Dynamic Models under Uncertainty for Large-Scale Enterprise Systems

Philip C. Placek

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Wolf Kohn, Chair

Zelda B. Zabinsky, Chair

Jonathan Cross

Program Authorized to Offer Degree:
Industrial & Systems Engineering

University of Washington

Abstract

Dynamic Models under Uncertainty for Large-Scale Enterprise Systems

Philip C. Placek

Co-Chairs of the Supervisory Committee:
Research Professor Wolf Kohn
Industrial & Systems Engineering

Professor Zelda B. Zabinsky
Industrial & Systems Engineering

Many enterprise applications such as assortment planning, inventory control and supply chain management rely on forecasting, optimization and machine learning methodologies. While many methods have been developed for static deterministic systems there is a need for methodologies that are dynamic, involve uncertainty and are computationally practical. Additionally, new methods are needed for online applications that have a constant stream of new data arriving. Current methods are often unable to handle these online applications as resolving the entire problem every time new data arrives is intractable.

This dissertation presents new models and algorithms for large-scale dynamic systems. The new methodology utilizes a framework for dynamic models applicable to online applications where there is not complete information and streams of new data arrive. The methodology for dynamic models consists of a probabilistic forecaster and an algorithm for approximating the solution to nonlinear least squares problems for updating parameters. The methods are developed to scale well with a large amount of data and to be used in near real-time applications.

The models and algorithms developed can be applied to an enterprise system. We analyze the methods developed in this dissertation for the example of demand forecasting. Like many

problems in enterprise systems, demand forecasting problems are dynamic and change over time and have a great deal of uncertainty. Furthermore demand forecasting problems are often large, containing either a lot of data or a complex model with many parameters.

Cloud computing is becoming a major computational resource for streaming large scale data. Cloud computing can be used to increase the effectiveness of shared resources and solve problems that require a massive amount of data. However, it is not always clear how to maximize the effectiveness of shared resources on the cloud. A meta-control framework is proposed to achieve good performance for cloud applications. The measures of performance for the meta-control are latency, delay, restart and cost. This framework will design rules to characterize cost, quality of service and other criteria for implementing the meta-control feedback schema.

Supervisory committee:

Wolf Kohn and Zelda B. Zabinsky (Co-Chairs)

Mehran Mesbahi (GSR)

Mike Ehrenberg

Jonathan Cross

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Enterprise systems contain large scale optimization problems that often need to be solved quickly as new data arrives. Problems in machine learning [3] and forecasting [13] use regression analysis, simulation and Markov models for decision making [8, 15, 29]. These problems are dynamic and change over time, involve uncertainty and may be in an online environment. These online environments contain data which is arriving periodically as opposed to the offline scenario where all the information and data is known beforehand. Traditional models for enterprise systems are frequently based around static optimization models in an offline environment. With new technology and the era of big data these models are no longer applicable. Now, dynamic models under uncertainty are needed which can scale well when all the information may not be known beforehand and arrives periodically.

Applications involving prediction and forecasting are typically dynamic in nature since the system changes over time. Static models are frequently used to get a snapshot of the state of the system. Because they are independent of time, static models are often more rigorous and require less resources to analyze. However static models fail to give any insight into how a system evolves or changes over time. Despite being more difficult to solve, dynamic models are more grounded in reality as they can describe how a system changes over time.

Stochastic models are useful for making predictions, deriving information and analyzing uncertainty. Many stochastic models make assumptions about the probability distributions of the data or parameters. However this simplification does not always reflect reality. Despite the difficulties of solving and analyzing stochastic models, there is a need for algorithms to build the probability distribution over time from data. Doing so allows for more accurate

predictions and forecasts to be made.

In some applications, such as online retailing and stock market pricing, new data (in the form of point-of-sales or transactions) may be occurring every microsecond. Algorithms are frequently built around a model which assumes that all or enough data is given beforehand. These algorithms may have difficulty if not enough data is given beforehand. They are also not tractable when new data is arriving frequently as resolving the entire problem is too inefficient. Models and algorithms are needed to quickly adapt to the new information as it becomes available while still using the knowledge gained from the previous solution.

Traditional forecasting techniques use regression analysis, simulation and Markov models to make predictions. These methods may have problems scaling when the amount of data increases or if the model function is highly nonlinear and contains a lot of parameters. Technology has allowed us to store and analyze large quantities of data occurring every microsecond. Models and algorithms need to be designed to scale up as problems evolve and grow over time.

1.2 Research Contribution

The contribution of this dissertation is the development of models and algorithms for enterprise systems. The first contribution is a generic framework for dynamic models with uncertainty that is applicable to online problems. This framework can offer scalability as the problem grows in size and allows for quick solutions to be obtained which is crucial in many applications. The models and algorithms designed in the rest of this dissertation fit into this framework. The primary example used to demonstrate this framework is demand forecasting for retailers.

The second contribution is a probabilistic forecaster which propagates a probability distribution directly (as opposed to propagating the mean). The forecaster is designed around the use of increments which are constructive relative to the mean of the state. This formulation is practical for many applications and allows for forecasts to be made without assuming that the data has a certain probability distribution. For example, in demand forecasting

knowing the mean and the variance is not enough information for many decisions. The use of increments also allows forecasts to be made in a timely fashion making use of online data.

The third contribution is a new methodology for approximating solutions to nonlinear least squares problems using the concept of I-frames and the fast marching algorithm. The methodology is developed to solve online problems, where new data is constantly arriving. This methodology benefits from being scalable as the number of parameters increases and as the amount of data increases. Furthermore, only the gradient of the model function is required, as opposed to the Hessian matrix. The approach uses the concept of I-frames from imaging and animation to divide the problem into smaller optimization problems that reduce the overall computation while still providing a good approximation.

The rest of the dissertation is organized as follows. Chapter 2 examines some of the ways cloud computing is being used to solve optimization problems. We also review parameter estimation and algorithms for solving nonlinear least squares problems. Chapter 3 presents a generic framework for dynamic models applicable to cloud environments. In Chapter 4 we present an incremental probability forecaster applicable to demand forecasting. Finally in Chapter 5 we present an algorithm for solving nonlinear least squares problems using the concept of I-frames from imaging and animation. We conclude in Chapter 6 and explain potential future work.

Chapter 2

LITERATURE REVIEW AND BACKGROUND

This chapter reviews the current literature and provides some background on the problems that are addressed in later chapters. We start by reviewing cloud computing and meta-control. The work developed in Chapters 3, 4 and 5 can be applied to cloud computing environments. Then we review forecasting, parameter estimation and nonlinear least squares problems. Chapter 4 develops a new, incremental probability forecaster and Chapter 5 presents a new methodology for solving nonlinear least squares problems.

2.1 Cloud Computing

Scheduling workflow applications in a cloud environment to optimize cost and completion time has been examined using particle swarm optimization [49] and simulated annealing [24]. These models examine the tradeoffs between the cost of using the cloud and computation time. However these models are only considered for static cases where the demand for resource requirements are known ahead of time. That is, there is complete information about the demand, what jobs need to be completed and how long the jobs will take to complete.

The INFORMS 2015 conference provided over two dozen talks about how the cloud can be utilized for operations research and optimization [21]. Many of these talks focused on the business decision of switching to the cloud. However there is little work on the operational decisions to be made once the cloud is already being used.

Provisioning resources from the perspective of the owner of the cloud is studied in [45] and [18]. These papers use dynamic programming and optimal control to allocate virtual machines or processors in the cloud based on the fluctuations in demand. However, the work

contributed only examines resource allocation from the cloud perspective. It is unclear how the user should best allocate the cloud resources to solve their problems in the cloud.

It is an open question of how to best utilize the resources in the cloud for solving problems which are occurring in real-time. Often, there is very little information about these problems, how long they will take to solve and how many resources are required to solve them in an efficient manner. There is a need to develop a framework or architecture to handle such a situation. This framework needs to account for all the variability in using the cloud: latency, delay, restart and cost while still providing a quality solution in a short amount of time. Furthermore, this framework should take advantage of the inherent structure of the cloud by utilizing its resources to the fullest potential. For example, we should be able to acquire and release resources on demand based on the expected amount of work to be done.

In Chapter 3, we present a framework for dynamic models which may be tailored to solve these problems in a cloud environment.

2.2 Meta-Control

Meta-control was introduced in 2006 as a generic methodology to iteratively determine algorithm parameters that optimize the performance of a deterministic optimization algorithm [28]. The meta-control methodology was extended to control stochastic optimization algorithms in [44]. This meta-control was used to control the temperature parameter of a simulated annealing type algorithm [43]. Meta-control was also used for solving binary integer problems in continuous time in [58], and in discrete time in [63].

The main idea is to solve a continuous time optimal control problem and get a feedback signal on the parameters based on the current state of the algorithm. This concept can be summarized in four basic steps:

1. Given an algorithm, find a dynamic operator that approximates the algorithm's behavior.

2. Identify the parameters of the algorithm to control.

3. Formulate and solve a control problem to determine a feedback signal on the parameters according to the desired performance metrics and updated data.

4. Implement the algorithm with the adjusted parameters and continuously provide data to the control problem.

Meta-control is discussed further in Section 3.3 for implementation in the cloud.

2.3 Dynamic Models for Forecasting and Parameter Estimation

Enterprise systems contain many forecasting problems to aid in decision-making, such as assortment planning, inventory control and supply chain management [13]. These decisions may be essential to the enterprise system and often involve a great deal of risk and uncertainty. It is crucial that forecasts are made with the highest degree of accuracy possible while still reflecting reality.

To make a forecast, the user needs to determine a model that can incorporate all of the known data to make an accurate prediction about the future. Ideally, the model should be dynamic, involve uncertainty, able to handle streams data and offer scalability. This poses a variety of challenges as developing an efficient model or system to contain these elements is not always easy or computationally practical.

Many forecasting models are based on an autoregressive moving-average (ARMA) model [13, 29]. ARMA models are reliable in some applications but it is often hard to interpret the coefficients. Sometimes the coefficients are hard to find and there is a danger of overfitting. Furthermore ARMA models are imposed with constraints to assume the process is stationary. This makes it easier to interpret the data, but it is not realistic. Some also assume that the noise in the system is Gaussian.

Probabilistic forecasts make predictions that are given with certain levels of uncertainty. However most probabilistic forecasts propagate the mean and the covariance directly, which assumes that the data follows a certain distribution. Maximum likelihood and Kalman filters have well-developed algorithms to use. However these methods assume the data follows a

certain probability distribution (e.g., Gaussian). There is a need for a forecaster that is dynamic without assuming a probability distribution.

In Chapter 4 we develop a probabilistic forecaster based on the use of increments around the mean of the state. This forecaster is applicable in many fields and benefits from not assuming a probability distribution while being computationally practical in online environments.

2.4 Nonlinear Least Squares

Regression analysis is a statistical tool for investigating the relationship between variables. For example in demand forecasting we might be interested in how changing price affects the demand, or if the demand is affected by the weather. Usually there is a model, given by a predictor function, that explains how the data affects the prediction. Knowing the relationship between the data and the prediction is crucial as an accurate prediction can be made should new data arise, either in practice or in theory. These predictor functions have a set of parameters (or coefficients) which need to be estimated from the data. A predictor function can be linear or nonlinear with respect to the parameters. Sometimes, such as in exponentially weighted models, these parameters have constraints imposed on them as well. The standard approach to estimate these parameters is the method of least squares, which finds a solution that minimizes the sum of the squares of the errors made in the results of the model function.

Many fields, such as robotics [56], optics [12] and machine learning [3] use a nonlinear least squares problem to estimate parameters of a complex model that fit the data well. Determining the parameters or coefficients of the model function provides the ability to predict future behavior. This is needed for forecasting a variety of things such as stock prices, item demands and weather [60].

Many model functions are chosen to be linear with respect to the explanatory (or independent) variables. Linear model functions are computationally easy to fit to the data because the linear least squares problem has a closed form solution that can be solved in a

finite number of standard operations [5]. However many systems are not well approximated by a linear function and it is necessary to solve the more difficult nonlinear least squares problem. Furthermore, while many traditional methods for solving one instance of a nonlinear least squares problem exist, these methods often do not scale well as the number of parameters increases nor as the amount of data increases [39, 47].

While solving one instance of a least squares problem has been studied extensively, some online applications require the problem to be repeatedly resolved as new data points arrive. For machine learning and demand forecasting, data points may be arriving multiple times each second and resolving the entire problem with each new point is not practical or efficient [52]. New methods are needed to efficiently solve the nonlinear least squares problem while taking advantage of the sequential nature of the data.

Formally, consider a set of m data points, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, and a model function $y = f(\mathbf{x}, \mathcal{P})$. This model function depends on the variable $\mathbf{x}$ as well as a set of n parameters, $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$, with $m \geq n$. Note that $\mathbf{x}$ is not limited to a single value and may instead be a vector of points. For example, in demand forecasting the model function may include point of sale (POS) data, order amounts, weather and promotional sales. All these items may affect the forecast and thus $\mathbf{x}$ is a vector containing all these values. Then y, which is the observed output associated with the input $\mathbf{x}$, would be the estimated demand.

The goal is to find the vector $\mathcal{P}$ of parameters such that the model best fits the given data in the least squares sense. Let r, commonly called the residual, be a vector function, $r : \mathbb{R}^n \to \mathbb{R}^m$ with m components $r_i, i = 1, \ldots, m$. Each of the m components measures the error made in the results of the model function, $r_i = y_i - f(\mathbf{x}_i, \mathcal{P})$. In the example of demand forecasting, r_i would be the difference (or error) between the observed demand at time i and the forecasted demand. In this dissertation, the model function is the same across the m data points. This is usually the case for demand forecasting, however some applications in imaging and machine learning do not have this form.

A good set of coefficients for the predictor function is a set that minimizes the sum of

the squares of the errors. This leads to the following optimization problem:

$$\min_{\mathcal{P}} G(\mathcal{P}) \tag{2.1}$$
$$\text{where } G(\mathcal{P}) = \frac{1}{2}\sum_{i=1}^{m}\Big(r_i(\mathcal{P})\Big)^2 = \frac{1}{2}r(\mathcal{P})^T r(\mathcal{P}),$$

where $\mathcal{P}$ is the set of parameters (or coefficients) that need to be estimated.

As an example, consider the following model function which gives the amount of a component in a chemical reaction over time (see [17]),

$$f(x, \mathcal{P}) = \mathcal{P}_1 + \mathcal{P}_2 e^{-\mathcal{P}_3 x}.$$

Given some observations of the component amount, $y_1, \ldots, y_m$ at times $x_1, \ldots, x_m$ the objective is to minimize the sum of the residuals at each time point,

$$\min_{\mathcal{P}} \frac{1}{2}\sum_{i=1}^{m}(r_i(\mathcal{P}))^2$$
$$\text{where } r_i(\mathcal{P}) = y_i - \mathcal{P}_1 - \mathcal{P}_2 e^{-\mathcal{P}_3 x_i}.$$

Note that this is a *nonlinear* least squares problem as each r_i cannot be expressed as a linear combination of the parameters, $\mathcal{P}$. The rest of the dissertation assumes that r is a nonlinear function as many efficient algorithms exist for the case where r is linear with respect to $\mathcal{P}$ [5, 39, 47].

Problem (2.1) is called the offline nonlinear least squares problem. For this offline problem, all of the data is known beforehand. An online least squares problem refers to the situation where data is arriving sequentially and the least squares problem needs to be updated with each new data point. Traditionally, this leads to solving a sequence of least squares minimization problems. However this is computationally expensive and does not exploit the sequential structure of the overarching problem. This is limiting in real-time applications where speed is crucial [52].

Because Problem (2.1) does not have a closed form solution, iterative methods are used to find a solution [5]. These iterative methods are typically based on Newton's method for

optimization. One Newton-based method is the Gauss-Newton method which is a well-known algorithm for solving Problem (2.1). This method is derived by linearly approximating the components of r in the neighborhood of $\mathcal{P}$ [39].

The Gauss-Newton method is appealing upfront: the Hessian does not need to be calculated (and in many situations it is closely approximated). However the Gauss-Newton method can only guarantee linear convergence when the Jacobian matrix, J, is full rank. If J is not full rank then the algorithm cannot guarantee convergence [16]. Furthermore, when n and m are both large and J is dense then solving for the step direction can become quite expensive.

The Levenberg-Marquardt method is similar to the Gauss-Newton method except the line search is replaced with a strategy analogous to a trust-region. This avoids one of the weaknesses of the Gauss-Newton method when J is rank-deficient. There is a damping parameter, μ, which influences both the direction and the size of the search step, reducing the need to perform a specific line search. While the Levenberg-Marquardt method is more robust than the Gauss-Newton method (it performs well even if it starts far away from a minimum) these two algorithms can only guarantee a linear convergence rate when J is full rank. However if the solution is close to the minimum then the convergence rate is quadratic.

Solving for the step direction can present some problems as inverting a large matrix (usually done by solving a system of linear equations) can be costly. A Quasi-Newton method is any method that approximates the inverse of the Hessian directly. One of the most well-known approximations is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm which has gained widespread use in many applications [47].

A hybrid method which combines the Levenberg-Marquardt method with a Quasi-Newton method was proposed in 1988 [37]. The iteration starts with a series of steps with the Levenberg-Marquardt method (since the Quasi-Newton method does not perform well when it is far away from a minimum). Then, if the performance indicates that $G(\mathcal{P})$ has improved to a significant level then the algorithm switches to the Quasi-Newton method. This hybrid method has the benefit of an overall faster convergence time since the Quasi-Newton method

guarantees superlinear performance and the algorithm only uses the Levenberg-Marquardt method for the case where its convergence is quadratic.

However the Quasi-Newton method is based on having an approximation to the Hessian at the current iterate. In practice the BFGS update has worked the best and in many cases guarantees superlinear convergence [16, 47]. For general, nonconvex predictor functions the BFGS update has not been shown to converge. Furthermore, implementing an algorithm with a BFGS update can be difficult due to the need to store a dense, possibly large Hessian matrix. This issue has been overcome with the advent of the limited-memory BFGS (L-BFGS) algorithm which only requires a linear amount of memory. The L-BFGS algorithm only stores a few vectors that implicitly approximate the Hessian matrix. These vectors are used to do operations requiring the Hessian matrix. For some machine learning problems these algorithms have shown promise [40].

A secant version of the Levenberg-Marquardt algorithm can be implemented for the situation where the Jacobian matrix is unavailable (e.g., see [39]). These implementations require approximating the Jacobian matrix by finite differencing. The resulting algorithms are often very costly in terms of computation since finite differencing is performed at every iteration in order to ensure that the approximation to the Jacobian is close enough to the real Jacobian.

While all the algorithms discussed so far are able to solve some set of problems efficiently, difficulties arise when n and m are both very large. Often, these algorithms do not solve such large problems in a time efficient manner and the amount of memory required to perform the necessary computation is substantial. For many applications, $r_i(\mathcal{P})$ depends only on a few of the elements in $\mathcal{P}$ and the resulting Jacobian matrix is sparse [39]. Many algorithms exist for exploiting sparse Jacobians [46] but this simplification is not always possible. Thus there is a need for an algorithm to solve large scale nonlinear least squares problem where storing and computing a Hessian matrix can be avoided without assuming a sparse Jacobian matrix.

Recent advances of online nonlinear least squares algorithms have achieved superlinear

convergence rates for problems with a sparse Jacobian matrix [22, 52]. While sparsity of the Jacobian frequently occurs in some applications such as bundle adjustment [57], many fields such as robotics [56] and price forecasting [35] do not have this simplification. An algorithm for online nonlinear least squares is needed for these applications where sparsity cannot be assumed.

Chapter 5 develops a new methodology for solving online nonlinear least squares problems. First, we present a generic framework for dynamic models. This framework can offer scalability and allows for solutions to be obtained when data is being streamed. This framework can be utilized in a cloud environment to improve performance and the methods developed in Chapters 4 and 5 fit into this framework.

Chapter 3

A FRAMEWORK FOR DYNAMIC MODELS IN CLOUD ENVIRONMENTS

3.1 Introduction

Enterprise systems contain many online applications where streams of information arrive over time. These applications often have problems, such as inventory control and forecasting, where quick decisions need to be made. With the advent of new technology companies have access to large amounts of data and the models to solve these problems are very complex.

Current methods and algorithms to solve these online problems were developed for static systems where all the information is known beforehand. To use these methods we would have to resolve the entire problem every time new information is obtained. This is no longer practical as new data may be arriving every second or the model is too complex to solve in a reasonable amount of time. Instead we need to adjust our models and algorithms to accommodate the large amount of data arriving over time.

This chapter presents a generic framework for dynamic models to solve online problems well-suited to handle many applications in enterprise systems. We present the framework in Section 3.2 and in Section 3.3 we discuss how cloud computing and meta-control can be used to aid in the solution process. The methodologies and algorithms in Chapters 4 and 5 are designed to fit into this framework.

3.2 A Generic Framework for Dynamic Models

The framework is designed around two main ideas. First, when new data arrives we should be able to use the knowledge from the previously obtained model to get a new model. This is highly beneficial as data may be arriving on the order of microseconds and treating every

instance of the problem as separate is not efficient. For example, if we are using a model to forecast demand and need to adjust it (by modifying the parameters of the model) to better match new data, we can use the old parameters as a starting point. This is likely to save computation time because if we discarded the old solution (i.e., the parameter values) entirely we would have to search the solution space all over again to find a good set of parameters.

Second, as long as a model is good enough we should continue to use it instead of wasting time finding the best or perfect model. Many models are complex so that finding the optimal model is too time consuming or resource consuming. Some problems may be NP-hard and it might be too difficult to prove that the model we have is optimal. For example, in demand forecasting if the model is correctly predicting sales for an item there is no need to adjust the model. We should continue to use the model to predict sales until it is no longer making accurate predictions, regardless of how long ago the model was adjusted.

A design of this framework is given in Figure 3.1. Given a model and possibly some historical data, the framework evaluates how close the model prediction is to reality. If the model is good then we keep using it until new data arrives. If the model is not good, then the framework adjusts it to better match reality.

Note that the model is only adjusted when the comparison indicates the model is no longer accurate and *not* every time new data arrives. This presents a tradeoff between model accuracy and computation time. If the criterion for an accurate model is very strict, the framework will need to adjust the model often which costs resources and time.

An example of this process for forecasting is illustrated in Figure 3.2. The initial forecast found at time $t = 1$ is fairly close to the observations at $t = 2$ and $t = 3$. However, the observation at $t = 4$ is quite far away from the initial forecast. We decide that our initial forecast is no longer performing well and calculate a new forecast at $t = 4$. We also use part of the data from the initial forecast in obtaining a new forecast.

As illustrated in Figure 3.2, this framework is well-suited for handling online problems in enterprise applications, particularly forecasting. However there are decisions to be made

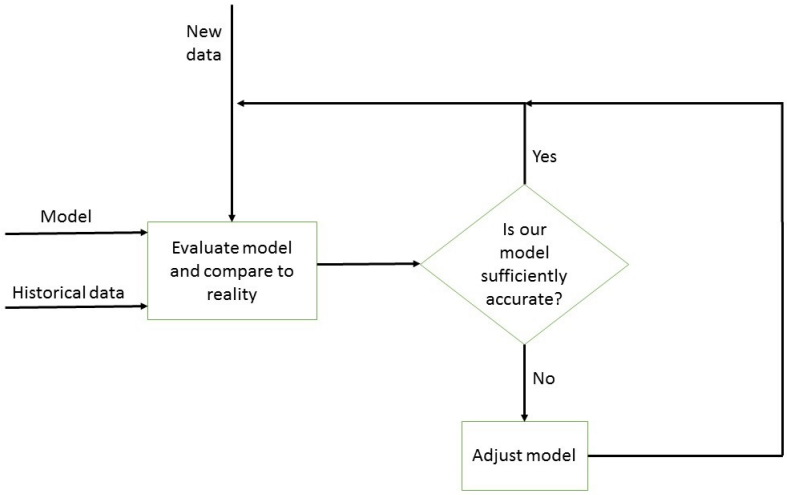

Figure 3.1: Given a model and historical data, the framework compares the model to reality. If the model is not sufficiently close to the data, then the model is adjusted. As new data arrives the model is again compared to the data.

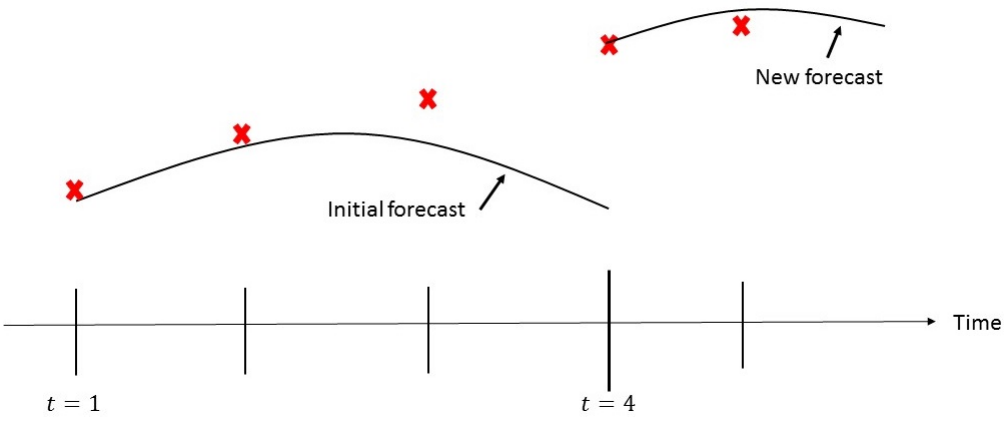

Figure 3.2: Framework for online forecasting. The dark x's represent data points (observations). Given an initial forecast each new data point is compared to the forecast. At time $t = 4$ the difference between the data point and the initial forecast is large enough that a new forecast is generated.

when using this framework: evaluating how well the model reflects reality and how to update the model if necessary. Usually it is up to the user to decide a threshold level between the model forecast and the real observations that initiates a model update. This decision may depend on a lot of factors: the problem we are trying to solve, what type of model we are using, constraints on our solution, etc. Updating the model can be a simpler process as we may already have the necessary algorithms to adjust our model. However these algorithms may be costly to perform.

The contributions in Chapters 4 and 5 of this dissertation fit into this framework. In Chapter 4 a probabilistic forecaster is developed which is adept at performing online forecasts and can quickly update when new data is obtained. Then in Chapter 5 an iterative algorithm for solving nonlinear least squares is developed which only updates if the solution deviates from reality. Next we discuss how cloud computing and meta-control can be useful tools for this framework.

3.3 Implementing the Framework in a Cloud Environment

Many online problems in enterprise applications need to solved or updated continuously. For example in demand forecasting it may be the case that thousands or even millions of items need to be forecasted every day. For some companies, such as online retailers, forecasts may need to be done multiple times a day. Ideally, a forecast should be recomputed every time a new sale is made. However, this proposes an immediate problem in scalability as online transactions may be happening on the order of microseconds. This leads to solving an enormous number of forecasting problems with very little time to do so.

A cloud implementation of the framework developed in Section 3.2 is used to help address these challenges. The cloud has access to streams of data in real-time and can store and access large amounts of data efficiently. Furthermore, the cloud has access to on-demand resources, useful when many problems need to be solved at the same time. Taking advantage of the cloud while utilizing the sequential nature of the framework in Section 3.2 allows for the enormous amount of computation to become tractable.

Since there is a nominal and opportunity cost of using the cloud the implementation will be parallelized to ensure a quality solution while minimizing the resources and time used in the cloud. Furthermore, decisions need to be made on how best to acquire and release resources based on how much computation is required. Thus a meta-control will be established over the resources of the cloud and the computation required of the framework. This meta-control will decide how and when to acquire and release processors in the cloud which are performing the computation. Meta-control is discussed in more detail in Section 3.4. First, implementing the generic framework for dynamic models in the cloud is presented.

The implementation is built around four key metrics of using the cloud:

1. Latency - time interval between cloud processors communicating with data storage or other processors based on the mechanics of the system (such as the speed of light and the medium being used, fiber optic cables, wireless, etc.)

2. Delay - time interval that exists when a processor performs a function (dependent upon the processing power of the processor itself).

3. Restart - time interval to stop a processor performing a function and having the processor restart the function.

4. Cost - monetary cost of using the cloud, dependent on how many resources are being used and the schematics of the resources being used (processing power, memory, etc.)

The cloud implementation assumes that evaluating the model and updating the model in Figure 3.1 are given as separate "black-boxes". That is, one inputs the model and the data and the black-box will output either a decision on whether to adjust the model or to continue to use the model. For simple models, explicit equations for these black-boxes may be known. For example, if we are using an ARMA model to forecast demand the decision on whether to adjust the model may be based on simply comparing the model's output to reality for the most recent day. If the comparison exceeds a certain threshold then the model is adjusted.

The black-box for updating the ARMA model may solve a nonlinear least squares problem to update the parameters of the ARMA model.

The model adjustment black-box in Figure 3.1 may be costly in terms of computation time or resources. If our model is really complex then evaluating it and comparing it to reality may be time consuming. Updating the model may take a lot of time if we are using a complex algorithm that does not run in polynomial time. In the best case scenario, the model update uses the information from the previous model to adjust itself quickly. If this is not possible we can run multiple instances of the model update in the cloud to improve the accuracy of the model.

Note that restarting computation on black-boxes may use the same processors. Acquiring and releasing a processor is time consuming and often variable. Instead, if a processor is performing a calculation that is determined to be not worthwhile, the processor will restart in one of two ways. The first way is to restart the calculation the processor started performing. For example, if a processor is updating a model that requires a random starting point and convergence criteria are not met within a certain time frame, then the processor will restart the calculation with a new random starting value. The second way is to restart with an entirely new calculation. If a processor is failing to update a model the processor will stop the calculation and pick up a new job.

The implementation for the generic framework in the cloud is shown in Figure 3.3. The implementation includes a director, who makes all the decisions regarding the performance; data storage, which represents the state of the system; and a collection of workers who perform a set of tasks. The director is responsible for making the key decisions: whether to add or remove workers (processors), when to restart a processor or a task should a failure occur and monitoring the overall state of the algorithm. Note that the director only communicates with the data storage. No direct communication is passed between the director and the workers nor is communication passed between workers. Instead, the director and the workers read and write from data storage only, which minimizes the horizontal and vertical communication (which is highly variable and time consuming).

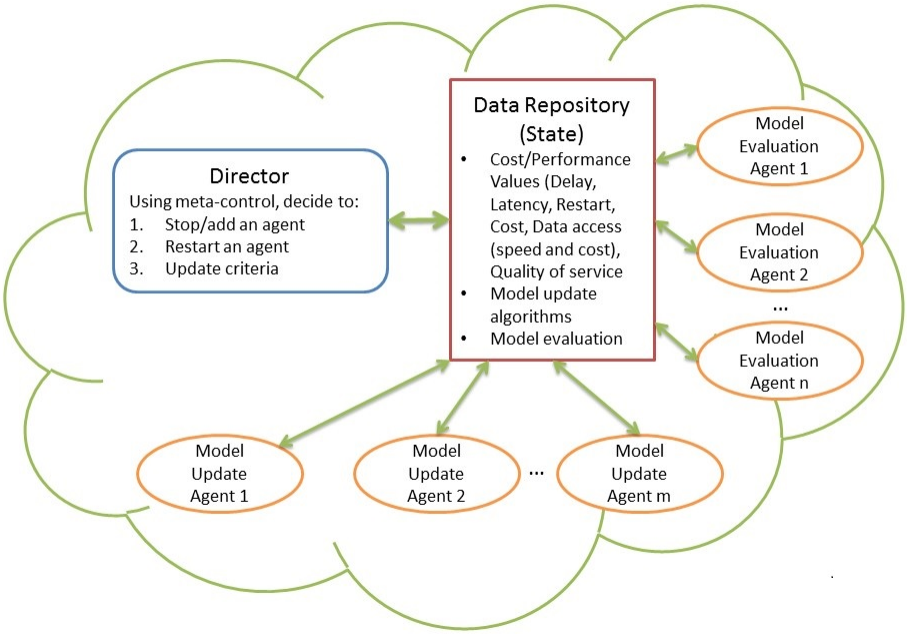

Figure 3.3: Implementation of the generic framework presented in Section 3.2 in the cloud. The director makes all the decisions regarding the agents, which perform the necessary calculations. The director and the agents only communicate with the data storage to minimize the horizontal and vertical communication.

The data storage contains two queues, one for model evaluation (and comparison) and one for model updating. Likewise there are two types of workers, one performs the model evaluations and the other performs the model updating. These workers look to their respective queue for jobs to do. When there is a job they remove it from the queue and write to the data storage that the job is being processed (which includes a time stamp). After finishing their job they write the results back to the data storage as well as changing the job status. If necessary, they will also write to other queues for more jobs to be completed (for example, after finishing a model evaluation the worker may write to the model update queue to indicate that the model needs to be updated).

Special considerations can also be made using the flexible data storage in the Microsoft Cloud. Since blobs and tables do not have a strict structure which must be adhered to, data may be stored in a way that pertains to the design. This will allow the director to make a more-informed decision in a faster amount of time. For example, it might be desirable to record the average run time it takes for the black-boxes to update a model. A traditional approach to recording this information would be to have a big table which contains several columns of information, for example, two columns with the start and end time stamps and a third column with the difference between the two. However, this is not a practical approach of storage it terms of scalability. Instead, the Microsoft Cloud allows the user to break data down into a format of their choosing. For this example, data can be broken down by item (a unique number assigned to the model) and then timestamps. Then, there may be an aggregate data set which includes all the time stamps. It is important to distinguish these two methods of accessing and storing data. In the first, traditional method, accessing data involves going to the table, then finding the relevant rows or columns in the table and looking at the values. In the second method, data is accessed by first specifying a model number, then specifying that start and end time stamps are needed, then specifying to subtract the two values. The cloud is designed in this way for scalability, as other data can be accessed in the same way, by first specifying a model number, then asking for the relevant data. By breaking the data down in this format, space is saved while still allowing for data to be

combined for aggregate analysis.

Note that in this implementation the director may be a bottleneck for computation, as the director must make decisions based on the results of all the workers. To help address this issue, the director will make decisions at regular time intervals instead of evaluating a stream of data and making decisions every microsecond.

This implementation in the cloud is very generic and any optimization algorithm can be implemented in this way. Population-based algorithms benefit the most from this framework as these algorithms are not only parallelized but have their population controlled to ensure computation is not wasted. Section 3.4 describes a meta-control framework for general optimization in the cloud.

3.4 A Meta-Control Framework for Cloud Computing

In the previous section, an implementation of the framework in Section 3.2 in the cloud was discussed. This section generalizes the implementation for solving optimization problems in the cloud using meta-control. The meta-control framework starts by generalizing the tasks to be solved in the cloud and implementing meta-control similar to the previous section.

First, some tasks arrive in the cloud that need to be fulfilled. The framework assumes that the tasks arrive with instructions on what needs to be done. The tasks can be specified hierarchically, for example, executing a subroutine, updating a database, solving a numerical problem. For example, in the scenario of demand forecasting, a specific task might be to forecast tomorrow's demand for a hundred different SKUs. This could be decomposed into a hundred separate forecasts or one processor could run all SKUs at once. Another example is to find a good set of parameters for a demand forecast model for a hundred different SKUs. This example would contain subtasks for initial parameter estimation, executing the forecaster and updating the parameters as new data arrives. Another example is to run three different forecasters for the same SKU and evaluate which forecaster is performing the best. The internal workings and design of each individual forecaster are unknown and may be treated as a black-box. Black-boxes can be modularized or hierarchical or combined in

many different ways.

The framework is modeled as an abstract state machine as shown in Figure 3.4. Some tasks arrive and the initial decision is how many processors to allocate to execute these tasks. The tasks may come with instructions on how to parallelize the individual components. If the tasks have been performed before, any metric associated with those tasks may be used in the initial allocation decision (e.g., average run time or storage space required). The tasks are then allocated and the processors begin performing their tasks. At some regular time intervals the processors report back their status. If a task has been completed then the solution is extracted, otherwise processors are examined and possibly restarted based on their current results.

Figure 3.4 provides a general framework as any task can be implemented with this design. For example, suppose the problem has three different forecasters (A, B, and C) which are run for the same SKU with the goal of determining which produces the most accurate forecast. In this example a processor is dedicated to each forecaster and each processor may report back its intermediate status. For example, a processor may report back that the first component of forecaster A is completed and is currently evaluating the second component. However a different processor might have reported back that its forecast algorithm B has already completed while the processor executing forecaster C is stalled. At this point a decision would be made whether to continue running the first forecaster A (at the expense of time and money), restart C or replace C with B in the hopes a better solution can be found.

With this design, a processor is not limited to strictly being a director or worker. That is, a processor may be a worker for a problem with many tasks but be its own director for a smaller sub-problem. In the example of demand forecasting, a task might arrive to find parameters for ten different SKUs. The meta-control may choose to initially allocate this problem by assigning one processor for each SKU. Each individual processor now has the task of finding parameters for one SKU, perhaps using the I-frame methodology presented in Chapter 5. These processors may then become their own director, as they assign the necessary computation to other processors.

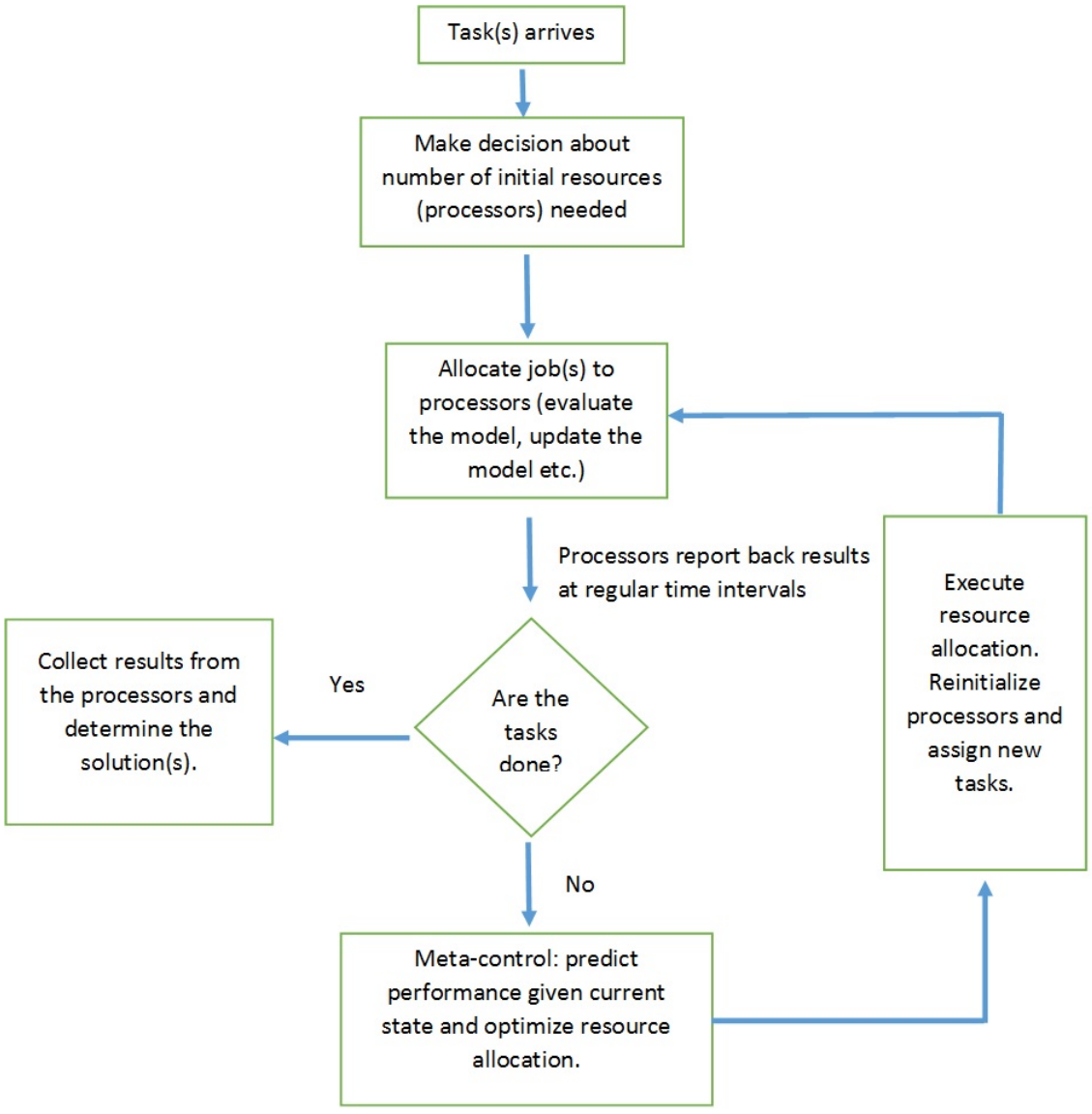

Figure 3.4: Framework for completing tasks in the cloud.

This framework allows one to utilize the resources in the cloud to their fullest extent. Many algorithms can be parallelized at once and each component of the algorithm can also be parallelized. The user is then focused on the results and does not have to worry about the intricacies of each algorithm or how the cloud is operating. In this way the user is able to make a better decision by using the cloud.

3.5 Summary

This chapter presented a generic framework for dynamic models. This framework is very practical as it accounts for a large amount of data, complex models and data arriving over time. The cloud is well-suited for implementing this framework due to its flexibility, access to on-demand resources and ability to store and quickly access a large amount of data. Using meta-control we can solve any optimization problem efficiently in the cloud. Chapter 4 presents a probabilistic forecaster which is apt for problems where data is arriving over time. Then, Chapter 5 presents an algorithm for approximating nonlinear least squares problems which is applicable for online problems.

Chapter 4

AN INCREMENTAL PROBABILITY FORECASTER

4.1 Introduction

We present an incremental probability forecaster that combines historical data and new data as it becomes available. The form of the probability distribution (e.g., Gaussian) does not need to be assumed. Our incremental model constructs probabilities relative to the nominal (or mean) of the state which is allowed to change over time. The user is able to choose the number of increments. Using increments provides a computationally efficient approximation to the large scale propagator problem.

The method is based on the Chapman-Kolmogorov equations for a continuous-time Markov process. This describes the evolution of the transition probability $P\big((x,t)|(x',t')\big)$ where x, x' are in the state space $\mathbb{R}^n$, x' is the state at time t' and similarly x is the state at time t with $t > t'$. One benefit of the Kolmogorov equations is that an underlying distribution of the state does not need to be assumed. However propagating the transition probabilities using just the Chapman-Kolmogorov equations is computationally intractable. To address this issue, we use the Fokker-Planck equation which is a linearized form of the Chapman-Kolmogorov equations.

However, the Fokker-Planck equation is a second-order, partial differential equation that may not always be easy to work with. To address this issue we discretize the Fokker-Planck equation with respect to state and time. This only serves as an approximation but allows the transition probabilities to be in a finite space in finite time as a Markov chain. The discrete time transition probabilities have parameters that explain how the system changes over time. These parameters need to be estimated as well as frequently updated when new data arrives.

Estimating these parameters may require a lot of computational time, especially if the state space is large. Our approach to create a computationally efficient algorithm is to formulate the Fokker-Planck equation relative to the mean of the state. This creates a finite set of transition probabilities which depend on the difference between the current state and the mean. The state space is then divided up into mutually exclusive intervals called the incremental levels. The model predicts the future incremental level instead of just calculating the expected value. The mean of the state is allowed to change over time as new data is obtained, and as a result the incremental levels may change as well. The dynamically changing levels allow the predictions to change over time in accordance with the new data. It is also possible to interpolate the resulting probabilities to approximate a distribution.

Section 4.2 presents an incremental probability model based around the mean of the state. This probability model is a Markov chain which is defined for the incremental state. In Section 4.3 we show the necessary conditions on the Markov chain to approximate the Fokker-Planck equation. Section 4.4 presents a parameter estimation problem which uses historical data to find the parameters for the incremental probability model. The parameter estimation problem, however, is not computationally feasible to use in practice. Thus, in Section 4.5 we approximate the incremental probability model borrowing ideas from mean field theory. This reduces the parameter estimation problem to a computationally feasible problem. The chapter concludes in Section 4.6.

4.2 An Incremental Probability Model Around the Mean

Our incremental model is based around the mean of the state instead of using the state directly. This may be an arithmetic mean, geometric mean or a weighted moving average mean. We assume that we already have the mean and that we can update it with each new data point that is obtained or observed. We let $\overline{x}_t$ be the mean of the state at time t and Δx_t be the difference between the current state and the mean of the state at time t, $\Delta x_t = x_t - \overline{x}_t$. Note that Δx_t is a vector with n components. The model is originally constructed from the

differential Chapman-Kolmogorov equation for the incremental state,

$$\frac{\partial P\big((\Delta x_t, t)|(\Delta x'_{t'}, t')\big)}{\partial t} = A(t)P\left((\Delta x_t, t)|(\Delta x'_{t'}, t')\right), \tag{4.1}$$

where $P\big((\Delta x_t, t)|(\Delta x'_{t'}, t')\big)$ is the probability that at time t the system will have the incremental state Δx_t having observed the incremental state $\Delta x'_{t'}$ at a previous time t'. We assume that $\overline{x}_t = \overline{x}_{t'}$ for small Δx_t. The operator $A(t)$ determines how the transition probabilities change with time. In general, $A(t)$ propagates any function $\rho(x)$ in the following manner:

$$A(t)\rho(x) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int dx' \bigg(P\big((x, t+\Delta t)|(x', t')\big) - P\big((x, t)|(x', t')\big) \bigg) \rho(x').$$

The Chapman-Kolmogorov equations in (4.1) generate a generic probability distribution. However these equations may be impractical to compute for near real-time applications. To address this issue, we use the Fokker-Planck equation which is a linearized form of the Chapman-Kolmogorov propagator (see [32]). The Fokker-Planck equation is

$$\begin{aligned}\frac{\partial P\big((\Delta x_t, t)|(\Delta x'_{t'}, t')\big)}{\partial t} =& \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2}{\partial x_i \partial x_j}\Big[a_{ij}(\Delta x_t, t) P\big((\Delta x_t, t)|(\Delta x'_{t'}, t')\big)\Big] \\ &- \sum_{k=1}^{n} \frac{\partial}{\partial x_k}\Big[b_k(\Delta x_t, t) P\big((\Delta x_t, t)|(\Delta x'_{t'}, t')\big)\Big]\end{aligned} \tag{4.2}$$

where $a_{ij}(\Delta x_t, t)$ is the (i, j) element of the symmetric, incremental diffusion matrix and $b_k(\Delta x_t, t)$ is the k^{th} element of the drift vector with a known initial distribution of the state at time 0. See Appendix A for the derivation of the Fokker-Planck equation.

Equation (4.2) is a useful approximation to the Chapman-Kolmogorov equations, however it is not always practical to use this equation as it is a second-order partial differential equation. To address this issue we discretize (4.2) using a finite difference approximation with respect to state and time to obtain our incremental probability model. This leads to a discrete-time, finite state Markov chain (see [31]).

We formulate the concept of incremental levels, which divides up the state space into three mutually exclusive levels (low, medium and high levels). Note it is possible to create more

levels at the cost of increased computation. Our model finds the probability of transitioning between the incremental levels, as opposed to calculating the transition probabilities for every state change. These incremental levels are constructed relative to the mean of the state. (Thus, it is helpful to think of the medium incremental level as being close to the mean, the low incremental level being far below the mean, and the high incremental level being far above the mean). See Figure 4.1.

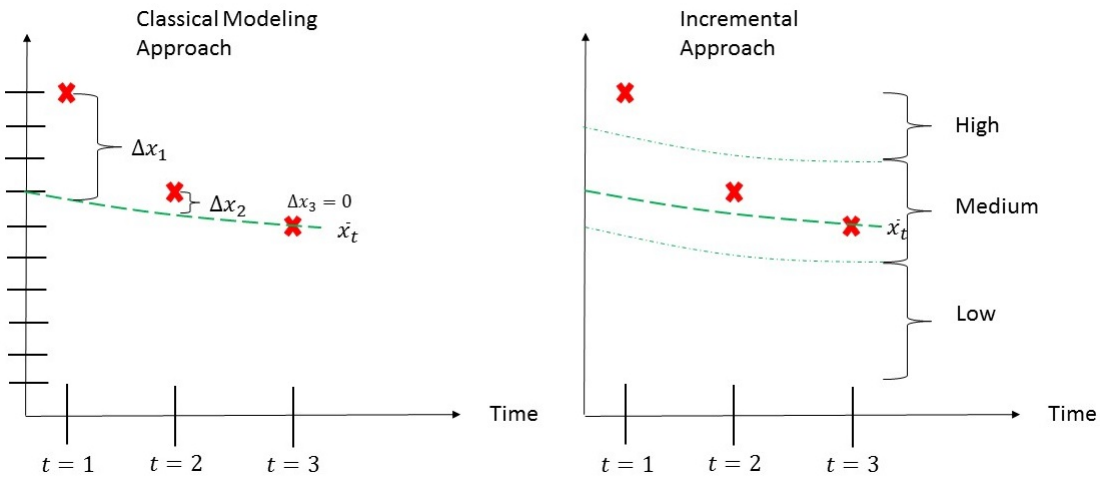

Figure 4.1: An example of incremental levels for a state with one component. The mean, $\overline{x}_t$, is tracked as data is observed. Note that Δx_t is the distance between the observation and the mean. The figure on the left represents a classical modeling approach where transition probabilities need to be calculated for every possible state change. On the right, the state space is divided into three mutually exclusive levels and the transition probabilities are only calculated for transitioning between incremental levels.

Using incremental levels is practical in applications that involve making a decision involving uncertainty around a mean value, such as price forecasting [13, 35] game theory [2, 33] and protein structure prediction [25]. A common approach is to propagate the mean and the covariance as a way of capturing the uncertainty [13, 60]. However doing so assumes that the distribution of the data is known, which may not reflect reality. Since the model is

derived by propagating the transition probabilities no assumption about the distribution of data is made.

We can construct the incremental levels using the maximum and minimum of the data. The high incremental level would be the range of values from the midpoint of the mean and the maximum value of the data up to the maximum value of the data. However in practice the incremental levels are constructed based on the application. In the stock market and demand forecasting it is usually known empirically what constitutes as the high incremental level (far above the mean). Note that the incremental levels can also dynamically adjust based on recent data. An example of this is illustrated in Figure 4.2.

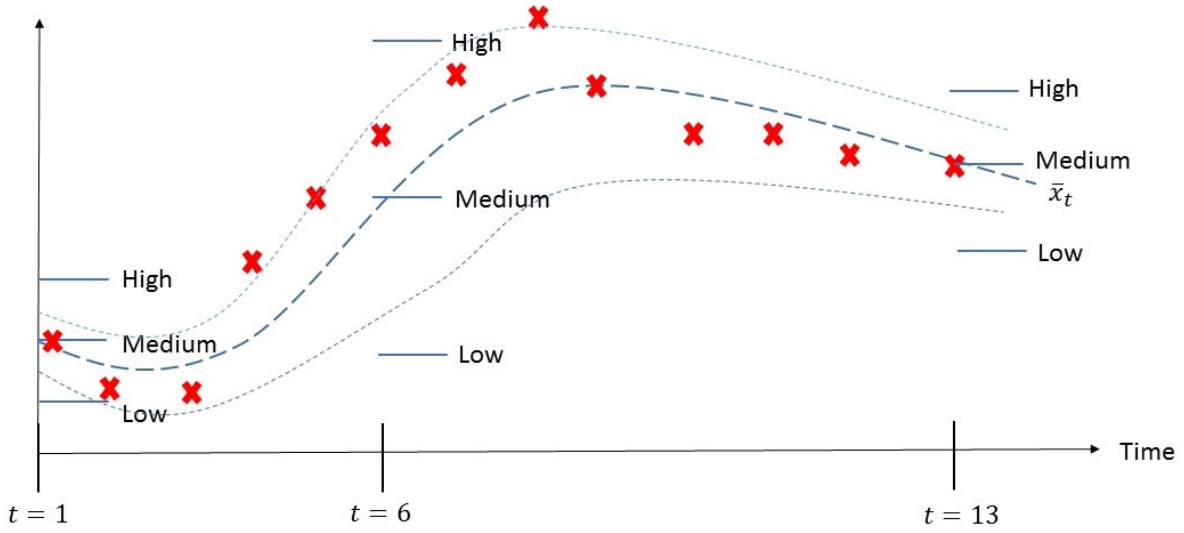

Figure 4.2: An example of dynamically adjusting the incremental levels. The incremental levels are initialized at time $t = 1$. At time $t = 6$ the levels are expanded to reflect the scattered data. At time $t = 13$ the incremental levels shrink to reflect the recent stability of the data.

The probability model is expressed as follows. Let $P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)$ be the probability of the incremental state $\Delta x_{t+\delta}$ occurring given that Δx_t was observed. The time step is represented by δ which is how often observations occur and the scalar h is the discretization

level of the state. For our model, the transition probabilities are only nonzero when either one or two components of the incremental state change. This is analogous to a stochastic process (such as a birth-death process) assuming that arrivals do not happen simultaneously. The model is given by the following equations:

$$P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t) = \begin{cases} 1 - \dfrac{\delta}{h^2}\left[2\sum_i a_{ii}(\Delta x_t, t) + Y\right] & \text{if no component changes, that is, } \Delta x_t = \Delta x_{t+\delta} \\ \dfrac{\delta}{h^2}\left[a_{ii}(\Delta x_t, t) + Z\right] & \text{if one component of the state changes} \\ \dfrac{\delta}{h^2}\left[a_{ij}^-(\Delta x_t, t)\right] & \text{if one component of the state increases and another decreases} \\ \dfrac{\delta}{h^2}\left[a_{ij}^+(\Delta x_t, t)\right] & \text{if two components of the state both increase or both decrease.} \\ 0 & \text{otherwise,} \end{cases} \tag{4.3}$$

where $a_{ij}(\Delta x_t, t)$ is the $n \times n$ diffusion matrix, $a_{ij}^-(\Delta x_t, t)$ and $a_{ij}^+(\Delta x_t, t)$ are defined as

$$a_{ij}^-(\Delta x_t, t) = \max\left[0, -a_{ij}(\Delta x_t, t), -a_{ji}(\Delta x_t, t)\right]$$
$$a_{ij}^+(\Delta x_t, t) = \max\left[0, a_{ij}(\Delta x_t, t), a_{ji}(\Delta x_t, t)\right]$$

and

$$Y = -\sum_{\substack{j \\ i \neq j}} |a_{ij}(\Delta x_t, t)| + h\sum_i |b_i(\Delta x_t, t)|$$

$$Z = -\sum_{\substack{j \\ i \neq j}} |a_{ij}(\Delta x_t, t)| + h b_i(\Delta x_t, t),$$

where $b_i(\Delta x_t, t)$ is the i^{th} entry of the $n \times 1$ drift vector. Note that the transition probabilities are only nonzero when one or two of the components change. In the simplified case where a_{ij} and b_i do not depend on t then the chain is homogenous. We assume that for all x and t,

$$a_{ii}(\Delta x_t, t) - \sum_{\substack{j \\ i \neq j}} |a_{ij}(\Delta x_t, t)| \geq 0, \qquad i = 1, \ldots, n$$

which ensures the transition probabilities are nonnegative.

Note that the probability model in (4.3) is defined for the incremental state and *not* for the specific discretization values of the state. For example if we had a state with two components then we could find the probability of going from incremental state {Low, Low} to the incremental state {Low, Medium} (using the second line of (4.3)). Also recall that these incremental levels are based around the mean of the state.

This probability model can be used to forecast the next incremental level. Since the incremental levels are constructed relative to the mean, this gives a forecast relative to the mean. However this only forecasts the incremental level and not the exact value of the state (or component of the state). Should we wish to forecast the exact value we can use a second order spline interpolation on the incremental levels to get an approximation [4, 50].

We can see that the probability model in (4.3) is directly dependent upon the diffusion matrix, a_{ij}, and the drift vector, b_k, in the Fokker-Planck equation in (4.2). Thus if the diffusion matrix and drift vector are known, we can construct the Markov chain in (4.3) which explains how the incremental state evolves over time and allows us to make predictions for the future. Similarly, if the dynamics of the Markov chain are known we can determine the values of the diffusion matrix and drift vector. Section 4.3 derives necessary conditions on the diffusion matrix and drift vector to show that the Markov chain in (4.3) is a good approximation to the Fokker-Planck equation. Then, Section 4.4 formulates an optimization problem to find the diffusion matrix and drift vector given historical data.

4.3 Necessary Conditions for the Incremental Probability Model

We now derive the necessary conditions to show that the Markov chain given in (4.3) is a good approximation to the Fokker-Planck equation. To do this, we use Euler's approximation for an ordinary differential equation for a twice differentiable function $f(x)$,

$$\frac{df(x)}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{4.4}$$

The second derivative can be approximated using this formula as well:

$$\begin{aligned} \frac{d^2 f(x)}{dx^2} &\approx \frac{\frac{f(x+2\Delta x)-f(x+\Delta x)}{\Delta x} - \frac{f(x+\Delta x)-f(x)}{\Delta x}}{\Delta x} \\ &= \frac{f(x+2\Delta x) - 2f(x+\Delta x) + f(x)}{(\Delta x)^2}. \end{aligned} \tag{4.5}$$

For any Markov chain where $P_k(t)$ is the probability we are in the k^{th} state at time t we can write the discrete space Chapman-Kolmogorov equation as

$$P_{k+1}(t) - P_k(t) = \sum_m A_{mk}(t)P_m(t) - A_{km}(t)P_k(t) \tag{4.6}$$

where $A_{mk}(t)$ is the transition rate from state m to state k at time t. Our Markov chain that we are considering is the incremental probabilistic forecaster represented in (4.3).

To use (4.6) for the incremental model, we need to impose three conditions on the Markov chain.

1. $\sum_k \sum_m A_{mk}(t)P_m(t) - A_{km}(t)P_k(t) = 0$ for any time t
2. $\sum_k P_k(t) = 1$ for any time t
3. When $P_k(t) = 0$ then $\left.\frac{dP_k(t)}{dt}\right|_{P_k(t)=0} = \sum_m A_{mk}(t)P_m(t) \geq 0$

The first condition says that at any time t, the rate entering all the states must equal the rate exiting all the states (rate in equals rate out). The second condition states that at any

time t the system must be in one of the states. The last condition says that if at a certain time t, the probability of residing in a state is zero, then it is only possible that things are leaving that state.

We have now defined the tools and restrictions to derive the necessary conditions for our incremental Markov chain. We first consider the scenario where our Markov process follows from a pure, symmetric diffusion master equation. This is a simpler version of the problem which will aid in forming the procedure necessary for the general transition diffusion master equation (the Fokker-Planck equation).

For the case of pure, symmetric diffusion the master equation is given by

$$\frac{dP_k(t)}{dt} = A(t)P_{k-1}(t) + A(t)P_{k+1}(t) - 2A(t)P_k(t).$$

This reads, the rate we enter state k (at time t) equals the rate we enter k from $k-1$ plus the rate we enter k from $k+1$ minus the rate we leave k to either $k-1$ or $k+1$. Here, $A(t)$ represents the transition rate.

Suppose that the process follows from a pure diffusion equation given by

$$\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} = a_{ij}(x,t)\frac{\partial^2 P\big((x,t)|(x',t')\big)}{\partial x^2}, \tag{4.7}$$

where $a_{ij}(x,t)$ is the symmetric diffusion matrix. Note that a_{ij} and $A(t)$ are closely related but not exactly the same. The a_{ij} matrix represents the diffusion rate for the propagator and $A(t)$ represents the transition rates for the master equation. We expand (4.7) using the Euler approximation in (4.4) on space (not time) to obtain

$$\frac{dP_k(t)}{dt} \approx (\Delta x)^2 \frac{A(t)P_{k-1}(t) + A(t)P_{k+1}(t) - 2A(t)P_k(t)}{(\Delta x)^2}.$$

Because we are looking for the parameters of the model, we must have $\lim_{\Delta x \to 0} A(t)\Delta x = a_{ij}(x,t)$ which is finite. Note that $A(t) \to \infty$ as $\Delta x \to 0$ for any fixed t. This is saying that as the discrete mesh gets smaller ($\Delta x \to 0$) then the transition rates $A(t)$ must be getting bigger. The condition that $A(t)\Delta x = a_{ij}(x,t)$ is finite means that $\frac{\partial P_k(t)}{\partial t}$ is a good approximation to the pure diffusion equation.

Now we are ready to explore the scenario of general transition diffusion where the transition rates are not symmetric. The process will be the same as the pure diffusion derivation. First we write the diffusion master equation, then we will approximate the Fokker-Planck equation with the Euler approximation. This will then lead to necessary conditions so that $\frac{\partial P_k(t)}{\partial t}$ is a good approximation to the Fokker-Planck equation.

The general transition diffusion master equation is given by

$$\frac{\partial P_k(t)}{\partial t} = A_{k-1,k}(t)P_{k-1}(t) + A_{k+1,k}(t)P_{k+1}(t) - \big(A_{k,k-1}(t) + A_{k,k+1}(t)\big)P_k(t),$$

where $A_{k-1,k}(t)$ is the transition rate from $k-1$ to k at time t.

Recall the Fokker-Planck equation given in (4.2),

$$\begin{aligned}\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} =& \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2}{\partial x_i \partial x_j}\Big[a_{ij}(x,t)P\big((x,t)|(x',t')\big)\Big] \\ &- \sum_{k=1}^{n}\frac{\partial}{\partial x_k}\Big[b_k(x,t)P\big((x,t)|(x',t')\big)\Big].\end{aligned}$$

Approximating the Fokker-Planck equation with the Euler approximation in (4.5) leads to

$$\frac{\partial P_k(t)}{\partial t} \approx \frac{a_{k-1}(t)P_{k-1}(t) - 2a_k(t)P_k(t) + a_{k+1}(t)P_{k+1}(t)}{(\Delta x)^2} + \frac{b_{k+1}(t)P_{k+1}(t) - b_{k-1}(t)P_{k-1}(t)}{2\Delta x}. \tag{4.8}$$

This gives the necessary conditions on the Markov chain:

1. $a_{ij}(\Delta x, t) = a_k(\Delta x, t) = \dfrac{(\Delta x)^2}{2}\left[A_{k,k-1}(t) + A_{k,k+1}(t)\right]$

2. $b_k(\Delta x, t) = \Delta x\left[A_{k,k-1}(t) - A_{k,k+1}(t)\right]$

To see this, substitute the two conditions into (4.8). Leaving out the dependency on t to save space, this gives

$$\begin{aligned}\frac{\partial P_k(t)}{\partial t} \approx& \frac{(\Delta x)^2}{2}\left[A_{k-1,k-2} + A_{k-1,k}\right]\frac{P_{k-1}}{(\Delta x)^2} - 2\frac{(\Delta x)^2}{2}\left[A_{k,k-1} + A_{k,k+1}\right]\frac{P_k}{(\Delta x)^2} \\ &+\frac{(\Delta x)^2}{2}\left[A_{k+1,k} + A_{k+1,k+2}\right]\frac{P_{k+1}}{(\Delta x)^2} + \Delta x\left[A_{k+1,k} - A_{k+1,k+2}\right]\frac{P_{k+1}}{2\Delta x} \\ &-\Delta x\left[A_{k-1,k-2} - A_{k-1,k}\right]\frac{P_{k-1}}{2\Delta x}.\end{aligned}$$

Cancelling terms leaves

$$\frac{\partial P_k(t)}{\partial t} = A_{k-1,k}(t)P_{k-1}(t) + A_{k+1,k}(t)P_{k+1}(t) - \big(A_{k,k-1}(t) + A_{k,k+1}(t)\big)P_k(t)$$

which is the general transition diffusion master equation.

This shows that as long as the conditions on a_{ij} and b_k are satisfied then $\frac{\partial P_k(t)}{\partial t}$ is a good approximation to the Fokker-Planck equation and that the choice of incremental discretization is not arbitrary. If a_{ij} and b_k (the diffusion matrix and drift vector) are known then we can find the transition rates A using historical data to represent P_k. Similarly, we can use historical data to estimate A and construct a parameter estimation problem to find a_{ij} and b_k. Section 4.4 discusses a parameter optimization problem in more detail. In Section 4.5 we examine how mean field theory is used to simplify the model to be computationally feasible.

4.4 A Parameter Estimation Problem

We develop the framework for estimating the parameters of the incremental forecaster in (4.3) using historical data. This involves formulating a maximum likelihood problem into a simpler optimization problem. In practice, this optimization problem may be too difficult to solve in full form, so simplifying assumptions may need need to be made. We formulate the optimization problem with regards to all the parameters in (4.3). Section 4.5 explains how the mean-field can be used to reduce the number of parameters to a computationally efficient level.

The parameters that need to be estimated are: $a_{ij}(\Delta x_t, t)$, $a_{ij}^{-}(\Delta x_t, t)$, $a_{ij}^{+}(\Delta x_t, t)$ and $b_i(\Delta x_t, t)$. The approach is to use historical data to estimate the dynamics of the Markov chain in (4.3). Using these dynamics an optimization problem is constructed which gives us values of the parameters (the diffusion matrix and drift vector). Recall that knowing the diffusion matrix and drift vector in (4.2) allows us to make predictions for the future.

We assume the incremental probabilistic model in (4.3) is continuously differentiable with respect to the parameters. Note that the parameters are dependent upon the incremental

state and the time t. We use a linear model to regress for these parameters:

$$a_{ij}(\Delta x_t, t) = a_{ij0} + a_{ij1}\Delta x_t + a_{ij2}t$$
$$b_i(\Delta x_t, t) = b_{i0} + b_{i1}\Delta x_t + b_{i1}t.$$

Since Δx_t is an $n \times 1$ vector, we define a_{ij1} and b_{i1} as $1 \times n$ vectors. Let Θ be the set of all parameters that need to be estimated.

To optimize the parameters for the probability equations, we construct a maximum likelihood optimization problem. Finding these parameters will allow us to make future predictions about the change in the state as well as analyze the state trajectory over time. The goal is to solve the following optimization problem,

$$\max_{\Theta} \log P\left(\Theta \middle| \left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\right)\right). \tag{4.9}$$

Using Bayes' theorem we can rewrite the probability in (4.9) as

$$P\left(\Theta \middle| \left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\right)\right) = \frac{P\left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\middle|\Theta\right) P(\Theta)}{P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)}. \tag{4.10}$$

$P(\Theta)$ is assumed to be a uniform distribution over the bounds of the parameters. $P_t^{h,\delta}$ is estimated using historical data so that this does not affect the optimization problem. The probability $P\left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\middle|\Theta\right)$ is given by the incremental forecaster, (4.3).

The optimization problem is the following,

$$\max_{\Theta} \sum_{\Delta x} \log P\left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\middle|\Theta\right) \tag{4.11}$$
$$s.t.\ \ 0 \le P_t^{h,\delta} \le 1 \text{ for all } t$$
$$\sum_{\Delta x} P_t^{h,\delta} = 1 \text{ for all } t.$$

The first condition ensures that all probabilities are between 0 and 1 inclusive, and the second condition ensures that at any time t the system is in one of the states. Solving this problem will give the entire set of parameters Θ (composed of $a_{i,j}$, $a_{i,j}^-$, $a_{i,j}^+$ and b_i in (4.3)) so that forecasts can be made.

The optimization problem in (4.11) can be reformulated into an unconstrained optimization problem using Lagrange multipliers. The problem becomes

$$\max_{\Theta} \sum_{\Delta x} \log P\left(P_t^{h,\delta}(\Delta x_{t+\delta}|\Delta x_t)\Big|\Theta\right) - \lambda\left(1 - \sum_{\Delta x} P_t^{h,\delta}\right)^2 - \mu(1 - P_t^{h,\delta}), \tag{4.12}$$

where λ and μ are Lagrange multipliers and thus are additional parameters to optimize. Note that there are multiple Lagrange multipliers in the problem, one corresponding to each state, as the sum of the transition probabilities for each state needs to sum to one. In practice solving (4.12) can be very time consuming, so additional assumptions need to be made to simplify the problem. For example, the sum constraints can be removed from the optimization problem. After the parameters have been calculated, the probabilities can be normalized to sum to one when performing forecasts.

4.5 An Approximation Model using Mean Field

The parameters for our model that need to be estimated are $a_{ij}(\Delta x_t, t)$, $a_{ij}^{-}(\Delta x_t, t)$, $a_{ij}^{+}(\Delta x_t, t)$ and $b_i(\Delta x_t, t)$. Since our model is designed around three increments for each component of the state there are a total of 3^n states. Note that all the parameters are dependent on Δx_t, the current value of the incremental state. Thus the number of parameters is $O(3^n)$. It may take a considerable amount of computation time to calculate the parameters. We address this concern by using a mean-field approach to decouple the parameters, reducing the number that need to be computed. The mean-field will approximate the coupling (analogous to a covariance matrix) between the parameters.

We first make a simplifying assumption that a component of the state can only move up or down *one* incremental level at a time. Thus a component cannot jump directly from a low level to a high level (and vice versa). Instead, a component would jump from low to medium and then immediately (in the next time step) jump from medium to high. This assumption is made as it decreases the number of possible transitions.

To illustrate how the mean-field is used we consider a simple example where the state has two components, $n = 2$. The transitions are illustrated in Figure (4.3). Note that in

this diagram we make a further simplification that only one component is allowed to change with each time step. This simplification will be discussed in further detail below.

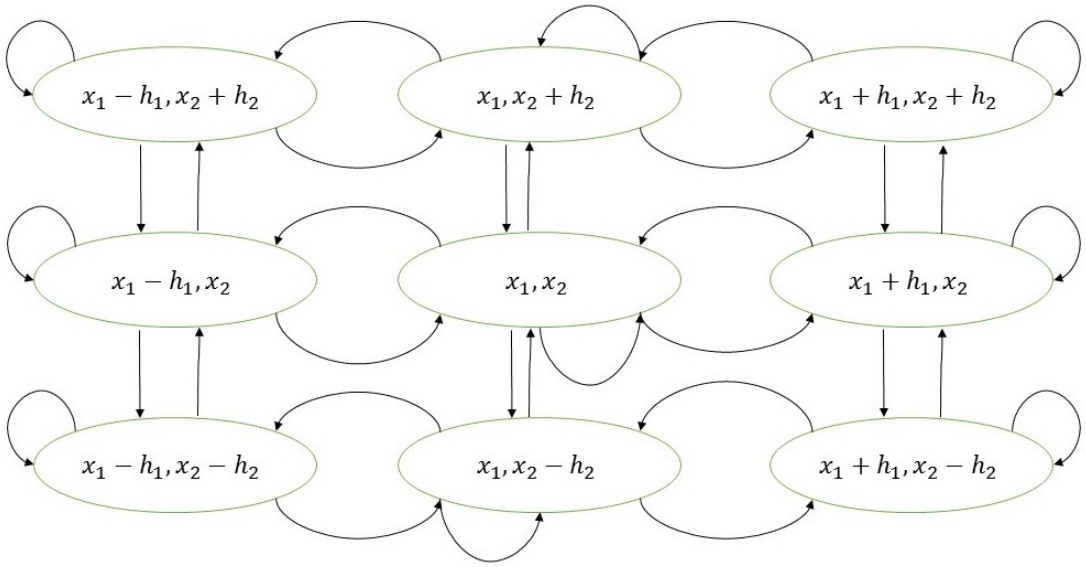

Figure 4.3: The Markov chain diagram for a state with two components, x_1 and x_2. The state is only allowed to move up or down *one* incremental level at a time. The time step is small enough that only one component changes between each time step.

Note that h_1 and h_2 are given the subscripts 1 and 2 to indicate which component they are changing. Since each component is divided into the same number of incremental levels in the same way, h_1 and h_2 are both the same from a computational standpoint. In implementation, h_i would be set to 1 for all $i \in 1, \ldots, n$ and we differentiate between moving up or down in incremental level by adding and subtracting h_i respectively.

First we will write out the transition probabilities into matrix form. We will write the matrix in a certain way so it can be broken down into blocks. We will then illustrate which blocks will be considered by the incremental model and which will be approximated with the mean field.

We write the transition probabilities in matrix form as

$$P_{t+\Delta} = AP_t \tag{4.13}$$

where P_t is a $3^n \times 1$ vector representing each of the incremental states and A is a $3^n \times 3^n$ transition probability matrix (the rows of A must sum to one). For example if the state only has one component, $n = 1$, the matrix equation in (4.13) is

$$\begin{bmatrix} x_1^{t+\Delta} & & \\ x_1^{t+\Delta} & + & h_1 \\ x_1^{t+\Delta} & - & h_1 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1^t & & \\ x_1^t & + & h_1 \\ x_1^t & - & h_1 \end{bmatrix}.$$

The first equation of this matrix would be written as

$$P(x_1^{t+\Delta}) = A_{11}P(x_1^t) + A_{12}P(x_1^t + h_1) + A_{13}P(x_1^t - h_1) \tag{4.14}$$

which we interpret as the probability we are in state x_1 at time $t+\Delta$ is equal to the probability we were in state x_1 at time t and did not change incremental level, plus the probability we were in state $x_1 + h_1$ at time t and decreased one incremental level, plus the probability we were in state $x_1 - h_1$ at time t and increased one incremental level. In general, we can define the i^{th} row of (4.13) by

$$P_{t+\Delta}^i = \sum_{j=1}^{m} A_{ij} P_t \tag{4.15}$$

where $m = 3^n$.

We write (4.13) for the situation where the state has two components in block form,

$$\begin{bmatrix} P_{t+\Delta}^1 \\ P_{t+\Delta}^2 \\ P_{t+\Delta}^3 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix} \begin{bmatrix} P_t^1 \\ P_t^2 \\ P_t^3 \end{bmatrix}$$

where P^1 contains all the transitions where x_1 is fixed, P^2 contains all the transitions where x_1 increases by h_1 and P^3 contains all the transitions where x_1 decreases by h_1. Each $\mathbf{A}_{ij}$ is a 3×3 matrix. We then have three equations of the following form:

$$P_{t+\Delta}^1 = \mathbf{A}_{11}P_t^1 + \mathbf{A}_{12}P_t^2 + \mathbf{A}_{13}P_t^3. \tag{4.16}$$

We approximate (4.16) using

$$P^1_{t+\Delta} = \mathbf{A}_{11} P^1_t + K_1. \tag{4.17}$$

where K_1 is a coupling term which approximates the other transition probabilities (in this case, $\mathbf{A}_{12} P^2_t + \mathbf{A}_{13} P^3_t$). There will be two other equations like this for the other two rows of the block matrix in (4.16). This K_1 will be approximated using the mean-field. Note that $\mathbf{A}_{11}$ contains all the transition probabilities where x_1 is fixed. Thus if we expanded out the first row in (4.17) we would get (remembering that $\mathbf{A}_{11}$ is a 3×3 matrix),

$$P(x_1, x_2)^{t+\Delta} = a_{11_{11}} P(x_1, x_2)^t + a_{11_{12}} P(x_1, x_2 + h_2)^t + a_{11_{13}} P(x_1, x_2 - h_2)^t + K_1$$

which is similar to (4.14) with the added coupling K_1.

By doing this, the A matrix has simplified to a tridiagonal matrix (two of these entries in each 3×3 block are also zero given the assumption that an incremental state cannot jump from low to high and vice versa). This leads to 21 entries in the A matrix that need to be determined (as opposed to the full matrix which contains 81). However the entries we have found only represent the second component x_2 changing. Thus these 21 entries would give a good approximation to the behavior of x_2 but not say much about x_1. This process would need to be repeated for x_1 (switching the two components in the above calculations). This has reduced the number of parameters considerably but it is still exponential in n, $O(n3^{n-1})$.

We further simplify this model by treating each component of the state as independent from the others in our parameter calculations (we can allow this with the assumption that our time step δ is small enough that only one component of the state is allowed to change within each time step). For example, this means that the probability of component x_1 increasing from the medium level to the high level is the same no matter what state the other components, $x_2, x_3, \ldots, x_n$, are in. This reduces the number of parameters to being linear in n.

We now return to our row matrix form in (4.15) to examine how to formulate the coupling K_i. As described above, instead of using (4.15) directly we use (4.17) for each component i,

$$\widetilde{P}^i_{t+\Delta} = \mathbf{A}_{ii} \widetilde{P}^i_t + K_i. \tag{4.18}$$

We use $\widetilde{P}_t$ to denote the mean-field approximation. We define K_i as a random variable,

$$K_i \sim N(\hat{k}_i, \sigma_i) \tag{4.19}$$

where

$$\hat{k}_i = \sum_{\substack{j=1 \\ j \neq i}}^{m} A_{jj} \widetilde{P}_t^j$$

and σ_i is an estimate of the square of the other transition rates $\mathbf{A}_{ij}$ for $j \neq i$. The mean of K_i is found by summing all the mean-field estimates of the transitions of the other components multiplied by their estimated transition rate. The variance, σ_i, is initially chosen to be large. As the mean-field is updated and new data arrives, the variance can be adjusted (decreased) when a better estimate of the interactions is known. In many applications an initial σ_i will be known empirically. Since each component i contains three rows of the matrix formulation in (4.18) the random variable K_i can be sampled three times, or the same sample can be used for each of the three rows.

Note that a correction term may need to be added to K_i to ensure that the probabilities are between 0 and 1. If K_i is sampled and is too small ($K_i << 0$) then the probability $\widetilde{P}^i_{t+\Delta}$ is corrected to 0. This means the mean-field estimates the transition rate to be zero at the current time. If K_i is sampled and is too large the probability is corrected to 1.

We now look at the impact this has on the incremental model in (4.3). Since the transition probabilities in (4.17) only contain instances where at most one component changes, we do need to find the parameters for $a^-_{ij}(\Delta x_t, t)$ and $a^+_{ij}(\Delta x_t, t)$ (as the interactions from these parameters are approximated by the mean field). For each component of the state, $a_{ij}(\Delta x_t, t)$ and $b_i(\Delta x_t, t)$ need to be found but these parameters are *only* dependent upon that component. The number of parameters that need to be found is now $10n$.

The implementation process is as follows. Use historical data to find each $\mathbf{A}_{ii}$ in (4.13) which is equivalent to solving the parameter estimation problem for the $10n$ parameters in the incremental model. This means treating each component of the state as independent and estimating the transition rates for changing incremental levels, finding $a_{ij}(\Delta x_t, t)$ and

$b_i(\Delta x_t, t)$ to satisfy the Markov chain given in (4.3). Start with a best guess of P_t which can be obtained analytically but in most applications a good estimate will be known empirically. Randomly pick a component to update using the mean-field. Calculate $\widetilde{P}^i_{t+\Delta}$ by sampling K_i and correcting if needed. Randomly pick a new component to update and repeat the process. Note that the mean when sampling K_i changes at each iteration as one component of $\widetilde{P}_t$ will have changed.

4.6 Summary

A probabilistic forecaster is constructed based around the idea of incremental levels which is practical for many applications. This forecaster is represented by a Markov chain which has parameters that need to be estimated. In order to estimate these parameters efficiently, simplifying assumptions are made and the mean-field is used. This allows for forecasts to be made and for the state of the overall system to be analyzed.

The incremental forecaster fits into the general framework presented in Chapter 3. The mean and the incremental levels can quickly update as new data arrives. The mean field coupling and the parameter estimation only need to be performed when the model is doing poorly. The incremental forecaster would benefit from a cloud implementation as data can be easily accessed and stored, and different parts of the computation can be distributed (the spline interpolation, the mean field coupling and the parameter estimation).

Chapter 5

AN I-FRAME METHODOLOGY FOR APPROXIMATING NONLINEAR LEAST SQUARES

5.1 Introduction

We present a new methodology to efficiently solve the nonlinear least squares problem defined in Section 2.4 that addresses the sequential nature of the data without computing a Hessian matrix. This methodology is designed to accommodate large nonlinear least squares problems in the sense that there may be a large number of parameters, a large amount of data or both.

The methodology for approximating a solution to the nonlinear least squares problem is inspired by a technique in video compression and animation called I-frames or key-frames [61]. This approach reduces computation time by only doing expensive computation occasionally, at I-frame intervals (called the I-frame optimization). In between the I-frames a simple, continuous procedure called the incremental optimization is performed. The incremental optimization defines how the parameters change in between the I-frames.

The methodology is constructed with the intent that the user has to solve many nonlinear least squares problems in a short amount of time. For example, an online retailer may need to solve a nonlinear least squares problem in order to make a prediction, or forecast, about the demand for an item. However, transactions may be occurring on the order of microseconds, twenty-four hours a day, every day of the year. In this example, the user cannot afford to spend hours calculating a forecast because many transactions may have occurred in the time between the start and finish of the forecast calculation. Instead the user may benefit more from a timely, less accurate forecast calculation since waiting a long time to get a very accurate result is not beneficial. Existing methods do not apply well for this problem as they lack any sort of control over the accuracy of the solution and the computation required.

In Section 5.2 an I-frame methodology with a modified fast marching algorithm is presented for efficiently solving nonlinear least squares problems. Section 5.3 includes a proof relating the criterion for generating I-frames to the average squared error of the final solution. Then, Section 5.4 provides some numerical results and the chapter concludes in Section 5.5.

5.2 A Fast Marching Algorithm with I-frames

Recall from Section 2.4 the offline nonlinear least squares problem:

$$\min_{\mathcal{P}} G(\mathcal{P}) \tag{5.1}$$
$$\text{where } G(\mathcal{P}) = \frac{1}{2}\sum_{i=1}^{m}\Big(r_i(\mathcal{P})\Big)^2 = \frac{1}{2}r(\mathcal{P})^T r(\mathcal{P}),$$

where $\mathcal{P}$ is the vector of parameters that need to be estimated. Note that zero is a lower bound on $G(\mathcal{P})$ because $G(\mathcal{P})$ is a sum of squares.

Solving Problem (5.1) directly can prove troublesome for large m and n. One often has to compute and store a large Jacobian and Hessian matrix (and continually update these as the algorithm progresses). Many algorithms have been developed that assume the Jacobian matrix is sparse. A Jacobian matrix for Problem (5.1) is sparse if not all of the summands (the r_i's) depend on all of the n parameters. We develop an algorithm to solve Problem (5.1) that does not need a sparse Jacobian or a Hessian matrix. To do this, we solve a continualized version of Problem (5.1) based on Newton's descent method.

To continualize Problem (5.1), we modify a fast marching algorithm, based on the work in [48] and [53]. The algorithm uses a modified Newton's descent method, which solves the optimization problem (5.1) based on the following sequence, for iteration k, $k = 0, 1, \ldots,$

$$\mathcal{P}_{k+1} = \mathcal{P}_k - \alpha\left(\frac{\partial^2 G(\mathcal{P}_k)}{\partial \mathcal{P}^2}\right)^{-1}\frac{\partial G(\mathcal{P})}{\partial \mathcal{P}} \tag{5.2}$$

given an initial starting point $\mathcal{P}_0$ and some α with $0 < \alpha \leq 1$. The iteration step in (5.2) is continualized by replacing the iterate $\mathcal{P}_k$ with the parameter function $\widetilde{\mathcal{P}}(t)$ to obtain

$$\frac{d\widetilde{\mathcal{P}}(t)}{dt} = -\alpha\left(\frac{\nabla G\left(\widetilde{\mathcal{P}}(t)\right)}{||\nabla G\left(\widetilde{\mathcal{P}}(t)\right)||}\right) \tag{5.3}$$

where t is a continuous time associated with the algorithm and $\nabla G(\mathcal{P}) = \frac{\partial G(\mathcal{P})}{\partial \mathcal{P}}$ and

$$||\nabla G\left(\widetilde{\mathcal{P}}(t)\right)|| = \max_{i=1,\ldots,n} \left(\frac{\partial G\left(\widetilde{\mathcal{P}}(t)\right)}{\partial \widetilde{\mathcal{P}}_i(t)} \right).$$

The continualization in (5.3) is appealing because there is no need to calculate the inverse of a Hessian matrix and sparsity does not need to be assumed. However, the integration can be time consuming to calculate as the number of parameters or the amount of data increases. Furthermore, the integration of (5.3) can also be time consuming if the model function, $f(\mathbf{x}, \mathcal{P})$, is costly to evaluate. To address these issues, we construct a methodology that eases the required computation by only doing costly function evaluations when the model function with the current parameters is not reflecting the data well.

Our modified fast marching algorithm integrates (5.3) over the interval $[0, T]$ for some T. Figure 5.1 illustrates a solution to (5.3). The solution $\widetilde{\mathcal{P}}(T)$ corresponds to an optimal solution to (5.1). Note that T is an algorithmic time used for integration and is independent of the iterative time associated with the data. Once the integration method is finished, our modified fast marching algorithm checks whether the solution evaluated at the objective function, $G(\widetilde{\mathcal{P}}(T))$, is less than a threshold (0.001 in our numerical results in Section 5.4). If the solution is less than the threshold, the modified fast marching algorithm stops and returns $\widetilde{\mathcal{P}}(T)$. If it is not, the modified fast marching algorithm recomputes the gradient at this new point and starts the integration method with $\widetilde{\mathcal{P}}(T)$ as the starting point. This continues until a maximum iteration count is reached (40 in our numerical results).

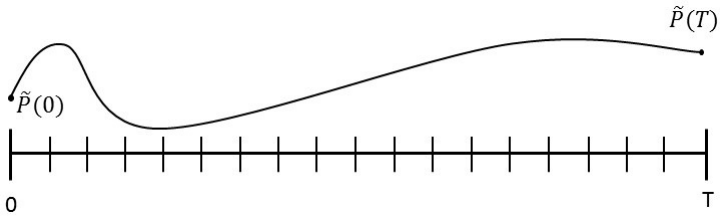

Figure 5.1: Evolution of $\widetilde{\mathcal{P}}(t)$ using numerical integration.

A question remains of how often to perform the numerical integration and the fast marching algorithm. The result $\widetilde{P}(T)$ is an approximate solution to (5.1) using all m data points. To reduce computation, instead of using all m data points to solve the nonlinear least squares problem, a subset of point is constructed using only some of the m data points. The approach is to create I-frames for subsets of the data points with small changes in the parameters. This restricts the large computation to the start of the I-frames and continualizes and linearizes the "flow information" between I-frames. The algorithm starts with the first data point and iterates through the data constructing I-frames at various points. These points are added to the subset and the I-frame optimization solves the nonlinear least squares problem over the expanding subset of data points.

The I-frame optimization problem is

$$\min_{P} G_I(P) \tag{5.4}$$
$$\text{where } G_I(P) = \frac{1}{2}\sum_{i\in I}\Big(r_i(P)\Big)^2$$

and I is the set of indices denoting the start of a new I-frame. The fast marching algorithm is used to solve (5.4) by solving (5.3), and the solution is denoted $P^C(t_k)$, where the I-frame starting at time t_k is denoted as the k^{th} I-frame. The gradient can be found analytically or estimated numerically. Note that initial values, or ranges, for the parameters P are needed to start the fast marching algorithm.

Having solved for $P^C(t_k)$ in the k^{th} I-frame, the approach next evaluates the residual at each data point $(\mathbf{x}_i, y_i)$ sequentially for $i > t_k$, $r_i\left(P^C(t_k)\right) = y_i - f\left(\mathbf{x}_i, P^C(t_k)\right)$. If $r_i\left(P^C(t_k)\right)^2$ is greater than a certain threshold Δ, that is,

$$r_i\left(P^C(t_k)\right)^2 > \Delta \tag{5.5}$$

then a new I-frame is created, (i.e., $t_{k+1} = i$), and the time i is added to I. If the function evaluation f is costly, the user may choose to evaluate it only when the data point y_i changes by a user defined value.

As an example, suppose there are 13 data points ($m = 13$) and I-frames are generated at times 1, 4, 10 and 13 (see Fig. 5.2). Thus $I = \{1, 4, 10, 13\}$. We denote the time of the k^{th} I-frame as $t_k = j$, referring to the $(\mathbf{x}_j, y_j)$ datapoint.

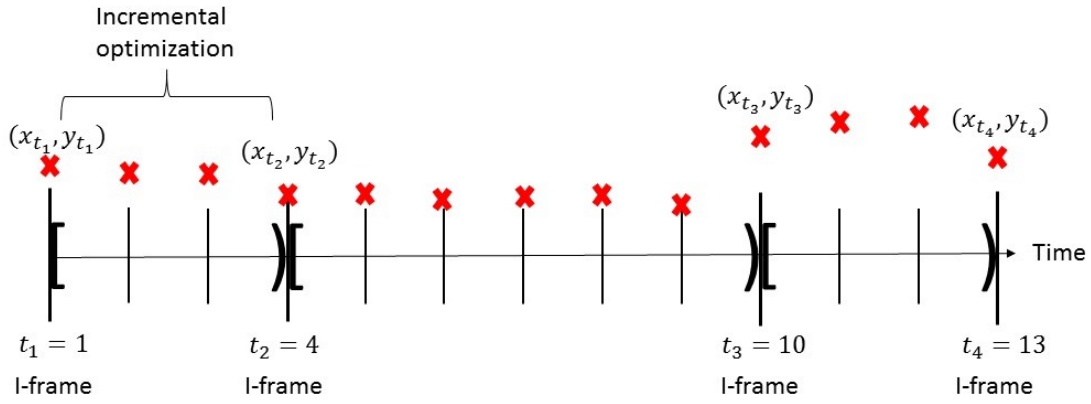

Figure 5.2: Illustration of I-frame intervals. The data points are represented by the dark x's. The incremental optimization occurs over the I-frame right up to the next I-frame. In this example, the first I-frame occurs at time $t_1 = 1$, referring to the point $(\mathbf{x}_1, y_1)$, and the second I-frame occurs at time $t_2 = 4$, referring to the point $(\mathbf{x}_4, y_4)$.

The main idea for this approach is that a set of parameters will be sufficiently good as long as the model function with the current set of parameters closely matches the data. Then, if there is a big change in the data, the parameters are modified to incorporate this change using an I-frame. This is analogous to the use of I-frames in animation as I-frames represent a fully specified picture while in-between the I-frames only the changes between frames are recorded. Also note that if an I-frame is created at every data point, then the index set I will include every data point at the last I-frame. This will result in solving the original problem (5.1).

Note that the I-frame generation criterion in (5.5) creates I-frames if the residual at the *current* data point is too large. This is not the only option one has for creating I-frames. For example, we can modify the I-frame generation criterion to create a new I-frame if the

average sum of squares over the entire data set up to the current point exceeds some Δ. This would create a new I-frame at time M if

$$\frac{1}{M}\sum_{i=1}^{M} r_i \left(P^C(t_k)\right)^2 > \Delta.$$

This criterion would give a more global perspective on the model function: I-frames are created if the model function does not accurately reflect the data over the entire data set up to the current point instead of just the current point. For some problems this may be a more desirable measurement. The rest of this dissertation uses the I-frame generation criterion in (5.5).

Recall that to run the fast marching algorithm an initial value of the parameters is needed. For the k^{th} I-frame, it is possible to use the previous set of parameters found at the $k-1$ I-frame, $P^C(t_{k-1})$, as the initial value. However, we can potentially save some computation in the fast marching algorithm (either in the overall iteration count or in the numerical integration which solves (5.3)) by constructing an incremental optimization problem. The incremental optimization adjusts the previously found parameters, $P^C(t_{k-1})$, over the $k-1$ I-frame to provide an initial value for the k^{th} I-frame.

To derive the incremental optimization, first the residuals are continualized and approximated. Then, the nonlinear least squares problem in (5.1) is rewritten as a partial summing formation, which is then continualized and linearized around the time t. This allows the continualized approximation to the residuals to be used. Next, a linear control bang-bang problem is derived which defines how the parameters change in-between I-frames. Finally, the Hamiltonian and necessary conditions for optimally are constructed to provide the bang-bang solution.

Denote the I-frame starting at time t_k as the k^{th} I-frame and use t as the "time" of an I-frame, $t \geq 0$. The data point at the beginning of the k^{th} I-frame is $(\mathbf{x}_{t_k}, y_{t_k})$. See Fig. 5.2. A continualized form of r_i on the k^{th} I-frame interval ($t_k = i$), with a linear perturbation of the model parameters, is defined based on Euler's integration algorithm,

$$\tilde{r}_{t_k,t}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t)\right) = y_{t_k} - f\left(\mathbf{x}_{t_k}, \widetilde{P}(t)\right) \tag{5.6}$$

where $\widetilde{P}(t) = P^C(t_k) + \delta\widetilde{P}(t)$ for $t > t_k$ and $\delta\widetilde{P}(t)$ is the parameter perturbation vector.

The start of the next I-frame is determined by the residual (error) between the model function, $f(\mathbf{x}_i, P)$ and the actual value, y_i. Because the residual between the model function and the actual value is less than the threshold Δ for $i > t_k$, the variation in data is small within an I-frame. This justifies approximating $\tilde{r}^2_{t_k,t}$ with a first order Taylor series expansion around the values of the data at the I-frame, denoted $\hat{r}$. This leads to

$$\hat{r}^2_{t_k,t}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t)\right) = \tilde{r}^2_{t_k,t_k}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t_k)\right) + \left(\frac{\partial \tilde{r}^2_{t_k,t}}{\partial P}\bigg|_{t=t_k}\right)^T \delta\widetilde{P}(t). \tag{5.7}$$

For ease of notation, let

$$Y_{t_k} = \tilde{r}^2_{t_k,t_k}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), 0\right) \tag{5.8}$$

and

$$Z_{t_k} = \left(\frac{\partial \tilde{r}^2_{t_k,t}}{\partial P}\bigg|_{t=t_k}\right)^T. \tag{5.9}$$

The partial derivatives can be calculated using (5.6) since f is assumed to be differentiable with respect to P. Note that $Y_{t_k} = \tilde{r}^2_{t_k,t_k}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), 0\right) = r_{t_k}(P^C(t_k))^2$ because $\delta P(t_k)$ is equal to zero (since t_k corresponds to the start of an I-frame). Equation (5.7) is now written as,

$$\hat{r}^2_{t_k,t}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t)\right) = Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t). \tag{5.10}$$

This approximation is *linear* with respect to $\delta\widetilde{P}(t)$.

In order to make use of the approximation $\hat{r}^2_{t_k,t}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t)\right)$ in (5.7), the least squares problem in (5.1) is rewritten as a partial summing formulation. The optimization problem in (5.1) is written in the following recursive form,

$$\begin{aligned} &\min_P \quad S_m \\ &\text{s.t. } S_{i+1} = S_i + r^2_{i+1}(P) \quad \text{for } i = 0, \ldots, m-1 \end{aligned} \tag{5.11}$$

with the initial condition $S_0 = 0$. The value S_i is the accumulated residual up to the i^{th} data point.

We next continualize problem (5.11) to obtain a continuous time approximation on the current k^{th} I-frame [27]. A continualized form of S_i with a linear perturbation of the accumulated residual is defined as

$$\tilde{S}_{t_k}(t) = \tilde{S}_{t_k}(t_k) + \delta\tilde{S}_{t_k}(t), \tag{5.12}$$

where $\tilde{S}_{t_k}(t_k) = S_{t_k}$, and $\delta\tilde{S}_{t_k}(t_k) = 0$. Taking derivatives of both sides of (5.12) yields

$$\dot{\tilde{S}}_{t_k}(t) = \frac{d\tilde{S}_{t_k}(t)}{dt} = 0 + \delta\dot{\tilde{S}}_{t_k}(t). \tag{5.13}$$

Now define,

$$\delta\dot{\tilde{S}}_{t_k}(t) = \beta\hat{r}^2_{t_k,t}\left(y_{t_k}, \mathbf{x}_{t_k}, P^C(t_k), \delta\widetilde{P}(t)\right) \tag{5.14}$$

for $t \geq t_k$. The descent parameter $\beta > 0$ in (5.14) normally needs to be estimated using empirical or learning methods, however this is not necessary as it cancels out later.

From (5.10), (5.13) and (5.14) a linear differential equation with constant coefficients is,

$$\delta\dot{\tilde{S}}_{t_k}(t) = \beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t)\right).$$

Now a linear control problem is formulated to find $\delta\widetilde{P}(t)$ in between the I-frames. Recall that $\delta\widetilde{P}(t)$ approximates how the parameters change between the I-frames. The goal is to maximize the time between I-frames (which indicates a good set of parameters) while minimizing the error at the end of the I-frame, $\delta\tilde{S}_{t_k}(T)$, where T denotes the time at the end of the I-frame. Note that $\delta\tilde{S}_{t_k}(T)$ is always greater than or equal to zero because it is approximating the squared error r_i^2. The linear control problem is

$$\begin{gathered}\max_{\delta\widetilde{P}(t)} \int_0^T 1ds - q\delta\tilde{S}_{t_k}(T) \\ \text{subject to } \delta\dot{\tilde{S}}_{t_k}(t) = \beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t)\right) \\ \delta\widetilde{P}_{min} \leq \delta\widetilde{P} \leq \delta\widetilde{P}_{max}\end{gathered} \tag{5.15}$$

where $\delta\tilde{S}_{t_k}$ is the state vector, $\delta\widetilde{P}(t)$ is the control vector and $q > 0$ is a weighting parameter. The bounds $\delta\widetilde{P}_{min}$ and $\delta\widetilde{P}_{max}$ can be updated as the algorithm progresses.

Problem (5.15) is known as a bang-bang problem. To see this, the Hamiltonian is written and the necessary conditions of optimality are constructed. This will give the values for how much the parameters change over the I-frame. The Hamiltonian is

$$H\left(\delta\tilde{S}_{t_k}(t), \delta\widetilde{P}, p(t)\right) = -1 + p(t)^T\beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t)\right),$$

where $p(t)$ is the costate vector. The necessary conditions for optimality are,

$$\delta\dot{\tilde{S}}_{t_k}(t) = \frac{\partial H}{\partial p} = \beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t)\right)$$

$$\dot{p}(t) = -\frac{\partial H}{\partial \delta\tilde{S}_{t_k}(t)}$$

$$p(T) = \frac{\partial\left(-q\delta\tilde{S}_{t_k}(T)\right)}{\partial\delta\tilde{S}_{t_k}} = -q$$

$$H^*\left(\delta\tilde{S}^*_{t_k}, \delta\widetilde{P}^*(t), p(t)\right) \geq H\left(\delta\tilde{S}^*_{t_k}, \delta\widetilde{P}(t), p(t)\right).$$

Because H does not depend on $\delta\tilde{S}_{t_k}(t)$ and $\dot{p}(t) = 0$ we know $p(t)$ is a constant. The third condition shows that $p(t) = -q$. The last condition implies that

$$-1 + p(t)\beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}^*(t)\right) \geq -1 + p(t)\beta\left(Y_{t_k} + Z_{t_k}\delta\widetilde{P}(t)\right). \tag{5.16}$$

Canceling terms and flipping the inequality since $p(t) < 0$ yields the following bang-bang condition,

$$Z_{t_k}\delta\widetilde{P}^*(t) \leq Z_{t_k}\delta\widetilde{P}(t).$$

If Z_{t_k} is greater than zero then the respective variables $\delta\widetilde{P}$ take on their minimum values, and if Z_{t_k} is less than zero then the respective variables take on their maximum values, yielding the bang-bang solution using (5.9),

$$\delta\widetilde{P}^*(t) = \begin{cases} \delta\widetilde{P}_{min} \text{ if } \left.\frac{\partial\tilde{r}^2_{t_k,t}}{\partial P}\right|_{t=t_k} > 0 \\ \delta\widetilde{P}_{max} \text{ if } \left.\frac{\partial\tilde{r}^2_{t_k,t}}{\partial P}\right|_{t=t_k} < 0 \end{cases} \tag{5.17}$$

Once this solution is calculated, the parameters are adjusted by setting $\widetilde{P}(t) = P^C(t_k) + \delta\widetilde{P}^*(t)$.

The algorithm is stated as follows for a set of m data points:

The I-frame Methodology for Nonlinear Least Squares

1. Initialize a starting set of parameters, $P^C(t_0)$. If initial values are not given, generate random values over the bounded set $[P^l, P^u]$. Set a value of Δ. Set $i = 1$, $k = 0$ and $I = \varnothing$.

2. Evaluate the residual $r_i \left(P^C(t_k)\right)^2$.

3. If the evaluated residual in Step 2 is greater than Δ then go to Step 4. Otherwise set $i = i + 1$ and go to Step 2. If i cannot be incremented because the end of the data has been reached then stop and output $P^C(t_k)$ as the final set of parameters.

4. Create a new I-frame at time i by doing the following steps:

   (4.1) Perform the incremental optimization by calculating the gradient and adjusting the set of parameters, $P^C(t_k)$, setting $\delta\widetilde{P}_{min}$ and $\delta\widetilde{P}_{max}$ to 1% of $P^C(t_k)$ and finding $\delta\widetilde{P}^*(t)$ as in (5.17). Let $\widetilde{P}(t) = P^C(t_k) + \delta\widetilde{P}^*(t)$.

   (4.2) Set $k = k + 1$, $t_k = i$ and add i to I.

   (4.3) Pick an algorithmic time T and solve the minimization problem (5.4) using the modified fast marching algorithm with the initial set of parameters $\widetilde{P}(t)$ for the integration of (5.3). This determines a new set of parameters $P^C(t_k)$. Set $i = i+1$ and go to Step 2.

To summarize the approach, begin with an initial set of parameters (obtained randomly if a set is not given) and continue through the data until a new I-frame is required. A new I-frame is created if $r_i \left(P^C(t_k)\right)^2 > \Delta$. Use the bang-bang result in (5.17) to update the parameters at the end of the I-frame and add the point to the subset I. Then, solve the minimization problem (5.4) at the new I-frame. This continues until the end of the data. See Fig. 5.3.

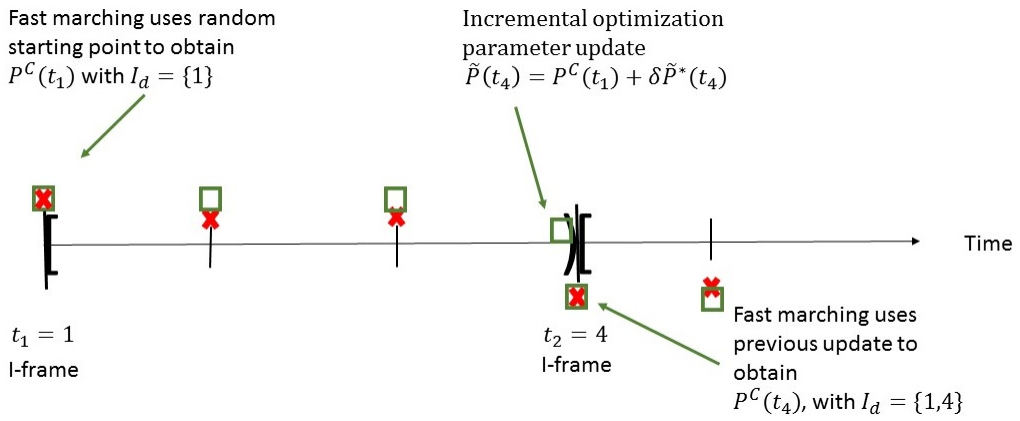

Figure 5.3: Illustration of the I-frame algorithm. The data points are represented by the dark x's. The squares represent the model function evaluated at the parameters, $f\left(\mathbf{x_i}, P^C(t_k)\right)$. At the first I-frame ($t_1 = 1$ in this example), the fast marching algorithm finds a set of parameters $P^C(t_1)$. Just before the next I-frame at time $t_2 = 4$, the incremental optimization modifies the parameters to obtain $\widetilde{P}(t)$. These are used as the starting parameters in the next fast marching algorithm at the next I-frame.

The I-frame methodology is well-suited for the online problem where data is arriving sequentially because the methodology already exploits the sequential nature of the problem. When a new data point arrives, we perform the same check to see if a new I-frame is created by evaluating the residual $r_i \left(P^C(t_k)\right)^2$ as in Step 2 of our algorithm. Further computation is only required if this residual is greater than the threshold Δ, in which case we perform the incremental optimization and run the fast marching algorithm to obtain a new set of parameters. The fast marching algorithm will solve (5.4) with a subset of points in I instead of solving (5.1) with all data points. This eliminates the need to resolve the entire nonlinear least squares problem, which may not be practical in real-time applications where speed is crucial.

In practice, effective values of Δ will be known empirically. Note that if Δ is too small the I-frame methodology may take a long time to run as a lot of I-frames will be created which increases the complexity of solving the minimization problem (5.4). If no information is known about the problem, effective values of Δ will have to be determined experimentally by picking a large value and decreasing Δ until an acceptable value is found. Note that if Δ is chosen to be too large then the I-frame methodology will not perform any optimization (as no I-frames will be created). This occurs when the generation criterion in (5.5) is never satisfied. Thus it is more time efficient to initially pick a large Δ and decrease it according to the results.

5.3 Convergence of the I-frame Methodology

The main result in Theorem 5.3.1 relates the criterion of generating I-frames to the averaged squared error of the final solution. The proof requires Lemma B.0.1 which is proved in the appendix.

Theorem 5.3.1. *Let $P^* = (P_1^*, \ldots, P_n^*)$ be the optimal set of parameters for the nonlinear least squares problem (5.1). There exists a Δ such that if I-frames are generated using the criterion in (5.5) (that is, the squared error at the I-frame is greater than Δ), then the*

I-frame methodology outputs a final solution $P^C(t_k)$ such that the average squared error is bounded by Δ, that is,

$$\frac{1}{m}\sum_{i=1}^{m}\left(r_i\left(P^C(t_k)\right)\right)^2 \leq \Delta. \tag{5.18}$$

Proof. Given an initial set of parameters $P^C(t_0)$ (as in Step 1 of the algorithm), suppose no I-frames are created. Then at each point the squared residual is less than Δ and the average squared error is less than Δ.

Now suppose there is exactly one I-frame created at $t_1 = i$ for some $1 \leq i \leq m$. For every point from t_1 forward we have

$$\frac{1}{m-t_1}\sum_{i=t_1+1}^{m}\left(r_i\left(P^C(t_1)\right)\right)^2 \leq \Delta,$$

since no new I-frames are generated. It remains to show that the average squared error is less than Δ for the points from the start of the data up to the I-frame at t_1, that is,

$$\frac{1}{t_1}\sum_{i=1}^{t_1}\left(r_i\left(P^C(t_1)\right)\right)^2 \leq \Delta.$$

For a proof by contradiction, consider any $\Delta > 0$ and assume that (5.18) does not hold. For notational convenience, let

$$y = \frac{1}{m-t_1}\sum_{i=t_1+1}^{m}\left(r_i\left(P^C(t_1)\right)\right)^2$$

and

$$w = \frac{1}{m}\sum_{i=1}^{m}\left(r_i\left(P^C(t_1)\right)\right)^2.$$

Since the first I-frame occurs at t_1 and no further I-frames are created we have $y \leq \Delta$.

By assumption, $w > \Delta$. This means that the average squared error in the points from the start of the data up to the I-frame is large enough to push the average squared error for the whole data set greater than Δ. The average error for the points from the start of the data up to the I-frame can be calculated as

$$\frac{\text{total error for whole data set - total error in I-frame forward}}{\text{number points in first I-frame}}$$
$$= \frac{wm - y(m-t_1)}{t_1}$$

From these points, $i = 1, \ldots, t_1$, pick the largest squared residual $\left(r_i\left(P^C(t_1)\right)\right)^2$ and set Δ to this new value. Note that this new Δ is larger than the original Δ by assumption that (5.18) is violated. When the I-frame methodology is executed with this new value of Δ and same initial $P^C(t_0)$, one of two things will happen. Either there will be no I-frames or there will be at least one I-frame. If there are no I-frames then we have reached a contradiction as we have found a Δ that satisfies the theorem.

If there is exactly one I-frame again, this I-frame will either be at the same point, t_k, or at a later point (since we have increased Δ). We can use the same technique above to pick a new Δ. Each new Δ picked will be greater than the previous Δ. However, this strategy cannot continue ad infinitum as that would result in an infinitely large Δ. If Δ is infinitely large, then so are the residuals that sum up to Δ. This contradicts Lemma B.0.1 that states there is a finite bound on the residuals. Thus for the situation where there is exactly one I-frame the theorem holds true.

If there are several I-frames the problem can be broken down by dividing up the I-frames. Since the data is ordered sequentially, we can apply the reasoning above to the points from the start of the data up to the second I-frame. This will result in a Δ that satisfies the theorem for the points up to the second I-frame. Using this value of Δ will reduce the number of I-frames, and the same strategy can be applied for each additional I-frame. Continuing in this manner will lead to a Δ that satisfies the theorem. □

5.4 Numerical Results

We numerically compared the fast marching algorithm with I-frames against a trust-region method in Matlab and Powell's dogleg algorithm. Matlab's optimization toolbox provides a function to calculate the solution to a nonlinear least squares problem using a trust-region algorithm based on an interior-reflective Newton method that requires an initial starting point [10, 11]. Powell's dogleg algorithm is described and provided in [39]. The test problems were taken from [55], a public database that provides reference datasets to evaluate statistical

algorithms.

The I-frame methodology was run with $\delta\widetilde{P}_{min}$ and $\delta\widetilde{P}_{max}$ set to 1% of the current parameter value as in Step 4.1. In Step 4.3, the parameter $\alpha = 0.9$ was used in (5.3). The integration step of the fast marching algorithm used the solver ode23s in Matlab [54]. The threshold in the fast marching algorithm was set to 0.001 (if the objective function evaluated at the current point is less than this value, then the algorithm stops). The max iteration count for the fast marching algorithm was set to 40.

The Δ values in the I-frame methodology were picked by first picking an arbitrarily large number and then decreasing Δ until I-frames were created. Recall that if Δ is very large then the squared residuals will be less than Δ at every point, meaning no I-frames are generated. If this happens then no computation is performed outside of calculating the residuals, and the algorithm will output the initial parameter values $P^C(t_0)$ as the final solution. We then decreased Δ further to explore how modifying Δ affects the number of I-frames, the overall time it takes for the I-frame methodology to output a final solution and the total sum of squares of the resulting final solution.

We considered two test problems: *gauss1*, an exponential function with 8 parameters and *ENSO*, a sum of trigonometric functions with 9 parameters. These were chosen as they contained a large number of parameters relative to the other sets available [55]. Each problem was expanded to include more data points. The problem *gauss1* was expanded from 250 to 1000 data points by adding three observations in between the ones given, with the observation values generated by using the optimal set of parameters on the model function and adding Gaussian noise. Similarly *ENSO* was expanded from 168 to 1000 data points.

The trust-region and dogleg methods were each run twenty times using a random starting point with the upper and lower bounds at $\pm 50\%$ of the optimal solution. The Δ value in the I-frame methodology was varied, and each value of Δ was run twenty times with random starting points.

Figure 5.4 presents the results on *gauss1*. The average time (over 20 runs), average solution, best solution and worst solution are reported for each algorithm (Matlab's trust

Method	Delta	Average Time (sec)	Average Solution	Best Solution	Worst Solution
Trust-Region		1429	188,431	6,085	272,865
dogleg		35	1,463,909	284,172	3,171,291
Iframe	1300	64	699,180	38,139	2,682,543
	1400	51	957,405	16,694	8,318,234
	1500	56	734,280	15,578	7,019,848
	1600	61	930,181	72,915	5,061,625
	1700	59	1,063,725	352,879	3,421,485
	1800	39	1,236,664	73,674	5,725,076
	1900	37	918,063	51,773	4,042,495
	2000	38	1,667,859	118,252	7,331,074

Figure 5.4: Results for *gauss1*. The I-frame methodology is able to find a quality solution with the proper choice of Δ. Decreasing Δ too much leads to longer computation time.

region, Powell's dogleg and the I-frame methodology). The optimal function value for this problem is 6085. The trust-region method was able to find the optimal value in 6 of the 20 runs. None of the other methods obtained the optimal solution in any of the runs.

Powell's dogleg method performed the fastest on average (at 35 seconds) however the average solution it found was over a million. Only 3 of the 20 dogleg runs found a solution under a million. In contrast, the I-frame methodology was also able to find a better solution no matter the choice of Δ and also found a better solution on average for all the choices of Δ except $\Delta = 2000$. We also see that the trust region method takes a considerable amount of time to find a solution even though it is able to sometimes find the optimal (6 out of the 20 runs).

As Δ is increased we see that the average solution time decreases. This is to be expected as a smaller choice of Δ is likely to lead to more I-frames, which means performing the optimization and numerical integration (which can be time consuming). We also note that decreasing Δ does not guarantee a better solution. This is shown in the Δ choice of 1700, which performed worse than all other choices. This could be explained by the randomness in choosing initial parameters. Another explanation is that different choices of Δ will naturally lead to I-frames at different points. The set of points where I-frames occur may end up

Method	Delta	Average Time (sec)	Average Solution	Best Solution	Worst Solution
Trust-Region		322	6,025	4,846	6,188
dogleg		41	17,294	9,384	38,788
Iframe	100	91	15,183	8,631	28,399
	125	50	16,716	8,004	33,343
	150	20	16,512	8,955	48,196
	175	7	15,395	8,677	26,978
	200	6	18,741	8,118	28,949

Figure 5.5: Results for *ENSO*. The I-frame methodology is always able to outperform the other two methods in computation time with the proper choice of Δ. Again, as the value of Δ is decreased the I-frame methodology takes longer to output a solution.

fitting the model function very well or they may be very bad choices. However there is a range of Δ choices for which the methodology performs very well (about 1300-1500).

The results for *ENSO* are reported in Figure 5.5. The optimal function value for this problem is 3806, which was not obtained by any of the methods.

Matlab's trust-region method was always able to find a quality solution, however it took a significantly longer time to do so, averaging about four minutes longer than any of the other methods. The dogleg algorithm averaged about 41 seconds to find a solution value which average about 17,000.

Compared to the dogleg method, the I-frame methodology was able to find a better solution on average in less time for Δ values of 150 and 175. For Δ values greater than 200 the I-frame method would frequently not do any optimization at all and output the initial parameters at the final solution. This means that the individual squared residuals did not exceed Δ very often.

From both of these examples we see that the I-frame methodology has local areas where different choices of Δ perform very well. For *gauss1* this occurred when Δ was in the 1300-1500 range and again around 1900. For *ENSO* this occurred when Δ was either 150 or 175. Different choices of Δ will naturally lead to I-frames occurring at different locations. We theorize that some choices of Δ lead to a sequence of I-frames occurring at points that

are a good snapshot for the entire dataset. Optimizing over this sequence produces a more accurate approximation for the entire problem than for other sequences of points.

5.5 Summary

This chapter constructed a methodology for approximating solutions to the nonlinear least squares problem using the concept of I-frames from imaging and animation. This methodology reduces the computation time needed while still producing a quality solution. Our algorithm is able to scale well as the amount of data increases with an appropriate I-frame generation criterion. Furthermore, our algorithm does not require computing a Hessian matrix or assuming a sparse Jacobian matrix.

This methodology fits into the framework for dynamic models presented in Section 3.2. The methodology only performs the expensive computation (updating the parameters) when the model function no longer accurately reflects the data. Furthermore the methodology is well-suited for handling online nonlinear least squares problems.

Chapter 6

SUMMARY AND FUTURE WORK

6.1 Summary

Enterprise applications are growing more complex. Technology allows us to collect and store an abundance of data and access that data quickly, whether it is through the cloud or private servers. Models are becoming more complex as computational power increases and the cost of computing decreases. Many models and algorithms are developed and designed for the static system where there is complete knowledge about the problem and the data. This is no longer applicable with streams of data arriving in real-time and systems growing more complex. There is a need for models and algorithms that adjust on the fly as new data arrives and are able to handle expensive computation when it is required.

This dissertation presents new models and algorithms for large-scale dynamic systems. Utilizing a generic framework for dynamic models, a probabilistic forecaster is developed and a new algorithm for nonlinear least squares is presented. Cloud computing and meta-control can both be used to implement this generic framework, offering storage space and computational speed for finding a quality solution in minimal time. These models and algorithms are used for the example of demand forecasting.

6.2 Future Work on Cloud Computing and Meta-control

Regarding implementation, there are a lot of areas to explore with cloud computing. Chapter 3 provides an implementation scheme in the cloud for dynamic models. This implementation is very generic and allows for any optimization algorithm to efficiently be implemented in the cloud, taking advantage of the internal design of the cloud. It is believed that population-based algorithms would benefit the most from this cloud implementation: it is very natural to

parallelize the population, with the cloud's resources the population size can be increased to ensure the search space is sufficiently explored and population points can be easily removed if they fail or are not performing well without affecting the overall algorithm.

An important metric to analyze would be the amount of time saved by parallelizing a population-based algorithm in the cloud. Furthermore, how much time is saved using algorithms that are not population-based? Some algorithms, such as the modified fast marching algorithm in Chapter 5, can benefit from the cloud using the dedicated resources to do expensive computation. This is expected to save time as a single computer needs to perform the expensive computation and reading and writing data using random access memory (RAM) simultaneously while these tasks are distributed in the cloud.

Computation time in the cloud can be saved by using the implementation in Section 3.3. However there are still decisions to be made when using this implementation. For example, what is the best way to store and access data in the cloud? Since data storage is flexible in the cloud, different methods of storage may result in different algorithm run times. The obvious choice is to pick the method that offers the greatest scalability but it is not immediately clear what this method is.

Decisions also need to be made with regard to acquiring and releasing processors. The cloud already has plenty of built-in analytic tools to make the decision easier, but there are a variety of variables to account for that make this a complex problem. For example, the decision to acquire and release processors should take into an account the time of day (and the day of the week), the current global demand for processors and the size and the location of the processor. Furthermore the decision is also problem dependent, as a population-based algorithm for a global optimization problem may need a lot of processors to effectively explore the search space.

There is a monetary cost to using the cloud and its resources. In order to justify using the cloud there needs to be a proper analysis of the tradeoff between the cost of the cloud and the quality of the solution or the amount of time saved. This analysis would also vary based on the problem being solved. For an algorithm that can be easily parallelized the decision

to use the cloud may be easy. For simpler algorithms the decision may be very complex. A potential user or company seeking to use the cloud also has to factor in extraneous costs: making the switch to the cloud and integrating it within their company. Since processors differ in cost based on their size and power, minimizing the cost of using the cloud is a complex optimization problem with important ramifications.

Implementing meta-control in the cloud can also lead to large cost savings. As described in Section 3.4 meta-control can be used to not only optimize an individual algorithm but optimize a collection of algorithms and the overall usage of the cloud. For example, meta-control can determine which forecaster out of a collection performs the best in a given situation. This is crucial as it allows the user to focus on the decisions to make rather than on the internal workings of the cloud and the forecaster being used. It is clear that further research on the cloud is needed due to the potential the cloud offers. There is great potential to save money in terms of time and resources. But using the cloud can also lend itself to the user being decision-focused with the cloud as a great tool to use in the decision making process.

6.3 Future Work on the Incremental Probability Forecaster

The incremental probabilistic forecaster presented in Chapter 4 propagates a probability distribution directly and uses incremental levels to generate a practical, computationally feasible forecaster. This forecaster has seen little in terms of actual implementation and numerical testing. The incremental forecaster needs to be compared to other forecasters and would be well-suited for a cloud implementation in order to access historical data with ease and distribute computation for the mean field. A big concern is how scalable the incremental forecaster is as the state space increases.

In Section 4.5 ideas from mean field theory are borrowed to create a simplified forecaster to be used in practice. A simplifying assumption is made that a single component of the state can only move up or down *one* incremental level at a time. This is done as it decreases the number of possible transitions. However it is believed that this simplification is not needed.

The mean field approximation would need to be redone to account for the added transitions, but a clever organizing of the transition matrices may lead to very similar results.

The mean field approximation simplified the number of parameters needed to be estimated from $O(3^n)$ to $O(n)$. Further analysis may be able to show how good or bad this simplification is. For example, can we guarantee that this simplification will always be somewhat close to the non-simplified forecaster? Some mean-field techniques may be useful in answering this question.

The optimization problem for solving for the parameters in the forecaster is difficult to solve due to the constraints and large number of variables. Simplifying assumptions can be made to ease the computation but in practice the amount of computation time needed is still large. Original testing solved the problem using the modified fast marching algorithm but there may be better algorithm choices.

6.4 Future Work on the I-frame Methodology

Chapter 5 presents a new methodology for approximating solutions to the nonlinear least squares problems. The main bottleneck for computation is in Step (4.3) of the methodology listed at the end of Section 5.2, solving the minimization problem using the modified fast marching algorithm. The modified fast marching algorithm can take a while to find an optimal solution and may fail to find one if the differential equation is too stiff or ill-posed.

In practice, the choice of which differential equation solver to use can affect the overall runtime of the algorithm and the quality of the solution. A solver for stiff equations (such as Rosenbrock's integration method) is needed for some problems, however quicker solvers exist for problems which are not stiff. There are inherent difficulties in identifying which equations are stiff and which solver will work best a priori. Instead, meta-control can be used to vary the solver as the methodology is running. If one solver is taking a long time or not outputting a good solution, meta-control can switch to a different solver. This has the benefit of identifying which problems may lead to stiff differential equations. However a decision needs to be made on how to switch solvers and identifying when a solver is not

performing well.

A population-based fast marching algorithm can be used to better explore the search space. This would replace Step (4.3) of the methodology with the following steps,

1. Start with M_1 initial starting points, denoted $P_1^C(t_0), \ldots, P_{M_1}^C(t_0)$.

2. Apply a numerical method, such as Rosenbrock's integration method, to each initial point and integrate (5.3) over $[t_0, t_1]$ to obtain M_1 approximate solutions.

3. Of the M_1 approximate solutions, select a subset of M_2 solutions with the "best" values - use the M_2 solutions that have the lowest objective function value, $G(P_i^C(t_1))$, for $i = 1, \ldots, M_1$.

4. Treat the M_2 solutions as initial points for the numerical integration over $[t_1, t_2]$ to obtain M_2 approximate solutions. Of these M_2 approximate solutions, select a subset of M_3 solutions with the "best" values. Continue in this manner until a stopping criterion is satisfied and output the final set of parameters, $P^C(t_k)$.

This population-based fast marching algorithm starts with a population of points which diminishes in each iteration of the algorithm (Step 3). See Figure 6.1. Different selection criterion would need to be explored. For example, the population can simply diminish by 10% at each time step, or a subset of the worst points can be moved to a neighborhood around the best performing point(s). A population-based I-frame methodology would be well-suited for the cloud implementation presented in Section 3.3.

There is a lot of potential for future work on the convergence properties for the I-frame methodology. In Section 5.3 a relationship is established between the criterion for generating I-frames and average squared error of the final solution. This property is beneficial as it is proven for any differentiable function. However the bound is rather crude and is not of much practical importance. In practice it has been found that the average squared error of the final solution never even gets close to this crude bound. There is reason to suspect that an

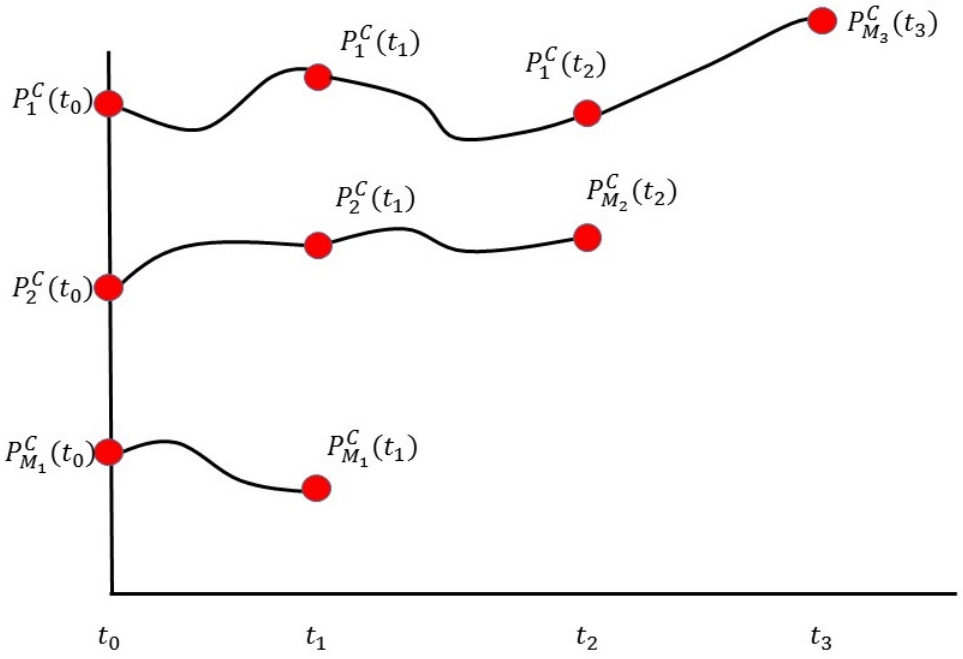

Figure 6.1: Population-based fast marching algorithm. After the integration is performed a subset of the population with the worst objective function values is removed.

even smaller bound can be obtained. Furthermore, it is believed that a smaller bound can be obtained if we place stricter conditions on the model function, such as limiting the model to quasi-convex functions.

It has been clear from practice that the locations of I-frames has an effect on the final solution. For example, consider the set of data in Figure 6.2. The goal is to fit a sinusoidal function to this data using the I-frame methodology. Figure 6.3 displays the locations of the I-frames for two different runs of the I-frame methodology. The top graph in Figure 6.3 is unlikely to produce a function that represents the entire dataset well due to the poor locations of the I-frames. The bottom graph is likely to give us a good result as most sine functions are going to closely match the original dataset.

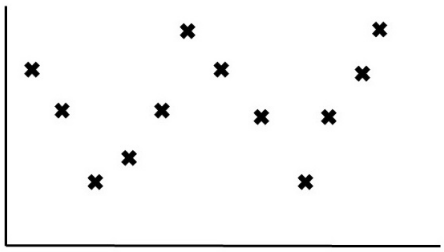

Figure 6.2: A set of data points. The goal is to find a sinusoidal function to the data using the I-frame methodology.

The example in Figure 6.2 is important in showing that the locations of the I-frames can have a large effect on the quality of the final solution. Since I-frames are *not* generated a priori, this poses a problem as it is possible a bad solution may be obtained due to unlucky locations of I-frames. For the offline nonlinear least squares problem, it may be possible to generate I-frames a priori by analyzing the data first. Generating I-frames at local minimums and maximums or at inflection points may be good choices. However, for the online nonlinear least squares problem it is difficult to determine where I-frames should be generated unless there is a large amount of historical data to analyze.

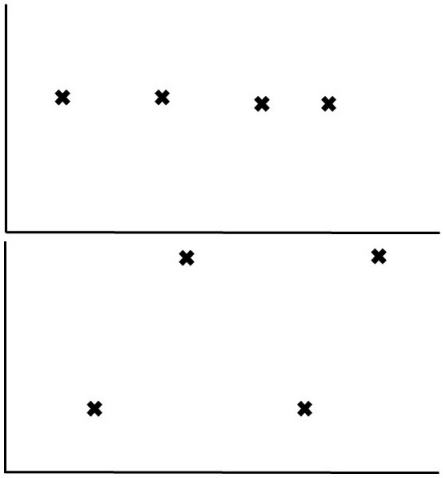

Figure 6.3: The locations of the I-frames for two separate runs of the I-frame methodology for the example in Figure 6.2. The top graph is unlikely to find a function that represents the entire dataset well, while the bottom graph is likely to give us a good result.

Section 5.4 presented some numerical results on the I-frame methodology. Some of these test problems had their data replicated to test the effect a large amount of data had on each of the algorithms. However all the test problems had a relatively small number of parameters. This is due to the difficulty of finding large test problems. It would be beneficial to test the I-frame methodology on problems with a large number of parameters (at least a hundred) to see how the I-frame methodology compares to existing methods. It is believed that the I-frame methodology would perform well due to the sequential nature of the methodology and the use of I-frames.

Chapter 7

Bibliography

[1] Armbrust, Michael; Fox, Armando; Griffith, Rean; Joseph, Anthony D.; Katz, Randy; Konwinski, Andy; Lee, Gunho; Patterson, David; Rabkin, Ariel; Stoica, Ion; Zaharia, Matei. *A View of Cloud Computing*. Communications of the ACM. Vol 53, No. 4. April 2010.

[2] Bensoussan et al. *Linear-Quadratic Mean Field Games*. Journal of Optimization Theory and Applications. Springer US. 2011.

[3] Bishop, C. *Pattern Recognition and Machine Learning*. New York: Springer Science+Business Media, 2006.

[4] Boor, Carl de. *A Practical Guide to Splines*. Applied Mathematical Sciences. Springer. 1978.

[5] Boyd, Stephen; Vandenberghe, Lieven. *Convex Optimization*. Cambridge University Press. 2004.

[6] Breuer, H.P.; Petruccione, F. *The Theory of Open Quantum Systems*. Oxford University Press. 2002.

[7] Bruus, Henrik; Flensberg, Karsten. *Many-Body Quantum Theory in Condensed Matter Physics*. Oxford University Press. 2004.

[8] Burnham, K. P.; Anderson, D. R. *Model Selection and Multimodel Inference*. Springer-Verlag. 2002.

[9] Byrd, R. H.; Lu, P; Nocedal, J. *A Limited Memory Algorithm for Bound Constrained Optimization*. SIAM Journal on Scientific and Statistical Computing, 16, 5, pp. 1190-1208. 1995.

[10] Coleman, T.F. and Y. Li. *On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization Subject to Bounds*. Mathematical Programming, Vol. 67, Number 2, pp. 189-224. 1994.

[11] Coleman, T.F. and Y. Li. *An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds*. SIAM Journal on Optimization, Vol. 6, pp. 418-445. 1996.

[12] Cornelio, Anastasia. *Regularized nonlinear least squares methods for hit position reconstruction in small gamma cameras*. Applied Mathematics and Computation. Volume 217, pp. 5589-5595. February 2011.

[13] Diebold, Francis X. *Elements of Forecasting*. 4th Edition. Thomson South-Western. 2007.

[14] Du, X.; Liu, L; Li, H. *Experimental Study on GPS Non-linear Least Squares Positioning Algorithm*. Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on, Changsha, pp. 262-265. 2010.

[15] Durbin, J.; Koopman, S. J. *Time Series Analysis by State Space Methods*. Oxford Press, NYC. 2001.

[16] Fletcher, Roger. *Practical methods of optimization*. 2nd edition. New York: John Wiley & Sons. 1987.

[17] Gander, Walter; Gander, Martin J.; Kwok, Felix. *Scientific Computing - An Introduction using Maple and MATLAB*. Texts in Computational Science and Engineering. Springer. 2014.

[18] Gao, X. Lu, Y.; Sharma, M.; Squillante, M. S.; Bosman, J.W. *Stochastic optimal control for a general class of dynamic resource allocation problems*. SIGMETRICS Perform. Eval. Rev. 41, 2. August 2013.

[19] Hairer, E.; Wanner, G. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics. 1993.

[20] Heinrich, Juan C.; Pepper, Darrell W. *Intermediate Finite Element Method: Fluid Flow and Heat Transfer Applications*. Series in Computational and Physical Processes in Mechanics and Thermal Sciences. Taylor & Francis. 1999.

[21] The INFORMS 2015 Conference. Available at http://meetings2.informs.org/wordpress/philadelphia/schedule/

[22] Kaess, M.; Ranganathan, A.; Dellaert, F. *iSam: Incremental Smoothing and Mapping*. IEEE Trans. Robotics. Volume 24, no. 6, pp. 1365-1378. December 2008.

[23] Kaps, Peter; Rentrop, Peter. *Generalized Runge-Kutta Methods of Order Four with Stepsize Control for Stiff Ordinary Differential Equations*. Numerische Mathematik, vol. 33, pp. 55-68. 1979.

[24] Kllapi, Herald; Sitaridi, Eva; Tsangaris, Manolis M.; Ioannidis, Yannis. *Schedule Optimization for Data Processing Flows on the Cloud*. Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. June 2011.

[25] Koehl, Patrice; Delarue, Marc. *Application of a Self-consistent Mean Field Theory to Predict Protein Side-chains Conformation and Estimate Their Conformational Entropy*. Journal of Molecular Biology. Volume 239, Issue 2. June 1994.

[26] Kohn, W.; Remmel, J. B.; Moser, W.; Cummings, B. *Free Flight ATC using Hybrid Agent Systems*. In Proceedings of the 36th IEEE CDC. San Diego, CA. 1997.

[27] Kohn, W.; Remmel, J. B. *Hybrid dynamic programming* in Hybrid and Real-Time Systems, Lecture Notes in Computer Science, Editor: Maler, O., Volume 1201, pages 391-396, Springer Berlin-Heidelberg, 1997.

[28] Kohn, Wolf; Zabinsky, Zelda B.; Brayman, Vladimir. *Optimization of Algorithmic Parameters using a Meta-Control Approach*. Journal of Global Optimization. Volume 34, Issue 2, pp. 293-316. February 2006.

[29] Konishi, S.; Kitagawa, G. *Information Criteria and Statistical Modeling*. Springer. 2008.

[30] Kumar, Pankaj; Narayana, S. *Solution of the Fokker-Planck equation by finite element and finite difference methods for nonlinear systems*. Sādhanā Vol. 31, Part 4. 2006. pp. 445-461.

[31] Kushner, Harold J.; Dupuis, Paul G. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer-Verlag. New York. 1992.

[32] Kushner, Harold J. *Probability Methods for Approximations in Stochastic Control and for Elliptic Equations*. Mathematics in Science and Engineering, Volume 129. Academic Press, Inc. 1977.

[33] Lasry, Jean-Michel; Lions, Pierre-Louis. *Mean field games*. Japanese Journal of Mathematics. Volume 2, Issue 1. March 2007.

[34] Levenberg, K. *A Method for the Solution of Certain Nonlinear Problems in Least Squares*. Quart. Appl. Math, Volume 2, pp. 164-168. 1944.

[35] Li, Guang; Liu, Chen-Ching; Mattson, Chris; Lawarrée, Jacques. *Day-Ahead Electricity Price Forecasting in a Grid Environment*. IEEE Transactions on Power Systems. Vol 22, No. 1. 2007.

[36] Liu, D. C.; Nocedal, J. *On the Limited Memory BFGS Method for Large Scale Optimization Methods*. Mathematical Programming 45, pp. 503-528. 1989.

[37] Madsen, K. *A Combined Gauss-Newton and Quasi-Newton Method for Non-Linear Least Squares*. Institute for Numerical Analysis (now part of IMM), DTU. Report NI-88-10. 1988.

[38] Madsen, K.; Nielsen, H.B. *Introduction to Optimization and Data Fitting*. DTU Informatics. Retrieved from `http://www.imm.dtu.dk/pubdb/p.php?5938`. 2010

[39] Madsen, Kaj; Nielsen, Hans Bruun; Tingleff, Ole. *Methods for Non-Linear Least Squares Problems*. 2nd edition. Informatics and Mathematical Modeling. 2004.

[40] Malouf, Robert. *A comparison of algorithms for maximum entropy parameter estimation*. Proc. Sixth Conf. on Natural Language Learning (CoNLL). pp. 1-7. 2002.

[41] Marquardt, D. *An Algorithm for Least-Squares Estimation of Nonlinear Parameters*. J. Soc. Indust. Appl. Math, Volume 11, no. 2, pp. 431-441. June 1963.

[42] Mitra, Sharmishtha; Mitra, Amit. *A Genetic Algorithms Based Technique for Computing the Nonlinear Least Squares Estimates of the Parameters of Sum of Exponentials Model*. Expert Systems with Applications. Volume 39, pp. 6370-6379. 2012.

[43] Molvalioglu, Orcun; Zabinsky, Zelda B.; Kohn, Wolf. *The Interacting-Particle Algorithm with Dynamic Heating and Cooling*. Journal of Global Optimization. Volume 43, Issue 2-3, pp. 329-356. March 2009.

[44] Molvalioglu, Orcun; Zabinsky, Zelda B.; Kohn, Wolf. *Meta-Control of an Interacting-Particle Algorithm for Global Optimization*. Nonlinear Analysis: Hybrid Systems. Volume 4, Issue 4, pp. 659-671. November 2010.

[45] Movahednejad, Mahyar. *Truthful Mechanisms For Resource Allocation And Pricing In Clouds*. Wayne State University Theses. Paper 308. 2014.

[46] Nielsen, H.B. *Direct Methods for Sparse Matrices*. Department of Mathematical Modelling, DTU. 1997. Available at http://www.imm.dtu.dk/~hbni/publ/.

[47] Nocedal, J.; Wright, S. *Numerical Optimization*. 2nd edition. New York: Springer Science+Business Media. 2006.

[48] Osher, Stanley; Paragios, Nikos. *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer. 2003.

[49] Pandey, S; Wu, Linlin; Guru, S.M.; Buyya, R. *A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments*. Advanced Information Networking and Applications. IEEE 2010.

[50] Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. Second Edition. Cambridge University Press. 1992.

[51] Risken, H. *The Fokker-Planck Equation: Methods of Solutions and Applications*. 2nd Edition. Springer-Verlag. New York. 1996.

[52] Rosen, David M.; Kaess, Michael, Leonard, John J. *RISE: An Incremental Trust-Region Method for Robust Online Sparse Least-Squares Estimation*. Proceedings of the 2012 IEEE International Conference on Robotics and Automation. 1262-1269.

[53] Sethian, J.A. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press. 1999.

[54] Shampine, L.F.; Reichelt, M.W. *The MATLAB ODE Suite*. SIAM Journal on Scientific Computing. Vol 18, 1997.

[55] Statistical Reference Datasets. In National Institute of Standards and Technology. Retrieved November 19, 2014, from http://www.itl.nist.gov/div898/strd/index.html. August 1, 1999.

[56] Thrun, S.; Burgard, W.; Fox, D. *Probabilistic Robotics*. Cambrdige, MA: The MIT Press, 2008.

[57] Triggs, B.; McLauchlan, P.; Hartley, R.; Fitzgibbon, A. *Bundle Adjustment - A Modern Synthesis* in Vision Algorithms: Theory and Practice. Lecture Notes in Computer Science. Triggs, Zisserman and Szeliski. vol. 1883, Springer Verlag, pp. 298-372. 2000.

[58] von Haartman, Kathrine; Kohn, Wolf; Zabinsky, Zelda B. *A Meta-Control Algorithm for Generating Approximate Solutions to Binary Integer Programming Problems*. Nonlinear Analysis: Hybrid Systems. Volume 2, Issue 4, pp. 1232-1244. November 2008.

[59] Wait, R.; Mitchell, A.R. *Finite Element Analysis and Applications*. John Wiley & Sons. 1985.

[60] Wei, William W.S. *Time Series Analysis: Univariate and Multivariate Methods*. 2nd edition. Pearson Addison Wesley. 2006.

[61] Williams, Robert. *The Animator's Survival Kit: A Manual of Methods, Principles, and Formulas for Classical, Computer, Games, Stop Motion, and Internet Animators*. Faber and Faber. 2001.

[62] Zhang, Qi; Cheng, Lu; Boutaba, Raouf. *Cloud Computing: state-of-the-art and research challenges*. Journal of Internet Services and Applications. Volume 1, Issue 1, pp. 7-18. May 2010.

[63] Zhang, Pengbo; Kohn, Wolf; Zabinsky, Zelda B. *A Discrete Meta-Control Procedure for Approximating Solutions to Binary Programs*. Entropy 15, no. 9: 3592-3601. 2013.

Appendix A

DERIVATION OF THE FOKKER-PLANCK EQUATION

The derivation of the Fokker-Planck equation follows the approach in [6]. The differential Chapman-Kolmogorov equation is given by:

$$\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} = A(t)P\left((x,t)|(x',t')\right), \tag{A.1}$$

where $P\big((x,t)|(x',t')\big)$ is the conditional transition probability (referred to as the propagator) and A is a linear operator which generates infinitesimal time translations on some density $\rho(x)$,

$$\begin{aligned} A(t)\rho(x) &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\int dx'\bigg(P\big((x,t+\Delta t)|(x',t)\big) - \delta(x-x')\bigg)\rho(x'), \\ &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\int dx'\bigg(P\big((x,t+\Delta t)|(x',t)\big)\rho(x') - \rho(x)\bigg) \end{aligned} \tag{A.2}$$

where δ denotes the Dirac Delta function.

We consider the case where our random variable performs instantaneous jumps, instead of being a smooth solution of a differential equation. We introduce the transition rates $W(x|x',t)$ for the jumps which are defined as follows. The quantity $W(x|x',t)\Delta t$ is equal to the probability density for an instantaneous jump from the state x' into the state x within the infinitesimal time interval $[t, t+\Delta t]$ under the condition that the process is in x' at time t. We can define the total rate for a jump at time t as

$$\Gamma(x',t) = \int dx W(x|x',t).$$

And thus $\Gamma(x',t)\Delta t$ is the conditional probability that the process leaves the state x' at time t by a jump to some other state.

We can formulate an appropriate short-term behavior for the propagator and insert this into (A.2). Putting the result back into (A.1) gives,

$$\begin{aligned}\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} &= A(t)P\left((x,t)|(x',t')\right) \\ &= \int dx''\Big[W(x|x'',t)P\big((x'',t)|(x',t')\big) - W(x''|x,t)P\left((x,t)|(x',t')\right)\Big],\end{aligned} \tag{A.3}$$

which is the differential Chapman-Kolmogorov equation for the jump process, also called the master equation.

We now derive the differential Chapman-Kolmogorov equation for a diffusion process by investigating a certain limit of the jump process described by (A.3). We write the transition rate as,

$$W(x|x'',t) = f(x'',y,t)$$

where $y = x - x''$. Thus W is written as a function f of the starting point x'' with jump increment y at time t. Inserting this into (A.3) gives

$$\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} = \int dy\Big[f(x-y,y,t)P\big((x-y,t)|(x',t')\big)\Big] - P\big((x,t)|(x',t')\big)\int dy f(x,y,t). \tag{A.4}$$

The fundamental assumption is that $f(x'',y,t)$ varies smoothly with x'' but that it is a function of y which is sharply peaked around $y \approx 0$. We can expand the first term on the right-hand side of (A.4) to second order in y,

$$\begin{aligned}\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} = &\int dy f(x,y,t)P\big((x,t)|(x',t')\big) \\ &- \sum_{k=1}^{n}\int dy y_k \frac{\partial}{\partial x_k}\Big[f(x,y,t)P\big((x,t)|(x',t')\big)\Big] \\ &+ \sum_{i=1}^{n}\sum_{j=1}^{n}\int dy \frac{1}{2}y_i y_j \frac{\partial^2}{\partial x_i \partial x_j}\Big[f(x,y,t)P\big((x,t)|(x',t')\big)\Big] \\ &- P\big((x,t)|(x',t')\big)\int dy f(x,y,t)\end{aligned} \tag{A.5}$$

We introduce the first and second moment of the jump distribution as,

$$b_i(x,t) = \int dy y_i f(x,y,t), \qquad a_{i,j}(x,t) = \int dy y_i y_j f(x,y,t).$$

Now we arrive at the Fokker-Planck equation,

$$\begin{aligned}\frac{\partial P\big((x,t)|(x',t')\big)}{\partial t} =& \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2}{\partial x_i \partial x_j}\Big[a_{ij}(x,t)P\big((x,t)|(x',t')\big)\Big] \\ &- \sum_{k=1}^{n}\frac{\partial}{\partial x_k}\Big[b_k(x,t)P\big((x,t)|(x',t')\big)\Big]. \end{aligned} \tag{A.6}$$

Appendix B

PROOF OF LEMMA B.0.1

Lemma B.0.1. *Let $P^* = (P_1^*, \ldots, P_n^*)$ be the optimal set of parameters for the nonlinear least squares problem (5.1) on m data points. Let $P^C(t_k)$ be the final solution given from the I-frame methodology on m data points. There exists a finite $\epsilon > 0$ such that*

$$G\left(P^C(t_k)\right) - G(P^*) = \sum_{i=1}^{m} \left(r_i\left(P^C(t_k)\right)\right)^2 - \sum_{i=1}^{m} \left(r_i(P^*)\right)^2 \leq \epsilon$$

Proof. We assume that $f\left(\mathbf{x}_i, P^C(t_k)\right)$ is continuously differentiable over a bounded set of parameters. This means that $f\left(\mathbf{x}_i, P^C(t_k)\right)$ has a maximum and a minimum over the bounded set which it attains by the extreme value theorem. Thus P^* exists and $G(P^*)$ is finite. Recall that we can write $G\left(P^C(t_k)\right)$ in the following form:

$$G\left(P^C(t_k)\right) = \sum_{i=1}^{m} \left(r_i\left(P^C(t_k)\right)\right)^2 = \sum_{i=1}^{m} \left(y_i - f\left(\mathbf{x}_i, P^C(t_k)\right)\right)^2.$$

Because $P^C(t_k)$ is obtained from the algorithm and y_i is given for all i, $\left|y_i - f\left(\mathbf{x}_i, P^C(t_k)\right)\right|$ has a maximum value (call it r_i^{max}). This means the whole sum is bounded and we can pick our ϵ to be $\sum_{i=1}^{m} (r_i^{max})^2$. □