Predicting Variable-Length Functional Outputs for Emulation of a Flight Simulator

Yuning He, Herbert K. H. Lee, and Misty Davies*

Abstract

Adaptive flight control systems using online-learning neural networks have great promise for improving the safety of aircraft. However, these systems are difficult or perhaps even impossible to certify using current methodology, yet it is required that the systems be demonstrated to be safe. A recent approach considers the use of statistical emulation to help understand the behavior of the system. We present a statistical framework to model and predict the output of a function of multiple real variables in which the output is itself a function of a real variable using statistical emulation. This approach has the benefit that it carries statistical uncertainties with its predictions. Through our model, we can assist NASA with development of their flight control simulator.

Key words: computer modeling; treed Gaussian process; principal components analysis; Bayesian statistics

1 Introduction

During the last decade, loss of control in flight was responsible for 23% of the 87 fatal accidents in the worldwide commercial jet fleet, and 37% of the 4774 onboard fatalities (Boeing Commercial Airplanes, 2011). Adaptive flight control systems augur an even safer airspace,

*Yuning He is graduate student, Department of Applied Mathematics and Statistics, University of California, Santa Cruz, USA (Email: yuning@ams.ucsc.edu), Herbert Lee is Professor, Department of Applied Mathematics and Statistics, University of California, Santa Cruz (Email: herbie@ams.ucsc.edu), and Misty Davies is Research Computer Scientist, National Aeronautics and Space Administration, Ames Research Center, Moffett Field, California, USA (Email: misty.d.davies@nasa.gov). This work was partially supported by NASA and its University Affiliated Research Center, as well as National Science Foundation grant DMS 0906720.

in which damage to an aircraft or unusual aerodynamic conditions need not lead to loss of control if the aircraft has sufficient control authority. This paper examines a particular automated flight controller, the Gen2 IFCS controller, being prototyped for F-15 fighter jets in the Advanced Control Technology for Integrated Vehicles (ACTIVE) flight tests (Burken et al., 2006; Smith et al., 2010). Like many adaptive control algorithms, it uses online learning neural networks (OLNNs) to approximate the aircraft and its environment (Jorgensen, 1997; Kim and Calise, 1997). However, the overall behavior of an online neural network is notoriously hard to predict (Jacklin et al., 2004), so in order to be able to understand and certify this adaptive control system, NASA engineers created a computer simulation of the whole flight and flight controller interaction. In particular, NASA engineers want to know when the controller will be successful and when it will fail, and they would also like to be able to predict the flight behavior as the process moves toward a successful stabilization or a loss of control.

The statistical problem of interest is to be able to predict the output of this simulator, and hence predict the result of the flight in response to the control algorithm. We create a statistical emulator, a computationally efficient statistical model that is used to approximate the outcome of a computationally expensive simulation. The standard approach to emulation in the literature is to use a Gaussian Process model (Sacks et al., 1989). As a nonparametric model, it provides a large amount of flexilibity while still imposing a certain degree of smoothness. Additional flexibility can be gained through extensions such as the Treed Gaussian Process (Gramacy and Lee, 2008).

Our problem here goes well beyond the standard formulation of predicting a single scalar output from multiple scalar inputs. Here we address the prediction of multiple variable-

length functional outputs from multiple scalar inputs. While a few approaches exist in the literature for predicting fixed-length functional outputs, the problem of predicting an output whose length may depend on the inputs is more difficult. We thus build a new statistical emulator for the NASA Gen2 IFCS adaptive control simulator with 11 input variables and 12 output variables, where each output variable is a function over time indicating some aircraft configuration measurement such as an altitude angle. While the specific application we explore in this paper is the Gen2 IFCS adaptive flight control system, we consider this example to be representative of many similar problems in physics-based systems.

The problem of functional data prediction has received some attention in recent years. Bayarri et al. (2007) addresses the problem of predicting functional data to emulate dynamic computer models using a wavelet basis. Conti and O'Hagan (2010) addresses the problem of predicting functional data using an approach based on stationary Gaussian Process model(GASP). What distinguishes our problem from the above problems is the the fact that our outputs are not all the same length, and these lengths are unknown until the simulator is actually run. GASP methodology does not scale easily and is computationally infeasible in problems with high-dimensional, time-dependent or functional outputs (Higdon et al., 2008).

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. This may occur when a simulator fails to run to completion for some configurations of inputs, and the time to failure depends on the input configuration. The use of a reduced basis

representation allows for a fixed length representation of output curves and computational tractability while retaining a high degree of fidelity.

When output curves can be grouped into a small set of clusters of similar curve length, shape, and frequency content, we find that fitting distinct models for different classes of output curves improves the prediction. Therefore, we add a class parameter to our statistical model. Our complete model is built on top of statistical models for individual 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 using non-stationary Treed Gaussian Process (TGP) models for both modeling the class membership as well as the mapping from inputs to outputs within a class.

We first consider predictions for multiple output variables by independently modeling each output variable. We next address the problem of predicting multiple outputs simutaneously by proposing the use of Principal Components Analysis (PCA) to transform the output variables so that correlations among the curves for the transformed variables is minimized. Then prediction can be done independently on the decorrelated variables using our single output variable model. Transforming back to the original variables gives the jointly predicted curves.

In Section 2, we discuss and show examples of the dynamic, variable-length NASA flight simulator data which our method emulates, as well as background and motivation for the NASA flight control application. In Section 3, we review the Gaussian Process and Treed Gaussian Process models which form the building blocks for our emulation model. We also discuss related models in the literature and provide some rationale for the modeling choices

we have made, and we provide the full specification of our statistical model. In Section 4, we discuss the resulting algorithm for implementing our statistical model, including a discussion of the output curve classification criteria and a comparison of different classification methods for our NASA flight simulator emulation problem, as well as some background material on the different bases considered. In Section 5, we give prediction results for our method on the NASA flight data with a comparison of results for different bases and different numbers of classes. Finally, we conclude in Section 6 with discussions.

2 The NASA Flight Simulator Data

The Generation 2 NASA Intelligent Flight Control System (IFCS) uses machine learning techniques to stabilize the aircraft under new and unexpected failure conditions (e.g., a 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 it tries to adapt the control signals in order to retain good flight characteristics in the presence of sudden damage, slow degradation, or new environmental situations.

A traditional flight control system receives stick inputs from the pilot and uses an aicraft reference model to create desired commands. The deviation between the desired commands and the aircraft's sensor signals are used by a proportional-integral (PI) controller to calculate 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 PI controller are sent to the OLNN. The OLNN compares the PI controller output with the output from the linearized plane reference model, which is the desired output response, and attempts 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.

It is obvious that one of the main goals is to keep the aircraft stable in both 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 that 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, helping to provide efficient learning of the system one step further by building a statistical emulator that describes the system in a simpler and more revealing manner, and providing similar outputs as the IFCS simulator when the same set of inputs is provided.

In practice, we do not work directly with the control software and live airplanes, we work instead with a computer simulation of the entire process (the airplane and the control system). The simulator represents a complex, non-linear mapping from the input variables to the output variables. The 11 input parameters for the simulator are: lateral stick gain,

longtitudinal stick gain, rudder gain, damping coefficient, the three proportional gains in the roll, pitch and yaw axes, controller gain in the yaw axis, and the three OLNN learning rates in roll, pitch and yaw axes. The 12 output variables are: the six errors with respect to the reference model for the roll, pitch and yaw axes (i.e., both proportional and integral errors for each axis), altitude, roll, pitch and yaw rates, angle of attack, and sideslip angle. Each input variable is a scalar, and each output variable is a function of time, with the value of that output measured at a discrete set of times. In our case, the timeseries have a length of at most 1901 time steps. For example, an input vector $\mathbf{x} \in \mathbb{R}^{11}$ results in 12 output vectors $\mathbf{y}^{(k)} = (y_1^{(k)}, y_2^{(k)}, \ldots, y_T^{(k)}) \in \mathbb{R}^T$, $T \leq 1901$.

A given set of experimental conditions (altitude, pilot input, speed) was provided and kept constant in our experiments. Sometimes the software is unable to adapt the control successfully to maintain aircraft stability, and the simulator terminates prematurely, having recorded fewer than the maximum number of measurements made during a successfully controlled flight. The software failure 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 failure time T (which is the last recorded sample for the output variables) varies with the input configuration $\mathbf{x}$. For many inputs $\mathbf{x}$, the system successfully controls the aircraft and the maximum number of samples $T = 1901$ for each output variable is recorded. Part of the control system seeks to minimize the error between the current output variable configurations and the expected configurations under a reference model. This often results in oscillatory behavior in the output curves as the corrections are being made.

Figure 1 shows the outputs obtained with two different sets of inputs, $\mathbf{x}1$ and $\mathbf{x}2$, which

correspond to examples of successful and failure runs respectively. For input configuration $\mathbf{x}1$ (top), all outputs stabilize after the 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 $\mathbf{x}2$ (bottom), signal excitements do not decrease over time. Rather, a high-frequency oscillation (caused by a bad adaptation of the network) can be detected, leading to instability at around $T = 380$ time steps. When there is a failure, all 12 outputs will always fail at the same time.

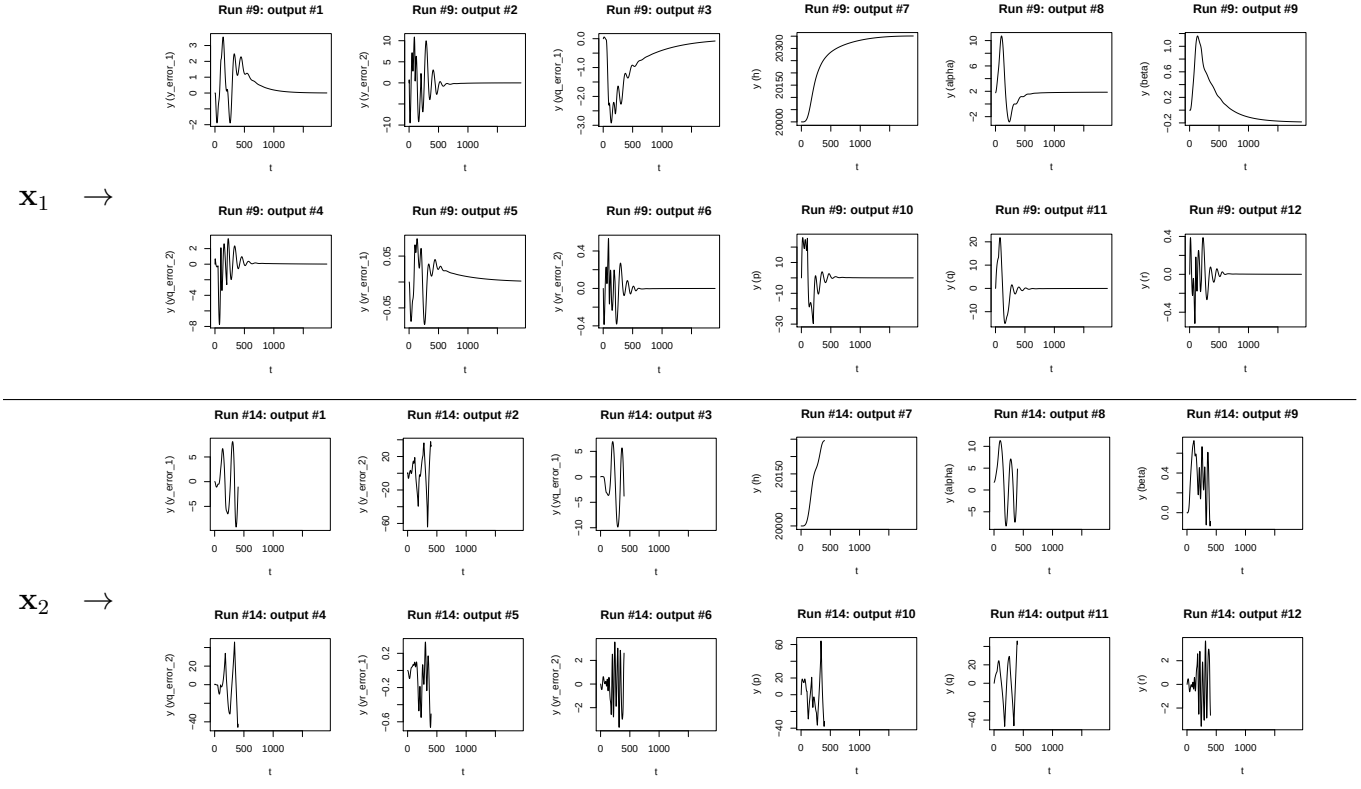

Figure 1: Typical output time series for successful (top) and failure (bottom) runs.

Some of the challenges in predicting the flight outputs include very different output

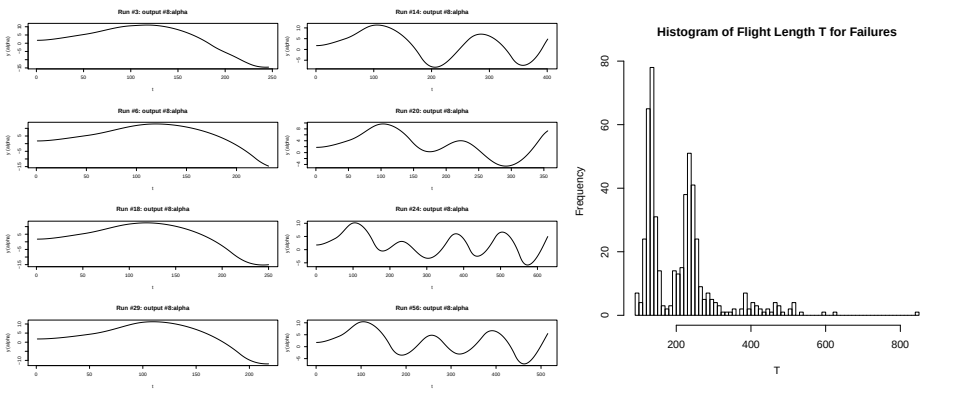

Figure 2: Curve classes

lengths with a wide variety of frequencies. The basic shapes of output curves can vary quite a lot. We observe that the output curves can be grouped into clusters of similar curve length, shape, and frequency content. General appearance and frequency content of success and failure curves are typically quite different (for example, the two runs in Figure 1). Some examples of the failure curves for a single output variable are shown in Figure 2 below. When the time to failure is about 250 time steps, the curves for different runs look similar to each other as shown in the first column of Figure 2. When time to failure is between 350 to 500 time steps, the curves look similar to those shown in the second column of Figure 2.

Therefore, 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. A histogram for time to failure, presented in Figure 2 (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 in the right tail. Finally class four (not shown) is comprised of the sucessful runs. This histogram is the basis for setting our thresholds to $(180, 280, 1900)$ time

steps as discussed in Section 5, Table 4.1. Thus our approach will be to use a two-stage model where we first classify a set of inputs into one of the four classes, and then perform prediction for that class.

3 Statistical Emulation

The traditional emulation problem starts by considering the modeling of 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):

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

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 remaining hyperparameters are given specified values.

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 specified hyperparameters. 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:

$$p(C(\mathbf{x}) = m) = \frac{\exp(-Z_m(\mathbf{x}))}{\sum_{m'=1}^{M} \exp(-Z_{m'}(\mathbf{x}))} \tag{2}$$

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. This 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 $\mathbf{x}$.

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. 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 4.1. We found that a CTGP model for $C(\mathbf{x})$ gave the best classifier accuracy.

4 Implementation

We have so far kept the modeling description somewhat generic. In this section 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. Finally, we fit additional TGP models for the coefficients of a basis representation of $\mathbf{y}(\mathbf{x}) \doteq \sum_{i=1}^{D} c_i \mathbf{b}_i$.

4.1 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 2 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 4.1 shows the total number of simulation runs in each of the four classes.

Our initial experiments focused on two-class classification into *failure* and *success*. Using CTGP resulted in an 11.8% error rate. In comparison, a nearest neighbor classifer had a 43.9% error rate, and a support vector machine classifier had a 19.4% error rate. Given this

Class	Length	# Runs
failure1	$0 < T \leq 180$	257
failure2	$180 < T \leq 280$	241
failure3	$280 < T \leq 1900$	71
success	$T = 1901$	398
Total	967 runs	967 runs

Table 1: Class membership of the available data.

success with CTGP, we applied it to the four-class problem. This resulted in a 27% error rate, which is somewhat high, but as we will see in Section 5.2, the errors that are made often still allow good curve prediction, as the curves that are difficult to classify tend to have shapes that are somewhat similar to curves in more than one class.

4.2 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 accomodate 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 accomodate 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 $c^{\text{new}}_{\text{pred}}$ for a new input $\mathbf{x}^{\text{new}}$, the predicted output curve $\mathbf{y}^{\text{new}}_{\text{pred}} = Bc^{\text{new}}_{\text{pred}}$ has length $T_{\max}$. In our testing phase, we know the true length T for the test case and we truncate $\mathbf{y}^{\text{new}}_{\text{pred}}$ 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.

4.3 Bases

We considered several different bases with which to represent output curves, of which we present three 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 paper.

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 out-

put 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. The number for each basis curve in the legend refers to the additional fraction of

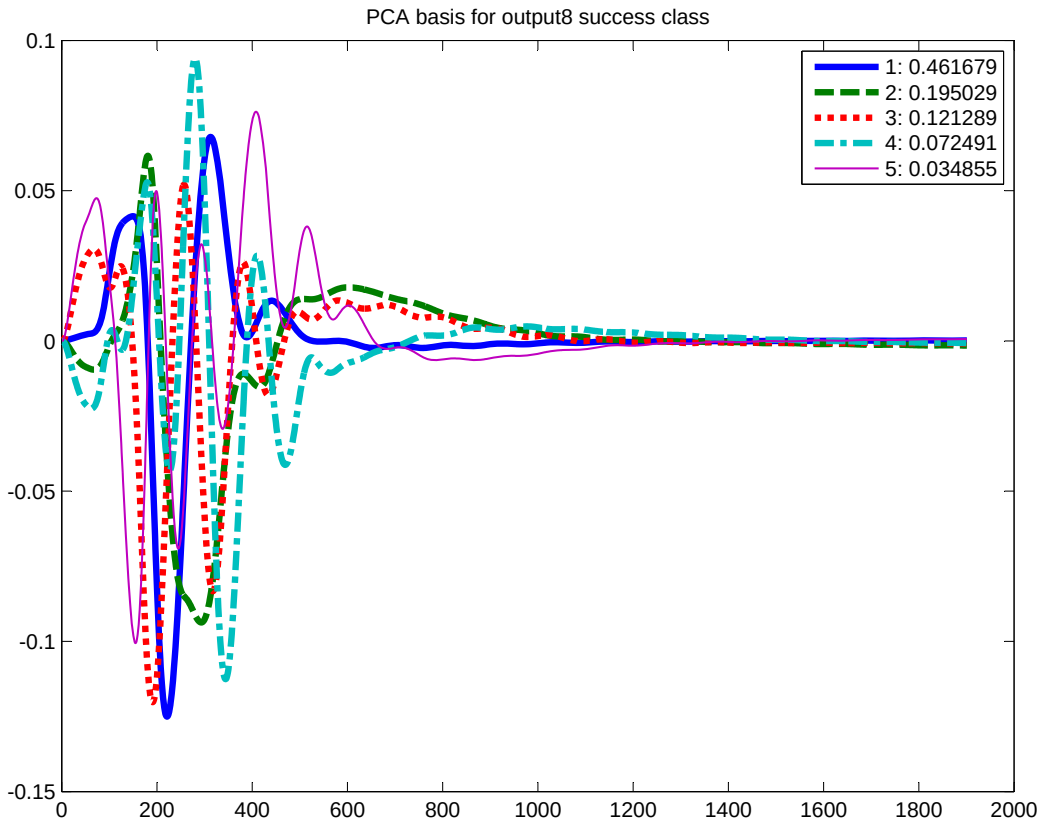

Figure 3: PCA basis (first five basis vectors) for output variable 8, success class; the legend shows the fraction of variance explained by each of the bases.

the variance in the data that is explained by using that basis curve.

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:

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

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.

The Daubechies wavelets (Jensen and la Cour-Harbo, 2003) 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. For example, D2 (the Haar wavelet) has one vanishing moment, D4 has two, etc. A vanishing moment limits the wavelet's ability to represent polynomial behavior or information in a signal. Daubechies wavelets are used in solving a broad range of problems, e.g., self-similarity properties of a signal or fractal problems, signal discontinuities.

4.4 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.

5 Results

In this section, we report results of our method for output curve prediction using perfect classification in Section 5.1 and then using the good, but not perfect CTGP classification results. Given a test input $\mathbf{x}$, we first assume knowledge of its true length $T(\mathbf{x})$ and we use a model trained for similar lengths in order predict the output curve $\mathbf{y}(\mathbf{x})$. We then remove this assumption and predict using our full multi-stage model. Errors in classification can lead to larger prediction errors because an incorrect class may be used for prediction. We will compare the errors with perfect and CTGP classification to see how much classification errors cost us in terms of prediction accuracy.

5.1 Output Curve Prediction Results

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 standarize 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

$$e_{v,c} = \frac{\sum_{i \in S_{v,c}} \frac{||\mathbf{y}_i^{\text{pred}} - \mathbf{y}_i||_1}{\sigma_i}}{\sum_{i \in S_{v,c}} T(\mathbf{x}_i)}. \tag{3}$$

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. The sums of 3 are over the total number of output curve values predicted, thus making the reported 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 4.2), 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 2. 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 paper.

Table 3 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 4 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 5.

5.2 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. Examples of predicted curves for output variable 8 are shown in Figure 6.

	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 2: Output curve predicition 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.

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

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 the model for the incorrectly predicted class $C^{\text{pred}}(\mathbf{x})$ and it is plotted for the maximum length $T_{\max}^{C^{\text{pred}}(\mathbf{x})}$ 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

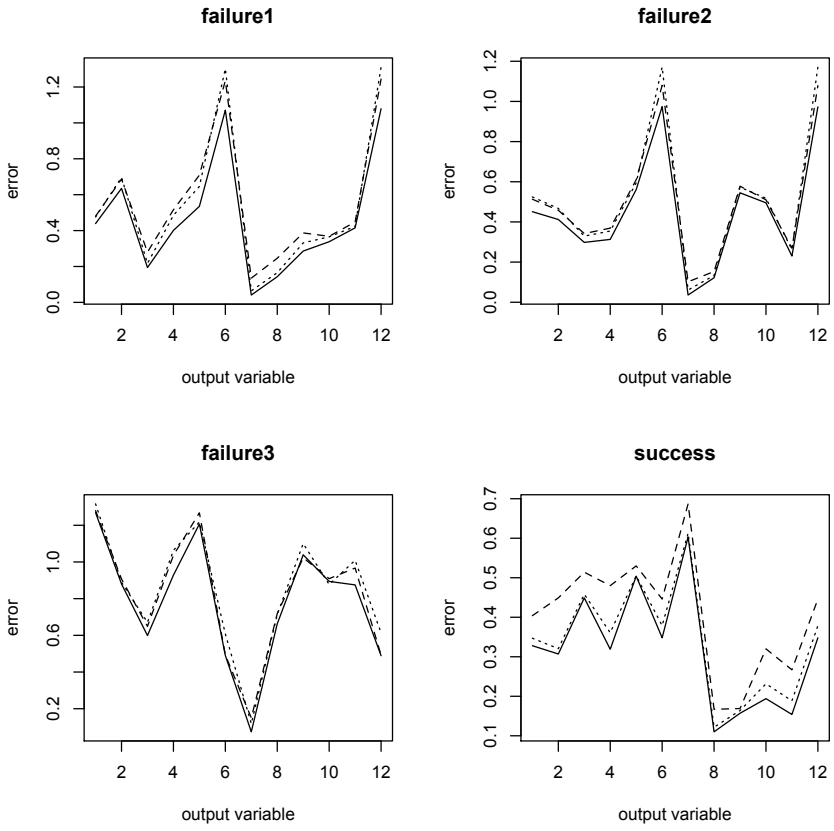

Figure 4: Output curve predicition errors for different bases, output variables, and classes in the 4 class strategy. The basis dimension is D = 25 for all bases (PCA: solid line, wavelet: dotted line, Fourier: dashed line)

black ground truth curve because the true length $T(\mathbf{x})$ must be less than 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 6, 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 second example in the right column of Figure 6, the correct class is *failure*2 with $T_{\max}^{failure1} = 580$, but the input was incorrectly

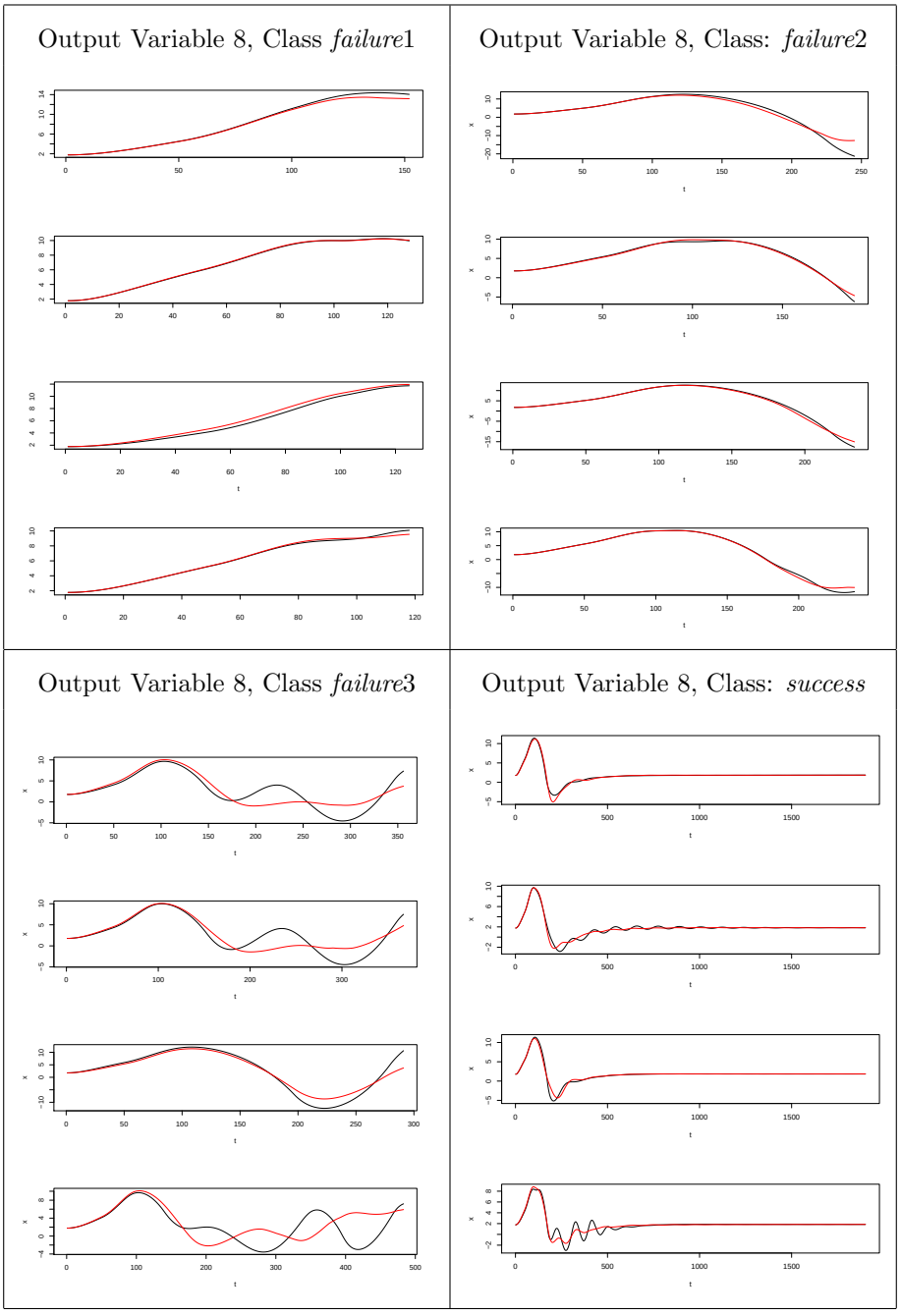

Figure 5: $D = 25$ PCA-based predictions: Output variable 8: (upper left) class *failure*1, (upper right) class *failure*2, (lower left) class *failure*3, (lower right) class *success*.

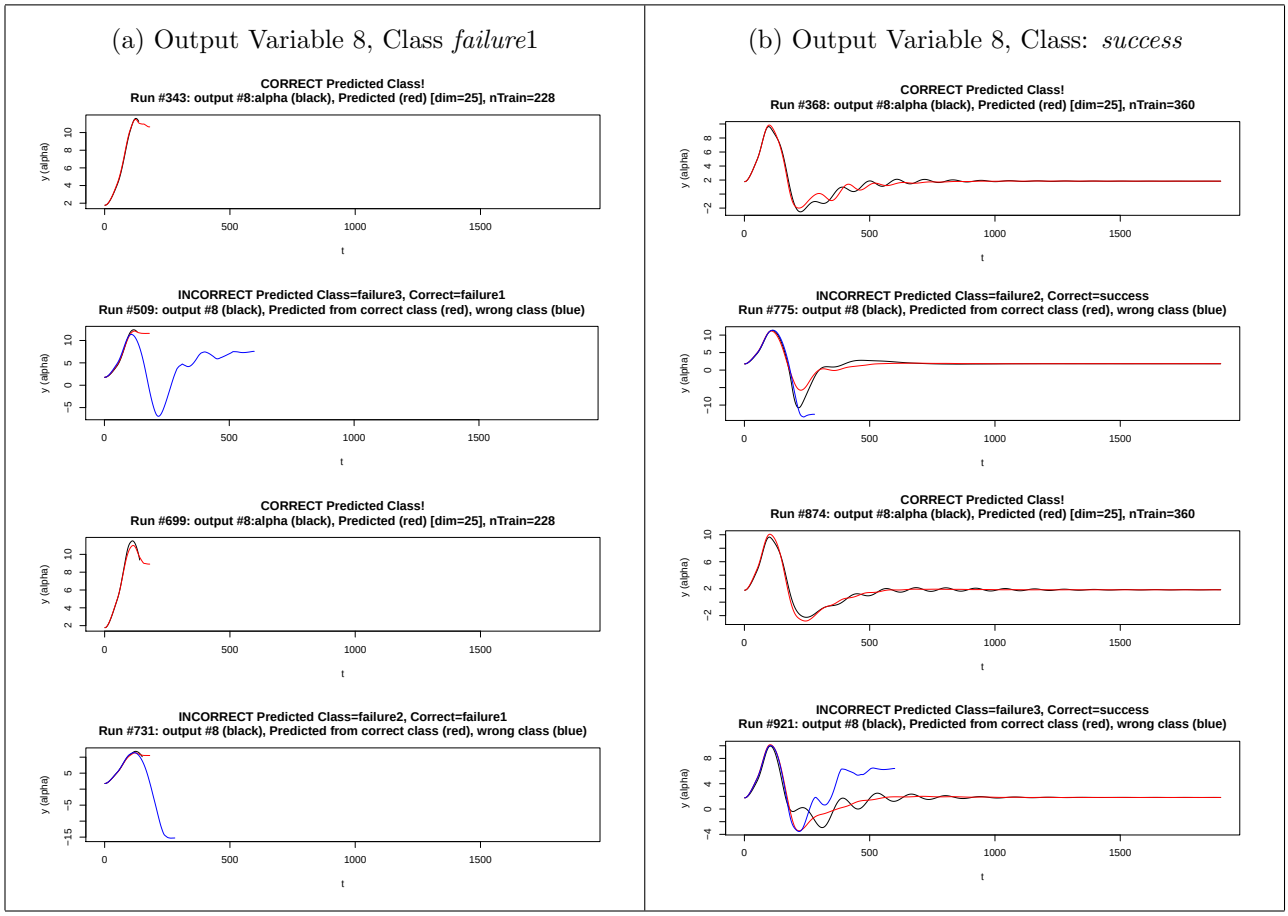

Figure 6: Predicted Output variable 8 curves using CTGP classification. $D = 25$ PCA-based predictions: (left) class *failure*1 and (right) class *success*.

classified as *success* with $T_{\max}^{success} = 1901$ so that the predicted curve is much longer. In the last example in the lower right of Figure 6, 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 6, 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. There are some tests for which the predictions are very similar using the correct and incorrect classes. Finally, there are even a few tests in which the predicted output curve is slightly better using the incorrect class.

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

Finally, we turn to the full multivariate output problem from Section 4.4. Let $y(i, v, t)$ 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 7. While our fits do not match exactly, the get quite close in shape and structure, and provide a sufficient fit for understanding the behavior of the controller.

6 Conclusions

In this paper, we discussed a statistical method for emulating an advanced and highly non-linear system, and applied it to a real problem in aircraft control. Instability and loss of control can occur when controller parameters are badly selected. Due to the large number of parameters in the controller, it is extremely hard to see which parameter settings yield a stable control system for a given pilot input, and which ones lead to instability after some

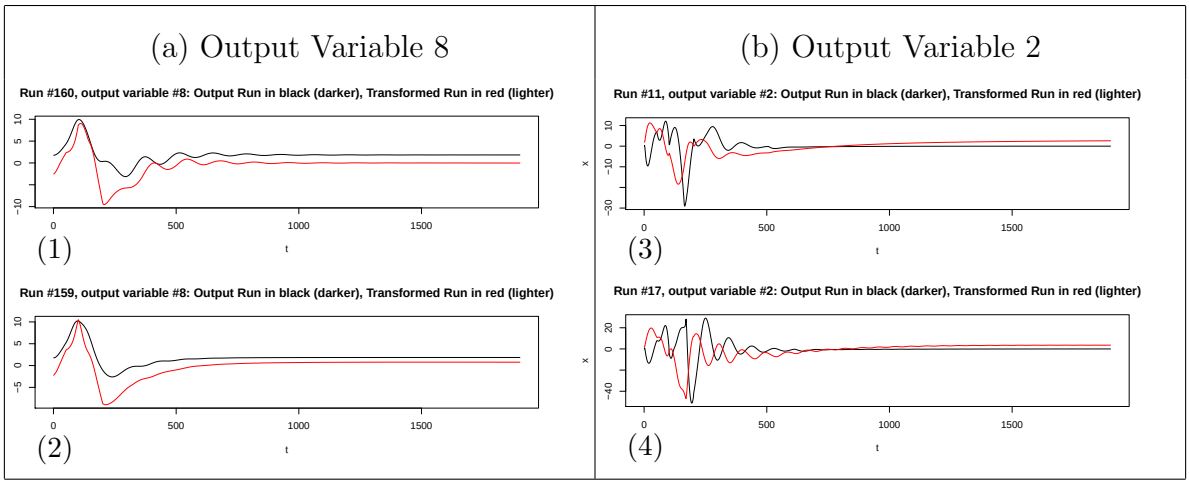

Figure 7: Predicted Output Curves for correlated multiple output curves

time. Statistical emulation can be a highly useful tool for understanding of the simulator.

We employ a two-stage approach where cases are classified based on their inputs. We then use TGP to predict the output time-series curves using a sequence of orthogonal decompositions. Future research will include the extension of the methodology presented in this paper to other, more complex prediction problems such as incorporating pilot inputs.

For safety-critical systems, like the NASA IFCS aircraft control system, the ability to predict the system behavior 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. Efficient in-flight prediction of the aircraft dynamic behavior can help answer questions such

as: will the system stay within given boundaries; will the system experience oscillations and of which type (damped, undamped); will the system remain stable. Research on advanced fault detection and vehicle health management will benefit from our approach.

Acknowledgments

This research was conducted at NASA Ames Research Center. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

Bayarri, M.J., Berger, J.O., Cafeo, J., Garcia-Donato, G., Liu, F., Palomo, Parthasarathy, R.J., Paulo, R., Sacks, and Walsh, D. (2007). "Computer model validation with functional output." *Annals of Statistics*, 35, 5, 1874–1906.

Boeing Commercial Airplanes (2011). "Statistical Summary of Commercial Jet Airplane Accidents: Worldwide Operations 1959-2010." `http://www.boeing.com/news/techissues/pdf/statsum.pdf`.

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

Burken, J. J., Williams-Hayes, P., Kaneshige, J. T., and Stachowiak, S. J. (2006). "Reconfigurable Control with Neural Network Augmentation for a Modified F-15 Aircraft." Tech. Rep. NASA/TM-2006-213678, NASA.

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

Gramacy, R. B. and Lee, H. K. H. (2008). "Bayesian treed Gaussian process models with an application to computer modeling." *Journal of the American Statistical Association*, 103, 1119–1130.

Higdon, D., Gattiker, J., Williams, B., and Rightley, M. (2008). "Computer Model Calibration Using High-Dimensional Output." *Journal of the American Statistical Association*, 103, 482, 570–583.

Jacklin, S., Lowry, M., Schumann, J., Gupta, P., Bosworth, J., Zavala, E., Kelly, J., Hayhurst, K., Belcastro, C., and Belcastro, C. (2004). "Verification, Validation, and Certification Challenges for Adaptive Flight Critical Control System Software." In *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Jensen, A. and la Cour-Harbo, A. (2001). *Ripples in Mathematics: The Discrete Wavelet Transform*. Berlin: Springer-Verlag.

Jorgensen, C. C. (1997). "Direct Adaptive Aircraft Control Using Dynamic Cell Structure Neural Networks." Tech. Rep. 112198, NASA.

Kennedy, M. and O'Hagan, A. (2001). "Bayesian Calibration of Computer Models (with discussion)." *Journal of the Royal Statistical Society, Series B*, 63, 425–464.

Kim, B. and Calise, A. (1997). "Nonlinear Flight Control Using Neural Networks." *Journal of Guidance, Control, and Dynamics*, 20, 1, 26–33.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). "Design and Analysis of Computer Experiments." *Statistical Science*, 4, 409–423.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. New York, NY: Springer-Verlag.

Smith, T., Barhorst, J., and Urnes Sr., J. M. (2010). "Design and Flight test of an Intelligent Flight Control System." In Schumann, J. and Liu, Y., eds. *Applications of Neural Networks in High Assurance Systems*, 57–76. Studies in Computational Intelligence. Berlin: Springer-Verlag.