of internal CPU utilization. This queue is subsequently submitted to the first host that matches the resource requirement. Basically, the bin-packing approach is used to relocate VMs.
- The proposed approach (E2M) scheme: the proposed method is implemented to create near-optimal energy consumption and preserve the quality of services.
- The optimal energy-aware scheme: an optimal solution is pre-calculated to achieve mini-

Table 6.3: Equivalent energy consumption exchanging scheme

Parameter	Value	Unit
E_{sleep}	107	Watt
E_{idle}	300.81	Watt
E_{peak}	600	Watt
$E_{\text{active}\rightarrow\text{sleep}}$	1.530556	Watt-hour
$E_{\text{sleep}\rightarrow\text{active}}$	1.183333	Watt-hour
$E_{\text{active}\rightarrow\text{off}}$	1.544444	Watt-hour
$E_{\text{off}\rightarrow\text{active}}$	11.95	Watt-hour

mum energy consumption. In this scheme, the quality of services is not taken into account. In order words, the quality of services is sacrificed to significantly save the energy.

Energy vs performance

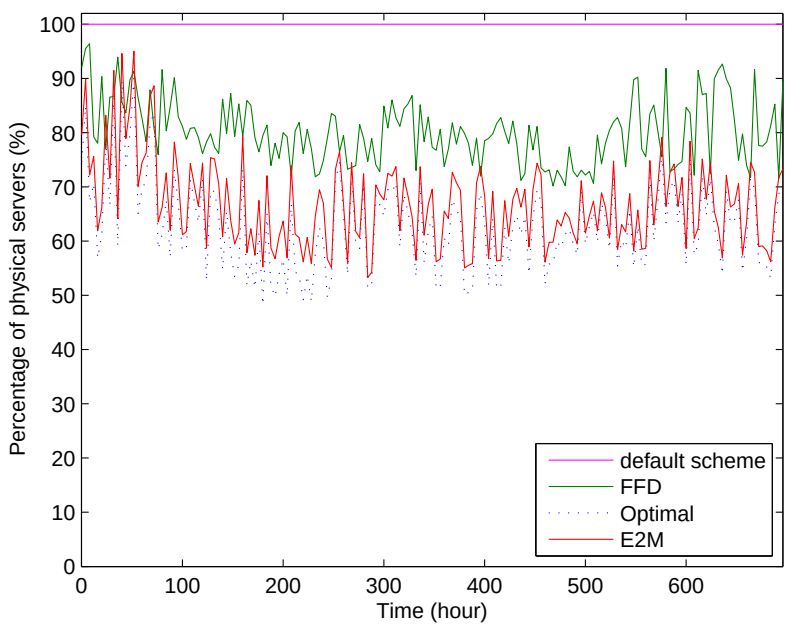


Figure 6.13: Evaluating active physical servers on Google traces.

The Google traces is actually a set of synthesized data. Therefore, in order to measure the

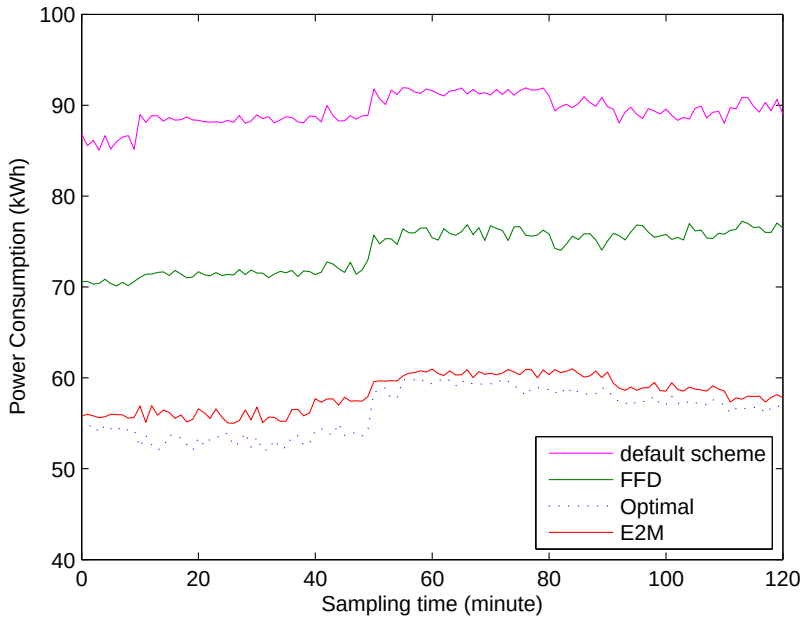
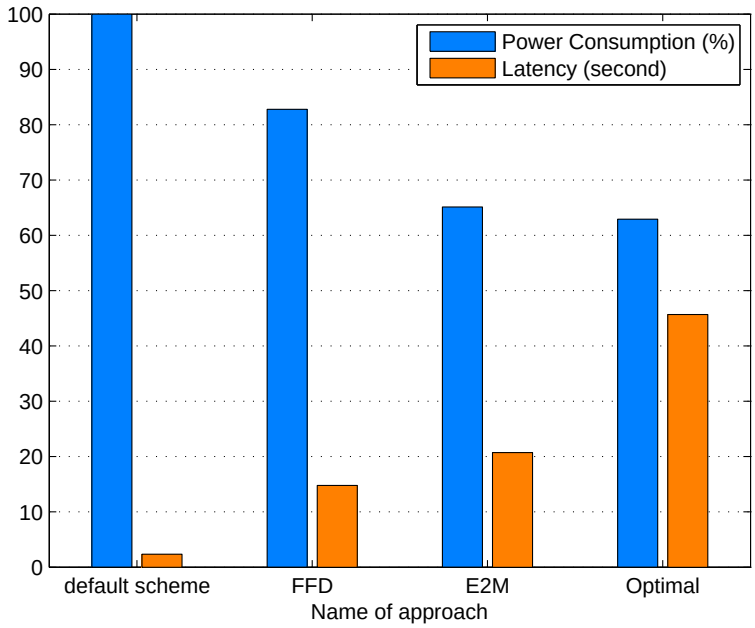
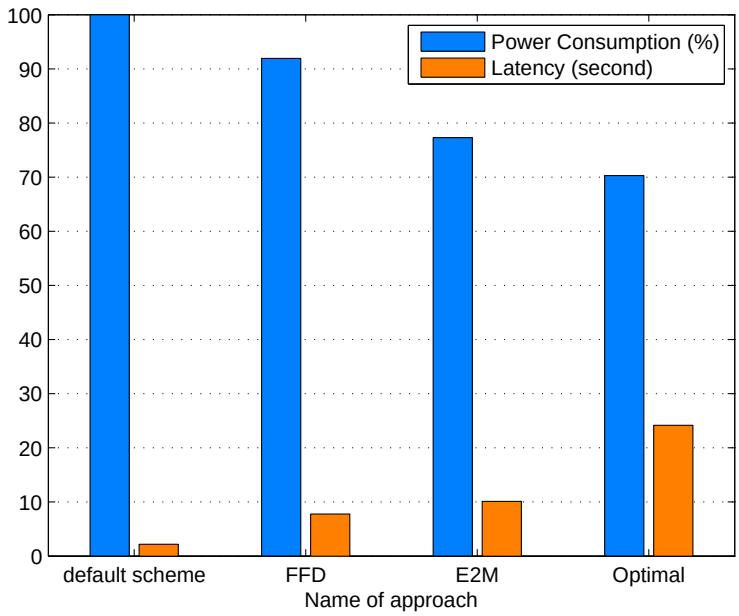


Figure 6.14: Evaluating power consumption on Google traces (lower is better).

energy consumption, an external equivalent energy calculation [29] is applied to compute the result. The description of calculation and related parameters are depicted in Table 6.3. For the result, let's take a look at Figure 6.13 and 6.14. Since the PMs are activated all the time, the default scheme consumes an egregious amount of power. Meanwhile, in the FFD scheme, even the power utilization is less than the default scheme, a remarkable amount of power is wasted since many idle PMs are kept alive when the workload fluctuates. The reason for this issue is that, without the capability of prediction, the FFD is unable to appropriately perform the bin-packing algorithm in a majority of times. Another reason is the obsolete status information of underlying computing facilities. Oppositely, the proposed approach, namely E2M, can save much better energy by equipping with the prediction on resource utilization and the optimization on the pool of active PMs. There is also another additional aspect of this achievement, which is the gap between E2M and the optimal scheme. Apparently, the optimal scheme has better energy savings regardless the system performance. Because the quality of service is totally not considered in this scheme, this optimal solution brings to the infrastructure too much overhead and tends to frequently violate the service-level agreement (SLA).



(a) Evaluation on Google traces experiment.



(b) Evaluation on *Montage* experiment.

Figure 6.15: Power consumption vs average latency.

For more detail on quantitatively measuring the energy savings, our proposal can obtain the reduction of power consumption up to 34.89% compared with the default scheme. The detail evaluation can be found in Figure 6.15a. This achievement can be taken into account as a significant improvement. As a side note, the optimal scheme can only achieve up to 37.08%. In the realistic case, the experiment on the *Montage* work-flow clearly shows that the E2M solution can mitigate 22.71% of the power expenditure in comparison to the default scheme, as reflected in Figure 6.15b. By observing the result, the proposed method can be considered as a near-optimal solution. Also in Figure 6.15a and 6.15b, our method suffers around 54.72% and 58.14% less than the optimal solution in term of average latency, when performing on Google traces and *Montage* work-flow, respectively. Therefore, the quality of services can be preserved at an acceptable level.

Prediction evaluation

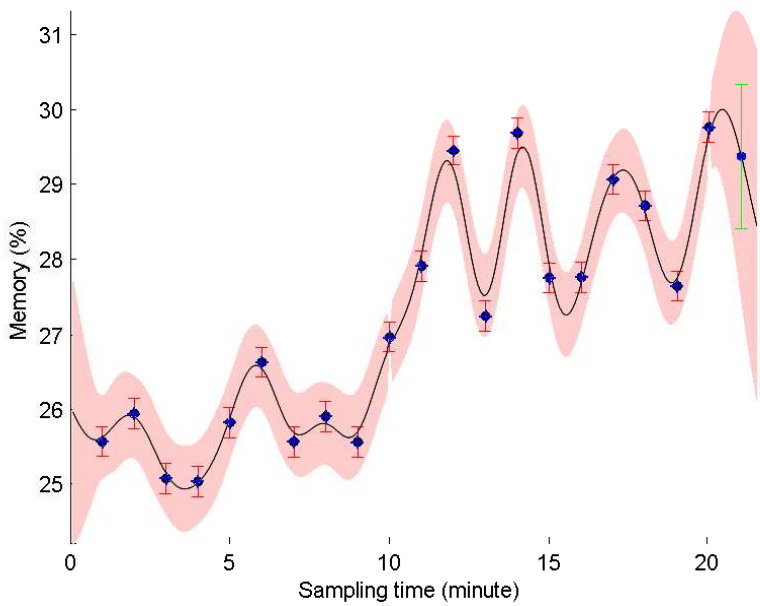
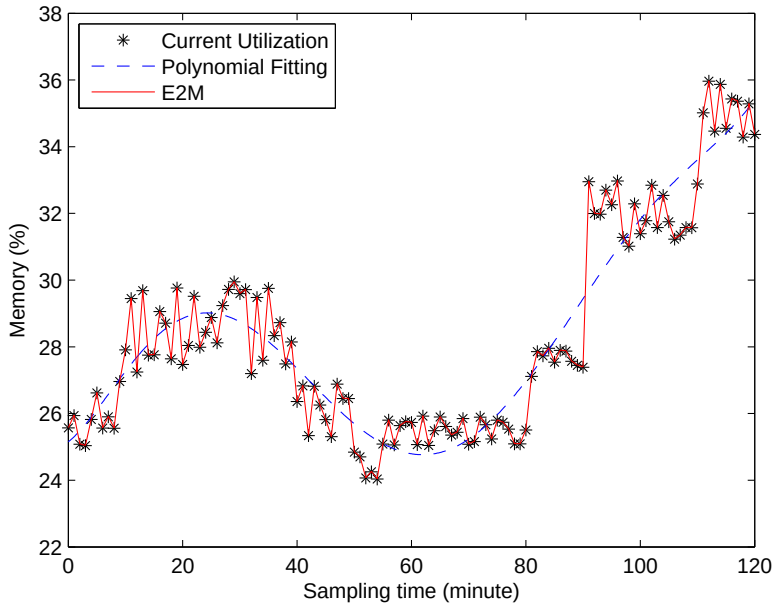
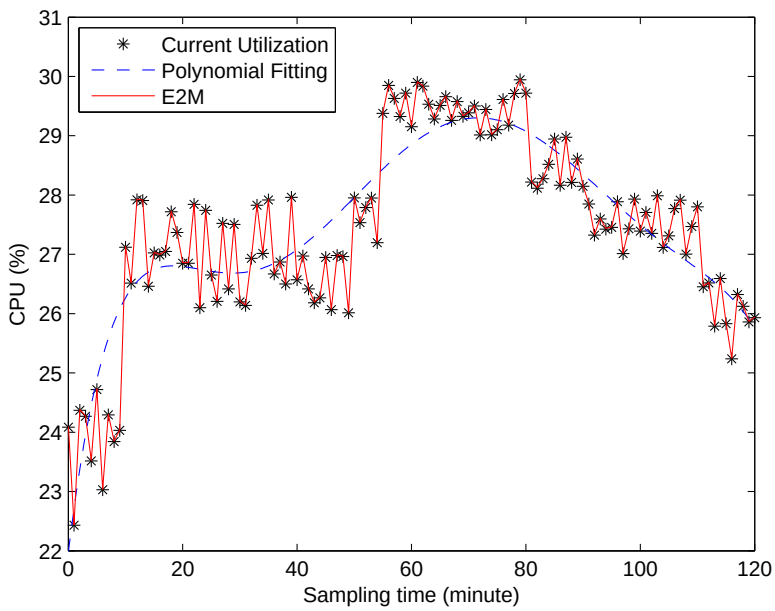


Figure 6.16: Prediction in memory utilization benchmark.

In this part, we would like to evaluate the prediction in terms of accuracy. The Google traces are again used as the dataset for training. In this evaluation, in order to be more realistic, we compare the proposed GPR method with the popular prediction algorithm in cloud computing area, namely the polynomial fitting regression. Figure 6.16 describes how the proposed method works



(a) Memory



(b) CPU

Figure 6.17: Prediction evaluation of the proposed method in Google traces experiment.

with 20 consecutive testing points in the memory utilization. In this prediction, the predictive value adapts quite well to the testing data with 95% of the confident region. Since the training phase of our prediction technique is constructed based on improved fast Gauss transform (IFGT) technique [72], which strictly controls the error bound limited to 10^{-11}, the output of the prediction would be very analogous to the empirical utilization of the target system. Indeed, the enhanced GPR obviously overcomes the PFR in the experiment on the memory and CPU utilization, respectively. This fact can be found in Figure 6.17a and 6.17b

Chapter 7

Conclusions and future work

7.1 Conclusions

The Gaussian process has become increasingly popular for modeling numerous inferences and reasoning solutions, due to the robustness and dynamic features. Particularly concerning regression and classification, the combination of Gaussian process and Bayesian learning is considered to be one of the most appropriate supervised learning approaches in terms of accuracy and tractability. However, due to the high complexity with regard to the time and space, the Gaussian process performs poorly when processing large datasets. Because of this limitation, the Gaussian process is ill-equipped to deal with large systems that require reasonable precision and fast reaction rate. In this dissertation, we would like to propose a thorough complexity reduction method to improve the Gaussian process regression. In order to do that, we thoroughly analyze the nature of Bayesian learning and Gaussian process regression (GPR). The analysis is not only in theory but also broadened to some potential applications in real world. This step is necessary to understand how to apply the aforementioned techniques to build the solution for realistic problems. Subsequently, the proposal of enhancement is introduced to each phase of GPR, which are hyper-parameters learning phase and training phase. In detail, our contribution and uniqueness focus on these below points:

- We propose a complexity reduction to hyper-parameters learning phase of GPR. This method is a cooperation of fast Fourier transform, convergence law of log determinant and stochastic gradient descent. The target of this cooperation is the possibility of indirectly optimizing and approximating the hyper-parameters. By applying this method, we can significantly improve the speed of finding the hyper-parameters with a slight degradation in

accuracy.

- We introduce the 'divide and conquer' coupled with parallel processing to the training phase to improve the performance of this phase, rather than relying only on iterative gradient methods like other related research.

These improvements increase the reaction rate of the prediction method and improve the output data, which subsequently enhances the rationality and timeliness of the decision making process. In addition, the low complexity feature of the proposal makes it feasible to integrate to any solution to deal with the large-scale systems. Moreover, as stated above, the proposed method also proves the capabilities *via* some potential applications such as enhancing energy efficiency in CPU multicore as well as in cloud computing. Especially, the proposed method is not only limited in distributed systems area but also possible to solve the GPR problems in other research fields such as communications and signal processing. Due to this reason, we believe that the enhancement would innovate the development of Gaussian process-based applications to deal with the challenges in many potential issues.

7.2 Future work

As described previously, our assumption in this research focuses more on the homogeneous system for the convenience of constructing and deriving the equations. Although this initialization does not hurt the generality, we also have the plan to extend our work to the heterogeneous system to extensively expand the capability of GPR to more realistic problems. Besides, we are going to apply the parallelism to the hyper-parameters learning phase to achieve an even faster speed of prediction. For the accuracy, we have future plan to improve the precision when dealing with the small dataset. This improvement might be achieved by adaptively tuning the hyper-parameters according to the size of data.

Bibliography

[1] C. E. Rasmussen, "Evaluation of gaussian processes and other methods for non-linear regression." Ph.D. dissertation, Toronto, Ont., Canada, Canada, 1997, aAINQ28300.

[2] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptive Computation And Machine Learning. MIT Press, 2005. [Online]. Available: http://www.gaussianprocess.org/gpml/chapters/

[3] D. Petelin and J. Kocijan, "Evolving gaussian process models for predicting chaotic time-series," in *Evolving and Adaptive Intelligent Systems (EAIS), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.

[4] R. Grande, G. Chowdhary, and J. How, "Nonparametric adaptive control using gaussian processes with online hyperparameter estimation." in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, Dec 2013, pp. 861–867.

[5] A. Banerjee, D. Dunson, and S. Tokdar, "Efficient Gaussian Process Regression for Large Data Sets." *ArXiv e-prints*, Jun. 2011.

[6] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data." *CoRR*, vol. abs/1309.6835, 2013.

[7] Y. Shen, A. Y. Ng, and M. Seeger, "Fast gaussian process regression using kd-trees." in *NIPS*, 2005.

[8] C. Yang, R. Duraiswami, and L. S. Davis, "Efficient kernel machines using the improved fast gauss transform." in *NIPS*, 2004.

[9] R. Beatson and L. Greengard, "A short course on fast multipole methods," *Wavelets, multilevel methods and elliptic PDEs*, vol. 1, pp. 1–37, 1997.

[10] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Aigrain, "Gaussian processes for time-series modeling." *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1984, 2012.

[11] J. Cunningham, Z. Ghahramani, and C. E. Rasmussen, "Gaussian processes for time-marked time-series data." in *AISTATS*, ser. JMLR Proceedings, N. D. Lawrence and M. Girolami, Eds., vol. 22. JMLR.org, 2012, pp. 255–263.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[13] M. E. Tipping, "Bayesian inference: An introduction to principles and practice in machine learning." in *Advanced Lectures on Machine Learning*, ser. Lecture Notes in Computer Science, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds., vol. 3176. Springer, 2003, pp. 41–62. [Online]. Available: http://dblp.uni-trier.de/db/conf/ac/ml2003.html

[14] S. Brahim-Belhouari and J. Vesin, "Bayesian learning using gaussian process for time series prediction." in *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, 2001, pp. 433–436.

[15] G. Chowdhary, H. Kingravi, J. How, and P. Vela, "Bayesian nonparametric adaptive control using gaussian processes." *Neural Networks and Learning Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[16] M. A. Álvarez and N. D. Lawrence, "Computationally efficient convolved multiple output gaussian processes." *J. Mach. Learn. Res.*, vol. 12, pp. 1459–1500, Jul. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021048

[17] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[18] I. Murray and Z. Ghahramani, "A note on the evidence and bayesian occam's razor," 2005.

[19] A. Duel-Hallen, J. Holtzman, and Z. Zvonar, "Multiuser detection for cdma systems," *Personal Communications, IEEE*, vol. 2, no. 2, pp. 46–58, 1995.

[20] S. V. Halunga and N. Vizireanu, "Performance evaluation for conventional and mmse multiuser detection algorithms in imperfect reception conditions," *Digital Signal Processing*, vol. 20, no. 1, pp. 166–178, 2010.

[21] J. J. Murillo-Fuentes and F. Perez-Cruz, "Gaussian process regressors for multiuser detection in ds-cdma systems," *IEEE Transactions on communications*, vol. 57, no. 8, pp. 2339–2347, 2009.

[22] D.-M. Bui and S. Lee, "Fast gaussian process regression for multiuser detection in ds-cdma," *IEEE Communications Letters*, vol. 21, no. 2, pp. 406–409, 2017.

[23] D.-M. Bui, H.-Q. Nguyen, Y. Yoon, S. Jun, M. B. Amin, and S. Lee, "Gaussian process for predicting cpu utilization and its application to energy efficiency," *Applied Intelligence*, vol. 43, no. 4, pp. 874–891, 2015.

[24] R. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013. [Online]. Available: http://books.google.co.kr/books?id=CGFbAgAAQBAJ

[25] D.-M. Bui, Y. Yoon, E.-N. Huh, S. Jun, and S. Lee, "Energy efficiency for cloud computing system based on predictive optimization," *Journal of Parallel and Distributed Computing*, vol. 102, pp. 103–114, 2017.

[26] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[27] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ACM Sigplan Notices*, vol. 44, no. 3. ACM, 2009, pp. 205–216.

[28] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "Vmware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.

[29] I. Sarji, C. Ghali, A. Chehab, and A. Kayssi, "Cloudese: Energy efficiency model for cloud computing environments," in *Energy Aware Computing (ICEAC), 2011 International Conference on*. IEEE, 2011, pp. 1–6.

[30] "What is ganglia?" http://ganglia.sourceforge.net/, accessed: 2015-08-13.

[31] A. Verma, P. Ahuja, and A. Neogi, "pmapper: power and migration cost aware application placement in virtualized systems," in *Middleware 2008*. Springer, 2008, pp. 243–264.

[32] E. T. Jaynes, *Probability theory: The logic of science*. Cambridge university press, 2003.

[33] H. Jeffreys, *Theory of Probability (Oxford Classic Texts in the Physical Sciences)*. Oxford University Press, 1998. [Online]. Available: https://www.amazon.com/Theory-Probability-Classic-Physical-Sciences/dp/0198503687?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0198503687

[34] G. L. Bretthorst, "The near-irrelevance of sampling frequency distributions," *Maximum Entropy and Bayesian Methods, W. von der Linden et al.(eds.)*, pp. 21–46, 1999.

[35] B. de Finetti, "Probabilities of probabilities: A real problem or a misunderstanding," *New Developments in the Applications of Bayesian methods*, pp. 1–10, 1977.

[36] B. De Finetti, "La prévision: ses lois logiques, ses sources subjectives," in *Annales de l'institut Henri Poincaré*, vol. 7, no. 1, 1937, pp. 1–68, translated into English by Henry E. Kyburg Jr., Foresight: Its Logical Laws, its Subjective Sources. In Henry E. Kyburg Jr. and Howard E. Smokler (1964, Eds.), Studies in Subjective Probability, 53-118, Wiley, New York.

[37] D. Heath and W. Sudderth, "De finetti's theorem on exchangeable variables," *The American Statistician*, vol. 30, no. 4, pp. 188–189, 1976.

[38] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms." *Neural Networks, IEEE Transactions on*, vol. 12, no. 2, pp. 181–201, Mar 2001.

[39] D. J. MacKay, "Introduction to gaussian processes," *NATO ASI Series F Computer and Systems Sciences*, vol. 168, pp. 133–166, 1998.

[40] S. Bochner, *Lectures on Fourier Integrals: With an Author's Supplement on Monotonic Functions, Stieltjes Integrals and Harmonic Analysis; Translated from the Original German by Morris Tenenbaum and Harry Pollard*. Princeton University Press, 1959.

[41] C. Chatfield, *The analysis of time series: an introduction*. CRC press, 2016.

[42] S. Ghosal and A. Roy, "Posterior consistency of gaussian process prior for nonparametric binary regression," *The Annals of Statistics*, pp. 2413–2429, 2006.

[43] A. W. van der Vaart and J. H. van Zanten, "Adaptive bayesian estimation using a gaussian random field with inverse gamma bandwidth," *The Annals of Statistics*, pp. 2655–2675, 2009.

[44] C. T. Brownlees, R. F. Engle, and B. T. Kelly, "A practical guide to volatility forecasting through calm and storm," 2011.

[45] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[46] M. Gibbs, "Bayesian gaussian processes for classification and regression," *PhD thesis, University of Cambridge, Cambridge, UK*, 1997.

[47] M. L. Stein, *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[48] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Courier Corporation, 1964, vol. 55.

[49] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[50] E. G. Tsionas, "Maximum likelihood estimation of stochastic frontier models by the fourier transform." *Journal of Econometrics*, vol. 170, no. 1, pp. 234–248, 2012.

[51] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization." in *NIPS*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 2546–2554.

[52] E. Rodner, A. Freytag, P. Bodesheim, and J. Denzler, "Large-scale gaussian process classification with flexible adaptive histogram kernels." in *ECCV (4)*, ser. Lecture Notes in Computer Science, A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., vol. 7575. Springer, 2012, pp. 85–98.

[53] R. Pace and J. P. LeSage, "Chebyshev approximation of log-determinants of spatial weight matrices." *Computational Statistics & Data Analysis*, vol. 45, no. 2, pp. 179–196, 2004.

[54] T. T. Cai, T. Liang, and H. H. Zhou, "Law of log determinant of sample covariance matrix and optimal estimation of differential entropy for high-dimensional gaussian distributions," *Journal of Multivariate Analysis*, vol. 137, pp. 161–172, 2015.

[55] T. W. Anderson, T. W. Anderson, T. W. Anderson, T. W. Anderson, and E.-U. Mathématicien, *An introduction to multivariate statistical analysis*. Wiley New York, 1958, vol. 2.

[56] R. J. Muirhead, "Aspects of multivariate statistical theory. john wiley and sons," *Inc., New York*, 1982.

[57] N. Goodman, "The distribution of the determinant of a complex wishart distributed matrix," *The Annals of mathematical statistics*, vol. 34, no. 1, pp. 178–180, 1963.

[58] H. H. Nguyen, V. Vu *et al.*, "Random matrices: Law of the determinant," *The Annals of Probability*, vol. 42, no. 1, pp. 146–167, 2014.

[59] J. Quinonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression." *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[60] J. de Baar, R. Dwight, and H. Bijl, "Speeding up kriging through fast estimation of the hyperparameters in the frequency-domain." *Computers & Geosciences*, vol. 54, no. 0, pp. 99–106, 2013.

[61] P. Sollich and C. K. I. Williams, "Understanding gaussian process regression using the equivalent kernel." in *Deterministic and Statistical Methods in Machine Learning*, ser. Lecture Notes in Computer Science, J. Winkler, M. Niranjan, and N. D. Lawrence, Eds., vol. 3635. Springer, 2004, pp. 211–228. [Online]. Available: http://dblp.uni-trier.de/db/conf/dsmml/dsmml2004.html

[62] L. Castro, M. Haque, M. Murshed, S. Saitoh, N. Tuan *et al.*, "Quadratic fourier transforms," *Annals of Functional Analysis*, vol. 5, no. 1, pp. 10–23, 2014.

[63] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. California Technical Pub., 1999.

[64] M. Stone and P. Goldbart, *Mathematics for physics: a guided tour for graduate students*. Cambridge University Press, 2009.

[65] M. T. Heideman and C. S. Burrus, *Multiplicative complexity, convolution, and the DFT*. Springer, 1988.

[66] A. Antoniou, *Digital signal processing*. McGraw-Hill, 2016.

[67] B. Champagne and F. Labeau, "Discrete time signal processing," *Class Notes for the Course ECSE-412, Department of Electrical & Computer Engineering, McGill University*, pp. 190–194, 2004.

[68] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. [Online]. Available: http://books.google.co.kr/books?id=mYm0bLd3fcoC

[69] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[70] H. Robbins and D. Siegmund, "A convergence theorem for non negative almost supermartingales and some applications," in *Herbert Robbins Selected Papers*. Springer, 1985, pp. 111–135.

[71] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain." Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-94-125, 1994.

[72] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis, "Improved fast gauss transform and efficient kernel density estimation." in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, Oct 2003, pp. 664–671 vol.1.

[73] T. I. Alecu, S. Voloshynovskiy, and T. Pun, "The gaussian transform." in *EUSIPCO2005, 13th European Signal Processing Conference*, 2005, pp. 4–8.

[74] L. Greengard and J. Strain, "The fast gauss transform." *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[75] Y. Yamamoto, "Efficient parallel implementation of a weather derivatives pricing algorithm based on the fast gauss transform," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006, pp. 8 pp.–.

[76] M. Spivak, S. K. Veerapaneni, and L. Greengard, "The fast generalized gauss transform." *SIAM J. Sci. Comput.*, vol. 32, no. 5, pp. 3092–3107, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1137/100790744

[77] R. S. Sampath, H. Sundar, and S. K. Veerapaneni, "Parallel fast gauss transform." in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.39

[78] V. Simoncini and D. B. Szyld, "Theory of inexact krylov subspace methods and applications to scientific computing." *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454–477, 2003.

[79] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 73, no. 2, pp. 325–348, 1987.

[80] "Boston housing dataset," http://lib.stat.cmu.edu/datasets/boston, accessed: 2016-12-10.

[81] "Uk land registry price paid dataset," http://data.gov.uk/dataset/land-registry-monthly-price-paid-data/, accessed: 2016-12-11.

[82] "What is google traces?" https://github.com/google/cluster-data/, accessed: 2016-01-21.

[83] H. Fangohr, "A comparison of c, matlab, and python as teaching languages in engineering," in *Computational Science-ICCS 2004*. Springer, 2004, pp. 1210–1217.

[84] X. Wu, *Performance Evaluation, Prediction and Visualization of Parallel Systems*, ser. The International Series on Asian Studies in Computer and Information Science. Springer US, 1999. [Online]. Available: http://books.google.co.kr/books?id=IJZt5H6R8OIC

[85] K. Chalupka, C. K. I. Williams, and I. Murray, "A framework for evaluating approximation methods for gaussian process regression." *CoRR*, vol. abs/1205.6326, 2012.

[86] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013.

[87] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," in *SPIE Astronomical Telescopes+ Instrumentation*. International Society for Optics and Photonics, 2004, pp. 221–232.

[88] "What is montage?" http://montage.ipac.caltech.edu/, accessed: 2016-08-02.

[89] T. K. Okada, A. D. L. F. Vigliotti, D. M. Batista, and A. G. vel Lejbman, "Consolidation of vms to improve energy efficiency in cloud computing environments," pp. 150–158, 2015.

Appendix A

List of publications

A.1 First author

A.1.1 SCI journals

1. **Dinh-Mao Bui**, Shujaat Hussain, Eui-Nam Huh, and Sungyoung Lee. "Adaptive replication management in HDFS based on supervised learning." *IEEE Transactions on Knowledge and Data Engineering* (**SCI, IF: 3.438**), 2016. https://doi.org/10.1109/TKDE.2016.2523510

2. **Dinh-Mao Bui** and Sungyoung Lee. "Fast Gaussian Process Regression for Multiuser Detection in DS-CDMA." *IEEE Communications Letters* (**SCI, IF: 1.988**), 2017. https://doi.org/10.1109/LCOMM.2016.2620430

3. **Dinh-Mao Bui**, YongIk Yoon, Eui-Nam Huh, SungIk Jun, and Sungyoung Lee. "Energy efficiency for cloud computing system based on predictive optimization." *Elsevier Journal of Parallel and Distributed Computing* (**SCI, IF:1.93**), 2017. https://doi.org/10.1016/j.jpdc.2016.11.011

4. **Dinh-Mao Bui**, Thien Huynh-The and Sungyoung Lee. "Early fault detection in IaaS cloud computing based on fuzzy logic and prediction technique." *The Journal of Supercomputing* (**SCI, IF: 1.326**), 2017. https://doi.org/10.1007/s11227-017-2053-3

5. **Dinh-Mao Bui**, Huu-Quoc Nguyen, YongIk Yoon, SungIk Jun, Muhammad Bilal Amin, and Sungyoung Lee. "Gaussian process for predicting CPU utilization and its application to energy efficiency." *Applied Intelligence* (**SCI, IF: 1.215**), 2015. https://doi.org/10.1007/s10489-015-0688-4

A.1.2 Conferences

1. **Dinh-Mao Bui**, Thien Huynh-The, Sungyoung Lee, and YongIk Yoon. "Complexity reduction for Gaussian process regression in spatio-temporal prediction." International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh, Vietnam, 2015.

2. **Dinh-Mao Bui**, Thien Huynh-The, Sungyoung Lee, Bin Li, and Jin Wang. "Replication Management Framework for HDFS Based on Prediction Technique." Third International Conference on Advanced Cloud and Big Data (CBD), Yangzhou, China, 2015.

3. **Dinh-Mao Bui**, Thien Huynh-The, YongIk Yoon, SungIk Jun, and Sungyoung Lee. "EAP: Energy-Awareness Predictor in Multicore CPU." International Conference on Ubiquitous Information Technologies and Applications (CUTE), Cebu, Philippines, 2015.

4. **Dinh-Mao Bui**, Thien Huynh-The, Sungyoung Lee, YongIk Yoon, and SungIk Jun. "Energy savings in processor based on prediction technique." International Conference on Information Networking (ICOIN), Kota Kinabalu, Malaysia, 2016.

5. **Dinh-Mao Bui**, Thien Huynh-The and Sungyoung Lee. "Fuzzy Fault Detection in IaaS Cloud Computing." The 10th International Conference on Ubiquitous Information Management and Communication (IMCOM), Danang, Vietnam, 2016.

6. **Dinh-Mao Bui** and Sungyoung Lee. "Placement Scheduling for Replication in HDFS Based on Probabilistic Approach." In International Conference on Smart Homes and Health Telematics (ICOST), Wuhan, China, 2016.

7. **Mao Bui Dinh**, Muhammad Idris, Sungyoung Lee, "Replication: Concept and Strategies in HDFS", Korea Computer Congress (KCC), Republic of Korea, 2014.

8. **Dinh-Mao Bui**, Sungyoung Lee, "Computational complexity issue of Gaussian Process in prediction and classication", Korea Computer Congress (KCC), Republic of Korea, 2015.

A.1.3 Patents

1. Sungyoung Lee, **Bui Dinh Mao**, ”CPU 사용률 제어 장치 및 방법”, 출원인: 경희대학교 산학협력단, 등록번호: 10-1657414, 2016년 9월 7일.

2. Sungyoung Lee, **Bui Dinh Mao**, ”복잡성 감소 기반 DS-CDMA에서 다중 사용자 검출 개선 방법”, 출원인: 경희대학교 산학협력단, 출원번호: 10-2016-0053521, 2016년 4월 29일.

3. Sungyoung Lee, **Bui Dinh Mao**, ”Improving the Multiuser Detection in DS-CDMA based on Complexity Reduction”, Application No. PCT/KR2016/005771, May 31, 2016.

A.2 Co-author

A.2.1 Journals

1. Amin Muhammad Bilal, Wajahat Ali Khan, Shujaat Hussain, **Dinh-Mao Bui**, Oresti Banos, Byeong Ho Kang, and Sungyoung Lee. ”Evaluating Large-Scale Biomedical Ontology Matching Over Parallel Platforms.” IETE Technical Review (SCIE, IF: 1.33), 2016 http://dx.doi.org/10.1080/02564602.2015.1117399

2. Amin Muhammad Bilal, Oresti Banos, Wajahat Ali Khan, Hafiz Syed Muhammad Bilal, Jinhyuk Gong, **Dinh-Mao Bui**, Soung Ho Cho et al. ”On curating multimodal sensory data for health and wellness platforms.” Sensors (SCIE, IF: 2.677), 2016. http://dx.doi.org/10.3390/s16070980

3. Thien Huynh-The, Banos Oresti, Ba-Vui Le, **Dinh-Mao Bui**, Yongik Yoon, and Sungyoung Lee. ”Traffic behavior recognition using the Pachinko allocation model.” Sensors (SCIE, IF: 2.033), 2015. http://dx.doi.org/10.3390/s150716040

A.2.2 Conferences

1. Huynh-The Thien, Oresti Banos, Ba-Vui Le, **Dinh-Mao Bui**, Sungyoung Lee, Yongik Yoon, and Thuong Le-Tien. ”PAM-based flexible generative topic model for 3D interactive activ-

ity recognition." International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh, Vietnam, 2015.

2. Huynh-The Thien, Oresti Banos, Ba-Vui Le, **Dinh-Mao Bui**, Sungyoung Lee, Yongik Yoon, and Thuong Le-Tien. "Background subtraction with neighbor-based intensity correction algorithm." International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh, Vietnam, 2015.

3. Huynh-The Thien, **Dinh-Mao Bui**, Sungyoung Lee, and Yongik Yoon. "Interactive Activity Recognition Using Articulated-Pose Features on Spatio-Temporal Relation." International Conference on Ubiquitous Information Technologies and Applications (CUTE), Cebu, Philippines, 2015.

4. Huynh-The Thien, **Dinh-Mao Bui**, Sungyoung Lee, and Yongik Yoon. "Improved PAM-based traffic behavior recognition using trajectory-wise features." International Conference on Big Data and Smart Computing (BigComp), Hong Kong, China, 2016.

5. Idris Muhammad, **Mao Bui Dinh**, and Sungyoung Lee. "Intelligent search in digital documents using MapReduce Korea Computer Congress (KCC), Republic of Korea, 2014.