Learning for Informative Path Planning

by

Sooho Park

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
Department of Electrical Engineering and Computer Science
September 1, 2008

Certified by .....
Nicholas Roy
Assistant Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Acknowledgments

I would like to thank my advisor Nicholas Roy for his support, suggestions and patience. Many thanks to my colleagues who worked on the DDDAS project together, Hanlim Choi, Daniel Gombos, Prof. Jonathan How and Dr. Jim Hansen. I also would like to thank my friends and labmates for their helpful discussions and moral support. I would like to thank the Samsung Scholarship Foundation for its financial support. Last but not least, I would like to thank my family for their love and sacrifices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The weather system affects our lives at every moment. Our daily work and leisure activities are all constrained by weather conditions. Many people remember a day when their great picnic plan was ruined or when they get soaked wet by sudden rainfall. The current and future weather conditions are particularly relevant to people such as seamen who risk their lives on the ocean. Furthermore, if we had a better forecast system, we would have been able to avoid tragedies like the hurricane Katrina incident by evacuating the area before the hurricane hit. For all these, the ability to predict the future state of the weather system has been the prime interest of many researches supported by many governments and individuals.

Numerical Weather Prediction The field of numerical weather prediction (NWP) studies on the mathematical modeling of the weather system and, in turn, techniques for estimating the current weather condition. In principle, if we know the mathematical model(dynamics) of the weather system and the current condition of the system perfectly, we can exactly predict the future weather state. It reduces to the simple problem of finding the numerical solution of the dynamics given the initial weather conditions. In order to get good weather forecasts, we need to have accurate

- Weather dynamics: the mathematical model of the weather system
- Initial weather condition: the current state of the weather system, which will be provided as the initial value for weather dynamics

For the first part, the advances in the mathematical and physical understanding of the weather dynamics has lead to significant improvement in weather prediction over the several decades.

State Estimation and Data Assimilation This thesis is concerned with the problem of determining accurate initial conditions for the weather system. The weather dynamics are *chaotic*. It is well known as *butterfly effect* that a single flap of a butterfly's wing could change the weather condition significantly in the future. Formally, it means that the solution of weather dynamics are very sensitive to the initial condition of the weather. Given two slightly different initial conditions, the difference between the resulting two solution to the dynamics increases with the time of prediction, how far we wish to predict. Thus, having the good estimate of initial weather condition is as important as learning good model of the weather system.

To get the initial condition of the weather system, we have to make measurements of the system. Quantities such as pressure, wind, temperature and humidity are of our interest. This is currently done by many types of observation system such as radiosonde stations, balloons, aircrafts, and satellites [2]. However, the current measurements alone does not give enough information about the current state as we do not have measurements at all parts of the world: we have limited number of sensors. Thus, we have to utilize information from the past measurements and the prediction of how they would have affected the unobserved regions, which is inferred based, in part, on the physical understanding of the system. In addition, the measurements are noisy as the sensors have limited accuracy. We would guess the true state of the system by balancing between noisy measurements and our prediction of the state. As we take more measurements of the system, we would iteratively improve our guess. This iterative or recursive process is called *state estimation*. In the context of NWP, the whole process of estimating weather conditions is called *data assimilation* [26].

Ensemble-based Filtering for Estimating Nonlinear and Complex System As mentioned, data assimilation refers to the process of combining all available information about the system in order to get the best representation of the system. The representation is expected to provide the best estimate of the current state of the system and also the *uncertainty* associated with the estimate.

For example, a weather forcast about possible rain usually includes an estimate for the probability of precipitation. While we try to get the best estimate, we also wish to minimize the uncertainty.

In many state estimation problems, Bayesian filtering schemes are used to estimate the system. It gives a principled and probabilistic way of getting the best estimate of the system given all past measurements and the knowledge of the dynamics governing the system. Furthermore, it quantifies the uncertainty of the estimates.

The Kalman Filter (KF) is probably the best known among these schemes and widely used in many areas [47]. However, its applicability is rather restricted to simple systems where the dynamics of the system is assumed to be linear. Its nonlinear extension, Extended Kalman Filter (EKF), is widely used for estimation when the system is governed by nonlinear dynamics but still can be well approximated locally by linear dynamics.

However, the high-nonlinearity of weather dynamics causes difficulties in applying these schemes in the weather system estimation or data assimilation. Specifically, if the system dynamics are non-linear, chaotic, and of large-scale, computationally intensive Monte-Carlo estimation techniques are often used to well incorporate these characteristics of the dynamics [14]. In these schemes, the condition of the weather system is represented by the distribution of large number of Monte-carlo samples, each representing a possible state of the weather system. The best estimate of the system is essentially decided through *voting*, by of these samples by taking the average value of the samples. The uncertainty of the estimate is decided by the *spread* of these samples, how much they deviate from the best estimate. The Ensemble-based filtering [48] is a technique of this type, which is intensively used for numerical weather prediction and many environmental sensing applications. In this thesis, we adopt the ensemble-based filtering as our estimation framework.

Weather Targeting: an Informative Path Planning Problem A big problem with current data assimilation is that the measurement instruments are unevenly distributed. The measurements on big ocean area such as Pacific ocean are sparse compared to the measurements on land due to many reasons such as maintenance cost and difficulty of deployment. This leads to inaccurate estimation of the state at these regions.

The accuracy of the forecast within these regions suffers due to the sparse measurement avail-

ability as does the forecasting in other regions, as the weather system is an interconnected system, where estimation error in one region propagates easily to other region when making predictions.

We can solve this problem by deploying mobile network of sensors such as aircrafts or Unmanned Autonomous Vehicles(UAV) to these observation-sparse regions (see, for example, NOAA's Winter Storm Reconnaissance Program [44]). As the weather system is complex, measurements at different spots of the system may produce significantly different results in terms of improving forecast performance [2, 33, 37]. Our goal, then, is to maximize the impact of the additional deployments of mobile sensors by carefully selecting the spots or paths that these mobile sensors will follow while taking measurements along the way. This is called *weather targeting* [2, 14].

The value, or *information gain*, that measurements provide are usually evaluated through information theoretic measures, quantifying how much information we acquire from a measurement, within the estimation framework we choose: Typically, the information gain is interpreted as the reduction in the uncertainty that is provided by the measurement The uncertainty is the expected error of the estimation and we expect to improve the forecast through reducing the uncertainty. The weather targeting through the use of mobile sensors is a special case of *informative* path planning problem. Basically, the problem is to evaluate the information gain of candidate paths and choose the best path which has the most information gain.

Computational Burden of Ensemble-based Filtering Ensemble-based filtering can better track and estimate large-scale nonlinear systems, but it incurs significant computational cost for informative path planning. In ensemble-based filtering, an exact calculation of the information gain for a particular path requires the full simulation of each path, which is a series of expensive Monte-Carlo simulations with nonlinear integrations. Information gain of a path is essentially the difference of current uncertainty and future uncertainty after taking a path. Therefore, an efficient way of calculating the uncertainty in future is essential to address informative path planning in a large-scale nonlinear system such as the weather system.

Learning Uncertainty Propagation We need the ability to emulate Monte-carlo simulations in a computationally efficient way to evaluate the information gain of paths. In this thesis, we

propose a strategy by which to learn this nonlinear propagation of uncertainty using past samples of Monte-Carlo simulations, using *machine learning* techniques. The general goal of learning is to find an arbitrary function $f(\mathbf{x}) = y$ given enough past samples of $(\mathbf{x}, y)$ pairs, where $\mathbf{x}$ is a vector of inputs or *features* and y is a scalar output or *label*. In our context, $\mathbf{x}$ is a representation of the current uncertainty and y is future uncertainty, and we then seek to learn a mapping between the current uncertainty and future uncertainty.

We will show that the function f, learned with past samples, is a computationally efficient means to predict the future uncertainty given the current uncertainty, and is much faster than Monte-carlo simulations. The problem of path selection then becomes one of applying this learned function to evaluate information gain of paths. Among many machine learning tools available, we will make use of *regression* to find a function $f(\mathbf{x}) = y$ whose output y is real value. Regression has been studied for decades and there are many approaches that we have to try in finding the best method for our data.

Thesis Statement Through the combined use of regression techniques, we will learn models of the uncertainty propagation efficiently and accurately to replace computationally intensive Monte-Carlo simulations in informative path planning. This will enable us to decrease the uncertainty of the weather estimates more than current methods by enabling the evaluation of many more candidate paths given the same amount of resources. The learning method and the path planning method will be validated by the numerical experiments using the Lorenz-2003 model [32], an idealized weather model.

Chapter 2

Background

In this chapter, we introduce the technical details that are necessary to formally define the informative path planning problem for the weather system estimation. The precise mathematical definition of the problem is needed to attempt to solve the planning problem computationally and evaluate its performance. Many of the details introduced in this chapter are generally applicable to the informative path planning problem in any system, but our focus is on the weather system, which may well be the most challenging system to deal with.

The goal of the informative path planning in the weather system is to ultimately improve the weather forecast. As explained in the introduction, a forecast requires that we have a mathematical model of the weather system and estimate the initial weather condition. In this chapter, we first introduce mathematical modeling of the weather system. Then, we present the details of data assimilation, the weather estimation process, and choose the ensemble-based filtering as the estimation framework for our problem. Given the estimation framework, we define the informative path planning problem as the uncertainty reduction of estimates in this framework. Having introduced the ensemble-based filtering, we then describe the computational challenges of planning within this framework.

2.1 Mathematical Modeling of the Weather System

Recent advances in numerical weather prediction (NWP) has enabled more precise mathematical modeling of the weather system. A mathematical model of a dynamical system such as the weather system is the mathematical representation of 1) the state of the weather system and 2) the dynamics governing the change of the state. We first describe the two parts in general terms, which is applicable to any dynamic system.

2.1.1 Gridspace Representation of Dynamic System

Mathematical representation of the state of a dynamic system is a collection of interacting state variables. These variables are associated with some location in the world we are modeling. Weather system can be abstracted as a collection of variables such as temperature and wind speed at different points of the world. Though the actual world is continuous, we discretize it into a gridspace, where a cell represents some rectangular space in the world. Then, we associate state variables to each cell: for instance, a state variable associated with a cell represent the temperature of that cell. An example gridspace is shown in Figure (2-1). Each dot is the center of a grid cell and has associated state variables. Without loss of generality, we consider the case where a grid cell is associated with only a single variable. We will use the notation $\mathbf{x}_t \in \mathbb{R}^{N_s}$ to denote the state

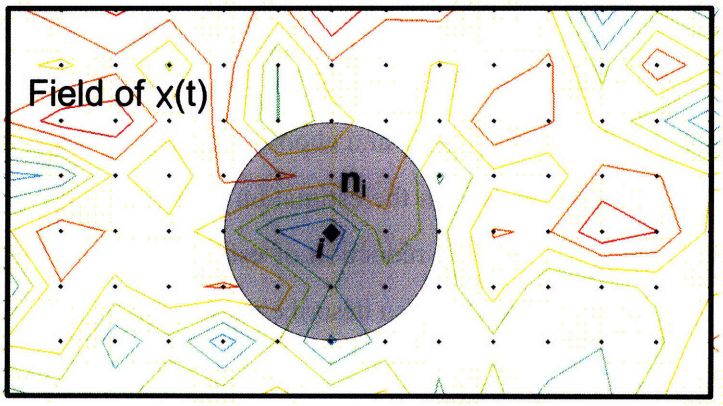


Figure 2-1: State variables over a gridspace; $\dot{x}_i$ is a function of neighboring state variables $\mathbf{n}_i$.

vector at time t, which is the collection of all state variables in the system, where N_s is the total

number of *cells*. Though gridspace may be 2D or 3D, we can always index them using a single variable in some order. To denote an individual state variable of i-th cell, we will use $x(i) \in \mathbb{R}$.

2.1.2 Dynamics on the Gridspace

In a dynamic system such as the weather system, the state variables change over time through interaction with other state variables, according to the actual or *true* dynamics of the system. It is almost impossible to know the true dynamics of a system exactly. However, in many cases, we can model the instantaneous change of the state variables through differential equations, *model* dynamics, with some reasonable error. The dynamics of a state variable at a cell are usually modeled as a function of the state variables of some spatial neighborhood of the cell: the physical interaction between cells in a spatial neighborhood is assumed to be good enough to describe an instant change at a cell. For the state variable of the i-th cell, its instant change is modeled as:

$$\dot{x}_i(t) = f_i(\mathbf{n}_i(t)), \quad i \in [1, N_s] \tag{2.1}$$

The term $\mathbf{n}_i$ represents the state variables in the neighborhood of the i-th cell, which is defined as

$$\mathbf{n}_i = \{x_j : \mathbf{d}(i,j) \leq L_s\} \tag{2.2}$$

where $\mathbf{d}(i,j)$ is some distance measure between grid i and grid j, and L_s is the radius of the neighborhood. The right-hand side function $f_i : \mathbb{R}^{|\mathbf{n}_i|} \mapsto \mathbb{R}$ can be an arbitrary nonlinear function. A dynamic system is called a large-scale dynamic system when N_s is a big number. A large-scale system also may have a higher degree of coupling between state variables, meaning the size of neighborhood $|\mathbf{n}_i|$ is big.

Chaotic Weather System Dynamics In different dynamic systems, the function f_i may have different characteristics. It may be a linear function in simple systems or a complicated nonlinear function for complex systems such as a weather system. Weather dynamics are especially characterized by its highly nonlinear and chaotic behavior.

Definition (Chaotic Dynamics): A dynamic system is called chaotic, if it satisfies the following:

1. It is sensitive to initial conditions.

2. It is topologically mixing.

3. Its periodic orbits are dense.

Suppose that we do not exactly know the temperature of a cell; whether it is 80.1°F or 80.2°F due to noisy measurements. We would not really care about the difference of 0.1°F. However, if the temperature has chaotic dynamics and we were to predict the future temperature, the difference will be significant as the solutions of dynamics are sensitive to that 0.1°F difference. For instance, 20.1°F may give 22°F while 20.2°F gives 30°F as the solutions when we predict the temperature of the cell a few days later.

The first property is most well known as the "butterfly effect". The "butterfly effect" states that even a flap of a butterfly's wing could change the final status of the weather system vastly due to chaotic dynamics. This is a fundamental limitation to weather prediction. The detailed description of chaotic dynamics can be found in [16] and it is not in the scope of this thesis.

2.1.3 Lorenz-2003 Model

There are many weather system models, which differ in terms of the actual region they represent, resolution of the gridspace, the complexity of the dynamics and so forth. Some of these models are used for operational weather prediction such as the Navy's Coupled Ocean Atmosphere Prediction System (COAMPS) [25]. However, it is the nonlinear and chaotic dynamics of these models that differentiates them the most from other dynamic systems.

In this work, we use the Lorenz-2003 weather model [32] for all experiments and for method validation. This is a reduced model that has a smaller number of variables than operational models as a coarser resolution is used. Also, the dynamics are simpler in terms of the size of the spatial neighborhood and the degree of coupling between variables. However, one should note that this model is non-trivial. The Lorenz model is known for its nonlinear and chaotic behavior, and has

been used extensively in the validation of weather targeting methods [34, 31, 12]. The primary reason of using this model instead of operational models is the computational constraint; operational models run on supercomputers and its simulation time is still not much faster than real-time.

The Lorenz-2003 model is an extended model of the well-known Lorenz-95 model [34] that addresses multi-scale feature of the weather dynamics in addition to the basic aspects of the weather motion such as energy dissipation, advection, and external forcing.

We use the two-dimensional Lorenz-2003 model, representing the mid-latitude region ($20 - 70$ deg) of the northern hemisphere. There are $L_{on} = 36\alpha$ longitudinal and $L_{at} = 8\beta + 1$ latitudinal grids in Lorenz models. In the case where $\alpha = \beta = 1$ it is the two-dimensional Lorenz-95 model[12]. The case of $\alpha = \beta = 2$ it is the two-dimensional Lorenz-2003 model. Thus, there are a total of $72 \times 17 = 1224$ state variables in the Lorenz-2003 model. The length-scale of the Lorenz models are proportional to the inverse of α and β in each direction: the grid size for $\alpha = \beta = 2$ amounts to 347 km $\times$ 347 km. The time-scale of the Lorenz models are such that 0.01 time units are equivalent to 1.2 hours in real-time; the duration of 0.01 time units is equivalent to 1 (discrete) timestep in the further discussions.

The equations of the Lorenz-2003 dynamics We introduce the actual equations of the Lorenz-2003 dynamics here. Notice that the instant change of a state variable is a function of its neighbors in these equations. Also, all state variables are governed by the same dynamics; we do not have separate dynamics for each grid though it can be easily generalized to the case with separate dynamics by changing some parameters in the models. In this section, we use a two-dimensional index (i, j) where i denotes the West-to-East grid index and j denotes the South-to-North grid index; $x_{(i,j)}$ is the state variable of (i, j)-th grid. The dynamic equations governing the state variables

is

$$\begin{aligned}\dot{x}_{(i,j)} = &- \xi_{(i-2\alpha,j)}\xi_{(i-\alpha,j)} + \frac{1}{2\lfloor\alpha/2\rfloor+1}\sum_{k=-\lfloor\alpha/2\rfloor}^{k=+\lfloor\alpha/2\rfloor}\xi_{(i-\alpha+k,j)}x_{(i+k,j)} \\ &- \frac{2}{3}\eta_{(i,j-2\beta)}\eta_{(i,j-\beta)} + \frac{\frac{2}{3}}{2\lfloor\beta/2\rfloor+1}\sum_{k=-\lfloor\beta/2\rfloor}^{k=+\lfloor\beta/2\rfloor}\eta_{(i,j-\beta+k)}x_{(i,j+k)} \\ &- x_{(i,j)} + F \\ &i = 1,\ldots,L_{on}, \quad j = 1,\ldots,L_{at}\end{aligned} \tag{2.3}$$

where

$$\xi_{(i,j)} = \frac{1}{2\lfloor\alpha/2\rfloor+1}\sum_{k=-\lfloor\alpha/2\rfloor}^{k=+\lfloor\alpha/2\rfloor}x_{(i+k,j)} \tag{2.4}$$

$$\eta_{(i,j)} = \frac{1}{2\lfloor\beta/2\rfloor+1}\sum_{k=-\lfloor\beta/2\rfloor}^{k=+\lfloor\beta/2\rfloor}x_{(i,j+k)} \tag{2.5}$$

where $\mu = \frac{2}{3}$, $L_{on} = 72$ is the number of variables on the longitudinal axis and L_{at} is the number of variables on the latitudinal axis. The equations contain quadratic, linear, and constant terms representing advection, dissipation, and external forcing. The dynamics of the (i,j)-th grid point depends on its longitudinal 2α-interval neighbors (and latitudinal 2β) through the advection terms, on itself by the dissipation term, and on the external forcing ($F = 8$ in this work).

The dynamics in (2.3) are subject to cyclic boundary conditions in longitudinal direction

$$x_{(i+L_{on},j)} = x_{(i-L_{on},j)} = x_{(i,j)}$$

and a constant advection condition

$$x_{(i,0)} = \cdots = x_{(i,-\lfloor\beta/2\rfloor)} = 3; \;\; x_{(i,L_{at}+1)} = \cdots = x_{(i,L_{at}+\lfloor\beta/2\rfloor)} = 0$$

is applied in the latitudinal direction.

Long-term dependency of the state variables As mentioned, the instantaneous change of the $x_i(t)$, $\dot{x}_i(t)$, can be represented in terms of $\mathbf{n}_i(t)$. Usually, $|\mathbf{n}_i| \ll N_s$; meaning that we only need to know small number of variables to calculate the instantaneous change. However, it should be noted that this local dependency does not mean that $x_i(t + \Delta t)$ is a function of $\mathbf{n}_i(t)$ for finite Δt. For the simplest example, consider a linear system:

$$\dot{x}_i = c_{-1}x_{i-1} + c_0 x_i + c_{+1}x_{i+1}, \qquad \forall i \in \{1, ..., N_s\} \tag{2.6}$$

with boundary conditions $x_0 = x_{N_s+1} = 0$. Then, the mapping from $\mathbf{x}(t)$ to $\mathbf{x}(t + \Delta t)$ becomes

$$\mathbf{x}(t + \Delta t) = e^{C\Delta t}\mathbf{x}(t) = \left[\sum_{k=0}^{+\infty} \Delta t^k C^k / k!\right] \mathbf{x}(t) \tag{2.7}$$

where $C \in \mathbb{R}^{N_x \times N_x}$ is the tri-diagonal matrix with $C_{i-1,i} = c_{-1}$, $C_{i-1,i} = c_0$, and $C_{i+1,i} = c_{+1}$. Even if C is sparse, C^k can be fully populated; therefore, full knowledge of $\mathbf{x}(t)$ is needed to calculate $x_i(t + \Delta t)$ exactly for finite Δt in general.

2.2 State Estimation (Data Assimilation)

Given a weather model, we can make a forecast if we know the initial condition or state of the weather, to input into the model. We must estimate this initial condition from our current and past measurements and also past predictions. This process is called state estimation in general and data assimilation in the context of NWP.

Estimation of state is needed since it is impossible to know the state perfectly. Our sensors can only take noisy measurements due to mechanical, electrical and other limitations. For instance, an odometer in a car can only roughly tell us how far we have traveled. In addition, we have a limited number of sensors. In the case of the weather system, we cannot put sensors everywhere in the space of the earth. We must estimate the state of these regions through knowledge of dynamics such as the Lorenz-2003 dynamics in equation (2.3) and the measured states of other regions.

A state estimation algorithm specifies how to combine the information from all measurements

seen so far and the dynamics. Furthermore, additional measurements may come in continually as a stream and the estimates may need to be recursively updated efficiently.

Bayesian filtering is a probabilistic way of performing this task. In this thesis, we focus on bayesian filtering as the state estimation technique. Its theory is well founded in probability theory, and it has many forms depending on certain assumptions we have about the system. The Kalman Filter (KF) is probably the most famous scheme among the family of Bayesian filters [9]. The KF assumes that the dynamics of the system is linear and that the next state can be described by linear combination of past states. It also assumes that the state variables are Normally distributed (Gaussian). It is the most restricted form of Bayesian filters but it has been successfully applied to many tasks [9]. However, as we have described, weather dynamics are highly nonlinear. As a result, the assumption of linear dynamics in the KF does not hold. We may need more sophisticated filtering techniques.

In the following sections, we first describe Bayesian filtering in general. Then, we introduce the KF to motivate the use of bayesian filtering in state estimation. Finally, we describe the Ensemble-based filtering that can be used for highly nonlinear systems such as the weather system, which is the estimation technique used in this thesis.

2.2.1 Bayesian Filtering

In a Bayesian setting, we represent the noisy information about state variables $\mathbf{x}$ using a probability distribution $p(\mathbf{x})$. For instance, we may not know the exact value of the temperature but know that it is around 80°F as sensed from a noisy sensor. We can represent this knowledge with a probability distribution which assigns higher probability to 80°F and smaller probabilities to the temperatures around 80°F. Bayesian filtering allows one to update this probability distribution according to newly available information. The original distribution is called the *prior* distribution and the updated one is called the *posterior* distribution.

In our state estimation problem, we have two sources of information. First, the measurements give information about the current state with some sensor noise. Secondly, the model dynamics predict the next state given the current state with some model error, the error between the ac-

tual dynamics of the system and our model dynamics. Formally, the measurements $\mathbf{z}_t$ give noisy information about the state variables $\mathbf{x}_t$. We may observe state variables directly or through a transformation:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{w}_t) \tag{2.8}$$

where h can be arbitrary function and $\mathbf{w}_t$ is the sensing noise. $\mathbf{w}_t$ for all t is assumed to be sampled from an identical noise distribution. The model dynamics predict the next state given the current state with some model error:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{r}_{t-1}) \tag{2.9}$$

where f can be arbitrary function and $\mathbf{r}_{t-1}$ is the model error. $\mathbf{r}_t$ for all t is assumed to be sampled from an identical error distribution. Using these two source of information, we have two steps to update the distribution: *prediction(forecast)* update and *measurement(analysis)* update.

$$\text{Prediction update: } p(\mathbf{x}_{t-1}|\mathbf{z}_{0:t-1}) \mapsto p(\mathbf{x}_t|\mathbf{z}_{0:t-1}) \tag{2.10}$$

$$\text{Measurement update: } p(\mathbf{x}_t|\mathbf{z}_{0:t-1}) \mapsto p(\mathbf{x}_t|\mathbf{z}_{0:t}) \tag{2.11}$$

where $\mathbf{z}_{t_1:t_2} = \{\mathbf{z}_{t_1}, \mathbf{z}_{t_1+1}, ..., \mathbf{z}_{t_2}\}$ and $p(\mathbf{x}|\mathbf{z}_{t_1:t_2})$ is the conditional probability distribution of $\mathbf{x}$ given (past) measurements $\mathbf{z}_{t_1:t_2}$. The actual mappings of the prediction and measurement updates depend on the form of functions f and h as well as the form of the distributions.

Different schemes of Bayesian filtering assume different forms of the dynamics f and the observation transformation function h. In addition, the probability distribution of $\mathbf{x}$, the error distribution and the noise distribution are assumed to be in some form. For example, the KF assumes a Gaussian form for all these distributions.

In the next section, we describe the KF, which assumes the most basic form of the functions and distribution.

Kalman Filter

In the KF, all probability distributions are assumed to be Gaussian which can be represented by a mean vector and a covariance matrix. The dynamics are assumed to be linear functions and the

model error is additive Gaussian. This means that the next state of the state variables $\mathbf{x}_t$ is a linear transformation of the current state $\mathbf{x}_{t-1}$ with added Gaussian model error:

$$\mathbf{x}_t = P\mathbf{x}_{t-1} + \mathbf{r}_{t-1}$$

where P is the linear transformation or *propagation* matrix and $\mathbf{r}_{t-1}$ is from a Gaussian distribution $N(0, R)$, or $\mathbf{r}_{t-1} \sim N(0, R)$. The measurements are also assumed to be linear functions of the state variables with added Gaussian sensing noise:

$$\mathbf{z}_t = H\mathbf{x}_t + \mathbf{w}_t$$

where H is the linear transformation or *observation* matrix and $\mathbf{w}_t \sim N(0, Q)$.

As mentioned, the probability distribution of state variables $\mathbf{x}$ is represented by a Gaussian distribution.

$$p(\mathbf{x}) \sim N(\mu, \Sigma)$$

This means that we only need to keep track of μ and Σ in updating the probability distribution of $\mathbf{x}$. Thus, the two update steps in Bayesian filtering become:

$$\text{Prediction update: } (\mu_{t-1}^a, \Sigma_{t-1}^a) \mapsto (\mu_t^f, \Sigma_t^f) \tag{2.12}$$

$$\text{Measurement update: } (\mu_t^f, \Sigma_t^f) \mapsto (\mu_t^a, \Sigma_t^a) \tag{2.13}$$

where the superscripts "f" and "a" denote the *forecast* and *analysis*, which is meteorological terminology. We will use the this notation and terminology throughout this thesis.

The important property of a Gaussian distribution is that a linear transformation of Gaussians is also a Gaussian distribution. The probability distribution of $P\mathbf{x}$ is a linear transformation of $\mathbf{x} \sim N(\mu, \Sigma)$ is given by:

$$p(P\mathbf{x}) \sim N(P\mu, P\Sigma P^T) \tag{2.14}$$

In addition, the sum of two Gaussian distributions are also Gaussian. For two Gaussian distribu-

tions $p(\mathbf{x}) \sim N(\mu_x, \Sigma_x)$ and $p(\mathbf{y}) \sim N(\mu_y, \Sigma_y)$, the distribution of $p(A\mathbf{x} + B\mathbf{y})$ is given by:

$$p(A\mathbf{x} + B\mathbf{y}) \sim N(A\mu_x + B\mu_y, A\Sigma_x A^T + B\Sigma_y B^T) \tag{2.15}$$

Using these properties, the KF algorithm gives the following update equations for the prediction and measurement update:

- Prediction update:

$$\mu_t^f = P\mu_{t-1}^a \tag{2.16}$$

$$\Sigma_t^f = P\Sigma_{t-1}^a P^T + R \tag{2.17}$$

- Measurement update:

$$K_t = \Sigma_t^f H^T (H\Sigma_t^f H^T + Q)^{-1} \tag{2.18}$$

$$\mu_t^a = \mu_t^f + K_t(\mathbf{z}_t - H\mu_t^f) \tag{2.19}$$

$$\Sigma_t^a = (I - K_t H)\Sigma_t^f \tag{2.20}$$

K_t is called the Kalman gain matrix and specifies the optimal balancing between the prior distribution and the new measurement, in a maximum likelihood sense. The exact derivation of the Kalman gain matrix and the update equations of the KF can be found in [9]. Computationally, the KF algorithm is simple linear algebraic operations on the mean vector and covariance matrix, which can be efficiently calculated using standard linear algebra algorithms. Note that the properties of Gaussian distributions in equation (2.14) and equation (2.15) play important roles in the efficiency of the KF algorithm.

However, the assumptions of the KF are too restrictive. In most cases, the dynamics are not linear so that we must resort to another form of Bayesian filter, which can deal with nonlinear dynamics.

Extended Kalman Filter

Extended Kalman Filter(EKF) extends the KF by allowing a nonlinear dynamics function f and nonlinear observation function h, while noises are still assumed to be additive. It deals with the nonlinearity through the linearization of these functions via Taylor expansion.

Specifically, the form of the dynamics in the EKF is:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) + \mathbf{r}_{t-1} \tag{2.21}$$

where f is a nonlinear function and $\mathbf{r}_{t-1} \sim N(0, R)$.

The form of measurement function is:

$$\mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{w}_t \tag{2.22}$$

where h is a nonlinear observation function and $\mathbf{w}_t \sim N(0, Q)$.

Recall that $p(\mathbf{x}) \sim N(\mu, \Sigma)$. First, let us look at the dynamics in the prediction update. To approximate the effect of nonlinear dynamics, given the current distribution $p(\mathbf{x}_{t-1})$, f is linearized around the current mean μ_{t-1} via Taylor expansion:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) \approx f(\mu_{t-1}) + f'(\mu_{t-1})(\mathbf{x}_{t-1} - \mu_{t-1}) + \mathbf{r}_{t-1} \tag{2.23}$$

$$= f(\mu_{t-1}) + P_t(\mathbf{x}_{t-1} - \mu_{t-1}) + \mathbf{r}_{t-1} \tag{2.24}$$

where P_t is the jacobian matrix of f at μ_{t-1}. The measurement function is also approximated similarly and H_t is the jacobian matrix of h at μ_{t-1}. The reason for linearizing around the current mean μ_{t-1} is simple as it is the best guess of $\mathbf{x}_{t-1}$ and it may minimize the error of the approximation.

Unfortunately, the posterior distribution of a Gaussian prior distribution $p(\mathbf{x}_t)$ through a nonlinear function is not a Gaussian; it's not a linear transformation of a Gaussian anymore. However, it is approximated by a Gaussian, meaning only the mean and the covariance of the non-Gaussian posterior distribution are exactly calculated. Thus, in the EKF, there are two sources of error due to nonlinearity; linearization of nonlinear function and a Gaussian approximation of a non-Gaussian

distribution.

Finally, the prediction and measurement update for the EKF are given by:

- Prediction update:

$$\mu_t^f = f(\mu_{t-1}^a) \tag{2.25}$$

$$\Sigma_t^f = P_t \Sigma_{t-1}^a P_t^T + R \tag{2.26}$$

- Measurement update:

$$K_t = \Sigma_t^f H_t^T (H \Sigma_t^f H_t^T + Q)^{-1} \tag{2.27}$$

$$\mu_t^a = \mu_t^f + K_t(\mathbf{z}_t - H_t \mu_t^f) \tag{2.28}$$

$$\Sigma_t^a = (I - K_t H_t)\Sigma_t^f \tag{2.29}$$

Note that now the propagation matrix P_t and the observation matrix H_t is indexed by time, as the result of linearization will be different every time.

The applicability of the EKF depend on the local linearity of dynamics f and observation function h. However, it is not appropriate to use in weather estimation as weather dynamics are significantly more nonlinear and complicated than the dynamics in typical applications of the EKF such as robot navigation [30]. In the next section, we introduce the Ensemble Kalman Filter(EnKF), which is the Monte-Carlo extension of the EKF which does not use linearization but use Monte-Carlo method to approximate the nonlinear propagation of the probability distribution.

2.2.2 Ensemble Kalman Filter: an Ensemble-based Filtering Scheme

Ensemble Kalman Filter(EnKF) is Monte-Carlo (ensemble) extension of the Extended Kalman Filter (EKF), which better tracks highly nonlinear systems such as the weather system used in this work [48]. [1]

[1] For implementation, Ensemble Square Root Filter (EnSRF), a variant of EnKF, is used for its numerical advantages. However, it does not affect the further discussion.

In the KF and EKF, we only kept the mean vector and covariance matrix of the Gaussian distribution of state variables. In the EnKF, we use *ensemble* which is a pool of Monte-carlo samples to represent the distribution of state variables. Each Monte-carlo sample in ensemble is called *ensemble member* and each ensemble member represents a possible state of the weather variables. Formally, ensemble is represented by the ensemble matrix $X \in \mathbb{R}^{N_s \times N_e}$

$$X = \begin{bmatrix} \mathbf{x}^1 & \mathbf{x}^2 & \dots & \mathbf{x}^{N_e} \end{bmatrix} \tag{2.30}$$

where $\mathbf{x}^i \in \mathbb{R}^{N_s}$ is the i-th ensemble member representing a possible state of state variables $\mathbf{x}$ and N_e is the size of the ensemble (number of Monte-Carlo samples).

Like the KF and EKF, the EnKF approximates the probability distribution of $\mathbf{x}$ by a Gaussian distribution. Specifically, EnKF uses the mean of ensemble $\bar{\mathbf{x}}$ and the perturbation matrix of ensemble $\widetilde{\mathbf{X}}$ to approximate the mean μ and the covariance Σ of $p(\mathbf{x}) \sim N(\mu, \Sigma)$:

$$\bar{\mathbf{x}} = \frac{1}{N_e} \sum_{k \in [1, N_e]} \mathbf{x}^k, \quad \widetilde{\mathbf{X}} \equiv \gamma(\mathbf{X} - \bar{\mathbf{x}} \otimes \mathbf{1}_{N_e}^T) \tag{2.31}$$

$$\mu \approx \bar{\mathbf{x}}, \quad \Sigma \approx \frac{1}{N_e - 1} \widetilde{\mathbf{X}} \widetilde{\mathbf{X}}^T \tag{2.32}$$

where $\otimes$ denotes the Kronecker product and $\gamma \geq 1$ is an inflation factor used to avoid underestimation of the covariance Σ due to the finite size of ensemble N_e [48].

It is the prediction step that differentiates EnKF from other two filters. In prediction step, EnKF propagates the ensemble through the nonlinear dynamics without linearization, corresponding to applying the nonlinear dynamics to each ensemble member:

$$\mathbf{x}_t^i = f(\mathbf{x}_{t-1}^i), i \in [1, N_e] \tag{2.33}$$

In our case, the dynamics f is given in a form of differential equation $\dot{x}$ so that equation (2.33) is a nonlinear integration:

$$f(\mathbf{x}_{t-1}^i) = \mathbf{x}_{t-1}^i + \int_{t-1}^{t} \dot{x} dt + \mathbf{r}_{t-1}^i \tag{2.34}$$

Note that the model error $\mathbf{r}_{t-1}^i$ may be different for each ensemble member. Then, using equa-

tion (2.31) and equation (2.32), we get the forecast mean μ_t^f and the forecast covariance Σ_t^f. Due to this nonlinear propagation without linearization, EnKF is able to track the nonlinear propagation of the probability distribution better than the EKF [49].

In this work, we assume that the state variables are directly observed without loss of generality. Furthermore, there may be different number of measurements at different times; we observe a fraction of the state variables in the system through a limited number of sensors which operate at different times.

The form of measurement function is now:

$$\mathbf{z}_t = H_t \mathbf{x}_t + \mathbf{w}_t \tag{2.35}$$

H_t is defined by:

$$H_t(i,j) = \begin{cases} 1, & \text{if } i = j \text{ and } x_i \text{ is observed;} \\ 0, & \text{if } i \neq j \text{ or } x_i \text{ is not observed.} \end{cases} \tag{2.36}$$

Note that H_t is a $N_s \times N_s$ matrix with the rank N.

In sum, the prediction update and measurement update of the EnKF are given by:

- Prediction update:

$$\mathbf{x}_t^i = f(\mathbf{x}_{t-1}^i), i \in [1, N_e] \tag{2.37}$$

$$\mu_t^f = \bar{\mathbf{x}}_t \tag{2.38}$$

$$\Sigma_t^f = \frac{1}{N_e - 1} \widetilde{\mathbf{X}}_t \widetilde{\mathbf{X}}_t^T \tag{2.39}$$

- Measurement update:

$$K_t = \Sigma_t^f H_t^T (H_t \Sigma_t^f H_t^T + Q_t)^{-1} \tag{2.40}$$

$$\mu_t^a = \mu_t^f + K_t(\mathbf{z}_t - H_t \mu_t^f) \tag{2.41}$$

$$\Sigma_t^a = (I - K_t H_t)\Sigma_t^f \tag{2.42}$$

Note that the observation matrix H_t and noise matrix Q_t are indexed with time. It is to consider the cases where we have different number of measurements at different times.

2.2.3 Computational Complexity of the Updates

The EnKF is a computationally intensive estimation framework. In this section, we discuss the computational cost of the two updates in the EnKF.

The prediction update involves the nonlinear integration of every state variables per each ensemble member. Let N_s be the number of state variables, N_e be the number of ensemble members and C_{int} be the nontrivial cost of the nonlinear integration of one variable through the dynamics such as the Lorenz dynamics in equation (2.3). Note that N_e has to be $\Omega(N_s^2)$ for a reasonable estimation of the system with bounded error growth [22]. Then, the nonlinear propagation, the integration of every state variables, takes $O(C_{int}N_sN_e) = \Omega(N_s^3)$ computations. After the propagation, to get a forecast covariance Σ^f from the ensemble, one has to calculate the sample covariance of the ensemble. It takes $O(N_s^2N_e)$ to fill up the total N_s^2 entries of the covariance matrix. Thus, the cost of the prediction update is given by:

$$\text{The cost of the prediction update: } \Omega(C_{int}N_sN_e + N_s^2N_e) = \Omega(C_{int}N_s^3 + N_s^4) = \Omega(N_s^4)$$

where we assume $C_{int} < N_s$ and $N_e = \Omega(N_s^2)$.

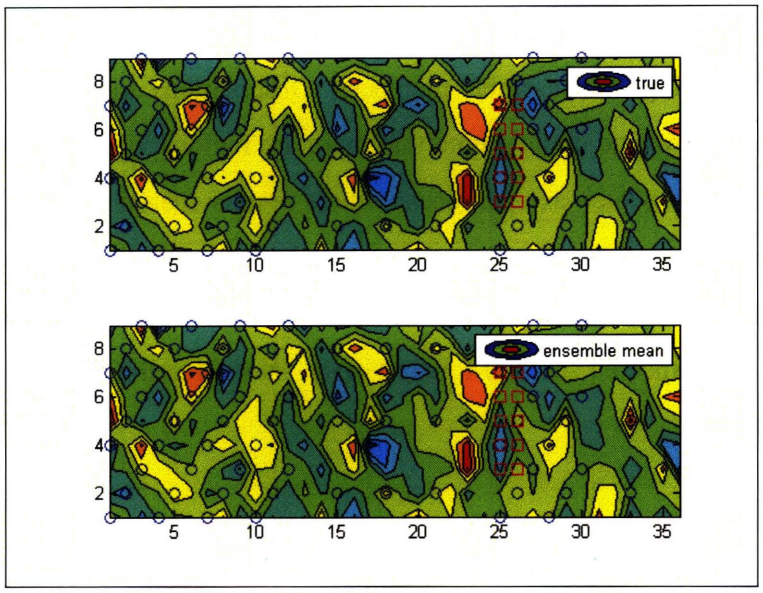
In the measurement update, updating all N_s variables of the system given N observations, the rank of H_t and Q_t becomes both N. Then, the inversion of the matrix $(H_t\Sigma_t^f H_t^T + Q)$ takes $O(N^3)$. The matrix multiplications in equation (2.40) and equation (2.42) involve the covariance matrix of size $N_s \times N_s$ and the rank N matrices, which takes $O(N_s^2N)$. Thus, the cost of measurement update is given by:

$$\text{The cost of measurement update: } O(N_s^2N + N^3)$$

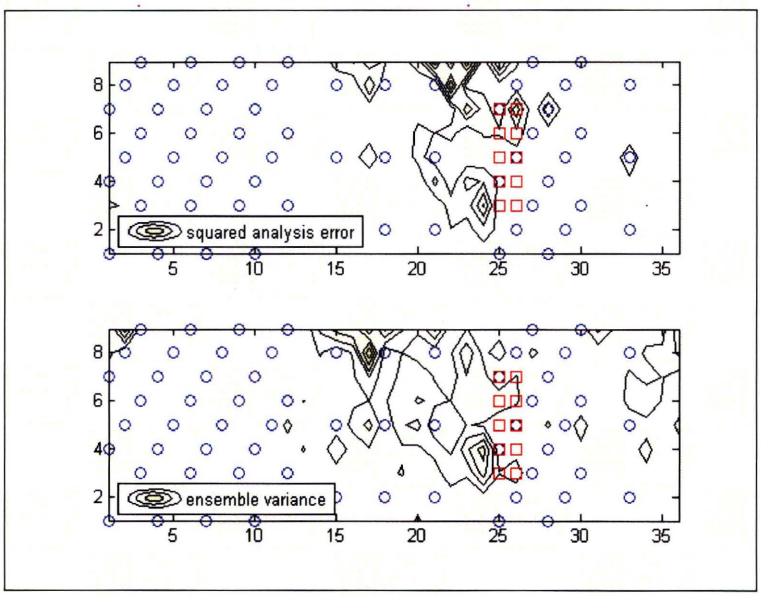
Usually, $N < N_s$.

2.2.4 EnKF in Operation

Figure 2-2(a) shows an example true state of the Lorenz-96 weather model (top) over the 36×9 state variables. The bottom frame shows the estimated state at the same time. This estimate is computed by EnKF using 200 ensemble members. Observations are taken at 66 fixed (routine) locations represented by blue circles; note that there are regions where routine observations are sparse, representing areas such as open ocean where regular measurements are hard to acquire. Figure 2-2(b) (top) shows the squared analysis error between true state and ensemble estimates from the upper figure, that is, the actual forecast error. The lower panel shows the ensemble variance, that is, the expected squared forecast error. Note that the expected and true error are largely correlated.



(a) True vs. Estimated State



(b) Performance Analysis

Figure 2-2: (a) Top panel: the true state of the Lorenz system, an idealized weather model, where the intensity correlates with the state value. Lower panel: The estimated state of the system, using ensemble kalman filter with 200 ensemble members. (b) Top panel: the actual forecast error. Lower panel: the ensemble variance.

2.3 Weather Targeting: an Informative Path Planning

A big problem with current data assimilation is that the measurement instruments are unevenly distributed. The measurements on big ocean area such as Pacific ocean are sparse compared to the measurements on land due to many reasons such as maintenance cost and difficulty of deployment. Figure (2-2)(b) shows an example of this case. The middle area is modeled as the Pacific ocean and the observations in the area are sparse.

This leads to inaccurate estimation of the state at these regions. In Figure (2-2)(b), the error of the estimates of the ocean is higher than other region. In turn, it leads to poor forecast of the regions but also of other regions as the weather system is an interconnected system. The Lorenz dynamics in equation (2.3) shows the example of weather dynamics where the future state of the variables depend on the other variables greatly.

We attempt to solve this problem by deploying mobile network of sensors such as Unmanned Autonomous Vehicles(UAV)s to these observation-sparse regions [44]. As the weather system is complex, measurements at different locations may produce significantly different results in terms of improving forecast performance [2, 33, 37].

Our goal, then, is to maximize the impact of the additional deployments of mobile sensors by carefully selecting the spots or paths that these mobile sensors will follow while taking measurements along the way. This is called *weather targeting* or adaptive observation [2, 14].

Strictly, the goal of original weather targeting is to improve the forecast of some region at some day; for instance, 4-days forecast at two days later from now at California, by choosing or *targeting* spots of the weather system for mobile sensors to take measurements. There are three important times in the problem; planning time T_p, forecast time T_f, and verification time T_v.

- Verification time T_v: the time that forecast will be tested. For instance, 4-days forecast of 9th Sep 2008 will be verified at 13th Sep 2008. Then, T_v is 13th Sep 2008.

- Forecast time T_f: the time to give forecasts about the weather of T_v. T_f was 9th Sep 2008 in the above example.

- Planning time T_p: the planning starts at T_p and the entire planning, execution and forecasting

has to be done by T_f. For instance, T_p may be a few days before T_f.

Fighting directly with the error of forecast is hard problem and we often choose to minimize the uncertainty of the weather condition at time T_v or T_f. This problem is discussed in detail in [14]. Note that we have limited time to plan a path for a mobile sensor from T_p as the plan has to be executed and forecast has to be made before the deadline T_f.

Weather targeting is a special case of informative path planning. The goal of informative path planning is to maximize some *information gain*, or uncertainty reduction, over the possible choices of measurements. The information gain is usually evaluated through information theoretic measures within the estimation framework of the problem.

To describe the problem more formally, we first start with the review of information theoretic measures of uncertainty, which we will use in this thesis. Then, we will formally define the informative path planning problem that we attempt to solve in this thesis.

2.3.1 Information Theoretic Measures of Uncertainty

There are two standard information theoretic measures of uncertainty of a probability distribution: trace and entropy of covariance. They have simple mathematical formula for Gaussian case. We will also explain the intuition behind the metrics. Then, we will define *information gain*, which is the value of certain information.

Trace

The trace of a square matrix is simply the sum of diagonals of the matrix. For a multivariate Gaussian case,

Definition (Trace for Gaussian) For a n-multivariate Gaussian random vector $\mathbf{x}$, with its mean vector μ and covariance matrix Σ, its trace is:

$$H(\mathbf{x}) = \sum_{i=1}^{n} \sigma_{(i,i)}^2$$

An useful property to note is that the trace of a symmetric (thus square) matrix is actually the sum of eigenvalues.

Proposition 2.3.1 *The trace of a symmetric matrix is the sum of eigenvalues of the matrix*

Remember that eigenvalues correspond to the size of the principal components, how stretched a component is to its direction. Therefore, the trace of a covariance matrix roughly corresponds to the circumference of the covariance ellipse. Thus, it may represents the uncertainty of a probability distribution. The smaller the circumference is the more certain (peaked) the distribution is.

Entropy

Roughly, entropy measures the volume of information in a probability distribution. Low entropy corresponds to a peaked distribution, which in turn corresponds to a distribution with low uncertainty. There is high entropy in distributions like a uniform distribution. In fact, a uniform distribution is most uncertain as every states are equally likely. For continuous random variables, it is also called *differential entropy*. *Conditional entropy* is the entropy of a distribution conditioned on new observations.

Definition (Differential Entropy) Given a probability distribution $P(x)$ of a variable X and observations A, *differential entropy* and *conditional differential entropy* are defined as:

$$H(X) = \int_x P(x) \log P(x) dx \tag{2.43}$$

$$H(X|A) = \int_x \int_a P(x,a) \log P(x|a) da dx = H(X,A) - H(A) \tag{2.44}$$

Not every probability distributions have the analytical solutions for the integral above. Fortunately, Gaussian distribution has the solution for the integral as in the theorem below [?].

Theorem 2.3.2 *(Differential Entropy for Gaussian) For n-multivariate Gaussian random vector $\mathbf{x}$, with its mean vector μ and covariance matrix Σ, its differential entropy is:*

$$H(\mathbf{x}) = \frac{1}{2} \ln (2\pi e)^n \det \Sigma$$

Note that the determinant of a symmetric matrix is product of its eigenvalues and that is why it roughly represent the volume of the uncertainty.

Proposition 2.3.3 *The determinant of a symmetric matrix is the product of eigenvalues of the matrix*

Information Gain(IG)

Information gain defines how much more certain and peaked a probability distribution has become after taking observations from the initial state.

Definition (Information Gain) Information gain I of observations A is defined as:

$$I(\mathbf{x}; A) = H(\mathbf{x}) - H(\mathbf{x}|A)$$

where $H(\mathbf{x})$ and $H(\mathbf{x}|A)$ are the uncertainty and conditional uncertainty of probability distribution of $\mathbf{x}$ given observations A. H is either trace or covariance.

Intuitively, information gain is positive when the uncertainty of a distribution decreases by observations and negative otherwise.

2.3.2 Formal Definition of Informative Path Planning

Mathematically, this work addresses the following path planning problem in gridworld described in Section (2.1.1) with a Bayesian state estimation scheme in Section (2.2.1):

$$\begin{aligned} \mathbf{p}^\star &= \arg\min_{\mathbf{p}\in\mathcal{P}} \mathcal{J}(\Sigma^a_{t_K}) \\ \text{subject to } & \Sigma^f_{t_{k+1}} = F(\Sigma^a_{t_k}), \quad \forall k \in [0, K-1] \\ & \Sigma^a_{t_k} = M(\Sigma^f_{t_k}, p_k), \quad \forall k \in [1, K] \\ & \Sigma^a_{t_0} = \text{given}, \end{aligned} \tag{2.45}$$

where $F(\cdot)$ represents the uncertainty propagation via a prediction step, while $M(\cdot, p)$ denotes the uncertainty evolution via an update step with measurement taken at the location p. $\mathbf{p} \equiv$

$\{p_1, p_2, \cdots, p_K\} \in \mathbb{Z}^K$ represents the sequence of grid points that the sensor platform will visit over the time window $[t_1, t_K]$.

The set $\mathcal{P}$ is the feasible set of $\mathbf{p}$ that satisfies certain constraints such as motion of the sensing platforms. $\mathcal{J}(\cdot)$ is the objective function that we want to minimize; trace or entropy. In other words, the informative path planning tries to minimize the *size* of uncertainty at the end of the planning window, where the size is parameterized by the trace or determinant of the final covariance.

Note that we only concern the evolution of Σ not of $\mathbf{x}$. In linear filtering case, Σ can be propagated through linear dynamics without the knowledge of the true measurements. However, in nonlinear filtering case, the true measurements actually change the propagation of Σ. We can only approximately get Σ without the actual measurements.

2.3.3 Related Work

There has been a number of interesting studies for sensor placement strategies [23, 17]. However, these studies focus on finding informative locations for static sensors and most do not consider path selection for mobile sensors. Also, the temporal correlation of measurements are usually not included, instead focusing on spatial correlations between measurements. Observation selection using principal components of ensemble perturbation and its linear transformation was examined in [35], and other work in adaptive sensor placement based on the EnKF used greedy strategies such as randomly choosing one of predefined set of paths [4].

Krause 2006

The work of Krause et al.[28] considers selecting observations for spatial monitoring. Gaussian Process(GP) was used to model the environment. GP is able to provide the estimate and the uncertainty of the estimate at every point in $\mathbb{R}^2$ space. Basically, it interpolates the values of observed points to give estimates for non-observed points. It provides the uncertainty information by defining a spatial prior information between two points in the model; the estimate of a point becomes more certain if there are more measurements around it.

Given the uncertainty information from GP, the algorithm chooses locations that maximize the

information gain through a greedy but efficient algorithm, with the help of the sub-modularity of information gain measure.

However, the temporal correlation of observations were not considered in this setting. An observation at time t has different informative value at other time t' in weather targeting problem because the environment is dynamic and the estimates should be propagated through the knowledge of dynamics at every timestep. The work was mostly interested in a rather static environment.

Hanlim 2007

The work of Choi et al. [14] examines targeting of mobile sensor network with EnKF, but their techniques still requires integration of ensemble members through dynamics, which is computationally intensive. Also, it assumes that the order of observation sequences is not critical to achieve computational feasibility. Basically, it propagates the ensemble to the end of the planning horizon by executing the prediction update multiple times. Then, it calculates an extended covariance matrix where not only spatial correlation but also temporal correlation is included. For every path, the same extended covariance is used. The measurement updates are done on the extended covariance depending on the locations that a path choose to visit. The effect of measurement is to linearly decrease the uncertainty. It is a linear measurement approximation to the true nonlinear uncertainty propagation, as the prediction update and measurement update have to be iterated in turn to give the true future uncertainty.

Chapter 3

Path Selection through Regression of Uncertainty Propagation

The informative path planning requires the prediction of future uncertainties at different times in the estimation framework. In Bayesian filtering schemes, it requires a multiple iteration of the prediction update $F(\cdot)$ and the measurement update $M(\cdot,\cdot)$ of the filter.

Ensemble-based filtering, specifically EnKF, is an computationally-expensive estimation framework that we have to use for highly nonlinear and complex weather system. Unfortunately, $F(\cdot)$ and $M(\cdot,\cdot)$ are computationally expensive in this framework. Computing $F(\cdot)$ requires long series of complex nonlinear integrations for each member of ensemble. Furthermore, if the state dimension is very large, evaluation of $M(\cdot,\cdot)$ is also very expensive.

Learning for Real-time Informative Path Planning Especially, we are concerned with real-time operation of mobile sensing platforms. With limited planning time before actual execution, constant feedback from the environment, and limited computational resources, we wish to have the flexibility to choose a reasonable planning horizon K and the region of interest (ROI) around the measurement locations, where the observations are likely to have the most impact.

For the measurement update, it is often approximated by localizing the effect of a measurement to a small region around the measurement, called *localized* measurement update of EnKF. It even

shows the better performance than the original measurement update as it ignores spurious correlations between two variables which are very far from each other [22]. Thus, we can localize the effect of the measurement to a ROI using the localized measurement update.

However, the prediction update still requires full Monte-carlo simulations, nonlinear integrations of all variables per each ensemble member even if we are only interested in the uncertainty of a ROI. If only we can emulate the Monte-carlo simulations in a computationally efficient way, we can solve the informative path planning problem through fast evaluation of information gain of paths.

In this chapter, we propose to learn the nonlinear uncertainty propagation through *regression* using past samples of Monte-Carlo simulations. Our approach is to learn only the prediction update and use the localized measurement update of EnKF. Let $R = \{c_i | c_i \in [1, N_s], i \in [1, |R|]\}$: the cells of the system belong to a ROI. The learned prediction update will be able to calculate quickly the covariance matrix of a ROI at next timestep $\Sigma_t^f(R)$ using only a small part of the current covariance matrix Σ_{t-1}^a. This step can be recursively done through using the prediction as the input to the next timestep. It will enable us to predict $\Sigma_{t+K}^f(R)$ where K is the planning horizon much faster than original EnKF. By replacing the original prediction update with the learned prediction update, we will be able to evaluate information gain of candidate paths much faster, rendering the informative path planning problem feasible.

In Section (3.1), we will first describe the regression methods. Then, we will formulate the regression problem to learn the uncertainty propagation (covariance updates) in ensemble-based filtering and explain how it can enable fast informative path planning in Section (3.2). Also, the challenges for the learning problem will be discussed, including the nonlinearity of the models we have to learn. Finally, we will show our first attempt to learn the uncertainty propagation and discuss the problems in Section (3.3).

3.1 Regression

Regression is a general learning problem of finding a (target) function or mapping f from input $\mathbf{x}$ to real-valued output y; $f(\mathbf{x}) = y, y \in \mathbb{R}$. The input will be assumed to be a vector of real values

without the loss of generality. For example, suppose that we wish to learn the temperature of a city the next day given the temperature of the current day. Then, the learned function f should give the temperature of the next day, y, with reasonable accuracy given the temperature of some day, x, as the input. Here, we are interested in *supervised* regression. It means that we will get *training* samples to estimate the function f; the pairs of input and output. In the case of above example, we have past examples of how the temperature of the next day was given the temperature of some day.

Let inputs be $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ and outputs be $\mathbf{y} = (y_1, y_2, ..., y_n)$, where n is the number of training samples. The goal is not to perfectly fit the training samples as in

$$f(\mathbf{x}_i) = y_i \tag{3.1}$$

This is easy as we can just let f be

$$f(\mathbf{x}) = \begin{cases} y_i, & \text{if } \mathbf{x} \text{ is equal to } \mathbf{x}_i, \text{ the i-th training sample;} \\ 0, & \text{Otherwise.} \end{cases} \tag{3.2}$$

Rather, we want to find f which will generalizes well to new samples as well as training samples. This will be done by restricting the form of possible functions and choosing a good loss function of residuals, $\epsilon_i = y_i - f(\mathbf{x}_i)$, from which we may predict the future performance.

We may assume different properties for the function f. The most basic form of regression is to assume that f is linear function of input $\mathbf{x}_i$. In that case, f can be represented by

$$f(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ...\beta_m x_{im} \tag{3.3}$$

where βs are the unknown coefficients and x_{ij} is j-th entry of a training sample $\mathbf{x}_i$. The size of each input $\mathbf{x}_i$ is assumed to be m. β_0 is called a *bias* term. We can rewrite equation (3.3) as

$$f(\mathbf{x}_i) = \beta_0 + \beta^T \mathbf{x}_i \tag{3.4}$$

where β is a vector of size m, $[\beta_1; \beta_2; ...; \beta_m]$. In addition, by extending the original feature vector

$\mathbf{x}$ to $\mathbf{x}' = [1; \mathbf{x}]$, we can rewrite equation (3.3) as

$$f(\mathbf{x}_i) = \beta'^T \mathbf{x}' \tag{3.5}$$

where β' is a vector of size $m+1$, $[\beta_0; \beta_1; \beta_2; ...; \beta_m]$. We use this form throughout the thesis.

Then, the estimation or learning of f is to find the coefficients β through minimizing some loss function of prediction errors, $e_i = f(\mathbf{x}_i) - y_i, i \in [1, n]$. The loss function should be a good indicator of the prediction performance of the learned function f. Different learning algorithms use different loss functions and it will lead to different estimation of f given the same training samples. We first introduce the most basic linear regression algorithm, the Least Squares Regression(LSR).

3.1.1 Linear Regression Algorithms

Least Squares Regression

The most well-known and basic regression technique is the Least-Squares Regression (LSR). It finds the function which minimizes the squared sum of errors over training samples. Let $SSE(\beta)$ be the sum of squared error over the training samples given the coefficients β,

$$SSE(\beta) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 \tag{3.6}$$

$$= \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{m} x_{ij}\beta_j)^2 \tag{3.7}$$

Then, the LSR finds β which has the minimum $SSE(\beta)$.

From a statistical point of view, this is maximum likelihood estimation(MLE) assuming that errors are independent random samples from a gaussian distribution. Also, the LSR is a very intuitive estimation which averages error over given samples.

The solution β can be found by different derivations. It is often easier to work with matrix form

of equation (3.7) as follows,

$$SSE(\beta) = (\mathbf{y} - X\beta)^T(\mathbf{y} - X\beta) \tag{3.8}$$

where X is $n \times (m+1)$ matrix of inputs $[\mathbf{x}_1; \mathbf{x}_2; ...; \mathbf{x}_n]$, where the original input is extended with a bias term, and $\mathbf{y}$ is $n \times 1$ vector of outputs $[y_1; y_2; ...; y_n]$.

The first derivation is using differentiation. Differentiating equation (3.8) with respect to β, we get

$$\frac{\partial SSE}{\partial \beta} = -2X^T(\mathbf{y} - X\beta) \tag{3.9}$$

$$\frac{\partial^2 SSE}{\partial \beta \partial \beta^T} = -2X^T X \tag{3.10}$$

If we set equation (3.9) to 0, we get

$$X^T(\mathbf{y} - X\beta) = 0 \tag{3.11}$$

$$\hat{\beta} = (X^T X)^{-1} X^T \mathbf{y} \tag{3.12}$$

equation (3.12) is often called *normal equation* and gives the unique solution of the LSR. The derivations of the solution for other regression algorithms are similar to the above derivation.

Actually inverting the matrix $X^T X$ is not desirable due to the computational load of inverse operation as well as the numerical instability. In practice, it is usually done via Cholesky decomposition of $X^T X$ or QR decomposition of X. With n observations and m features, the Cholesky decomposition takes $m^3 + nm^2/2$ and the QR decomposition takes nm^2.

Cholesky Decomposition for LSR Given a positive definite matrix Σ, there exist many matrices C such that $\Sigma = CC^T$. Cholesky decomposition finds the C which is lower triangular. We will write it as L to emphasize that it's lower triangular. We can also use upper triangular matrix R

where $L^T = R$,

$$LL^T = R^T R \tag{3.13}$$

Note that the inverse or the transpose of a triangular matrix is also triangular. Solving a linear system which corresponds to a triangular matrix is straightforward, as it's already in the form where back-substitution can be used directly.

Given the cholesky decomposition of $\Sigma = X^T X$, we can rewrite equation (3.12) as

$$\hat{\beta} = (R^T R)^{-1} X^T \mathbf{y} \tag{3.14}$$

$$(R^T R)\hat{\beta} = X^T \mathbf{y} \tag{3.15}$$

$$R^T w = X^T \mathbf{y} \tag{3.16}$$

where $R\hat{\beta}$ is substituted by w. Now, we solve two linear systems to get $\hat{\beta}$.

1. Solve the lower triangular system $R^T w = X^T \mathbf{y}$ to get w
2. Solve the upper triangular system $R\hat{\beta} = w$ to get $\hat{\beta}$

Cholesky decomposition is a general technique which can be used to solve most of the regression problems introduced in this thesis.

Regularized Least Squares

The LSR algorithm is known to be numerically instable and sensitive to the training set; using different training set may lead to significantly different estimation of function f. This sensitivity to training set is problematic as we expect the function to generalize well to new examples; it must be less sensitive to specific choice of training samples. This problem is solved by *regularization* [20].

Regularization is to put the extra information or *prior* over the regression coefficients β. We would prefer to have small $|\beta|^2$ as big $|\beta|^2$ means that the function is less *smooth* and thus more sensitive to training samples [20]. Thus, we penalize the norm of regression coefficients through

adding *regularization penalty* to the original loss function of the LSR. Effectively, we get biased estimates of β with lower variance through regularization.

The Regularized Least Squares(RLS) algorithm minimizes the sum of the squared-error of the training samples with regularization penalty,

$$\min_{\beta} \quad ||(X\beta - \mathbf{y})||^2 + \lambda\beta^T\beta \tag{3.17}$$

where λ is regularization parameter which controls the contribution of the L2-norm of regression coefficients β to the total loss function; small λ encourages big $|\beta|^2$ and big λ encourages small $|\beta|^2$.

To minimize (6.6), we set the derivative of (6.6) with respect to β to 0, and get

$$\hat{\beta} = (X^{\mathrm{T}}X + \lambda I)^{-1}X^{\mathrm{T}}\mathbf{y}. \tag{3.18}$$

The RLS and regularization technique has been proven successful in learning functions which can generalize well [20, 50, 38]. Specifically, the RLS perform better than the LSR with the careful selection of λ through some model selection criteria such as cross-validation.

3.1.2 Nonlinear Regression Algorithms

In many regression problems, often linear regression is not enough to learn the good mapping between X and $\mathbf{y}$; the relationship between the input and the output is not linear. For instance, the dynamics in the weather models are nonlinear so that the input of the dynamics and the output has a nonlinear relationship. In that case, we have to use nonlinear regression algorithms.

There are two ways of doing nonlinear regression:

- Linear regression with explicit higher-order features: One can build higher-order feature set from original feature set. We will discuss how linear regression with this higher-order feature set is nonlinear regression with original feature set.

- Kernel regression with implicit higher-order features: Through the use of kernels, one can learn a

nonlinear function in original feature space. Kernel implicitly maps original features into higher-dimensional space. For example, Gaussian kernel maps features into infinite-dimensional space, which is not possibly done through explicit feature construction.

We introduce the two methods in the following sections.

Nonlinear Regression through Feature Transformation

One should note that linear function form in equation (3.3) is fairly flexible and it can model nonlinear functions of original input by some nonlinear transformation of original input. For a simple example, we may transform original sample

$$\mathbf{x}_i = (x_{i1}, x_{i2}, ...x_{im})$$

to

$$\mathbf{x}'_i = (x_{i1}, x_{i1}^2, x_{i2}, x_{i2}^2, ...x_{im}, x_{im}^2)$$

by adding squared terms of original features; this is called *basis expansions*. Then, a linear function of $\mathbf{x}'_i$ will be a nonlinear or quadratic function of original feature $\mathbf{x}_i$.

Nonlinear Regression through Kernel methods

Kernel Regularized Least Squares Kernel methods use implicit higher-order feature mapping; it does not require explicit higher-order feature construction as in Section (3.1.2). We derive the Kernel Regularized Least Squares (KRLS) algorithm from the original RLS solution to show an example of kernel methods.

Let X be a row concatenation of training samples and $\mathbf{y}$ be a column vector of labels. The original RLS minimizes the sum of the squared-error of the training samples with a regularization penalty,

$$\min_{\beta} \quad \{||(X\beta - \mathbf{y})||^2 + \lambda\beta^{\mathrm{T}}\beta\} \tag{3.19}$$

where β is a column vector of regression coefficient(weights), and λ is a regularization parameter which controls the L2-norm of weights β. To minimize (6.6), we set the derivative of (6.6) with respect to w to 0, and get

$$\beta = (X^{\mathrm{T}}X + \lambda I)^{-1}X^{\mathrm{T}}\mathbf{y}. \tag{3.20}$$

The prediction for a new sample $\mathbf{x}^*$ is given by

$$y^* = \beta^{\mathrm{T}}\mathbf{x}^* \tag{3.21}$$

Now, let $\phi(\mathbf{x})$ be a mapping of the original features $\mathbf{x}$ to a higher-dimensional space; for instance, basis expansions. The solution of the RLS in equation (3.20) using the new features $\phi(\mathbf{x})$ can be written as:

$$\beta = (K + \lambda I)^{-1}K'^{T}\mathbf{y}. \tag{3.22}$$

where

$$K = \begin{pmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_n) \\ \dots & \dots & \dots \\ \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_n) \end{pmatrix} \tag{3.23}$$

$$K' = \begin{pmatrix} \phi(\mathbf{x}_1) \\ \dots \\ \phi(\mathbf{x}_n) \end{pmatrix} \tag{3.24}$$

K is called the kernel matrix. The prediction for a new sample $\phi(\mathbf{x}^*)$ is now given by

$$y^* = \beta^{\mathrm{T}}\phi(\mathbf{x}^*) = \mathbf{y}^{\mathrm{T}}(K + \lambda I)^{-1}K'\phi(\mathbf{x}^*) = \mathbf{y}^{\mathrm{T}}(K + \lambda I)^{-1}K^* \tag{3.25}$$

where

$$K^* = \begin{pmatrix} \phi(\mathbf{x}_1)\phi(\mathbf{x}^*) \\ \dots \\ \phi(\mathbf{x}_n)\phi(\mathbf{x}^*) \end{pmatrix} \tag{3.26}$$

Note that the prediction for a new sample, the solution of the model, only depend on the inner prod-

ucts of features. Thus, instead of choosing a mapping ϕ to explicitly build a higher-order feature set, we can use a kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)$ to build the kernel matrix K and the matrix K^*. The mapping ϕ is chosen implicitly through the choice of the kernel function. Intuitively, kernel function k defines some *similarity* or distance metric between two samples. However, note that the solution of regression is not parameterized; the function f has to be reconstructed for each prediction through the calculations of kernel function k to build the matrix K^*.

A kernel function should satisfy Mercer conditions [10]. Mercer conditions basically requires the kernel matrix to be positive semidefinite; the inverse of $K + \lambda I$ should exist.

The examples of kernel functions Two popular kernels are polynomial kernel and Gaussian kernel:

- Polynomial Kernel (of degree d): $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$
- Gaussian Kernel (of bandwidth σ): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2})$

It may be beneficial to discuss some characteristics of kernels. For polynomial kernel, a higher d will increase the capacity of a classifier so that the resulting classifier can have more complex decision boundary. Basically, d-th polynomial kernel includes the effect of up-to-d-th order interaction terms of original features. The Gaussian kernel maps the original features to a infinite-dimensional space. The influence of a sample to other sample will be controlled by the bandwidth parameter σ.

Support Vector Machines Support Vector Machines (SVM) is another very popular kernel method, which is regarded as the state-of-the-art learning algorithm [15]. The SVM minimizes the hinge-loss function, or the ϵ-tube loss function, instead of the squared-error loss in the KRLS. The hinge-loss function is not smooth function, encouraging sparsity in the solution; only a fraction of the training samples contribute to the solution. For more details, refer to [39].

The actual computation of the KRLS is to solve a simple linear system while the SVM needs a quadratic solver. Through the use of well-developed linear solvers, the KRLS can be computationally quite effective in practice. In our experiments, the KRLS was much faster than the SVM using a same type of kernel.

The KRLS has shown comparable generalization results to the state-of-the-art SVM [39, 51]. Compared to the SVM, the KRLS also enjoy fast and exact incremental updates of the solution when a new training sample is available [18, 46].

3.2 Applying Regression to Learn Uncertainty Propagation

We have described the basic regression algorithms. As described, (supervised) regression is to find the target function f given training samples; features $X = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ and labels $\mathbf{y} = (y_1, y_2, ..., y_n)$.

In this section, we attempt to use regression to learn uncertainty propagation in ensemble-based filtering. In propagating the uncertainty for one timestep in ensemble-based filtering, there are two update steps; prediction and measurement update. To propagate the uncertainty multiple timesteps we have to iterate these two steps; for k-step lookahead planning, we have to iterate the two updates for k times.

We expect to replace the iteration of two updates with some learned function(model) f or a set of functions F which can be evaluated faster than the original updates. For instance, we may replace k iterations of updates with a direct mapping f from the current covariance to the trace of the final covariance after k iterations:

$$f : \Sigma_t \mapsto \mathrm{tr}\{\Sigma_{t+k}(R)\} \tag{3.27}$$

The evaluation of f should be faster than the updates through a series of Monte-carlo simulations for prediction update and linear algebra operations for measurement update. However, the accuracy of f will be very important. It the error of the prediction through f is too high, we cannot use it for reliable informative path planning.

In using regression, we have to decide

1. What to learn: Section (3.2.1)
2. What features to use: Section (3.2.2)

3. What algorithms to use: Section (3.2.3)

In deciding what to learn, we have to consider the fact that as the target function f is more complicated, we will need more training samples, good features and sophisticated method to learn. Preliminary result showed us that learning certain functions, such as the function in equation (3.27), is extremely hard. Also, we may want to learn common functions which can be applied to different paths of different lengths. If we decide on the target function, we have to generate features $\mathbf{x}$ from each training sample. In our case, the most obvious features are state estimates itself such as mean and covariance of ensemble distribution. Other possible features may be related to structure of routine network. In terms of algorithms, there are many regression algorithms that we can choose from, which may have different computational requirements, sample complexity, capacity of learning and generalization performance.

3.2.1 Learning the Prediction Update of Ensemble-based Filtering

In this thesis, we propose to learn the (one timestep) prediction update of ensemble-based filtering. Remember that the prediction update is the computationally expensive part of ensemble-based filtering, which is done through Monte-carlo simulations. In addition, prediction update has to be done at every timestep to get the forecast covariance of next timestep while measurement update has to be done only when a measurement is available. Furthermore, the measurement update can be done exactly as in ensemble-based filtering if we have the forecast covariance. If we can emulate the *original* expensive prediction update through the *learned* prediction update, we can use it to evaluate any path of any length faster than original ensemble-based filtering.

Learning the prediction update will be done by learning the *propagation* functions of each covariance entries, which is the set of functions $\hat{F}$:

$$\hat{F} = \{\hat{f}_{(i,j)} | (i,j) \in (1...N_s, 1..N_s)\} \tag{3.28}$$

$$\hat{f}_{(i,j)} : \mathbf{E}_{(i,j)}(\Sigma^a_{t-1}) \mapsto \Sigma^f_t(i,j) \text{ for all } t \tag{3.29}$$

where $\mathbf{E}_{(i,j)}$ is some feature extractor for the propagation function of the (i,j)-th entry of covari-

ance. Recall that $\Sigma^a_{t-1} = \Sigma^f_{t-1}$ if there is no measurement at time $t-1$. In that case, we will assume that a measurement update with zero computational cost is done on Σ^f_{t-1} to get Σ^a_{t-1}. To learn the functions $\hat{F}$, we will use the past samples of Monte-Carlo simulations in ensemble-based filtering as training samples.

Note that unlike the original prediction update, we can only update a part of the analysis covariance Σ^a. For instance, we can get $\Sigma^f_t(R)$ from Σ^a_{t-1} by only applying functions $\hat{f}_{(i,j)}, (i,j) \in R$. Also, remember that the original measurement update can be done locally on a part of the covariance matrix. These two properties will enable to evaluate the uncertainty of ROI faster after taking a path.

Recursive prediction for multistep updates

Note that one application of $\hat{F}$ give only the covariance of the next timestep but the covariance after a number of timesteps is needed for evaluating the information gain of a path at that time. We will use *recursive prediction* to resolve this. Recursive prediction is to use predicted values of a function as the input to the function. Suppose we are given the analysis covariance Σ^a_0. To predict the forecast covariance $\hat{\Sigma}^f_t$:

1. Predict $\hat{\Sigma}^f_1$ from Σ^a_0.
2. Do measurement update on $\hat{\Sigma}^f_1$ to get $\hat{\Sigma}^a_1$.
3. Predict $\hat{\Sigma}^f_1$ from $\hat{\Sigma}^a_1$
4. Repeat until we get $\hat{\Sigma}^f_t$.

This is recursive prediction as we are using the predicted covariances as the input to get the covariance of next timestep.

Applying the propagation functions for path selection

Once we have learned the functions $\hat{F}$, we can predict the uncertainty of the system after taking a series of observations by applying the propagation functions for prediction update and using

the original measurement update. Given the uncertainty at future time through it, path planning problem becomes trivial as to choose the path which reduces the uncertainty the most, as in the algorithm 1.

Algorithm 1 Path Selection Algorithm

Input: Initial (analysis) covariance Σ_0^a and learned models $\hat{F}$.
for all paths p^k **do**
  for all locations $l_t \in p^k$ **do**
    Propagate (needed) elements of the covariance $\Sigma(i,j)$ according to the learned model $\hat{F}(i,j)$
    Perform measurement update at location l_t.
  end for
  Compute the posterior trace of the R sub-block of Σ, $\Sigma(R)$.
end for
Return the path with minimum posterior trace of $\Sigma(R)$.

3.2.2 Spatial Neighborhood Features

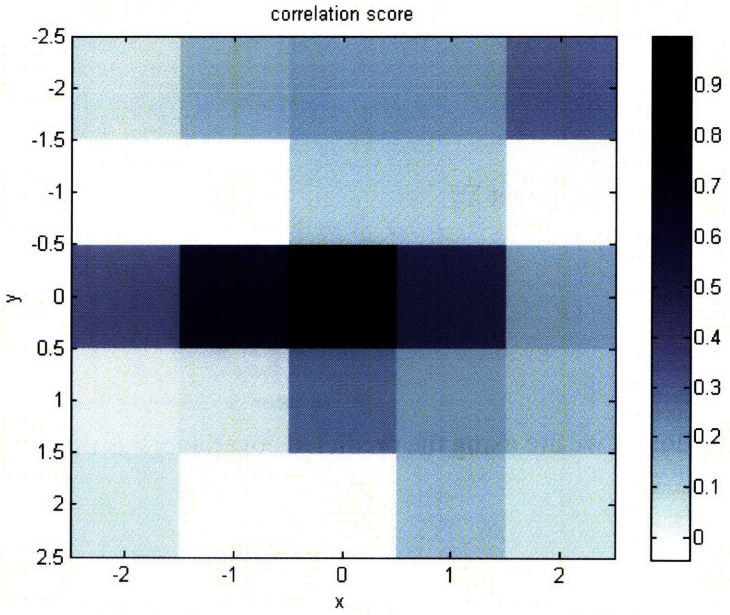


Figure 3-1: Correlation coefficient(score) between $\Sigma_t^f(k,k)$ and $\Sigma_{t-1}^a(k+i,k+i), i \in [-2,2]$ from 4000 samples. The coefficient is the biggest for $\Sigma_{t-1}^a(0,0)$ and roughly decrease with distance. Correlation score is an indicator for the usefulness of $\Sigma_{t-1}^a(k+i,k+i)$ as a feature to learn the propagation function $\hat{f}_{(k,k)}$

For many physical systems, the dynamics are inspired by actual physical phenomena which often has the spatial locality of interaction at any moment. Hence, the next state of a variable is strongly affected by its small spatial neighbors as in equation (2.1).

We conjecture that the uncertainty of a variable, a collective behavior of Monte-Carlo samples, will also depend largely on small spatial neighbors, with trade-off between accuracy and size of spatial neighborhood. Figure (3-1) shows the correlation score between a part of analysis covariance $\Sigma^a_{t-1}(k+i, k+i), i \in [-2, 2]$ and $\Sigma^f_t(k,k)$ for some k. The correlation score between a feature and labels is a good indicator of the usefulness of that feature to learn the function of labels [24]. It shows the evidence that the propagation functions may be learned using the features extracted from a spatial neighborhood.

Therefore, in learning a model $\hat{f}_{(i,j)}$ to predict a covariance element $\Sigma(i,j)$, we will extract features from the statistics of the spatial neighborhood of grid cells i and j. Let the spatial neighborhood of grid cell i be $\mathbf{N}_i$. $\mathbf{N_i}$ is defined as a set of grid cells $\{l \,|\, ||l-i||^2 \leqq L\}$ within some distance L. The possible features are:

- Mean estimates of neighbors (Mean): $\mu(\mathbf{N}_i)\ \mu(\mathbf{N}_j)$
- Covariance diagonal elements (Var): $\Sigma(\mathbf{N}_i, \mathbf{N}_i)\ \Sigma(\mathbf{N}_j, \mathbf{N}_j)$
- Covariance rows or columns (CovR): $\Sigma(i, \mathbf{N}_i)\ \Sigma(i, \mathbf{N}_j)$
- Covariance blocks (CovB): $\Sigma(\mathbf{N}_i)\ \Sigma(\mathbf{N}_j)$

where $\Sigma(\mathbf{N}_i) = \{\Sigma(i,j) | i,j \in \mathbf{N}_i\}$ and $\Sigma(i, \mathbf{N}_i) = \{\Sigma(i,k) | k \in \mathbf{N}_i\}$. We gave names for each type of features such as "Mean" and "Var".

In sum, for a model $\hat{f}(i,j)$, the features X and labels $\mathbf{y}$ are given as

$$X = \begin{pmatrix} \mathbf{E}(\Sigma_0) \\ \mathbf{E}(\Sigma_1) \\ \cdots \\ \mathbf{E}(\Sigma_{W-1}) \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} \Sigma_1(i,j) \\ \Sigma_2(i,j) \\ \cdots \\ \Sigma_W(i,j) \end{pmatrix}$$

where $\mathbf{E}$ is a feature extractor to extract one or combination of the possible features listed above.

3.2.3 Computational Requirements

Note that we have to learn a set of function $\hat{F}$ where the $|\hat{F}|$ is a nontrivial number. Thus, learning or training time of an algorithm is important in solving the problem. Especially, if we do *online* learning where one constantly improves the learned functions $\hat{F}$, the learning time will be ever more critical. An algorithm with very high accuracy but very slow learning time is useless in that case.

Another concern is that we have to apply $\hat{F}$ multiple times. To get future uncertainty after K timesteps from the current state, we have to apply a subset $\hat{F}'$ of $\hat{F}$ for K times, which is total $O(|\hat{F}'| \times K \times C)$ times where C is the computation time of a model and $|\hat{F}'|$ is again nontrivial. C has to be small enough.

3.3 Experiments

In this section, we apply regression algorithms to the data to learn the prediction step of ensemble-based filtering. First, we introduce the concept of *normalization*, which is often important in practice. Then, we show the results of linear regression and nonlinear regression. Surprisingly, the best performance of linear regression is comparable to the nonlinear regression though the target functions are expected to be nonlinear. Furthermore, nonlinear regression is prone to overfit so that it performs poorer than linear regression in many cases. We speculate on the reason why nonlinear regression did not work and suggest a solution at the end.

3.3.1 Normalization

We first introduce normalization, which is to normalize each feature of input vectors to the $[-1, 1]$ range. Normalization is important in practice as prediction result often significantly differ depending on the normalization scheme used [1]. Normalizing features make each feature have the similar average magnitude so that they can be penalized fairly in regularization. Also, it renders the regression problem more numerically stable in many cases; for instance, the linear sum of features $\beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m$ is not dominated by a single feature x_i for some i, if features are normalized.

There are many ways to do normalization. One standard approach is fitting a normal distribution on the data and standardize data as in table 3.3.1. We are going to use the method in the normalization method in our experiments. One advantage of the method is that it is less sensitive to extreme values. Other popular normalization method choose the minimum and maximum value of a feature and scale the feature by: The method assumes that each feature is normally distributed. However, in many cases, the features are not exactly normally distributed and we may lose some information in the features through normalization step.

One possible remedy is to find a transformation that changes a original feature to a normally distributed variable. For instance, the distribution of a variance, $\Sigma_{(i,j)}$ where $i = j$, is originally more close to χ^2 distributed and it becomes close to normally distributed by log transformation as in Figure (3-2). Also, a covariance term, $\Sigma_{(i,j)}$ where $i \neq j$, is close to normally distributed after

1. Given features matrix X, $n \times d$ matrix, where n is the number of samples and d is the number of features, let $\mathbf{x}^i$ be the column vector of i-th feature of size n.

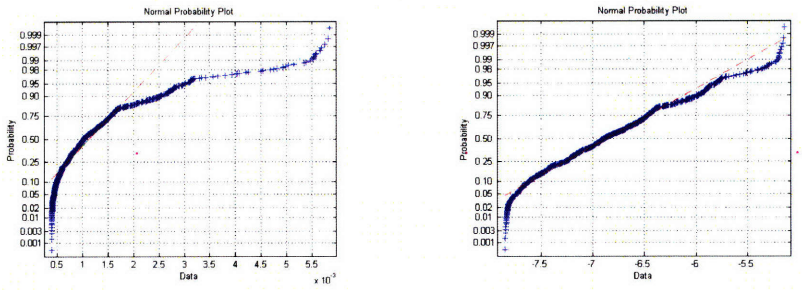
2. Fit a normal distribution for each feature $\mathbf{x}^i$: get

$$\hat{\mu}^i = \frac{1}{n}\sum_{j=1}^{n} x_j^i, \hat{\sigma}^i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j^i - \mu)^2}$$

3. Standardize each feature: $x_j^i = \frac{x_j^i - \mu^i}{\sigma^i}$ for all i and j

Table 3.1: Normalization through fitting a normal distribution

transformed into a correlation term as in Figure (3-3).

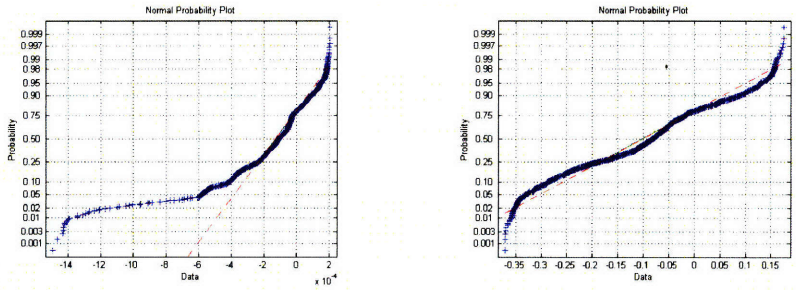


(a) Before log transformation (b) After log transformation

Figure 3-2: Normalization plot of empirical distribution of a variance (4000 samples) before/after log transformation. A distribution is more like normal distribution if the blue dots are more like a straight line.

It is not clear *a priori* how normalization would affect the prediction result and it has to be decided by experiments. In the table 3.2, we show the result of the prediction using the LSR and the RLS with different methods of normalization. In this thesis, we will use the Normalized Mean Squared Error (NMSE) as the error measure. Different size of spatial neighborhood and the quadratic basis expansion of original features for nonlinear regression are also tested. For the RLS, the best regularization parameter λ is chosen by 5-fold cross validation.

From the tables 3.2, we see that normalization method didn't change the result much except the normalization after transformation case, where the prediction performance was the worst. This is contrary to the many other problems where normalization changes the result of prediction sig-



(a) Before correlation transformation (b) After correlation transformation

Figure 3-3: Normalization plot of empirical distribution of a covariance (4000 samples) before/after correlation transformation. A distribution is more like normal distribution if the blue dots are more like a straight line.

	Features			
Model	Var(L=0)	Var(L=1)	q(Var(L=1))	Cov(L=1)
without normalization				
LSR	2.7E-3	3.1E-3	6.3E-3	5.5E-3
RLS	2.7E-3	3.1E-3	6.2E-3	5.5E-3
with normalization				
LSR	2.7E-3	3.1E-3	6.3E-3	5.5E-3
RLS	2.7E-3	3.1E-3	6.3E-3	5.5E-3
with normalization after transformation				
LSR	1.2E-1	1.5E-1	4.9E-2	2.1E-1
RLS	1.2E-1	1.5E-1	4.9E-2	2.1E-1

Table 3.2: Test error (NMSE) (3000 training samples and 1000 test samples) with various methods of normalization of features. L is the length of spatial neighborhood. $q(\mathbf{x})$ is the quadratic basis expansion of the original feature vector $\mathbf{x}$. The simple linear model using Var(L=0), $f(\Sigma_t^f(i,j)) = \beta_0 + \beta_1 \Sigma_{t-1}^a(i,j)$, performed the best regardless of the normalization method used. The normalization after transformation method performed poorly.

nificantly. It was true for all feature type tested in the table 3.2.

Normalization method using the transformation beforehand performed the worst. It may be due to the fact that the transformation itself is nonlinear; both correlation and log transformation nonlinearly transforms the features.

Due to the small effect of normalization, in following experiments, we do not report specific normalization method used and only report the best results.

Note also that the nonlinear regression using the quadratic expansion of original features did not perform better than linear regression. However, further experiments are needed to approve or disapprove the usefulness of nonlinear regression in our learning problem.

3.3.2 Nonlinear Regression Result

The original dynamics of weather system is nonlinear. In prediction step of ensemble-based filtering, each ensemble member is propagated through that nonlinear dynamics and we take the sample covariance of the ensemble as the forecast covariance. This results in nonlinear propagation of uncertainty. Thus, the propagation functions $\hat{F}$ are also expected to be nonlinear functions, which can be better learned by nonlinear regression methods.

In this section, we experiment with nonlinear regression with both implicit and explicit nonlinear features. For explicit nonlinear features, RLS was used. For implicit nonlinear features, kernel methods such as KRLS and SVM were tested.

		Features	
Model	Var(L=0)	Var(L=1)	Var(L=2)
RLS	2.7E-3 (3.9E-3)	3.1E-3 (1.9E-3)	7.5E-3 (1.3E-3)

		Features	
Model	q(Var(L=0))	q(Var(L=1))	q(Var(L=2))
RLS	2.7E-3 (3.9E-3)	6.3E-3 (5.8E-4)	9.3E-2 (2.8E-7)

		Features	
	q(CovR(L=0))	q(CovR(L=1))	q(CovR(L=2))
RLS	2.7E-3 (3.9E-3)	6.2E-3 (4.7E-4)	1.0E-1 (1.8E-7)

Baseline performance(simple linear regression): 2.7E-3 (3.9E-3)

Table 3.3: Test and training error of nonlinear regression with explicit higher-order features (NMSE). Train error is in the parenthesis. Test and training error of linear regression is shown on the top for comparison. $q(\mathbf{x})$ is the quadratic basis expansion of the original feature $\mathbf{x}$. The best generalization(test error) performance was the same for nonlinear and linear regression. The training error of nonlinear regression was much lower, implying the overfitting of training samples.

The prediction result with explicit nonlinear features is shown in the table 3.3. The regularization parameter for RLS was chosen through 5-fold cross validation of training samples. The original features $\mathbf{x}$ are expanded through quadratic basis expansion:

$$\mathbf{x} = [x_1, x_2, ..., x_d] \mapsto [x_1^2, x_2^2, ..., x_d^2, x_1x_2, x_1x_3, ..., x_1x_d, ..., x_{d-1}x_d] \tag{3.30}$$

Thus, RLS on the new features is quadratic function of the original features $\mathbf{x}$. We see that the

training error decrease significantly with more features; q(Var(L=2)) and q((CovR(L=2)) showed the lowest training error. However, the test error was worse than even the simplest linear model using Var(L=0); $f(\Sigma_t^f(i,j)) = \beta_0 + \beta_1 \Sigma_{t-1}^a(i,j)$. The simple linear model was more consistent in train and test error.

The prediction result with kernel methods is shown in the table 3.4. It was much slower than RLS in this case as $d \ll n$; RLS models takes $O(d)$ to evaluate and SVM or KRLS takes $O(n^2+nd)$ to evaluate. The trend is similar to the result in the table 3.3 that training error is smaller with the more number of features but test error is worse.

In sum, nonlinear regression clearly overfit the data and give worse generalization performance than simple linear regression. However, nonlinear regression fits the training samples better due to its increased modeling capacity; the training error of nonlinear regression was much lower than training error of linear regression.

So far, in terms of the generalization performance, the simplest linear model using Var(L=0) as the feature was the best. However, we would believe that the linear model would not be good enough for capturing the nonlinear propagation of uncertainty and there should be better models.

We speculate that there is *locality* or *nonstationarity* in the function we are learning. It means that the function is changing over the training samples and we cannot fit a single function on all training samples. In the next section, we experiment with *local* regression which only uses a few number of recent samples to learn a local model.

	Features		
Model	Var(L=0)	Var(L=1)	Var(L=2)
Gaussian SVM ($\sigma = 10^-3$)	1.4E-2 (4.9E-8)	7.5E-3 (3.6E-8)	9.9E-3 (2.2E-7)
Poly KRLS ($d = 3$)	2.7E-3	2.7E-3	4.8E-2
Baseline performance(simple linear regression): 2.7E-3			

Table 3.4: Nonlinear regression result(NMSE) with kernel methods. σ is the bandwidth of the Gaussian kernel. d is the degree of polynomial kernel. The best performance was the same for nonlinear and linear regression.

3.3.3 Local Regression Result

In 3.3.2, we applied nonlinear regression to fit a global function. In other words, it was assumed that one function is good enough to fit all training samples. Thus, we trained the function only once using all training samples and applied the same function to all test samples. There is another regression technique which is called *local* regression. It uses only a few number of recent samples to learn a local model. In our case, we will use a few number of recent, in terms of time, prediction update samples to learn the prediction update for the current timestep.

For instance, given a sequence of covariances

$$F(\Sigma^a_{t-W}) = \Sigma^f_{t-W+1}$$
$$F(\Sigma^a_{t-W+1}) = \Sigma^f_{t-W+2}$$
$$\cdots\cdots$$
$$F(\Sigma^a_{t-1}) = \Sigma^f_t,$$

where W is very small, we will use these samples as the train set to learn the prediction update for time t.

	Features		
Model	Var(L=0)	Var(L=1)	p(Var(L=1))
LSR (global)	2.7E-3	6.3E-3	9.3E-2
Local RLS (W=20)	5.8E-4	2.1E-7	2.5E-7

Table 3.5: Local regression results (NMSE): Only recent 20 samples are used to train a model. For every prediction, a new model was trained. Local model performed significantly better than global linear models.

The result of local regression was shown in the table 3.5. The local model learned through RLS with the small train set of size W=20 performed much better than the models learned with all training samples. The regularization parameter was chosen by learning and testing local models with different parameter in the train set. In the next chapter, we will discuss local models in much more detail, however, we will stop short of showing the fact here.

3.4 Summary

In this chapter, we formulated the problem of learning the prediction update of ensemble-based filtering. It was to learn the prediction update by learning propagation functions for each entry in the covariance matrix. The learned prediction update will be the flexible mean to evaluate the information gain of any path of any length, though we expect that the accuracy will decrease with the length of planning window, due to the accumulation of the errors in recursive prediction.

Considering spatial correlation of variables, we proposed to extract features from a spatial neighborhood of the covariance entry we are to predict. The correlation score was the initial evidence to show the validity of this approach. Then, we proposed to use nonlinear regression as we expected a nonlinear relationship of the input, the prior covariance, and the output, the posterior covariance. However, nonlinear models seriously overfit the train set as its generalization performance was worse than simple linear models.

At the end, we showed that the local regression worked better than global regression. In the next chapter, local learning will be described in detail and we will finally apply learned prediction update to path selection.

Chapter 4

Local Regression of Uncertainty Propagation: Cure for Nonstationarity

In Section (3.3), we formulated the learning problem of learning the prediction update, or equivalently propagation functions, of ensemble-based filtering.

Though the original prediction update nonlinearly propagates the uncertainty, nonlinear regression methods were not able to learn the propagation functions well to generalize to new samples, and performed worse than simple linear regression method. At the end of Section (3.3), we showed some evidence that *local* learning, using a few training samples, of the propagation functions would work better than global learning, using all training samples.

Local learning [6] is to deal with *nonstationarity*. Nonstationarity here means that the function, the mapping between features and labels, is changing over the samples. In global learning case, we assumed that there is a function f which satisfies

$$f(\mathbf{x}_i) = y_i + \epsilon_i, E[\epsilon_i^2] = \sigma^2, E[\epsilon_i \epsilon_j] = 0 \text{ for } i \neq j \tag{4.1}$$

with some small σ for all training samples $(\mathbf{x}_i, y_i)$. However, if the function is nonstationary, we basically have to assume f is different for all samples

$$f_{\mathbf{x}_i}(\mathbf{x}_i) = y_i + \epsilon_i, E[\epsilon_i^2] = \sigma^2, E[\epsilon_i \epsilon_j] = 0 \text{ for } i \neq j \tag{4.2}$$

The hope is that f is changing smoothly over the sample space that learning the function $f_{\mathbf{x}_i}$, with training samples similar to $\mathbf{x_i}$, is possible.

Note that the lack of good representations or features of the samples can be a possible reason for the nonstationarity of a function. In Section (3.3), the mean (the first moment) and the covariance (the second moment) of ensemble distributions were suggested as the features. However, an ensemble distribution can only be approximated by these two moments; the next state of the distribution also depends on other moments and ultimately individual ensemble members. As a result, two ensemble distributions with the same mean and covariance can still behave differently; $F(\Sigma_t^a) \neq F(\Sigma_{t'}^a), t \neq t'$ even if $\Sigma_t^a = \Sigma_{t'}^a$. We may attempt to use the higher moments of a ensemble distribution as the features. However, using more features would cause overfitting, require more training samples and is computationally expensive so we have to make a trade-off. Especially, if we were to use recursive predictions, features have to be predicted as well for a recursive prediction, thus using many features for a one-step prediction would not be desirable.

Some of the existing approaches of nonstationary regression will be introduced in Section (4.1). One of the distinct characteristics of our learning problem is that the training samples are temporally ordered; the filter generates the samples of the prediction update in temporal order. It will be explained that the sliding-window approach, which choose training samples from a temporal window around the current sample, is suitable for our purpose.

Especially, we introduce the sliding-window KRLS in Section (4.2) which is the combination of the KRLS and the sliding-window approach. We show that the sliding-window KRLS gives much better accuracy than global models in Section (3.3), in learning the propagation functions to give one-step and multi-step prediction of uncertainty propagation. It is also fast due to the small size of models.

Finally, in Section (4.3), we show the result of path selection using the learned propagation functions. It is able to reduce the uncertainty of a ROI significantly better than greedy methods and much faster than original ensemble-based filtering updates.

4.1 Nonstationary Regression Methods

Dealing with nonstationarity has been the topic of many research in physics and statistics, especially for predicting nonlinear chaotic system [5, 19, 29, 21]. Many machine learning researchers have found that local learning worked better than global learning in many cases such as the handwriting recognition task [6, 27]. The basic approach is to learn local models through utilizing only a fraction of training samples: the *local* training samples. Different approaches choose different samples as the local training samples. We will introduce two popular approaches in the following sections.

4.1.1 Sliding Window Approach

The sliding-window approach can be applied to data which has a temporal order. The local training samples are defined as the samples of recent past; a small size of temporal window is used to choose the training samples. It assumes that the target function is smoothly changing temporally.

Suppose that we have the training samples $X = (\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_t)$ and $\mathbf{y} = (y_1, y_2, ..y_t)$ where $(\mathbf{x}_i, y_i)$ is earlier generated sample than $(\mathbf{x}_j, y_j)$ if $i < j$. We have a new sample $\mathbf{x}_{t+1}$ and wish to predict y_{t+1}. To learn $f_t(\mathbf{x}_{t+1}) = y_{t+1}$, we use the temporal window of size W to select the training samples $X_t = (\mathbf{x}_{t-W+1}, ..., \mathbf{x}_t)$ and $\mathbf{y}_t = (y_{t-W+1}, ..., y_t)$. For the next new sample $\mathbf{x}_{t+2}$, the temporal window would be moved forward to include $(\mathbf{x}_{t+1}, y_{t+1})$ as a training sample.

By moving the window forward every time, the dynamic change of the target function can be approximately estimated. The computational advantage of the algorithm is that only recent samples has to be kept, assuming small size of temporal window. In addition, one can utilize *online* learning methods which allow fast update of a model when adding a new sample to the training set [45].

However, it has limitations. This approach can be applied only for the cases where the target function doesn't change significantly within the temporal window; The target function is assumed to be changing over time but still a single function is learned over the training samples. The window has to be small enough so that the dynamic change can be captured. On the other hand, if the window is too small, the regression problem become ill-posed; we may have more parameters to estimate than the number of training samples. We can partly solve this problem with regularization.

In sum, in applying the sliding-window approach, the assumption of the smoothness of the function change over time has to be valid and the size of temporal window has to be chosen carefully.

4.1.2 Locally Weighted Regression Approach

Another approach to local learning is the Locally Weighted Regression (LWR) [40]. In the LWR, the local training samples are chosen by k-nearest neighbor search, given some distance metric between samples. The local training samples are weighted according to the distance metric and weighted regression is done on the local training samples to learn the target function. We introduce the weighted version of the LSR and the RLS in the next sections.

The LWR generalizes the sliding-window approach as it can use any distance metric; the sliding-window approach used the temporal distance to choose the local training samples.

The LWR would work if we have samples densely located in the sample space. In other words, sufficient number of samples are needed to find good neighbors of the new sample that the prediction is made for. The training process involves the search of k-nearest neighbors, which can be computationally expensive. This search has to be done for every new models and a new model has to be trained on the new training samples. It is contrasted to the sliding-window approach where one can utilize some online learning algorithm to efficiently update the models. It also implies that all samples has to be kept in some database for a nearest neighbor search. With the sliding-window approach, we only need to keep a few recent samples.

In sum, the LWR is a general local learning technique which can deal with any distance metric between samples. However, it has computational disadvantages over the sliding-window approach due to its generality.

Weighted Regression Algorithms

In this section, the weighted version of regression algorithms are introduced. Applying these algorithms combined with a nearest neighbor search is the LWR.

Weighted Least Squares The optimization problem of the Weighted Least Squares (WLS) algorithm is given by

$$SSE(\beta, w) = \sum_{i=1}^{n} w_i (y_i - f(X_i))^2 \tag{4.3}$$

$$= \sum_{i=1}^{n} w_i (y_i - \beta_0 - \sum_{j=1}^{m} x_{ij}\beta_j)^2 \tag{4.4}$$

$$= (\mathbf{y} - X\beta)^T W (\mathbf{y} - X\beta) \tag{4.5}$$

where w_i is the weight of sample i and W is the diagonal matrix of weights.

The solution of WLS is given by

$$\hat{\beta} = (X^T W X)^{-1} X^T W \mathbf{y} \tag{4.6}$$

Weighted Regularized Least Squares From equation (4.6) and equation (3.20), it is easy to see that the solution of the Weighted Regularized Least Squares (WRLS) is given by

$$\hat{\beta} = (X^{\mathrm{T}} W X + \lambda I)^{-1} X^{\mathrm{T}} W \mathbf{y}. \tag{4.7}$$

4.1.3 Local RLS with Global Prior

We propose another possible solution to learn local models. It is to learn local models with a prior from global learning. Specifically, we use the RLS to get the model coefficients of the global model and use it as the prior instead of the zero-prior in fitting a local model with the RLS.

The algorithm can be used either with the LWR or the sliding-window approach. In both cases, the information from not only neighbors but all samples will be included in the regression problem, so that the solution is expected to be less sensitive to neighborhood size k or temporal window size W. In the next section, we derive the RLS algorithm with a nonzero prior.

Regularized Least Squares with Nonzero Prior

The RLS in equation (6.6) uses the prior $\bar{\beta} = \mathbf{0}$. The optimization problem for the case where $\bar{\beta} \neq \mathbf{0}$ is

$$\min_{\beta} \quad ||(X\beta - \mathbf{y})||^2 + \lambda(\beta - \bar{\beta})^T(\beta - \bar{\beta}) \tag{4.8}$$

If the covariance $\Sigma_{\bar{\beta}}$ of $\bar{\beta}$ is also available, we may minimize

$$\min_{\beta} \quad ||(X\beta - \mathbf{y})||^2 + \lambda(\beta - \bar{\beta})^T\Sigma_{\bar{\beta}}(\beta - \bar{\beta}) \tag{4.9}$$

To minimize (4.8), we set the derivative of (4.8) with respect to β to 0, and get

$$\hat{\beta} = (X^{\mathrm{T}}X + \lambda I)^{-1}(X^{\mathrm{T}}\mathbf{y} + \lambda\bar{\beta}). \tag{4.10}$$

To minimize (4.9), we set the derivative of (4.9) with respect to β to 0, and get

$$\hat{\beta} = (X^{\mathrm{T}}X + \lambda\Sigma_{\bar{\beta}})^{-1}(X^{\mathrm{T}}\mathbf{y} + \lambda\bar{\beta}). \tag{4.11}$$

Solving these problems are just as easy as solving the LSR or the RLS problems.

4.2 Sliding-window KRLS

As we explained in Section (4.1.1), the sliding-window approach has the computational advantage over the LWR in that neighbor search is not needed. The training samples are given sequentially from the filter in our case; we just get a new sample from the filter and discard the oldest sample to get the new training set.

It is possible to use the sliding-window approach with any learning algorithm. We introduce the sliding-window KRLS algorithm which uses the sliding-window approach with KRLS. KRLS has a few advantages over other methods for the sliding-window approach.

- Efficient nonlinear model: Due to the use of kernel, computation time of the sliding-window KRLS is proportional to the window size w, which is fixed regardless of the dimension of the re-

sulting model. In contrast, the running time of nonlinear models with explicit nonlinear features is proportional to the features size, which can increase fast with the dimension of the model.

- Exact incremental update: An exact solution for incremental update with a new training sample is available. Other kernel methods such as SVM does not have this property.

In the following section, the algorithm for exact incremental update is described. Then, a technique to further improve the speed of the sliding-window KRLS will be introduced. These two algorithms will make the sliding-window KRLS very suitable for learning uncertainty propagation and real-time informative path planning.

4.2.1 Incremental Update of KRLS

In the KRLS, exact update of the models with a new training sample is possible, which can save the training time in updating models. Here, we assume the case where the oldest training sample is discarded thus fixing the training set size at all time. From the KRLS solution in Section (3.1.2), we see that the inverse of the kernel matrix is required to make a prediction. In other words, the training process is essentially to build the kernel matrix and get the inverse of the kernel matrix. Let the training set at time t be

$$X_t = (\mathbf{x}_{t-W+1}, \mathbf{x}_{t-W+2}, ..., \mathbf{x}_t), \mathbf{y}_t = (y_{t-W+1}, y_{t-W+2}, ..., y_t) \tag{4.12}$$

and the new training set at time $t+1$ be

$$X_{t+1} = (\mathbf{x}_{t-W+2}, \mathbf{x}_{t-W+3}, ..., \mathbf{x}_{t+1}), \mathbf{y}_t = (y_{t-W+2}, y_{t-W+3}, ..., y_{t+1}) \tag{4.13}$$

Given a (regularized) kernel matrix K_t of the samples X_t and its inverse matrix K_t^{-1}, we can get the inverse of the new kernel matrix K_{t+1} from K_t^{-1} in $O(W^2)$, compared to $O(W^3)$ required to perform an inversion of K_{t+1} directly without K_t^{-1} [46]. Thus, the initial training takes $O(W^3)$

but the update takes $O(W^2)$. The algorithm is described below.

$$K_t = \begin{pmatrix} a & \mathbf{b} \\ \mathbf{b}^{\mathrm{T}} & C \end{pmatrix} \quad K_t^{-1} = \begin{pmatrix} f & \mathbf{g} \\ \mathbf{g}^{\mathrm{T}} & H \end{pmatrix}$$

$$K_{t+1} = \begin{pmatrix} C & \mathbf{d} \\ \mathbf{d}^{\mathrm{T}} & e \end{pmatrix} \tag{4.14}$$

Then, the inverse of the new kernel matrix K_{t+1} is given by:

$$K_{t+1}^{-1} = \begin{pmatrix} H(I + \mathbf{d}\mathbf{d}^{\mathrm{T}}Hg) & -H\mathbf{d}g \\ -(H\mathbf{d})^{\mathrm{T}}g & g \end{pmatrix} \tag{4.15}$$

$$g = (e - \mathbf{d}^{\mathrm{T}}H\mathbf{d})^{-1}$$

4.2.2 Faster Kernel Matrix Building

Examining the set of features in Section 3.2.2, neighboring learners $\hat{f}_{(l_1,l_2)}$ and $\hat{f}_{(l_1,l_2+1)}$ will clearly share features, and in fact many models share features. Let a training sample $\mathbf{x}_i = \Sigma_i^a$ and $\mathbf{x}_i(l) = \Sigma_i^a(l, \mathbf{N}_l)$. Then, an entry of the kernel matrix for $\hat{f}_{(l_1,l_2)}$ with kernel function k is

$$k(\mathbf{x}_i, \mathbf{x}_j) = k([\mathbf{x}_i(l_1); \mathbf{x}_i(l_2)], [\mathbf{x}_j(l_1); \mathbf{x}_j(l_2)])$$

If k is a polynomial kernel function,

$$\begin{aligned} & k([\mathbf{x}_i(l_1); \mathbf{x}_i(l_2)], [\mathbf{x}_j(l_1); \mathbf{x}_j(l_2)]) \\ &= ([\mathbf{x}_i(l_1); \mathbf{x}_i(l_2)]^{\mathrm{T}}[\mathbf{x}_j(l_1); \mathbf{x}_j(l_2)] + a)^d \\ &= (\mathbf{x}_i(l_1)^{\mathrm{T}}\mathbf{x}_j(l_1) + \mathbf{x}_i(l_2)^{\mathrm{T}}\mathbf{x}_j(l_2) + a)^d \\ &= (k_{l_1}(\mathbf{x}_i, \mathbf{x}_j) + k_{l_2}(\mathbf{x}_i, \mathbf{x}_j) + a)^d \end{aligned}$$

where $k_{l_1}(\mathbf{x}) = \mathbf{x}(l_1)^{\mathrm{T}}\mathbf{x}(l_1)$. Similarly, the kernel matrix entry for $\hat{f}_{(l_1,l_2+1)}$ is

$$(k_{l_1}(\mathbf{x}_i, \mathbf{x}_j) + k_{(l_2+1)}(\mathbf{x}_i, \mathbf{x}_j) + a)^d$$

As a result, $k_{l_1}(\mathbf{x}_i, \mathbf{x}_j)$ does not need to be calculated again. We assume that a kernel evaluation takes $O(m)$, where m is the number of the features. Then building a kernel matrix takes $O(W^2m)$, which is significant as $W \leq m$ in our case. In recursive prediction, one has to build additional models to predict features and need to build kernel matrices. So, we build *partial* kernel matrices K_l for each k_l separately, which takes $O(W^2\frac{m}{2})$, and add the matrices when needed, which takes $O(W^2)$. Suppose that we use the all covariance elements as features with a polynomial kernel. In an r-step recursion tree with branching factor m, a naive algorithm would take $O(W^2m\sum_{i=1}^{r} m^i)$, as a new kernel matrix would be built at every node except the root. Sharing of *partial* kernel matrices take $O(\frac{1}{2}W^2(m+1)\sum_{i=1}^{r} m^i)$. So, we build matrices 2 times faster, and it is actually faster than that as neighbors of grids overlap and we need to build fewer *partial* kernel matrices.

4.2.3 Learned Model Accuracy

In this section, we compare the predicted covariance from the sliding window KRLS with the full EnKF covariance. Figure 4-1 shows contiguous one-step prediction of covariance between two neighboring grid cells. The sliding-window KRLS prediction at time t is made with the model trained with the training set in the sliding window of size $W = 10$; $X_t = (\mathbf{x}_{t-W}, ..., \mathbf{x}_{t-1}), \mathbf{y}_t = (y_{t-W}, ..., y_{t-1})$. The model was trained again at every timestep with the new training set. As a comparison, we also trained an (global) SVM with 500 samples (t=-499,...,0), before making predictions at t=1, the first sample at Figure (4-1).

Both models used a polynomial kernel of degree 3. The top figure shows the covariance of two grid cells from the full EnKF model, the KRLS-predicted covariance and the SVM-predicted covariance. The lower figure shows the error between each model and the EnKF covariance. The result shows that sliding-window approach achieved very low prediction error compared to global SVM predictor.

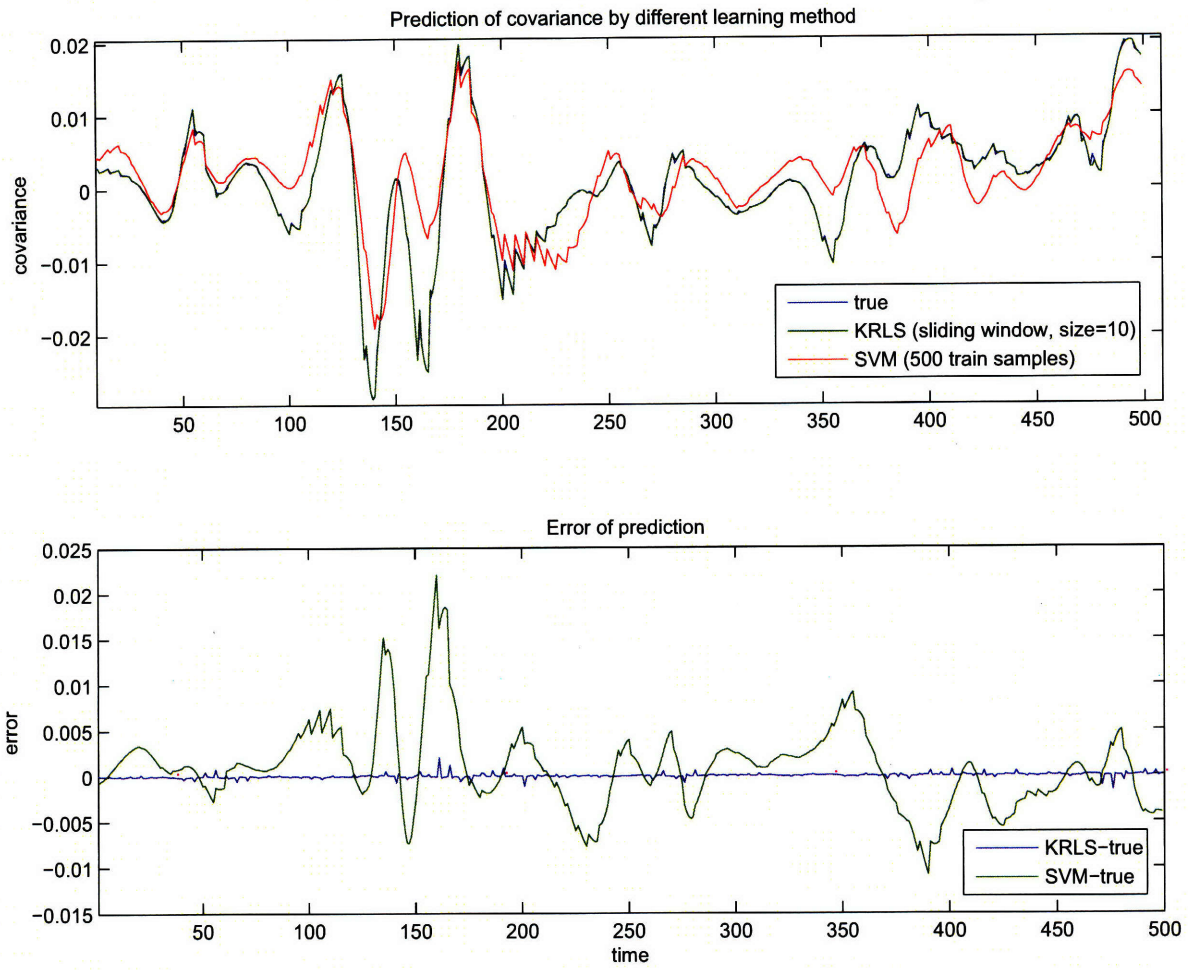


Figure 4-1: One-step prediction accuracy of a single covariance entry. An SVM model was trained with 500 timestep samples of the time series before prediction time. A KRLS model was trained at every timestep with sliding-window of size 10. Both methods used a polynomial kernel of degree 3. The top graph shows predictions and the ground truth from the full EnKF. The bottom graph shows the difference between the learned models and the ground truth.

Table 4.1 shows the recursive prediction performance using different kernels and features with the KRLS algorithm in the Lorenz 2003 models. The ground truth was the result of a full EnKF prediction update executed for 5 timesteps. The prediction results for the learned models were the result of using the predictor recursively 5 times. The nonlinear kernels were significantly better ($3 \sim 10$ times) than linear kernels in accuracy represented by the mean absolute error (MAE). Using all covariance elements (CovB) as features provide a slight reduction in error. It is though slower than the case using the features CovR as there are more entries to predict in recursive predictions.

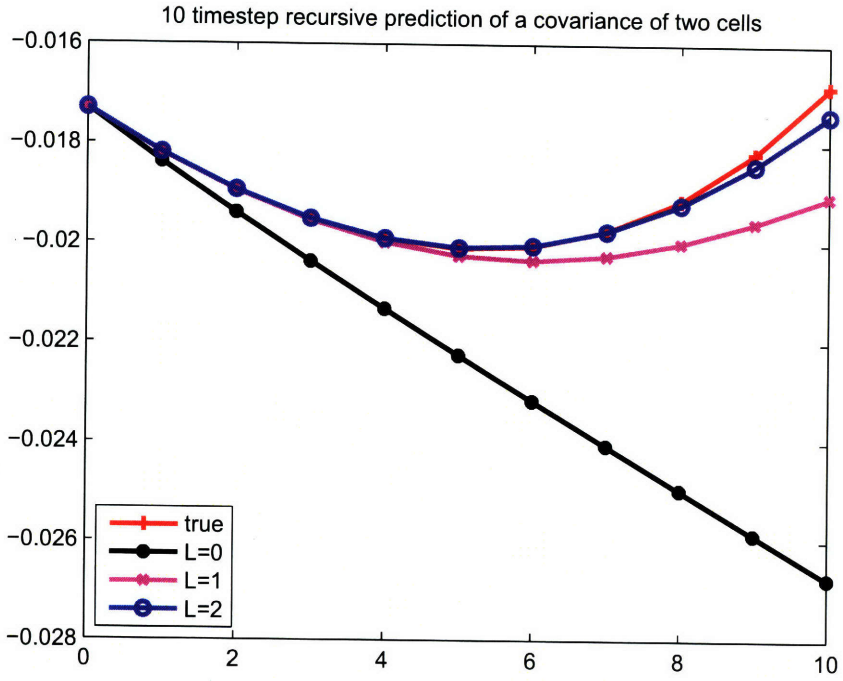


Figure 4-2: Recursive prediction with different size of spatial neighborhood. It shows that using and predicting uncertainty of bigger neighborhood gave better prediction for long time steps.

As a final validation, we also learned a model of the ensemble mean, although the mean value itself is not relevant in predicting information gain and choosing sensor trajectories. The results of the mean prediction performance are given in 4.2, and again show that the best prediction was achieved using sliding window regression.

The effect of spatial neighborhood size L is shown in Figure (4-2). With larger spatial neighborhood, the model performed better in recursive prediction. However, it is also more computationally demanding as it needs more features to be predicted. A trade-off has to be made between the accuracy and speed in this case.

4.3 Experimental Results of Path Selection

Given the ability to predict the change in covariance of the EnKF efficiently, we can now use the learned function to identify sensor trajectories that maximize information gain of some region of interest. We define the region of interest as a set of grids $R = \{g_i | g_i \in [1, N_s], i \in [1, |R|]\}$. The

	Features	
Kernel	CovR	CovB
Linear (W=20)	0.109	0.0764
Poly (d=3,W=10)	**0.0252**	0.0222
Poly (d=3,W=20)	0.0261	**0.0181**
Poly (d=5,W=20)	0.0746	0.0448

Table 4.1: Mean absolute error (10^{-2}) over 100 test samples with Lorenz-2003 model. Baseline was the EnKF with 1000 ensemble members. Each model recursively predicted the trace of covariance after 5 forecast updates. W is sliding-window size, d is the degree of polynomial kernel. The feature sets were covariance rows and blocks respectively, of neighbors in distance 1

	Features		
Model	Auto	Mean	p(Mean)
SVM(Gaussian)	0.047	0.084	0.13
SVM(Linear)	0.012	0.14	0.038
RLS(W=15)	0.0045	0.0004	**0.00004**
Propagation of ensemble mean: 0.0055			
Propagation of 1/4 ensembles: 0.00035			

Table 4.2: Normalized mean squared error(NMSE) of predicting the ensemble mean $\bar{\mathbf{x}}(i)$. The ground truth is from the full EnKF. 500 training samples were used for SVM. W is sliding-window size. Auto: 10-timestep time series of $\bar{\mathbf{x}}(i)$, Mean: the ensemble mean of neighbors $\bar{\mathbf{x}}(N_i)$, p(Mean): explicit product features of $\bar{\mathbf{x}}(N_i)$, which is similar to use polynomial kernel of degree 2

measure of information gain is the change in the trace of the R sub-block of the covariance $\Sigma(R)$.

Without loss of generality, we consider the case where R is 5x5 region and the planning horizon K is 10. We assumed that an agent moves from one cell to other cell in every 2 timesteps. At the first timestep at the cell, the agent takes an measurement of the cell. Thus, the agent takes total 5 measurements in the ROI R within the planning horizon $K = 10$. The specific choice was made by the fact that this planning problem was a part of the multi-level planning scheme, where 5x5 region was chosen by a high-level planner and assigned to a local planner to optimize the local path in the region.

A sample scenario of path planning in that case is given in figure 4-3. Given the initial state of the EnKF, an agent plans to observe a sequence of 5 locations, one for every two timesteps, in the region of interest before arriving at the end point. We fixed the start point and there were three choices of end point, thus there were a total of 51 choices of paths in this case. The execution result of the best and worst path is given in figure 4-4. The best path decreased the uncertainty of

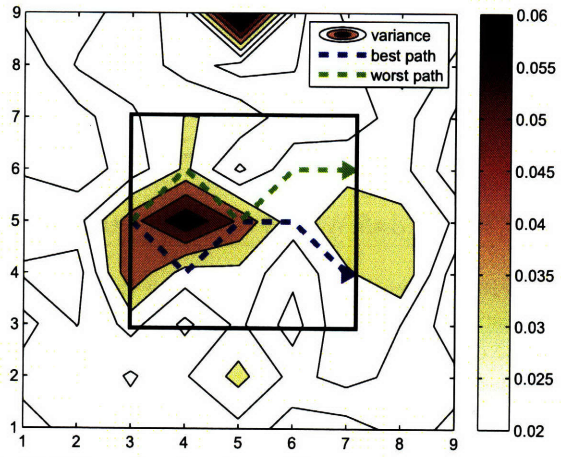


Figure 4-3: Original state of the EnKF at planning time. The best path and worst path are shown. The rectangle represent the local region of interest that we plan to minimize the uncertainty (trace of covariance).

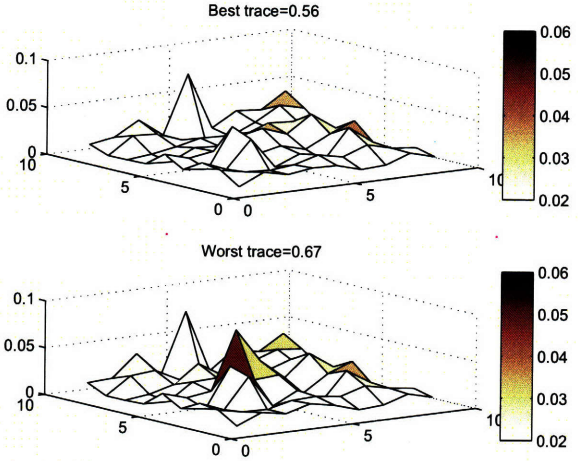


Figure 4-4: The variance of the EnKF after path execution. The best path decreased the uncertainty of the region by 15% while the worst path increased it by 1.5%. The average reduction of 51 paths was 8.3% with standard deviation of 4.6%.

the region by 15% while the worst path actually increased the uncertainty 1.5%. The average reduction was 8.3% with standard deviation of 4.6% for 51 paths. It shows that targeting informative locations is critical for the goal of uncertainty reduction.

The simple algorithm given in Algorithm 1 tells how to select the path with the learned propagation functions. The path selected by the KRLS predictions was compared with a baseline greedy strategy. The greedy strategy chooses a path with most uncertainty at *current* time t; this strategy is quite common in practice [23]. In other words, it chooses path p_i with largest $\mathrm{tr}(\Sigma_t(P_i))$ where P_i is the set of grid locations that p_i visits. While greedy strategies perform reasonably in the general

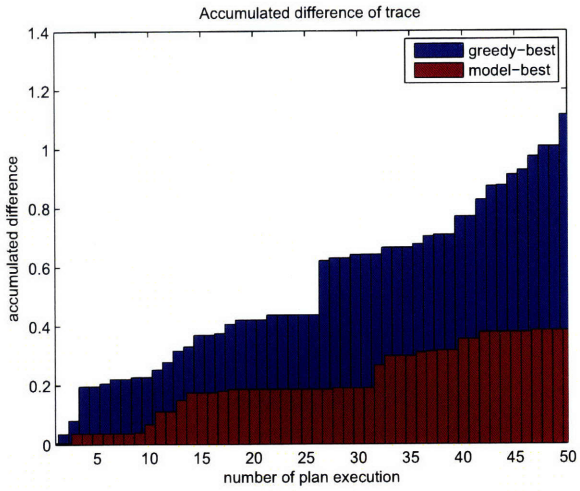


Figure 4-5: Accumulated difference of trace between the best and model/greedy for 50 trials of 5 x 2-timestep planning in Lorenz-2003 model

case, our prediction model allows us to evaluate the effect of sensing into the future and capture spatio-temporal effects.

As shown in figure 4-5, the performance gap between greedy and model planning increased as we executed more plans.

4.3.1 Computation Time

The error between the predicted covariance from the learned model and full EnKF propagation is substantially compensated by the computational advantage as in Table 4.3.

In particular, the computation time of full EnKF propagation scales with the model size, which is proportional to the number of grids N_s and the size of the ensemble N_e, as given in Section (2.2.3). The path evaluation in the Lorenz 2003 model incurs a substantial cost because the complexity of this model requires a large ensemble set.

In contrast, the computation time of the model prediction only depends on the size of neighborhood and sliding-window size and is independent of the model size, so that the computation time in both the Lorenz-95 and Lorenz-2003 models was approximately equivalent. Our approach actually is expected to improve with large ensemble size as the initial estimate becomes more accurate and stable (smooth) [22], thus providing less noisy training samples.

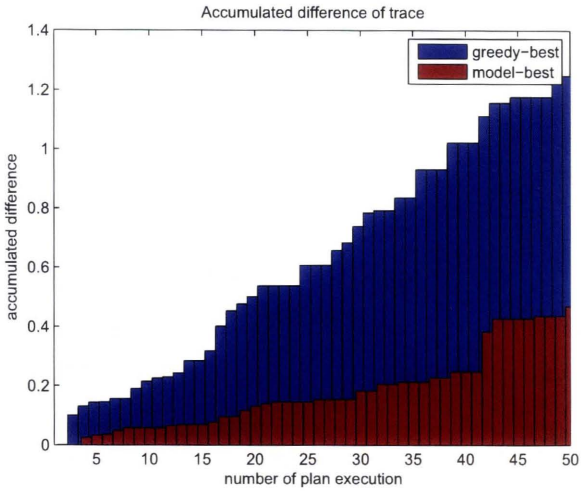


Figure 4-6: Accumulated difference of trace between the best and model/greedy for 50 trials of 5 x 2-timestep planning in Lorenz-95 model

	Lorenz-95 (N_e=400)	Lorenz-2003 (N_e=1000)
Full Propagation	13.42s	1984.48s
KRLS Prediction	12.65s	14.97s

Table 4.3: Average computation time for path planning in two weather models. A total of 51 paths were evaluated using full propagation of the EnKF and the KRLS prediction. The computation time of the KRLS prediction planning was independent of the chosen weather model, while the EnKF computation time increases severely as the model size grows. (Pentium-4 3.0 Ghz Dual Core was used.)

4.4 Conclusions

We showed that the sliding-window KRLS algorithm successfully approximates the nonlinear evolution of covariance in the EnKF for multiple timesteps, and makes fast informative path selection possible.

The learner trains on covariance samples generated by the EnKF during the standard prediction update, but does not require additional EnKF updates at the planning time. The efficiency of the KRLS prediction allows a large set of candidate trajectories to be considered quickly. The approach scales independently of actual model size, which makes it possible to use this approach in large models. It is expected to perform even better with large ensemble size of the EnKF, which is limited by computational resources but expected to increase continually.

However, there is an assumption to be hold for this method to work. Using the sliding-window

approach is to assume that the learned propagation functions will be valid in the planning horizon. Due to the nonstationarity, this assumption will break with longer planning horizon. The predict and measurement updates could change the trajectory of uncertainty significantly after a few iterations and break the assumption. In addition, the prediction error will be accumulated in recursive predictions. The current approach does not deal with this problem explicitly.

In the next chapter, we will try to solve the first problem by building models which is valid for longer timesteps. In the future, we want to deal with the propagation of errors explicitly. There is an approach in Gaussian Process(GP) framework to deal with this propagation of error [11]. We would like to apply similar concept to our problem in the future.

Chapter 5

Improving Generalization with Time-series Features

5.1 Overview

In Section (4), we were able to learn the models of uncertainty propagation with good accuracy using the sliding-window KRLS. The models enable fast selection of informative path which was significantly better than the path selected by a greedy algorithm.

The fundamental assumption in the approach was that the local models are accurate enough within the planning window. However, we note that the measurement update could change the uncertainty greatly in some cases, breaking the assumption; the new sample may differ too much from the samples in the sliding-window or the local training samples that we may need a new local model after a measurement update. The problem is that we do not have the training samples to learn the new local model in the sliding-window. The efficiency of the sliding-window KRLS approach was from the fact that it kept only small number of samples. The LWR may be suitable for resolving this issue but it is computationally expensive as we have to retrieve the local training samples from a database of all past samples.

Here, we pay more attention to the fact that we can treat the data as time-series. In the previous sections, we tried to fit functions with single timestep features extracted from a spatial neighbor-

hood:

$$f(\mathbf{E}_{(i,j)}(\Sigma_{t-1})) = \Sigma_t(i,j)$$

where $\mathbf{E}_{(i,j)}$ is the feature extractor for learning $\Sigma(i,j)$. However, it is also possible to fit functions using time-series features of degree k:

$$f(\mathbf{E'}_{(i,j)}(\Sigma_{(t-k):t-1})) = \Sigma_t(i,j)$$

where $\Sigma_{t_1:t_2} = \{\Sigma_{t_1}, \Sigma_{t_1+1}, ..., \Sigma_{t_2}\}$ and $\mathbf{E'}_{(i,j)}$ is the feature extractor that extract features from a time-series $\Sigma_{t_1:t_2}$ instead of a single Σ for learning $\Sigma(i,j)$. The latter is called *time-series regression* and has been used for many signal processing problems where the data is a stream of signals [8]. The first case is the special case of time-series regression with $k = 1$. Note that there are two types of covariances at time t; forecast covariance Σ_t^f and analysis covariance Σ_t^a. We will deal with this issue later in Section (5.3). Figure (5-1) shows why time-series regression with $k > 1$

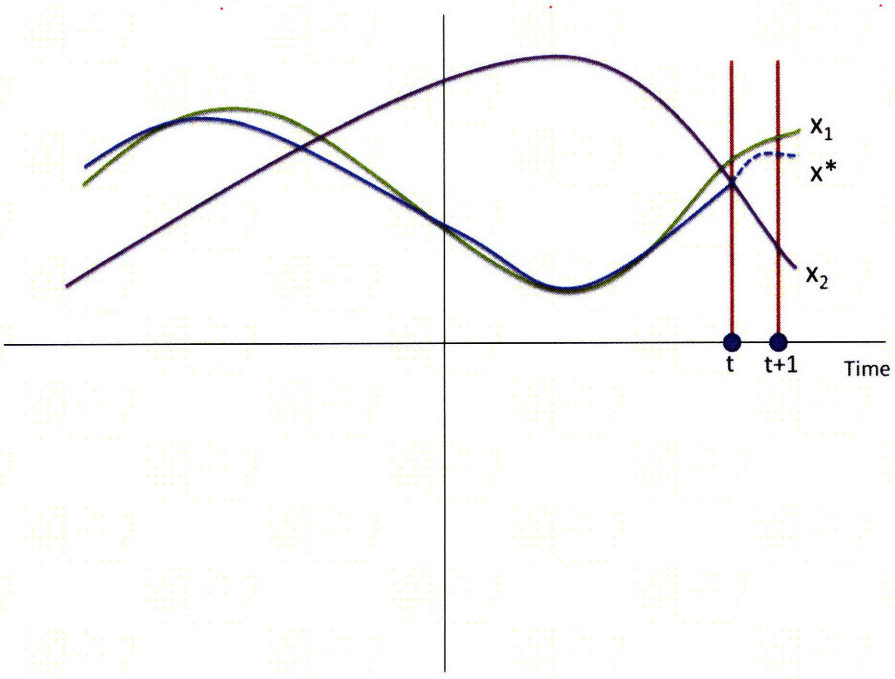


Figure 5-1: The possible benefit of time-series regression. Three artificial time-series are shown. x_2 and x^* are close at one-point of time-series at t. However, if we look at the whole trajectory instead of a point, x_1 is closer to x^*. To better predict x^* at time $t+1$, we can look at x_1 instead of x_2.

may work better than $k = 1$. By looking at a longer *profile*, one can get a better representation of the time-series. Suppose we want to make a prediction for the time-series x^* in Figure (5-1) at time $t + 1$ by looking at the value of x^* up to time t and training samples that we have observed from the source of the time-series; and we have only x_1 and x_2 as training samples. If we want to use a single neighbor to make the prediction, we should choose x_1 over x_2 as the nearest neighbor.

In this chapter, we attempt to improve the generalization of the models of uncertainty propagation using higher-order time-series features. In Section (5.2), the classical models of time-series will be introduced and further motivate the use of time-series regression. In Section (5.3), we will discuss ways to model the evolution of covariances as a time-series. In Section (5.4), the result of learning global models using time-series features will be presented, which we expect to perform better than global models learned using a single timestep features. In Section (5.5), we will also attempt to improve the result of local regression using time-series features. Finally, the result of path selection using the models will be presented, and we show that it can achieve significant improvement in terms of accuracy and computation time over other methods.

5.2 Classic Linear Time-series Models

As there is a large literature in time-series regression, it may be beneficial to look at previous works first. [7] provides a good reference for time-series modeling.

Autoregressive(AR) model

Definition (AR(d) model): The autoregressive(AR) model of an order d is written as AR(d) and defined by

$$x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + ...\beta_d x_{t-d} + w_t$$

where w_t is a purely random process where $E[w_t] = 0, E[w_t w_{t'}] = 0$ and $E[w_t w_t] = \sigma^2$.

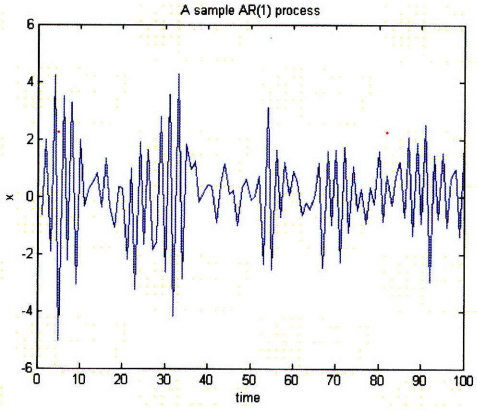
Once we have the model, we can generate a one-step forecast by:

$$\hat{x}_t = \beta^T \mathbf{x}_{t-1} \tag{5.1}$$

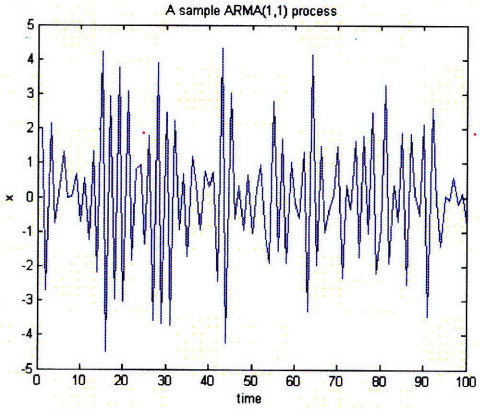
where $\mathbf{x}_{t-1} = [x_{t-1}; x_{t-2}; ...; x_{t-d}]$. The name *autoregressive* comes from the fact that the value of x at time t is regressed on past values of itself; x at time $t-1$ and so on. w_t term can be regarded as prediction error term, which is modeled as a random process.

If the function $\beta^T \mathbf{x}$ models the time-series well so that there is no more structure to be modeled in the data,the error term w_t at time t would appear as white noise; it will not be correlated with the other error terms $w'_t, t \neq t'$. A sample time-series of AR(1) model is shown in Figure (5-2(a)).

Taken's Theorem Using AR to model a time series may be justified by Taken's theorem [41]. Taken's theorem tells us that we can reconstruct a dynamical system of many state variables using the time series of a state variable (or a scalar measurement function of the variables) of the system, which is what AR does.



(a) AR(1) model $x_t = -0.8x_{t-1} + w_t$



(b) ARMA(1,1) model $x_t = -0.8x_{t-1} - 0.5w_{t-1} + w_t$

Figure 5-2: Sample time-series of AR and ARMA. A simple linear function generates seemingly complicated time-series.

Autoregressive moving average model(ARMA)

Definition (ARMA(p,q) model): The autoregressive moving average model of an order (p,q) is written as ARMA(p,q) and defined by

$$x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + ...\beta_p x_{t-p} + \alpha_1 w_{t-1} + \alpha_2 w_{t-2} + ...\alpha_q w_{t-q} + w_t \tag{5.2}$$

$$= \beta^T \mathbf{x}_{t-1} + \alpha^T \mathbf{w}_{t-1} + w_t \tag{5.3}$$

where β is a $p \times 1$ vector and α is a $q \times 1$ vector.

The autoregressive-moving average (ARMA) model has the moving average (MA) term $\alpha^T \mathbf{w}$. Note that AR is a special case of ARMA with $q = 0$. The literature suggests that ARMA is appropriate when a system is a function of its own behavior and a series of unobserved factors, which is modeled by the MA part. For example, stock prices may be modeled by observed prices and unobserved effects of activities of market participants. A sample time-series of ARMA(1,1) is shown in Figure (5-2(b)).

Once we have the model, we can generate a one-step forecast by:

$$\hat{x}_{t+1} = \beta^T \mathbf{x}_t + \alpha^T \mathbf{w}_t \tag{5.4}$$

where we can get w_t by the prediction error at time t: $w_t = x_t - \hat{x}_t$. Note that we can only make a one-step prediction with this model as we need the last prediction error, though we may treat the errors as missing values to make longer step predictions.

A stochastic version of ARMA is introduced in [42]. It treats observations x_t as a random variable instead of a fixed value. The stochastic version provides a smoothing technique that produces more accurate estimates.

Autoregressive integrated moving average model(ARIMA)

ARIMA extends ARMA by adding difference of values of a time-series at two or more time lags as features. For instance, $\Delta x_t = x_t - x_{t-1}$ or $\Delta^2 x_t = (x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$ are

added as features. It was shown that the difference operation tends to render time-series stationary in many cases. An evidence is shown in Figure (5-3) that higher-order difference operation makes the time-series appear more structured and predictable. However, we note that the class of linear

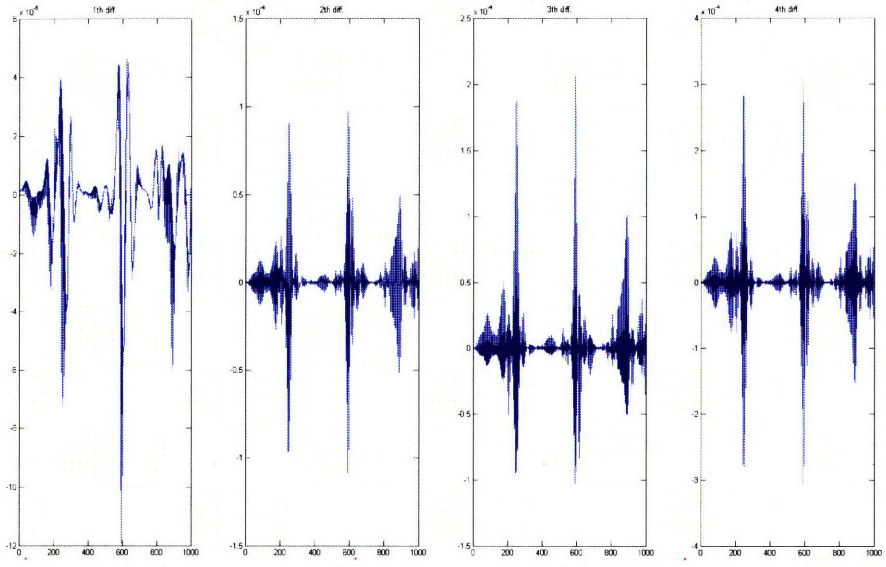


Figure 5-3: The d-th order finite differences of covariance time-series. As the order becomes higher, the time-series appears model predictable.

.

functions which can be learned with ARIMA is the same as the ones with ARMA. The difference operation and the features generated does not add modeling capacity to linear models of the data. It can be proved in a way similar to theorem 5.3.1. Still, the fact that it has worked well in practice suggests that the difference operation is helpful in extracting a good feature set.

5.3 Time-series Modeling of Uncertainty Propagation

To use time-series regression in our problem, we have to model the evolution of covariance as a time-series. It was mentioned that there are two kinds of covariances at time t in our setting. One is from the prediction update Σ_t^f, and the other is from the measurement update Σ_t^a. To deal with

this, we may simply extend the definition of $\Sigma_{t:t'}$ to:

$$\mathbf{F}_1 : \Sigma_{t:t'} = \{\Sigma_t^f, \Sigma_t^a, \Sigma_{t+1}^f, \Sigma_{t+1}^a ..., \Sigma_{t'}^f, \Sigma_{t'}^a\} \tag{5.5}$$

That is not the only choice. We can also use:

$$\mathbf{F}_2 : \Sigma_{t:t'} = \{\Sigma_t^f, \Sigma_t^a - \Sigma_t^f, \Sigma_{t+1}^f, \Sigma_{t+1}^a - \Sigma_{t+1}^f ..., \Sigma_{t'}^f, \Sigma_{t'}^a - \Sigma_{t'}^f\} \tag{5.6}$$

$$= \{\Sigma_t^f, u_t, \Sigma_{t+1}^f, u_{t+1} ..., \Sigma_{t'}^f, u_{t'}\} \tag{5.7}$$

$\mathbf{F}_2$ models the change of Σ_t through the measurement update at time t as exogenous control input u_t. Note that in propagating uncertainty in ensemble-based filtering, the prediction update always changes the values of covariances but the measurement update does not. It only changes the covariances when new measurements are available. Thus, it may be natural to model the measurement update as control input to autoregressive time-series of covariances, which is changed through the prediction update at every timestep.

Actually, the set of linear models using $\mathbf{F}_1$ is the same as the set of linear models using $\mathbf{F}_2$. In other words, one can model any linear model that uses $\mathbf{F}_2$ by using $\mathbf{F}_1$ instead, and vice versa.

Theorem 5.3.1 *(Equivalence of linear models of* $\mathbf{F}_1$ *and* $\mathbf{F}_2$*): The set of linear models of* $\mathbf{F}_1$ *are the same as the set of linear models of* $\mathbf{F}_2$*.*

Proof: Consider the case $k = 0$. A linear model of $\mathbf{F}_1$ is

$$\beta_{10} + \beta_{11}\Sigma_t^f + \beta_{12}\Sigma_t^a$$

A linear model of $\mathbf{F}_2$ is

$$\beta_{20} + \beta_{21}\Sigma_t^f + \beta_{22}(\Sigma_t^a - \Sigma_t^f)$$

Given the vector β_1, we can choose $\beta_{21} = (\beta_{11} + \beta_{12})$ and $\beta_{22} = \beta_{12}$ such that the two models become identical. We can generalize this argument to $k > 0$ case easily. ∎

Thus, it may appear that the choice of $\mathbf{F}_1$ and $\mathbf{F}_2$ does not matter in our problem. However, in the nonlinear case, the choice of F_1 and F_2 could make a difference. Suppose there are two

samples of the time-series $\Sigma_{1:2}$ and $\Sigma_{3:4}$ and there were no measurements at times $t = 1, ..., 4$. Using $\mathbf{F}_1$, the features will be

$$\Sigma_{1:2} = \{\Sigma_1^f, \Sigma_1^a, \Sigma_2^f, \Sigma_2^a\} = \{\Sigma_1^f, \Sigma_1^f, \Sigma_2^f, \Sigma_2^f\}$$

$$\Sigma_{3:4} = \{\Sigma_3^f, \Sigma_3^a, \Sigma_4^f, \Sigma_4^a\} = \{\Sigma_3^f, \Sigma_3^f, \Sigma_4^f, \Sigma_4^f\}$$

Using $\mathbf{F}_2$, the features will be

$$\Sigma_{1:2} = \{\Sigma_1^f, 0, \Sigma_2^f, 0\}$$

$$\Sigma_{3:4} = \{\Sigma_3^f, 0, \Sigma_4^f, 0\}$$

In nonlinear regression the interaction between features plays an important role, so that the difference between 0 and the nonzero values will make a difference. In addition, the second and the fourth of the four features are the same value 0 for $\Sigma_{1:2}$ and $\Sigma_{3:4}$ using $\mathbf{F}_2$, but have different values using $\mathbf{F}_1$. So, we may have to try different choice of features in a nonlinear regression case.

5.4 Learning Global Models with Time-series Features

In Section (3.3.2), we showed that learning global models of uncertainty propagation using all training samples and single timestep features was unsuccessful; a simple linear model performed the best and nonlinear models overfit the data.

In this section, we experiment with time-series features for learning global models. we have explained the possible benefit of time-series features in representing the covariance evolution. The expectation is that the time-series features combined with nonlinear regression will enable better learning of uncertainty propagation.

Table (5.2) shows the prediction performance of global models using time-series features. On the top of the table, the linear models of quadratic and cubic basis expansion of original features were tested along with the linear model of the original features. The result shows that the prediction error decreases with the degree of time-series features, proving the usefulness of time-series features.

We also compared the result with local regression with one timestep features. The sliding-window KRLS approach was used as in the last chapter. It is shown that global models with time-series features perform better than the local models.

The application of higher-order time-series of spatial features with $L > 0$ performed better than with $L = 0$ in linear RLS. Its quadratic and cubic basis expansions were not tested. Note that quadratic and cubic basis expansions quickly increases the number of features; quadratic basis expansion increases the number of features d to $\begin{pmatrix} d \\ 2 \end{pmatrix}$ and cubic basis expansion increases it to $\begin{pmatrix} d \\ 3 \end{pmatrix}$. Due to the increased number of features, the regression algorithms may run into computational issue. In this case, partial basis expansion which limits the choices may have to be used instead.

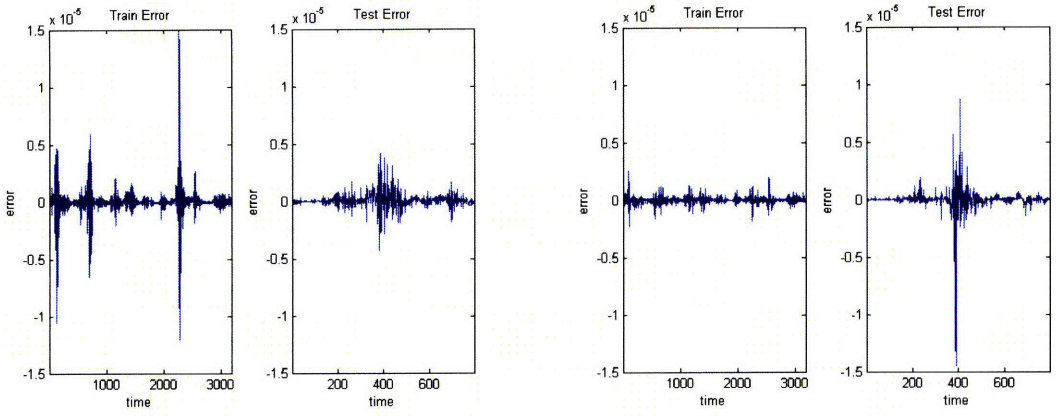
Model	Degree of time-series 1	5	10
	Features: Var(L=0)		
RLS	2.8E-3 $\pm$ 4.6E-3	3.2E-6 $\pm$ 1.3E-5	5.4E-7 $\pm$ 2.6E-6
RLS(quadratic)	2.8E-3 $\pm$ 5.1E-3	9.2E-7 $\pm$ 4.0E-6	2.3E-7 $\pm$ 1.1E-6
RLS(cubic)	3.0E-3 $\pm$ 5.9E-3	4.6E-7 $\pm$ 2.0E-6	2.3E-7 $\pm$ 9.1E-7
	Features: Var(L=1)		
RLS	2.8E-3 $\pm$ 4.6E-3	3.0E-6 $\pm$ 1.1E-5	4.2E-7 $\pm$ 2.4E-6
Local Poly KRLS ($w = 10, d = 3$)	2.0E-5 $\pm$ 4.5E-5		
Local Poly KRLS ($w = 20, d = 3$)	1.6E-5 $\pm$ 2.9E-5		
	Features: Var(L=2)		
Local Poly KRLS ($w = 10, d = 3$)	2.3E-5 $\pm$ 6.8E-5		
Local Poly KRLS ($w = 20, d = 3$)	1.1E-5 $\pm$ 2.8E-5		

Table 5.1: Time-series regression result(NMSE) (3200 training samples and 800 test samples): with time-series features, the prediction performance was significantly improved. Nonlinear regression with explicit nonlinear features through quadratic and cubic basis expansion were tested and performed better than linear model counterpart.

However, global modeling of the data has its limitations. Figure (5.4) shows the train and test error of linear and nonlinear AR(20) models. The prediction error still shows some trend: the error is bigger or smaller in some regions than in others, suggesting that the error is structured and not random. There are local variations in the data that are not captured by global models.

Still, the prediction error for global modeling is much lower than the error for the local models

used in the last chapter, which were able to choose good paths for informative path planning. We expect to see a better performance of path selection with the new models.



(a) Linear AR(20)

(b) Nonlinear AR(20)

Figure 5-4: Training and test error of linear and nonlinear AR(20) model. Though the average size of error is small, the magnitude depends on the region. Nonlinear model overfit in this case as the train error is smaller than linear model but test error is bigger.

5.5 Improving Local Models with Time-series Features

We expect that time-series features will also be able to improve local models. Preliminary attempts to utilize local regression with time-series features are presented in this section.

Table (5.2) shows the prediction performance of various local and global models using time-series features. We explain the LWR and the local RLS with a global prior methods in the sections below. The result shows that the direct application of the sliding-window approach, local RLS, tend to show varying performance, that is sensitive to the degree of time-series features and the sliding-window size. This is not desirable, as there are many models to be learned in our problem setting. We cannot simply choose the best model by trying different parameters for each propagation function in $\hat{F}$. However, using lower degree time-series features seems to work well with sliding-window approach. It appears that using higher degree time-series features is prone to overfit as the train set consists only of the small number of samples in the sliding-window, compared to features we get from the time-series. In addition, the samples in the sliding-window may not be

the best training samples to learn the local model.

The problem of finding better training samples is tested with the LWR and the overfitting issue is tested with the local RLS with a global prior.

5.5.1 Locally Weighted Regression

Due to the unsatisfactory results with sliding-window learning with time-series features, we have tried locally weighted regression (LWR). Though there is much debate about appropriate weight functions, here we try a simple weight function, which is just to calculate Euclidean distance between samples. The weight of a sample $\mathbf{x}_j$ for fitting a function to a sample $\mathbf{x}_i$ is:

$$w(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{|\mathbf{x}_i - \mathbf{x}_j|^2}$$

Figure (5.5.1) shows that the locally weighted model performed better than the global AR(10) model. The error variance decreased significantly as well.

However, LWR requires finding a set of nearest neighbors and fitting a function with the set every time for prediction. This will slow down our path planning algorithm. Furthermore, unlike sliding-window KRLS, there is no recursive solution for adding or deleting a sample into the nearest neighbors set. We expect that one can solve this problem by using nearest neighbor search with a pre-trained model database. We leave this to future work.

5.5.2 Local RLS with Global Prior

Another technique we tried is to use the sliding-window approach with a global prior. We see that global models with time-series features perform well but still the errors are heteroscedastic. Also, local learning with time-series features is prone to overfitting. A reasonable solution is to learn local models using the sliding-window approach with some prior knowledge about the global model. Effectively, this will constrain the flexibility of the local model so that it balances between deviation from the global model and local training error. This approach maintains the computational advantage of the sliding-window approach while possibly enjoying better generalization

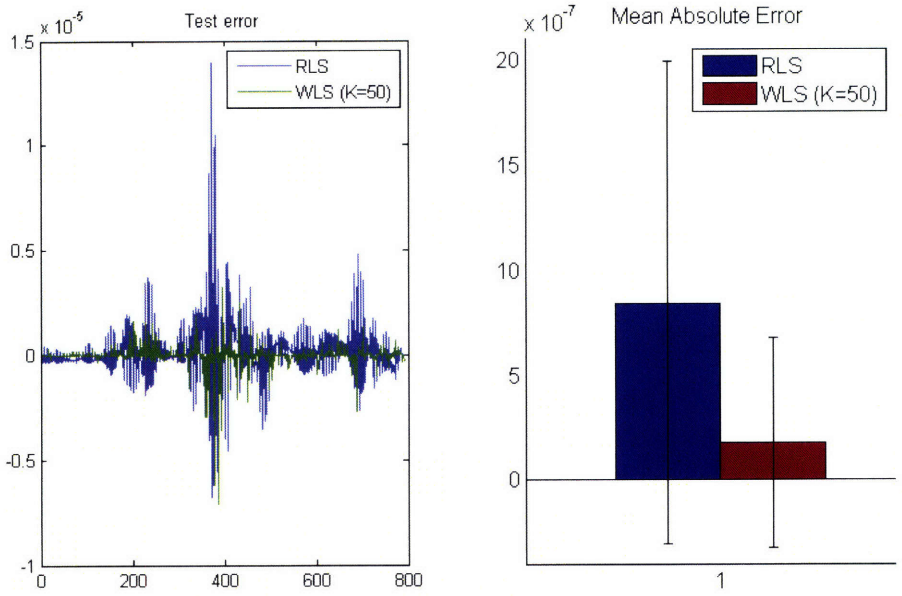


Figure 5-5: Test error (MAE) of global and locally weighted linear AR(10) model.

performance of higher-order time-series features.

Using RLS as the learning algorithm, we can put the prior over model coefficients, as in 4.1.3. This simply replaces the zero-prior on coefficients in the original RLS with the global model coefficients, so that the deviation from the global model is penalized.

We expect that we need to use different regularization parameters for global and local learning. This multi-stage approach of regularization, where we use multiple regularization parameters to iteratively fit models, is tested in other problems such as explicit feature selection. It is reported to give better results [36] than single-stage regularization.

From Table (5.2), we see that the (linear) RLS with a global prior performs better than some nonlinear global models and linear local models without a global prior. Further experiments are needed to verify the approaches we introduced in this section. However, the initial result is encouraging.

Model	Degree of time-series 5	10	20
Features: Var(L=0)			
RLS(quadratic)	3.5E-6	1.1E-7	1.8E-7
Poly KRLS ($d = 3$)	5.1E-7	2.7E-7	1.4E-6
Gaussian SVM ($\sigma^2 = 10^{-3}$)	1.3E-2	1.4E-2	9.2E-3
Local RLS($w = 10$)	2.5E-7	1.0E-5	4.4E-5
Local RLS($w = 20$)	5.5E-7	1.0E-5	8.0E-7
LWR ($K = 50$)	3.1E-6	**7.5E-8**	3.4E-7
Local RLS with Global Prior ($w = 10$)	2.7E-6	6.3E-7	4.0E-7

Table 5.2: Time-series regression result (NMSE) (3200 training samples and 800 test samples): Various local and global models with time-series features were tested. LSR of 50 nearest neighbors with degree 10 time-series features performed the best. However, further experiments are warranted to prove the better performance of the method.

5.6 Experimental Results of Path Selection

Here, we conduct another path selection experiment. Compared to 4.3, we do not use the features from spatial neighborhood in this section, meaning that we use pure autoregressive time-series features. This gives a significant computational advantage, especially for recursive predictions; we do not need to predict covariance entries beyond the region of interest (ROI). Before, we had to predict covariances of cells outside the ROI for recursive predictions as these entries are needed to generate spatial neighborhood features. For instance, suppose l is the one-cell ROI. Using the covariance block features CovB(L=1) to learn the propagation functions, the prediction of the next state of $\Sigma_t(l, l)$ is given by

$$\hat{\Sigma}_{t+1}(l, l) = f_{(l,l)}(\Sigma_t(\mathbf{N}_l)) \tag{5.8}$$

To make a recursive prediction to get $\hat{\Sigma}_{t+2}(l, l)$, we use

$$\hat{\Sigma}_{t+2}(l, l) = f_{(l,l)}(\hat{\Sigma}_{t+1}(\mathbf{N}_l)) \tag{5.9}$$

However, we do not know the entries of $\hat{\Sigma}_{t+1}(\mathbf{N}_l)$ except $\hat{\Sigma}_{t+1}(l, l)$ which was predicted at the last step. Thus, the other entries has to be predicted as well though they are not of the ROI. In contrast, with pure autoregressive time-series features, recursive predictions requires no additional predictions for generating features.

We also use global models in this experiment, meaning that we do not need to retrain models every timestep. Thus, the computation time will be even faster than in 4.3.

We report that the accuracy of path planning using AR global models was better than sliding-window KRLS with faster computation time. In the future, we expect to improve the accuracy with local AR models using LWS or RLS with global prior algorithms.

Figure (5.6) shows the accumulated trace difference between the greedy planning and our planning method. Our planning method does better than the greedy method though, in some cases, we may choose the same path as the greedy method. Only the cases where two algorithms pick different path is shown.

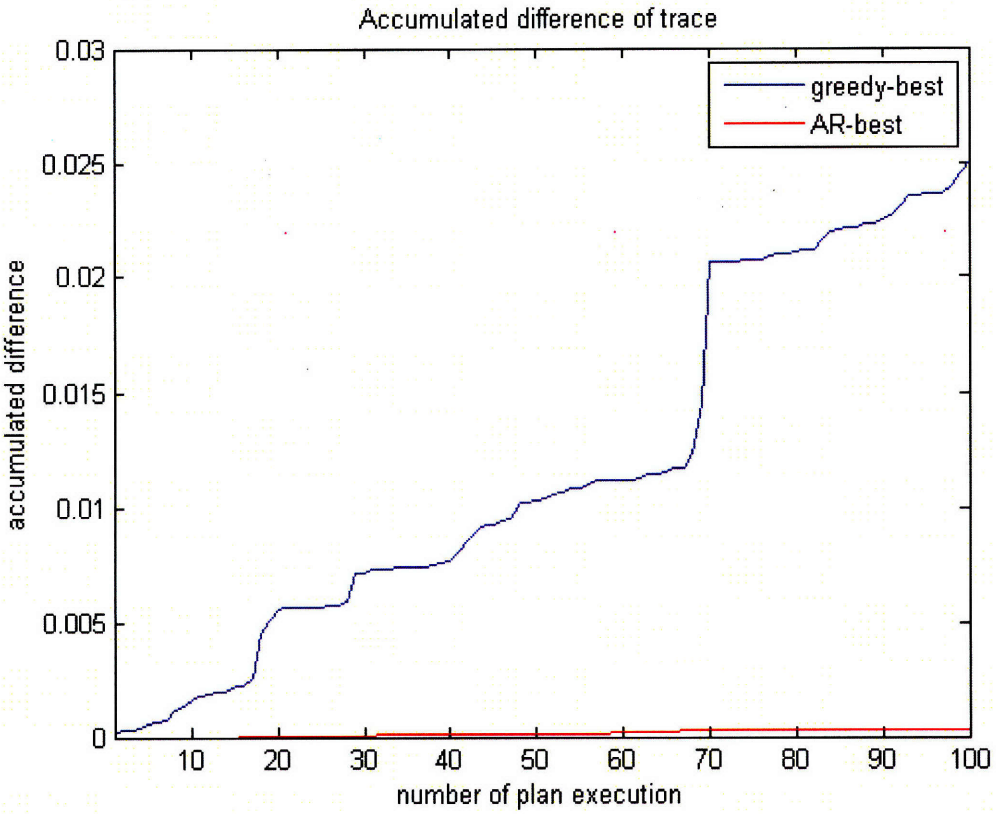


Figure 5-6: Accumulated difference of trace between global AR(20) model prediction and greedy of 5 x 1-timestep planning in Lorenz-2003 model

5.6.1 Computation Time

We show that AR prediction is much faster than full propagation and also faster than sliding-window KRLS. This is proved with the lower time complexity of the algorithm as in Table (5.3). The actual computation time in our simulation is shown in Table (5.4). We need additional training time in case we update models online. The online update is also fast for RLS as KRLS as it provides recursive solution.

	Complexity
Full Propagation	$\Omega(C_{int}N_eN_s + N_eR^2)$
KRLS Prediction	$O((dw + w^2)R^2)$
AR Prediction	$O(dR^2)$

Table 5.3: The computational complexity of the algorithms for the one-step prediction update. C_{int} is the nontrivial cost of nonlinear dynamics integration of one state variable. N_e is the ensemble size. N_s is the number of state variables in the system. N_e has to be $\Omega(N_s^2)$ [22]. R is the size of the region of interest. d is the size of the features for AR models. $d \ll N_e$, $R \ll N_s$ in our case so that AR prediction is significantly faster.

	Lorenz-2003 (N_e=1000)
Full Propagation	1984.48s
KRLS Prediction	14.97s
AR(20) Prediction	6.81s

Table 5.4: Average computation time for path planning for different methods. A total of 51 paths were evaluated using full propagation of EnKF, KRLS prediction and AR prediction (Pentium-4 3.0 Ghz Dual Core was used).

5.7 Conclusions

In this chapter, we experimented with the use of higher-order time-series features. The success of classical linear time-series models in many applications of signal processing motivated the use of time-series features in learning the prediction update of EnKF. There were two sources of change in the covariance evolution: the prediction update and the measurement update. The past effects of both update steps were incorporated into the time-series features.

The result of learning the prediction update was significantly improved with the use of time-series features. Combined with the spatial neighborhood features, the result was improved more.

We expected that recursive predictions would be more accurate with the global models as the models will be still valid for the new states of the covariances during the predictions. We showed its accuracy indirectly through the result of path selection.

With the global AR models, we are able to evaluate all candidate paths much faster than full propagation and even the sliding-window KRLS models. The training was done before the planning and no additional training was needed. The result of path selection was close to the optimal solution, as shown in Figure (5.6). The information gain achieved with the new models appears much better than the result of using local models, though the direct comparison was not made.

However, the global models were not able to capture all local variations. In the future, we aim to further improve the models by local modeling strategies with the use of time-series features.

Chapter 6

Explicit Feature Selection

Kernel method has been very popular in recent machine learning literature [15]. However, we argue that kernel method may have a few problems especially for the uncertainty prediction problem in our setting where many number of predictions has to be made and there are many possible features which are not certain to be useful a priori.

Kernel method implicitly maps original features into higher-dimensional space and find a linear regressor in that space. For instance, using a Gaussian kernel, one can map original features into a infinite-dimensional space. As we fit a function in higher-dimensional space, the modeling capacity is greater than in the original feature space. Simply, there are more functions that we can represent with the new features. Combined with careful regularization as in the SVM or KRLS algorithms, kernel methods has worked well in practice to find complex functions.

However, this approach has a number of problems. First, the model is non-parametric so that the model for a new sample needs to be constructed for every predictions. The KRLS algorithm in equation (3.25) shows that one needs to calculate the kernel matrix between original samples and the new sample, $K(X, x^*)$, to get the prediction for the new sample x^*. If the training set size is n and the original feature size is m, assuming that kernel calculation takes $O(m)$, this costs $O(nm)$ for every prediction. This is just to build a model and the actual prediction takes $O(n^2)$, so the total time for a prediction is $O(n^2 + nm)$. This is compared to the RLS which takes $O(m)$ calculations for one prediction.

This is significant for our planning problem as we make many prediction repeatedly, one for every covariance entry at each timestep. Especially, if the model is a global model trained with many training samples, this becomes almost infeasible.

Second, there are two sources of overfitting, original features and its kernelized form, and we have to control both. Using kernelized method and original features, one may find the best kernel (with best kernel parameters) according to some model selection criterion such as cross-validation. However, we also have choices of what to use as original features. In our case, we have spatio-temporal neighborhood of a cell (its value) that would be helpful to predict its value of the next timestep. We can choose different size of spatio-temporal neighborhood to include as features. Furthermore, we can extract lots of features from samples, such as ratio, difference, or boolean features (for example, a feature describes if the value went 'up' or 'down' at last time step). These features can't be created through implicit kernel mapping and has to be constructed manually. When we train a kernelized model with these many features, it's hard to select the model as we have to select the right kernel as well as the right feature set.

Another consequence is that kernelized method may not produce a good model when there are many redundant and noisy features. Suppose that we only use boolean features so that

$$x_{ij} = \begin{cases} 1, & \text{the boolean feature corresponding to } x_{ij} \text{ is true;} \\ -1, & \text{Otherwise.} \end{cases} \tag{6.1}$$

Imagine a bad case scenario where $\mathbf{x}$ has only one good feature and all others are noisy and non-discriminative features (e.g. all have same values). The polynomial kernel of degree d, $K_{poly=d}(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T \mathbf{x}_2 + 1)^d$, calculates the inner product of two samples. The inner product of any two sample will be dominated by noisy features. For example, if $\mathbf{x_1} = [01111...1]$ and $\mathbf{x_2} = [11111...1]$, where only the first feature is discriminative,

$$K_{poly=d}(\mathbf{x_1}, \mathbf{x_2}) = (0 + (T-1))^d = (T-1)^d \tag{6.2}$$

$$\simeq K_{poly=d}(\mathbf{x_1}, \mathbf{x_1}) = (T-1)^d \tag{6.3}$$

$$\simeq K_{poly=d}(\mathbf{x_2}, \mathbf{x_2}) = T^d \tag{6.4}$$

In these cases, one has to filter features first, which is *feature selection*. However, if we were to filter features first, we may better do feature selection and model selection simultaneously. It can be done with *embedded* feature selection, which is explained in the following section.

6.1 Background

[24] provides a good introduction to feature selection. We would distinguish between feature selection and model selection in that feature selection is to find good or *relevant* features before building a model and model selection is to choose a good model using the given features.

However, there are many choices to define what *relevant* means. Different method of feature selection choose to use different criteria. The feature selection can be categorized into

- Filters methods
- Wrappers
- Embedded methods

6.1.1 Filters methods

Filters method is to rank features according to some scoring statistic. A popular one is correlation score between a feature of samples and the label of the samples.

Definition Correlation score:

$$s_{corr} = \frac{Cov(X_i, \mathbf{y})}{\sqrt{Var(X_i)Var(\mathbf{y})}} \tag{6.5}$$

where X_i is a column vector of i-th feature of all samples.

The correlation score tells you how much a feature tend to move together (in the same or opposite direction) with labels. If some features are almost independent with correlation score near 0, we may safely remove it from consideration. Filters methods is *preprocessing* and fast.

However, there is no explicit criteria to choose the threshold, which we remove features with the scores below than that. For instance, it is hard to decide whether we will remove features with

correlation score below 0.1 or 0.2. Also, a filter method doesn't consider the algorithm we will use to learn a model. Different algorithm may prefer different features. One algorithm may not be able to exploit a feature with low score so that it just becomes noise, but other may be. We would like to know how these features actually contribute to prediction performance, and that leads into 'Wrappers'.

6.1.2 Wrappers

Wrappers is basically a strategy to choose subsets of features to find the best subset for a given learning algorithm. The best subset is chosen according to some criteria such as cross-validation. There are two main approaches:

- backward elimination: start with all features and eliminate one iteratively
- forward selection: start with no features and add one iteratively

with some stopping criteria. However, as one can guess, this require huge amount of computation. Basically, one need to train model multiple times for each stage of feature selection. For forward selection, if we select a variable every stage among remaining variables, we need to run the learning algorithm $O(m!)$ times, where m is the number of features.

6.1.3 Embedded methods (Lasso)

In embedded methods, model selection and feature selection becomes indistinguishable as a learning algorithm gives a *sparsified* model which only uses a subset of features, with only one training process. This is usually done by *sparsifying* regularization.

Lasso [43] was introduced as an embedded method of feature selection. Recall that the original RLS equation was given by:

$$\min_{\beta} \quad ||(X\beta - \mathbf{y})||^2 + \lambda\beta^T\beta \tag{6.6}$$

where λ is regularization parameter which controls the contribution of the L2-norm of regression coefficients β to the total loss function; small λ encourages big $|\beta|^2$ and big λ encourages small $|\beta|^2$.

However, in the original RLS, a coefficient will be usually not zero due to the squared penalty for the coefficients β [43]. Basically, the contribution of $\Delta\beta_i$ for some i to the loss function in equation (6.6) increase or decrease with the magnitude of β_i.

In the Lasso, instead of L2-norm of regression coefficients, L1-norm of regression coefficients are penalized. Thus, the optimization problem of Lasso is

$$\min_{\beta} \quad ||(X\beta - \mathbf{y})||^2 + \lambda \sum_{i=1}^{d} |\beta_i| \tag{6.7}$$

Using L1-norm as the penalty, the contribution of the change of a coefficient $\Delta\beta_i$ for some i to the loss function in equation (6.7) does not depend on the magnitude of β_i; it encourages sparsification.

The above optimization problem is not quadratic and cannot be solved as quickly as the original RLS problem. Still, the problem is convex and there are optimization algorithms available to solve the problem [43, 36]. In the following section, we will experiment with sparsification property of Lasso and the generalization performance of the solution produced by Lasso.

6.2 Result

We compare the generalization performance and sparsity of the resulting model, using the RLS and Lasso, in learning a global model. The target function is the prediction update of the full EnKF in the Lorenz-2003 model with 1000 ensemble members.

In building explicit nonlinear features, we choose to allow only a certain degree of interaction between features. For example, let a sample covariance evolution of t-timesteps be

$$\Sigma_{1:t}(i) = [\Sigma_1(i)^f, \Sigma_1(i)^a, \Sigma_2(i)^f, \Sigma_2(i)^a, ..., \Sigma_t(i)^f, \Sigma_t(i)^a] \tag{6.8}$$

$$= [\Sigma_1(i), \Sigma_2(i), ..., \Sigma_t(i)] \tag{6.9}$$

where $\Sigma_t(i)$ represents $[\Sigma_t(i)^f, \Sigma_t(i)^a]$ for simplicity. We only allowed to use the interaction terms

Model	Timesteps			
	5	10	20	40
RLS	1.02E-06 (10)	3.73E-07 (20)	2.17E-07 (38)	1.90E-07 (78)
Lasso	1.23E-06 (10)	7.55E-07 (15)	4.56E-07 (30)	2.54E-07 (30)

Table 6.1: NMSE of 1-step forecast predictions of a variance on 1000 test samples. 4000 samples were used to train. Only first order features were used. The numbers inside the parenthesis are the size of the model (the number of non-zero coefficients). RLS always outperform Lasso in terms of prediction accuracy. However, the model size of Lasso is always smaller. For 40-timesteps case, Lasso's model is twice smaller. The model size difference will be critical with more features.

Model	Timesteps		
	5	10	20
RLS	6.8E-07 $\pm$ 2.2E-06 (42)	1.6E-07 $\pm$ 4.9E-07 (92)	0.85E-07 $\pm$ 2.9E-7 (192)
Lasso	9.5E-07 $\pm$ 3.7E-06 (28)	2.5E-07 $\pm$ 8.4E-07 (77)	1.4E-07 $\pm$ 6.6E-7 (140)

Table 6.2: NMSE of 1-step forecast predictions of a covariance on 1000 test samples. 4000 samples were used to train. First and second order (2 degree of interaction) features were used. The numbers inside the parenthesis are the size of the model (the number of non-zero coefficients). RLS always outperform Lasso in terms of prediction accuracy. However, the model size of Lasso is always smaller.

of first k-timesteps.

$$\Sigma(i)^2_{1:t} = [\Sigma_1(i)\Sigma_1(i), \Sigma_1(i)\Sigma_2(i), ...\Sigma_1(i)\Sigma_k(i), ..., \tag{6.10}$$

$$\Sigma_2(i)\Sigma_2(i), ..., \Sigma_2(i)\Sigma_{k+1}(i), ..., \Sigma_t(i)\Sigma_t(i)] \tag{6.11}$$

It was primarily for computational efficiency and numerical problems, as the number of features increase greatly with the higher-order basis expansions.

The tables 6.2, 6.2 and 6.2 show the result of feature selection and prediction performance. The Lasso solution is compared with the RLS solution.

Model	Timesteps	
	5	10
RLS	6.4E-07 $\pm$ 2.3E-06 (150)	1.5E-07 $\pm$ 5.5E-07 (330)
Lasso	8.2E-07 $\pm$ 2.5E-06 (103)	2.4E-07 $\pm$ 9.3E-07 (218)

Table 6.3: NMSE of 1-step forecast prediction of a covariance on 1000 test samples. 4000 samples were used to train. Covariance and mean were used this time. First and second order (4 degree of interaction) features were used. With mean as features, the prediction performance got a little better, but not significantly.

6.3 Discussion

The result suggest that the RLS has always better prediction performance than the Lasso. However, other concern is the model size. In informative path planning, to get the future uncertainties, we have to use the models recursively for each entry of the covariance. If the model size is m and we have n^2 entries to predict for future t timesteps, the time complexity is $O(mn^2t)$. For the Lasso solution, m will be smaller than the RLS solution. Though a sparse model would only allow constant-time speed-up, it would be still valuable for certain cases when real-time operation is critical.

We also have not tried to use many other possible feature sets due to time limitations. When one can generate a huge set of relevant features, it would be valuable to use an embedded feature selection method such as the Lasso.

Chapter 7

Conclusion and Future Work

In this work, we have introduced and attempted to solve an informative path planning problem in a highly dynamic weather system. The EnKF, a standard nonlinear estimation technique and spatio-temporal model, was used to estimate state of the weather system and also to model the uncertainty of the state estimate.

The goal of the planning problem was to maximize the uncertainty reduction, or equivalently the information gain, in the estimation framework. For that purpose, the information gain of each candidate path had to be evaluated with the multiple iteration of the uncertainty propagation procedures in the EnKF. However, the procedures in the EnKF are computationally-intensive; it involves a series of Monte-carlo simulations. The exact evaluation of information gain in the EnKF would only allow only few paths to be evaluated due to the timing constraints.

We explored a variety of regression approaches to learn models of uncertainty propagation in the EnKF. In Chapter 3, we formulated the learning problem to learn the prediction update, the "propagation" functions, of the EnKF with spatial neighborhood features. After an initial training period, the learned functions were expected to provide a fast means to predict the future uncertainties within a planning horizon.

The prediction update was expected to be a nonlinear function thus nonlinear regression algorithms were applied. However, the direct application of state-of-art nonlinear regression algorithms failed to learn the uncertainty propagation accurately, failing short to improve over simple linear

models.

The nonstationarity, or locality, of the propagation functions were suspected and we investigated local regression approaches in Chapter 4. The sliding-window KRLS algorithm was proved to be successful in learning the propagation functions accurately so that multi-step propagations of uncertainty was predicted with good accuracy. The efficiency of the KRLS prediction allowed a large set of candidate trajectories to be considered quickly. The path selected with the learned functions was shown to be significantly better in reducing the uncertainty of the estimates. The computation was almost free, compared to the EnKF.

However, the limitation of the local models in Chapter 4 was essentially its locality. The local models may not be the valid function for the new state of covariance after a few timesteps. The new state of uncertainty requires a new local model, thus a new set of training samples, that we do not have in the sliding-window.

In Chapter 5, we introduced time-series features to improve the generalization of the learned models. Instead of looking at a single point of the covariance evolution, we used the history of evolution as the features. This increased the modeling capacity and we were able to learn global models of evolution utilizing all the training samples. The models were tested for informative path selection and showed the improvement in terms of accuracy and speed over local models.

Still, it was shown that there is local trend, heteroscedasticity, in the prediction errors and local modeling with time-series features would be able to improve over global models. The RLS with a global prior method and the Locally Weighted Regression (LWR) method both showed some promise in improving the prediction. We achieved the best prediction with the LWR approach but it is not computationally-efficient, as multiple nearest neighbor search is required to train new models at every timestep.

In future work, we would like to propose a computationally-efficient solution to the locality of models by building a database of models. Local models can be trained beforehand using the LWR and later retrieved by a nearest neighbor search. We may use only one or a few neighbors. It was shown that averaging models of neighbors gave better result [3]. There are many nearest neighbor search method with $log(n)$ complexity, thus rendering this approach feasible. Also, by training

models beforehand, we do not need to train models in execution time of informative path planning. The database can be incrementally updated or rebuilt after some period of operation by a separate thread of computations.

Another important problem is to deal with the accumulation of errors in recursive prediction. In our approach, we did not deal with it explicitly. There are approaches with various regression methods that explicitly aims to improve recursive prediction [3, 11]. With some modification, we may be able to adapt the works to our problem.

In terms of improving the weather forecast, our approach will be tested in operational weather models combined with some high-level planner such as the one in [13]. Additionally, all learning approaches are expected to perform even better with larger ensemble, whose size is limited by computational resources but expected to increase continually. We would like to confirm the utility of the approach in the operational models estimated with large ensemble.

Bibliography

[1] S. Aksoy and R. M. Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern Recognition Letters*, 22(5):563–582, 2001.

[2] L. M. Berliner, Z. Q. Lu, and C. Snyder. Statistical design for adaptive weather observations. *Journal of the Atmospheric Sciences*, 56(15):2536–2552, 1999.

[3] M. Birattari, G. Bontempi, and H. Bersini. Lazy learning meets the recursive least-squares algorithm. *Advances in Neural Information Processing Systems*, 11:375381, 1999.

[4] C.H. Bishop, B.J. Etherton, and S.J. Majumdar. Adaptive Sampling with the Ensemble Transform Kalman Filter. Part I: Theoretical Aspects. *Monthly Weather Review*, 129(3):420–436, 2001.

[5] G. Bontempi, M. Birattari, and H. Bersini. Local learning for iterated time-series prediction. *Machine Learning: Proceedings of the Sixteenth International Conference*, page 3238.

[6] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[7] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[8] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer, 1991.

[9] R. G. Brown, P. Y. C. Hwang, and R. G. Brown. Introduction to random signals and applied kalman filtering. 1992.

[10] C. J. C. Burges. Geometry and invariance in kernel based methods. *Advances in Kernel MethodsSupport Vector Learning*, page 89116, 1999.

[11] J. Q. Candela, A. Girard, J. Larsen, and C. E. Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, 2, 2003.

[12] H. L Choi, J. P How, and J. A Hansen. Ensemble-based adaptive targeting of mobile sensor networks. *American Control Conference, 2007. ACC'07*, page 23932398, 2007.

[13] H.L. Choi and J.P. How. A multi-uav targeting algorithm for ensemble forecast improvement. In *Submitted to the AIAA Guidance, Navigation, and Control Conference*, 2007.

[14] H.L. Choi, J.P. How, and J.A. Hansen. Ensemble-Based Adaptive Targeting of Mobile Sensor Networks. *American Control Conference, 2007. ACC'07*, pages 2393–2398, 2007.

[15] N. Cristianini and J. Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press New York, NY, USA, 1999.

[16] R. L. Devaney. *A First Course in Chaotic Dynamical Systems: Theory and Experiment*. Westview Press, 1992.

[17] SS Dhillon and K. Chakrabarty. Sensor placement for effective coverage and surveillance in distributed sensor networks. *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, 3, 2003.

[18] Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 52(8), 2004.

[19] D. Engster and U. Parlitz. 3 local and cluster weighted modeling for time series prediction. *Handbook of Time Series Analysis: Recent Theoretical Developments and Applications*, 2006.

[20] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):150, 2000.

[21] G. Francisco and P. Muruganandam. Local dimension and finite time prediction in spatiotemporal chaotic systems. *Physical Review E*, 67(6):66204, 2003.

[22] R. Furrer and T. Bengtsson. Estimation of high-dimensional prior and posterior covariance matrices in kalman filter variants. *Journal of Multivariate Analysis*, 98(2):227255, 2007.

[23] Carlos Guestrin, Andreas Krause, and Ajit Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

[24] I. Guyon, A. Elisseefi, and L. P. Kaelbling. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(7-8):1157–1182, 2003.

[25] R. M. Hodur. The naval research laboratorys coupled ocean/atmosphere mesoscale prediction system (coamps). *Monthly Weather Review*, 125(7):1414–1430, 1997.

[26] P. L. Houtekamer and H. L. Mitchell. Data assimilation using an ensemble kalman filter technique. *Monthly Weather Review*, 126(3):796811, 1998.

[27] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[28] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. *Proceedings of the fifth international conference on Information processing in sensor networks*, pages 2–10, 2006.

[29] D. Kugiumtzis, O. C. Lingjcrde, and N. Christophersen. Regularized local linear prediction of chaotic time series. *Physica D: Nonlinear Phenomena*, 112(3-4):344–360, 1998.

[30] J. J. Leonard and H. F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Springer, 1992.

[31] M. Leutbecher. A reduced rank estimate of forecast error variance changes due to intermittent modifications of the observing network. *Journal of the Atmospheric Sciences*, 60(5):729–742, 2003.

[32] E. N. Lorenz. Designing chaotic models. *Journal of the Atmospheric Sciences*, 62(5):1574–1587, 2005.

[33] E. N. Lorenz and K. A. Emanuel. Optimal sites for supplementary weather observations: Simulation with a small model. *Journal of the Atmospheric Sciences*, 55(3):399–414, 1998.

[34] E. N. Lorenz and K. A. Emanuel. Optimal sites for supplementary weather observations: Simulation with a small model. *Journal of the Atmospheric Sciences*, 55(3):399–414, 1998.

[35] S.J. Majumdar, C.H. Bishop, B.J. Etherton, and Z. Toth. Adaptive sampling with the ensemble transform kalman filter: Part ii field programming implementation. *Monthly Weather Review*, 130(3):1356–1369, 2002.

[36] L. Meier and P. Buhlmann. Smoothing l-penalized estimators for high-dimensional time-course data. *Electronic Journal of Statistics*, 1:597–615, 2007.

[37] R.E. Morss, K.A. Emanuel, and C. Snyder. Idealized adaptive observation strategies for improving numerical weather prediction. *Journal of the Atmospheric Sciences*, 58(2), 2001.

[38] T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. *Nature*, 317(6035):314–319, 1985.

[39] R. Rifkin, G. Yeo, and T. Poggio. Chapter 7 Regularized Least-Squares Classification. *Advances in Learning Theory: Methods, Models and Applications*, 2003.

[40] D. Ruppert and M. P. Wang. Multivariate locally weighted least squares regression. *ANNALS OF STATISTICS*, 22:1346–1346, 1994.

[41] T. Sauer, J. A. Yorke, and M. Casdagli. Embedology. 1991.

[42] B. Thiesson, D. M. Chickering, D. Heckerman, and C. Meek. Time-series modeling with graphical models.

[43] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[44] Zoltan Toth. Winter storm reconnaissance program. http://www.aoc.noaa.gov/article_winterstorm.htm, 2005.

[45] S. Van Vaerenbergh, J. Va, and I. Santamara. A sliding-window kernel rls algorithm and its application to nonlinear channel identification. *Proc. of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[46] S. Van Vaerenbergh, J. Vıa, and I. Santamarıa. Nonlinear System Identification using a New Sliding-Window Kernel RLS Algorithm. *JOURNAL OF COMMUNICATIONS*, 2(3):1, 2007.

[47] G. Welch and G. Bishop. An Introduction to the Kalman Filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[48] J.S. Whitaker and H.M. Hamill. Ensemble data assimilation without perturbed observations. *Monthly Weather Review*, 130(7):1913–1924, 2002.

[49] J.S. Whitaker and H.M. Hamill. Ensemble data assimilation without perturbed observations. *Monthly Weather Review*, 130(7):1913–1924, 2002.

[50] P. Zhang and J. Peng. Svm vs regularized least squares classification. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1, 2004.

[51] P. Zhang and J. Peng. SVM vs regularized least squares classification. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1, 2004.