UNIVERSITY OF CALIFORNIA
SANTA CRUZ

VARIABLE-LENGTH FUNCTIONAL OUTPUT PREDICTION AND BOUNDARY DETECTION FOR AN ADAPTIVE FLIGHT CONTROL SIMULATOR

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

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS AND STATISTICS

by

Yuning He

December 2012

The Dissertation of Yuning He
is approved:

Professor Herbert Lee, Chair

Professor Bruno Sanso

Professor Gong Qi

Dr. Karen Gundy-Burlet

Dr. Misty Davies

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Variable-length Functional Output Prediction and Boundary Detection for an Adaptive Flight Control Simulator

by

Yuning He

The general problem addressed in this work concerns the analysis of a function of multiple real variables in which the output is itself a function of a real variable as well as categorical information. We are interested in the prediction of the output and the analysis of the boundaries separating regions during classification. Efficient and accurate prediction of the output curves is important for safety analysis and validation of a complex system, like an adaptive flight control system. An understanding of the boundaries between regions, where the aircraft can maintain stable flight or will break up is essential for safety certification. As a motivating application we are using the NASA IFCS (Intelligent Flight Control System) flight simulator.

For output prediction, we developed a new statistical method for emulation of computer models with multiple outputs. An emulator is a computationally efficient statistical model that is used to approximate a computationally expensive simulation by treating the simulator as a black box and learning a mapping from inputs to outputs. Our approach for emulating the curves is to represent them in an orthogonal basis (e.g., PCA, Fourier, or Wavelet), which captures curve characteristics, and then to predict the coefficients. We allow for the possibility of output curves whose length varies with

input, which may occur when a simulator fails to run to completion for some inputs and the failures occur at different output time steps. To the best of our knowledge, the variable-length output problem has not yet been addressed. We have developed a hierarchical model, which first uses classification into a few groups and then fitting distinct models for different classes of output curves.

For the analysis of boundaries, we developed a new sequential approach based upon design of computer experiments. A dictionary of suitable linear or non-linear parameterized boundary shapes, which capture underlying physical and design knowledge can be provided by the domain expert. We incorporate this knowledge into our modeling and determine the most likely shapes and its parameters. Since each iteration requires a costly run of the system simulator, we developed a candidate selection mechanism, which is specifically tailored toward boundary detection and which can take priors into account in order to reduce the number of required simulation runs. We present results of experiments with artificial and IFCS simulation data sets.

Chapter 1

Introduction

The general problem addressed in this work is the prediction of the output of a function of multiple real variables in which the output is itself a function over time. We propose a new statistical method for the emulation of computer models with multiple dynamic outputs. An emulator is a computationally efficient statistical model that is used to approximate a computationally expensive simulation by treating the simulator as a black box and learning a mapping from inputs to outputs.

Emulation Our approach for emulating the curves for a single output variable is to represent the curves in an orthogonal basis, which captures curve characteristics, and then to predict the output curve for a new input by predicting the coefficients for the desired output curve's basis representation. We allow for the possibility of output curves whose length varies with input, which may occur when a simulator fails to run to completion for some inputs and failures occur at different output time steps. To the

best of our knowledge, the variable-length output problem has not yet been addressed. In many applications, output curves can typically be represented accurately with only a small number of curves from the full basis. The use of a reduced basis representation allows for a fixed length representation of output curves so that existing methods can be applied, with a large reduction in data down to the correct degrees of freedom so that our solution is practical even when the output curve lengths are in the hundreds or thousands of time samples. In this work, we compare different types of bases, like PCA, wavelet, and Fourier Transform.

When output curves can be grouped into a small set of clusters of similar curve length, shape, and frequency content, we hypothesize that fitting distinct models for different classes of output curves will improve the prediction. Therefore, we add a class parameter to our statistical model. Our complete model is built on top of statistical models for the single output functions from input to class, input to output curve length, and input to each of the coefficients in a reduced basis curve representation. The class function has a categorical output, while the length and coefficient functions have real-valued outputs. We propose modeling the real-valued single output functions using non-stationary Treed Gaussian Process (TGP) models or regression Support Vector Machines (SVMs). For modeling the function from input to class, we propose using an extension of TGP for categorical outputs called CTGP or a classification SVM.

Case Study The motivating application for our prediction method is the statistical emulation of the NASA Intelligent Flight Control System (IFCS). The IFCS is a neuro-

adaptive flight control system, which is capable of accommodating toward damage of the aircraft during the flight. This technology was developed at NASA and successfully flown on a manned F-15 jet aircraft. For our experiment, we use a Matlab/Simulink flight simulator with many input output variables, where each output variable is a function over time indicating some aircraft configuration measurement such as a pose angle. This is a very difficult prediction problem with a relatively high-dimensional input space and a very high-dimensional, variable size output space with curve lengths in the hundreds for failure runs and curve lengths of 1901 for successful runs. Many of the output curves have high frequency oscillations caused by the neural network part of the simulator used in adaptive flight control whose goal is to stabilize the aircraft by removing differences between measured configuration parameters and expected values under a reference model.

Experiments and Results We present the results of our curve prediction method for the NASA flight simulator assuming for now that the output length and, therefore, correct curve classification are known. Several possible orthogonal bases have been explored, including Principal Components Analysis (PCA), Wavelet, and Fourier bases, as well as three different class structures, including 2 classes for simulator runs which complete successfully and fail to run to completion, 4 classes to indicate successful runs and 3 failure classes based on failure time (which equals output length), and fitting just one model for all successful and failure runs. The current prediction results are very promising, and clearly demonstrate the potential of the basis approach in which

coefficients are predicted using a statistical model such as TGP or SVM for modeling scalar-valued functions. The lowest prediction errors are obtained using the PCA basis with 4 classes for output curves.

Boundary Detection The IFCS flight control system can, like any other complex system only operate safely within a given operational envelope. Such an envelope is spanned by multiple parameters, which can be design parameters, environmental parameters, or operational parameters. Of most importance for the analyst is the identification of areas, in which the aircraft can operate safely and the boundaries between those safety regions and unsafe areas. Many well-known analysis techniques in aircraft design are based upon linear analysis. In many practical applications, boundaries of regions are simply approximated by ranges of individual parameters. Advanced aircraft design and control techniques like the IFCS damage adaptive aircraft control exhibit strong non-linear effects that require novel analysis techniques. In this work, we develop a methodology for finding and characterizing safety boundaries. We use techniques of computer experiments and active learning in order to find the necessary data points with as few runs of the system simulator as possible. The shape of the boundaries are estimated using a small library of possible shape classes, which can be provided by the domain expert. The availability of boundaries described by parameterized shapes is helpful for manual system analysis by the domain expert and can be used for aircraft monitoring and vehicle health management.

Chapter 2

Case Study: Intelligent Flight Control

The Generation 2 (Gen 2) NASA Intelligent Flight Control System (IFCS) uses machine learning techniques to stabilize the aircraft under new and unexpected failure conditions (e.g., stuck rudder or damage to a wing), with the ultimate goal of providing increased safety for the aircraft. This is accomplished through the use of an online-adaptive neural network in the inner loop of the flight control system. The IFCS adaptive flight control system utilizes online neural networks (OLNNs), one for each of the axes roll, pitch and yaw, in real time, to try to adapt the control signals in order to retain good flight characteristics in the presence of sudden damage, slow degradation, or new environmental situations. Figure 2.1 shows the basic IFCS architecture. A traditional flight control system receives stick inputs from the pilot and uses an aircraft reference model to create desired commands for the actuators (ailerons, elevators, rudder). The deviation between the desired commands and the aircraft's sensor signals are used by a PI (proportional integral) controller to calculate

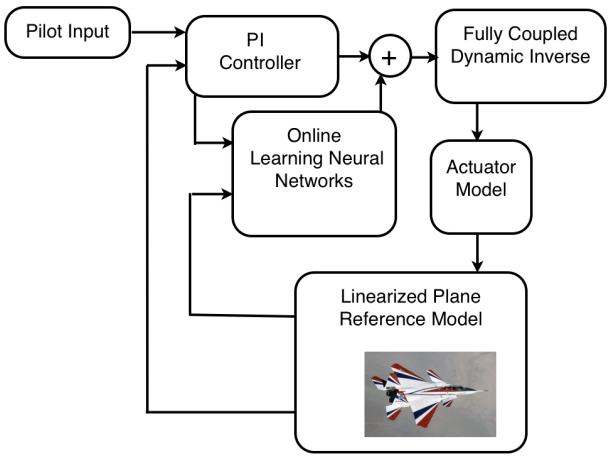

Figure 2.1: Gen 2 IFCS architecture

desired rates in pitch, roll, and yaw axes. A dynamic inverse and actuator model then converts the rates into actual deflection commands for the surfaces. Our simulation model uses a linearized model of the F-15 ACTIVE aircraft, a highly customized NASA jet plane. This basic controller has been augmented by a neural network, which provides control augmentation, i.e., the neural network's output is added to the control command. The aircraft's sensor signals and outputs of the standard proportional-integral (PI) controller is sent to the OLNNs. The OLNNs compare the PI controller output with the output from the linearized plane reference model, which is the desired output response, and attempt to drive the errors between the two outputs to zero, by augmenting the command from the PI controller before it is fed into the nonlinear dynamic inverse.

The IFCS has been implemented in a manned F-15 aircraft and extensive piloted evaluations have been performed [Burken *et al.*, 2006; Smith *et al.*, 2010]. Each online adaptive control system is provided with a learning algorithm that dynamically adjusts the weights of the neural network as data are coming in. This version of the IFCS architecture uses a Sigma-Pi ($\Sigma\Pi$) neural network [Rumelhart *et al.*, 1986], where the inputs are $\mathbf{x}$ subjected to arbitrary basis functions (e.g., square, scaling, logistic function). The output of the network o is a weighted sum (Σ) of the Cartesian product of the basis function values (Figure 2.2):

$$o = \sum_i w_i b_i \quad \text{where} \quad b_i = \prod_j \beta(\mathbf{x}_j)$$

with weights w_i and basis functions $\beta(\mathbf{x}_j)$. Online adaptation (learning) is taking place while the adaptive controller is operating using a simple, yet effective weight adaptation rule governed by a learning rate γ (for details see [Calise and Rysdyk, 1998; Rysdyk and Calise, 1998]). For this controller, asymptotic stability can be proven using a Lyapunov method [Rysdyk and Calise, 1998]; however, large damages, low convergence speed or overtraining can prevent stabilization [Schumann and Liu, 2007]. However, the overall behavior of the adaptive control system is hard to predict. [Jacklin *et al.*, 2004; Liu *et al.*, 2004; Broderick, 2004; Jacklin *et al.*, 2005; Menon *et al.*, 2006; Menon *et al.*, 2007] The adaptation learning gain has a large effect on algorithm stability and learning convergence. The analysis of the parameters, which can yield a potentially unstable system is of particular importance. The IFCS online adaptive system is highly nonlinear and therefore it is difficult to model. This is where Statistical Emulation

techniques are coming to play, to develop an algorithm that would provide efficient learning of the system one step further. That is, to build a Statistical Emulator that describes the system in a simpler and more revealing manner, and provides similar outputs as IFCS simulator when the same set of inputs are provided.

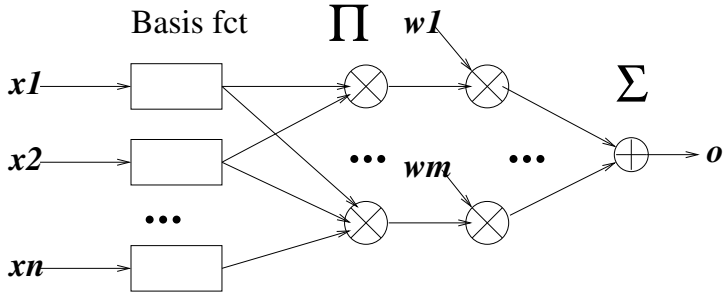

Figure 2.2: Architecture of ΣΠ network

For this work, we did not obtain data from live airplanes and actual control software. Rather, we work instead with a computer simulation of the entire process (the airplane and the control system). Our simulator is an early prototype implemented in Mathworks' Simulink [Mathworks, 2012], and has MATLAB wrappers to control the values of simulation. A given set of experimental conditions (altitude, pilot input, speed) were provided and kept constant in our experiments. Other parameters (Table 2.1) were modified in a systematic way (3-factor combinatorial exploration, see Section 5.3.2.1) to obtain a total of 967 test runs. Each simulation run was covering 20 seconds simulation time with a basic time step of 0.01s. Each step within the simulation is *deterministic*—a set of the same inputs will always lead to the same time series outputs. In order to avoid start-up effects, the first second of simulation time was ignored, yielding simulator

outputs of 12 variables over time for a length of up to $T_{\max} = 1901$ time steps. However, the current version of the control system is not always able to stabilize the aircraft for the entire time. In such cases, the simulator fails at $T < T_{\max}$ time steps. This time-to-failure is an important characteristic, which we attempt to predict in this work. Table 2.1 shows the name and description of the 11 input variables, the 12 output variables, and the time to failure, which can be considered to be a derived output.

Figure 2.3 shows the outputs obtained with two different sets of input x_1 and x_2 which correspond to two examples of successful and failure runs respectively. For input configuration x_1 (top), all outputs stabilize after initial excitation caused by pilot input (pilot input happens around T=0) and online adaptation. After some initial oscillations, the curves dampen and finally reach another stable state. The simulation ends successfully with $T = 1901$ time steps. For input configuration x_2 (bottom), on the other hand, signal excitation does not decrease over time. Rather, some high-frequency oscillation (caused by a bad adaptation of the network) can be detected, leading to instability at around $T = 380$ time steps. All the 12 outputs failed at the same time with different oscillations. This example of an aborted or failure run, typically manifests itself as numerical calculation problems caused by one or more program variables becoming very large or going out of expected bounds in the currently implemented control algorithms. The flight simulator represents a complex, non-linear mapping from the 11 input variables to the 12 output variables. The names of the input and output variables are given below. The input variables are aircraft characteristics such as mass (with a fixed set of pilot inputs) and the outputs are various measurements of the aircraft

configuration such as attitude angles.

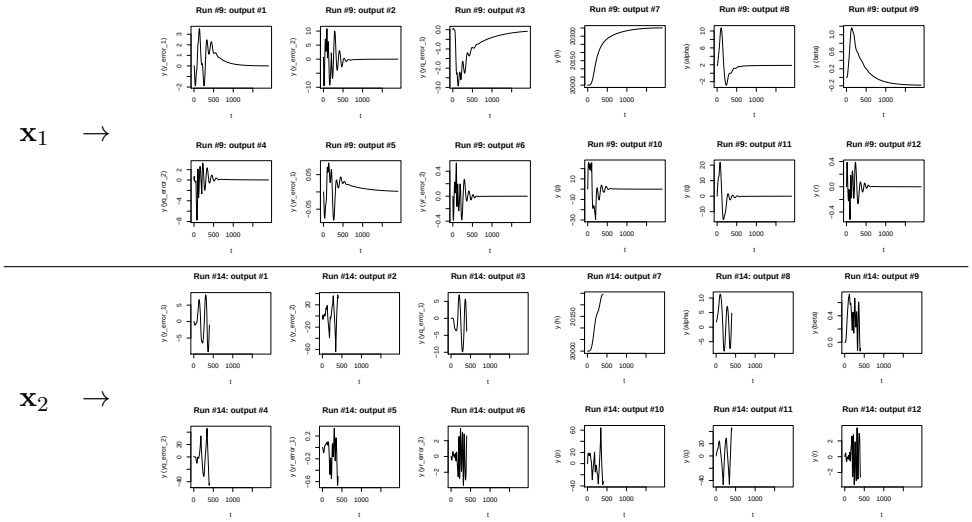

Figure 2.3: Typical output time series for successful (top) and failure (bottom) runs.

Name	Description
Input parameters	
K_{lat}	lateral stick gain
K_{lon}	longitudinal stick gain
K_{pedal}	rudder gain
ζ	damping coefficient
$w_{p,q,r}$	proportional gain (three axes)
K_3	controller gain (yaw axis)
$\lambda_{p,q,r}$	NN learning rates (three axes)
Outputs	
$e\vec{r}r^1_{p,q,r}$	proportional error wrt. reference model, three axes [degrees]
$e\vec{r}r^2_{p,q,r}$	integral error wrt. reference model, three axes [degrees*sec]
$\vec{h}$	altitude [feet]
$\vec{p}, \vec{q}, \vec{r}$	roll, pitch, and yaw rates [degrees/sec]
$\vec{\alpha}$	angle of attack [degrees]
$\vec{\beta}$	sideslip angle [degrees]
F	a Boolean variable: TRUE for cases which fail and FALSE otherwise
T	time to failure [number of time steps]

Table 2.1: Input parameters and output variables. Note that the pilot inputs are kept constant in this experiment and are not considered to be a input.

Chapter 3

Emulation

It is obvious that one of the main goals is to keep the aircraft stable in the nominal case and in the presence of damage. Many parameters of the IFCS, like gain parameters for the PI controller, govern the behavior of the entire system. The adaptation learning gain has a large effect on algorithm stability and learning convergence. The analysis of the parameters, which can yield a potentially unstable system is of particular importance. The IFCS online adaptive system is highly nonlinear and therefore it is difficult to model. This is where Statistical Emulation techniques are coming to play, to develop an algorithm that would provide efficient learning of the system one step further. That is, to build a Statistical Emulator that describes the system in a simpler and more revealing manner, and provides similar outputs as the IFCS flight simulator when the same set of inputs are provided.

In this chapter, we explore the possibility that we can use *statistical emulation* to overcome the limitations of traditional validation techniques. Statistical emulation

has a key benefit in that it allows for *uncertainty quantification*—a necessity for the validation of safety-critical systems. Statistical emulation, particularly statistical emulation based on Classification Treed Gaussian Processes, has been used to reliably predict the behavior of complex, non-linear, high-dimensional systems that are locally smooth. We have demonstrated that statistical emulation can be used to reliably predict time-to-failure for an adaptive flight control simulator. We have also shown that, given an input, we can predict the time series curve outputs of the simulator with high accuracy. As a byproduct of this prediction, the behavior of the simulator becomes quantified, and can be compared against experimental data for validation. We have extended statistical emulation to explicitly handle time series outputs of varying lengths. With some work, we will be able to explicitly handle time series inputs, as well.

3.1 Hierarchical Approach

Time series, in general, are not guaranteed to have the same length, since the time duration to events or failures is likely to be different for each given set of inputs. For our application, each output variable is a time series, in other words a vector that is a function of time (with discrete time steps). In our experiments, we executed the simulator for a maximum of $T_{max} = 1901$ time steps. As we described above, for many inputs $\vec{x}$ the system successfully controls the aircraft. For successful cases we record $T_{max} = 1901$ as the time to 'failure'. But there are other situations, when the software is unable to adapt the control successfully to maintain aircraft stability. In these cases the

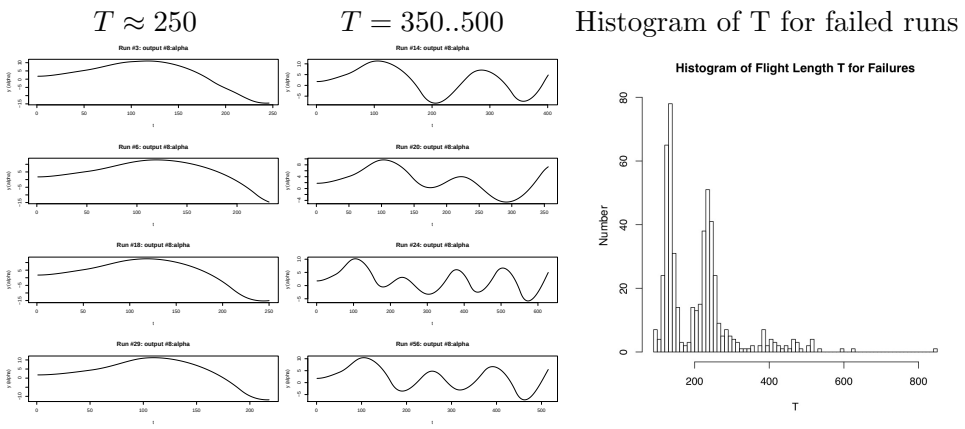

Figure 3.1: Curve classes

simulator terminates prematurely, having recorded fewer than the maximum number of timesteps as the case for a successfully controlled flight.

We observe that the output curves can be grouped into sets of clusters of similar curve length, shape, and frequency content. General appearance and frequency content of success and failure curves are typically quite different (see Figure 2.3). The output curve clusters tend to correlate well with failure curve length (i.e., the time to failure). Some examples of the failure curves for the same output variable are shown in Figure 3.1 below. As we can see, when the time to failure is about 250 time steps, the curves for one output variable but different runs look similar to each other as shown in the first column of Figure 3.1. When time to failure is between 350 to 500 time steps, the curves look similar to each other as shown in the second column of Figure 3.1.

Therefore, we have examined classification strategies not using two classes (success and failure) but also four classes with classes defined by the ranges of the

output length. A histogram for time to failure presented in Figure 3.1(right) nicely supports our four-class model. Two clusters are shown quite clearly in the histogram, the third class picks up the remaining failure runs. Finally, class four (not shown) is comprised of the successful runs. This histogram is the basis for setting our thresholds to $(180, 280, 1900)$ time steps as discussed in Section 3.8, Table 3.1.

We have implemented a toolkit for prediction by proposing a two-stage hierarchical statistical model (Figure 3.2(left)). Our architecture consists of a statistical model for the mapping from input to class, and within each given class, a distinct statistical model for the relationship between input and output. And we hypothesize that, by fitting distinct models for each of the distinct classes, the overall results of prediction accuracy will be improved. This hypothesis is proven to be true supported by our experiment results in the later Section 3.8. Our hierarchical model can capture the possibility of model parameters variation across groups. The hierarchy arises because the model for the parameters sits above the model for the data.

In order to train our model, we use a set of time series data as obtained from our simulation model with different input parameters x (Figure 3.2(right)). According to the actual obtained time to failure, we split this data set into successful runs and two or four classes for the failure runs (as discussed above). As we are using supervised learning methods, each of the data sets will be in turn split into a training set and test set, respectively. Details will be discussed below. In order to learn the mapping from input to class, we use a supervised classification method. Classification means assigning a given piece of input data into one of a given number of categories or classes.

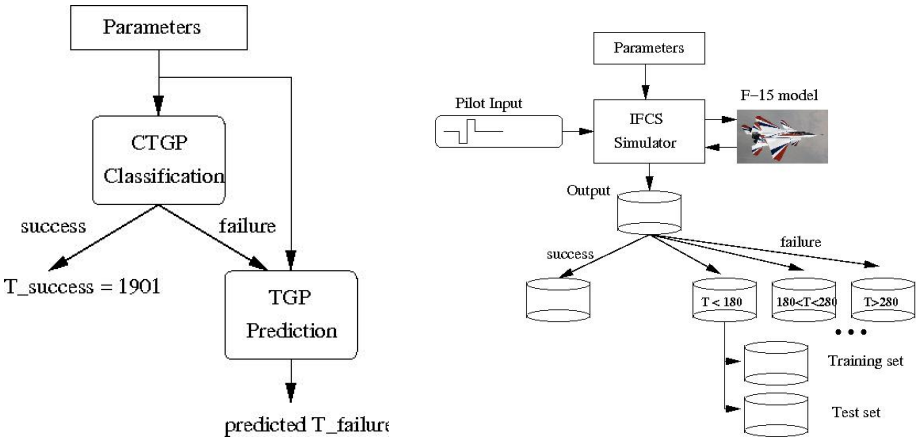

Figure 3.2: Our methodology: 2-stage prediction of time to failure (left) and supervised learning (right).

Classification normally refers to a supervised procedure, i.e., a procedure that learns to classify new instances based on learning from a training set of instances, which have been properly labeled with the correct classes. The classification problem can be stated as follows: given training data, establish a rule which can be evaluated for any possible value of input (not just those included in the training data) and such that the group attributed to any new observation.

For modeling the real-valued function, i.e., learning the relationship between inputs to Time to failure values, we propose using the Treed Gaussian Process (TGP) model, a statistical modeling method, which is an extension of the popular Gaussian Process (GP) described in Section 3.3. In Section 3.8 we discuss results obtained with TGP and compare results with other learning methods in the literature, for example Support Vector Machines (SVMs).

3.2 Statistical Emulation

The traditional emulation problem starts by considering modeling a scalar-valued function $y = f(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^p$. A widely used statistical model for such a regression problem is a Gaussian process (GP) [Sacks *et al.*, 1989; Kennedy and O'Hagan, 2001; Santner *et al.*, 2003]. For inputs $\mathbf{x} \in \mathbb{R}^p$, a GP is formally a distribution on the space of functions $y : \mathbb{R}^p \to \mathbb{R}$ such that the function values $\{y(\mathbf{x}_1), \ldots, y(\mathbf{x}_n)\}$ at any finite set of input points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ have a multivariate Gaussian distribution. A particular GP is defined by its mean function $m(\cdot)$ and its correlation function $c(\cdot, \cdot)$: $y = f(\cdot)|\beta, \sigma^2, \mathbf{r} \sim N(m(\cdot), c(\cdot, \cdot)\sigma^2)$. Here we use a linear mean function, $m(\mathbf{x}) = \mathbf{x}^T\beta$ and the Gaussian correlation function $c(\mathbf{x}, \mathbf{x}') = \exp\{-(\mathbf{x} - \mathbf{x}')^T R(\mathbf{x} - \mathbf{x}')\}$, where $R = \text{diag}(\mathbf{r})$ is a diagonal matrix of p positive roughness parameters $\mathbf{r} = (r_1, \ldots, r_p)$. The smoothness of the Gaussian correlation function is typically appropriate for computer simulators, but other classes of correlation functions are possible, such as the Matérn family.

The stationarity of the GP model can limit its applicability. The Treed Gaussian Process (TGP) model [Gramacy and Lee, 2008] is more flexible and overcomes this limitation by subdividing the input space and modeling each region r_ν of the input space using a different GP, thus leading to a non-stationary model. The TGP model

includes a *hierarchical* model for the GP in region r_ν given in [Gramacy and Lee, 2008]:

$$\begin{aligned}
\mathbf{y}_\nu|\beta_\nu,\sigma_\nu^2,K_\nu &\sim N_{n_\nu}(\mathbf{F}_\nu\beta_\nu,\sigma_\nu^2 K_\nu), & \sigma_\nu^2 &\sim IG(\alpha_\sigma/2,q_\sigma/2),\\
\beta_\nu|\sigma_\nu^2,\tau_\nu^2,W,\beta_0 &\sim N_{p+1}(\beta_0,\sigma_\nu^2\tau_\nu^2 W), & \tau_\nu^2 &\sim IG(\alpha_\sigma/2,q_\sigma/2),\\
\beta_0 &\sim N_{p+1}(\mu,B), & W^{-1} &\sim W((\rho V)^{-1},\rho),
\end{aligned} \tag{3.1}$$

where N_d, IG, and W indicate d-dimensional Normal, Inverse-Gamma, and Wishart distributions, respectively. Here $\mathbf{y}_\nu$ is a vector of response values for the n_ν training inputs $\mathbf{X}_\nu \in r_\nu$ and $\mathbf{F}_\nu = (\mathbf{1}, \mathbf{X}_\nu)$. The correlation matrix K_ν for region r_ν contains the values of the correlation function c for pairs of training inputs: $K_\nu(i,j) = c(\mathbf{X}_i, \mathbf{X}_j)$.

The TGP subdivision process is hierarchical and done by recursively partitioning the input space via a binary tree. The parameters defining the subdivision process are part of the TGP statistical model and ultimately determine the number of regions in the subdivision. A prior on the size of the subdivision tree $\mathcal{T}$ is specified through a tree generating process that splits a leaf node ν with probability $p_{\text{SPLIT}}(\nu,\mathcal{T}) = a(1+q_\nu)^{-b}$, where q_ν is the depth of ν in $\mathcal{T}$ and a and b are model parameters. See [Gramacy and Lee, 2008] for the complete prior specification for the subdivision tree $\mathcal{T}$ and the TGP model fitting algorithm. The tree itself can be fit simultaneously with the GP parameters in the leaves using Reversible Jump Markov chain Monte Carlo.

As noted in the previous section, we can improve our predictive performance by first separating the curves into four classes. To predict class membership we use an extension of TGP called Classification TGP (CTGP) [Broderick and Gramacy, 2011]. For predicting M classes, the CTGP model introduces M latent variables Z_m, $m =$

$1, \ldots, M$, to define class probabilities via the softmax function:

$$p(C(\mathbf{x}) = m) = \frac{\exp(-Z_m(\mathbf{x}))}{\sum_{m'=1}^{M} \exp(-Z_{m'}(\mathbf{x}))} \tag{3.2}$$

Each class function $Z_m(\mathbf{x})$ is modeled using TGP.

Within each class, we actually want to predict an output curve, not just a scalar response. One approach for predicting output curves is to extend the GP model to functions $\mathbf{y} : \mathbb{R}^p \to \mathbb{R}^q$, where the vector output $\mathbf{y}$ represents samples of a curve at $q = T$ time points. In the context of statistical emulation of a computer simulator, [Conti and O'Hagan, 2010] call this the *Multi-output (MO) emulator* and provide the statistical model for q-dimensional Gaussian Processes, which is analogous to the standard model. In addition to the MO emulator, [Conti and O'Hagan, 2010] outline two other possible approaches for multi-output emulation: Ensemble of single-output (MS) emulators and the Time Input (TI) emulator. In the MS approach, each of the T curve values are predicted independently using T single-output emulators. On the other hand, the TI approach adds the time parameter t to the input $\mathbf{x}$ and builds one, single-output emulator for $y(\mathbf{x}, t) : (\mathbb{R}^p \times \mathbb{R}) \to \mathbb{R}$. The MO emulator is the simplest from the computational perspective with a computational load that is comparable to a single-output GP emulator in which the bottleneck is $n \times n$ matrix inversion for n training inputs $\mathcal{S}$. The MS method uses T single-output GP emulators and thus has a computational burden T times more than that of the MO method. A naive implementation of the TI emulator would require nT times the computation of the MO emulator as the training samples are now $\mathcal{S} \times \{1, \ldots, M\}$, but the structure of the problem allows the required

$nT \times nT$ matrix inversions to be done via $n \times n$ and $T \times T$ matrix inversions. Of course, this is still more computation than required by the MO emulator. Using a linear mean specification, the mean functions for the MO and MS methods are the same, but the mean function for the TI method is more restrictive because it assumes a mean function $m(\mathbf{x}, t) = \beta_0 + \beta_{\mathbf{x}}^T \mathbf{x} + \beta_t t$ that is linear in t, where the coefficient vector is decomposed as $\beta = (\beta_0, \beta_{\mathbf{x}}, \beta_t)^T$, although a different mean function can be used for the TI emulator that results in an equivalent mean function as the linear mean for the other two methods. More differences arise in the correlation structure of predictions. The MS method estimates different roughness parameters $\mathbf{r}$ for each output time, which allows the most flexibility, but is unrealistic for computer model emulation because of the lack of correlation over time. The MO and TI methods estimate a single $\mathbf{r}$ for all times. The MO method allows more generality in the covariance between different outputs $f_{t_1}(\mathbf{x}_1)$ and $f_{t_2}(\mathbf{x}_2)$ at different locations $\mathbf{x}_1$ and $\mathbf{x}_2$, with the TI method constraining it to an exponentially decreasing squared time difference.

In practice, we find that the stationary modeling assumption can often be overly restrictive. Instead of using GPs, we use TGPs. However, the above methods are not as easily adapted to TGP models, so we consider a different approach. The size and multivariate nature of the data lead to computational challenges in implementing the framework.

To overcome these challenges, [Higdon *et al.*, 2008] makes use of basis representations (e.g., principal components) to reduce the dimensionality of the problem and speed up the computations required for exploring the posterior distribution. However,

the success of their approach largely depends on whether or not the simulator can be efficiently represented with the GP model on the basis weights. This is apparent in the principal component decomposition, which partitions nearly all of the variance in the first few components. These systems also tend to exhibit smooth dependence on the input settings. In contrast, more chaotic systems seem to be far less amenable to a low-dimensional description such as the PC-basis representations used here. Also, system sensitivity to even small input perturbations can look almost random, making it difficult to construct a statistical model to predict at untried input settings. This approach also has the issue of extrapolating outside the range of experimental data. The quality of such extrapolative predictions depends largely on the trust one has for the discrepancy term at tried experimental conditions applicable to a new, possibly far away, condition. Because of our variable length outputs, we can end up in somewhat of an extrapolation situation when predicting curves that are longer than the related training samples.

In our approach to predicting one output variable curve, we represent the output curve $\mathbf{y} \in \mathbb{R}^T$ in terms of a linearly independent set of D orthogonal curves $B \in \mathbb{R}^{T \times D}$: $\mathbf{y} \doteq B\mathbf{c}$. We use orthogonal curves which measure different curve characteristics so that the basis coefficients in $\mathbf{c} = (c_i) \in \mathbb{R}^D$ are uncorrelated. Then we model the coefficients c_i, $i = 1, \ldots, D$, independently using D TGP models. By changing the curve representation, we can use an MS approach without being subject to the criticism of not modeling correlations between the outputs. Now the multiple output values being modeled are not values of the curve at distinct time points but rather the

coefficients in a basis representation of the curve. In addition to the MS advantages of simplicity and flexibility of modeling correlations of the same coefficient output value over different inputs $\mathbf{x}$, the MS implementation can be parallelized by running each single output emulator, both fitting and prediction, in parallel. In many applications, using $D \ll T$ orthogonal curves will suffice to accurately represent output curves and the use of a good basis provides a substantial data reduction. Another advantage of our basis representation approach is that it allows us to have a fixed size output representation D for applications which have output curves whose lengths T vary with input $\mathbf{x}$. The problem of functional data prediction has received some attention in recent years. In her thesis work, [Liu, 2007] also uses a basis approach (with a wavelet basis). There are, however, a few significant differences between the prediction problem considered in our work and in her work. First, we seek to predict multiple, correlated output functions instead of a single output function. The second, and most fundamental difference, is that we consider the case of variable-length output functions which naturally arise in the application of emulating the NASA flight simulator. Third, our application is for emulation of a computer model in which the simulator is deterministic, always returning the same output for a fixed input.

In the case of variable length output curves, modeling the output curve $\mathbf{y}(\mathbf{x})$ requires modeling both the length $T(\mathbf{x})$ as well as the $T(\mathbf{x})$ curve values at different times. In the NASA flight simulator application, we found that the output curves for a single output variable could be grouped into a small set of clusters of similar curve length, shape, and frequency content (see Section 3.1). Fitting distinct models for

different classes of output curves improves the predictive performance. Therefore, we add a class parameter C to our statistical model. We allow for different bases to be used for different classes and so the basis matrix B_C is now indexed by the class C. Our model must now be able to predict the class C from the input $\mathbf{x}$ and the prediction of the output curve $\mathbf{y}$ is now conditioned on the determined class $C(\mathbf{x})$. The details of the class criteria for the NASA application are given in Section 3.4. We found that a CTGP model for $C(\mathbf{x})$ gave the best classifier accuracy.

3.3 Our Statistical Models

We have so far kept the modeling description somewhat generic. Now we fill in the model details. Once the model is fully specified, we then need to fit the model. We may separate our data into a training set used to fit the model, and a hold-out test set used to verify the accuracy. To fit our full model, we use the training data to first fit a CTGP classifier $C(\mathbf{x})$. Next we train a TGP model to fit the curve length $T(\mathbf{x})$, conditional on the class (Figure 3.2). Finally, we fit additional TGP models for the coefficients of a basis representation of $\mathbf{y}(\mathbf{x}) \doteq \sum_{i=1}^{D} c_i \mathbf{b}_i$:

1. *Fitting.* Find coefficients $\{ c_i \}_{i=1}^{D}$ to approximate the training output curves $\mathbf{y}$ for one variable in some basis $\{ \mathbf{b}_i \}_{i=1}^{D}$:

$$\mathbf{y} \doteq \sum_{i=1}^{D} c_i \mathbf{b}_i \tag{3.3}$$

2. *Learning Coefficient Mappings.* Learn the D mappings from inputs $\mathbf{x}$ to coefficients c_i for $i = 1, \ldots, D$.

3. *Coefficient Prediction.* Predict the coefficients $c_{i,\text{pred}}^{\text{new}}$ for a new input $\mathbf{x}^{\text{new}}$ for $i = 1, \ldots, D$.

4. *Output Curve Prediction.* Predict the output curve $\mathbf{y}^{\text{new}}$ corresponding to input $\mathbf{x}^{\text{new}}$ using the predicted coefficients $c_{i,\text{pred}}^{\text{new}}$:

$$\mathbf{y}_{\text{pred}}^{\text{new}} := \sum_{i=1}^{D} c_{i,\text{pred}}^{\text{new}} \mathbf{b}_i \tag{3.4}$$

The above strategy can be carried out on subsets of the data, allowing for different bases for different classes of output curves. Successful runs produce output curves of length $T(\mathbf{x}) = 1901$. For runs on inputs $\mathbf{x}$ that cause simulator failures, the output curves $\mathbf{y}(\mathbf{x})$ have lengths $T(\mathbf{x}) \leq 600$. Furthermore, the general appearance and frequency content are typically quite different for the success and failure curves. We therefore hypothesize that using different bases for the two classes of success and failure will improve the prediction of output curves. Even the failure curves seem to cluster in terms of appearance and the clusters tend to correlate well with failure curve length (i.e. the time to failure). Thus we consider running the above emulation strategy in two class and four class settings with classes defined by the following ranges of the output (flight) length.

3.4 Classification

Learning models for different classes offers the possibility of high quality prediction in a given class, because the learning method can focus just on the variance

within that class. In contrast, if we apply our method to all data as one class, then the variance among a larger group needs to be captured and the prediction may not be as good. However, the disadvantage of the multi-class strategy is that classification errors may lead to errors in curve prediction.

We hypothesized that there was structure in the data that we could exploit in order to improve our predictions. In particular we observed that the output curves could be grouped into sets of clusters of similar curve length, shape, and frequency content. We examined several classification strategies, including classifying into just two classes of success and failure runs. We found that a four-class solution which breaks out three different groups of failures offers the best balance between improved prediction within classes and tolerable error in predicting the class. As discussed previously, the histogram in Figure 3.1 is the basis for setting our thresholds to 180, 280, and 1900 time steps. Our data set consists of a total of 967 runs obtained from the simulator. We randomly split the 967 runs into 637 training examples and 300 test examples, using the training data to fit the models and the test data to evaluate classifier performance. Table 3.1 shows the total number of simulation runs in each of the four classes. Table 3.1 summarizes the results for our two different classification strategies: dividing the data into two classes (*Two Class Problem*) and into four classes (*Four Class Problem*). The *Two Class Problem* strategy separates the data into *failure* and *success* categories, where runs in *success* complete at 1901 time steps (the length of the simulation) and runs in the *failure* category end earlier. The *Four Class Problem* strategy separates the data into four different classes: three different failure classes (*failure1*, *failure2*, and *failure3*

respectively), and one class for all of the successes. Each of the three failure classes has a characteristic time series output shape, and we correctly hypothesized that using a different mathematical model for each of these shapes would improve performance.

Two Class Problem			Four Class Problem		
failure	$0 < T \leq 1900$	569 runs	*failure1*	$0 < T \leq 180$	257 runs
			failure2	$180 < T \leq 280$	241 runs
			failure3	$280 < T \leq 1900$	71 runs
success	$T = 1901$	398 runs	*success*	$T = 1901$	398 runs
total		967 runs	*total*		967 runs

Table 3.1: A summary of the data classification strategies we used in our experiments.

The results for several classifiers for the two class problem are shown in Table 3.2. As a benchmark, we implemented a simple nearest neighbor classifier (first row) that finds the closest training input $\mathbf{x}^{\text{train}}$ to a given $\mathbf{x}$ and classifies $\mathbf{x}$ as the same class as $\mathbf{x}^{\text{train}}$. This gave the highest error rate, 43.9%, of all the classification methods tested, probably in part because the density of the training inputs is not very high. The generalized linear model (GLM) is a flexible generalization of ordinary linear regression that allows for response variables that have other than a normal distribution.

Method	Classification Error	Training Time
Nearest Neighbor	43.9%	
GLM with binomial link	39.7%	few minutes
TGP_Flight_Length	27.3%	3 hours
R-Tree	26.9%	few minutes
TGP_binary	26.7%	3 hours
CTGP	11.8%	10+ hours
SVM	19.4%	$<$ 1 second

Table 3.2: Two Class { *failure*,*success* } Classification Results. See the text for details.

The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value. Its results are labeled "GLM with binomial link" in Figure 3.2.

As a whole, the TGP-based classification methods gave the lowest classification error rates. First we used TGP to learn the mapping $\mathbf{x} \to T(\mathbf{x})$ from input to flight length, and then we classified $\mathbf{x}$ into *success* if the predicted $T(\mathbf{x}) \geq \tau$ and *failure* otherwise. Here we used the threshold $\tau = 1500$. This method is identified in Figure 3.2 as "TGP_Flight_Length", and resulted in an error rate of 27.3%. R-trees are tree data structures used for spatial access methods, i.e., for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons. The key idea of this data structure is to group nearby objects and represent them by their minimum bounding rectangle in the next higher level of the tree. Thus the "R" in R-tree is for rectangle. At the leaf level, each rectangle describes a single object; at higher levels the aggregation of an increasing number of objects. This can also be seen as an increasingly coarse approximation of the data set. This method ("R-tree") classified the data set within a few minutes with an error rate of 26.9% similar to the TGP method. A slight improvement in error rate to 26.7% was obtained by using TGP to learn the mapping $x \to P(\mathit{success})$ from inputs to the probability of success and then thresholding the resulting real-valued TGP predictions at 0.5 to make the final classification into *success* and *failure*. This method is named "TGP_binary" in Figure 3.2. It is similar to "TGP_Flight_Length" except that the known class of training inputs is used to input 0 or 1 for the mapping

range instead of the flight length $T(\mathbf{x})$. The original TGP [Gramacy, 2007] does not do classification and subsequent predictions are real-valued even though all the training examples were binary-valued. Thus the real-valued predictions are thresholded to obtain the final classification ("TGP_binary").

Recently an extension of TGP called CTGP [Broderick and Gramacy, 2009b; Broderick and Gramacy, 2009a] was developed to use Treed Gaussian Processes for classification. The training data for CTGP is the same as for TGP_binary, but for CTGP the mapping range is categorical data representing the *success* and *failure* classes and the CTGP predictions are exactly these two output classes. CTGP gave by far the lowest error rate, 11.8%, of all the classification methods tested. The disadvantage of the TGP methods is that they require large training times in comparison to other methods, with CTGP taking more than 10 hours to train. The increased training time for CTGP over the 3 hours required for TGP_Flight_Length and TGP_binary did, however, provide a greatly reduced error rate. Using Support Vector Machines (SVMs) for classification provided the second lowest error rate, 19.4%, after CTGP, but with almost instantaneous training on our small training data set. The critical factor for our application, however, is classifier accuracy. We can train offline for a long time if the result is a low classification error rate.

3.5 Handling Variable-length Output Curves

In this work, we face a fundamental challenge of output curves with different lengths. Consider the fitting step of our emulation strategy for a particular class. Suppose that the upper limit in flight length that defines a given class is $T_{\max}$. For example, for the *failure* 2 class $T_{\max} = 280$. The basis $B \in \mathbb{R}^{T_{\max} \times D}$ for a class will have vectors of length $T_{\max}$ to accommodate the longest curves in the class. However, some curves in a class will have lengths less than $T_{\max}$. So how should we determine the coefficients $c \in \mathbb{R}^D$ for a training curve $\mathbf{y}$ of length $T < T_{\max}$? The linear system $Bc = \mathbf{y}$ has too many rows on the left hand side or too few entries on the right hand side depending on how you look at it.

One option is to remove the bottom $T_{\max} - T$ rows from the bottom of B to match the length T of training output $\mathbf{y}$. Learning the D functions from inputs $\mathbf{x}$ to coefficients c_i computed in this way from the training data resulted in terrible predictions. At first we thought this was due to the destruction of orthogonality when cropping the basis vectors. We had carefully chosen B to have orthogonal columns in an effort to make independent learning and prediction of the coefficients c_i a plausible strategy. The new columns in the truncated B will not be orthogonal. But the prediction results did not improve when we repeated the experiment after re-orthogonalizing the truncated basis vectors. Truncating the basis vectors seems flawed in that we are using different Bs for training outputs of different lengths and then trying to learn $\mathbf{x} \to c = B^{-1}\mathbf{y}$. Shorter curves will lead to different coefficients than longer curves, because they

are not needing to fit the missing end segment. This discrepancy can lead to unstable behavior in the coefficients, resulting in poor predictive performance.

A second option is to extend the training output curve $\mathbf{y}$ by padding it with an extra $T_{\max} - T$ elements to bring its length up to $T_{\max}$. This is not ideal since it requires making up data at the end of $\mathbf{y}$. By breaking the prediction problem into classes, the amount of padding needed is reduced in comparison to solving with a single class (which requires $T_{\max} = 1901$ to accommodate the *success* curves). We increase the length of training vectors $\mathbf{y}$ to $T_{\max}$ by repeating the last element in $\mathbf{y}$. Once we have predicted the coefficient vector $c_{\text{pred}}^{\text{new}}$ for a new input $\mathbf{x}^{\text{new}}$, the predicted output curve $\mathbf{y}_{\text{pred}}^{\text{new}} = Bc_{\text{pred}}^{\text{new}}$ has length $T_{\max}$. In our testing phase, we know the true length T for the test case and we truncate $\mathbf{y}_{\text{pred}}^{\text{new}}$ to length T for comparison with the true output curve $\mathbf{y}^{\text{new}}$.

A third option for determining the coefficients when the basis vectors are longer than the curve being fit is to truncate the basis vectors but then place a zero-mean Normal prior on the basis coefficients to shrink the coefficients toward zero. This strategy avoids the need to extend curves to have the same length as the basis vectors. As mentioned earlier (option one), simply truncating the basis vectors to the curve length and then doing an unconstrained least squares fit gave terrible results. Instead, we put a zero-mean Normal prior (with variance 10) on the coefficients in the truncated problem to bias the fitted coefficients toward zero and keep them from becoming too large. This shrinkage induces enough stability to enable good predictions.

3.6 Bases

We considered several different bases with which to represent output curves, of which we present two in this section. The length N of each basis vector corresponds to the length of the curves being represented. A true *basis* would require N basis vectors. This representation would be too large for our output curves, for example requiring 600 basis vectors for *failure* and *failure* 3 classes and 1901 basis vectors for the *success* class. Thus we keep only the most important $D \ll N$ basis vectors for representing the training data, where the *reduced basis* dimension D is a parameter chosen to balance the size and accuracy of our representation. For ease of discussion, we refer to reduced bases as simply *bases*. In the bases we consider, we can obtain very good fits for relatively small dimensions such as $D = 15$ or $D = 25$; we use $D = 25$ in the rest of this chapter.

3.6.1 Principal Components Basis

The *Principal Components Analysis (PCA) basis* for a given dimension D is an orthonormal, training data-dependent basis that identifies the D basis vectors that capture the most variance in the data using a PCA decomposition. Suppose the n_{train} training data output curves are placed into the columns of a matrix $Y \in \mathbb{R}^{N \times n_{\text{train}}}$. Then subtract off the mean output curve $\mu \in \mathbb{R}^{N \times 1}$ from each of the columns of Y to obtain the centered data $Y_c \in \mathbb{R}^{N \times n_{\text{train}}}$. If $Y_c = U\Sigma V^T$ is a Singular Value Decomposition (SVD) of Y_c, then the columns of $U \in \mathbb{R}^{N \times N}$ form an orthonormal basis for the centered output curve data. Assuming the singular values are ordered from largest to smallest as

usual, the reduced PCA basis is $B = U(:, 1{:}D)$ and explains $\sum_{i=1}^{D} \sigma_i^2 / \sum_{i=1}^{\min(N, n_{\text{train}})} \sigma_i^2$ of the variance in the output curves Y. The first five PCA basis vectors for output variable 8 success class is shown in Figure 3.3. The number for each basis curve in the

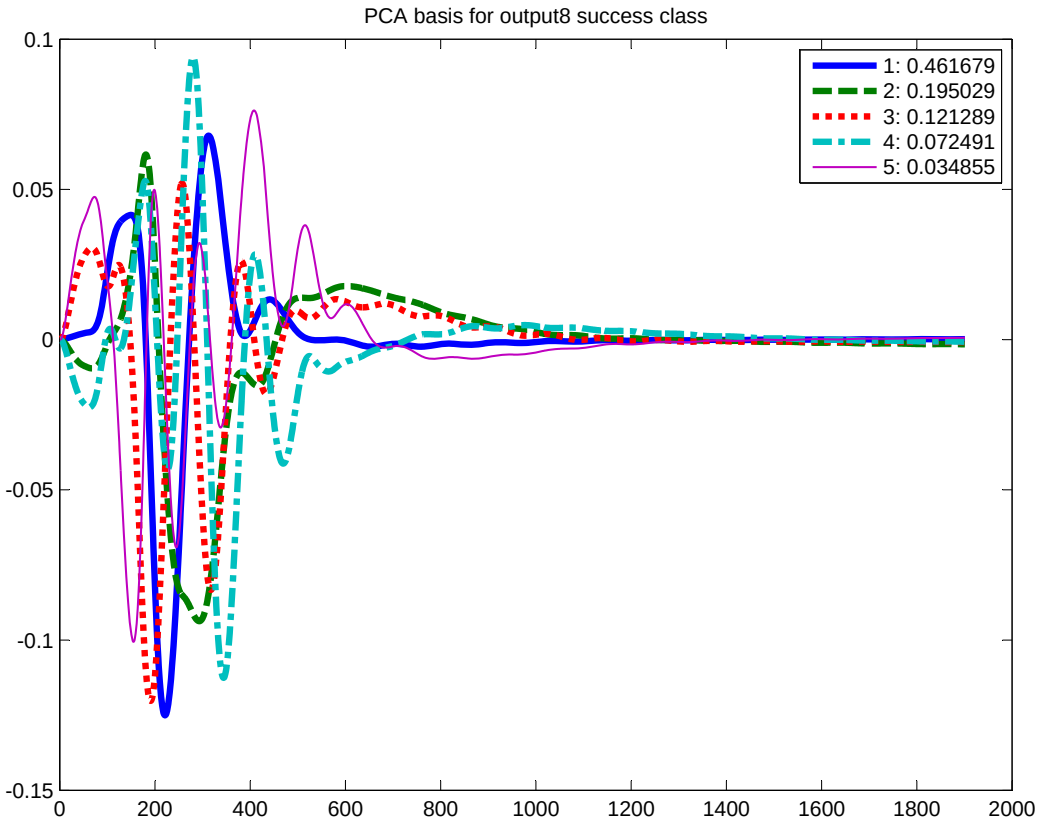

Figure 3.3: PCA basis (first five basis vectors) for output variable 8, success class; the legend shows the fraction of variance explained by each of the bases.

legend refers to the additional fraction of the variance in the data that is explained by using that basis curve.

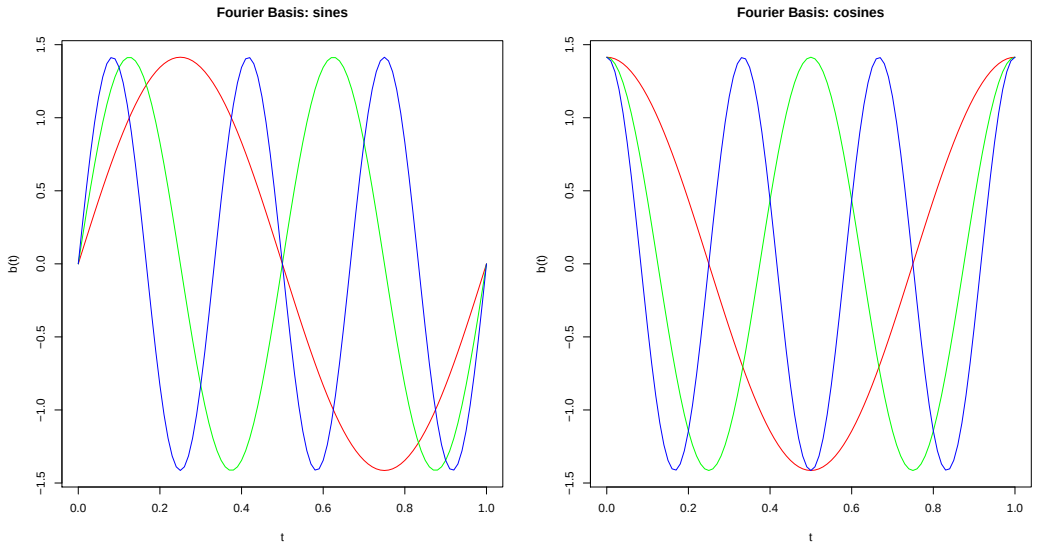

Figure 3.4: Fourier basis for $K = 3$, $D = 2K + 1$. Constant function $b_0(t)$ not shown.

3.6.2 Fourier Basis

The *Fourier basis* for a given dimension $D = 2K + 1$ is an orthonormal basis over the interval $[0, 1]$ containing sines and cosines with frequencies $1, \ldots, K$ as well as a constant function. That is, the continuous Fourier basis contains the $2K + 1$ basis functions:

$$\begin{aligned} b_0(t) &= 1 && t \in [0,1] \\ b_{2f-1}(t) &= \sin(2\pi f t) && f = 1, \ldots, K,\ t \in [0,1] \\ b_{2f}(t) &= \cos(2\pi f t) && f = 1, \ldots, K,\ t \in [0,1]. \end{aligned}$$

A picture of the sines and cosines in the Fourier basis for $K = 3$ is shown in Figure 3.4. Because we have discrete data in our application, for a class defined by a maximum flight length $T_{\max}$, the continuous Fourier functions are sampled at times $t_i = (i-1)/(T_{\max}-1)$

where $i = 1, \ldots, T_{\max}$, to form the discrete Fourier basis used for that class. The more basis functions D that are used, the higher the frequencies in the data that can be accurately represented. In our particular application, $D = 25$ is sufficient to capture the observed frequencies.

3.6.3 Wavelet Basis

The Daubechies wavelets [Jensen, 2001] are a family of orthogonal wavelets characterized by a maximal number of vanishing moments for given support. Each wavelet has a number of zero moments or vanishing moments equal to half the number of coefficients. A vanishing moment limits the wavelet's ability to represent polynomial behavior or information in a signal. Daubechies wavelets are widely used in solving a broad range of problems, e.g. self-similarity properties of a signal or fractal problems, signal discontinuities, etc. Unlike the Fourier basis functions, wavelet basis functions are localized in space.

Of course, we want to reduce the number of basis functions down to a manageable level. As for our PCA basis, our choice of the best D basis vectors to use is dependent on the training data. We first fit every training example $\mathbf{x}^i$ in a given class to compute the wavelet coefficients c_{jk}^i. Then we compute the importance η_{jk} of the wavelet basis function ψ_{jk} by accumulating the squared coefficient values over all the training examples: $\eta_{j,k} = \sum_i (c_{jk}^i)^2$. The more a basis functions ψ_{jk} is needed to accurately represent the training data, the larger the value of η_{jk}. Thus we select the D basis functions with the largest importance score η_{jk} to be the (reduced) wavelet basis

for the given class of output curves.

3.7 Multiple Outputs

Thus far we have discussed predicting the curves for a single output variable. But computer models may have multiple outputs, and, in fact, the NASA flight simulator has 12 output variables. The complete NASA flight simulator data set $\mathbf{y}(v, \mathbf{x}, t)$ is indexed by output variable v, simulator input parameters $\mathbf{x} \in \mathbb{R}^{11}$, and output time t. There may be correlations over the output variables v which can be used to improve prediction results over all variables.

As for the single curve case $\mathbf{y}(\mathbf{x}, t)$, we could view the problem as modeling a single scalar-valued function by appending time t and output variable v to the input parameters $\mathbf{x}$ and design a correlation structure that accounts for correlations over v. Another possibility is to jointly model the 12 output functions $\mathbb{R}^{11} \to \mathbb{R}^T$, where once again a non-diagonal covariance structure that captures correlations among the output variables could be used to improve prediction. As in the single output curve case, we suggest a simpler possible solution that allows independent prediction. The idea is to use PCA to transform the output variables so that correlations among the curves for the transformed variables is minimized, then predict independently on the decorrelated variables using our proposed single output variable model, and finally transform back to the original variables to obtain the final predicted curves. In this approach we essentially split the modeling task into two pieces: data decorrelation and independent prediction

instead of a single joint prediction model that accounts for correlation among the output variables. The proposed approach for effective handling of multiple output variables has the practical advantages of being simpler and allowing prediction to be done in parallel for different output variables.

3.8 Results

In this section, we report results of our method for output curve prediction using perfect (known) classification. In order to relax this assumption, we are using the hierarchial model shown in Figure 3.2 using CTGP classification prior to determination of time-to-failure or curve prediction. Since both questions are highly interrelated, we present those results after discussing our approach to detect time-to-failure in the next chapter, Section 4.2.

Let $\mathbf{x}_i$ denote the test inputs with true output curves $\mathbf{y}_i \in \mathbb{R}^{T(\mathbf{x}_i)}$ and predicted output curves $\mathbf{y}_i^{\text{pred}} \in \mathbb{R}^{T(\mathbf{x}_i)}$. We use the standard deviation σ_i for the true output curve $\mathbf{y}_i$ to standardize errors across different output curves and different output variables. The error $e_{v,c}$ for output variable v and class c with the set of test output curves $S_{v,c}$ is given by

$$e_{v,c} = \frac{\sum_{i \in S_{v,c}} \frac{||\mathbf{y}_i^{\text{pred}} - \mathbf{y}_i||_1}{\sigma_i}}{\sum_{i \in S_{v,c}} T(\mathbf{x}_i)}. \tag{3.5}$$

The true and predicted output curves $\mathbf{y}_i$ and $\mathbf{y}_i^{\text{pred}}$ are for the given output variable v, but we leave out the dependence on v to simplify the error formula 3.5. The sums of 3.5 are over the total number of output curve values predicted, thus making the reported

	pad				truncate + prior			
outvar	*failure* 1	*failure* 2	*failure* 3	*success*	*failure* 1	*failure* 2	*failure* 3	*success*
1	0.480	0.513	1.280	0.404	0.525	0.536	1.117	0.403
2	0.687	0.458	0.907	0.449	0.737	0.428	0.758	0.439
3	0.280	0.342	0.649	0.514	0.277	0.297	0.686	0.498
4	0.516	0.369	1.036	0.480	0.548	0.396	0.990	0.469
5	0.709	0.610	1.268	0.530	0.604	0.562	1.033	0.517
6	1.235	1.080	0.493	0.446	1.037	0.886	0.506	0.440
7	0.135	0.102	0.150	0.687	256.411	49.443	43.411	23.513
8	0.246	0.153	0.719	0.167	0.617	0.204	0.761	0.192
9	0.387	0.578	1.023	0.169	0.475	0.551	1.009	0.185
10	0.368	0.510	0.909	0.320	0.439	0.527	0.764	0.316

Table 3.3: Output curve prediction errors using curve padding versus a combination of basis truncation with a zero-mean Normal prior on the coefficients. The PCA basis of dimension $D = 25$ is used. The shrinkage parameter $\sigma_c^2 = 10$ is used. See the text for details of the error measure.

error an average over all predicted points.

To test the effectiveness of the shrinkage idea as a way to avoid padding the curve data (the third option described in Section 3.5), we did an experiment using each of the bases with dimension $D = 25$ with all the coefficients c_i having the same prior $c_i \sim N(0, \sigma_c^2)$ for all the output variables. We predicted the output curves from the first ten output variables using 4 classes. The results for the PCA basis for $\sigma_c^2 = 10$ are shown in Figure 3.3. Results for the other bases were comparable.

Except for output variable 7 (with function values near 20000) where the shrinkage approach fails, the resulting prediction errors are quite similar between the padding and shrinkage cases. Because of the difficulties with variable 7 (shared by all the bases), we use the padding approach for the rest of this chapter.

Table 3.4 shows that for our 4-class model, all three types of bases are suitable

to represent the curves. This comparison assumes that the actual (perfect) classification of each run into one of the three failure classes or success is known. Figure 3.5 shows how the prediction error differs between the different output variables and different bases. The predictors' performance is worst for failure class 3, because failure class 3 contains by far the fewest number of training data. The PCA basis gives the best overall performance, and this result was consistent across other divisions of training and test datasets. Thus we focus on PCA going forward. Some representative predictions using the PCA basis with dimension $D = 25$ for output variable 8 can be seen for the 3 failure classes and the success class in Figure 3.6.

	failure1	*failure2*	*failure3*	*success*
Daubechies wavelets				
mean	0.5402	0.5132	0.8498	0.3389
std	0.4019	0.3501	0.3289	0.1455
Fourier				
mean	0.5613	0.5052	0.8244	0.4066
std	0.3577	0.3113	0.3357	0.1527
PCA				
mean	0.4647	0.4508	0.7839	0.3184
std	0.3290	0.2927	0.3379	0.1497

Table 3.4: Prediction error (mean and std) for 4 class model and different bases with perfect classification.

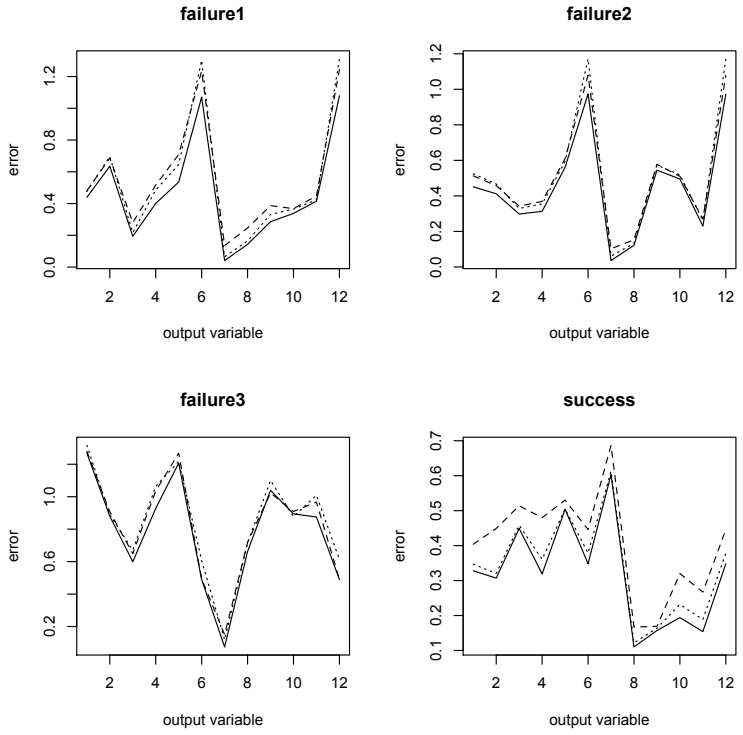

Figure 3.5: Output curve prediction errors for different bases, output variables, and classes in the 4 class strategy. The basis dimension is D = 25 for all bases (PCA: solid line, wavelet: dotted line, Fourier: dashed line)

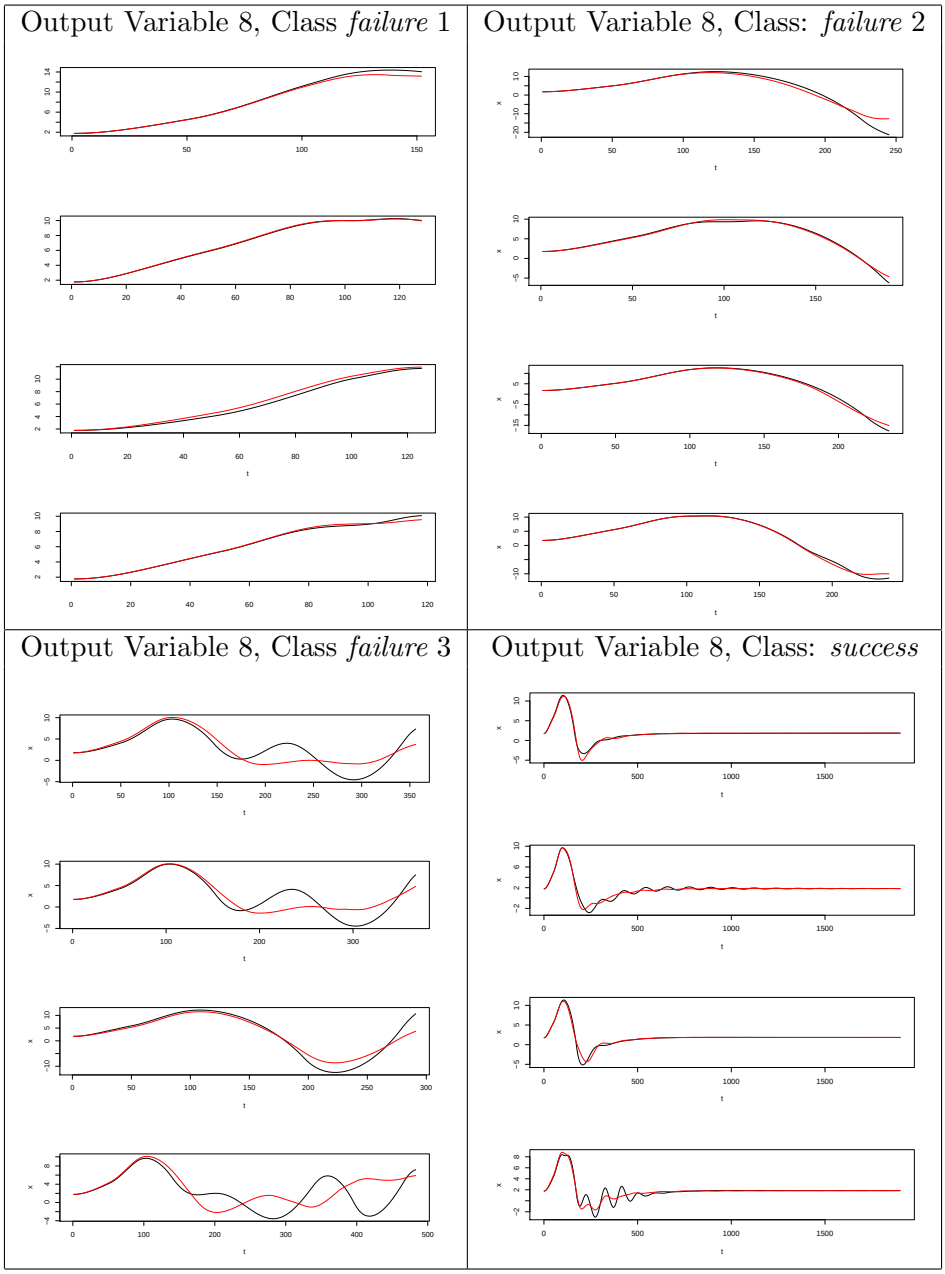

Figure 3.6: $D = 25$ PCA-based predictions: Output variable 8: (upper left) class *failure* 1, (upper right) class *failure* 2, (lower left) class *failure* 3, (lower right) class *success*.

Chapter 4

Predicting Flight Length

In the last chapter, we discussed how statistical methods can be used to predict the output curves of a complex system, in our case, the IFCS adaptive flight control system. One important question, which always must be asked within the realm of a safety-critical system, is, how long the aircraft, given a specific set of input parameters, is able to fly. In the IFCS simulation, such a failure is characterized by a failure to complete the full simulation time of 20 seconds. The question is therefore: can the time-to-failure T_{fail} be predicted in a reliable way, given a set of input parameters. In this chapter, we present our Bayesian approach for the prediction of flight length, again using techniques of Gaussian processes and Treed Gaussian processes, which have been discussed in detail in the previous chapter. We will also discuss the implications of a wrongly predicted failure case ("false alarm") or the case, where a failure was not predicted. These metrics are commonly used in the area of system safety. For this task, we will again use our hierarchical model, which first learns a classification using CTGP

as discussed in the previous chapter. Since curve prediction and prediction of flight length use the same classification results, we will present results for curve prediction using CTGP classification in this chapter as well.

4.1 Methodology

We implemented a Bayesian approach for the prediction of Flight Length, i.e., output variable length T, by sampling $T|\mathbf{x}$, then $\mathbf{y}|T, \mathbf{x}$ under a statistical model with appropriate priors, for example using TGP. So we can obtain a posterior distribution for output curve $\mathbf{y}$ using

$$p(\mathbf{y}|\mathbf{x}) = \int_T p(\mathbf{y}, T|\mathbf{x})dT = \int_T p(\mathbf{y}|T, \mathbf{x})p(T|\mathbf{x})dT.$$

The strategy for sampling $\mathbf{y}|T, \mathbf{x}$ would use the basis representation framework described in the sections above, using TGP or a Bayesian SVM model to learn the mappings from $\mathbf{x}$ to basis coefficients c_i, given that we know the output curve length T, then draw samples $c_i|T, \mathbf{x}$ from which we obtain samples $\mathbf{y}|T, \mathbf{x}$. In this section we will discuss the prediction model for T. The samples for $T|\mathbf{x}$ are obtained by using an appropriate model, to learn the mapping $\mathbf{x} \to T(\mathbf{x})$. The models we have implemented include TGP and SVM models which we will discuss in details as follows. We first used 867 output flight lengths as training data, and 100 as test data. The first model we tried uses Treed Gaussian Process to model the relationship between the 11 inputs and the flight length variable as output. The second model we tried uses a Support Vector Machine to learn the mapping between inputs and the flight length. SVMs use

an implicit mapping of the input data into a high-dimensional feature space defined by a kernel function, i.e., a function returning the inner product between the images of two data points in the feature space. The learning then takes place in the feature space, and the data points only appear inside dot products with other points. In all the models above, we have examined both strategies of predicting flight length without clustering, and prediction of T within each cluster. Results will be discussed below.

4.2 Results

4.2.1 Flight Time Prediction Using CTGP Classification

We have examined classification strategies using not just two classes (success and failure) but also four classes, with classes defined by the ranges of the output length. Our experimental data set consists of a total of 967 runs obtained from the simulation. Table 3.1 on page 26 summarizes the results for our two different classification strategies. For curve prediction, we randomly split the 967 runs into 867 training examples and 100 test examples. The CTGP 2-class and 4-class classification results for the prediction training and test sets are shown in Table 4.1 (a) and (b), respectively.

In addition to choosing the number of classes, we also performed experiments to determine the best classifier for the data. We split the 967 runs into 637 training examples and 300 test examples to evaluate classifier performance. We implemented five different classification methods, as a whole, the TGP-based classification methods gave the lowest classification error rates of 11.8%. For the 2-class case, the 867 training

	failure	*success*	total
training	507	360	867
test	62	38	100
# errors	3	8	11
error rate	4.5%	21.1%	11.8%

(a)

	failure 1	*failure* 2	*failure* 3	*success*	total
training	228	213	66	360	867
test	29	28	5	38	100
# errors	5	9	5	8	27
error rate	17.2%	32.1%	100%	21.1%	27%

(b)

Table 4.1: CTGP classification performance on the curve prediction training and test sets. (a) 2-class. (b) 4-class. See the text for details.

examples have 507 examples in the *failure* class and 360 in the *success* class. The CTGP overall 2-class error rate is very low at 11%, making 11 incorrect classifications out of 100 tests.

In Table 4.2 we have a further breakdown for the types of errors made by the CTGP classification algorithm. For the 2-class problem, the error rate was just 4.5% for classifying *failure* cases, incorrectly classifying only 3 of the 62 true *failure* tests. CTGP incorrectly classified 8 of the 38 test *success* runs as *failure*, for a *success* error rate of 21.1% — this type of error would correspond to a 'false alarm' if the algorithm was used in real time to try to predict failures. We have plans for how to bring down the overall false alarm rate.

The number of examples in the training and test sets for the 4-class case, as well as the classification error rates per class and overall are reported in Figure 4.1 (b).

The overall CTGP classification error rate is 21%, making 21 incorrect classifications out of the 100 tests. Although the overall error rate is higher for the 4-class problem than the 2-class problem, the error rate for the *success* class is lower. In the 4-class case, CTGP misclassified only 2 of the 38 *success* tests, for a *success* error rate of just 5.3%. The *success* class had more training examples than the other 3 classes. The *failure* 1 and *failure* 2 error rates were 17.3% and 32.1%, respectively. All 5 test cases in class *failure* 3 were classified incorrectly. Because the number of training data for this class is substantially lower than the other classes, this result is not surprising and should be disregarded. It is expected that the error rate will improve with more training samples.

	Two Class Problem	Four Class Problem
false alarm rate	21.1%	21.1%
missed failure rate	4.5%	30.6%

Table 4.2: False Positives and False Negatives Percentage

We computed the mean absolute prediction error from both TGP and Support Vector Machine (SVM) models. SVM is a commonly used technique for the prediction of nonlinear systems. We measured the average of the absolute difference $\bar{E}$ of the predicted time to failure T_{est} and the corresponding ground truth T over the 100 test runs as $\bar{E} = \frac{1}{N}\sum_{i=1}^{N}|T - T_{est}|$.

The first row of Table 4.3 gives the mean absolute prediction error from TGP and SVM without classification. Even TGP's prediction error of 368.7 is large given the range of time to failure. All of the other methods that we tried gave even worse results. For example, the average absolute prediction error for the next-best model after TGP

and SVM is 706.47. Since none of the methods give a good result when we fit a single model to the data globally, we clearly need another approach.

	TGP	SVM
all	368.7	481.8
failure only	33.9	42.5

Table 4.3: Mean absolute prediction error for time to failure using TGP and SVM

Our new method utilizes the classification strategy we described in the above sections. And we have concluded that the two class strategy is our preferred strategy, using CTGP we can accurately predict whether an input will result in a failure or success, which also provides useful information for improving the flight control system. We would like to go further to predict the actual time to failure. If CTGP predicts success for a given input $\mathbf{x}$, then we predict T as 1901 since all successful runs have 1901 time steps. If CTGP predicts failure for a given input $\mathbf{x}$, then we use another method to predict T. Our strategy in this case is to learn the mapping from $\mathbf{x}$ to $T(\mathbf{x})$ for failure runs only. In other words, we fit a model using only training runs that result in simulator failure. Among the 867 training inputs that we selected, there are 505 on which the simulator failed. We fit TGP and SVM models on these 505 failure training runs. Of the 100 test inputs, 62 failed. We tested the prediction of TGP and SVM trained on the failure runs to predict the time to failure for these 62 tests. In the second row of Table 4.3, we present the mean absolute prediction error for both TGP and SVM over the 62 failure test cases. For both models, the prediction error improved an order of magnitude using our classification strategy over training and predicting

without classification (first row). Once again TGP outperformed SVM. Furthermore, now we have a good prediction method using classification strategy and TGP model. The average absolute prediction error of TGP on the failure test runs is small compared to the range of time to failure.

Figure 4.1 and Figure 4.2 give a summary of the 100 test results. The results for TGP are shown in Figure 4.1 and the results for SVM are shown in Figure 4.2. The 62 tests which failed are on the left of each plot; the independent axis is the test number and test numbers 1-62 are failure cases. Similarly, the 38 cases which succeed are on the right of each plot. Each data point in the plot is a prediction of the time to failure. Means for the failure and success times are given as dashed lines. TGP results in predictions that are much closer to the true mean failure times. What's more: TGP has an overall lower misclassification rate, and only misclassifies one true failure.

4.2.2 Output Curve Prediction Results Using CTGP Classification

In this section we work with the more realistic setting of not knowing the classification in advance, and we give curve prediction results using CTGP to determine which output class model is applied (Table 4.1). Examples of predicted curves for output variable 8 are shown in Figure 4.3. In these figures, the black curve is the ground truth curve and is plotted for the correct number of time steps $T(\mathbf{x})$ for test input $\mathbf{x}$. The red (lighter) curve is the predicted curve using the model for the correct class $C(\mathbf{x})$, and it is plotted for the maximum length $T_{\max}^{C(\mathbf{x})}$ that defines the class $C(\mathbf{x})$. If CTGP incorrectly predicted the class, then the blue (darker) curve is the predicted curve using

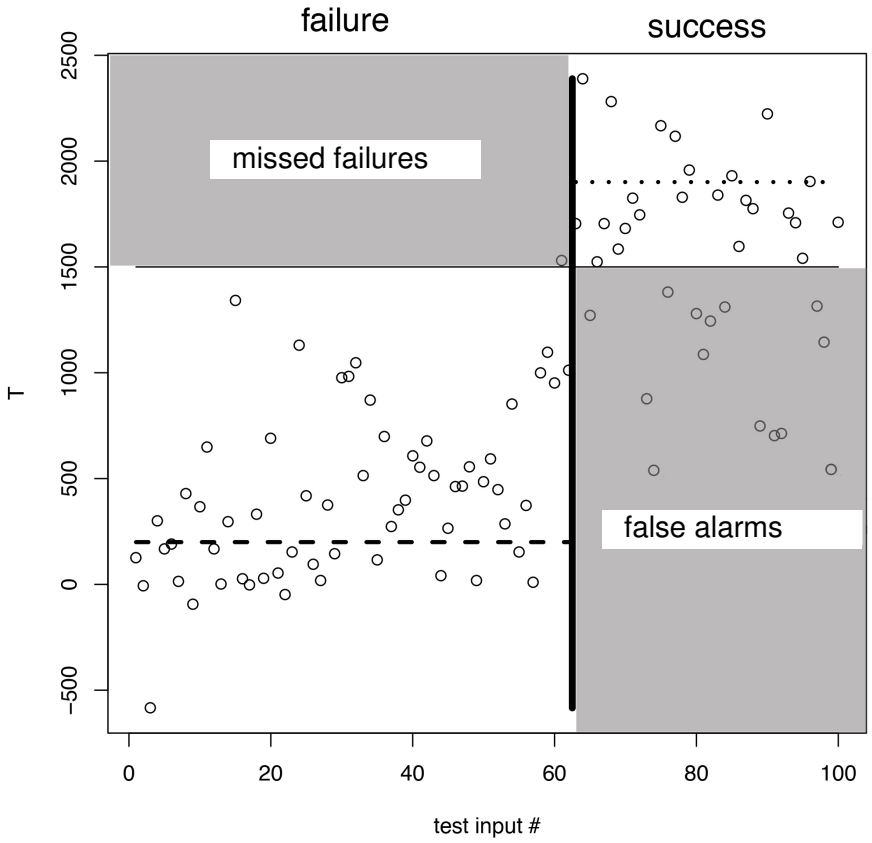

Figure 4.1: Prediction of time to failure with TGP.

the model for the incorrectly predicted class $C^{\text{pred}}(\mathbf{x})$ and it is plotted for the maximum length $T_{\text{max}}^{C^{\text{pred}}(\mathbf{x})}$ of that class. In the title for each test plot there is an indication of whether the CTGP classifier predicted the correct class or not, and the incorrect class prediction is given for classification errors.

The red (lighter) prediction curve for the correct class is always at least as long as the black ground truth curve because the true length $T(\mathbf{x})$ must be less than

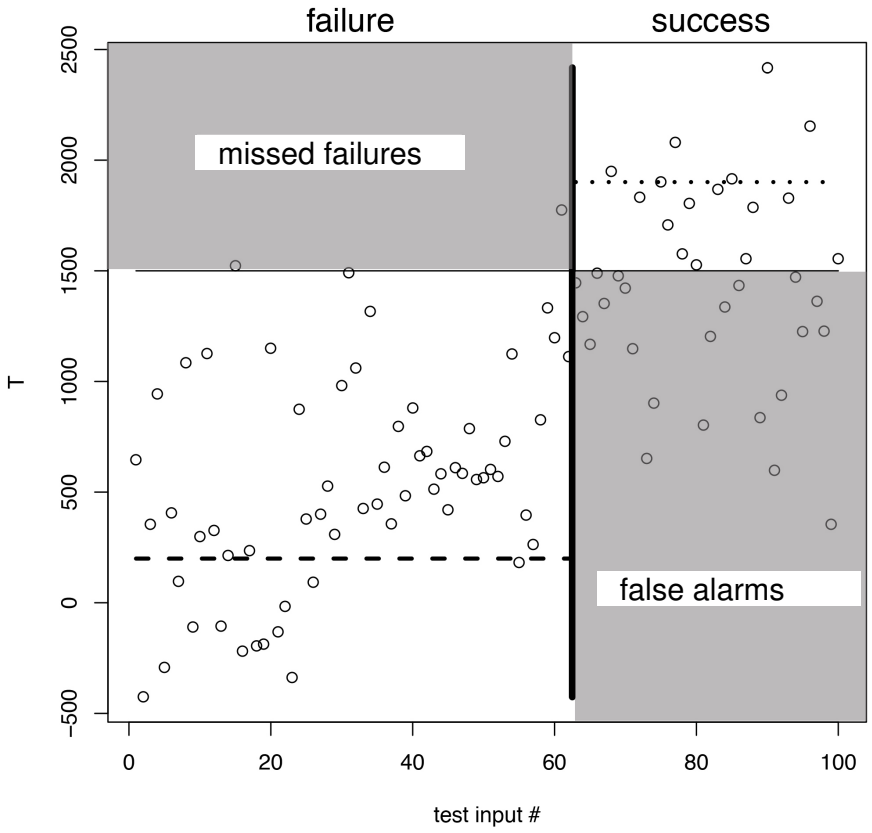

Figure 4.2: Prediction of time to failure with SVM.

or equal to $T_{\max}^{C(\mathbf{x})}$ in order for $\mathbf{x}$ to be in the class $C(\mathbf{x})$. (Note that this is different from the plots in Figure 4.3, in which we truncated the predicted curve to the known correct length $T(\mathbf{x})$.) For tests in which CTGP predicts the wrong class $C^{\text{pred}}(\mathbf{x})$, the predicted blue curve may be shorter or longer than the black ground truth curve. In the third example in the upper left of Figure 4.3, the correct class is *failure* 1 with $T_{\max}^{failure1} = 180$, but the input was incorrectly classified as *success* with $T_{\max}^{success} = 1901$

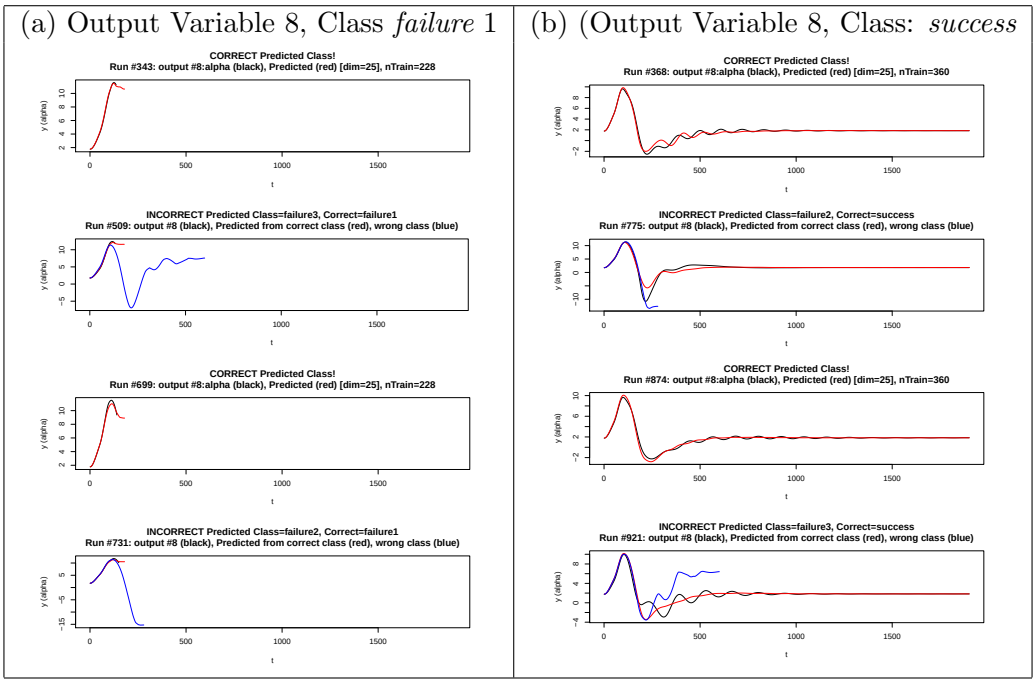

Figure 4.3: Predicted Output variable 8 curves using CTGP classification. $D = 25$ PCA-based predictions: (left) class *failure* 1 and (right) class *success*.

so that the predicted curve is much longer. In the last example in the lower right of Figure 4.3, the correct class is *success* with $T_{\max}^{success} = 1901$ but the predicted class was *failure* 3 with $T_{\max}^{failure3} = 600$ so that the predicted curve is shorter.

In Figure 4.3, we see more examples of excellent curve predictions using the correct output class predicted by CTGP. We also see something quite interesting in the examples with incorrect classification: the predicted output curve using the model from an incorrect class is typically quite good near the beginning of the flight run and often does reasonably well over a significant fraction of the true output curve. As expected, the predicted curve using the correct class is usually better than the one using the incorrect class. For some examples, see Figure 4.3(b1),(b3),(d4). There are some tests for which the predictions are very similar using the correct and incorrect classes, for example in Figure 4.3(a4). Finally, there are even a few tests in which the predicted output curve is slightly better using the incorrect class. Table 4.4 shows the prediction error when the class is considered unknown and predicted with CTGP. Error rates are quite similar to those in Table 3.4, showing that little is lost when estimating the class with CTGP.

	failure1	*failure2*	*failure3*	*success*
PCA with CTGP classification				
mean	0.4613	0.4875	0.7488	0.3350
std dev	0.2917	0.2784	0.2869	0.1436

Table 4.4: Prediction error (mean and standard deviation) for PCA and 4 class CTGP classification.

Finally, we turn to the full multivariate output problem from Section 3.7. Let

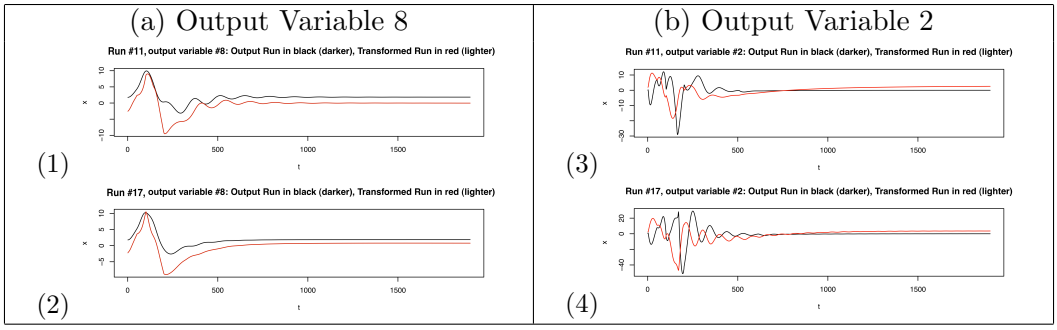

Figure 4.4: Predicted Output Curves for correlated multiple output curves

$y(i, v, t)$ denote the value of output curve v at time t for run i (for input x_i). Let Y_t be a matrix of size $12 \times N_t$, where N_t is the number of output curves that have length greater than t. Row number k of Y_t contains the row vector $[y(i_1, k, t)...y(i_{N_t}, k, t)]$ where $i_1, ..., i_{N_t}$ are indices of the run numbers. So Y_t contains output curve values at time t, and each row contains data from different output variables. Form $Y = [Y_1|Y_2|...|Y_{1901}]$ of size $12 \times N$, where N is the sum of all the N_ts. Each row contains all the curve values for output variable k. We performed PCA on Y to obtain a transformed matrix $\hat{Y}$, which gives a transformed set of curves $\hat{y}(i, v, t)$. Thus we generated a transformed set of curves for all output variables that are uncorrelated and for which the strategy of predicting each output variable independently makes more sense. Then we predict the transformed output variable curves independently and transform back to the original output curve space to get the final curve predictions for the original output variables. The results are presented in Figure 4.4. While our fits do not match exactly, they get quite close in shape and structure, and provide a sufficient fit for understanding the behavior of the controller.

Chapter 5

Boundary Detection

5.1 The Problem

All complex mechanical, electrical, or software systems can only operate safely within a given operational envelope. Such an envelope is spanned by multiple parameters, which can be design parameters, environmental parameters, or operational parameters. For example, a stable flight of an aircraft is limited by a minimal and maximal airspeed, depends on the altitude the aircraft is flying, and aircraft design parameters (e.g., drag-over-lift). In many disciplines, a wealth of knowledge and methods exist to determine such limits in an analytical manner or by selected experiments. Typically, such limits are provided as project requirements or are prescribed by applicable standards. However, there are limits to such analytical approaches: some systems can be of such a high complexity and governed by nonlinear and interacting physical laws that an analytical analysis of the system is not possible.

In particular, systems which are controlled by software often exhibit such behavior, because the multitude of possible execution paths through the software (e.g., different controller modes) and transients arising from discrete mode switches prohibit proper mathematical treatment. A third class of systems, which cannot be fully analyzed analytically are adaptive systems, like the IFCS adaptive flight controller (Chapter 2). Here, only generic and often rather weak and unsatisfying boundaries for proper and safe operation can be calculated analytically [Rysdyk and Calise, 1998]. For such cases, the *boundaries* for the appropriate envelopes must be estimated from actual measurements or by system simulation. The high-dimensional and noisy data, which are obtained from measurements or simulation are usually categorical data, which denote if a given safety property has been met or not (success or fail). Typical properties include: stability of the system ("Does the aircraft remain stable in the air?"), performance criteria ("Does the aircraft follow the pilot's stick movements smoothly?"), or a system failure ("Is the wing ripped off or does the engine explode?").

Obtaining new data points is slow and expensive, because for each specific parameter setting, a system simulation of experiment must be carried out. All or most components have to be considered as "black boxes", i.e., there's no information about their internal structure or their underlying design or operational principles. In many cases, the geometrical shape of the safety envelope is known to the design expert, because it is based upon certain physical laws and design principles. Some of the typical examples include: variable ranges, linear boundaries, boundaries of a quadratic shape, polygons, circles (or spheres), and ellipses. In most cases, the defining parameters for boundary

shapes must be estimated.

Typical examples include location and shape of the landing ellipse upon a spacecraft re-entry [Gundy-Burlet *et al.*, 2008]. Only if the spacecraft is landing close to the desired landing spot, this landing is considered a success. Due to orbital mechanics, the desired landing area, which is governed by a multitude of parameters is an ellipse. Another example directly concerns the IFCS adaptive flight control. The analytical proof of stability for the adaptive controller [Calise and Rysdyk, 1998; Rysdyk and Calise, 1998] guarantees eventual stability of the entire system as long as the errors remain within a ϵ boundary. Again this boundary depends of a number of gain and design parameters. Yet another set of safety boundaries are found in safe operational enveloped of aircraft. Standards like MIL-1797 [Department of Defense, 1997] define several levels of handling quality. If the handling quality of an aircraft is good, it is easy to fly and forgives many small errors. Aircraft with less handling quality are difficult to handle and can cause a security risk. Such boundaries can be physics-based: for example, the operational envelope of an aircraft depends on altitude, speed, design of the aircraft and available power. Figure 5.1(left) shows the safety boundaries for a commercial transport (taken from the final report of the ill-fated Air France flight AF447 [BEA, 2012]). The non-linearity of the boundaries are due to physical laws. Figure 5.1(right) shows areas of different handling qualities over a given set of parameters. In this standard, the boundaries are polygons.

Obviously, the analysis of the safety regions and envelopes must cover the entire state and parameter space according to a given coverage metric. "Holes" in an

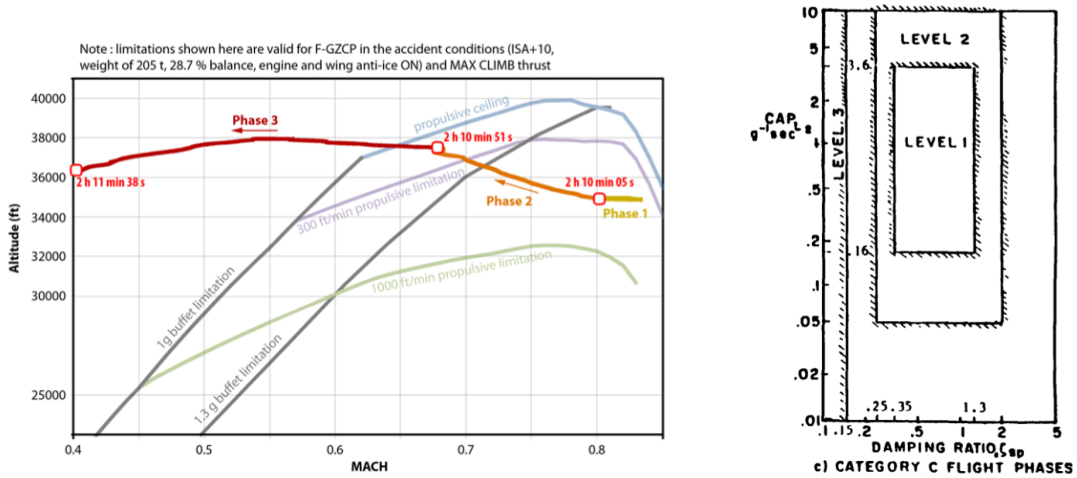

Figure 5.1: Safety and quality boundaries in Aeronautics

operational envelope can have disastrous consequences and must therefore be recognized properly. In particular, errors in a software system can lead to such dangerous situations. For example, when a group of six F-22 Raptors were flying to Okinawa, Japan, they experienced multiple computer crashes coincident with their crossing of the 180th meridian of longitude (the international date line) [F-22, 1992]. A software problem caused these issues. In this work, we will address the problem of identifying shapes of boundaries as an experimental design problem and will use powerful statistical algorithms to determine location and shape of the boundaries using active learning in order to minimize the number of required simulation runs.

5.2 Goals

When analyzing a high-dimensional parameter space of a complex system with respect to a given property (e.g., succeeds or fail to stabilize the aircraft), the analyst will be interested in getting answers to a number of questions. Most basically, the question arises if the entire parameter space has areas belonging to different classes in the first place. Failure to identify such regions could mean that the system performs uniformly well or poorly over the entire parameter space, or there are failure regions, which are too small to be detected. In many practical applications, boundaries of regions are approximated by ranges of individual parameters or sets of parameters. Most traditional engineering analyses work on individual variables only, not taking into account any correlations between different variables. The Margins tool, developed at NASA Ames uses treatment learning [Menzies and Sinsel, 2000; Menzies and Hu, 2003] to identify sets of parameter ranges, which comprise an approximation of boundaries as axis-aligned hyperboxes. A representation of boundaries in the form of hyper planes is more difficult to interpret by the analyst but can be achieved using, for example, SVM based techniques.

In the literature, learning about regions where the surface changes rapidly is called Wombling [Banerjee and Gelfand, 2006]. Curves with gradients orthogonal to the curve track a path through regions, where the surface is changing rapidly, thus marking curvilinear boundaries. [Banerjee and Gelfand, 2006] develops a statistical framework for curvilinear boundary analysis, which is based on spatial process models. This non-

parametric approach is well suited to spatial response surfaces and spatial residual surfaces. A curvilinear representation of boundaries is of particular use, if the exact boundary is very rugged and contains sharp edges, as typically found in boundaries on geographical maps. However, our intended appplications feature smooth boundaries, suggesting a different form of representation.

A numerical approximation of a boundary using a piecewise linear function, splines, or other universal function approximators has only limited usability, because the overall shape and location cannot be represented in closed form and thus is very hard to interpret manually. Potential applications of such a boundary representation might be found in the area of monitoring, vehicle health management, or prognostics, where an algorithm has to decide (based upon parameter and sensor readings) if the current point of operation is inside a certain region or not. The analysis tasks discussed above are generic and do not use any knowledge available from the domain expert or analyst. During safety- or performance analysis, the actual shape of the boundary is known, as it is based upon some physical laws or design decisions. However, such an analytical shape can often be obtained only under strong simplifying assumptions, or only taking a single component of a system into account. So an important analysis task is to determine if the actual boundary is located in the same area and location as expected from the original requirements. For example, an analysis might require to determine, how the boundary of the minimal stable airspeed of an aircraft (a boundary as shown in Figure 5.1) moves, when a part of the wing is damaged; something that can be established through simulation. In such cases is to estimate the parameters for

a given boundary shape that best describe the boundary.

Finally, the shape of a boundary itself can change. For example, in a simplified model might have a linear boundary. However, due to unmodeled effects or implementation details, the boundary might be linear only in a small area of the parameter space and deviates into a shape similar to a logistic function in the other regions as certain saturation effects take place. Operational points in the "wedges" between the linear (theoretical) boundary and the actual boundary can cause severe problems. For the analyst it is therefore important to determine which shape the boundary has (selected from a small number of shape candidates), and which the most likely parameters are. Many well-known analysis techniques in aircraft design are based upon linear analysis. Advanced aircraft design and control techniques like the IFCS damage adaptive aircraft control exhibit strong non-linear effects that require novel analysis techniques. In this work, we therefore will focus on addressing the latter two analysis questions, namely the determination of most likely boundary shapes from a dictionary of shapes and to determine parameters that best fit the boundary.

5.3 Methodology

The given task clearly belongs to a class of classification problems. In the following, we formulate our methodology using classification nomenclature. In particular, we note that points with a high entropy correspond to points near a boundary. For the active learning, however, it turned out that a consideration of the problem as a

regression problem provides a better handle on the problem. We represent our task as using regression to learn and represent the response surface for the function f, where $f(x) = 1 + \epsilon$ if the experiment succeeds and $f(x) = 0 + \epsilon$ otherwise. In this representation a boundary is determined by points x with $\hat{f}(x) = 0.5$. This representation allows us to formulate a powerful method to select the next data point.

5.3.1 Algorithm Overview

We are developing a sequential method for the estimation of parameterized boundary shapes in high dimensional spaces. A dictionary of m shape classes $M_1, \ldots, M_m$ with $m \geq 1$, which are parameterized by $\Theta_1, \ldots, \Theta_m$ is provided by the domain expert. Additional constraints on the parameters, e.g., parameter ranges and other prior information can be given. Typical examples for such shape classes include (hyper-)surfaces, polygons, spheres, or ellipses. For our algorithm, we assume that the different shapes to be considered are behaving sufficiently different in the parameter space. Such a constraint can easily given in the form of a prior, e.g., by providing an upper bound on the radius of a curve to be able to distinguish it from a straight line. Given an initial set of labeled data D_0, the algorithm iteratively determines the most likely shape and its parameters and confidence intervals. The overall process is depicted in Figure 5.2. The active learning algorithm builds an initial classifier based upon D_0. Then, candidate points (i.e., sets of input parameters) are selected by the algorithm and handed over to the computer experiment, in our case a Matlab/Simulink simulation system. This system runs a version of the flight control simulator using the given parameters and

returns a categorical result (success or failure). In our implementation, the coordination between the boundary detection algorithm implemented in R and Matlab/Simulink is accomplished using the `R.Matlab` package [Bengtsson and Riedy, 2012]. Since each run of the simulator requires a substantial amount of computational resources (several seconds), the overall number of new data points should be kept as small as possible.

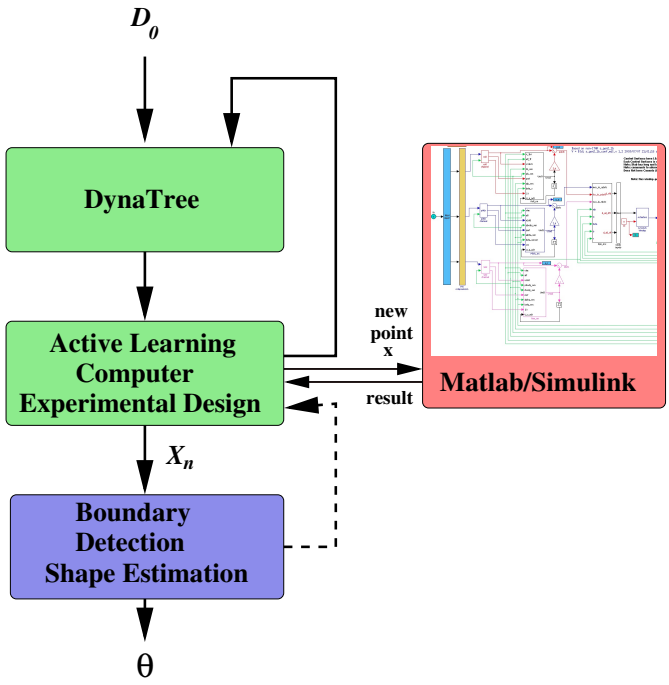

Figure 5.2: Overview of active learning procedure

More specifically, the algorithm is based upon the sequential classification and regression framework as given by the DynaTree [Taddy *et al.*, 2011; Gramacy, 2007] package. It features dynamic regression trees and a sequential tree model. Particle learning for posterior simulation makes Dynatrees a good candidate for applications, where new data points are processed sequentially. We will discuss Dyntrees in Sec-

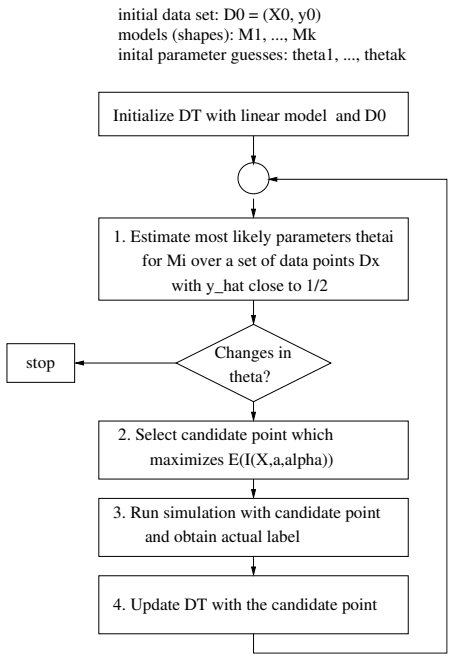

Figure 5.3: Overview of active learning procedure

tion 5.4.1. At any given point in time, the classifier is represented by a Dynatree. Figure 5.3 shows the individual steps of the algorithm. In the initial phase, a classifier using the data set D_0 is constructed. It provides an initial partitioning of the space and provides the information to estimate posteriors over given sets of data points. The main body is an iterative loop where, by adding new data points, the classifier will be extended and improved with the main goal of identifying and characterizing the boundaries. This is accomplished by the tree steps in the algorithm shown in Figure 5.3. In the first step, the current classifier is used to estimate a set of data points, which are close to current prediction of the boundary. These are a subset of data points from a regular grid or a Latin hyper square, for which their entropy measure is high (classifi-

cation representation) or the estimated response value is close to 0.5. The location of these points do not only depend on the actual boundary, but also on the shape of the dynamic tree and the size of the partitions, because points in the same partition have the same values. This set of data points is then used to estimate the currently best parameters Θ for each of the the boundary shapes, together with a confidence interval for each of the parameters.

The candidate point selection in this active learning algorithm can use as much information as is available at the current stage, for example, prior information given by the domain expert. It then selects a new point (i.e., set of input parameters), for which the label is obtained by running the Simulink simulator. Since running the simulator is usually slow and computational costly, a main goal of this selection process is to reduce the number of necessary new data points. For selection of the new data points we use a regression representation of our problem. In the following, we will present and discuss the individual steps in detail.

5.3.2 The Initial Classification

5.3.2.1 Sparse n-factor Combinatorial Exploration

The full combinatorial exploration of all possible values of the input parameters soon reaches infeasible numbers. We are therefore experimenting with two alternative methods of generation for the initial data set D_0: Monte Carlo testcase generation and n-factor combinatorial exploration. The Monte Carlo (MC) testcase generation treats each input variable as a statistical variable with a given probability density function, from

which values for the test cases are randomly drawn. In most applications, a uniform random distribution (within the given variable ranges) or a Gaussian distribution is assumed for all continuous inputs. Here, the mean usually is the nominal value. For discrete or discretized variables, a uniform probability distribution is assumed. Monte Carlo (MC) test cases can be generated very easily. However, they provide no guarantee whatsoever regarding uniqueness and coverage of the input space. This means that for a reasonable coverage of the input space, a very large number of MC cases have to be executed, again quickly reaching the limits on what can be done practically. Even with optimizations like discretization or binning of variables, pre-filtering of test cases, or exploitation of domain-specific features, like symmetry, only an overall probabilistic measure on the coverage of the input space can be given. In practice, MC testcase generation is usually used to quickly get an a reasonably dense coverage in the vicinity of nominal conditions.

Methods for systematic coverage of a high-dimensional state space originate from observations about software errors in Software Engineering: in real software programs, most errors are caused by a specific, single value of one input variable. A typical example (see above) is that the input "longitude" reaches the value of 180° causing problems in the navigation computer. The case that a fault is triggered by a specific combination of two variables is much less likely. Even more unlikely is the case that 3 input variables must have specific values in order to trigger the failure; the involvement of 4 or more variables can be, for most purposes, ignored. This observation (e.g., [Cohen *et al.*, Sep 1996; Dunietz *et al.*, 1997; Wallace and Kuhn, 2001]) can be used to specif-

ically tailor the generation of test cases, resulting in a substantially smaller number of test cases. Nevertheless, these cases completely cover all combination of variables up to a given bound n, hence the method is called n-factor combinatorial exploration.

A number of efficient algorithms for the n-factor combinatorial testcase generation can be found in the literature (e.g., [Grindal *et al.*, 2005]). For testcase generation with continuous variables as in our case, the input space for each dimension is split up into a number of discrete ranges (bins), from which values are uniformly drawn in a Monte Carlo fashion. For details see [Schumann *et al.*, 2009]. [Giannakopoulou *et al.*, 2011] compares full combinatorial exploration with n-factor for testing of a NASA air traffic control software (TSAFE). The full test set consists of 81 million test cases; a 3-factor set has approximately 6,100 elements. Code coverage, however, only was reduced minimally, demonstrating the advantage of n-factor test coverage. For our experiments, we used a tool developed at JPL [Schumann *et al.*, 2009] which extends the IPO algorithm [Tai and Lie, 2002]. Although the tool features a number of extensions, we only use the core algorithm to generate 3-factor combinatorial permutations for the discretized (binned) input variables of the IFCS system. Table 5.1 illustrates the results for various number of input parameters and values of n. A repetition factor of 2 was used for all runs. All IFCS parameters have been discretized into $k = 5$ bins. Thus, a full combinatorial exploration would require $k^N = 5^{11} = 4.9 \times 10^6$ simulation runs.

	24 input parameters				
n=	2	3	4	5	24
Runs	125	967	6731	> 10000	5.96×10^{16}
T[s]	0.07	5.4	35.6		–
	11 input parameters				
n=	2	3	4	5	11
Runs	98	654	4064	23044	4.88×10^{6}
T[s]	0.03	0.34	24.8	1383	–

Table 5.1: Number of generated runs and execution time for n-factor

5.3.2.2 Minimal Coverage metric for initial data set

A certain minimal coverage of the space must be provided by the initial data set D_0. Due to the greedy nature of entropy-based candidate selection, boundaries in sparsely populated areas might be missed. In our approach, we are using n-factor combinatorial exploration to obtain a small D_0, which nevertheless provides coverage as D_0 contains all possible combinations of values, value pairs, value triples, up to n-tupels. When a boundary shape is defined as constraining k dimensions (e.g., a sphere in dimensions 1,3,4 of an 11-dimensional space), that boundary can be located if an n-factor combinatorial exploration is used with $n \geq k$, and the boundary shape spans at least two of the n-factor bins. In that case, we are sure that D_0 contains at least one data point belonging to class 1 and one to class 2, giving rise to partitions with elevated entropy.

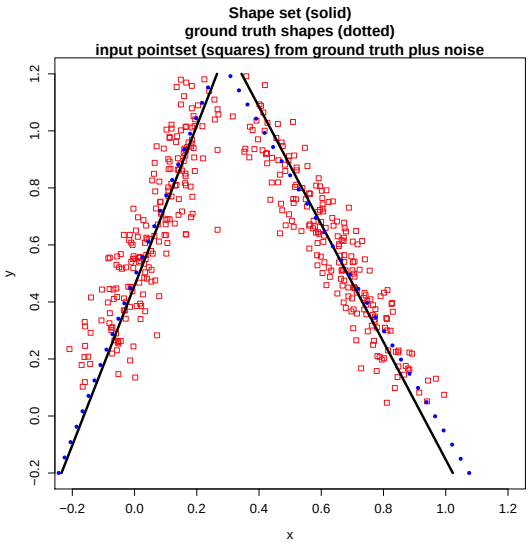

Figure 5.4: A good shape set $\mathcal{S}$ (black lines) for the input point set X_n (red squares).

5.3.3 Modeling Classifier Boundaries with Simple Shapes

Given a classifier P_n, we want to fit simple, parameterized shapes (from a dictionary provided by experts) to areas of high entropy that approximate the boundaries between two classes. We assume that a set of classifier boundary points X_n at step n of our adaptive boundary modeling is given. Figure 5.4 shows a good shape set $\mathcal{S}$ for an input point set X_n. The red points are the input point set X_n. The black lines are two shapes fit to the point set. This figure illustrates the goal of our shape fitting problem.

5.3.3.1 Notation

Suppose there are m shape classes $M_1, \ldots, M_m$ with $m \geq 1$ which are parameterized by $\Theta_1, \ldots, \Theta_m$. The task is to fit l shapes $S_1, \ldots, S_l$, $l \geq 1$, where $S_1 =$

$(i_1, \Theta_1), \ldots, S_l = (i_l, \Theta_l)$ and i_j denotes the shape class for the j^{th} shape with $i_j \in \mathcal{M} = \{M_1, \ldots, M_m\}$. Several of the i_j can be the same to accommodate more than one shape belonging to the same class. The Θ_i should be different since we do not want to represent the same boundary shape twice. We also seek to determine the correct number of shapes l that represents the input point set X_n.

For example, we may consider the $m = 2$ shape classes $M_1 = \textit{hyperplane}$ and $M_2 = \textit{sphere}$ in R^d. Hyperplanes are represented as $a_1x_1 + \cdots + a_dx_d + a_{d+1} = 0$ with parameter vector $\Theta_1 = (a_1, \ldots, a_d, a_{d+1}) \in R^{d+1}$. In the same d-dimensional space, a sphere of radius r with center $c = (c_1, \ldots, c_d)$ is described by $(x_1 - c_1)^2 + \cdots + (x_d - c_d)^2 = r^2$ with parameter vector $\Theta_2 = (c, r) \in R^{d+1}$. Now suppose we are in the plane ($d = 2$) and that the true class boundaries are described by the line $5x + y - 0.1 = 0$ (a hyperplane in R^2), the circle $(x - 0.3)^2 + (y - 0.4)^2 = 0.2^2$ (a sphere in R^2), and the vertical line $x + 0y - 0.7 = 0$. This is represented in our model as $l = 3$ with the specific shapes $S_1 = (i_1 = \textit{hyperplane}, \Theta_1 = (5, 1, -0.1))$, $S_2 = (i_2 = \textit{sphere}, \Theta_2 = (0.3, 0.4, 0.2))$, and $S_3 = (i_3 = \textit{hyperplane}, \Theta_3 = (1, 0, -0.7))$.

5.3.3.2 What is a Good Shape Set $\mathcal{S}$ for an Input Point Set X_n?

There are three conditions that specify when a shape set $\mathcal{S}$ provides a good fit to the data X_n:

(i) *Summary*: each point on a shape $S \in \mathcal{S}$ is close to some classifier boundary point in X_n,

(ii) *Completeness*: each classifier boundary point in X_n is close to some shape point on one of the shapes $S \in \mathcal{S}$, and

(iii) *Minimality*: the shapes in $\mathcal{S}$ are as different from one another as possible.

Let us now explain why the above properties are desirable in a fitted shape set. These properties are illustrated in Figure 5.5

Condition (i) encourages each shape $S \in \mathcal{S}$ to be a good *summary* of one of the parts of the boundary of classifier P_n. That is, the points of a shape should lie along high entropy areas of P_n. The shape in the top-left of Figure 5.5 is *not* a good summary of any part of the input point set. The shape in the top-right of Figure 5.5 is a good summary of the points on the left side. Condition (ii) encourages $\mathcal{S}$ to be a *complete* summary of the boundary input points. In a complete summary $\mathcal{S}$, each classifier boundary point is "covered" by $\mathcal{S}$ in the sense that it is close to a point in $\mathcal{S}$. The shape set in the top-right of Figure 5.5 is *not* a complete summary because it does not cover the points on the right side. On the other hand, the shape set in the bottom-left of Figure 5.5 is a complete summary. Condition (iii) encourages that shape set $\mathcal{S}$ to be *minimal*; i.e., $\mathcal{S}$ will not use any extra shapes to form a complete summary of the boundaries of classifier P_n. A complete summary $\mathcal{S}$ (i.e. one satisfying (i) and (ii)) remains a complete summary if one of its shapes $S \in \mathcal{S}$ is added to $\mathcal{S}$ either exactly or after a small perturbation. In fact, adding a small perturbation $\widehat{S}$ of S may actually improve completeness slightly since $\widehat{S}$ can be even closer to some high entropy points than S. And if S were a good summary, then so too would $\widehat{S}$ since it is close to S and

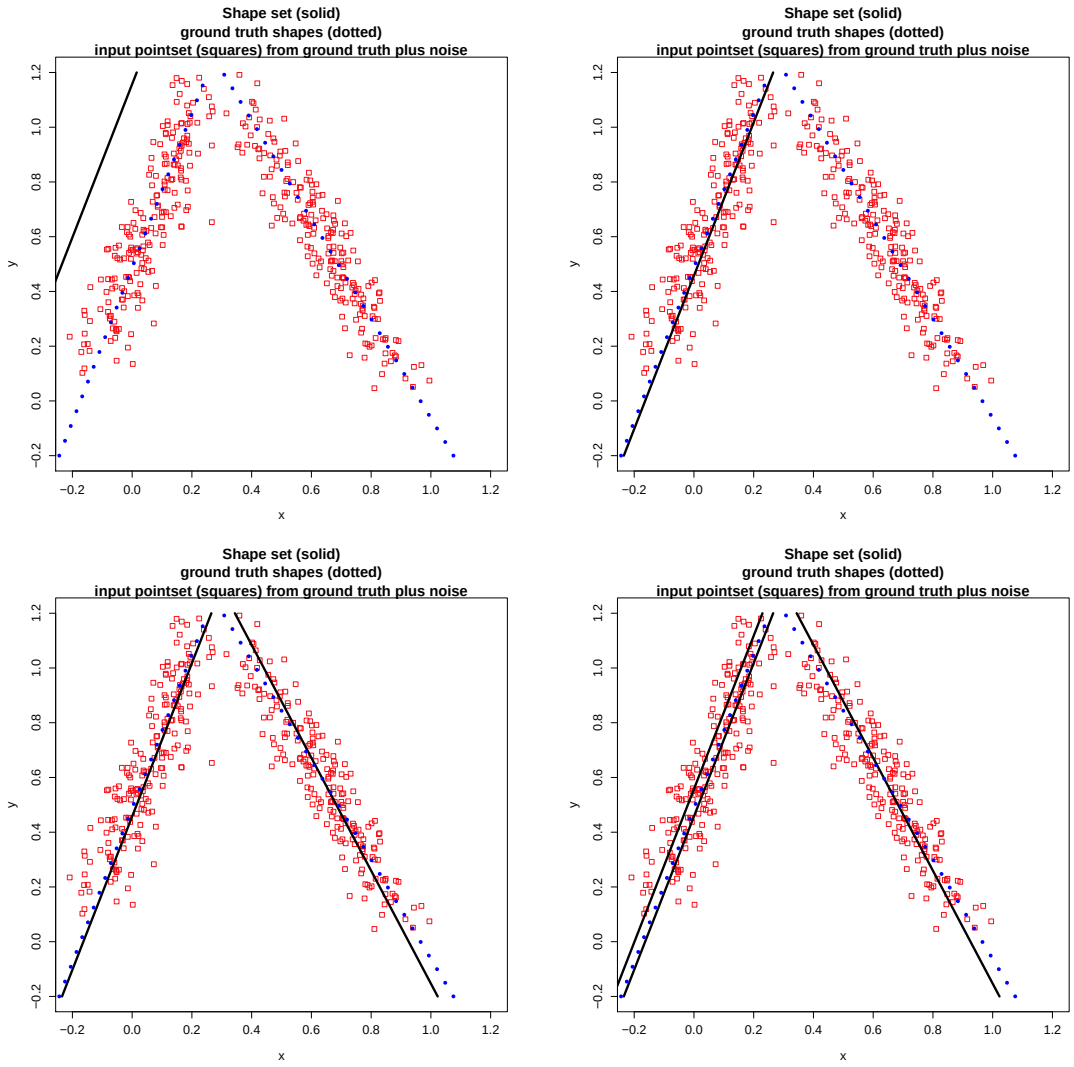

Figure 5.5: (top-left) The shape is a poor summary of any part of the input point set. (top-right) The shape is a good summary of the points on the left. But this shape set with one shape is not a complete summary of the point set. (bottom-left) This shape set is a complete, minimal summary of the point set. (bottom-right) This shape set is a complete summary, but it is not minimal.

points on S are close to high entropy points. We need the minimality condition (iii) to be able to obtain the simplest (i.e. smallest) shape set that is a complete summary of the classifier boundaries. The shape set in the bottom-left of Figure 5.5 is minimal, but the shape set in the bottom-right is *not* minimal.

5.3.3.3 Statistical Modeling

The shape set posterior is given by

$$P(\mathcal{S}|X_n) = \frac{P(X_n|\mathcal{S})P(\mathcal{S})}{P(X_n)} \propto P(X_n|\mathcal{S})P(\mathcal{S}).$$

We build the posterior model $P(\mathcal{S}|X_n)$ by modeling the likelihood $P(X_n|\mathcal{S})$ and the shape set prior $P(\mathcal{S})$. In the posterior $P(\mathcal{S}|X_n) \propto P(X_n|\mathcal{S})P(\mathcal{S})$, we will model the likelihood $P(X_n|\mathcal{S})$ to encourage *completeness* and the prior $P(\mathcal{S})$ to encourage distance between shapes and therefore *minimality*. It makes sense that the data likelihood accounts for completeness because completeness requires observed points to be close to a shape and the observed points arise from the ground truth shapes with the addition of noise.

We will encourage good *summary* using a Bayesian loss function that increases with increasing distance of the shapes to the point set. We will determine the number of shapes l by minimizing the expected posterior loss.

Likelihood Our likelihood will encourage completeness. For the completeness condition (ii), we are interested in making the average squared distance $\overline{D}^2_{X_n,\mathcal{S}}$ of boundary

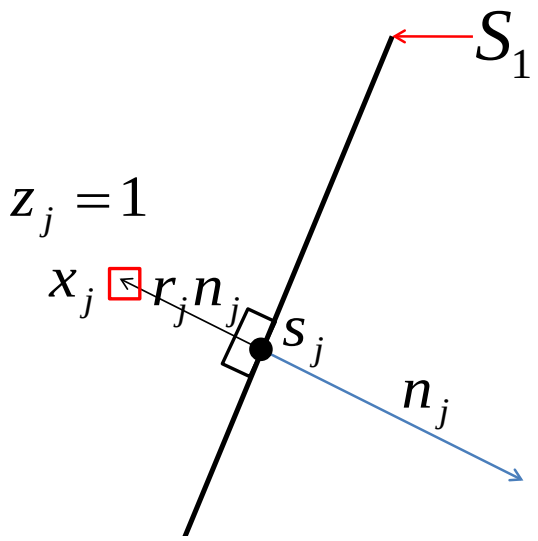

Figure 5.6: Likelihood $P(X_n|\mathcal{S})$ concepts and notation.

points in $X_n = \{x_1, \ldots, x_n\}$ to shapes in $\mathcal{S}$ small:

$$\overline{D}^2_{X_n,\mathcal{S}} = \frac{\sum_{x \in X_n} d^2_{X_n,\mathcal{S}}(x)}{|X_n|} = \frac{\sum_{j=1}^{n} d^2_{X_n,\mathcal{S}}(x_j)}{|X_n|}, \tag{5.1}$$

where

$$d^2_{X_n,S}(x) = \min_{s \in \mathcal{S}} ||x - s||_2^2 \tag{5.2}$$

is the minimum squared distance of a high entropy point x to a point on any shape in the collection $\mathcal{S} = (S_1, \ldots, S_l)$.

We now describe our likelihood model $X_n|\mathcal{S}$ to encourage completeness. Figure 5.6 shows our notation. An observed point $x_j \in X_n$ is assumed to have been generated from a shape S_{z_j}, where z_j gives the shape number that explains x_j. Given z_j, we model the likelihood of x_j as a decreasing function of the minimum distance from x_j to S_{z_j}. The closer x_j is to shape S_{z_j}, the higher the likelihood of x_j. The observations

x_j are assumed to be independent and modeled as

$$\begin{aligned} x_j &= s_j + \varepsilon_j, \quad \text{independent} \\ \varepsilon_j &= r_j n_j, \\ r_j &\sim N(0, \sigma_r^2), \end{aligned}$$

where

$$\begin{aligned} s_j &= \arg\min_{s \in S_{z_j}} ||x_j - s||_2^2, \\ n_j &= \text{a unit normal to } S_{z_j} \text{ at } s_j, \\ r_j &= (x_j - s_j) \cdot n_j. \end{aligned}$$

Here the noise vector $\varepsilon_j = r_j n_j$ is along a unit normal n_j to the shape S_{z_j} at the closest shape point s_j to x_j. The scalar residual r_j is the signed distance along n_j from the shape S_{z_j} to x_j. We model the observation error ε_j by modeling the signed residual as a $N(0, \sigma_r^2)$ random variable. In Figure 5.6, the observed point x_j is explained by shape S_1 and thus $z_j = 1$. Since the error vector ε_j is opposite the chosen normal direction, the residual r_j is negative in this case.

Note that the squared residual r_j^2 is just the minimum distance squared from x_j to the closest point s_j on shape S_{z_j}:

$$r_j^2 = \min_{s \in S_{z_j}} ||x_j - s||_2^2,$$

where the minimum occurs at $s = s_j$. Let $Z = (z_1, \ldots, z_n)$. Then we derive the

likelihood as follows:

$$\begin{aligned}
P(X_n|Z,\mathcal{S}) &= P(x_1,\ldots,x_n|z_1,\ldots,z_n,S_1,\ldots,S_l) \\
&= \prod_{j=1}^{n} P(x_j|z_1,\ldots,z_n,S_1,\ldots,S_l) \quad \text{independence} \\
&= \prod_{j=1}^{n} P(x_j|z_j,S_{z_j}) \quad x_j \text{ depends only on shape } S_{z_j} \\
&= \prod_{j=1}^{n} N((x_j - s_j)\cdot n_j|0,\sigma_r^2) \\
&= \prod_{j=1}^{n} N(r_j|0,\sigma_r^2) \\
&= \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_r} \exp\left(-\frac{r_j^2}{2\sigma_r^2}\right) \\
&= K\sigma_r^{-n} \exp\left(-\frac{1}{2\sigma_r^2}\sum_{j=1}^{n} r_j^2\right) \\
P(X_n|Z,\mathcal{S}) &= K\sigma_r^{-n} \exp\left(-\frac{1}{2\sigma_r^2}\sum_{j=1}^{n} \min_{s_j\in S_{z_j}} ||x_j - s_j||_2^2\right), \qquad (5.3)
\end{aligned}$$

for a constant K. Note that if the observed point set X_n is close to the shapes in $\mathcal{S}$, then $P(X_n|Z,\mathcal{S})$ is high. This statement assumes, of course, that the correct shape S_{z_j} explaining each point x_j has also been identified.

We can obtain the likelihood $P(X_n|\mathcal{S})$ by modeling $Z|\mathcal{S}$ and integrating out Z as in $P(X_n|\mathcal{S}) = \int_Z P(X_n|Z,\mathcal{S})P(Z|\mathcal{S})dZ$. We could, for example, model $Z|\mathcal{S}$ by modeling a count vector $C = (c_1,\ldots,c_l)$ which holds the number of observations c_i explained by shape S_i. Here $c_i = \sum_{j=1}^{n} 1_{z_j=i}$. We can encourage good summary by modeling $C \sim \text{multinomial}(n,(1/l,1/l,\ldots,1/l))$ where each of the l shapes in $\mathcal{S}$ has the same probability $1/l$ of generating an observed point. This would make shape sets with

any shapes that are from the data quite unlikely because we would expect to see points around each shape according to the given multinomial distribution.

It is difficult, however, to optimize over shape sets with the hidden random variables Z in our models. Instead, we make a simple but accurate and effective approximation in our models and assume that the shape S_{z_j} that explains observation x_j is the shape in $\mathcal{S}$ which is closest to x_j. Thus we replace the minimization in equation (5.3) over $s_j \in S_{z_j}$ with a minimization $s_j \in \mathcal{S}$ over the entire shape set to obtain the approximation

$$P(X_n|\mathcal{S}) \quad = \quad K\sigma_r^{-n} \exp\left(-\frac{1}{2\sigma_r^2}\sum_{j=1}^{n}\min_{s_j\in\mathcal{S}}||x_j - s_j||_2^2\right). \tag{5.4}$$

From equations (5.1),(5.2), we can see that the inner sum in equation (5.4) is just a scaled version $|X_n|\,\overline{D}^2_{X_n,\mathcal{S}}$ of our completeness measure. (We will still incorporate the summary measure in our Bayesian loss function.) We can easily write our likelihood in terms of the completeness measure $\overline{D}^2_{X_n,\mathcal{S}}$. To do so cleanly, define $\sigma^2_{\text{complete}} = \sigma_r^2/|X_n|$. Then

$$P(X_n|\mathcal{S}) \quad = \quad K\sigma_{\text{complete}}^{-n} \exp\left(-\frac{1}{2\sigma^2_{\text{complete}}}\overline{D}^2_{X_n,\mathcal{S}}\right), \tag{5.5}$$

where another constant factor has been absorbed into K.

Shape Set Prior We build the shape set prior $P(\mathcal{S})$ based on the distances of points on each shape S_i to the rest of the shape set $\mathcal{S}_{-i} = \mathcal{S}\backslash\{S_i\}$. To keep shapes apart from one another, we want a large average squared distance from points on each shape to the rest of the shapes. Let $d^2_{S_i,S_j}(s_i)$ be the minimum squared distance of a point $s_i \in S_i$ to

another shape S_j:

$$d^2_{S_i,S_j}(s_i) = \min_{s_j \in S_j} ||s_i - s_j||_2^2.$$

Then the squared distance of $s_i \in S_i$ to the shape set $\mathcal{S}_{-i}$ is

$$d^2_{S_i,\mathcal{S}_{-i}}(s_i) = \min_{S_j \in \mathcal{S}_{-i}} d^2_{S_i,S_j}(s_i),$$

which finds the closest point in the rest of the shapes $\mathcal{S}_{-i}$ to $s_i \in S_i$. Finally we average the inter-shape squared distances over all points on all shapes to get

$$\overline{D}^2_{\mathcal{S}} = \frac{\sum_{S_i \in \mathcal{S}} \sum_{s_i \in S_i} d^2_{S_i,\mathcal{S}_{-i}}(s_i)}{\sum_{S_i \in \mathcal{S}} |S_i|} \tag{5.6}$$

To keep the shapes apart a priori, we want $\overline{D}^2_{\mathcal{S}}$ to be large, indicating that on average the inter-shape distance is large. Equivalently, $1/\overline{D}_{\mathcal{S}}$ should be small. Therefore we model the prior for $\mathcal{S}$ using the normal distribution

$$\mathcal{S} \sim N(\overline{D}_{\mathcal{S}}^{-1}; 0, \sigma^2_{\text{shapesim}}). \tag{5.7}$$

Bayesian Loss Next we define a Bayesian loss function that encourages good summary. We can think of the summary condition (i) as requiring a small distance from each shape $S \in \mathcal{S}$ to the set of classifier boundary points X_n. Let $d^2_{S,X_n}(s)$ denote the squared distance from a shape point $s \in S$ to the point set X_n:

$$d^2_{S,X_n}(s) = \min_{x \in X_n} ||s - x||_2^2.$$

We capture the average squared distance $\overline{D}^2_{\mathcal{S},X_n}$ from the shape set $\mathcal{S}$ to the input points X_n by averaging over all points on all shapes in $\mathcal{S} = (S_1, \ldots, S_l)$:

$$\overline{D}^2_{\mathcal{S},X_n} = \frac{\sum_{a=1}^{l} \sum_{s \in S_a} d^2_{S_a,X_n}(s)}{\sum_{a=1}^{l} |S_a|}. \tag{5.8}$$

We define our Bayesian loss function as

$$\text{loss}(\mathcal{S}, X_n) = \lambda_{\text{summary}} \overline{D}^2_{\mathcal{S},X_n}$$

The smaller the distance from each shape in $\mathcal{S}$ to the point set X_n, the smaller the loss. Thus minimizing the loss will encourage good summary.

5.3.3.4 Shape Fitting Method

Our shape fitting method has two main steps:

Step 1 Minimize the expected posterior loss

$$g(l) = E[\text{loss}(\mathcal{S}, X_n)], \quad |\mathcal{S}| = l$$

over the shape set size l to obtain the number of shapes l^*

Step 2 Compute the MAP shape set $\mathcal{S}^{*,l^*}$ for shape sets of size l^*

As we shall see, our method in Step 1 for choosing the number of shapes l^* to fit requires sampling from the shape set posterior. While drawing shape set samples, we can keep track of the maximum posterior probability shape set for each l to obtain the MAP shape set output in Step 2. Another option in Step 2 is to return an entire posterior shape set summary with confidence intervals around posterior mean shape sets of size l^*. During Step 1 processing, we can save all the posterior shape set samples for an l that gives a new minimum expected loss. Then we will have the shape set samples for the chosen number of shapes l^* and we simply compute a summary of those samples to output.

Determining the Number of Shapes We assume that we can apriori limit the number of shapes l to some set $\mathcal{L}$. For example, if we know that there will not be more than five boundaries then we can set $\mathcal{L} = \{1, 2, 3, 4, 5\}$.

For each l in $\mathcal{L}$, we compute the expected loss over the posterior

$$g(l) = E[\text{loss}(\mathcal{S}, X)] = \int_{\{\mathcal{S}:|\mathcal{S}|=l\}} \text{loss}(\mathcal{S}, X_n)\widehat{P}(\mathcal{S}|X_n)d\mathcal{S}. \tag{5.9}$$

Here we denote the shape set posterior probability distribution for shape sets with a fixed number of shapes as $\widehat{P}(\mathcal{S}|X_n)$. Then we choose the number of shapes to minimize the expected posterior loss:

$$l^* = \arg\min_{l\in\mathcal{L}} g(l).$$

The integral in equation (5.9) is difficult to compute analytically. Therefore we approximate the integral for $g(l)$ by drawing K shape set samples $\mathcal{S}^{(k)}$ of size l from the posterior $\mathcal{S}|X_n$:

$$g(l) = E[\text{loss}(\mathcal{S}, X)] \approx \sum_{k=1}^{K} \text{loss}(\mathcal{S}^{(k)}, X_n)\widehat{P}(\mathcal{S}^{(k)}, X_n).$$

Thus our method for determining the number of shapes to fit requires the ability to draw posterior shape set samples of a fixed number of shapes l. The technique to do so is the subject of the next section.

5.3.3.5 Our Shape Set Posterior Sampling Method

For a fixed shape set size $|\mathcal{S}| = l$, we will draw samples from the posterior $P(\mathcal{S}|X_n) \propto P(X_n|\mathcal{S})P(\mathcal{S})$ using an iterative procedure. Shape set samples $\mathcal{S}$ with a

small value for

$$-\log(P(X_n|\mathcal{S})P(\mathcal{S})) = -\log(P(X_n|\mathcal{S})) - \log(P(\mathcal{S}))$$

should be more likely to occur. Suppose we have an initial shape set $\mathcal{S}^0$, and let $\mathcal{S}[a] = S_a$ denote the a^{th} shape in the collection $\mathcal{S}$. Then we generate N shape set samples from the posterior $\mathcal{S}|X_n$ as follows:

for $j = 1..N$

$\mathcal{S}^j := \mathcal{S}^{j-1}$

for $a = 1..l$

(*) draw a sample S_a^* from $S_a|\mathcal{S}_{-a}^j, X_n$

$\mathcal{S}^j[a] := S_a^*$

end for a

end for j

In order to implement the sampling step $(*)$, we developed an algorithm, which is inspired by RANSAC (RANdom Sample And Consensus), which was first introduced by [Fischler and Bolles, 1981], as a method to estimate the parameters of a certain model starting from a set of data contaminated by large amounts of outliers. In a first hypothesis step, minimal sample sets (MSSs) are randomly selected from the input dataset and the model parameters are computed using only the elements of the MSS. This in contrast to other approaches, such as least squares, where the parameters are estimated using all the data available, possibly with appropriate weights. In finding

lines in 2D, for example, our algorithm chooses samples of size two from the point set since two points determine a line. In finding circles in 2D, each iteration chooses three points to define a circle.

In the test step, our algorithm checks which elements of the entire dataset are consistent with the model instantiated with the parameters estimated in the first step. The set of such elements is called consensus set (CS). Our algorithm terminates when the probability of finding a better ranked CS drops below a given threshold or a maximum number of iterations is reached.

Our algorithm essentially discovers the correct model when one of its iterations chooses a minimal sample of points that contains only inliers of the true model. Let us define this as *success* of the algorithm because a model close to the true model will be returned. Given an assumed fraction of inliers w, it is therefore possible to compute the number of iterations needed such that the probability of success is greater than some threshold p_{success} (although fewer iterations may be run to tradeoff quality for speed). If the minimal sample size is k (e.g. $k = 2$ for lines in 2D) and we assume random point selection with replacement, then the probability of selecting at least one outlier in an iteration is $1 - w^k$. The probability of selecting at least one outlier in all N trials is $(1 - w^k)^N$. We want this failure probability to be less than or equal to $1 - p_{\text{success}}$. Solving $1 - p_{\text{success}} \geq (1 - w^k)^N$ leads to requiring $N \geq \log(1 - p_{\text{success}}) / \log(1 - w^k)$ trials to have probability of success greater or equal to p_{success}. For example, to find a line in 2D with only $w = 0.25$ inliers and with probability of success at least $p_{\text{success}} = 0.999$, we would need to run only $N = 108$ iterations.

In order to implement the sampling step $(*)$, we fit each of the potential shape classes $M_1, \ldots, M_m$ and pick the shape out of all iterations that maximizes the conditional posterior $S_a|\mathcal{S}_{-a}, X_n$. To do so, we maximize $P(\mathcal{S}|X_n)$ as a function of S_a with the rest of the shapes $\mathcal{S}_{-a}$ held constant. Equivalently, we actually minimize $-\log(P(X_n|\mathcal{S})P(\mathcal{S}))$ as a function of S_a.

To draw a sample S_a^* from a particular shape class with $\mathcal{S}_{-a}$ fixed, we run a small number of iterations and pick the shape with the smallest cost:

$$\begin{aligned} \text{cost} &= -\log(P(X_n|\mathcal{S})P(\mathcal{S})) \\ &= -\log(P(X_n|\mathcal{S})) - \log(P(\mathcal{S})) \\ &= \text{cost}_{\text{complete}} + \text{cost}_{\text{shapesim}}, \end{aligned}$$

where

$$\begin{aligned} \text{cost}_{\text{complete}} &= \lambda_{\text{complete}} \overline{D}^2_{X_n,\mathcal{S}}, \\ \text{cost}_{\text{shapesim}} &= \lambda_{\text{shapesim}} \overline{D}_{\mathcal{S}}^{-2}, \end{aligned}$$

with the distance terms defined in equations (5.8),(5.1),(5.6) and the weights given in terms of the model variances as

$$\begin{aligned} \lambda_{\text{complete}} &= 1/(2\sigma^2_{\text{complete}}), \\ \lambda_{\text{shapesim}} &= 1/(2\sigma^2_{\text{shapesim}}). \end{aligned}$$

This evaluation criterion gives sample shape sets $\mathcal{S}$ that are complete and minimal for the classifier boundaries of P_n.

This procedure is summarized below.

(*) draw a sample S_a^* from $S_a|\mathcal{S}_{-a}^j, X_n$ as follows:

mincost := ∞; S_a^* := NONE

for shapetype = $M_1, \ldots, M_m$

for iter = 1..num_Iters

k := min_sample_size(shapetype)

pick k points $x_{j_1}, \ldots, x_{j_k}$ from X_n with $p(x) = 1 - \exp(-\gamma d^2_{X_n, \mathcal{S}_{-a}}(x))$

S^c := fitshape(shapetype, $x_{j_1}, \ldots, x_{j_k}$)

$\mathcal{S}^c := \mathcal{S}_{-a} \cup \{S^c\}$

$\text{cost}_{\text{complete}} := \lambda_{\text{complete}} \overline{D}^2_{X_n, \mathcal{S}^c}$

$\text{cost}_{\text{shapesim}} := \lambda_{\text{shapesim}} \overline{D}^{-2}_{\mathcal{S}^c}$

cost := $\text{cost}_{\text{complete}} + \text{cost}_{\text{shapesim}}$

if (cost < mincost)

mincost := cost; S_a^* := S^c

end if

end for iter

end for shapetype

Figure 5.7 shows a few iterations to draw a sample S_1^* with $\mathcal{S}_{-1} = \{S_2, S_3\}$ fixed. Each iteration does the following: a random line candidate S_1^c is determined by choosing two points, shown as filled red squares, at random from X_n. The random shape S_1^c completes a potential shape set, and then the posterior $P(\{S_1^c, S_2, S_3\}|X_n)$ is

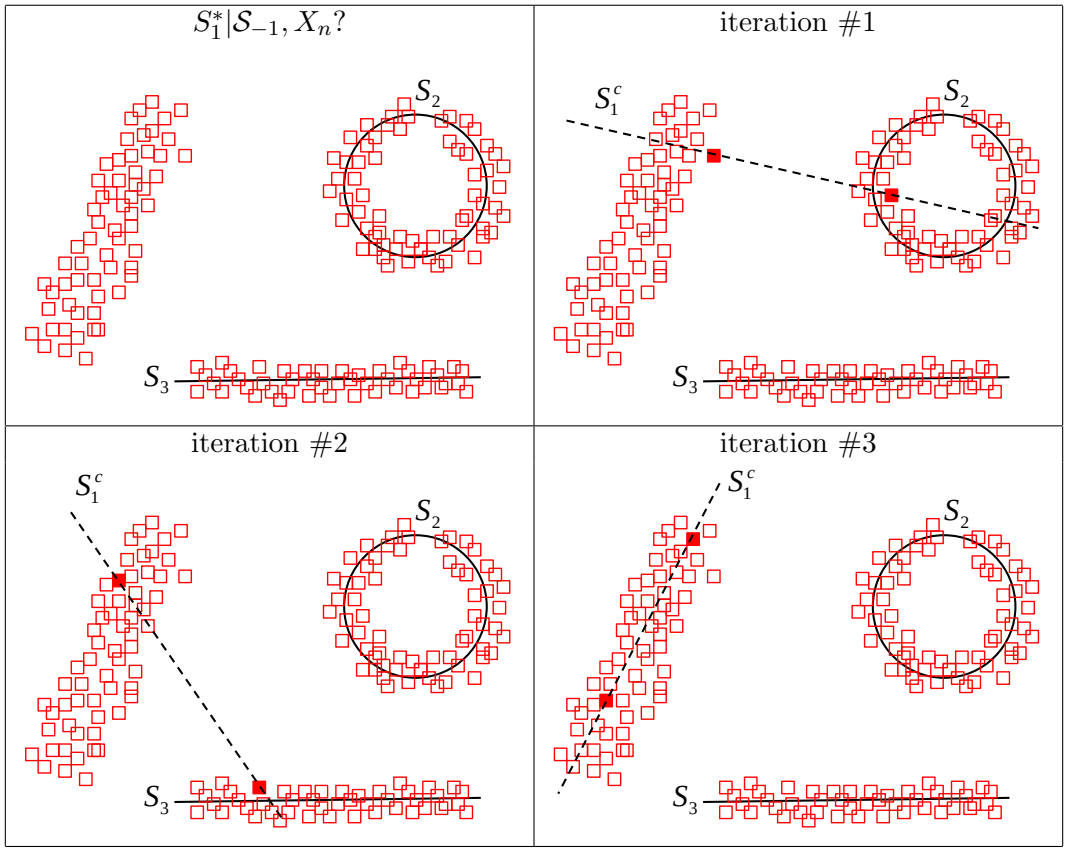

Figure 5.7: A few iterations to draw a sample S_1^* with $\mathcal{S}_{-1} = \{S_2, S_3\}$ fixed. One random shape is considered in each iteration. The random shape candidate S_1^c chosen in iteration #1 results in a shape set with low posterior probability. The same is true for iteration #2. The random shape candidate S_1^c chosen in iteration #3 results in a shape set with high posterior probability.

evaluated to see if a new maximum posterior probability has been achieved. The first two iterations in this example fail to yield a high posterior shape set. The third iteration does produce a high posterior shape set. Subsequent iterations are unlikely to find a substantially better candidate shape S_1^c. Once we have reached the maximum number of iterations or we have found a shape set with sufficiently high posterior probability, we return the best candidate shape as the sample $S_1^*|\mathcal{S}_{-1}, X_n$.

In the sampling step (*), we are trying to find a high posterior probability shape S_a^* within the set of high entropy points given the rest of the shapes $\mathcal{S}_{-a}$. The boundary points explained by the rest of shapes $\mathcal{S}_{-a}$ are essentially outliers in the computation of S_a^*. This is because we want to find a shape S_a^* that covers a part of the classifier boundary that is not already covered by $\mathcal{S}_{-a}$.

In our procedure we choose samples based on the squared distance

$$d^2_{X_n,\mathcal{S}_{-a}}(x) = \min_{S \in \mathcal{S}_{-a}} d^2_{X_n,S}(x)$$

of points $x \in X_n$ to the rest of the shapes $\mathcal{S}_{-a}$. When updating S_a, we want a shape that is far from the other shapes $\mathcal{S}_{-a}$ being held constant. Thus we choose a point sample $x \sim p$, where $p(x) = 1 - \exp(-\gamma d^2_{X_n,\mathcal{S}_{-a}}(x))$. This heuristic encourages samples farther from $\mathcal{S}_{-a}$ to chosen. In turn, this leads to shape sets that have higher posterior probability than uniformly sampling from X_n during an iteration. Furthermore, we evaluate the shape implied by a minimal point set using the posterior shape set probability rather than the number of model inliers. Our iterative sampling procedure is a probabilistic version of multiple shape detection algorithms that repeatedly find a model and then

remove all the inlier points for the model before searching for the next model.

5.3.3.6 Results: Shape Set Posterior Sampling

We now show shape set posterior sampling results for fixed shape set size l. Our first test example is shown in Figure 5.8. The examples in this section consist of randomly generated data with ground truth shape sets in 2D consisting of lines and circles in the domain $\Omega = [0,1] \times [0,1]$. Once a random ground truth shape set has been chosen, we project a Latin Hypercube Sampling (LHS) of Ω of size 200 points onto the ground truth shapes to get a point set representation of the ground truth shapes. We then add noise $n \sim N(0, \sigma_n^2 I)$ to the point set to produce the input point set X_n for our shape fitting procedure. For all examples in this section, we use a noise standard deviation of $\sigma_n = 0.05$, which represents a moderate amount of noise.

Unless otherwise noted, the computed results in this section use the parameter settings $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, and $\lambda_{\text{shapesim}} = 0.007$. Recall that the summary and completeness statistical models are based on average squared distance between points, while the shape similarity term is based on *inverse* average squared distance. This accounts for the difference in magnitudes of the three weights. These λ-weights correspond to model standard deviations in distance and inverse distance units of $\sigma_{\text{summary}} \doteq 0.02$, $\sigma_{\text{complete}} \doteq 0.02$, and $\sigma_{\text{shapesim}} \doteq 8.45$.

The example in Figure 5.8 has $l_{\text{true}} = 2$ shapes, one line and one circle. The ground truth shapes are shown as dotted blue lines. The input point set after adding noise to the ground truth is shown as red squares. Recall that the input point sets

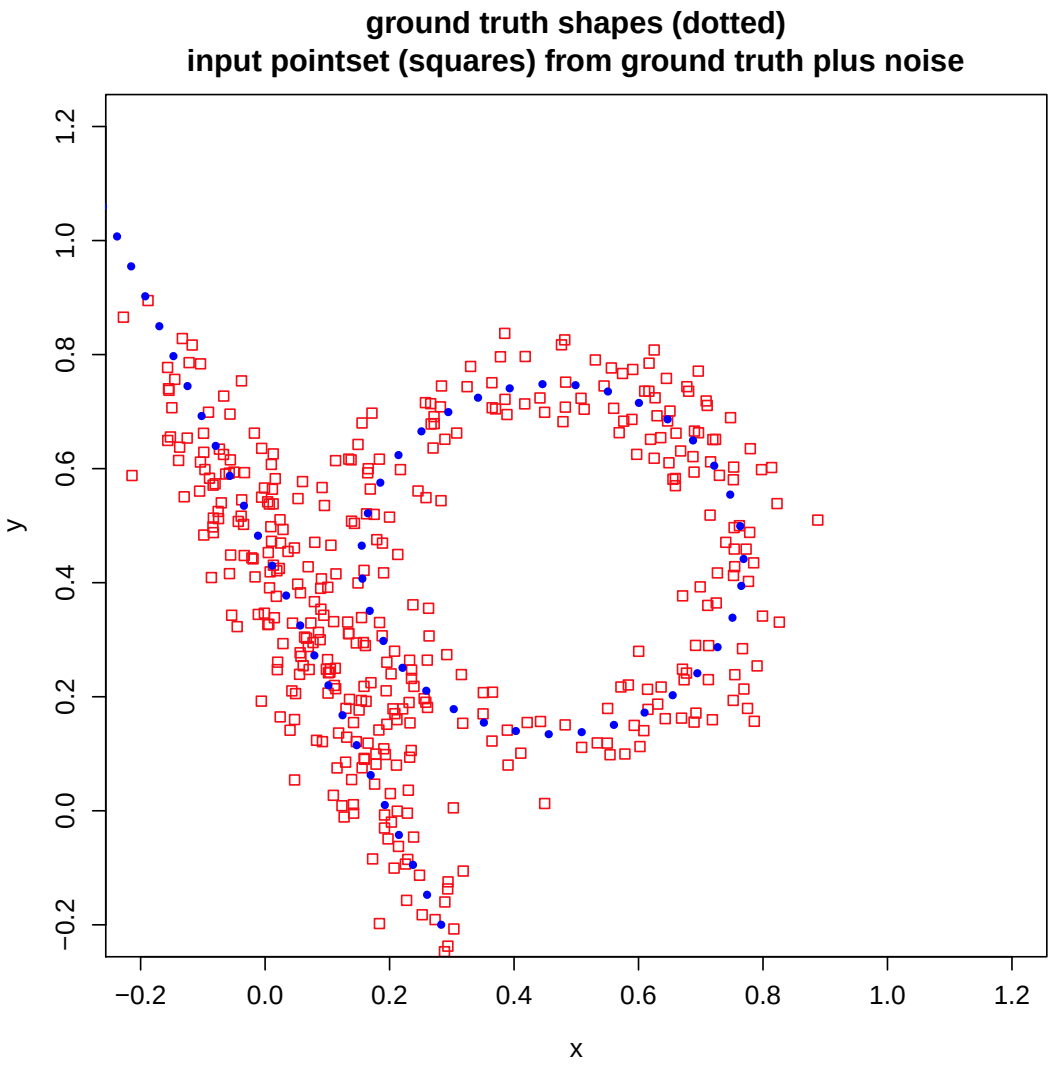

Figure 5.8: Example. 2D hyperplanes and spheres input with $l_{\text{true}} = 2$ shapes.

represent high entropy points of a classifier whose class boundaries we are trying to summarize. Although in our main NASA application we are not expecting close or overlapping classifier boundaries, we have designed our shape fitting procedure to be robust to these difficult inputs.

Next we show several examples for shape set posterior sampling assuming the correct $l = l_{\text{true}}$ number of shapes. We use the first example in Figure 5.8 to explain the details of our posterior shape set plots. The results of our posterior shape set sampling procedure are shown in Figure 5.9. The black solid lines show the posterior mean shapes computed. As one can see, our statistical models and sampling procedure discovers the correct shape types and provides shape estimates close to the ground truth. The dotted black lines illustrate shape "confidence intervals" around the mean shapes. It is a bit tricky to show confidence intervals for the shapes, even in 2D, because there is uncertainty in more than one parameter defining the shapes. We could simply overlay plots of the posterior shape samples to give an idea of density, but it quickly becomes difficult to see what is happening in the plot with even a handful of posterior samples and the input point set and ground truth shapes. Instead let us explain what we are showing for the shape confidence intervals.

First consider confidence intervals for lines in 2D. There is uncertainty in both the slope of the line and its intercept. In the results of this section, we are showing only the uncertainty in the intercept around an average slope defined by the posterior shape samples. Recall our line representation $\theta_1 x + \theta_2 y + \theta_3 = 0$. Denote the set of posterior line samples as $\{(\theta_1^{(n)}, \theta_2^{(n)}, \theta_3^{(n)})\}_n$. For non-vertical lines, our representation

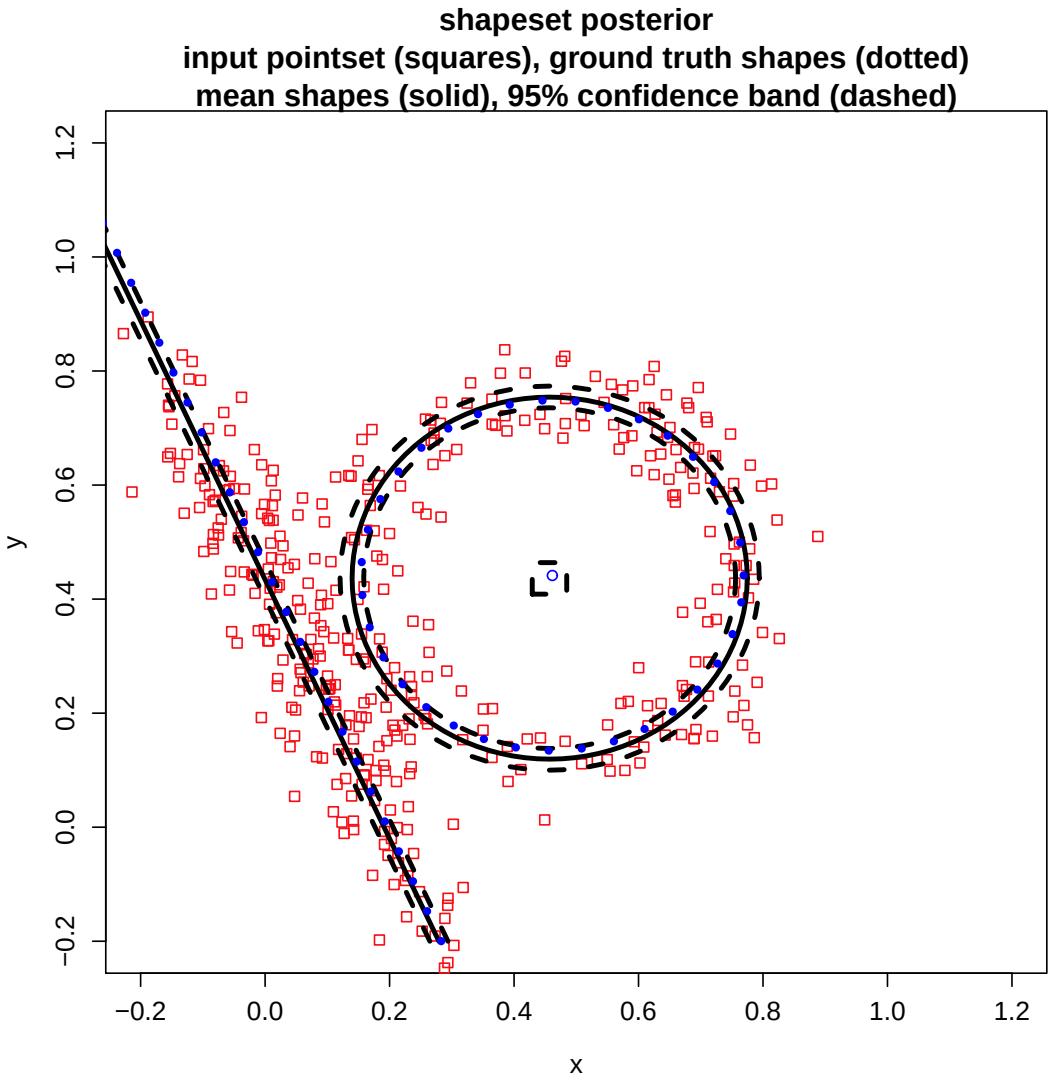

Figure 5.9: Shape Set Posterior. 2D hyperplanes and spheres input with $l_{\text{true}} = 2$ shapes. Sampling Parameters: $l = 2$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

is equivalent to $y = mx + b$, with slope $m = -\theta_1/\theta_2$ and intercept $b = -\theta_3/\theta_2$. For a $(1-\alpha)\%$ confidence interval we compute the $\alpha/2$ and $1-\alpha/2$ quantiles b_{low} and b_{high}, respectively, of the set $\{b^{(n)} = -\theta_3^{(n)}/\theta_2^{(n)}\}_n$. Let $b_{\text{mid}} = 0.5 * (b_{\text{low}} + b_{\text{high}})$ be the midpoint between b_{low} and b_{high}. We also compute the mean slope $\overline{m}$ as the mean of the set $\{m^{(n)} = -\theta_1^{(n)}/\theta_2^{(n)}\}_n$. This then defines a mean solid line $y = \overline{m} + b_{\text{mid}}$ surrounded by $(1-\alpha)\%$ confidence dotted lines $y = \overline{m} + b_{\text{low}}$ and $y = \overline{m} + b_{\text{high}}$.

In order to handle vertical lines, the above procedure for generating the confidence interval is modified slightly. First, we rotate all sample lines so that the line with parameters equal to the mean of the parameter samples is horizontal. Then we apply the procedure in the rotated space to obtain the low, mid, and high confidence lines. Finally, we rotate these lines back to the original space to get the final shape mean and $(1-\alpha)\%$ confidence lines.

Since the variation in angle is not represented in the plotted confidence interval, it is not unusual to see the ground truth line go slightly outside of the plotted confidence interval near the edges of the domain. In the line result in Figure 5.8, the ground truth line is almost entirely within the plotted 95% confidence interval except near the top of plot where it is approaching the confidence boundary.

Next consider confidence intervals for circles in 2D. There is uncertainty in both the radius and the center. The results in this section show confidence intervals for the radius around the mean circle center and for the center as well. Recall our circle representation $(x-\theta_1)^2 + (y-\theta_2)^2 = \theta_3^2$ with center $c = (\theta_1, \theta_2)$ and radius $r = \theta_3$. We compute the mean circle center $\overline{c} = (\overline{\theta_1}, \overline{\theta_2})$ as the mean of the center posterior samples

$\{(\theta_1^{(n)}, \theta_2^{(n)})\}$. This center point $\overline{c}$ is shown with a blue little circle in Figure 5.9 and in the rest of the results of this section. The mean radius circle is shown as a solid black curve centered at $\overline{c}$ with radius that is $\overline{r} = \overline{\theta_3}$, the mean of the radius posterior samples $\{\theta_3^{(n)}\}_n$. We also show $(1-\alpha)\%$ confidence circles around the mean circle as dashed black curves. These confidences bands are defined by the circles with center $\overline{c}$ and radii r_{low} and r_{high}, where r_{low} and r_{high} are the $\alpha/2$ and $1-\alpha/2$ quantiles of the radius posterior samples $\{\theta_3^{(n)}\}_n$.

We also show a $(1-\alpha)\%$ confidence rectangle for the center itself as a dashed black rectangle. This confidence rectangle has corners $(c^x_{\text{low}}, c^y_{\text{low}})$, $(c^x_{\text{low}}, c^y_{\text{high}})$, $(c^x_{\text{high}}, c^y_{\text{high}})$, and $(c^x_{\text{high}}, c^y_{\text{low}})$, where c^x_{low} and c^x_{high} are the $\alpha/2$ and $1-\alpha/2$ quantiles of the center x-coordinate posterior samples $\{\theta_1^{(n)}\}_n$ and c^y_{low} and c^y_{high} are the $\alpha/2$ and $1-\alpha/2$ quantiles of the center y-coordinate posterior samples $\{\theta_2^{(n)}\}_n$. Again, this is not a perfect confidence interval display for circles since it does not show the uncertainty of all circle parameters in just one set of boundaries. A ground truth circle may slightly escape the mean circle confidence band along certain parts of its boundary since that band does not account for uncertainty in the center. In the result in Figure 5.9, the ground truth circle does stay inside the 95% mean circle confidence band all the way around its perimeter, although it does approach the confidence boundary quite closely in some parts. The mean circle center also lies inside the 95% confidence rectangle, although it not at the center of this confidence rectangle.

We can also give some intuition here about high posterior probability posterior shape sets. High posterior probability posterior shape sets will generally go through the

means of the input points that arise from each shape. The shape similarity term can push the shapes away from the means to separate the shapes more, but should do so in moderate amounts as long as shape similarity is weaker than the completeness term in our model. In the result in Figure 5.9, the mean posterior line and circle indeed basically pass through the means of the regions of noisy points generated from the ground truth shapes. This is evidenced by the close proximity (relative to the added noise amount) of the mean posterior shapes to the ground truth shapes, as the noise was added equally in all directions. Our default parameter settings give a lower relative weight to the shape similarity term because we first want to ensure that the shape set is complete and then we bias complete shape sets to be minimal. Our modeling assumptions keep almost all of the posterior probability mass in a narrow band around the true shapes.

The example in Figure 5.10 contains just one shape boundary, namely a line. Our posterior shape set sampling procedure correctly discovers the shape and provides a tight 95% confidence region that almost entirely contains the ground truth line. Here, we specified $l = 1$ and restricted the set of potential shape types to $\mathcal{M} = \{\text{hyperplane}\}$.

Another example with just a single shape boundary is shown in Figure 5.11, except here the ground truth shape is a circle. Here we set the sampling parameters according to the ground truth: $l = 1$ and $\mathcal{M} = \{\text{sphere}\}$. The computed 95% radius confidence region contains most of the circle and the 95% center confidence rectangle indeed contains the ground truth circle center.

Our posterior sampling algorithm handles shape sets with more than a single shape. The examples in Figures 5.12 and 5.13 each consist of two hyperplanes, the

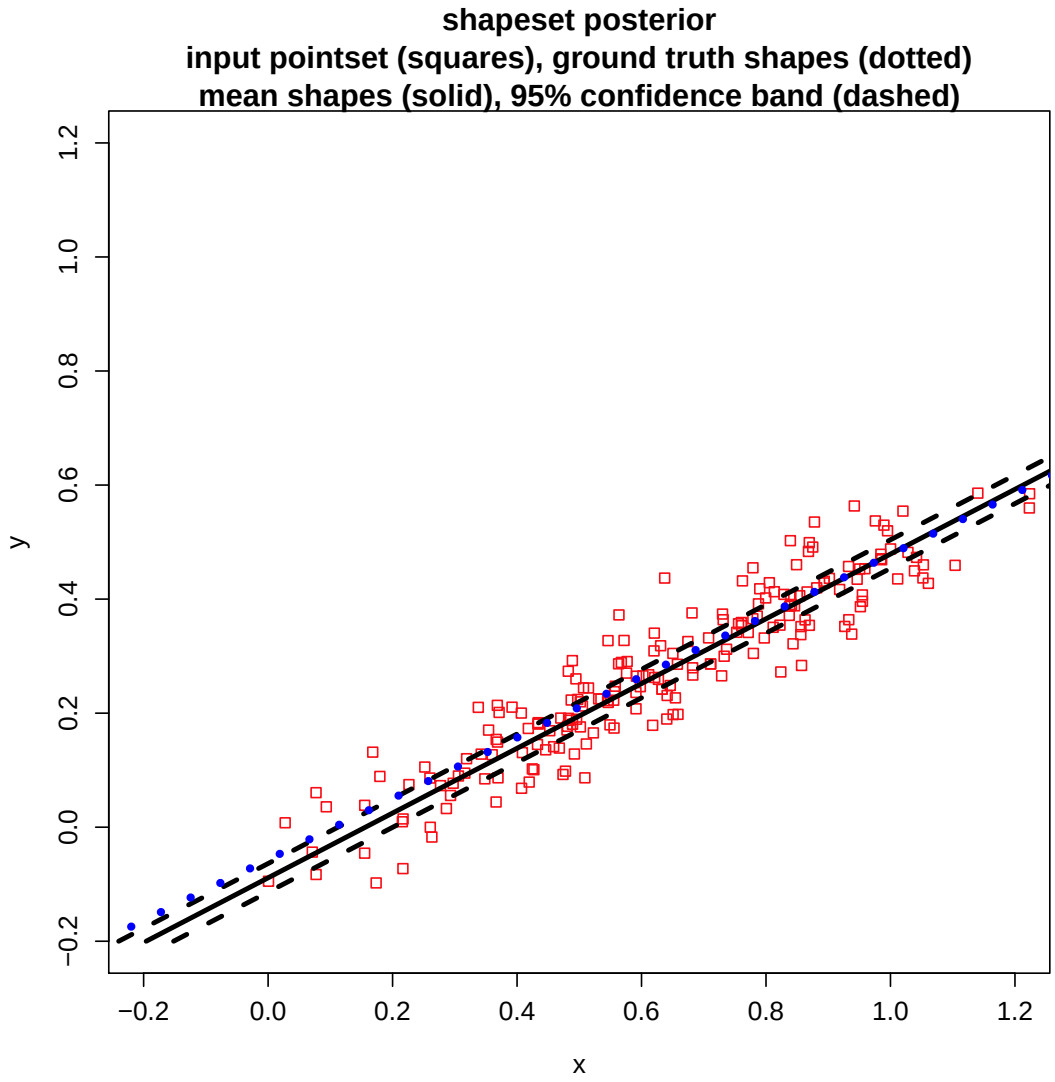

Figure 5.10: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 1$ shape. Sampling Parameters: $l = 1$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

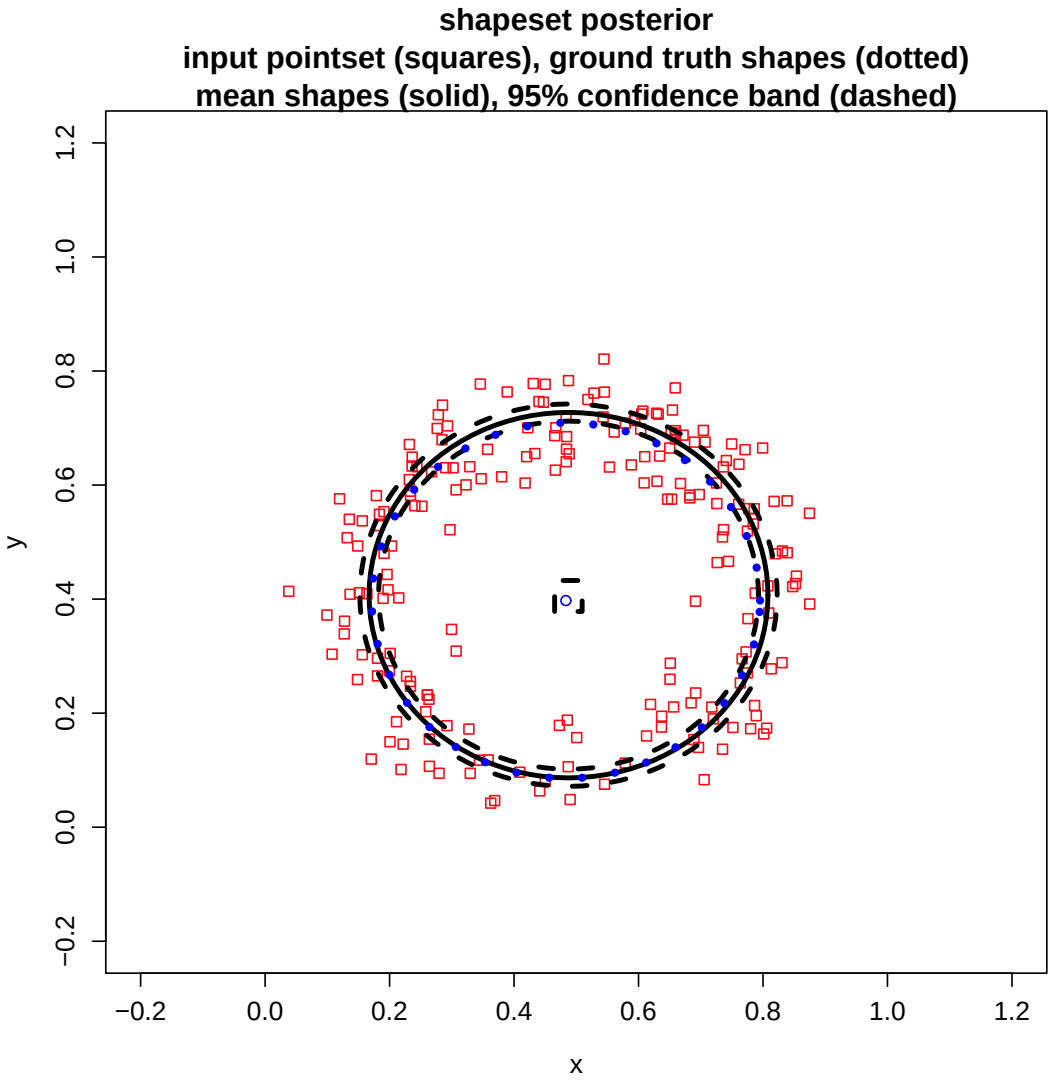

Figure 5.11: Shape Set Posterior. 2D spheres input with $l_{\text{true}} = 1$ shape. Sampling Parameters: $l = 1$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

former without overlapping shapes in the domain Ω and the latter with overlapping shapes. For both inputs, our posterior shape set sampling accurately computes the correct shape set output based on the input $l = 2$ and $\mathcal{M} = \{\text{sphere}\}$. Note how the 95% confidence intervals contain the ground truth shapes.

Our shape set posterior sampling method can also handle two circles, as shown in the result in Figure 5.14. Both the circle center and the perimeter of the mean circle are within the 95% confidence interval computed by our shape set sampling procedure. The inputs to our shape set posterior sampling method are $l = 2$ and $\mathcal{M} = \{\text{sphere}\}$.

We made the problem even harder in Figure 5.15 where we have three hyperplanes that come close to another near the boundary of the input domain Ω. Nonetheless, the posterior means and confidence intervals capture the three ground truth lines quite well. Here we input $l = 3$ and $\mathcal{M} = \{\text{hyperplane}\}$ to the sampling procedure.

Figure 5.16 shows the results of our posterior shape set sampling method on another example with three shapes, but containing both lines and a circle. The shapes do not overlap in this example. Again, the shape set confidence intervals contain the entire ground truth shapes in Ω. For this example we input $l = 3$ and $\mathcal{M} = \{\text{hyperplane}, \text{circle}\}$ to the sampling procedure.

Our last example in Figure 5.17 is a difficult one with four overlapping hyperplanes. The posterior shape set results are quite good for three of the four lines. The 95% confidence interval for the nearly horizontal line at the top does contain the ground truth line, but the posterior mean is not very close to the ground truth line and the 95% confidence interval is very wide. The problem here for this overlapping shapes case

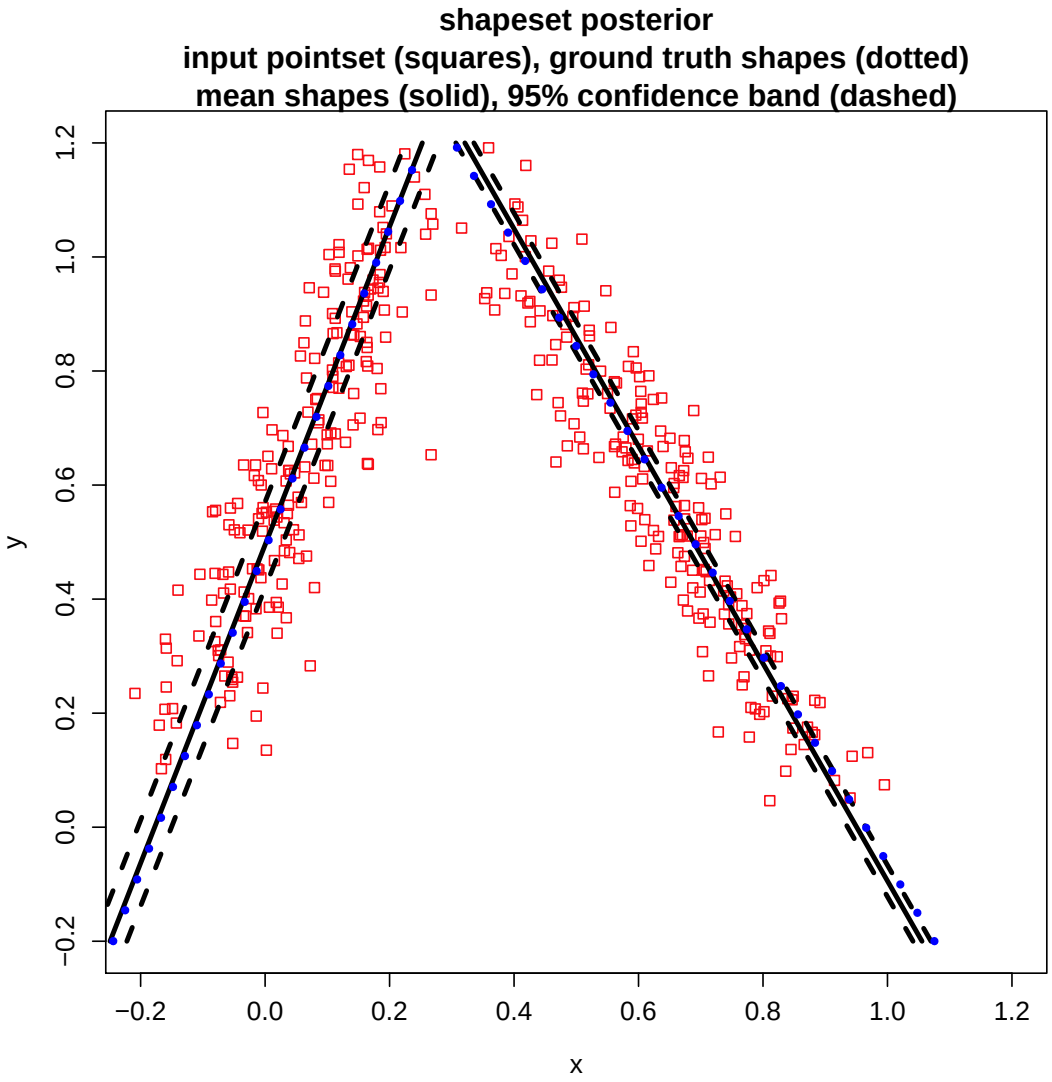

Figure 5.12: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 2$ shapes. Sampling Parameters: $l = 2$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

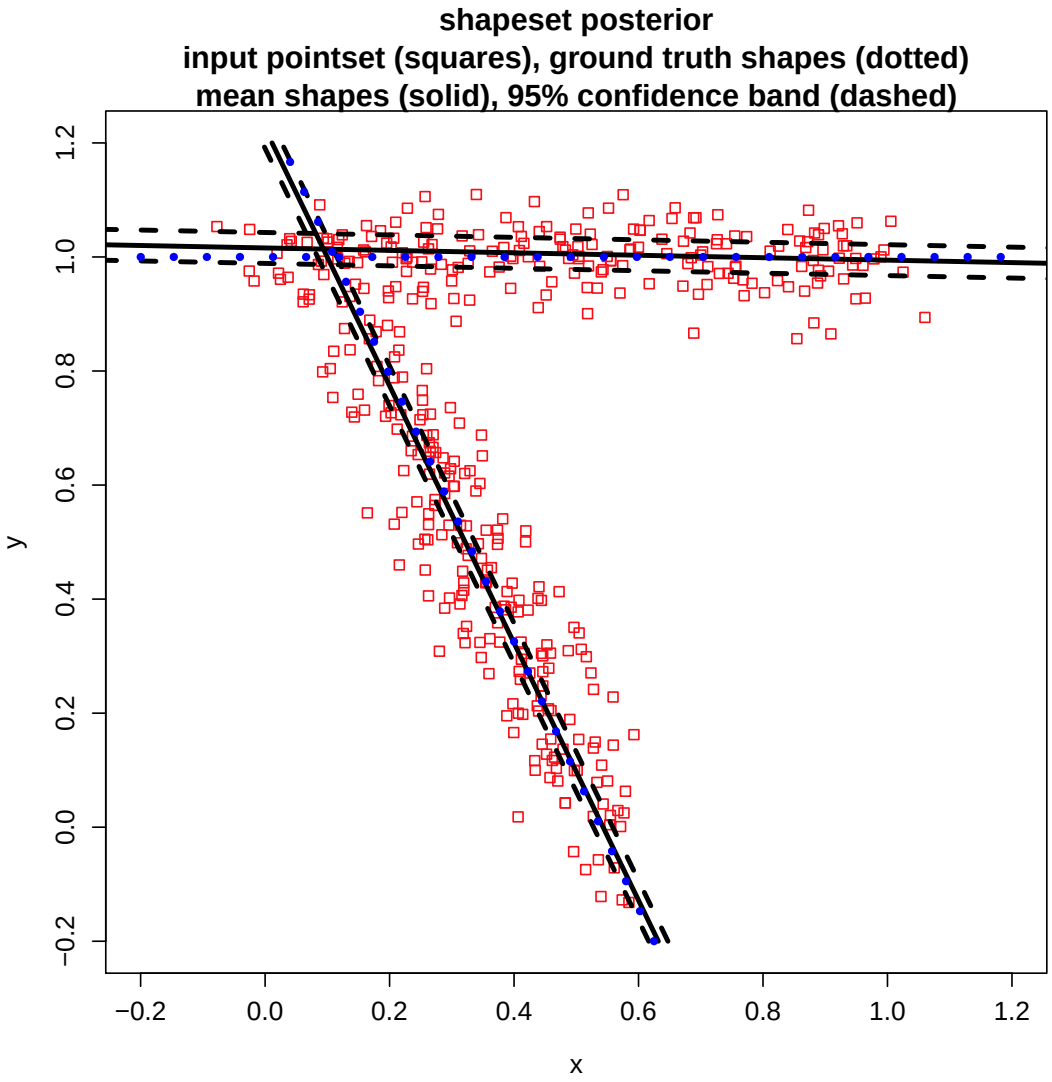

Figure 5.13: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 2$ shapes. Sampling Parameters: $l = 2$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

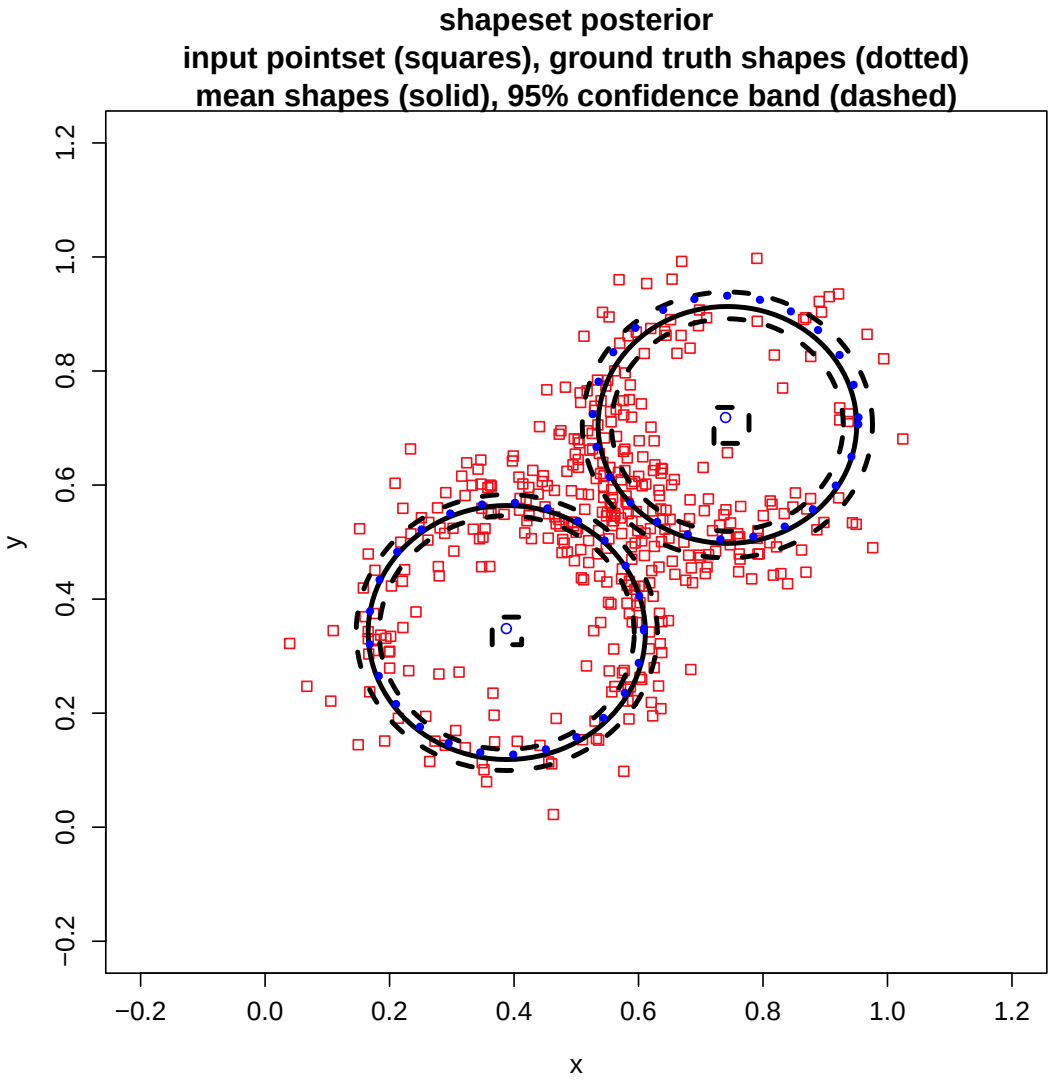

Figure 5.14: Shape Set Posterior. 2D spheres input with $l_{\text{true}} = 2$ shapes. Sampling Parameters: $l = 2$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

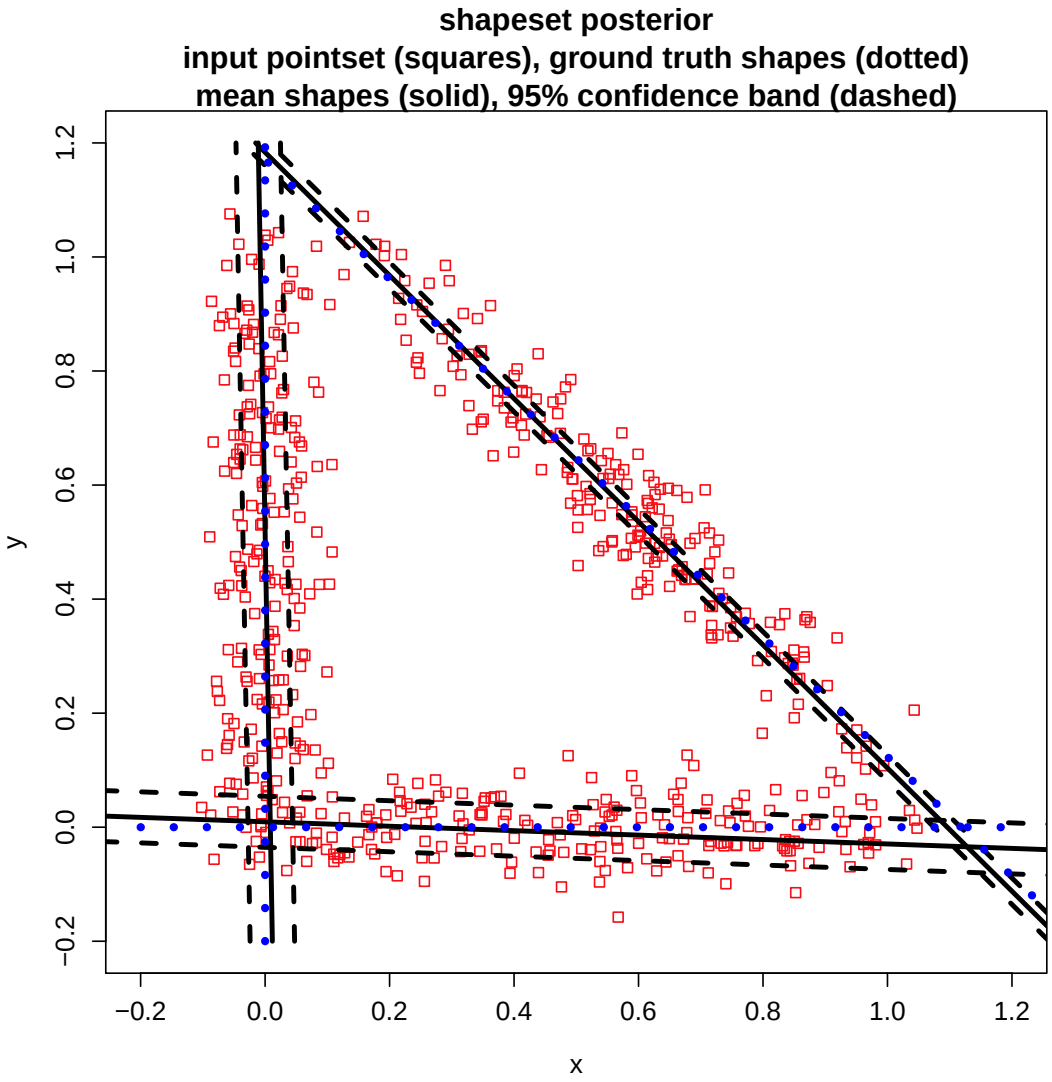

Figure 5.15: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 3$ shapes. Sampling Parameters: $l = 3$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

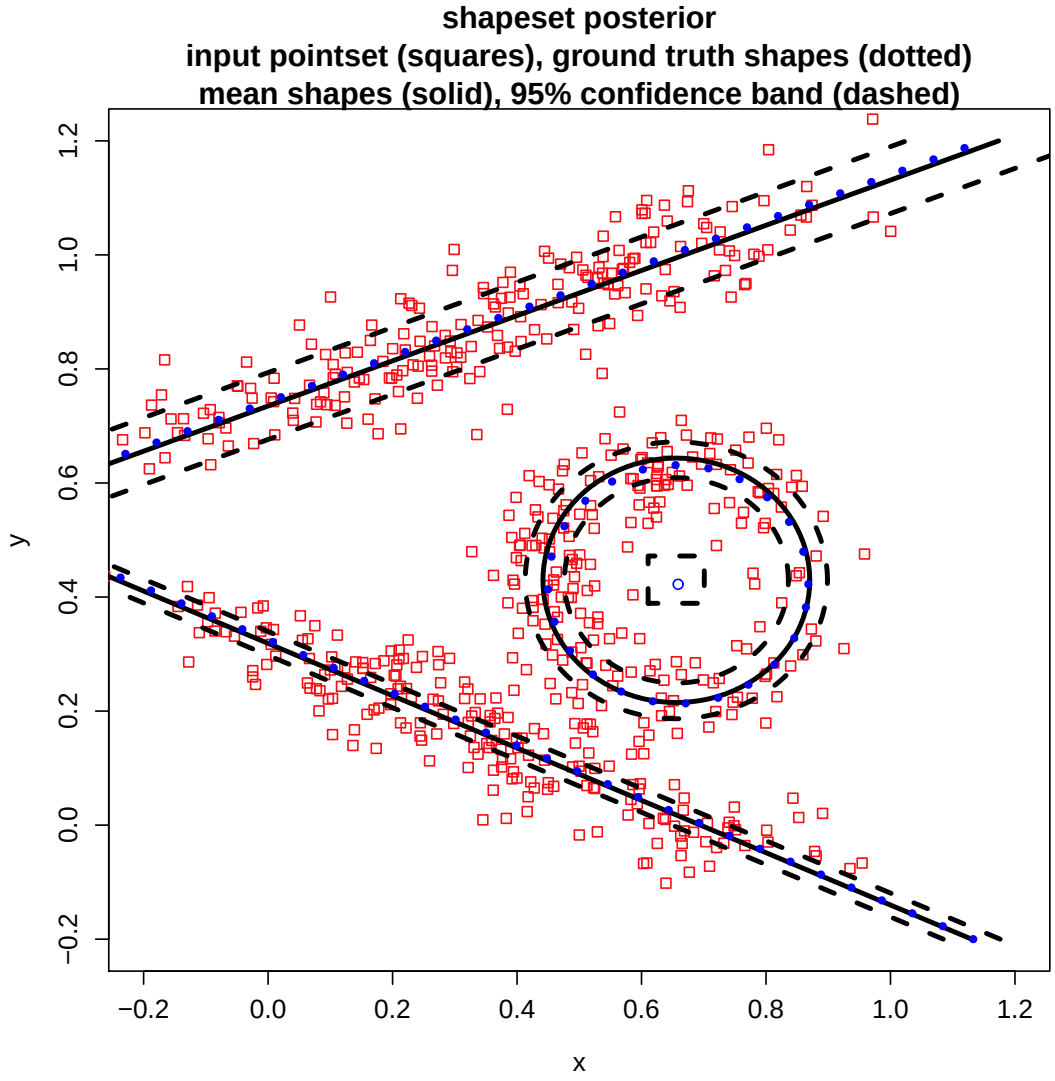

Figure 5.16: Shape Set Posterior. 2D hyperplanes and spheres input with $l_{\text{true}} = 3$ shapes. Sampling Parameters: $l = 3$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

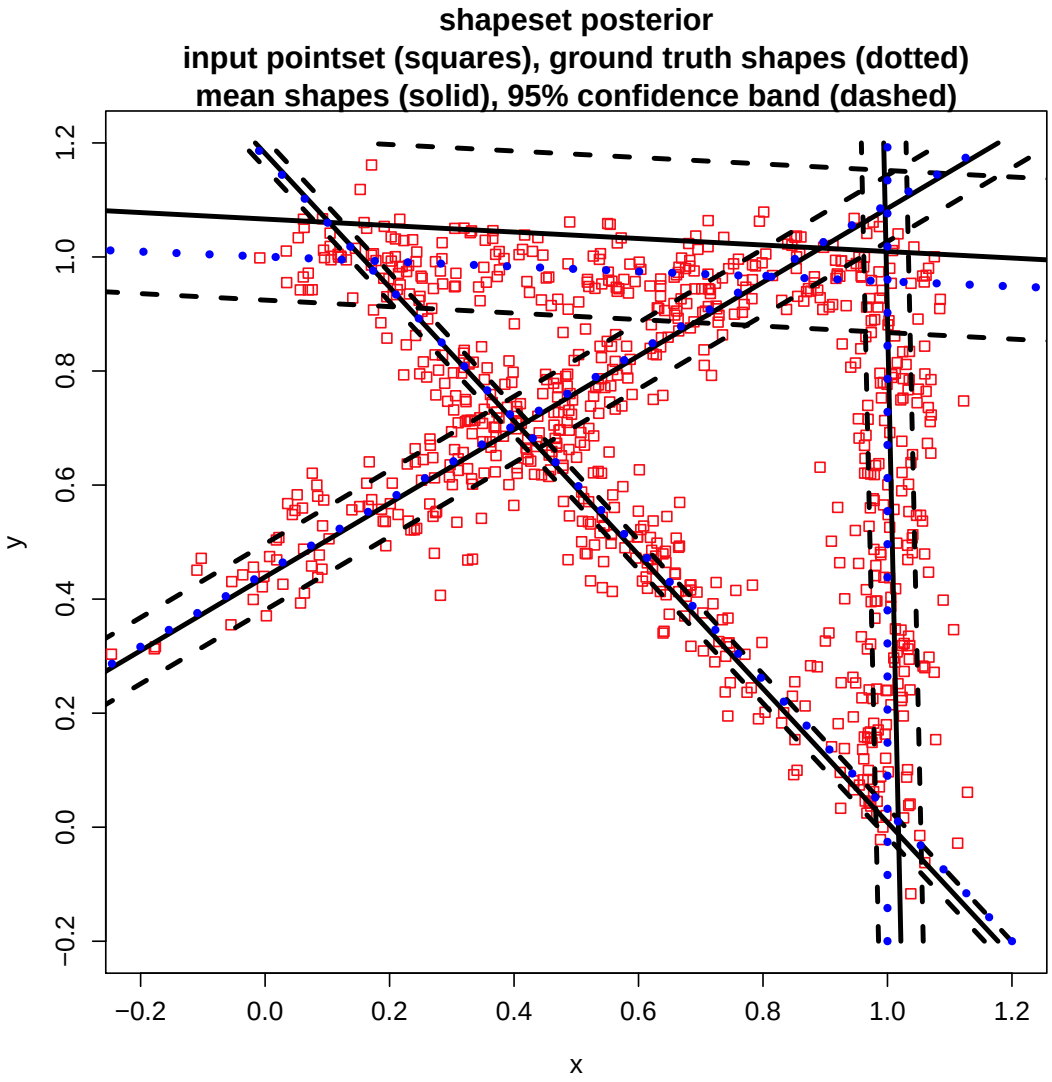

Figure 5.17: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 4$ shapes. Sampling Parameters: $l = 4$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.007$.

with many shapes is that the similarity cost is weighted too high. Indeed if we decrease the weight on the similarity cost, our statistical models are willing to keep the shapes a little bit closer together as shown in Figure 5.18. For these parameter settings, the posterior mean becomes much closer to the ground truth near horizontal line and the 95% confidence interval is narrower than the result in Figure 5.17 with a larger weight on the shape similarity term.

5.3.3.7 Results: Number of Shapes

Thus far we have shown posterior shape set sampling results for shape sets of a fixed, specified number of shapes l. In this section we show results for determining the number of shapes by minimizing the expected posterior loss. Equivalently, we actually maximize the expected posterior gain, where

$$\text{gain}(\mathcal{S}, X_n) = \exp(-\text{loss}(\mathcal{S}, X_n)) = \exp(-\lambda_{\text{summary}} \overline{D}^2_{\mathcal{S},X_n})$$

So the location of the maximum in the graphs in this section (y-axis: expected gain, x-axis: l) indicates the predicted number of shapes. The graphs of l versus the expected posterior gain $g(l) = E[\text{gain}(\mathcal{S}, X_n)]$ where $|\mathcal{S}| = l$ are shown in Figures 5.19–5.23. Note that the number of shapes in all of these examples is correctly identified.

Now let us take a closer look at a few of these examples. By examining the MAP shape set estimate for each number of shapes l, we will gain further intuition for our shape set statistical model. Once we have selected the number of shapes l^* by maximizing the expected posterior gain, we also output the MAP shape set over sets with size $\mathcal{S} = l^*$.

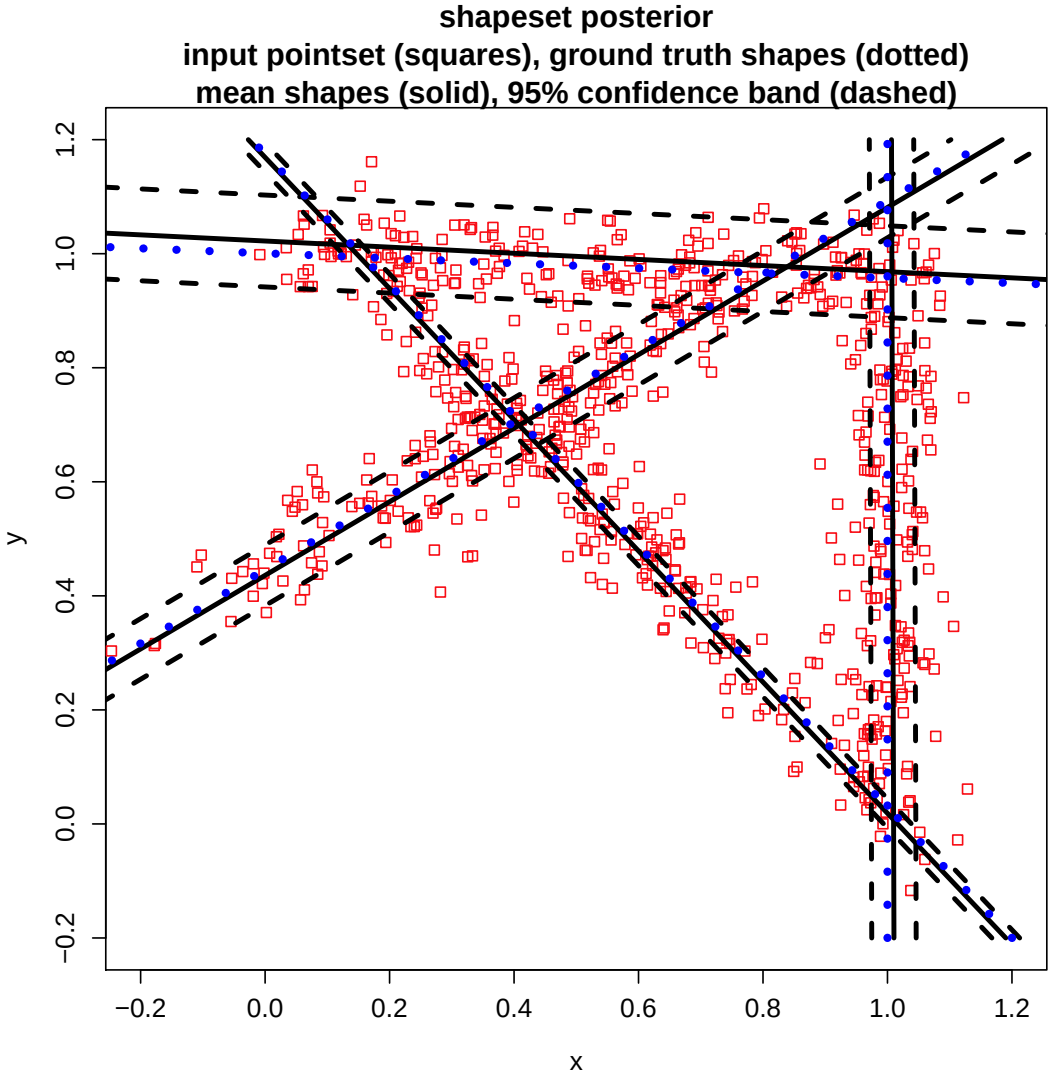

Figure 5.18: Shape Set Posterior. 2D hyperplanes input with $l_{\text{true}} = 4$ shapes. Sampling Parameters: $l = 4$, $\lambda_{\text{summary}} = 1000$, $\lambda_{\text{complete}} = 1000$, $\lambda_{\text{shapesim}} = 0.003$. Here we have lowered the shape similarity weight compared to the result in Figure 5.17, thus allowing shapes to be a little bit closer together (i.e. more similar). Here the posterior mean becomes much closer to the ground truth near horizontal line and the 95% confidence interval is narrower than the result in Figure 5.17.

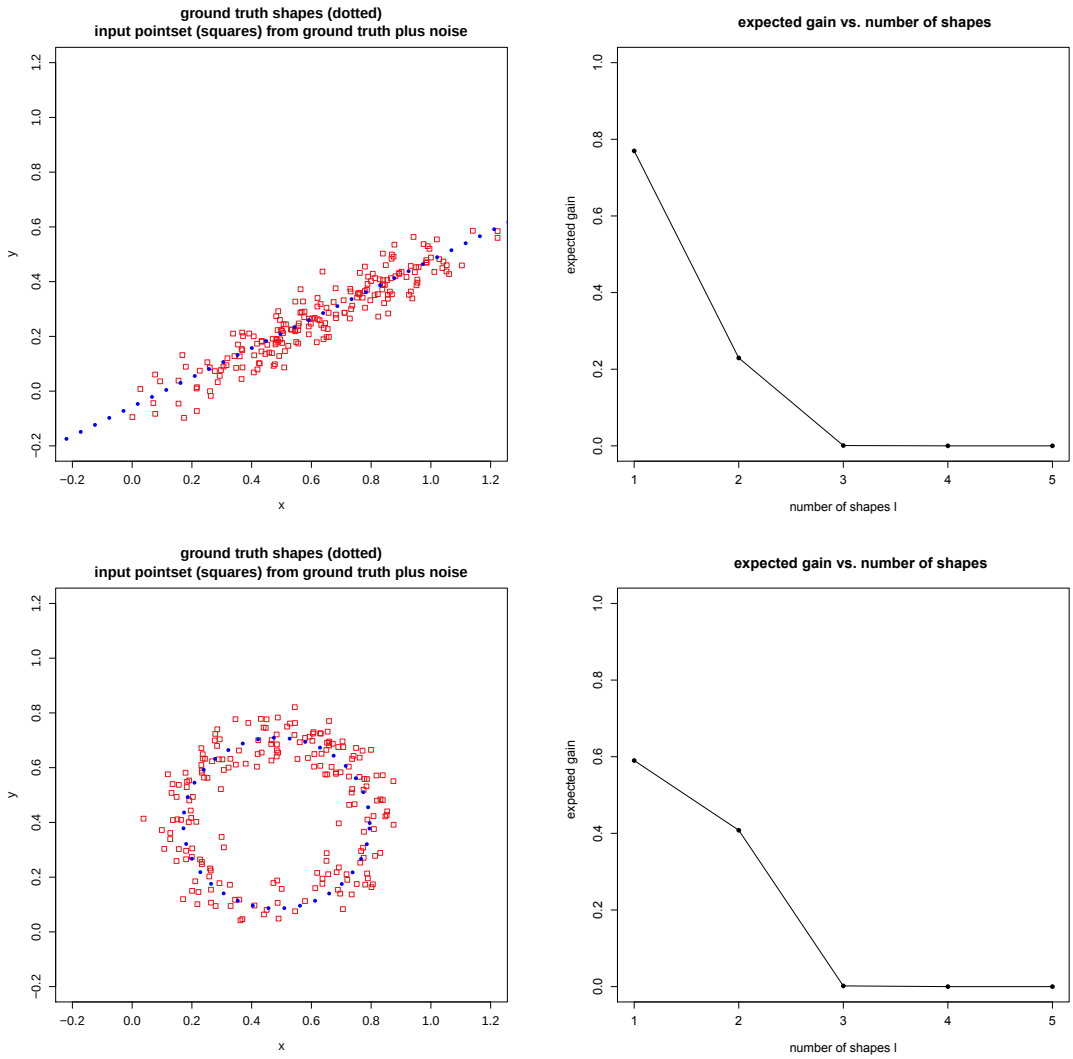

Figure 5.19: Number of Shapes – Result Set 1. (left) Inputs. (right) Expected posterior gain versus number of shapes l. (row 1) $l_{\text{true}} = 1$. (row 2) $l_{\text{true}} = 1$.

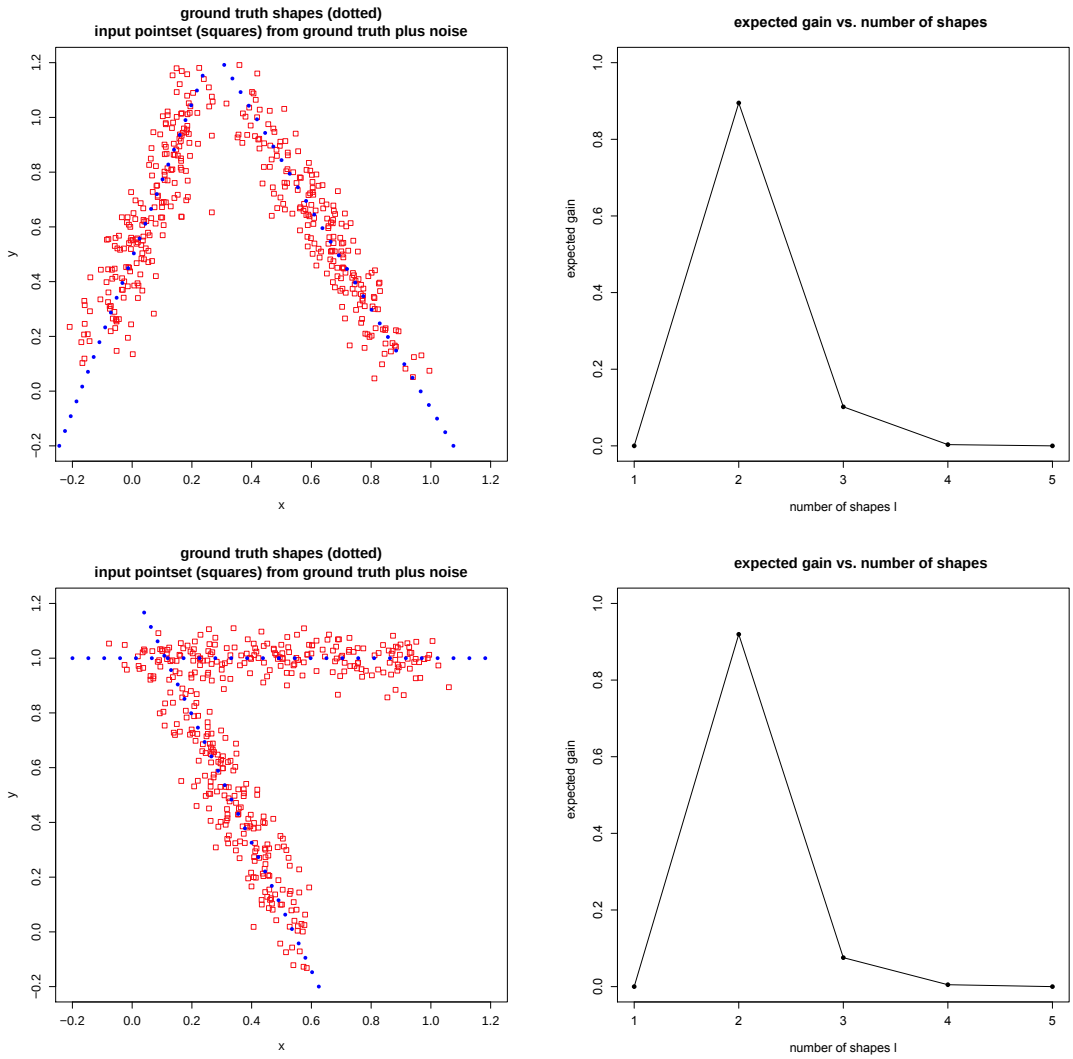

Figure 5.20: Number of Shapes – Result Set 2. (left) Inputs. (right) Expected posterior gain versus number of shapes l. (row 1) $l_{\text{true}} = 2$. (row 2) $l_{\text{true}} = 2$.

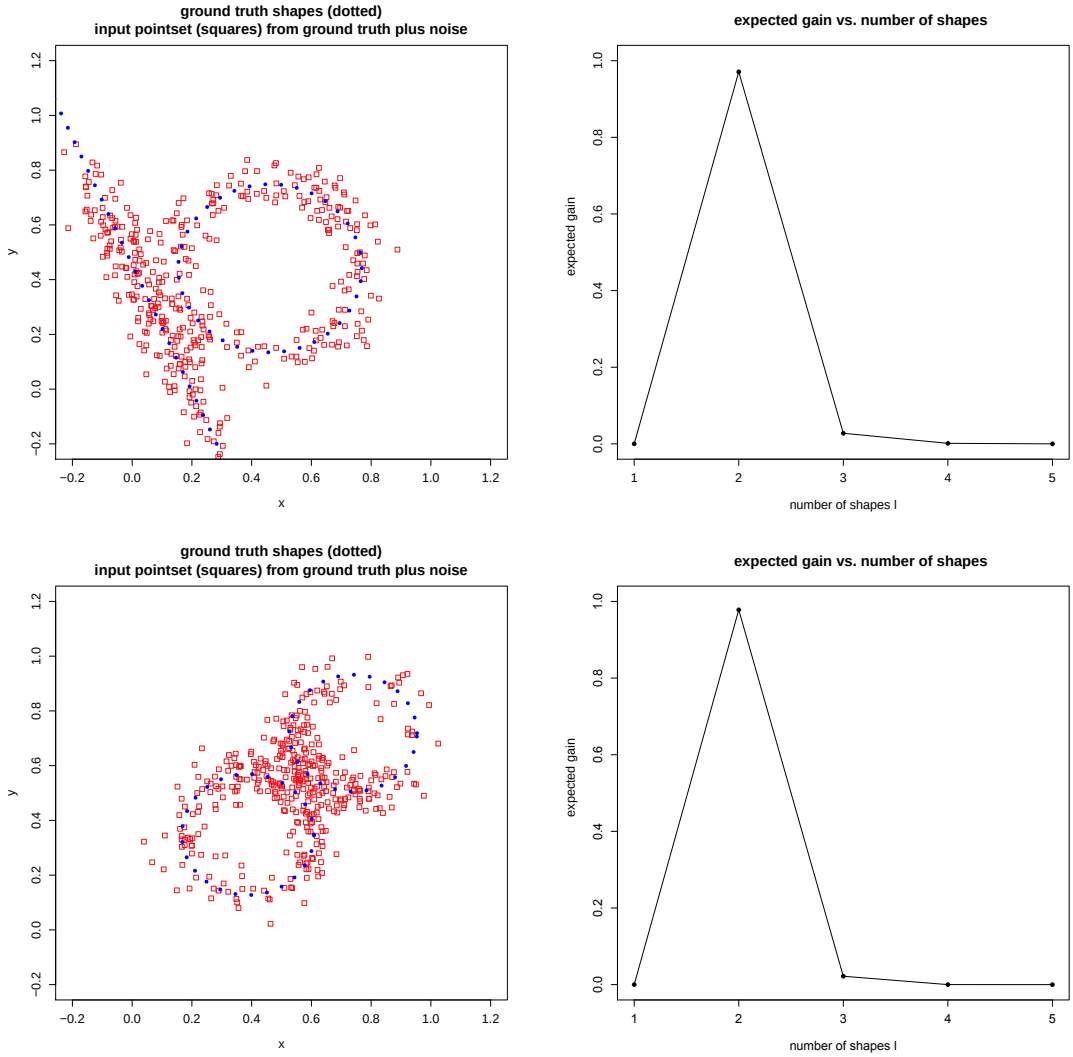

Figure 5.21: Number of Shapes – Result Set 3. (left) Inputs. (right) Expected posterior gain versus number of shapes l. (row 1) $l_{\text{true}} = 2$. (row 2) $l_{\text{true}} = 2$.

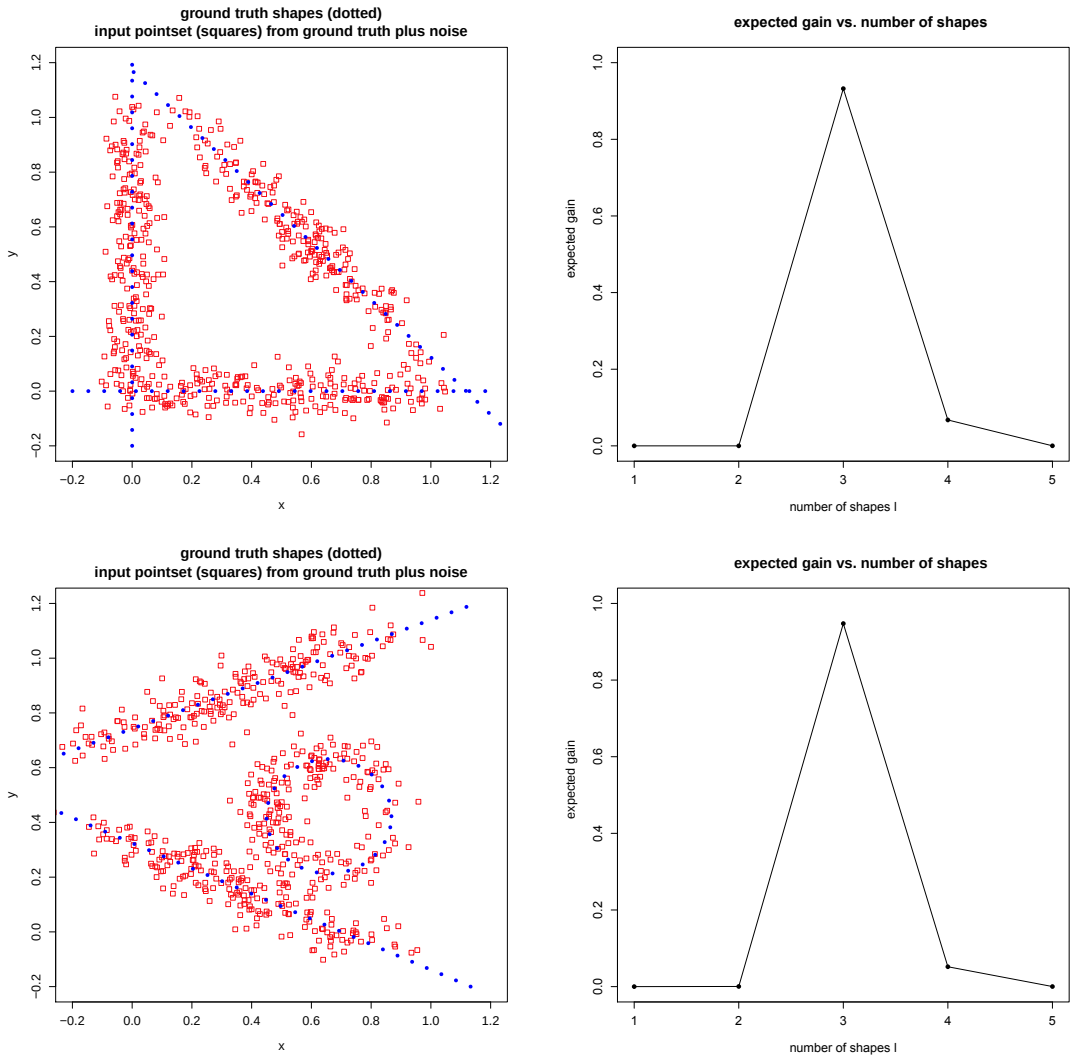

Figure 5.22: Number of Shapes – Result Set 4. (left) Inputs. (right) Expected posterior gain versus number of shapes l. (row 1) $l_{\text{true}} = 3$. (row 2) $l_{\text{true}} = 3$.

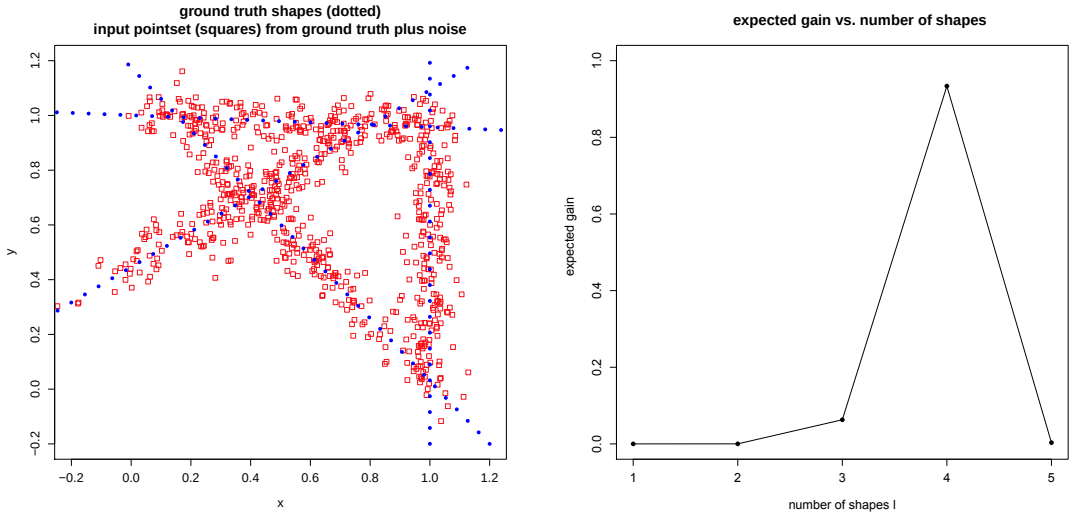

Figure 5.23: Number of Shapes – Result Set 5. (left) Inputs. (right) Expected posterior gain versus number of shapes l. (row 1) $l_{\text{true}} = 4$.

The MAP shape set with a high gain for each l contributes a relatively large value to the approximation of the expected gain:

$$\begin{aligned} g(l) = E[\text{gain}(\mathcal{S}, X)] &= \int_{\{\mathcal{S}:|\mathcal{S}|=l\}} \text{gain}(\mathcal{S}, X_n)\widehat{P}(\mathcal{S}|X_n)d\mathcal{S} \\ &\approx \sum_{k=1}^{K} \text{gain}(\mathcal{S}^{(k)}, X_n)\widehat{P}(\mathcal{S}^{(k)}, X_n). \end{aligned}$$

If $\mathcal{S}^{*,l}$ is the map shape set for size l, then it is instructive in understanding our posterior gain to look at the contribution $\text{gain}(\mathcal{S}^{*,l}, X_n)\widehat{P}(\mathcal{S}^{*,l}, X_n)$ to the integral broken down by summary (loss), completeness, and shape similarity costs. Here we use the term *cost* because we will actually give the quantities

$$\begin{aligned} \text{total cost} &= -\log(\text{gain}(\mathcal{S}^{*,l}, X_n)\widehat{P}(\mathcal{S}^{*,l}, X_n)) \\ &= \text{loss}(\mathcal{S}^{*,l}, X_n) + \text{cost}_{\text{complete}}(\mathcal{S}^{*,l}, X_n) + \text{cost}_{\text{shapesim}}(\mathcal{S}^{*,l}, X_n), \end{aligned}$$

num shapes	summary cost	completeness cost	shape similarity cost	total cost
1	0.28	2.34	0.00	2.62
2	0.53	1.37	1.93	3.83
3	1.61	1.66	6.17	9.44
4	5.82	1.80	9.74	17.35
5	7.19	1.55	20.08	28.82

Table 5.2: Decomposition of MAP costs in Figure 5.24 into the summary, completeness, and shape similarity costs.

where

$$\begin{aligned} \text{loss}(\mathcal{S}^{*,l}, X_n) &= \lambda_{\text{summary}} \overline{D}^2_{\mathcal{S}^{*,l}, X_n}, \\ \text{cost}_{\text{complete}}(\mathcal{S}^{*,l}, X_n) &= \lambda_{\text{complete}} \overline{D}^2_{X_n, \mathcal{S}^{*,l}}, \\ \text{cost}_{\text{shapesim}}(\mathcal{S}^{*,l}, X_n) &= \lambda_{\text{shapesim}} \overline{D}^{-2}_{\mathcal{S}^{*,l}}. \end{aligned}$$

First consider the example shown in the top row of Figure 5.19. For each number of shapes $l \in \mathcal{L}$, the MAP shape set is shown in Figure 5.24 and the decomposition of the total cost into the summary (loss), completeness, and shape similarity costs is shown in Table 5.2.

The true number of shapes in this example is $l_{\text{true}} = 1$. The MAP shape set for $l = 1$ gives a low cost. Note that when $l = 1$ the shape similarity cost is zero (for all examples, not just this one). This is because this term measures the similarity between shapes in the set. For $l = 2$, the MAP shape set essentially adds another line near the input points and the shape similarity term keeps the two lines slightly separated. The addition of a second line actually decreases the summary plus completeness cost compared to $l = 1$, but the shape similarity cost increases more and the total cost is still smaller for $l = 1$. To add lines near the input points to keep the summary and

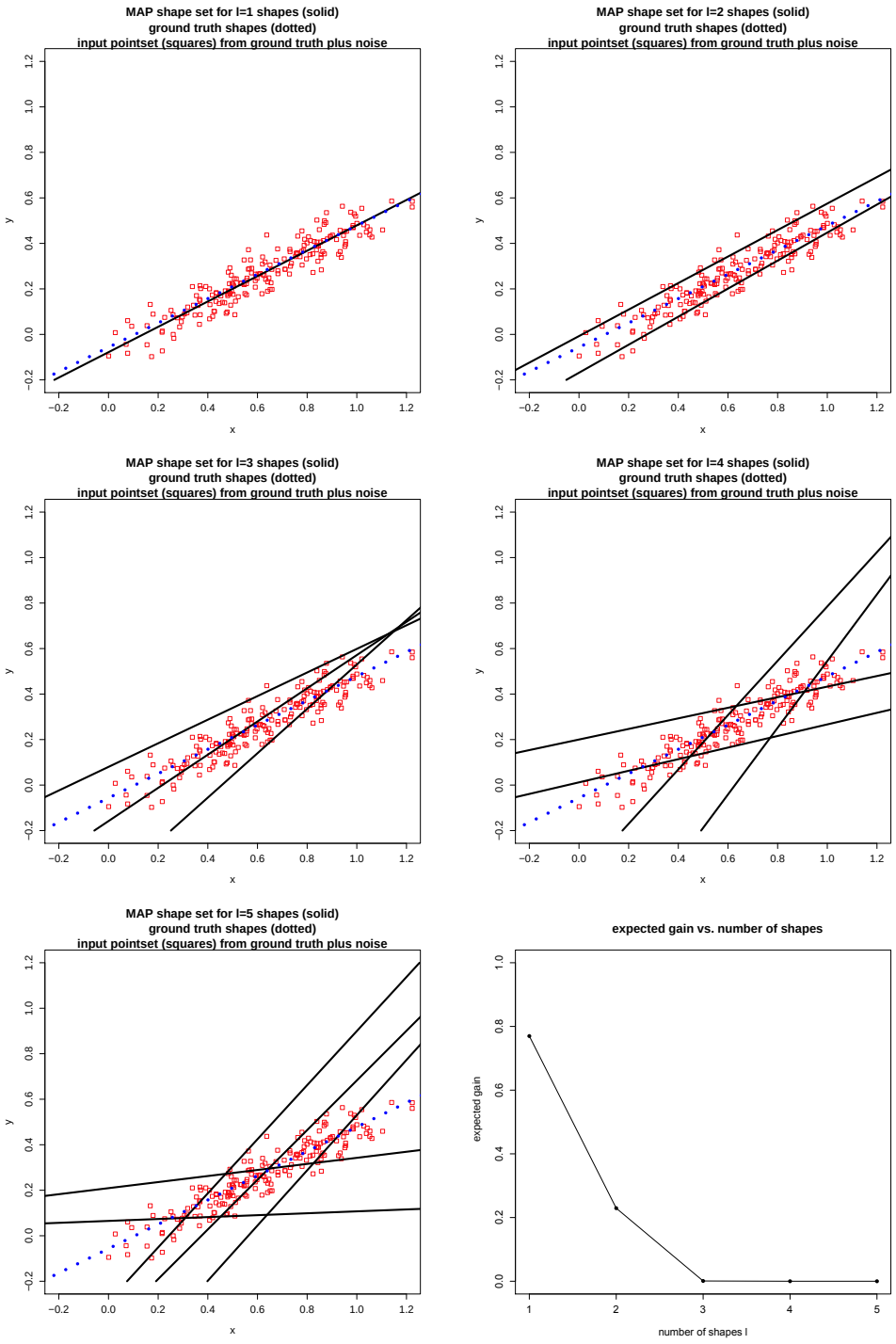

Figure 5.24: MAP shape set estimates over number of shapes l for the top row of Figure 5.19. From top-to-bottom, left-to-right: $l = 1$, $l = 2$, $l = 3$, $l = 4$, and $l = 5$. The bottom right plot shows the expected gain versus l.

num shapes	summary cost	completeness cost	shape similarity cost	total cost
1	3.70	65.85	0.00	69.55
2	0.27	2.67	0.09	3.03
3	0.42	2.22	2.56	5.20
4	0.59	1.81	6.25	8.65
5	2.24	2.01	10.47	14.72

Table 5.3: Decomposition of MAP costs in Figure 5.25 into the summary, completeness, and shape similarity costs.

completeness costs low, the lines must be close to one another and the shape similarity cost rises with increasing l (the last column of Table 5.2). The best configurations for $l \geq 3$ try to cover the input points while keeping a little distance between lines, but no configuration yields a smaller cost than the MAP shape set for $l = l_{\text{true}} = 1$.

Next consider the example shown in the top row of Figure 5.20. For each number of shapes $l \in \mathcal{L}$, the MAP shape set is shown in Figure 5.25 and the decomposition of the total cost into the summary (loss), completeness, and shape similarity costs is shown in Table 5.3.

In this example there are $l_{\text{true}} = 2$ lines. If only one line is allowed ($l = 1$), then the minimum cost is a line that crosses the ground truth lines in an attempt to be close to both lines. But the summary and completeness costs (first row of Table 5.3) are still high. When $l = 2$ lines are allowed, the MAP shape set is close to the ground truth lines and the summary and completeness costs are decreased significantly. Since the lines are well separated, the shape similarity cost increases only slightly and the total cost is decreased significantly for $l = 2$ compared to $l = 1$. When a third line is allowed, the MAP shape set essentially double covers one of the lines. The extra line allows the sum of the summary and completeness costs to be lower for $l = 3$ than for

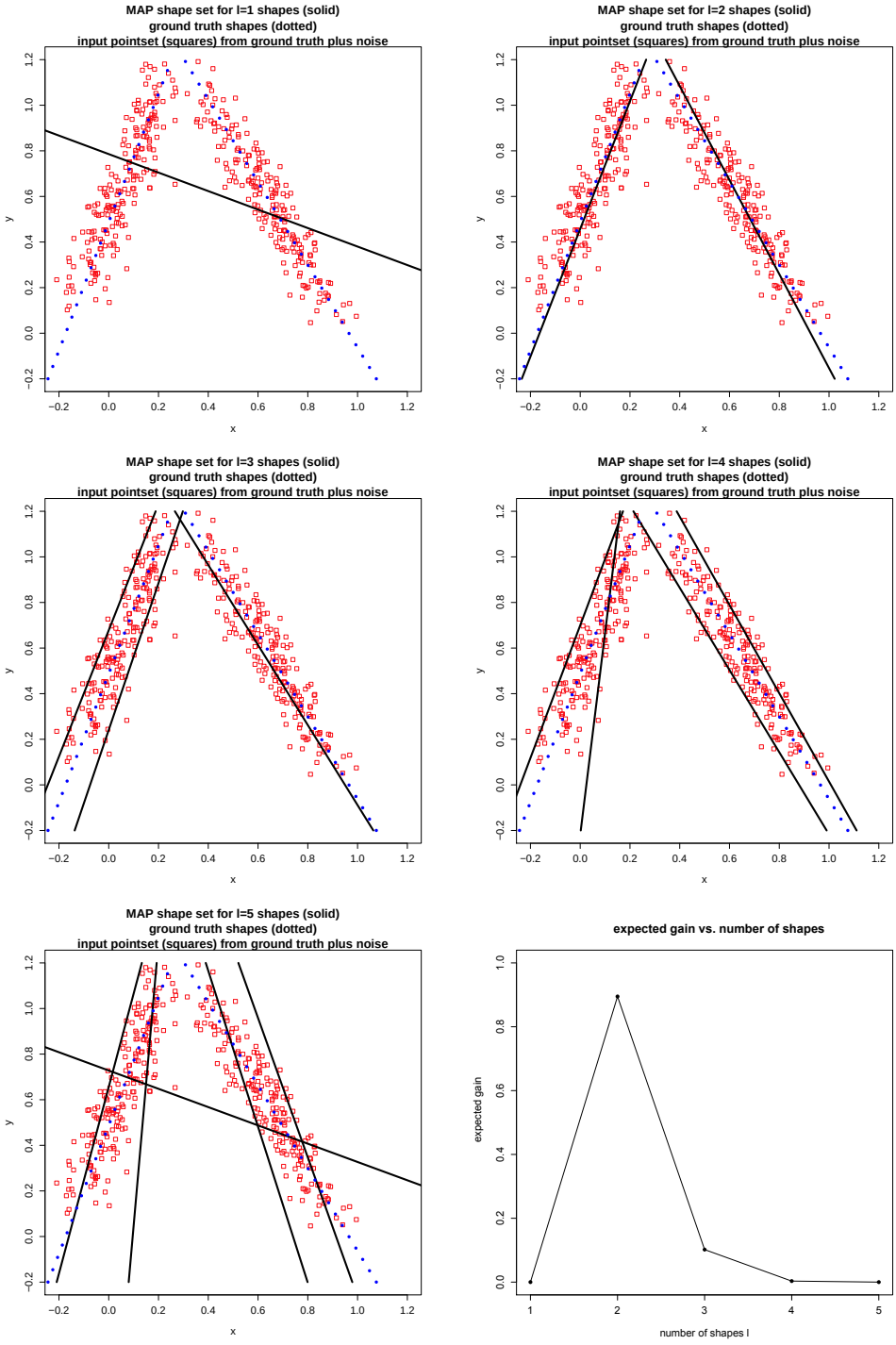

Figure 5.25: MAP shape set estimates over number of shapes l for the top row of Figure 5.20. From top-to-bottom, left-to-right: $l = 1$, $l = 2$, $l = 3$, $l = 4$, and $l = 5$. The bottom right plot shows the expected gain versus l.

num shapes	summary cost	completeness cost	shape similarity cost	total cost
1	0.65	11.92	0.00	12.57
2	0.33	2.27	0.41	3.02
3	0.53	1.81	4.24	6.58
4	1.34	1.83	6.39	9.56
5	3.03	1.64	10.11	14.77

Table 5.4: Decomposition of MAP costs in Figure 5.26 into the summary, completeness, and shape similarity costs.

$l = 2$, but the shape similarity cost rises more and $l = 2$ still gives the minimum cost. It is a similar story for $l = 4$, but now both ground truth lines are double covered. The best placement of a fifth line stays away from both double covered ground truth lines because of the shape similarity cost, but this model is clearly wrong and the cost is maximum for $l = 5$. The minimum cost occurs for the MAP shape set for $l = l_{\text{true}} = 2$.

Our third example is the one in the top row of Figure 5.21. For each number of shapes $l \in \mathcal{L}$, the MAP shape set is shown in Figure 5.26 and the decomposition of the total cost into the summary (loss), completeness, and shape similarity costs is shown in Table 5.4.

In this example there are $l_{\text{true}} = 2$ shapes, one circle and one line. When $l = 1$, the MAP shape set is a circle larger than the ground truth circle in order to cover some of the nearby points on the line. Some of the input points at the extremes of the line segment are not covered, so the completeness cost is still relatively high. Using a single line would not have been able to get close to as many of the input points as the single circle, and therefore the MAP shape set for $l = 1$ is the circle. The completeness cost is reduced significantly when $l = 2$ and a second shape can be used as a line to cover the entire ground truth line. When $l = 3$, the best model uses two circles to try to cover

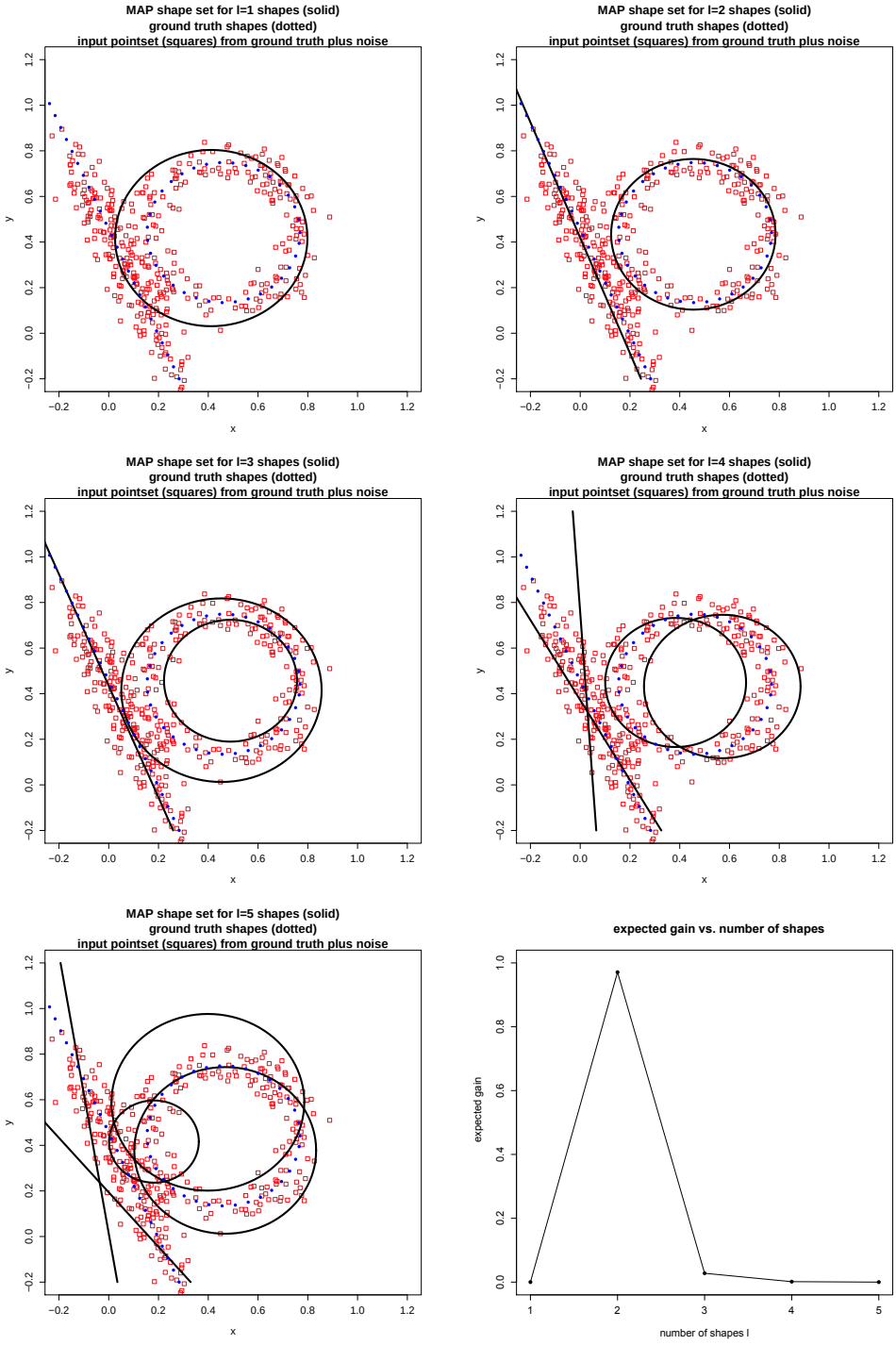

Figure 5.26: MAP shape set estimates over number of shapes l for the top row of Figure 5.21. From top-to-bottom, left-to-right: $l = 1$, $l = 2$, $l = 3$, $l = 4$, and $l = 5$. The bottom right plot shows the expected gain versus l.

the noisy point set from the one ground truth circle. The summary plus completeness cost decreases from $l = 2$, but the shape similarity cost keeps $l = 2$ as the better model. For $l = 4$, the best model double covers the line and the circle but the total cost still increases. For $l = 5$, the model is clearly wrong and the MAP shape set does not make much intuitive sense beyond trying to cover all the input points while keeping some distance between the shapes (thus driving the lines a little bit away from the ground truth line and the three circles). The minimum cost occurs for the MAP shape set for $l = l_{\text{true}} = 2$.

5.3.4 Selecting Candidate Points

The goal of the selection of a candidate point in sequential design is to find a point x at its most informative input location. Such active learning procedures are intended to provide efficient automatic exploration of the covariate space, thus guiding an on-line minimization of prediction error. During each active learning iteration will draw candidate locations $\tilde{X} = \{\tilde{x}_i\}_{i=1}^{M}$, and select the next design point to be $x^\star \in \tilde{X}$, which maximizes a heuristic statistic. For each selected point, we start a run of the system simulation, which in our cases executes the IFCS Simulink model. The result of this simulation (success of fail) is the true label $\tilde{y}$ for $\tilde{X}$.

In the literature, a number of selection heuristics have been developed (see Section 5.4.4.1 for a detailed discussion of related work). In a classification-based setting, a popular heuristics selects the point with the maximum entropy, where the entropy Y is defined by $Y = -\sum_{c \in c_1,..,c_n} p_c \log p_c$, given the posterior probability surface p_c for n

classes. Entropy is used in classification settings to assess response variability and to illustrate change in expected value of information over the input space. However, the entropy heuristics behaves very greedily [Gramacy and Polson, 2011] and is thus not useful for our purposes. For regression problems, two common heuristics are active learning MacKay (ALM, [MacKay, 1992]) and active learning Cohn (ALC, [Cohn, 1996]). An ALM scheme selects the $x^\star$ that leads to maximum variance for $y(x^\star)$, whereas ALC chooses $x^\star$ to maximize the expected reduction in predictive variance averaged over the input space. Again, these heuristics are in general not suited for the boundary-finding task, because they do not take the specifics of the boundaries into account and tend to also explore sparsely populated regions far away from current boundaries.

Finding a boundary between two classes can be considered as finding a contour with $a = 0.5$ in the response surface of the system response. Inspired by (Jones1998) and work on contour finding algorithms, we loosely follow [Ranjan *et al.*, 2008], and define our heuristics by using an improvement function. In order to use the available resources as efficient as possible for our contour/boundary finding task, one would ideally selects candidate points, which lie directly on the boundary, but that is unknown. Therefore, new trial points are selected, which belong to an ϵ-environment around the current estimated boundary. This means, that $0.5-\epsilon \leq \hat{y}(x) \leq 0.5+\epsilon$. New data points should maximize the the information in the vicinity of the boundary. Follwing [Jones *et al.*, 1998; Ranjan *et al.*, 2008], we define an improvement function fo x as

$$I(X) = \epsilon^2(x) - \min\{(y(x) - 0.5)^2, \epsilon^2(x)\} \tag{5.10}$$

here, $y(x) \sim N(\hat{y}(x), \sigma^2(x))$, and $\epsilon(x) = \alpha\,\sigma(x)$ for a constant $\alpha \geq 0$. This term defines an ϵ-neighborhood around the boundary as a function of $\sigma(x)$. This formulation [Ranjan *et al.*, 2008] makes it possible to have a zero-width neighborhood around existing data points. For boundary sample points, $I(X)$ will be large, when the predicted $\sigma(x)$ is largest.

The expected improvement $E[I(x)]$ can be calculated easily, again following [Ranjan *et al.*, 2008]. From Eq 5.10 we directly obtain

$$I(X) = \begin{cases} \epsilon^2(x) - (y(x) - 0.5)^2 & for \;\; 0.5 - \epsilon \leq \hat{y}(x) \leq 0.5 + \epsilon \\ 0 & otherwise \end{cases} \tag{5.11}$$

Using the fact that $y(x) \sim N(\hat{y}(x), \sigma^2(x))$

$$\begin{aligned} E[I(x)] &= \epsilon^2(x)\left[\Phi(\tfrac{0.5+\epsilon(x)-\hat{y}(x)}{\sigma x}) - \Phi(\tfrac{0.5-\epsilon(x)-\hat{y}(x)}{\sigma x})\right] \\ &\quad - \int_{0.5-\epsilon(x)}^{0.5+\epsilon(x)} (y-0.5)^2 \phi\left(\tfrac{y-\hat{y}(x)}{\sigma(x)}\right) dy \end{aligned} \tag{5.12}$$

Replacing $\epsilon(x)$ and some minor calculations ([Ranjan *et al.*, 2008], eq (14)) on the integral.

$$\begin{aligned} &\int_{0.5-\epsilon(x)}^{0.5+\epsilon(x)} (y-0.5)^2 \phi\left(\tfrac{y-\hat{y}(x)}{\sigma(x)}\right) dy \\ &= \int_{0.5-\epsilon(x)}^{0.5+\epsilon(x)} (y-\hat{y}(x))^2 \phi\left(\tfrac{y-\hat{y}(x)}{\sigma(x)}\right) dy + \\ &\quad (\hat{y}(x)-0.5)^2 \left[\Phi\left(\tfrac{0.5+\epsilon(x)-\hat{y}(x)}{\sigma(x)} + \alpha\right) - \Phi\left(\tfrac{0.5-\epsilon(x)-\hat{y}(x)}{\sigma(x)} - \alpha\right)\right] \\ &\quad 2(0.5-\hat{y}(x))\sigma^2(x)\left[\phi\left(\tfrac{0.5+\epsilon(x)-\hat{y}(x)}{\sigma(x)} + \alpha\right) - \phi\left(\tfrac{0.5-\epsilon(x)-\hat{y}(x)}{\sigma(x)} - \alpha\right)\right] \end{aligned} \tag{5.13}$$

Finally

$$
\begin{aligned}
E[I(x)] &= -\int_{0.5-\alpha s(x)}^{0.5+\alpha s(x)} (y-\hat{y}(x))^2 \phi\left(\tfrac{y-\hat{y}(x)}{\sigma(x)}\right) dy \\
&\quad +2(\hat{y}-0.5)\sigma^2(x)\left[\phi\left(\tfrac{0.5-\hat{y}(x)}{\sigma(x)}+\alpha\right)-\phi\left(\tfrac{0.5-\hat{y}(x)}{\sigma(x)}-\alpha\right)\right] \\
&\quad +(\alpha^2\sigma^(x)-(\hat{y}(x)-0.5)^2)\left[\Phi\left(\tfrac{0.5-\hat{y}(x)}{\sigma(x)}+\alpha\right)-\Phi\left(\tfrac{0.5-\hat{y}(x)}{\sigma(x)}-\alpha\right)\right]
\end{aligned}
\tag{5.14}
$$

Each of these three terms are instrumental in different areas of the space. The first term summarized information from regions of high variability within the ϵ-band. The integration is performed over the ϵ-band as $\epsilon(x) = \alpha\,\sigma(x)$. The second term is concerned with areas of high variance farther away from the estimated boundary. Finally, the third term is active close to the estimated boundary. After the expected improvement has been calculated, the candidate point is selected as the point, which maximizes the expected improvement: $m = argmax_x E[I(x)]$.

Using this heuristic yielded promising results for one or more boundaries that are located close together. Most of the newly selected data points are located close to the boundary and they nicely are located along the boundary line.

For two boundaries located farther apart, this heuristic usually explores one boundary preferably, or sometimes even fails to detect and explore the second boundary. In particular in cases where the sizes of the boundaries strongly differ, this can subsequently lead to problems in shape estimation. The reason for this behavior is that the current ϵ-band is too small to reach out to the other boundaries. Although increasing α helps, it will result in a much larger number of data points necessary to characterize the boundary. Experiments with dynamically decreasing α during active learning will

be future work.

For our experiments, a simple switch of the heuristics addressed that problem: in approximatly 10% of all candidate selections, ALM or ALC is used instead of the above selection heuristic. This switch forces the active learning to "on and off" explore some new areas with high variance that can be away from known boundaries.

5.3.5 Improvements

The dynatree algorithm is partitioning the space according to a splitting rule p_{split}, which gives an indication if a leaf node should be splitted up or not. For any given leaf node η in tree $\mathcal{T}$, the split probability is $p_{\text{split}}(\mathcal{T}, \eta) = \alpha(1 + D_\eta)^{-\beta}$ for $0 \leq \alpha \leq 1$ and $\beta > 0$ (defaults set to $\alpha = 0.95, \beta = 2$). In [Taddy *et al.*, 2011], the location of the split and its coordinate $\langle i, x \rangle_{eta}$ for $1 \leq i \leq p$ is a discrete uniform distribution over all split points. It also restricts that partitions with too few points are not subject to further splits. This uniform distribution is oblivious to the fact that we are searching for boundaries. We propose the use of a depth-dependent split probability $p_{\text{split}}(\mathcal{T}, \eta)$, which takes into account if a current leaf contains a boundary (or is close to a boundary). Informally, such leaf nodes should have a higher prior probability to be split, resulting in a much finer partition grid close to the boundary.

5.4 Background and Related Work

5.4.1 DynaTree

5.4.1.1 Dynamic Trees

In order to properly carry out our task for finding boundaries in a high dimensional setting, dynamic trees [Taddy *et al.*, 2011] are used. Compared to standard basis function models for nonparametric regression, the dynamic trees have some striking advantages (e.g., flexible response surface, nonstationarity, heteroskedasiticity), which are important to solve our task. Dynamic trees are easy to specify and allow for conditional inference, given the global partition tree-state, to be marginalized over all model parameters. Prediction is very fast, requiring only a search for the rectangle containing a new x, and individual realizations yield an easily interpretable decision tree for regression and classification problems. Most importantly, dynamic trees models are suitable for sequential particle algorithms, which are necessary for boundary detection using computer experimental design.

The use of partition trees to represent input-output relationships is a classic nonparametric modeling technique. A decision tree is imposed with switching on input variables and terminal node predictions for the relevant output. Although other schemes are available (e.g., based on Voronoi tessellations), the standard approach relies on a binary recursive partitioning of input variables, as in the classification and regression tree (CART) algorithms of [Breiman *et al.*, 1984]. This forces axis-aligned partitions, although pre-processing data transformations can be used to alter the partition space.

In general, the computational and conceptual simplicity of rectangular partitions will favor them over alternative schemes. Figure 5.27 shows how the paritioning is encoded in the tree. Each inner node shown contains the split value and the dimension for which the partition value is applicable. For example, the root node splits dimension 1 into values less than 2 by going into the left subtree (see the top left leaf). By going to the right, values of greater or equal than 2 are selected and the partition is further refined in that subtree. Figure 5.28 shows such a partition tree in practice.

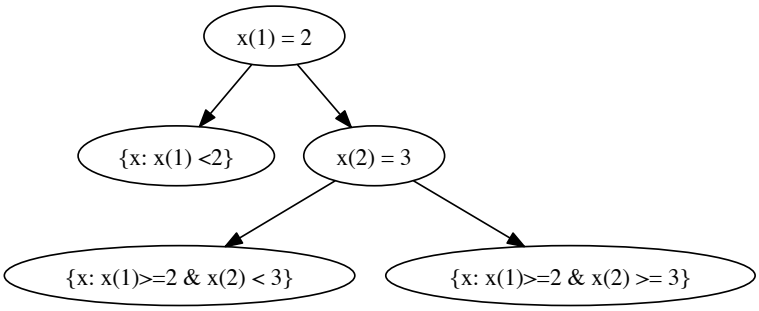

Figure 5.27: Dynamic tree with split values and dimensions

5.4.1.2 Definition of the tree

Consider covariates $x^t = \{x_s\}_{s=1}^t$. A corresponding tree T consists of a hierarchy of nodes associated with different subsets of x_t. The subsets are determined through a series of splitting rules, and these rules also dictate the terminal node associated with any new x. Every tree has a root node, RT , which includes all of x_t, and every node is itself a root for a sub-tree containing nodes (and associated subsets of x_t) below in the

hierarchical structure defined by T . A node is positioned in this structure by its depth, D, defined as the number of sub-trees of T other than T which contain. The tree is completed with a decision rule (i.e., a simple regression or classification model) at each leaf. Suppose that every covariate vector xs is accompanied by response ys, such that the complete data is $[\mathbf{x}, y]^t = \{x_s, y_s\}_{s=1}^t$. Then, with regression models parameterized by for each leaf $\eta \in L_T$ independence across tree partitions leads to likelihood

$$p(y^t|x^t, T, \theta) = \prod_{\eta \in L_T} p(y^\eta|x^\eta, \theta_\eta)$$

where $[x, y]^\eta = \{x_i, y_i : x_y \in \eta\}$ is the data subset in leaf η. A novel approach to regression trees was first described by Chipman, George, and McCulloch [Chipman *et al.*, 1998; Chipman *et al.*, 2002], who designed a prior distribution, $\pi(T)$, over possible partition structures. This allows for coherent inference via the posterior, $p(T|[x, y]^t) \propto p(y^t|T, x^t)\pi(T)$, including the assessment of partition uncertainty and the calculation of predictive bands.

5.4.1.3 Split Rules

Any given leaf node may be split with depth-dependent probability $p_{split}(T, \eta) = \alpha(1 + D_\eta)^{-\beta}$, where $\alpha, \beta > 0$. The coordinate (i.e. dimension of x) and location of the split, $(i, x)_\eta$, have independent prior $p_{rule}(T, \eta)$, which is typically a discrete uniform distribution over all potential split points in $x^\eta = \mathbf{x}^t \cap \eta$. Implicit in the prior is the restriction that a partition may not be created if it would contain too few data points

to fit the leaf model. Ignoring invalid partitions, the joint prior is thus

$$\pi(T) \propto \prod_{\eta \in I_T} p_{split}(T, \eta)\pi(T) \propto \prod_{\eta \in L_T} (1 - p_{split}(T, \eta))$$

That is, the tree prior is the probability that internal nodes have a split and leaves nodes have not.

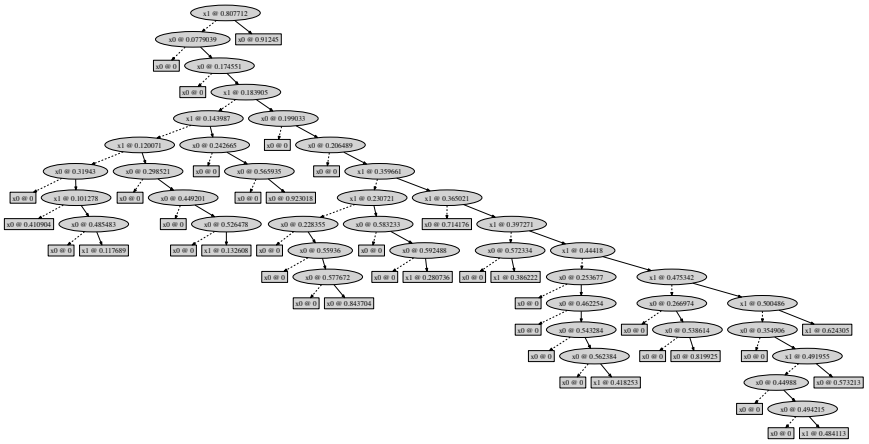

Figure 5.28: Graphical tree representation for a dynamic tree for larger 2D data set

5.4.1.4 Dynamic Trees

We now redefine partition trees as a dynamic model for predictive uncertainty. We introduce a model state T_t, which includes the recursive partitioning rules associated with x_t, the set of covariates observed up-to time $t-1$ defines the mechanisms of state transition $T_{t-1} \rightarrow T_t$, as a function of x_t, the newly observed covariates, such that Tt is only allowed to evolve from T_{t-1} through a small set of operations on partition structure in the neighborhood of x_t. That is, we specify a prior distribution for the evolution, $p(T_t|T_{t-1}, x^t)$. The tree likelihood $p(y_t|T_t, x_t)$ as a product of leaf node

marginal likelihoods, and hence allows us to assign posterior weight over the discrete set of potential trees generated through the evolution prior. details the conditional predictive distribution and describes model particulars for three regression leaves: constant, linear, and multinomial. A particle learning algorithm for on-line posterior simulation is outlined below. In this, each particle consists of a tree (partitioning rules) and sufficient statistics for leaf node predictive models, and our filtering update at time t combines small tree changes around x_t with a particle resampling step that accounts for global uncertainty. Finally, we describe marginal likelihood estimation.

Given these three possible moves, we can now define an evolution prior as the product of two parts: a probability on each type of tree move and a distribution over the resultant tree structure. In the former case, we assume that possible moves among stay, prune, and grow are each a priori equally likely. For the latter distribution, we build on the inferential framework and assume a CGM prior for tree structure such that (Tt) is as in (2) based on $p_{split}(T_t, \eta) = \alpha(1 + D_\eta)^{-\beta}$ We then have $p(T_t|T_{t-1}, x_t) \propto \sum_{m \in M(x^t)} p_m \pi(T^m)$, where T_m is the tree that results from applying move m to T_{t-1} and $M(x^t)$ is the set of possible moves. Hence, one may view $p(T_t|T_{t+1}, x^t)$ as a covariate dependent prior for the next tree,

Our dynamic tree model is such that posterior inference is driven by two main quantities: the marginal likelihood for a given tree and the posterior predictive distribution for new data. This section establishes these functions for dynamic trees, and quickly details exact forms for our three simple leaf regression models. First, with each leaf $\eta \in L_{T_t}$, the likelihood function is available after marginalizing over regression

model parameters as

$$p(y^t|T_t, x^t) = \prod_{\eta \in L_{T_t}} p(y^\eta|x^\eta) = \prod_{\eta \in L_{T_t}} \int p(y^\eta|x^\eta, \theta_\eta) d\pi(\theta_\eta)$$

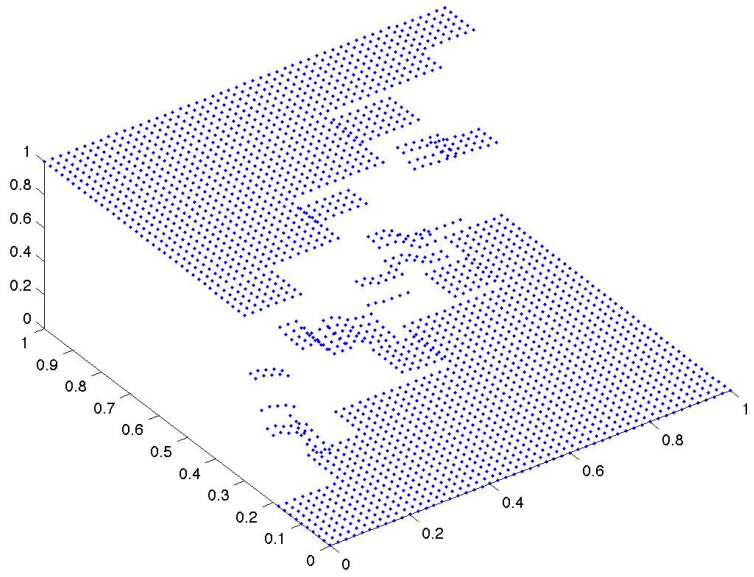

Figure 5.29: Posterior representation of partitions for data set with linear boundary

5.4.2 Finding boundaries

Each data point, describing one simulation run (experiment) is defined as $\mathbf{x} = \langle P_1, \ldots, P_p \rangle$, where P_i are the input parameter settings and the outcome $o(\mathbf{x}) \in \{success, failure\}$. Thus these data define a classification problem with $C = 2$ classes. Informally, a boundary can be found between regions, where all experiments yield success $p(\mathbf{x} = success) = 1$ and those, where the experiments do not meet the success

criterion $p(\mathbf{x} = failure) = 1$. Therefore, we can define a point $\mathbf{x}$ to be on the boundary if $p(\mathbf{x} = success) = p(\mathbf{x} = failure) = 0.5$. Although this condition can easily be generalized to more than 2 classes, in this work, we will focus on $C = 2$. Figure 5.30 shows a simple example. The surface denotes $p(\mathbf{x} = success)$ over two parameters P_1, P_2. The actual boundary is shown as a red line.

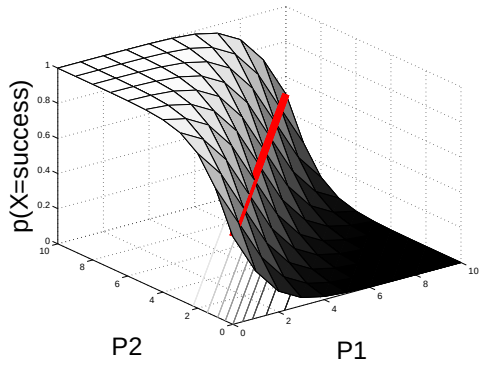

Figure 5.30: Probability surface and boundary line

A common metric to characterize points on the boundary is based upon the entropy, The entropy $entr = -\sum_{c \in c_1,..,c_C} p(\mathbf{x} = c) \log p(\mathbf{x} = c)$ becomes maximal at the boundary. In cases of more than two classes, [Gramacy and Polson, 2011] use a BVSB (Best vs. Second Best) strategy. [Wickham, 2008] defines a metric *advantage*

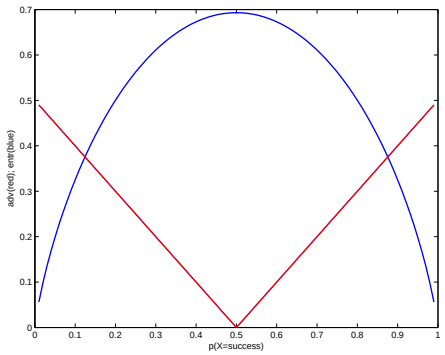

Figure 5.31: Advantage (red) and entropy (blue) metrics

as essentially $adv(\mathbf{x}) = |p(\mathbf{x} = success) - p(\mathbf{x} = failure)|$. Then [Wickham, 2008] considers points with minimal advantage to be close to the boundary. In the general case with more than two dimensions, [Wickham, 2008] proposed to use the difference between the two most likely classes. Figure 5.31 shows that both metrics are expressing essentially the same, as their extreme values are reached with $p(\mathbf{x} = success) = 0.5$. There are two basic methods: explicitly from knowledge of the classification function, or by treating the classifier as a black box and finding the boundaries numerically. For some classifiers it is possible to find a simple parametric formula that describes the boundaries between groups, for example, LDA or SVM. Most classification functions

can output the posterior probability of an observation belonging to a group. Much of the time we do not look at these, and just classify the point to the group with the highest probability. Points that are uncertain, i.e. have similar classification probabilities for two or more groups, suggest that the points are near the boundary between the two groups. For example, if point A is in group 1 with probability 0.45, and group 2 in probability 0.55, then that point will be close to the boundary between the two groups. We can use this idea to find the boundaries. If we sample points throughout the design space we can then select only those uncertain points near boundaries. The thickness of the boundary can be controlled by changing the value which determines whether two probabilities are similar or not. Ideally, we would like this to be as small as possible so that our boundaries are accurate. Some classification functions do not generate posterior probabilities. In this case, we can use a k-nearest neighbors approach. Here we look at each point, and if all its neighbors are of the same class, then the point is not on the boundary and can be discarded. The advantage of this method is that it is completely general and can be applied to any classification function. The disadvantage is that it is slow ($O(n^2)$), because it computes distances between all pairs of points to find the nearest neighbors. In general, finding of the boundaries faces the "curse of dimensionality": As the dimensionality of the design space increases, the number of points required to make a perceivable boundary (for fitting or visualization purposes) increases. This problem can be attacked in two ways, by increasing the number of points used to fill the design space (uniform grid or random sample), or by increasing the thickness of the boundary.

5.4.3 Visualizing boundaries

In general, the input to a classification algorithm is high dimensional, and the resulting boundaries will thus be high dimensional and perhaps curvilinear or multi-faceted. [Wickham, 2008] discusses methods for understanding the division of space between the groups, and provides an implementation in an R package *classifly* [Wickham, 2012], which links R to the GGobi data visualization system [GGobi, 2008]. It provides a graphical environment for experimenting with classification algorithms and their parameters and then viewing the results in the original high dimensional design space. Since the approach of [Wickham, 2008] aims at visualization of the boundary, they took advantage of the fact that a line is indistinguishable from a dense series of points. It is easier to describe the position of a discrete number of points instead of all the twists and turns a line might make. The challenge then becomes generating a sufficient number of points that an illusion of a boundary is created. [Wickham, 2008] generates a perturbed grid of high-dimensional data points and calculates the advantage metric for the posterior. The 15% quantile is used for visualizing the boundary. The tool also draws two boundary lines, one for each group on the boundary. In contrast to our approach, [Wickham, 2008] uses a large number of data points to visualize the boundaries in higher dimensions. The high cost of new data points in our setting makes this visualization approach less suitable.

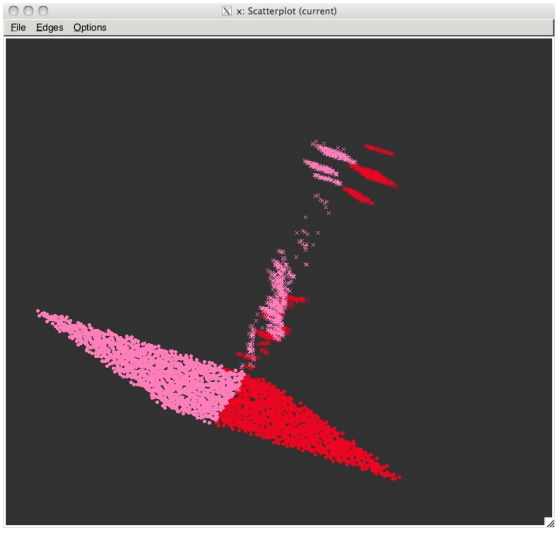

Figure 5.32: GGobi representation of DT boundary for data set with linear boundary

5.4.4 Computer Experiment Design

Computer simulation of a complex system like those discussed above, is frequently used as a cost-effective means to study complex physical and engineering processes. It typically replaces a traditional mathematical model in cases where such models do not exist or cannot be solved analytically. While a computer simulator can only find an approximate solution, simulation is often be viewed as an inexpensive way to gain insight into a system. However, it can still be computationally costly. Therefore it is important to perform only a small number of simulation trials and the runs must be selected carefully. computer experiment. A computer experiment frequently involves the modeling of complex systems using a deterministic computer code, which means replicate runs of the same inputs will yield identical responses. To deal with the lack of randomness in the response, [Sacks *et al.*, 1989] proposed modeling the response as a

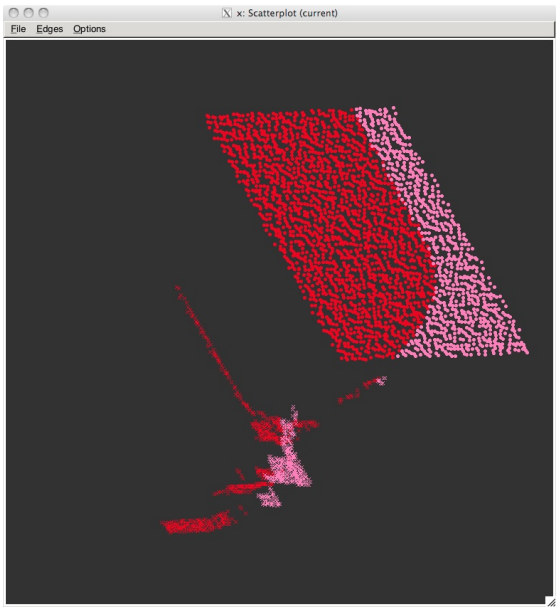

Figure 5.33: GGobi representation of DT boundary for data set with quadratic boundary

realization from a Gaussian stochastic process (GASP). Most recent work on the design of computer experiments has focused on experiments where the goal is to fit a response surface or to optimize a process. It is important to note that the IFCS controller, despite being "adaptive" is, in this sense a purely deterministic system.

Active learning, or sequential design of experiments (DOE), in the context of estimating response surfaces (in our case boundaries), is called adaptive sampling. Adaptive sampling starts with a relatively small space-filling input data, and then proceeds by fitting a model, estimating predictive uncertainty, and then choosing future samples with the aim of minimizing some measure of uncertainty, or to try to maximize information. The process repeats until some threshold in predictive uncertainty or information is met, or a maximum number of samples have been taken. In this iterative fashion the model adapts to the data, and the new data either reinforces or suggests a

modification to the old model.

Active learning is defined by contrast to the passive model of supervised learning where all the labels for learning are obtained without reference to the learning algorithm, while in active learning the learner interactively chooses which data points to label. The hope of active learning is that interaction can substantially reduce the number of labels required, making solving problems via machine learning more practical. This hope is known to be valid in certain special cases, both empirically and theoretically.

5.4.4.1 Active Learning

We now outline our general approach to active learning. The key step in our approach is to define a notion of a model M and its model quality (or equivalently, model loss, $Loss(M)$) . The definition of a model and the associated model loss can be tailored to suit the particular task at hand. Now, given this notion of the loss of a model, we choose the next query that will result in the future model with the lowest model loss. Note that this approach is myopic in the sense that we are attempting to greedily ask the single next best query. In statistics, a standard alternative to minimizing the expected loss is to minimize the maximum loss [Wald, 1950]. In other words, we assume the worst case scenario: for us, this means that the response x will always be the response that gives the highest model loss. $Loss(q) = \max_x Loss(M)$: If we use this alternative definition of the loss of a query in our active learning algorithm we would be choosing the query that results in the minimax model loss. our general approach for active

learning is as follows. We first choose a model and model loss function appropriate for our learning task. We also choose a method for computing the potential model loss given a potential query. For each potential query we then evaluate the potential loss incurred and we then chose to ask the query which gives the lowest potential model loss. We will perform active learning with new data until the boundary is characterized with sufficient accuracy and confidence. Also the whole space has been sufficiently explored, to not miss any boundary areas in the space.

5.4.4.2 ALM and ALC

One key part of the active learning algorithm is the specialized function f_s. The ALM/ALC algorithms are suitable for classification but not primarily for boundary detection [Gramacy, 2005]. Both ALM and ALC algorithms are based on the posterior predictive distribution $P(z|x)$. For example, consider an approach which maximizes the information gained about model parameters by selecting the location $\mathbf{x}$ which has the greatest standard deviation in predicted output. This approach has been called ALM for Active Learning-Mackay, and has been shown to approximate maximum expected information designs [MacKay, 1992]. MCMC posterior predictive samples provide a convenient estimate of location-specific variance, namely the width of predictive quantiles. An alternative algorithm is to select Σ^2 minimizing the expected reduction in squared error averaged over the input space [Cohn, 1996], called ALC for Active Learning-Cohn. Rather than focusing on design points which have large predictive variance, ALC selects configurations that would lead to a global reduction in predictive variance.

An ALM scheme selects the $x^\star$ that leads to maximum variance for $y(x^\star)$, whereas ALC chooses $x^\star$ to maximize the expected reduction in predictive variance averaged over the input space. A comparison between approaches depends upon the model and the application, however it may be shown that both approximate a maximum expected information design and that ALC improves upon ALM under heteroskedastic noise. Both heuristics have computational demands that grow with $|\tilde{X}|$: ALC requires time in $O(|\tilde{X}|^2)$, wheres ALM is in $O(|\tilde{X}|)$. [Taddy *et al.*, 2011] shows that ALM and ALC methods compare favorably to existing MCMC-based alternatives. Constant and linear leaf models lead to closed-form calculations of heuristic functionals conditional on a given tree and can be easily integrated into the particle filter framework: the necessary statistics are evaluated across candidates, for each particle, to obtaine the optimal $x^\star$. The trees are then updated for $y(x^\star)$, and the process is repeated.

More specifically ALM sekes to maximize $Var(y(x))$. For a given tree T such that x is allocated to leaf node $\eta \in L_{T_t}$, let $\mu_\eta(x) = E[y(x)|\eta]$ and $v_\eta = Var(y(x)|\eta)$ denote the conditional predictive mean and variance respectively for $y(x)$. According to [Taddy *et al.*, 2011] the unconditional predictive variance for a given particle set $\{T_t^{(}i)\}_{i=1}^N$ can be calculated as

$$\begin{aligned} Var(y(x)) &= E[Var(y(x)|T) + Var(E[y(x)|T]) \\ &\approx \tfrac{1}{N}[\textstyle\sum_{i=2}^N v_\eta^{(i)}(x) + \mu_\eta^{(i)}(x)^2] - [\tfrac{1}{N}\textstyle\sum_{i=1}^N \mu_\eta^{(i)}(x)]^2 \end{aligned}$$

which is evaluated for all $x \in \tilde{X}$.

The ALC metric can be calculated for regression models by maximizing the sum of $\Delta\sigma^2_{x^\star}(x'|T_t)$ over both $x \in \tilde{X}$ and the trees in the particle set $T_t \in \{T_t^{(}i)\}_{i=1}^N$

[Taddy *et al.*, 2011].

Another heuristic for selection of the next point can be based upon the expected improvement (EI) statistic [Jones *et al.*, 1998]. The improvement of point x is defined as $I(x) = max(f_{min} - Z(x), 0)$, where f_{min} is the current minimum and $Z(x)$ is the posterior predictive distribution. Then, the new point $x^\star$ is selected as

$$x^\star = \arg\max_{x \in \tilde{X}} E\{I(x)\}$$

. [Gramacy and Lee, 2010] use this statistic in the framework of GP. In order to handle constraints, the EI statistic has been extended to the integrated expected conditional improvement (IECI) [Gramacy and Lee, 2010]. Here a conditional improvement at a reference point y is defined as $I(y|x) = max(f_{min} - Z(y|x), 0)$. Given a density $g(y)$, the ICEI is the defined as

$$E_g\{I(x)\} = -\int_{\tilde{X}} E\{I(y,x)\}g(y)dy.$$

5.4.4.3 Entropy-based Selection

Traditionally, ALM, and ALC selection is very powerful, but candidate points are not concentrated along the boundaries [Gramacy, 2005] (pg. 86). [Gramacy and Polson, 2011] uses entropy e (and BVSB "Best vs Second Best") to guide the sequential algorithm. The entropy is defined as $e = -\sum_{c \in c_1,..,c_n} p_c \log p_c$ for n classes. Performance is improved, if the calculation is restricted to include only the two classes with highest posterior p, thus only taking into account the best and second best class only. In our case ($n = 2$) the standard definition of entropy and BVSB coincide. Although selection

methods based upon the entropy (or advantage) tend to select points near the boundary as boundary points are the most "chaotic". However, those selection methods tend to be too greedy when using off-grid data [Gramacy and Polson, 2011], i.e., new points are selected in areas, which are already explored. [Gramacy and Polson, 2011] proposed to use kernels, but according to the authors, no substantial improvement could be obtained.

5.4.5 Sensitivity Analysis

In a high-dimensional space, a possible reduction of the dimensionality is of high importance, because it can dramatically reduce the number of simulation runs necessary to find the boundary. Typical techniques for sensitivity analysis include PCA. [Taddy *et al.*, 2011] describes how a sensitivity analysis can be carried out on the framework of dynamic trees. In all cases, the aim is to find a transformation T such that the data, after undergoing the transformation are of much lower dimensionality and no important information has been lost. Most modern data mining and classification approaches use these techniques. It should be noted that, obviously, when we want to fit the boundary shapes, these shapes must be transformed as well.

For sequential and active learning algorithms, however, the situation is quite different. If a sensitivity analysis is carried out on the initial data set D_0, a transformation T_0 is established, which then is used for further processing all data. If we assume a sparse set of initial data, a sensitivity analysis on these data might produce unfavorable results. Due to lack of data points in a certain area, boundaries will not be found, because the initial sensitivity analysis removed that area. For example, if D_0

only contains few data points with the same label in variable 10, sensitivity analysis will most likely remove that dimension, because it does not contribute. A data set, which is more dense would contain data points with success and failure labels for this variable.

Obviously, a large and dense D_0 will overcome this issue, however, this is not in alignment with our sequential approach, where one wants to start with a small D_0 and successively add new data points. If the initial data set has a certain guaranteed structure rather than based upon a Latin hyper square, sensitivity analysis on the initial D_0 can be helpful. In our algorithm, we use n-factor combinatorial exploration (Section 5.3.2.1) to obtain the initial data set. This algorithm guarantees that D_0 contains all possible values for each variable, pairs of variables, triples, until n-tuples. Only few combinations of $n+1$ tuples are present in D_0. For the continuous parameters used in our work, a uniform random number is drawn from one of the k (here: 5) bins covering the entire range for each variable. We can use these properties about the coverage and density for the sensitivity analysis of the initial data set. If a boundary is constrained to $1, \ldots, n$ dimensions and the size of the boundary spans at least $1/k$ of the variable range, then that boundary can be detected in D_0 and remains detectable after dimension reduction. Because n-factor combinatorial exploration has been used to generate D_0, we know that values from each of the k bins for up to n-tuples are present in D_0. So if a boundary manifests itself in $d \leq n$ dimensions and it is at least of a minimal size, then we know that there are at least two data points with different labels, marking a boundary. If dimension reduction is limited to less or equal n dimensions, this boundary remains detectable after dimension reduction.

Sensitivity analysis can answer a number of questions, which are important to the analyst. Even in the case, the domain expert has a very good knowledge of the shapes and parameters of the boundaries, results of any sensitivity analysis should be considered carefully, because it may indicate unknown additional parameter dependencies (not considered in the design), which can pose a safety risk. On the other hand, missing sensitivity of a parameter might point to an error in the simulation or the system.

5.5 Experiments and Results

5.5.1 Artificial Data Set

For selected experiments, we use artificial data, where the (known) boundary shapes are hyperplanes and/or spheres. We limit all data to a 0...1 cube in each dimension. For illustration purposes, we use a 2D data set; the other experiments are carried out using a 5D data set. For all experiments we start with $N_{init} = 126$ data points, which are the result of an 3-factor, 10-bin combinatorial experiment, or a latin hyper square with the same number of data points. After that, active learning with the selected update rule(s) for additional N points. Shape selection and fitting was carried out in selected intervals (e.g., each 100 new data points) or at the end of the run. Where applicable, results were averaged over 10 runs with identical true shape parameters, but newly drawn latin hyper squares.

5.5.1.1 Active Learning

In order to measure and compare the different candidate point selection methods, we use a simple metric: the difference in parameters between the estimated boundary and the true boundary, using a subset of data points. $D_a = \{x \in X | 0.5 - \delta < \hat{y}(x) < 0.5 + \delta\}$ and estimate the parameters $\hat{\theta}$ for those points. For a single hyperplane boundary, our metric is calculated as $C = || \, |\hat{\theta}| - |\theta| \, ||_1$. The abolute values are taken to accomodate for the fact that for a hyperplane, θ and $-\theta$ describe the same hyperplane. Figure 5.34 shows C, averaged over 10 runs over the same initial for random selection, ALC, and our method.

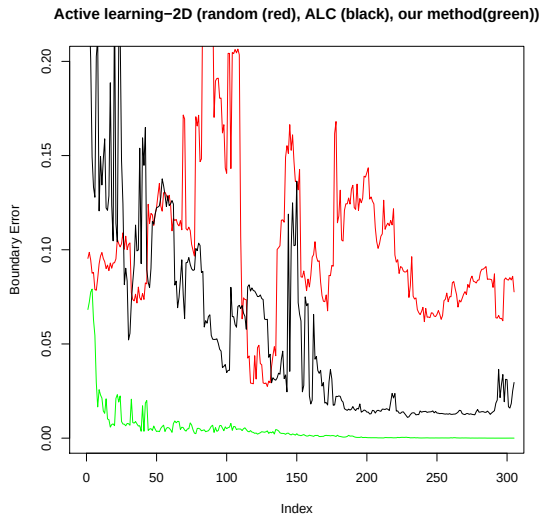

Figure 5.34: Metric $C = || \, |\hat{\theta}| - |\theta| \, ||_1$ for different selection strategies (random=red, ALC=black, our method=green) over active learning iterations (2D case)

Figure 5.35 illustrates this behavior. With the data points of D_0 shown in green and blue (according to their class), new points are shown in cyan and magenta.

The different selection methods obviously select the new data points in different regions. In the 5 dimensional case, a similar behavior can be observed as shown in Figure 5.36. Surprisingly, the results for the LHS initialization is much better than for the 3-factor initialization.

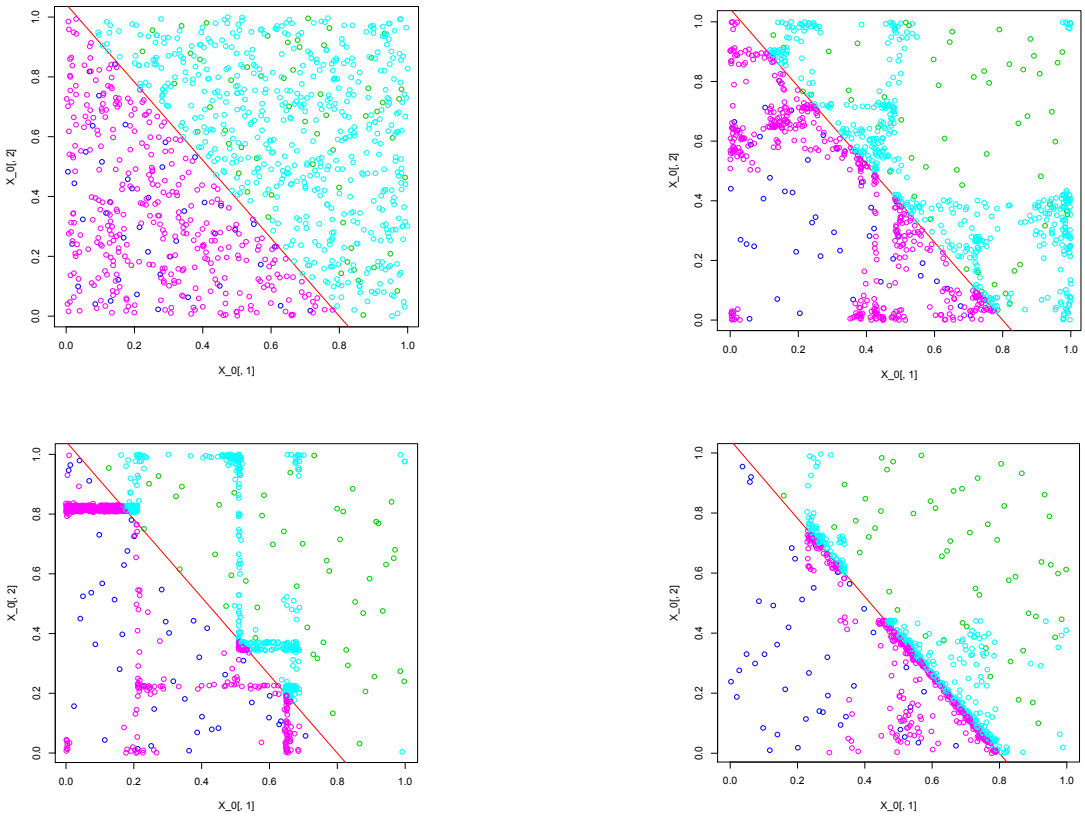

Figure 5.35: Locations of new data points during active learning: random update (top left), ALC (top right), Dynatree EI (bottom left), our boundary-oriented EI (bottom right)

In the case of more than one shape, things are more difficult. Selection methods that are greedy and stick to one boundary might fail to explore the other boundaries

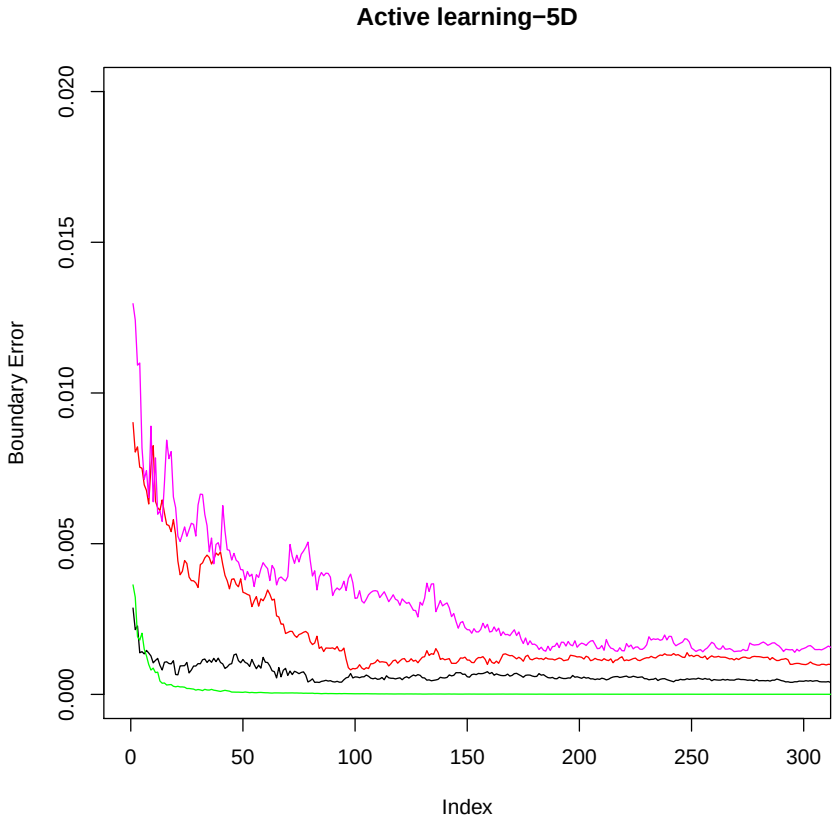

Figure 5.36: Metric $C = || \, |\hat{\theta}| - |\theta| \, ||_1$ for different selection strategies (random=red, ALC=black, our method=green) over active learning iterations (5D case, LHS initial data set)

in enough detail. As a result, the density of data points near each boundary can be substantially different, which can pose problem for the subsequent shape estimation. In general, the α parameter of our point selection governs how far the new points "stray away" from a boundary. However, values of α, which are too large produce poor overall results. We therefore propose a randomized combination of point selection rules. In the results below, we used our update method in most of the cases, in 10% of the iterations,

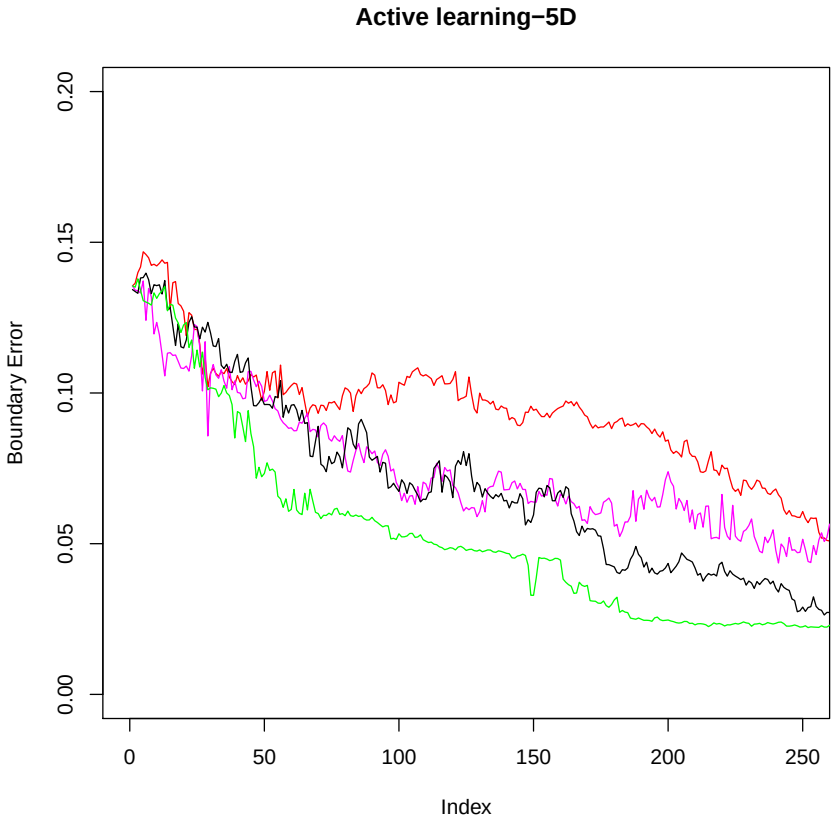

Figure 5.37: Metric $C = || \, |\hat{\theta}| - |\theta| \, ||_1$ for different selection strategies (random=red, ALC=black, our method=green) over active learning iterations (5D case, 3-factor initial data set)

however, an ALM or ALC update was carried out. The aim of this combination is to sometimes pick points, which are far away, thus increasing the chance to find and explore other boundaries as well. Figure 5.38 and Figure 5.39 show such a typical behavior for a situation in 5D with two hyperplane boundaries. The relative number of data points near each of each boundary (which are of the same size in the explored range) are shown. In the ideal case, 50% of the data points would go to each boundary. Different values

are shown for different strategies and values of α. Table 5.5 shows the behavior averaged over 10 runs. Small values of α tend to prefer one boundary over the other.

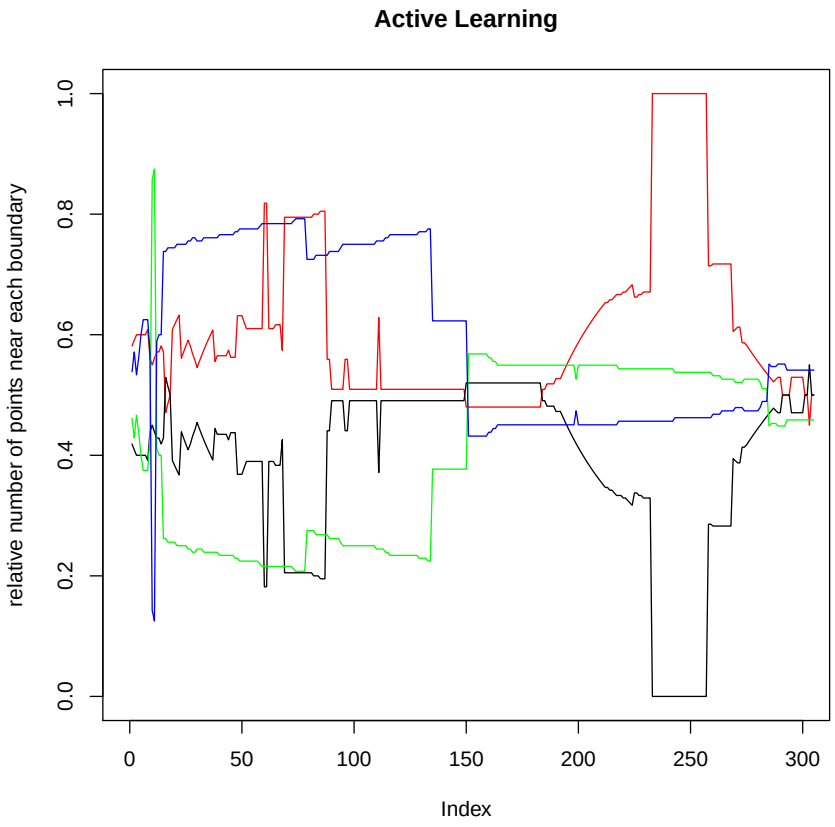

Figure 5.38: Typical convergence behavior for 2 shapes over various α (black/red: $\alpha = 1$, blue-green: $\alpha = 0.2$

5.5.1.2 Shape Selection

The selection and fitting of shapes using our algorithm stongly depends on the density of points near the boundary. The algorithm is particularly sensitive in the case of hyperplanes. We carried out experiments with an artifical data set in 2 and 5

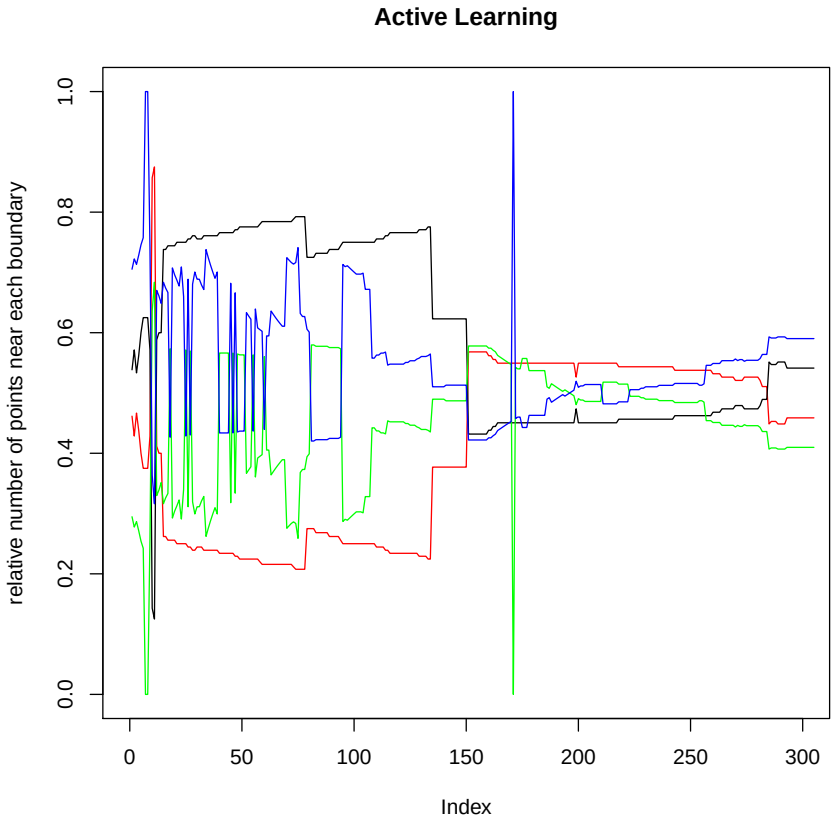

Figure 5.39: Typical convergence behavior for 2 shapes over various α (black/red: $\alpha = 0.2$, no switching; red-gree: $\alpha = 1$ and 20% of updates are ALM

dimensions, which contains two hyperspherical boundaries. Table 5.6 shows the results in the two-dimensional case. Given the true parameters of $x_0 = y_0 = 0.3, r = 0.2$ and $x_0 = y_0 = 0.7, r = 0.2$, the algorithm was started with 126 initial data points and active learning was carried out to add 700 points using $\alpha = 1$ and a 10% fallback rate to ALC. Then our algorithm for shape and parameter estimation was executed on data points in an $\epsilon = 0.3$ band, yielding 74 usable data points. With parameters of $\lambda_{SS} = 0.1$ and $\lambda_{bd} = 1.5$ the algorithm was started $n = 25$ times. Table 5.6 shows mean values

update rule			combination/ALM	
$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 1$	$\alpha = 0.2$	$\alpha = 1$
0.35(0.12)	0.28(0.08)	0.53(0.28)	0.55(0.30)	0.50(0.24)
0.65(0.41)	0.72(0.5)	0.47(0.22)	0.45(0.2)	0.50(0.24)

Table 5.5: Mean relative number of new data points near boundary. 2 boundaries (hyperplanes, 5D). Variance shown in parentheses as obtained over 5 runs (126+200 points)

and variance for the cases the actual global minimum was reached (n in the table). In the other cases, the numerical optimization algorithm returned values at the upper and lower limits for the parameters, indicating issues with optimization. If the global minimum was reached, the estimated parameters are close to the true values.

	sphere1	sphere2
x_0	0.295(1.4e-5)	0.715(8.5e-5)
y_0	0.289(3e-5)	0.72(8.6e-5)
r	0.20(6.6e-6)	0.20(5.2e-5)
n	25/25	5/25

Table 5.6: Estimated parameters for 2D spheres. Variance shown in parentheses.

Table 5.7 shows the situation in a 5D space. The two spheres are located at $\vec{c}_1 = (0.3, 0.3, 0.3, 0.3, 0.3)^T$, and $\vec{c}_1 = (0.7, 0.7, 0.7, 0.7, 0.7)^T$, respectively; their radius is in both cases $r = 0.3$. With a total of 126+1000 data points, the algorithm selects 155 points, 77 of which are closer to the second hypersphere. In this case, the algorithm is only able to reasonably estimate parameters for sphere 1. Even then, the algorithm terminates in a local minimum (reaching a hard boundary on the parameters) in 2 cases out of 10. This experiment also revealed that the number of new data points must be

substantially larger for higher dimensions. The relative size of the success and failure regions, which determines the size of the boundaries strongly influences results. For example, simply carrying over the previous shape with $r = 0.2$ into the 5-dimensional case did not work, because in 5 dimensions, less than 2% of the data points are failures inside the spheres. For such small boundaries it turned out that parameters cannot be estimated reasonably.

	sphere1
c_1	0.29(7e-3)
c_2	0.26(5e-3)
c_3	0.32(8e-3)
c_4	0.31(7e-3)
c_5	0.27(9e-3)
r	0.29(8e-4)
n	8/10

Table 5.7: Estimated parameters for 5D spheres. Variance shown in parentheses.

5.5.2 IFCS Data Set

For experiments with the IFCS adaptive aircraft controller, we used two different data sets. The full data set contains 11 input parameters, which are defined in Table 2.1 on page 11. A 3-factor initial data set with 967 records was provided. This data set 398 successful runs and 578 failures. In this data set, locations and shapes of actual boundaries are not known. In order to be able to assess our algorithm, we produced another data set of 5 dimensions. This data set has 7992 successful runs and 24,776 failed runs with $T_{fail} < 20s$. The input parameters for this data set are

$w_p, w_q, w_r, K_{lat}, \zeta$. This selection of a subspace enabled us to produce a full combinatorial exploration of the space with up to 8 values per dimension, yielding a total of 32,768 data points. Figure 5.40 shows a projection into 3D, which exhibits some boundaries. We show projections for low and high values of ζ, and low and high values of K_{lat} into the dimensions of the main gain parameters w_p, w_q, and w_r. Obviously, the boundaries (visualized using Matlab's iso-surface functions) depend on K_{lat} and one of the boundaries is non-linear.

In a different projection into the dimensions of w_p, w_q, and K_{lat}, the shape of the "outer" boundary can be seen even more clearly as shown in Figure 5.41. This spherical shape is a consequence of the IFCS design, and the shape can be given as $(\frac{w_p - x_0}{\phi_1})^2 + (\frac{w_q - \phi_2 - y_0}{\phi_3})^2 = 0.6 - K_{lat}$ Here, the stability boundary concerning the input parameters w_p and w_q, and K_{lat} is parameterized by unknown ϕ_i and x_0, y_0 are design-time constants.

Figure 5.42 shows results of finding and detecting this boundary using our approach. The blue surface corresponds, as a reference, to this boundary as visualized in Matlab based upon the combinatorial 5D IFCS data set. For our experiment, we used an initial 5D LHS data set D_0 with $|D_0| = 1000$ and ran our active learning procedure to obtain 5000 new data points. Based upon 485 points, which were selected near the boundary within an ϵ-band with $\epsilon = 0.2$, shape selection and fitting was performed. The green dots in Figure 5.42 show a part of the sphere as determined by the estimated parameters.

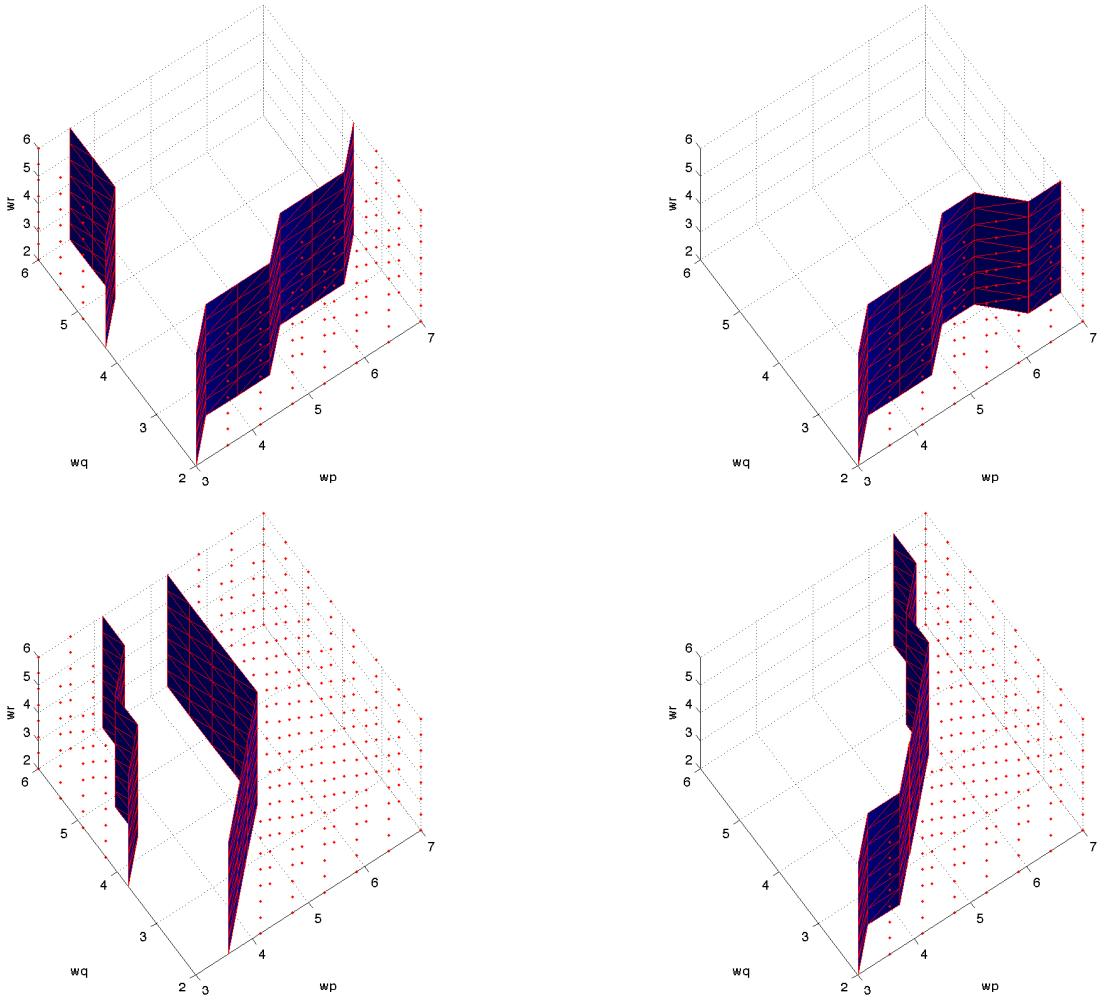

Figure 5.40: Boundaries low klat top, high bottom, low zeta left

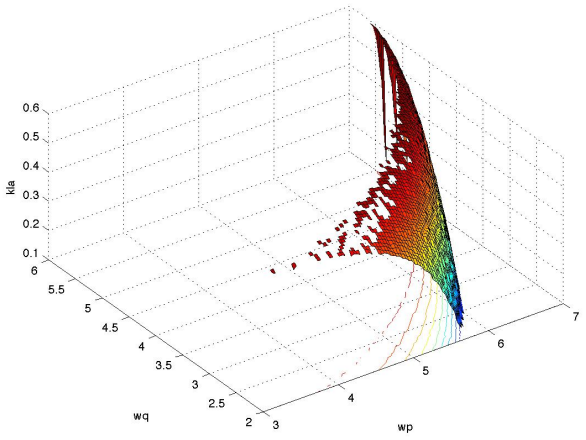

Figure 5.41: Shape of boundary in parameters w_p, w_q, K_{lat}.

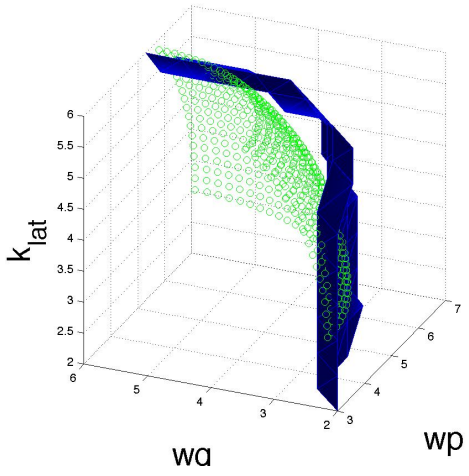

Figure 5.42: Boundary (blue) and estimated boundary (green) for IFCS data in parameters w_p, w_q, K_{lat}.

Chapter 6

Conclusions and Future Work

In this work, we developed methods for the analysis of a function of multiple real variables in which the output is itself a function of a real variable as well as categorical information. For safety-critical systems, like the NASA IFCS aircraft control system, the ability to predict the system behavior (seen as a function over many parameters and with time-series as outputs) is necessary for ensuring reliable operation and aircraft safety. Classical control theory can provide an answer for simple systems (fixed-gain, non-adaptive control systems), but modern control systems are typically of such a high complexity that they cannot be studied analytically. Hence, mechanisms to learn the behavior of such a system treated as a black box are needed for safety assurance. We can see from our analysis that learning the behavior of this complex system is possible within a certain accuracy. Once trained, prediction can be generated much more efficiently compared to system simulation. Statistical methods for prediction of the aircraft's behavior as, for example, represented by output curves and time-to-failure,

as well as the identification of unsafe regions and their boundaries are important steps in a framework for the statistical validation of complex, nonlinear aerospace systems. We have made an argument [He *et al.*, 2012c] that traditional methods for validation really only work well when the quantities of interest are dependent on only a few variables, and that the relationships between the variables and the quantities of interest should be more-or-less linear. We have argued that the National Airspace has already reached a point in which the assumptions traditional validation is based on no longer hold.

We have explored the possibility that we can use *statistical emulation* to overcome the limitations of traditional validation techniques. Statistical emulation has a key benefit in that it allows for *uncertainty quantification*—a necessity for the validation of safety-critical systems. Statistical emulation, particularly statistical emulation based on Classification Treed Gaussian Processes (CTGP), has been used to reliably predict the behavior of complex, non-linear, high-dimensional systems that are locally smooth. We employ a hierarchical, two-stage approach where data are first classified based on their inputs. We then use TGP to predict the output time-series curves using a sequence of orthogonal decompositions. We have demonstrated that statistical emulation can be used to reliably predict the temporal output curves of the simulator with high accuracy. We have also shown that, given an input, we can predict the time-to-failure for a high-dimensional adaptive flight control simulator. As a byproduct of this prediction, the behavior of the simulator becomes quantified, and can be compared against experimental data for validation. We have extended statistical emulation to explicitly handle functional outputs of varying lengths. Our framework can be extended easily to

explicitly handle time series inputs, as well.

The detection and analysis of safety regions, i.e., regions in the input space where the aircraft is controllable and exhibits a stable flight, and the characterization of operational safety boundaries are important tasks during aircraft design, validation, and certification. Complex aircraft control systems like the IFCS cannot be studied fully analytically and have non-linear boundaries. In this work, we have developed a sequential algorithm for finding and describing boundaries, using an approach based upon design of computer experiments. In many cases, shapes of boundaries are dictated by physical laws and design information, but unknown parameters and other effects makes it important to automatically find a suitable functional shape. A dictionary of suitable linear or non-linear parameterized boundary shapes, which capture underlying physical and design knowledge can be provided by the domain expert. We incorporate this knowledge into our modeling and determine the most likely shapes and its parameters. Since each iteration requires a costly run of the system simulator, we developed a candidate selection function, which is specifically tailored toward boundary detection in order to reduce the number of required simulation runs.

The work presented in this thesis can be extended into various directions. More efficient criteria for locating the boundary in a high dimensional space are needed. The use of adaptive sampling for locating and the boundary and incorporating the shape selection mechanism into the sampling can improve the ability to locate and characterize boundaries with fewer runs of the system simulator.

Finally, the analysis of model sensitivity with mixed-integer inputs is of inter-

est. Sensitivity analysis is concerned with quantifying and describing the sensitivity of a simulator output to variation in its inputs. Usually based on formulating uncertainty in the model inputs by a joint probability distribution, and then analyzing the induced uncertainty in the output. We will incorporate sensitivity analysis into our model.

In the current setting, the analysis is performed during design and validation. The benefit of our method using statistical emulation and sequential algorithms for boundary analysis is its high efficiency, which can make it possible to perform such analysis during the actual operation of the aircraft. Future work can bring statistical emulation to a state, where its applications are closer to real life scenarios, for example, unexpected conditions of the aircraft while it is in the air, e.g. sudden change of weather or aircraft being damaged while flying. These data need to be incorporated in time to update the model and safety, dangerous boundary conditions, to give effective guidance to the plane and information to the pilot to control the plane to safety regions.

Bibliography

[Banerjee and Gelfand, 2006] Sudipto Banerjee and Alan E. Gelfand. Bayesian wombling: Curvilinear gradient assessment under spatial process models. *J. Am. Statist. Ass*, pages 1487–1501, 2006.

[BEA, 2012] BEA. Final Report on the Accident on 1st June 2009 on the Airbus A330-203. Technical report, Bureau d'Enquêtes et d'Analyses pour la sécurité de l'aviation civile, 2012.

[Bengtsson and Riedy, 2012] Henrik Bengtsson and Jason Riedy. R.matlab: Read and write of MAT files together with R-to-Matlab connectivity. http://cran.r-project.org/web/packages/R.matlab/index.html, 2012.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R. Olshen, and C Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[Broderick and Gramacy, 2009a] T. Broderick and R. B. Gramacy. Classification and categorical inputs with treed Gaussian process models. *ArXiv e-prints*, April 2009. http://arxiv.org/PS_cache/arxiv/pdf/0904/0904.4891v2.pdf.

[Broderick and Gramacy, 2009b] Tamara Broderick and Robert B. Gramacy. Treed Gaussian Process Models for Classification. *Proc. 11th Conference of the International Federation of Classification Societies (IFCS-09)*, pages 101–108, March 2009. `http://www.statslab.cam.ac.uk/~bobby/gfkl-274.pdf`.

[Broderick and Gramacy, 2011] T. Broderick and R. B. Gramacy. Classification and Categorical Inputs with Treed Gaussian Process Models. *Journal of Classification*, 28(2):244–270, 2011.

[Broderick, 2004] R.L. Broderick. Statistical and adaptive approach for verification of a neural-based flight control system. In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, volume 2, pages 6.E.1 – 61–10 Vol.2, oct. 2004.

[Burken *et al.*, 2006] John J. Burken, Peggy Williams-Hayes, John T. Kaneshige, and Susan J. Stachowiak. Reconfigurable Control with Neural Network Augmentation for a Modified F-15 Aircraft. Technical Report NASA/TM-2006-213678, NASA, 2006.

[Calise and Rysdyk, 1998] A. Calise and R. Rysdyk. Nonlinear adaptive flight control using neural networks. *IEEE Control Systems Magazine*, 21(6):14–26, 1998.

[Chipman *et al.*, 1998] H. Chipman, E. George, and R. McCulloch. Bayesian cart model search (with discussion). *Journal of the American Statistical Association*, 93:935–960, 1998.

[Chipman *et al.*, 2002] H. Chipman, E. George, and R. McCulloch. Bayesian treed models. *Machine Learning*, 48(3):303–324, 2002.

[Cohen *et al.*, Sep 1996] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. The combinatorial design approach to automatic test generation. *Software, IEEE*, 13(5):83–88, Sep 1996.

[Cohn, 1996] D. A. Cohn. Neural network exploration using optimal experimental design. *Advances in Neural Information Processing Systems*, 6(9):679–686, 1996.

[Conti and O'Hagan, 2010] Stefano Conti and Anthony O'Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640 – 651, 2010. `http://www.tonyohagan.co.uk/academic/ps/multioutput.ps`.

[Department of Defense, 1997] Department of Defense. Interface standard: Flying qualities of piloted aircraft. Technical report, Department of Defense, 1997.

[Dunietz *et al.*, 1997] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing: experience report. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 205–215, 1997.

[F-22, 1992] F-22. F-22 Raptor Stealthfighter. `http://www.f-22raptor.com/index_airframe.php1992`, 1992.

[Fischler and Bolles, 1981] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[GGobi, 2008] GGobi. Ggobi data visualization system. http://www.ggobi.org, 2008.

[Giannakopoulou *et al.*, 2011] Dimitra Giannakopoulou, David H. Bushnell, Johann Schumann, Heinz Erzberger, and Karen Heere. Formal testing for separation assurance. *Annals of Mathematics and Artificial Intelligence*, 63(1):5–30, September 2011.

[Gramacy and Lee, 2008] Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103:1119–1130, 2008.

[Gramacy and Lee, 2010] R. Gramacy and H. Lee. Optimization under unknown constraints. In *Bayesian Statistics 9*, volume 9. Oxford University Press, 2010.

[Gramacy and Polson, 2011] R. Gramacy and N. Polson. Particle learning of gaussian process models for sequential design and optimization. *Journal of Computational and Graphical Statistics*, 20(1):467–478, 2011.

[Gramacy, 2005] Robert B. Gramacy. *Bayesian Treed Gaussian Process Models*. PhD thesis, University of California at Santa Cruz, December 2005. http://faculty.chicagobooth.edu/robert.gramacy/papers/gra2005-02.pdf.

[Gramacy, 2007] Robert B. Gramacy. tgp: An r package for bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. *Journal of Statistical Software*, 19(9):1–46, June 2007. http://www.jstatsoft.org/v19/i09/paper.

[Grindal *et al.*, 2005] Mats Grindal, Jeff Offutt, and Sten F. Andler. Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.

[Gundy-Burlet *et al.*, 2008] Karen Gundy-Burlet, Johann Schumann, Tim Menzies, and Tony Barrett. Parametric analysis of antares re-entry guidance algorithms using advanced test generation and data analysis. In *Proc. iSAIRAS 2008 (9th International Symposium on Artifical Intelligence, Robotics and Automation in Space)*, 2008.

[He *et al.*, 2012a] Y. He, H. Lee, and M. Davies. A new method for predicting the behavior of adaptive flight control systems. Technical report, UCSC AMS, 2012.

[He *et al.*, 2012b] Y. He, H. Lee, and M. Davies. Predicting variable-length functional outputs for emulation of a flight simulator. Technical Report UCSC-SOE-12-16, UCSC AMS, 2012.

[He *et al.*, 2012c] Y. He, H. Lee, and M. Davies. Towards validation of an adaptive flight control simulation using statistical emulation. In *Infotech@Aerospace*, 2012.

[Higdon *et al.*, 2008] D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583, 2008.

[Jacklin *et al.*, 2004] S. Jacklin, M. Lowry, J. Schumann, P. Gupta, J. Bosworth, E. Zavala, J. Kelly, K. Hayhurst, C. Belcastro, and C. Belcastro. Verification, valida-

tion, and certification challenges for adaptive flight critical control system software. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004.

[Jacklin *et al.*, 2005] S. Jacklin, J. Schumann, P. Gupta, M. Richard, K. Guenther, and F. Soares. Development of advanced verification and validation procedures and tools for the certification of learning systems in aerospace applications. In *AIAA Infotech*, 2005.

[Jensen, 2001] A. Jensen. *Ripples in Mathematics: The Discrete Wavelet Transform.* Springer Verlag, 2001.

[Jones *et al.*, 1998] D. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[Kennedy and O'Hagan, 2001] M. Kennedy and A. O'Hagan. Bayesian calibration of computer models (with discussion). *Journal of the Royal Statistical Society, Series B*, 63:425–464, 2001.

[Liu *et al.*, 2004] Y. Liu, S. Yerramalla, E. Fuller, B. Cukic, and S. Gururajan. Adaptive control software: Can we guarantee safety? In *International Computer Software and Applications Conference; Workshop on Software Cybernetics*, 2004.

[Liu, 2007] Fei Liu. *Bayesian Functional Data Analysis for Computer Model Validation.* PhD thesis, Duke University, May 2007. `http://www.stat.duke.edu/people/theses/LiuF.pdf`.

[MacKay, 1992] D. J. C. MacKay. Information–based objective functions for active data selection. *Neural Computation*, 4(4):589–603, 1992.

[Mathworks, 2012] Mathworks. Simulink. `http://www.mathworks.com/products/simulink/`, 2012.

[Menon *et al.*, 2006] P. Menon, D. Bates, and L. Postelthwaite. Computation of worst-case pilot inputs for nonlinear flight control system analysis. *Journal of Guidance, Control, and Dynamics*, 29(1):195–199, 2006.

[Menon *et al.*, 2007] P. Menon, D. Bates, and L. Postelthwaite. Nonlinear robustness analysis of flight control laws for highly augmented aircraft. *Control Engineering Practice*, 15:665–662, 2007.

[Menzies and Hu, 2003] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.

[Menzies and Sinsel, 2000] T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://menzies.us/pdf/00ase.pdf`.

[Ranjan *et al.*, 2008] Pritam Ranjan, Derek Bingham, and George Michailidis. Sequential experiment design for contour estimation from complex computer codes. *Technometrics*, 50(4):527–541, 2008.

[Rumelhart *et al.*, 1986] Rumelhart, McClelland, and the PDP Research Group. *Parallel Distributed Processing.* MIT Press, 1986.

[Rysdyk and Calise, 1998] R. Rysdyk and A. Calise. Fault tolerant flight control via adaptive neural network augmentation. *AIAA American Institute of Aeronautics and Astronautics*, AIAA-98-4483:1722–1728, 1998.

[Sacks *et al.*, 1989] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–423, 1989.

[Santner *et al.*, 2003] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments.* Springer-Verlag, New York, NY, 2003.

[Schumann and Liu, 2007] J. Schumann and Y. Liu. Tools and methods for the verification and validation of adaptive aircraft control systems. In *IEEE Aerospace Conference*. IEEE Press, 2007.

[Schumann *et al.*, 2009] J. Schumann, K. Gundy-Burlet, C. Pasareanu, T. Menzies, and T. Barrett. Software v&v support by parametric analysis of large software simulation systems. In *Proc. IEEE Aerospace*. IEEE Press, 2009.

[Smith *et al.*, 2010] T. Smith, J. Barhorst, and J. M. Urnes Sr. Design and flight test of an intelligent flight control system. In J. Schumann and Y. Liu, editors, *Appli-*

cations of Neural Networks in High Assurance Systems, Studies in Computational Intelligence, pages 57–76. Springer Verlag, 2010.

[Taddy *et al.*, 2011] Matthew A. Taddy, Robert B. Gramacy, and Nicholas G. Polson. Dynamic trees for learning and design. *Journal of the American Statistical Association*, 106(493):109–123, 2011.

[Tai and Lie, 2002] K.C. Tai and Y. Lie. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, 2002.

[Wald, 1950] A. Wald. *Statistical Decision Functions*. Wiley, New York, 1950.

[Wallace and Kuhn, 2001] D. R. Wallace and D. R. Kuhn. Failure modes in medical device software: an analysis of 15 years of recall data. *International Journal of Reliability, Quality and Safety Engineering*, 8(4), 2001.

[Wickham, 2008] H. Wickham. *Practical tools for exploring data and models*. PhD thesis, Iowa State, 2008.

[Wickham, 2012] H. Wickham. Classifly: Explore classification boundaries in high dimensions. `http://had.co.nz/model-vis/`, 2012.