UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EMULATION AND UNCERTAINTY QUANTIFICATION FOR MODELS WITH FUNCTIONAL RESPONSE USING BAYESIAN ADAPTIVE SPLINES

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

DOCTOR OF PHILOSOPHY

in

STATISTICS AND APPLIED MATHEMATICS

by

Devin Francom

September 2017

The Dissertation of Devin Francom
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

Emulation and Uncertainty Quantification for Models with Functional Response Using Bayesian Adaptive Splines

by

Devin Francom

When a computer code is used to simulate a complex system, a fundamental task is to assess the uncertainty of the simulator. In the case of computationally expensive simulators, this is often accomplished via a surrogate statistical model, a statistical output emulator. An effective emulator is one that provides good approximations to the computer code output for wide ranges of input values. In addition, an emulator should be able to handle large dimensional simulation output for a relevant number of inputs; it should flexibly capture heterogeneities in the variability of the response surface; it should be fast to evaluate for arbitrary combinations of input parameters; and it should provide an accurate quantification of the emulation uncertainty.

In this work, we develop Bayesian adaptive spline methods for emulation of computer models that output functions. We introduce modifications to traditional Bayesian adaptive spline approaches that allow for fitting large amounts of data and allow for more efficient Markov chain Monte Carlo sampling. We develop a functional approach to sensitivity analysis that can be performed using this emulator. We present a sensitivity analysis of a computer model of the deformation of a protective plate used in pressure driven experiments. This example serves as an illustration of the ability of Bayesian adaptive spline emulators to fulfill all the necessities of computability, flexibility and reliable calculation on relevant measures of sensitivity.

We extend the methods to emulation of an atmospheric dispersion simulator that outputs a plume in space and time based on inputs detailing the characteristics of the release, some of which are categorical. We achieve accurate emulation using Bayesian adaptive splines to model weights on empirical orthogonal functions. We extend the adaptive spline methodology to allow for categorical inputs. We use this emulator as well as appropriately identifiable simulator discrepancy and observational error models to calibrate the simulator using a dataset from an experimental release of particles from the Diablo Canyon Nuclear Power Plant in Central California. Since the release was controlled, these characteristics are known, allowing us to compare our findings to the truth.

We further extend the methods to emulate a computer model that outputs misaligned functional data. We do this by modeling the aligned, or warped, data as well as the warping functions, using separate Bayesian adaptive spline models. We explore inference methods that treat these models jointly and separately, and establish methods to ensure that the warping functions are non-decreasing. These methods are applied to a high-energy-density physics model that outputs a curve representing energy as a function of time.

for Lyndee and Grace

Acknowledgments

I owe much of my success in this work to the substantial support of Bruno Sansó, who helped me develop the ideas and navigate research, graduate school, and career decisions. I consider it an honor to have worked with him. The graduate students and faculty of the Applied Mathematics and Statistics Department have also been extremely supportive, especially Herbie Lee.

I am thankful for the support of Lawrence Livermore National Laboratory (LLNL) through the Livermore Graduate Scholar Program as well as the Data Science Summer Institute for multiple years of support and exposure to the compelling challenges that the Department of Energy works to confront. My mentors there, Vera Bulaevskaya and Ana Kupresanin, provided invaluable support and connected me to the applications that led to the practical development of this work. Many LLNL scientists, including Ryan Vignes, Gardar Johannesson, Donald Lucas, Matthew Simpson, Ronald Baskett, Philip Cameron-Smith, Brian Spears, Luc Peterson, and John Field also provided important support. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was partially funded by Laboratory Directed Research and Development at LLNL under project tracking code PLS-14ERD006. The work is released under UCRL number LLNL-TH-734468.

To the many great things my Mother has done, an addendum might be that she taught her children math. Through the course of this work, my wife, Lyndee, my greatest friend and advocate, provided support and unabated encouragement more than anyone else. Regardless of how the research or commute or anything else was going, my beautiful daughter, Grace, could brighten my day from the moment she joined our family. To these people, and so many others, I am profoundly grateful.

Chapter 1

Introduction

1.1 Motivation and Background

Computer simulation plays an indispensable role in modern scientific research and discovery. Hypotheses can be tested in computer simulated experiments that are impractical or impossible to test in true experiments. These simulators, or computer models, incorporate current scientific understanding of the phenomena of interest into a mathematical model that is solved using a computer, sometimes at great computational cost. Hence, supercomputers have become essential tools for furthering understanding of complex processes in astrophysics, materials science, engineering, fluid mechanics, nuclear physics, climate science, ecology, evolutionary biology, economics, epidemiology, medicine, sociology, and many other fields.

Computer simulations are inherently theoretical. They are based on models that, for simplicity or because of incomplete scientific understanding, do not perfectly represent reality. Hence, analyses of computer simulations that fail to account for uncertainty in the simulator are likely to reach incomplete or incorrect conclusions. This uncertainty could be due to general inadequacy of the underlying mathematical model used to describe the process, lack of understanding of the inputs to the simulator, or numerical and computational instabilities. Quantification of these uncertainties (generally called uncertainty quantification or UQ in the computer simulation community) is an essential step to ensure the reliability of a simulator.

Two uncertainty quantification problems of great interest to the those analyzing computer simulations are (1) analysis of the sensitivity of the simulator output to changes in its inputs and (2) using observations of the system of interest to determine appropriate settings (or inputs) of the simulator and possible simulator bias. The first task, called sensitivity analysis, can provide valuable information concerning which inputs have the largest effect on the output, and hence deserve further study. The second task, called calibration, is essential if the results of a simulated experiment are expected to provide useful information about reality.

Often, uncertainty quantification tasks like sensitivity analysis and calibration cannot be done analytically for complex simulators. In these cases, a Monte Carlo approach to UQ typically requires a large number of computer model evaluations using different input combinations, essentially treating the simulator as a black box. When a computer model is expensive to evaluate and a large number of evaluations is impossible, a common approach to UQ is to evaluate the computer model at a reasonable number of input combinations and build a response surface to predict the computer model output at arbitrary input settings. Building suitable response surfaces, called emulators or surrogate models, is a topic of ongoing research in many fields. An effective emulator is one that provides good approximations to the computer code output for wide ranges of input values. In addition, an emulator should be able to handle large dimensional simulation output for a relevant number of inputs; it should flexibly capture heterogeneities in the variability of the response surface; it should be fast to evaluate for arbitrary combinations of input parameters, and it should provide an accurate quantification of the emulation uncertainty. Using a statistical model for emulation is a natural choice. Because linear models are likely to be too inflexible, research has focused on semi- and non-parametric regression models, with most attention given to the Gaussian process (GP). However, traditional GP models lack the scalability and the flexibility to handle the larger and more complex datasets produced in modern computer experiments. Improvements in these areas is a topic of broad and current interest.

Emulation is further complicated in cases where a realization of the computer model results in a function rather than a scalar. The computer model output could be a curve, a surface, a time series, a spatio-temporal field or some other type of function that results from a combination of inputs. Computer models with functional response have become increasingly common as computational capacity has increased. In some cases, realizations of a computer model using different inputs may result in functional output on different grids, such as time series on different time scales. Incorporating functional data analysis methodology into emulation is an active research area, though minimal attention has been given to including registration of functional data in the emulation framework.

The purpose of this dissertation is to describe how Bayesian adaptive spline models can be used for emulation and in turn sensitivity analysis and calibration, especially for computer models with functional response.

1.2 Adaptive Splines

Traditional spline models represent a curve $f(x)$ using a linear combination of truncated polynomial basis functions, where each basis function takes the form $[x-t]_+^\alpha$ or $[t-x]_+^\alpha$ and $[x]_+ = \max\{0, x\}$. The truncation point t in each polynomial basis function is called a knot. Choosing the number of knots to use and where to put them can be arbitrary. Using many knots allows for more complex shapes, but also increases the number of parameters to be estimated, since each basis function weight in the linear combination is unknown. Representing a surface, rather than a curve, can be done by using tensor products of truncated polynomial basis functions. For instance, for $f(x_1, x_2)$ we may have basis functions of the form $[x_1 - t_1]_+^\alpha [x_2 - t_2]_+^\alpha$. This requires that knots (t_1, t_2) be placed in multiple dimensions. The curse of dimensionality leads to an exponential increase in the number of knots needed to yield complex shaped surfaces, quickly depleting degrees of freedom.

Adaptive spline models choose the number and placement of knots based on the complexity of the surface being modeled. This means that more knots will be used in parts of the input space where they are needed, and fewer where they are not needed. While similar in the spirit of parsimony to recursive partitioning, or tree based models, the end result when using spline basis functions is continuous.

Inference for adaptive spline models is complicated by the transdimensionality of the model space. Because the number of basis functions is unknown and there can be a massive number of possible basis functions to choose from, methods of inference need to be able to explore the model space in an efficient way. Further, the flexibility of the model specification can result in overfitting unless precautions are taken to limit model complexity. To use adaptive spline models for emulation, estimating their uncertainty is important so that it can be appropriately reflected in the UQ tasks that require the emulator.

1.3 Research Objectives

The primary goals of this research are to develop the capability of Bayesian adaptive spline models for emulation, sensitivity analysis, and calibration for computer models with functional response. Research contributions in these areas are motivated by three computer models. While each simulator has functional response, they each have unique challenges associated with performing UQ tasks.

The first research objective is to develop Bayesian adaptive spline models that can produce a functional response. Special attention is given to advancing inference methods and model specification to efficiently explore the adaptive spline model space and to avoid overfitting. The second objective is to use functional Bayesian adaptive spline models to do sensitivity analysis. This requires the development of functional sensitivity analysis methods. These two objectives are achieved in the work presented in Chapter 2. This work focuses on a materials science computer model that outputs curves representing the deformation of a protective plate during a high pressure experiment. The model has seven inputs determining the configuration of the plate.

In the process of achieving the objectives above, the creation of general purpose software to fit and analyze Bayesian adaptive spline models became a secondary research objective. A description of this software is given in Chapter 3.

The third research objective is to calibrate a computer model for the movement of a plume of particles in space and time in an area of complex terrain. The particular application focuses on the region near the Diablo Canyon Nuclear Power Plant in Central California. In this case, the emulator needs to be able to handle vast amounts of data and must be able to incorporate both categorical and continuous inputs. Further, the bias of the computer model needs to be quantified and incorporated into the analysis in a way that does not confound the calibration. The data obtained for calibration of this model are from an experimental release of benign particles into the atmosphere performed at the nuclear power plant in 1986. The experimental release allows us to determine the accuracy of our methods, since the conditions of the release are known. This work is presented in Chapter 4.

The fourth research objective is to build an emulator for a computer model that outputs misaligned functional data. That is, the emulator models both the phase and amplitude variation in the functional response based on the inputs to the simulator. This is presented in Chapter 5 and is demonstrated with the emulation of a high-energy-density physics simulator that outputs time series on different time scales. This emulation also requires the assimilation of large amounts of data.

Finally, the research is summarized and possible extensions are described in Chapter 6.

Chapter 2

Sensitivity Analysis and Emulation for Functional Data using Bayesian Adaptive Splines

Sensitivity analysis, as defined in Saltelli et al. 2004, is the study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input. Determining these relationships has become a fundamental step in the use of complex models because of the possible ways a modeler or model user can utilize such information. Uses include finding which inputs require the most attention (because varying them causes the output to vary substantially), finding inputs to which the model is robust and conveying the reasons that a decision based on the model may not be totally certain. More details about the practical relevance of sensitivity analysis are found in Saltelli et al. (2008) Section 1.2.14 and Pannell (1997). For the purposes of this chapter, we are most interested in performing sensitivity analyses of expensive-to-run computer models that are used to simulate complex processes. Further, we would like to do this while treating the computer model as a black box, also known as non-intrusive sensitivity analysis.

While many methods for sensitivity analysis exist, they are not all of equal value. Most methods use a set of inputs at which the model needs to be evaluated, called the experimental design. Saltelli & Annoni (2010) describe the downfalls of the "one-factor-at-a-time" (OAT or OFAT) experimental design, which determines sensitivity by changing only one input at a time from some nominal input values. The OAT design is often paired with methods of sensitivity analysis deemed "local", because they only take into account variation in the model output in some small neighborhood of the inputs, usually by taking or approximating a derivative (Saltelli et al., 2000). By contrast, "global" sensitivity analysis methods (Sobol', 1990; Saltelli et al., 2008) take into account the entire space of uncertain inputs by eliciting probability distributions over the inputs. Global methods are thus able to discover when changes in the model output are due to simultaneous changes in multiple inputs (interactions), as well as the implications of extreme, but possible, input settings. In this chapter, we limit our attention to global sensitivity analysis.

When a model is expensive to evaluate, as is the case for many computer models, being able to perform a sensitivity analysis using a limited number of model evaluations is essential. Hence, much research has been done to determine which experimental designs yield the best sensitivity analyses with the least number of evaluations (Sobol', 2001; Sacks et al., 1989). Another approach is to use a set of model evaluations to build a fast, statistical alternative to the model. Sensitivity analysis and other analyses of uncertainty can then be performed on this surrogate model, also called an emulator or metamodel. The Gaussian process has been a popular surrogate model choice (Welch et al., 1992; Kennedy & O'Hagan, 2001) because of its flexibility and simplicity, though it is sometimes avoided because of its lack of scalability to large numbers of model evaluations, large numbers of input variables, and high dimensional output. In general, sensitivity analysis for the surrogate model will require the surrogate to be evaluated for very many different combinations of the input parameters. Oftentimes, the result is prone to Monte Carlo error, depending on the number of model evaluations. This undermines the value of surrogate models that are not extremely fast to evaluate for large numbers of inputs.

In this chapter, we detail the benefits of using Bayesian multivariate adaptive regression splines (BMARS) as an emulator for the purposes of sensitivity analysis. MARS and BMARS have been recently introduced to the literature on computer model analysis (Storlie et al., 2009; Chakraborty et al., 2013; Stripling et al., 2013; Maljovec et al., 2013), as a flexible and scalable alternative to more traditional Gaussian process based methods. We emphasize, in particular, that the sensitivity analysis of a BMARS surrogate model can be performed without requiring evaluations of the surrogate, and thus without Monte Carlo error. This is the case for a limited number of surrogate models, perhaps the most popular of which is polynomial chaos (Sudret, 2008). While both BMARS and polynomial chaos use polynomial expansions that facilitate the analytical calculations of the integrals required for a global sensitivity analysis, BMARS is especially well suited for high dimensional problems. This is because BMARS follows an adaptive strategy to fitting the response surface, producing flexible and parsimonious emulators. For completeness, Oakley & O'Hagan (2004) propose a method for getting sensitivity indices under a scalar Gaussian process emulator that is Monte Carlo free and analytical in some cases. However, the BMARS method we introduce is better suited to much larger datasets than Gaussian process methods.

Our interest lies especially in the application of these methods to computer models that generate functional data. Such models present a particular challenge as they often produce output of massive dimension, as, for every combination of the input parameters, there are hundreds or even thousands of simulated values. The example that motivated our interest in BMARS emulators is a computer model of the deformation of a metal plate during pressure driven experiments. This model has seven inputs detailing the configuration of the plate and outputs a curve representing the profile of the plate after deformation, as shown in Figure 2.1. The most natural way to approach emulating a model that outputs functions, say of r, would be to think of r as though it was another input to the model. However, if we have m model evaluations that each output a function on a grid of size n, then this approach to emulation would need to be able to handle data of size mn, which can be difficult for large m or large n.

In this chapter, we show that the BMARS formulation is well suited for functional output. We introduce sensitivity analysis methods for functional data, and give analytical sensitivity measures for the functional BMARS model. We also introduce some alterations to the BMARS priors to induce regularization as well as parallel tempering in the MCMC sampling scheme to allow for efficient exploration of the highly multimodal model space.

The structure of the chapter is as follows. We first review the definition of the Sobol' index, which we use as a measure of global sensitivity, in Section 2.1. We detail some approaches to obtaining the Sobol' index for models with functional output. We then explain our approach to fitting the BMARS surrogate model in

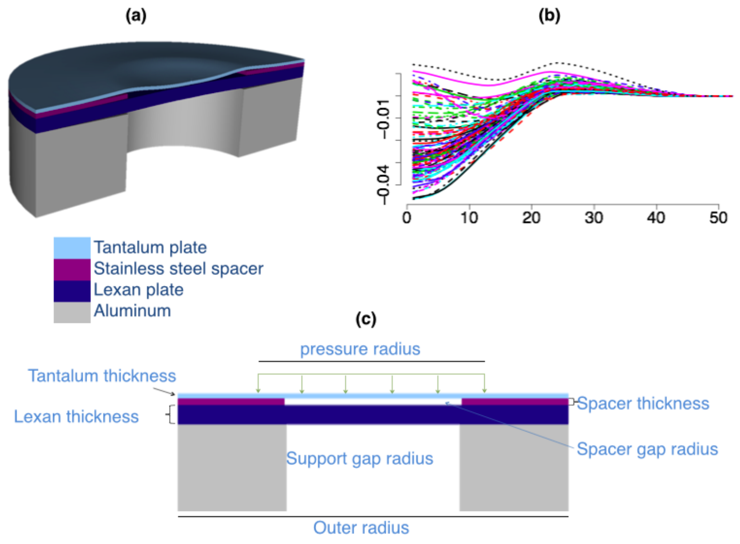


Figure 2.1: Formulation of the plate deformation model: (a) shows the configuration of the protective (half) plate, (b) shows the output from 104 evaluations of the computer model with different variable combinations, and (c) shows the variables. Each curve in (b) is the output from one model evaluation and represents the profile of the tantalum plate after the experiment. Zero in the x-axis of (b) represents the center of the plate.

Section 2.2, including our prior specification, computational approach for functional model output, tempering scheme, and analytical expressions for the Sobol' sensitivity indices. In Section 2.3, we present a simulation study to demonstrate the effectiveness of the sensitivity analysis approach. In Section 2.4, we perform a sensitivity analysis of a model of the deformation of a protective plate used in pressure driven experiments. Finally, we discuss our findings in Section 2.5.

2.1 Global Sensitivity Indices for Functional Output

The Sobol' sensitivity indices (Sobol', 1990) decompose the variance of the model output in terms of the variance due to main effects (first order effects) for each of the inputs, and variance due to interaction effects (higher order effects). This is the same task that the ANOVA decomposition accomplishes in linear models, but for models that may be highly nonlinear. To start, let the function f represent our model (or surrogate model). Say that f is a function of p inputs, $\mathbf{x} = (x_1, \ldots, x_p)$, each with domain in the unit interval. Further, say that f has functional output that is a function of r, also with domain in the unit interval. Then $f(r, \mathbf{x})$ is a scalar.

The Sobol' decomposition of such a functional output model could proceed in two ways. First, consider writing f as

$$f(r, \mathbf{x}) = f_0(r) + \sum_{i=1}^{p} f_i(r, x_i) + \sum_{1 \leq i < j \leq p} f_{ij}(r, x_i, x_j) + \ldots + f_{1 \ldots p}(r, x_1, \ldots, x_p)$$

where

$$\begin{aligned} f_0(r) &= \int_0^1 \ldots \int_0^1 f(r, \mathbf{x}) d\mathbf{x} \\ f_i(r, x_i) &= \int_0^1 \ldots \int_0^1 f(r, \mathbf{x}) d\mathbf{x}_{-i} - f_0(r) \\ f_{ij}(r, x_i, x_j) &= \int_0^1 \ldots \int_0^1 f(r, \mathbf{x}) d\mathbf{x}_{-ij} - f_0(r) - f_i(r, x_i) - f_j(r, x_j) \end{aligned}$$

and so on with $\mathbf{x}_{-ij}$ being the vector $\mathbf{x}$ without elements i and j. These terms are interpretable as the overall mean function, $f_0(r)$, the main effect function for variable i, $f_i(r, x_i)$, the two way interaction effect function for variables i and j, $f_{ij}(r, x_i, x_j)$, and so on. Proceeding with the method for obtaining Sobol' indices, we have that

$$\begin{aligned} Var(f(r, \mathbf{x})) = \sum_{i=1}^{p} Var(f_i(r, x_i)) + \sum_{1 \leq i < j \leq p} Var(f_{ij}(r, x_i, x_j)) + \ldots \\ + Var(f_{1 \ldots p}(r, x_1, \ldots, x_p)). \end{aligned} \tag{2.1}$$

Thus, we have decomposed the variance of the function in terms of the variance from the main effects of each input and the variance from the interactions between inputs. Each of these terms is a function of r, as are the associated sensitivity indices $S_{i_1 \dots i_l}(r) = Var(f_{i_1 \dots i_l}(r, i_1, \dots, i_l))/Var(f(r, \mathbf{x}))$, so that $S_{i_1 \dots i_l}(r)$ is the proportion of variance in the model output at r explained by the interaction between inputs $i_1 \dots i_l$ in addition to the variance explained by main effects coming from these inputs and interactions of lesser order between these inputs. We may also be interested in the cumulative sensitivity of the model to a particular input at r. The total sensitivity for input i at r is defined as $T_i(r) = S_i(r) + \sum_{j \neq i} S_{ij}(r) + \dots + S_{1 \dots p}(r)$ and interpreted relative to $T_j(r)$ as the importance of input i compared to input j at r. Though these are no longer interpretable as proportions, they give us an idea of the overall importance of an input relative to the other inputs at a particular r.

The second way we could obtain the Sobol' decomposition of such a functional output model would be to augment the vector of inputs to $\mathbf{z} = (r, \mathbf{x})$. We then obtain the decomposition

$$f(z_1, \dots, z_d) = f_0 + \sum_{i=1}^{d} f_i(z_i) + \sum_{1 \leq i < j \leq d} f_{ij}(z_i, z_j) + \dots + f_{1 \dots d}(z_1, \dots, z_d), \quad (2.2)$$

where $d = p + 1$. We proceed with the traditional Sobol' variance decomposition to obtain S_i, S_{ij}, T_i, etc., which are no longer functions of r. Letting $r = z_1$, S_1 is the proportion of variance in the model output due to the main effect produced by the functional variable r. This provides interesting insights, especially when determining whether the bulk of the variance in the model output is due to the functional variables or the inputs to the computer model.

These approaches to functional sensitivity analysis have complementary strengths, as will be demonstrated in the plate deformation example in Section 2.4.

2.2 BMARS

We now discuss the problem of fitting a BMARS model to a set of simulated output in more detail. Multivariate adaptive regression splines (MARS) were proposed in Friedman (1991c) as a continuous alternative to recursive partitioning methods like CART. The adaptive part of MARS is what makes it work for high dimensions (large numbers of input variables). Multivariate (non-adaptive) regression splines might take a tensor product of one dimensional splines to get a multivariate spline, but with only a few dimensions the number of knots explodes and the curse of dimensionality becomes debilitating. The MARS model instead chooses knots adaptively, learning where to put them in the same way that partitions are learned in classification and regression tree (CART) models (Breiman et al., 1984). Thus, if there is not sufficient utility in having a knot at some point in the high dimensional input space, it will not be included. Further, the MARS model has a natural ANOVA type decomposition, making main effects and interactions easy to understand. Unlike CART, MARS produces continuous models, creating computational difficulties but resulting in more realistic models.

The original MARS inference is done using a forward stepwise algorithm followed by a backward stepwise algorithm similar to versions of inference for recursive partitioning. The Bayesian version (Denison et al., 1998b; Nott et al., 2005) considers all the unknowns as random variables and assigns prior distributions to them. Reversible Jump Markov chain Monte Carlo (RJMCMC) (Green, 1995) methods are used to obtain transdimensional samples from the joint posterior. Our approach uses aspects of Denison et al. (1998b) and Nott et al. (2005) but with a number of significant alterations to the priors, a strong focus on efficient handling of functional data, and tempered RJMCMC to achieve more efficient sampling.

We note that MARS is not an interpolator, meaning it does not have the property that the fitted emulator replicates the computer model runs exactly. When emulating deterministic computer models, using an interpolator seems like a natural choice. However, Gramacy & Lee (2012) point out that emulation is often improved when a small-scale measurement error is included.

We will first formulate the model and discuss our choice of priors. We then discuss efficient computation for functional data. Next, we discuss the need for tempering, and our tempering approach. Finally, we discuss how to analytically obtain the Sobol' decomposition of a BMARS model.

2.2.1 Model formulation

To use the BMARS approach for functional data, we include the functional variable as an additional input to the model. At this point, we consider only the case of output as a function of one variable, as is the case in the plate deformation example. Let $y_i(r)$ denote the simulator output at r using input vector $\mathbf{x}_i$, where r denotes the functional variable and $i = 1, \ldots, n_x$. Then we model $y_i(r)$ as

$$y_i(r) = f(r, \mathbf{x}_i) + \epsilon_i(r), \quad \epsilon_i(r) \sim N(0, \sigma^2)$$

where we use a basis expansion to specify f,

$$f(r, \mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m B_m(r, \mathbf{x}).$$

If we let $\mathbf{z} = (r, \mathbf{x})$, the m^{th} basis function $B_m(r, \mathbf{x}) = B_m(\mathbf{z})$ is given by

$$B_m(\mathbf{z}) = \prod_{k=1}^{K_m} \left[s_{km}\left(z_{v_{km}} - t_{km}\right)\right]_+^{\alpha} \tag{2.3}$$

which is a tensor product of piecewise polynomials of degree α. The value of K_m determines the degree of interaction in the basis function, where $K_m \in \{1, \ldots, K_{\max}\}$. The term v_{km} is an index to determine which variable is used (which element of $\mathbf{z}$). We allow each variable to be used at most one time in each basis function. The term t_{km} is called a knot, and is a value in the domain of the variable $z_{v_{km}}$. Following previous MARS implementations, a knot t_{km} is only allowed at one of the marginal locations where we have a value of $z_{v_{km}}$ in the data for identifiability purposes. The term s_{km} is a value in $\{-1, 1\}$. The function $[\cdot]_+$ is defined as $\max(\cdot, 0)$, meaning that it makes any negative values zero.

This notation follows that of Friedman (1991c), with the exception of the inclusion of the functional variable. The functional variable could be space, time, or any other such variable over which the output is measured. In practice, functional output is usually given on a grid. While nothing in the formulation above requires the output for $y_i(r)$ to be on the same grid of r values as $y_j(r)$, for simplicity we assume that this is the case: $y_1(r), \ldots, y_n(r)$ are given on the same grid of n_r values. We will denote this grid as $\mathbf{r}$. For example, if r denotes time, n_r would be the number of time points on which the output is given, and $\mathbf{r}$ would be the vector of those time points. Treating the functional variable as an additional input results in a univariate MARS fitted over $N = n_x \times n_r$ data points. If we define the $n_x \times p$ matrix $\mathbf{X}$ such that the i^{th} row of $\mathbf{X}$ is $\mathbf{x}_i$, then the data used to fit the MARS model are the rows of $[\mathbf{1}_{n_x} \otimes \mathbf{r}, \mathbf{X} \otimes \mathbf{1}_{n_r}]$, where $\otimes$ denotes the Kronecker product. For large

n_x or n_r manipulating, and even storing the full matrix $\mathbf{X}$ can be challenging. More specifically, in the course of inference and prediction, we will plug in values of $\mathbf{x}$ and r to Equation 2.3 to get discretized versions of the basis functions, which we call basis vectors. We denote the m^{th} basis vector as $\mathbf{B}_m$, and the $(M+1) \times N$ matrix of basis vectors (with an additional column of ones) as $\mathbf{B}$. When N is large and M is moderately large, storing this matrix becomes costly. We will discuss a computational strategy that simplifies this approach in Section 2.2.2.

To complete the Bayesian specification of the model, we will specify priors for the unknown parameters. Our unknowns include the number of basis functions M, the basis function coefficients $\mathbf{a} = (a_1, \ldots, a_M)$, the variance σ^2, and the parameters used to build each basis function. For the m^{th} basis function, these are K_m, $\mathbf{s}_m = (s_{1m}, \ldots, s_{K_m m})$, $\mathbf{v}_m = (v_{1m}, \ldots, v_{K_m m})$, $\mathbf{t}_m = (t_{1m}, \ldots, t_{K_m m})$. For notation purposes, let $\mathbf{K} = (K_1, \ldots, K_M)$, $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_M)$, $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_M)$, and $\mathbf{t} = (\mathbf{t}_1, \ldots, \mathbf{t}_M)$. We will formulate our prior as

$$p\left(M, \sigma^2, \mathbf{a}, \mathbf{K}, \mathbf{s}, \mathbf{v}, \mathbf{t}, \lambda, \tau \,\middle|\, \mathbf{X}, \mathbf{r}\right) = p(\lambda) p(\tau|\mathbf{X}, \mathbf{r}) p(\sigma^2) p(M|\lambda) p(\mathbf{a}|M, \sigma^2, \mathbf{B}, \tau) \prod_{m=1}^{M} p(K_m|M) p(\mathbf{s}_m, \mathbf{v}_m, \mathbf{t}_m | K_m, M, \mathbf{X}, \mathbf{r})$$

where λ and τ are hyper parameters to be discussed below. We point out that $\mathbf{B}$ is a function of M, $\mathbf{K}$, $\mathbf{s}$, $\mathbf{v}$, $\mathbf{t}$, $\mathbf{X}$, and $\mathbf{r}$. We also note that priors conditional on such quantities as $\mathbf{X}$ and $\mathbf{r}$ do not violate the Bayesian formulation, as they are considered known.

For the number of basis functions we use a Poisson prior

$$p(M|\lambda) \propto e^{-\lambda} \lambda^M / M!$$

truncated to $M = 0, \dots, M_{\max}$, where $M_{\max}$ is the maximum allowable number of basis functions. We use a $Gamma(a_\lambda, b_\lambda)$ hyperprior for λ. For the error variance, we use a default prior, $p(\sigma^2) \propto 1/\sigma^2$. For the basis function coefficients, we use a variant of Zellner's g prior (Zellner, 1986; Liang et al., 2008) with $\mathbf{a}|\mathbf{B}, \tau, \sigma^2 \sim N\left(\mathbf{0}, \frac{\sigma^2}{\tau}\left(\mathbf{B}'\mathbf{B}\right)^{-1}\right)$ where $\tau|\mathbf{X}, \mathbf{r} \sim Gamma(1/N, 1)$. Thus, τ is centered over the unit information prior. This prior simplifies computations when compared to previous approaches that use a ridge regression prior. It also induces regularization by introducing shrinkage (which the ridge regression prior also does). For the interaction order we use a discrete uniform prior $K_m|M \sim Unif\{1, \dots, K_{\max}\},\ m = 1, \dots, M$. Most implementations of this model take $K_{\max} = 2$ as a default. However, we would like to allow for two way interactions between the input variables that can also interact with the functional variable, so we use $K_{\max} = 3$. We note that one could easily adopt a prior that assigns decreasing probability to larger values of K in order to allow for higher dimensional interactions only if the data really dictate their inclusion.

For the signs, variables, and knots, previous Bayesian approaches have considered a discrete uniform prior over all possibilities. We use a similar discrete uniform prior, but over a limited set of possibilities. The reason for this is that we want to limit how localized a basis function can become in the same way that recursive partitioning approaches do, which is to require that each partition contain a certain number of data points. This is because very local fitting often produces overfitting. In our experience functional output magnifies this issue. In the MARS approach, we do not exactly have regular partitions, since basis functions are usually overlapping in the input space. We get a gauge on how local the structure a certain basis function is trying to explain by counting the number of non-zero points in the basis vector. Although this is only a reliable measure of how localized the model is near the edge of the input space (in p dimensions), this is the part of the space where getting such a measure is particularly important since it is where MARS tends to become unstable.

As an example of the instabilities that we can encounter if we allow for all possible knot, sign, and variable combinations, consider the dataset given in Figure 2.2. There are 100 random uniform (x_1, x_2) pairs, given as black circles. If we were interested in choosing knots for the MARS basis function $[x_1 - t_1]_+[x_2 - t_2]_+$, the red dots in the left plot give the possible locations if we do not constrain the prior. The partition created by the choice of (t_1, t_2) would, in many of these cases, contain few data points. The corresponding basis function would then be trying to fit the very local structure in those few points. The right panel shows the possible knot locations (in red) if we require the partition to contain at least 20 points. It may appear that a simpler way to fix the edge instability of MARS would be to not allow marginal knots too close to the endpoints of the space, as suggested in Friedman (1991c). For instance, if we required each knot in the example to have at least 20 points marginally between it and the edge of the space, we would allow for a knot at the intersection of the lines shown in the right panel. Note, however, that there would only be one data point driving the fit of that basis function, which could lead to overfitting.

To specify the prior for the signs, variables, and knots, that correspond to

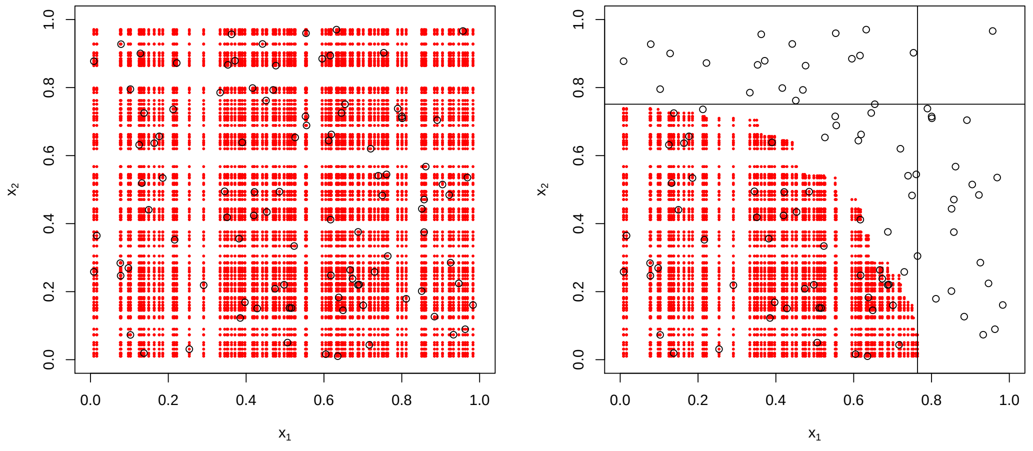


Figure 2.2: In both plots, 100 random uniform (x_1, x_2) pairs are shown with black circles. If we wanted to build the basis function $[x_1 - t_1]_+[x_2 - t_2]_+$, the red dots in the left panel show the possible (t_1, t_2) knot locations if the prior for knots is unconstrained. On the right are the possible knot locations when the resulting basis vector is constrained to have at least 20 non-zero values. The lines are placed so that there are 20 data points larger than these values in each dimension marginally to illustrate the approach advocated by Friedman.

our proposed constraint, we use the discrete uniform distribution

$$p(\mathbf{s}_m, \mathbf{v}_m, \mathbf{t}_m | K_m, M, \mathbf{X}, \mathbf{r}) = \begin{cases} c_{K_m} & \text{if } b_m \geq b \\ 0 & \text{otherwise} \end{cases}$$

where b_m is the number of non-zero values in the basis vectors and b is the minimum number of non-zero points. In practice, since we have the entire functional output for each input combination, we might consider choosing b based only on the part of the basis function that corresponds to the non-functional inputs. We can do this by replacing b above with bn_r where b is chosen as the minimum number of input points allowed to contribute to the local structure of the function, which we do in the plate deformation problem. In particular, we use $b = 20$, though we find the results are fairly robust for $b > 10$. We note again, as with the prior for K, that we could instead use a prior that places lower probability on basis functions with smaller b_m in order to allow for more localized basis functions only if the data gave strong enough evidence for them, though we consider only the discrete uniform prior in this thesis. The value of c_{Km} would usually be unimportant and this prior would merely add an indicator function to the posterior. However, in the RJMCMC algorithm, c_{Km} will play a role. The actual value of c_{Km} is the reciprocal of the number of possible basis functions with interaction order K_m, which depends on $\mathbf{X}$ and $\mathbf{r}$. For some datasets, it is feasible to run a pre-processing step to count the basis functions that meet the required criteria. For datasets where this is not the case, we recommend using, as a conservative proxy, the constant that would result from the unconstrained prior as an estimate,

$$c_{Km} = \left(\frac{1}{2}\right)^{K_m} \binom{p}{K_m}^{-1} \prod_{k=1}^{K_m} \frac{1}{n_{v_{km}}}$$

where the first term is obtained from the count of all $\mathbf{s}_m$, the second from that of the $\mathbf{v}_m$, and the third from that of the $\mathbf{t}_m$, conditional on the corresponding $\mathbf{v}_m$. Here, $n_{v_{km}}$ is the number of unique marginal values of the v_{km} input in the dataset. This prior for the knots is slightly different than previous approaches because we have n_x possible knot locations if v_{km} is one of the regular inputs and n_r possible knot locations if v_{km} is the functional variable. We find that using this estimate does not negatively influence the results in test cases.

Now, to address our settings of the remaining parameters we have not yet addressed, we first consider the spline order α. While in many cases, setting α to an integer that would ensure continuous derivatives would make sense, we often encounter instabilities when $\alpha > 1$. This is because of the erratic tail behavior of higher order polynomials, as is documented in Friedman (1991c). In practice, $\alpha = 1$ tends to work quite well even for smooth surfaces. We find that the value of $M_{\max}$ need not be finite unless computer memory constraints are encountered in the fitting. $M_{\max}$ too small should be avoided in order to allow the necessary exploration of the parameter space. The setting of the hyperparameters for λ, the mean of the distribution of the number of basis functions, can be difficult. This is because under the model formulation we have given, the only way to prevent overfitting, even with a reasonable setting of b and $K_{\max}$, is to have a strong prior keeping the number of basis functions small. Hence, this is a prior that may require tuning. Other possibilities are to use a strong prior keeping the value of σ^2 large or, as has been done in other approaches, use a large value of τ to shrink the values of $\mathbf{a}$ towards zero. In our experience, changing the value of τ has little effect. A prior keeping the value of M small seems less invasive than one that keeps σ^2 large, especially when uncertainty quantification is important. Thus, we advocate for a prior that keeps λ smaller than the actual number of basis functions that we expect. Otherwise, the slow rate at which the tail of the Poisson distribution decays makes it a weak prior.

2.2.2 Computation

We adapt the original RJMCMC stochastic search algorithm for BMARS (Denison et al., 1998b) with the additions of Nott et al. (2005) to work with the priors discussed in the previous section. The RJMCMC has three possible steps: birth (create a new basis function, add it to the model), death (remove a basis function), change (change a knot and sign in one basis function). The additions of Nott et al. (2005) allow for fast variable selection by proposing variables to be used in a new basis function with probability proportional to the number of times they have been used in the current set of basis functions.

To efficiently deal with large N, we will break each of our tensor product basis functions into two parts, the part that uses the variables $\mathbf{x}$ and the part that uses the functional variable r. We can then write basis function m as $B_m(r, \mathbf{x}) = B_m^x(\mathbf{x})B_m^r(r)$. If we let $\mathbf{S}_m^x = \{k \in (1, \ldots, K_m) : z_{km} \neq r\}$ and $\mathbf{S}_m^r = \{k \in (1, \ldots, K_m) : z_{km} = r\}$, then

$$B_m^x(\mathbf{x}) = \begin{cases} \prod_{k \in \mathbf{S}_m^x} [s_{km}(z_{v_{km}} - t_{km})]_+^\alpha & \text{if } |\mathbf{S}_m^x| > 0 \\ 1 & \text{otherwise} \end{cases}$$

$$B_m^r(r) = \begin{cases} [s_{km}(r - t_{km})]_+^\alpha|_{k \in \mathbf{S}_m^r} & \text{if } |\mathbf{S}_m^r| > 0 \\ 1 & \text{otherwise} \end{cases}.$$

We then create the n_x-vector $\mathbf{B}_m^x = (B_m^x(\mathbf{x}_1), \ldots, B_m^x(\mathbf{x}_{n_x}))'$ and the n_r-vector $\mathbf{B}_m^r = (B_m^r(r_1), \ldots, B_m^r(r_{n_r}))'$. Note, then, that a MARS basis vector is written as $\mathbf{B}_m = \mathbf{B}_m^x \otimes \mathbf{B}_m^r$. Thus, we have broken the MARS basis vector into the part from the input variables ($\mathbf{B}_m^x$) and the part from the functional variable ($\mathbf{B}_m^r$). The cases when these are vectors of ones occur if the basis function uses only the functional variable or only the design variables.

Now, the matrix of basis functions $\mathbf{B}$ can be written as

$$[\mathbf{1}_N, \mathbf{B}_1^x \otimes, \mathbf{B}_1^r, \ldots, \mathbf{B}_M^x \otimes, \mathbf{B}_M^r].$$

If $\mathbf{B}^x = [\mathbf{1}_{n_x}, \mathbf{B}_1^x, \ldots, \mathbf{B}_M^x]$ and $\mathbf{B}^r = [\mathbf{1}_{n_r}, \mathbf{B}_1^r, \ldots, \mathbf{B}_M^r]$ then we have that $\mathbf{B} = \mathbf{B}^x * \mathbf{B}^r$ where $*$ denotes the Khatri-Rao product (Kolda, 2006; Lev-Ari et al., 2005). This Kronecker structure simplifies many matrix calculations. For instance, from properties of Khatri-Rao products, $\mathbf{Ba} = \text{vec}(\mathbf{B}^r \text{diag}(\mathbf{a})\mathbf{B}^{x\prime})$. This is an important quantity in prediction, and under this formulation it can be calculated without building explicitly the whole matrix $\mathbf{B}$. Other important quantities used for inference also have simplified forms, such as $\mathbf{B}'\mathbf{B} = (\mathbf{B}^{x\prime}\mathbf{B}^x) \circ (\mathbf{B}^{r\prime}\mathbf{B}^r)$ where $\circ$ denotes the Hadamard (elementwise) product, and $\mathbf{B}'\text{vec}(\mathbf{Y}) = \text{vecd}(\mathbf{B}^{r\prime}\mathbf{Y}\mathbf{B}^x)$ where $\mathbf{Y}$ is the $n_r \times n_x$ matrix of data. Here, $\text{vec}(\cdot)$ denotes the columnwise stacking of a matrix and $\text{vecd}(\cdot)$ denotes the vectorization of the diagonal of a matrix. These simplifications mean that we do not need to build the large matrix $\mathbf{B}$ explicitly, but can instead work with the smaller $\mathbf{B}^r$ and $\mathbf{B}^x$.

Further simplifications come from the fact that in the RJMCMC algorithm, we are only adding, deleting, or changing one basis function at a time. For instance, if we were adding a basis function, $\mathbf{B}_*$, we need only calculate $\mathbf{B}_*{}'\mathbf{B}_* = (\mathbf{B}_*^{x\prime}\mathbf{B}_*^x)(\mathbf{B}_*^{r\prime}\mathbf{B}_*^r)$ and $\mathbf{B}'\mathbf{B}_* = (\mathbf{B}^{x\prime}\mathbf{B}_*^x) \circ (\mathbf{B}^{r\prime}\mathbf{B}_*^r)$ in order to update $\mathbf{B}'\mathbf{B}$. Deleting is simpler in that if we have selected the m^{th} basis function to delete, we need only remove the $(m+1)^{\text{th}}$ row and column of $\mathbf{B}'\mathbf{B}$. We use similar updates for $\mathbf{B}'\text{vec}(\mathbf{Y})$.

2.2.3 Tempering

The BMARS model, along with Bayesian versions of CART (Denison et al., 1998a; Chipman et al., 1998), yields a posterior distribution over a space of models that is often highly multimodal. As such, the RJMCMC algorithm tends to have difficulty exploring the entire posterior. In the BMARS case, after selecting a set of basis functions and ending up in one of these modes, it is unlikely that enough basis functions will be deleted to allow for the chain to move to another mode. BMARS and Bayesian CART approaches to dealing with this problem have often centered around restarting the MCMC algorithm after a certain number of iterations.

Restarting the MCMC, or equivalently, taking samples from parallel chains, has the undesirable property of representing modes based on their "basins of attraction" instead of the total probability associated with each mode (Neal, 1996).

We overcome these problems by using parallel tempering, also known as Metropolis coupled MCMC (Geyer, 1991). That is, we run multiple MCMC chains in parallel, each with a slightly different stationary distribution, and allow them to swap states. Particularly, if we denote our parameter vector as $\boldsymbol{\theta} = (M, \sigma^2, \mathbf{a}, \mathbf{K}, \mathbf{s}, \mathbf{v}, \mathbf{t}, \lambda, \tau)$, our data as $\mathbf{y}$, and the posterior of interest as $\pi(\boldsymbol{\theta}|\mathbf{y})$, we will define a series of altered posterior distributions as $\pi_1(\boldsymbol{\theta}|\mathbf{y}), \ldots, \pi_T(\boldsymbol{\theta}|\mathbf{y})$. In this thesis, we define $\pi_i(\boldsymbol{\theta}|\mathbf{y}) \propto \pi(\boldsymbol{\theta}|\mathbf{y})^{t_i}$, where t_i is called the inverse temperature parameter and the sequence $1 = t_1 > t_2 > \ldots > t_T > 0$ is called the temperature ladder. Small values of t_i flatten posterior modes and raise troughs, and correspond to "heated" chains where mixing is easier. A swap of the current state from chain i (called $\boldsymbol{\theta}_i$) with that of the current state from chain j (called $\boldsymbol{\theta}_j$) is accepted with probability

$$\alpha_{\text{swap}} = \min\left\{1, \frac{\pi_i(\boldsymbol{\theta}_j|\mathbf{y})\pi_j(\boldsymbol{\theta}_i|\mathbf{y})}{\pi_i(\boldsymbol{\theta}_i|\mathbf{y})\pi_j(\boldsymbol{\theta}_j|\mathbf{y})}\right\}.$$

We note that state vectors $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ are possibly of different dimension, so quantities like $\pi_i(\boldsymbol{\theta}_j|\mathbf{y})$ are not necessarily intuitive. Furthermore, it is not immediately obvious that the posterior normalizing constants will cancel out because of the dimensionality difference. We will show explicitly what we mean when we write $\pi_i(\boldsymbol{\theta}_j|\mathbf{y})$ and why it works with the unnormalized posterior.

Call the unnormalized posterior distribution function $g(\boldsymbol{\theta}|\mathbf{y})$. Then, $\pi(\boldsymbol{\theta}|\mathbf{y}) = g(\boldsymbol{\theta}|\mathbf{y})/c$ where c is obtained by marginalizing $g(\boldsymbol{\theta}|\mathbf{y})$ over all the parameters in $\boldsymbol{\theta}$. Note that c is the same no matter the dimension of $\boldsymbol{\theta}$ since we are marginalizing over all possible dimensions. What we mean by $\pi_i(\boldsymbol{\theta}|\mathbf{y})$, no matter the dimension of $\boldsymbol{\theta}$, is $\pi_i(\boldsymbol{\theta}|\mathbf{y}) = [g(\boldsymbol{\theta}|\mathbf{y})/c]^{t_i}/a_i$ where a_i is obtained from marginalizing $[g(\boldsymbol{\theta}|\mathbf{y})/c]^{t_i}$ over all the parameters in $\boldsymbol{\theta}$. Now, let $c_i = c^{t_i}a_i$ and we have that $\pi_i(\boldsymbol{\theta}|\mathbf{y}) = g(\boldsymbol{\theta}|\mathbf{y})^{t_i}/c_i$. Note then that $\pi_i(\boldsymbol{\theta}_j|\mathbf{y})$ and $\pi_i(\boldsymbol{\theta}_i|\mathbf{y})$ have the same normalizing constants, c_i. Thus, the acceptance ratio can be written as

$$\frac{\pi_i(\boldsymbol{\theta}_j|\mathbf{y})\pi_j(\boldsymbol{\theta}_i|\mathbf{y})}{\pi_i(\boldsymbol{\theta}_i|\mathbf{y})\pi_j(\boldsymbol{\theta}_j|\mathbf{y})} = \left(\frac{g(\boldsymbol{\theta}_j|\mathbf{y})}{g(\boldsymbol{\theta}_i|\mathbf{y})}\right)^{t_i-t_j}$$

which means that the normalizing constants cancel and we need only keep track of the unnormalized posterior for each chain.

This can be done in parallel, as in Altekar et al. (2004), so that it requires very little communication between chains. Rather than explicitly swapping states, we swap temperatures. Also, we only keep samples from the true posterior (the coolest chain), meaning that the memory footprint of such an algorithm is not substantially larger than when we do not use tempering. While simulated tempering (Geyer & Thompson, 1995) might provide better mixing than parallel tempering (Atchadé et al., 2011), specification of a pseudo-prior for the inverse temperature is difficult. Further, being able to run the chains in parallel is very useful to those who have access to large parallel computers, which is common among those working in uncertainty quantification.

2.2.4 Sobol' Indices for BMARS

We now discuss the Sobol' decomposition of a BMARS model. In particular, we note that the required integration can be done in closed form. Chen et al. (2005) describe conditions under which tensor product basis function expansions have analytical Sobol' decompositions. For MARS models, these conditions are that we can analytically perform the integration

$$C_v(s,t) = \int_0^1 [s(x_v - t)]_+^\alpha dx_v \tag{2.4}$$

$$C_v(s_1,t_1,s_2,t_2) = \int_0^1 [s_1(x_v - t_1)]_+^\alpha [s_2(x_v - t_2)]_+^\alpha dx_v. \tag{2.5}$$

We find that this is possible when α is a positive integer. Particularly

$$C_v(s,t) = \begin{cases} \frac{(1-t)^{\alpha+1}}{\alpha+1} & s = 1 \\ \frac{t^{\alpha+1}}{\alpha+1} & s = -1 \end{cases}$$

$$C_v(s_1,t_1,s_2,t_2) = \begin{cases} \int_{t_2}^1 \left[(x_v - t_1)(x_v - t_2)\right]^\alpha dx_v & s_1 = s_2 = 1 \\ \int_0^{t_1} \left[(x_v - t_1)(x_v - t_2)\right]^\alpha dx_v & s_1 = s_2 = -1 \\ (-1)^\alpha \int_{t_1}^{t_2} \left[(x_v - t_1)(x_v - t_2)\right]^\alpha dx_v & s_1 = 1, s_2 = -1 \\ 0 & s_1 = -1, s_2 = 1 \end{cases} \tag{2.6}$$

assuming, without loss of generality, that $t_1 \leq t_2$. The integrals in Equation (2.6) are

$$\int_a^b \left[(x - t_1)(x - t_2)\right]^\alpha dx = \left[\sum_{i=0}^{\alpha} p_i (x - t_1)^{\alpha - i}(x - t_2)^{\alpha+1+i}\right]_{x=a}^{x=b}$$

where $p_i = \frac{(\alpha!)^2(-1)^i}{(\alpha-i)!(\alpha+1+i)!}$. We can use using Equation 2.4 to write quantities like $\hat{f}_{ij}(z_i,z_j) = \int_0^1 \dots \int_0^1 f(\mathbf{z}) d\mathbf{z}_{-ij}$, the non centered version of $f_{ij}(z_i,z_j)$ in Equation 2.2. We can write quantities like $\int_0^1 \int_0^1 \hat{f}_{ij}(z_i,z_j)^2 dz_i dz_j$, important in finding

$Var(f_{ij}(z_i, z_j))$, using Equations 2.4 and 2.5. Similarly, we can use Equations 2.4 and 2.5 to write $f_{ij}(r, x_1, x_2)$ and $Var(f_{ij}(r, x_1, x_2))$ where we never integrate over r.

Specifically, if we are considering a set of variables indexed by $W = \{i_1, \ldots, i_l\}$, the quantity of interest from Equation 2.1 can be obtained as $Var(f_W(r, W)) = \sum_{U \in P}(-1)^{|W|-|U|}V_U(r)$ where P is the power set of W excluding the empty set, $|U|$ denotes the size of set U, and

$$\begin{aligned} V_U(r) = &\sum_{m_1=1}^{M}\sum_{m_2=1}^{M} a_{m_1}a_{m_2}B^r_{m_1}(r)B^r_{m_2}(r) \\ &\left\{\prod_{k \in U_1} C_{v_{km_1}}(s_{km_1}, t_{km_1}) \prod_{k \in U_2} C_{v_{km_2}}(s_{km_2}, t_{km_2}) \prod_{k \in U_{12}} C_{v_{km_1}}(s_{km_1}, t_{km_1}, s_{km_2}, t_{km_2}) \right. \\ &\left. - \prod_{k \in \mathbf{S}_x} C_{v_{km_1}}(s_{km_1}, t_{km_1}) \prod_{k \in \mathbf{S}_x} C_{v_{km_2}}(s_{km_2}, t_{km_2})\right\}. \end{aligned}$$

Here $U_1 = \{k : \forall l, v_{km_1} \neq v_{lm_2}\}$, U_2 is defined similarly with m_1 and m_2 switched, $U_{12} = \{k : v_{km_1} \in U\} \cap \{k : v_{km_2} \in U\}$, and $B^r_m(r)$ and $\mathbf{S}_x$ are defined as in Section 2.2.2.

For each posterior sample, we can calculate the Sobol' decomposition to quantify the uncertainty of the sensitivity indices. This would be uncertainty due to variance in the emulator, not due to Monte Carlo error, as is usually the case.

2.3 Simulation

Consider the function $f(\mathbf{x}) = 10\sin(2\pi x_1 x_2) + 20\,(x_3 - 0.5)^2 + 10x_4 + 5x_5$, with each $x_i \in [0, 1]$, which is a slight alteration of the Friedman function (Friedman, 1991c; Denison et al., 1998b; Gramacy & Lee, 2012; Surjanovic & Bingham, 2017).

We treat x_1 as a functional variable. In order to get more flexible functional output we replace π in the Friedman function with 2π. The integrals necessary to obtain the Sobol' sensitivity indices for this function are mostly analytical, with derivations given in Appendix A. The integrals that are not analytical have solutions that can be represented using series approximations to an arbitrary degree of accuracy.

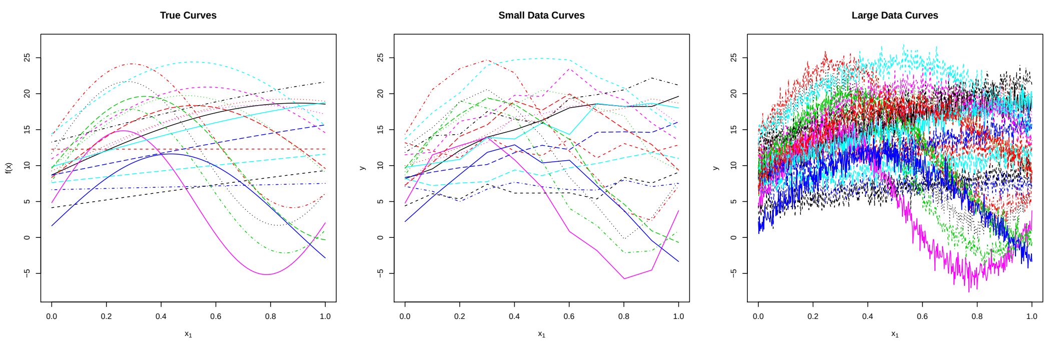


Figure 2.3: A few simulated curves, $f(\mathbf{x})$, are given on the left in terms of the functional variable x_1. The middle and right plots show the small and large data curves corresponding to the curves on the left.

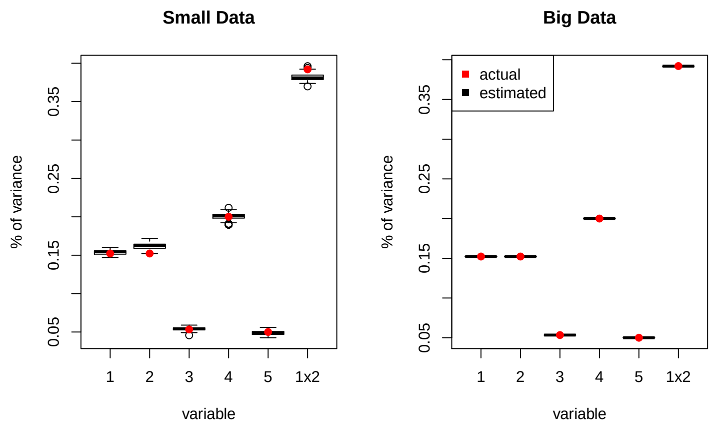


Figure 2.4: Posterior distributions of sensitivity indices represented with boxplots for the small and large datasets. Actual values of sensitivity indices are given with red dots.

We simulate n_x values of $x_2, \ldots, x_5$ from the uniform hypercube and set x_1 to be a grid of values of length n_r and use these to generate $f(\mathbf{x})$ for the $n_x \times n_r$ combinations of $\mathbf{x}$. We add standard Normal errors to the simulated values of $f(\mathbf{x})$.

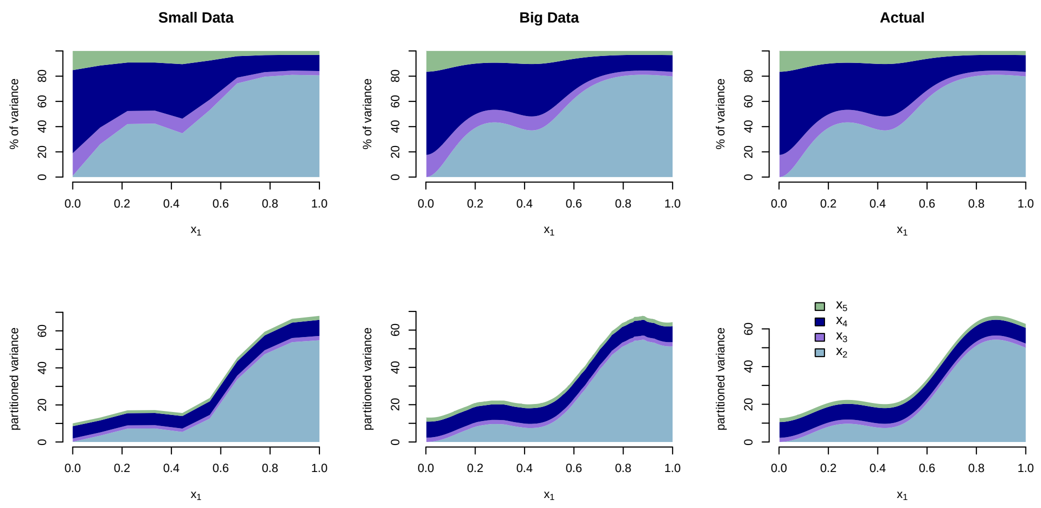


Figure 2.5: Functional pie charts of posterior mean sensitivity indices compared to true values. The left panels show the sensitivity indices using the smaller data set. The middle shows these indices using the larger dataset. The right panels show true values. The top panels show functional pie charts while the bottom show the partitioned variance.

We further generate n_x values of $x_6, \ldots, x_p$ that will be extraneous variables. We will fit our BMARS emulator to these data. This corresponds to having n_x model runs, each of which outputs a curve on a grid of n_r values. Further, the computer model takes p inputs, but only four of them are meaningful.

We consider two settings of n_x, n_r, and p that demonstrate cases with small and large data. For the small data case, we use $n_x = 100$, $n_r = 10$, and $p = 5$. For the large data case, we use $n_x = 20000$, $n_r = 500$, and $p = 200$. To demonstrate what these functional data look like, twenty curves with different settings of $\mathbf{x}$ are shown in Figure 2.3 under these two settings of n_r with standard Normal error, in addition to the true curves. Fitting BMARS models to the small and large datasets took 14 and 966 seconds, respectively, on a personal computer with a 1.7 GHz Intel Core i7 processor. The posterior median number of basis functions used for the small and large datasets is 26 and 95. In these simulations, we did not use tempering. We used default values of priors for the small data case. In the large data case, we used a strong prior on M to keep the number of basis functions small for computation purposes.

We demonstrate our two functional sensitivity analysis approaches using BMARS emulators built using these datasets. The sensitivity indices when x_1 is treated as one of inputs are given in Figure 2.4, along with the true values. The functional sensitivity indices (posterior mean) are given in the form of functional pie charts (Saltelli et al., 2000; Lamboni et al., 2009) in Figure 2.5, along with the true functional sensitivity indices. In both approaches, we are able to capture the important elements quite well for both the small and large data cases. Since the Sobol' indices are available analytically when we use the BMARS emulator, discrepancy in the sensitivity indices (which is quite small in our simulation) is due only to emulation discrepancy. The sensitivity analysis is more accurate when we use more data and has more uncertainty when we have small data, as we would expect. Our BMARS methods are able to efficiently handle the large data analysis, where use of the Gaussian process would have been infeasible.

We note that deterministic computer model output would not have unexplained noise like our simulation does. Similar simulations with noiseless data produce similar sensitivity indices, though regularization becomes more important when fitting the BMARS model to limit the number of basis functions used. This limitation prevents overfitting in the small data case. While the large data case has no issues with overfitting, limiting the number of basis functions is desirable for computational purposes.

2.4 Plate Deformation Model Sensitivity Analysis

In this section, we use the above formulation to create an emulator for the plate deformation model and perform a sensitivity analysis. The plate deformation model has seven inputs controlling the configuration of a tantalum plate used to protect a diagnostic imager during pressure driven experiments, as shown in Figure 2.1(c). The output from one model run is a curve representing the profile of the deformed plate starting from the center of the plate and extending along the radius to the end of the plate. Hence, the functional variable r in this case is called radial position. Each curve is given on the same grid of $n_r = 517$ equally spaced points. We have model runs corresponding to $n_x = 104$ combinations of $\mathbf{x}$ specified using a Latin hypercube design, for a total of 53,768 simulated values. A separate Latin hypercube was used to obtain 34 model runs that will be used to test the fit of the emulator.

We use the BMARS formulation given above to build a surrogate model. We use $\alpha = 1$, $M_{\max} = 300$, and a hyperprior for λ that keeps it close to zero. Specifically, we use $a_\lambda = 1$ and $b_\lambda = 10^{300}$ (chosen by cross-validation). Such a large value of b_λ is necessary in this case because of the combination of the smoothness of the curves and the large number of possible basis functions. If we did not limit basis functions based on partition size, there would be more than 1.2×10^9 possible basis functions, which means more than $2^{1.2\times 10^9}$ possible models. Hence, $b_\lambda = 10^{300}$, which controls the regularization on the number of basis functions, is not an exorbitantly large number when compared with the size of the model space. We note that the resultant Gamma hyperprior has so little variance that it may be unnecessary here, but we include it for consistency. When cross-validating, we compared mean squared error averaged over radial position for values of b_λ on the $\log_{10}$ scale from 0 (results in the maximum number of basis functions) to 305. We considered other values of the maximum degree of interaction (fixed at $K_{\max} = 3$) and the minimum number of non-zero values in a basis function (fixed at $b = 20$), but did not find better cross-validated fits.

We note that when the curves are smooth, σ^2 plays the role of quantifying the variance in the data not explained by the model, and will thus be larger as we impose stronger regularization. Posterior exploration is especially difficult in this case because moving from one posterior mode to another may require a substantial increase in σ^2, which is not likely to be favored in a typical RJMCMC chain. This is where tempering becomes extremely useful. We choose a temperature ladder through experimentation, and find that the maximum inverse temperature at which we are able to mix well over the model space is 0.001. We find that mixing is reasonable when we use 100 inverse temperatures in the pattern $t_i = 1 - \Phi\left(\frac{i-50.5}{13.2}\right)$, where Φ denotes the standard Normal cdf. Plots of predicted curves (point-wise posterior means) and residual curves for the training and test data are given in Figure 2.6. These plots show that we are able to explain most of the variation in the computer model runs with the emulator. The posterior predictions for four individual curves from the test set are shown in Figure 2.7 with 95% central regions (curve-wise) constructed following Sun & Genton (2012).

Upon obtaining a suitable BMARS surrogate model, we use the analytical Sobol' decomposition as described above to obtain the sensitivity indices for each posterior sample. First, consider the model sensitivity when we treat the functional variable as one of the inputs. Figure 2.8 shows the posterior distributions of the sensitivity indices for the most important main effects and interactions as boxplots,

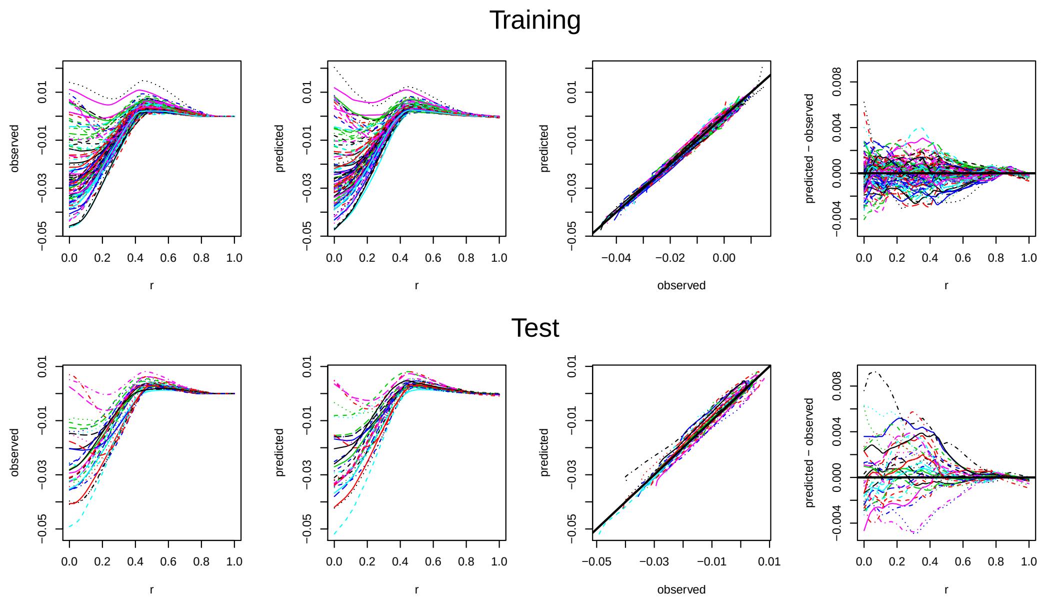


Figure 2.6: Model fit for the training and test data. From left to right the panels show the model runs (observed data), our prediction (posterior mean) of the model runs, the observed curves against the predicted curves, and residual curves. The top panels show predictions of the training data while the bottom panels show predictions of the test data.

as well as the posterior distributions of the total sensitivity indices. Importance was determined by ranking means. These show that radial position (the functional variable) explains most of the variance, and that the most important input variable is the spacer thickness. These variables have a strong interaction with each other and show up in interactions with other variables. The reality of this problem is that all the effects should be interactions with the functional variable, since there is nearly zero variance in the model output at the large radial positions. However, because of the sequential nature of the Sobol' decomposition (sensitivity indices for interactions are the additional variance explained when the main effects and lower order interactions are already included), it is not surprising to see important terms that do not interact with radial position. It could be that the higher order interaction with radial position only contributes a small amount.

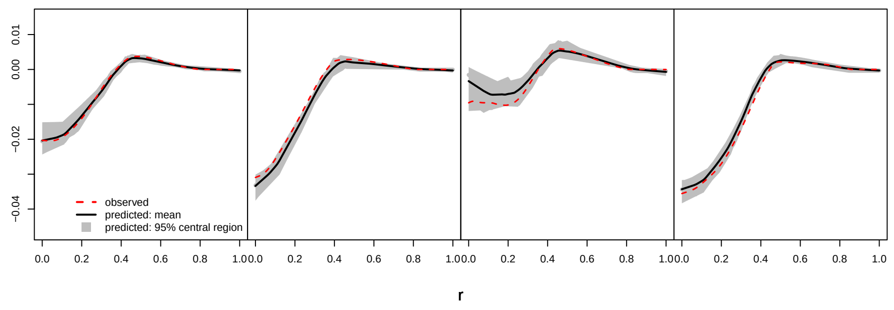


Figure 2.7: Predicted mean curves with 95% central regions for four model runs from the test data.

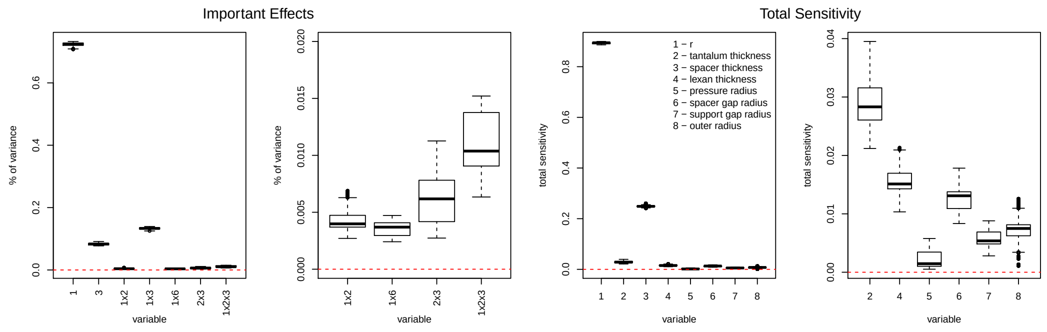


Figure 2.8: Sensitivity analysis including the functional variable. The first boxplots are the posterior distributions of the largest sensitivity indices, indicating which main effects and interactions explain the most variance. The second plot is an enlarged version to show the sensitivity indices for terms that are hard to compare in the first plot. The next two plots similarly show the posterior distributions of the total indices. Note that the functional variable, labeled as 1, explains most of the variance.

Now, consider the approach to sensitivity analysis that takes sensitivity as a function of radial position. For each main effect and interaction we now have a sensitivity index that is a function of r. Plots of posterior mean sensitivity for the main effects and most important effects are shown in functional pie charts in Figure 2.9. Also included in Figure 2.9 are plots of how the standard deviation as a function of radial position is partitioned. Importance was determined by integrating the partitioned variance functions over radial position. We note that the plots are blank when r approaches 0.9 because many of the MCMC draws yield zero variance after that point, and we do not try to decompose zero variance. This is a desirable property, since the actual data have zero variance for these values of r. These plots show us which variables and interactions are most important at different radial positions. Clearly, the spacer thickness is most important because of the large main effect and the important interactions. These plots also show us that the spacer gap radius plays an important role for $0.3 < r < 0.4$. The lexan thickness plays an important role on its own and interacting with spacer thickness for $r > 0.3$. The tantalum thickness is important in combination with spacer thickness and on its own throughout the range of r. The spacer gap radius plays a small role for $r < 0.2$, the most varied part of the functions.

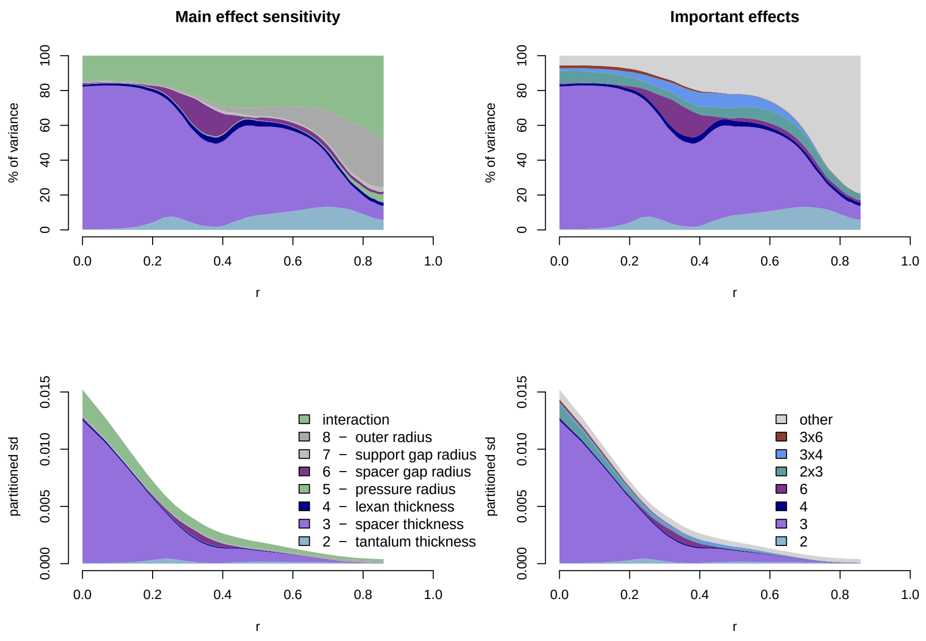


Figure 2.9: Sensitivity analysis as a function of radial position. The top plots show functional pie charts of the sensitivity indices for the main effects and the most important effects. The bottom plots show the way the actual standard deviation of the model (a function of r) is partitioned.

We see that the pressure radius, support gap radius, and outer radius of the specified ranges do not play an important role in the simulator. Under the specified ranges, it is clear that the spacer thickness deserves the most attention when it comes to designing a protective plate configuration that will be most reliable.

2.5 Discussion

We have outlined the potential benefits of performing sensitivity analysis using BMARS as an emulator for functional data. We introduced a way of doing sensitivity analysis with functional data. We gave an altered version of BMARS with priors that improve the fit in many circumstances, specifically in cases where the traditional priors lead to overfitting. We also presented modifications for the efficient handling of one dimensional functional data on a fixed grid. We showed that BMARS is fast, accurate, flexible, and can handle datasets with many variables and thousands of curves, each represented with hundreds of observations. We described how global sensitivity analysis can be performed effortlessly and without Monte Carlo error. We introduced a tempering scheme that leads to more satisfactory posterior sampling than previous approaches. Finally, we stress the fact that the Bayesian nature of the method allows for a full assessment and propagation of the uncertainties.

We note that automatic specification of a model of this form is an ongoing problem. For instance, using the automatic settings of the BMARS code that accompanies Denison et al. (2002) resulted in extreme overfitting in the plate deformation problem. Our approach requires some tuning in specifying the hyperprior for the mean number of basis functions, and we are interested in other ways of regularizing. Another area for improvement in this model is the assumed homoscedasticity. As the rightmost plots in Figure 2.6 show, the unexplained variance changes with radial position, though we assume it is constant. Conceptually, there would be no problem introducing heteroscedasticity by assuming variance depends on the functional variable and learning the form of the variance functions. However, computationally, this is troublesome because we would need to manipulate very large matrices.

Owing to the many computer models coming into use that output large and complex data, we feel that the BMARS methods outlined have promising possibilities in the field of uncertainty quantification.

Chapter 3

BASS: An R Package for Sensitivity Analysis and Fitting of Bayesian Adaptive Spline Surfaces

3.1 Introduction

The purpose of the `R` (`R` Core Team, 2016) package `BASS` (Francom, 2017) is to provide an easy-to-use implementation of Bayesian adaptive spline models for nonparametric regression. It provides a combination of flexibility, scalability, interpretability and probabilistic accuracy that can be difficult to find in other nonparametric regression software packages. As we have demonstrated in the previous chapter, the model form is flexible enough to capture local features that may be present in the data. It is scalable to moderately large datasets in both the number of predictors and the number of observations. It performs automatic variable selection. It can build nonparametric functional regression models and incorporate categorical predictors. The package can partition the variability of a resultant model using the Sobol' decomposition, providing valuable interpretation to the predictors. The Bayesian approach allows for model estimates and predictions that can be evaluated probabilistically. The package is protected under the GNU General Public License, version 3 (GPL-3), and is available from the Comprehensive `R` Archive Network (CRAN) at `https://CRAN.R-project.org/package=BASS`.

As we have already discussed, the BASS framework builds on multivariate adaptive regression splines (MARS) from Friedman (1991c). Well-developed software implementations of the MARS[1] model are available in the `R` packages `earth`, `polspline` and `mda`. The Bayesian version of MARS (BMARS) was first developed in Denison et al. (1998b). A `MATLAB` implementation of BMARS is available from the software website accompanying Denison et al. (2002).

There are a number of other `R` packages that use splines, such as `crs`, `gss`, `mgcv` and `R2BayesX`, the latter two of which include options to implement Bayesian inference methods. These packages allow (or require) the user to specify which variables are allowed to interact in what way, as well as which variables are allowed to have nonlinear main effects. The `crs` package is more similar to the packages that fit MARS models in that it can learn the structure of the model from the data. These packages report a single best model. `BASS` reports an ensemble of models (posterior draws from the model space) that can be used to make probabilistic predictions. In this way, it is more similar to Bayesian nonparametric regression packages like

[1]MARS, as a software product, is a trademark of Jeril, Inc., and is licensed exclusively to Salford Systems.

`BayesTree` and `tgp`.

We introduce the package as follows. In Section 3.2, we describe how the modeling framework is controlled by the user, including our methods for posterior sampling, modeling functional responses, and incorporating categorical inputs. In Section 3.3, we describe how to utilize the sensitivity analysis methods. Then, in Section 3.4, we walk through six examples of how to use the package. Finally, in Section 3.5, we present a summary of the package capabilities.

3.2 Bayesian adaptive spline surfaces

Here, we discuss the notation for the priors, and how it translates to parameters in the package. First, consider the priors for the σ^2 and $\mathbf{a}$. Let $\mathbf{B}$ be the $n \times (M+1)$ matrix of basis functions (including the intercept). Then we have

$$\mathbf{a}|\sigma^2, \tau, \mathbf{B} \sim N(\mathbf{0}, \sigma^2(\mathbf{B}'\mathbf{B})^{-1}/\tau) \tag{3.1}$$

$$\sigma^2 \sim InvGamma(g_1, g_2) \tag{3.2}$$

$$\tau \sim Gamma(a_\tau, b_\tau) \tag{3.3}$$

with default settings $a_\tau = 1$ and $b_\tau = 1/n$ (shape and rate) to center the prior over the unit information prior and $g_1 = g_2 = 0$ resulting in the non-informative prior $p(\sigma^2) \propto 1/\sigma^2$. In practice, the default settings are sufficient for most cases, though it can be helpful to encode actual prior information into the prior for σ^2.

Now, consider the prior for the number of basis functions, M. We use a

Poisson prior for M, truncated to be between 0 and $M_{\max}$, so that

$$p(M|\lambda) \propto \frac{e^{-\lambda}\lambda^M}{M!} \tag{3.4}$$

$$\lambda \sim Gamma(h_1, h_2) \tag{3.5}$$

where the default settings of $h_1 = h_2 = 10$ (shape and rate) in most cases induce a small number of basis functions. In practice, these hyperparameters can be key in order to prevent overfitting. More specifically, we increase h_2 (by many orders of magnitude in some cases) to bring the prior for λ very close to zero in an effort to thin out the tails of the Poisson and have fewer basis functions. We use $M_{\max}$ to give an upper bound to the computational cost, rather than to prevent overfitting. This strategy results in better fitting models since setting $M_{\max}$ too small often results in posterior sampling from only one mode.

The priors for $\mathbf{K}$, $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{v}$ are uniform over a constrained space as described in Chapter 2. The constraint in this prior makes sure basis functions have more than b non-zero values. In addition to specifying b, we also specify $K_{\max}$, the maximum degree of interaction for each basis function.

Table 3.1 shows the parameters used in the `bass` function that we have discussed thus far, and what their mathematical symbols are.

Symbol	$K_{\max}$	b	h_1	h_2	g_1	g_2	α	$M_{\max}$	a_τ	b_τ
`bass` input	`maxInt`	`npart`	`h1`	`h2`	`g1`	`g2`	`degree`	`maxBasis`	`a.tau`	`b.tau`

Table 3.1: Translation from mathematical symbols to parameters used in `bass` function.

3.2.1 Efficient posterior sampling

Our RJMCMC scheme allows us to add, delete, or change a basis function consistent with the approach of Nott et al. (2005). That is, instead of proposing to add a completely random new basis function in a reversible jump step, we use a proposal generating distribution that favors the variables and degrees of interaction already included in the model. For example, say there were five basis functions already in the model, each with degree of interaction two. Say the maximum degree of interaction was three. Then if we were proposing a new basis function we would sample the degree of interaction from $\{1, 2, 3\}$ with weights $\{w_1, w_1 + 5, w_1\}$, thus favoring two way interactions since we have seen more of them. If the nominal weight w_1 is large compared to the number of basis functions, this distribution looks more uniform. The value w_2 is the equivalent nominal weight for sampling variables to be included in a candidate basis function. Both w_1 and w_2 default to five. If there are a large number of unimportant variables in the data, a small value of w_2 (relative to M) helps to make posterior sampling more efficient by not proposing basis functions that include the unimportant variables.

We extend the framework of Nott et al. (2005) to allow for more than two-way interactions. This ends up being non-trivial, since the RJMCMC acceptance ratio requires us to calculate the probability of sampling the proposed basis function. The difficulty comes when we try to calculate the probability of sampling the particular variables, as this requires calculating the probability of a weighted sample without replacement (weighted since we do not sample variables uniformly, without replacement since variables cannot be used more than once in the same basis function). This is equivalent to sampling from the multivariate Wallenius' noncentral hypergeometric distribution. To determine the probability of such a sample, we use a function from the `R` package `BiasedUrn` (Fog, 2015). Since the CRAN version of `BiasedUrn` allows for only 32 possible variables, we include a slightly altered version of the function in `BASS` to quickly evaluate the approximate density function of the multivariate Wallenius' noncentral hypergeometric distribution.

We perform N_{MCMC} RJMCMC iterations and discard the first N_{burn}, after which every N_{thin} iterations is kept. This results in $(N_{\text{MCMC}} - N_{\text{burn}})/N_{\text{thin}}$ posterior samples. Table 3.2 shows the parameters to the `bass` function discussed in this section, as well as their mathematical symbols.

Symbol	w_1	w_2	N_{MCMC}	N_{burn}	N_{thin}
`bass` input	`w1`	`w2`	`nmcmc`	`nburn`	`thin`

Table 3.2: Translation from mathematical symbols to parameters used to specify nominal weights of proposal distributions and number of RJMCMC iterations in the `bass` function.

3.2.2 Parallel tempering

We are able to achieve better mixing by using parallel tempering. This requires the specification of a temperature ladder, $1 = t_1 < t_2 < \cdots < t_T < \infty$. Only samples in the lowest temperature chain (t_1) are used for inference. We allow the chains to run without swapping for N_{st} iterations at the beginning of the run to allow them to get close to their stationary distributions.

Specifying a temperature ladder can be difficult. Temperatures need to be close enough to each other to allow for frequent swaps (with acceptance rates between 20 and 60% (Altekar et al., 2004)), and the highest temperature (t_T) needs to be high enough to be able to explore all the modes. Future versions of this package may make some attempt at automatically specifying and altering a temperature ladder.

Further, a message passing interface (MPI) approach to handling the multiple chains could result in substantial speedup, and may be implemented in future versions of the package.

Table 3.3 shows the translation from parameters used for parallel tempering in the `bass` function to symbols we have used in this section.

Symbol	$(t_1, \ldots, t_T)$	N_{st}
`bass` input	`temp.ladder`	`start.temper`

Table 3.3: Translation from mathematical symbols to parameters used for parallel tempering in the `bass` function.

3.2.3 Functional response

When the functional response is output onto the same functional variable grid for all samples, this results in more efficient calculations involving basis functions because of the Khatri-Rao product structure, as shown in Chapter 2. For example, this software is well suited to fit a model where the data are such that a combination of independent variables results in a time-series and the grid of times (say, $r_1, \ldots, r_q$) is the same for each combination.

If there are multiple functional variables, we must specify a maximum degree of interaction for them, $K^F_{\max}$. For instance, if the functional output was a spatio-temporal field (a function of three variables) and we specify a maximum degree of functional interaction of two, we would not allow for interactions between both spatial dimensions and time. We would specify the grid of spatial locations and time points as a matrix with three columns rather than a vector like we did in the time series example above. We can also specify a value b_F, possibly different from b, that indicates the number of non-zero values required in the functional part of basis functions. When functional responses are included, the values of b and b_F should be relative to the sample size and the size of the functional grid, respectively.

Table 3.4 shows parameters necessary to model functional responses in the `bass` function. The response `y` should be specified as a matrix when the response is functional.

Symbol	$(r_1, \ldots, r_q)$	$K^F_{\max}$	b_F
`bass` input	`xx.func`	`maxInt.func`	`npart.func`

Table 3.4: Translation from mathematical symbols to parameters used in the `bass` function when modeling functional data.

3.2.4 Categorical inputs

The ability to include categorical inputs in our framework will be crucial in the application presented in Chapter 4. Categorical variables are included by allowing for basis functions to have indicators for categorical variables being in certain categories. Our approach is the Bayesian version of Friedman (1991b) and is described further in Chapter 4. If a set of independent variables is separated into continuous variables $\mathbf{x}$ and categorical variables $\mathbf{c}$, then the m^{th} basis function equivalent of Equation 2.3 can be written as

$$B_m(\mathbf{x}, \mathbf{c}) = \prod_{k=1}^{K_m} g_{km}[s_{km}(x_{v_{km}} - t_{km})]_+^{\alpha} \prod_{l=1}^{K_m^c} 1\left(c_{v_{lm}^c} \in C_{lm}\right) \tag{3.6}$$

where K_m^c is the degree of interaction for the categorical predictors, $1(\cdot)$ is the indicator function, v_{lm}^c indexes the categorical variables and C_{lm} is a subset of the categories for variables $c_{v_{lm}^c}$. We now allow for K_m or K_m^c to be zero, and specify a $K^c_{\max}$ (`maxInt.cat` in the `bass` function).

The priors we use for the degree of interaction, variables used and categories used are, in combination with the priors we use above, the same constrained uniform.

Thus, basis function $(B_m(\mathbf{x}_1, \mathbf{c}_1), \ldots, B_m(\mathbf{x}_n, \mathbf{c}_n))$ is required to have at least b non-zero values.

3.3 Sensitivity analysis

We can obtain the Sobol' decomposition for each posterior sample to get posterior distributions of sensitivity indices. This can be time consuming, so the `sobol` function has an argument `mcmc.use` to specify which RJMCMC iterations should be used. Calculations of the integrals above can be vectorized when basis functions are the same and only basis function coefficients change. This is the case for many of the RJMCMC iterations, and the `sobol` function automatically determines this and accounts for it. (As a side note, this is also the case for the `predict` function).

3.3.1 Functional response

By default, the `sobol` function gets sensitivity indices for the functional variables the same way it does for the other variables. Setting `func.var = 1` gets the sensitivity indices as functions of the first (possibly only) functional variable (if there are multiple functional variables, this refers to the first column of the matrix `xx.func` passed to the `bass` function).

3.4 Examples

We now demonstrate the capabilities of the package on a few examples. For each example, we start by setting the seed (`set.seed(0)`) so that readers can replicate the results. First we load the package

```
R> library("BASS")
```

which we use for all the examples.

3.4.1 Curve fitting

We first demonstrate how the package can be used for curve fitting. We generate $y \sim N(f(x), 1)$ where $x \in [-5, 5]$ and

$$f(x) = \begin{cases} -0.1x^3 + 2\sin(\pi x^2)(x-4)^2 & 0 < x < 4 \\ -0.1x^3 & \text{otherwise} \end{cases} \tag{3.7}$$

for 1000 samples of x. The data are shown in Figure 3.3.

We generate the data with the following code.

```
R> set.seed(0)
R> f <- function(x) {
+   -.1 * x^3 + 2 * as.numeric((x < 4) * (x > 0)) * sin(pi * x^2) *
+        (x - 4)^2
+ }
R> sigma <- 1
R> n <- 1000
R> x <- runif(n, -5, 5)
R> y <- rnorm(n, f(x), sigma)
```

We then call the `bass` function to fit a BASS model using the default settings.

```
R> mod<-bass(x, y)
```

```
MCMC Start #-- Mar 14 21:28:51 --# nbasis: 0
MCMC iteration 1000 #-- Mar 14 21:28:53 --# nbasis: 33
MCMC iteration 2000 #-- Mar 14 21:28:55 --# nbasis: 33
MCMC iteration 3000 #-- Mar 14 21:28:57 --# nbasis: 32
MCMC iteration 4000 #-- Mar 14 21:28:59 --# nbasis: 34
MCMC iteration 5000 #-- Mar 14 21:29:02 --# nbasis: 31
MCMC iteration 6000 #-- Mar 14 21:29:04 --# nbasis: 39
MCMC iteration 7000 #-- Mar 14 21:29:06 --# nbasis: 31
```

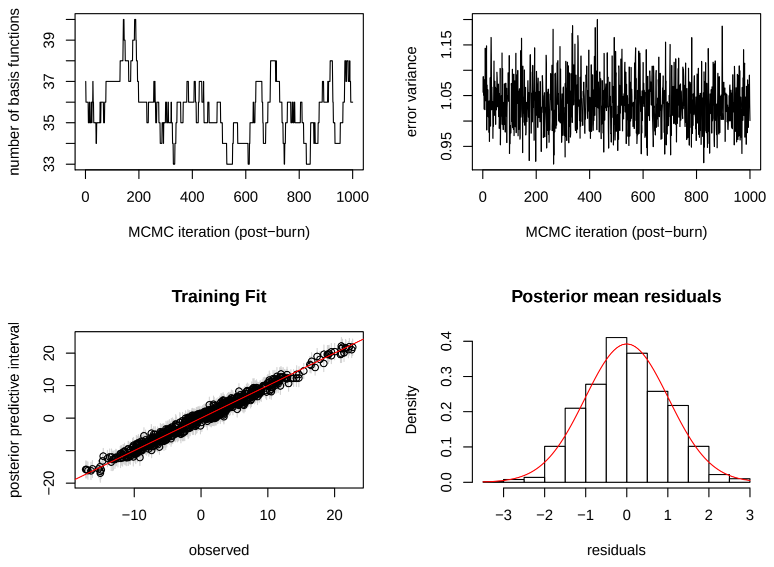


Figure 3.1: Diagnostic plots for BASS model fitting.

```
MCMC iteration 8000 #-- Mar 14 21:29:08 --# nbasis: 33
MCMC iteration 9000 #-- Mar 14 21:29:10 --# nbasis: 38
MCMC iteration 10000 #-- Mar 14 21:29:12 --# nbasis: 36
```

The result is an object that can be used for prediction and sensitivity analysis. By default, the bass function prints progress after each 1000 MCMC iterations, along with the number of basis functions. To diagnose the fit of the model, we call the plot function.

```
R> plot(mod)
```

This generates the four plots shown in Figure 3.1. The top left and right plots show trace plots (after burn-in and excluding thinned samples) of the number of basis functions (M) and the error variance (σ^2). The bottom left plot shows the response values plotted against the the posterior mean predictions (with equal tail posterior probability intervals as specified by the `quants` parameter). The bottom right plot shows a histogram of the posterior mean residuals along with the assumed Gaussian distribution centered at zero and with variance taken to be the posterior mean of

σ^2. This is for checking the Normality assumption.

Next, we can generate posterior predictions at new inputs, which we generate as `x.test`.

```
R> n.test <- 1000
R> x.test <- sort(runif(n.test, -5, 5))
R> pred <- predict(mod, x.test, verbose = T)
```

```
Predict Start #-- Mar 14 21:29:13 --# Models: 164
Predict #-- Mar 14 21:29:14 --# Model: 100
```

By default, the `predict` function generates posterior predictive distributions for all of the inputs. We can use a subset of posterior samples by specifying the parameter `mcmc.use`. For instance, `mcmc.use = 1` will use the first posterior sample (after burn-in and excluding thinned samples), and will thus be faster. Rather than iterating through the MCMC samples to generate predictions, we instead iterate through "models." The model changes when the basis functions change, which means that we can build the basis functions once and perform vectorized operations for predictions for all the MCMC iterations with the same basis functions.

The object resulting from the `predict` function is a matrix with rows corresponding to MCMC samples and columns corresponding to settings of `x.test`. Thus, the posterior mean predictions are obtained by taking the column means. We plot the posterior predictive means against the true values of $f(x)$ as shown in Figure 3.2.

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

Note that the predictive distributions in the columns of `pred` are for $f(x)$. To obtain predictive distributions for data, we would need to include Gaussian error

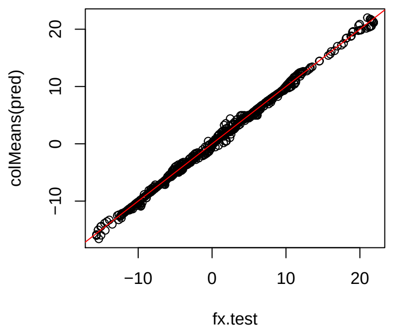


Figure 3.2: BASS prediction on test data.

with variance σ^2 (demonstrated in Section 3.4.5). Posterior samples of σ^2 are given in `mod$s2`.

In the curve fitting case, we can plot predicted curves. Below, we plot 10 posterior predictive samples along with the true curve (Figure 3.3). We also show knot locations (in the rug along the x-axis) for one of the posterior samples.

```
R> plot(x, y, cex = .5)
R> curve(f(x), add = T, lwd = 3, n = 1000, col = 2, lty = 2)
R> matplot(x.test, t(pred[seq(100, 1000, 100), ]),
+                                     type='l', add=T, col=3)
R> rug(BASS:::unscale.range(mod$curr.list[[1]]$knots.des, range(x)))
R> legend('topright', legend = c('true curve',
+                       'posterior predictive draws'), col = c(2:3),
+                        lty = c(2, 1), lwd = c(3, 1), bty = 'n')
```

If we are interested in using fewer knots (fewer basis functions), we can change the prior for the number of basis functions to be more restrictive. For instance, setting `h2=100`

```
R> mod <- bass(x, y, h2 = 100)

R> pred <- predict(mod, x.test)
R> plot(x, y, cex = .5)
R> curve(f(x), add = T, lwd = 3, n = 1000, col = 2, lty = 2)
R> matplot(x.test, t(pred[seq(100, 1000, 100), ]),
+                                 type='l', add=T, col=3)
R> rug(BASS:::unscale.range(mod$curr.list[[1]]$knots.des, range(x)))
```

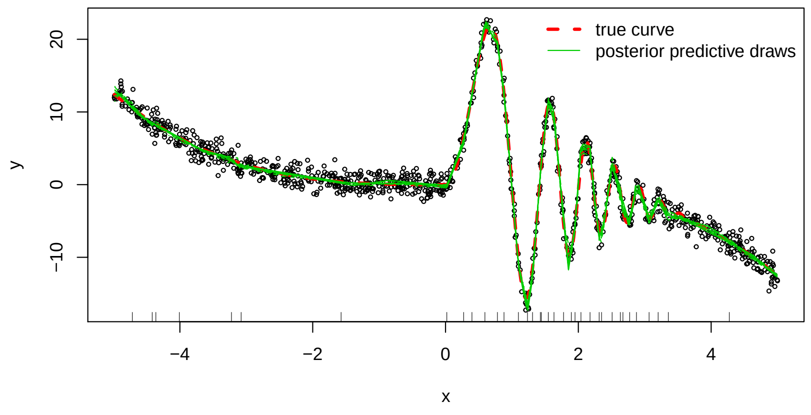


Figure 3.3: True curve with posterior predictive draws.

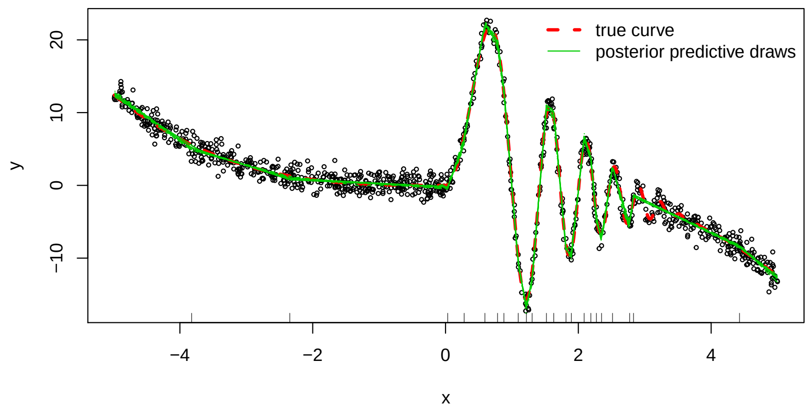


Figure 3.4: True curve with posterior predictive draws and more restrictive prior on the number of basis functions.

```
R> legend('topright', legend = c('true curve',
+                           'posterior predictive draws'), col = c(2:3),
+                            lty = c(2, 1), lwd = c(3, 1), bty = 'n')
```

results in knots as shown along the x-axis of Figure 3.4. This results in fewer knots, but perhaps slight underfitting in the part of the curve around $x = 3$. The `h2` parameter can be used to prevent overfitting, but the setting is not intuitive. Thus, this parameter may require tuning (perhaps by cross-validation).

Two final issues to discuss with this example are why we use linear splines (the default `degree = 1`) and how to tell if we have achieved convergence before taking MCMC samples as posterior samples. We use linear splines almost exclusively when using this package because of their stability and ability to capture nonlinear curves and surfaces. Using a higher degree, such as `degree = 3`, results in smoother models but suffers from stability problems and is more difficult to fit. We suggest settings of `degree` other than `degree = 1` be used with care, always with scrutiny of prediction performance. Convergence is best assessed by examining the trace plots shown in Figure 3.1. Especially if the trace plot for σ^2 shows any sort of non-cyclical pattern, the sampler should be run for longer. As a side note, a new sampler can be started from where the old sampler left off by using the `curr.list` parameter. For instance, we can run `mod2 <- bass(x, y, curr.list = mod$curr.list)` to start a new sampler from where `mod` left off.

3.4.2 Friedman function

For our next example, we will test the package on the Friedman function (Friedman, 1991c). This function will have 10 inputs, five of which contribute nothing. The other five are used to generate

$$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5. \tag{3.8}$$

We generate 200 input samples uniformly from a unit hypercube, calculate $f(x)$ for each and add standard Normal error to obtain data to model.

```
R> set.seed(0)
R> f <- function(x) {
+   10 * sin(pi * x[, 1] * x[, 2]) + 20 * (x[, 3] - .5)^2 +
+     10 * x[, 4] + 5 * x[, 5]
+ }
R> sigma <- 1
R> n.vars <- 10
R> n <- 200
```

```
R> x <- matrix(runif(n * n.vars), n, n.vars)
R> y <- rnorm(n, f(x), sigma)
```

Here we will show how we can change the length of the MCMC chain and use parallel tempering. We run the RJMCMC chain for 40000 iterations, discarding the first 30000 as burn-in and thinning by keeping every tenth sample. We supply a temperature ladder with smallest value one (the “cold chain”, or true posterior) and largest value 8.15 (the “hottest” chain) using geometric spacing. Thus, $t_i = (1 + \Delta_t)^{i-1}$ where Δ_t is a spacing parameter we set at 0.3. We use nine chains. By default, chains at neighboring temperatures will be allowed to swap after the first 1000 iterations.

```
R> mod <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+              temp.ladder = (1 + .27)^(1:9 - 1), verbose = F)
```

We can generate posterior predictive samples just as we did in the curve fitting example.

```
R> n.test <- 1000
R> x.test <- matrix(runif(n.test * n.vars), n.test)
R> pred <- predict(mod, x.test, verbose = T)

Predict Start #-- Mar 14 23:08:16 --# Models: 876
Predict #-- Mar 14 23:08:17 --# Model: 100
Predict #-- Mar 14 23:08:17 --# Model: 200
Predict #-- Mar 14 23:08:17 --# Model: 300
Predict #-- Mar 14 23:08:17 --# Model: 400
Predict #-- Mar 14 23:08:17 --# Model: 500
Predict #-- Mar 14 23:08:17 --# Model: 600
Predict #-- Mar 14 23:08:17 --# Model: 700
Predict #-- Mar 14 23:08:17 --# Model: 800
```

Plotting these samples against true values of $f(x)$ shows that we have a good fit (Figure 3.5).

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

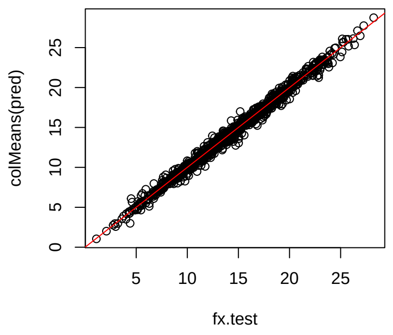


Figure 3.5: BASS prediction on test data - Friedman function.

Now that we are considering a function of many variables, we may be interested in sensitivity analysis. To get the Sobol' decompostion for each posterior sample, we use the `sobol` function.

```
R> sens <- sobol(mod, verbose = F)
```

Note that when `verbose = T`, this function prints after every 10 models (as with the `predict` function, vectorizing around models rather than MCMC iterations saves a large amount of time). Depending on the number of basis functions and the number of models, this function can take significant amounts of time. If that is the case, using a smaller set of MCMC iterations by specifying `mcmc.use` may be useful.

The default plotting for this kind of object (Figure 3.6) shows boxplots of variance explained for each main effect and interaction that shows up in the BASS model. It also shows boxplots of the total sensitivity indices.

```
R> plot(sens, cex.axis = .5)
```

If there are a large number of main effects or interactions that explain very small percentages of variation, we can show only the effects that are most significant. For instance, we could show only the effects that, on average, explain at least 1% of the variance (Figure 3.7).

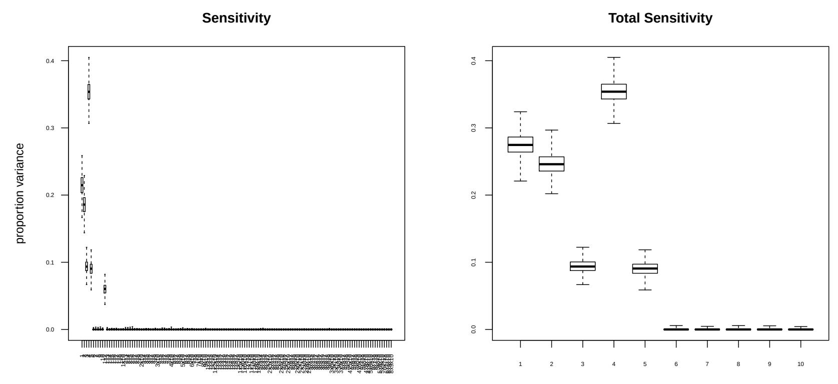


Figure 3.6: BASS sensitivity analysis - Friedman function.

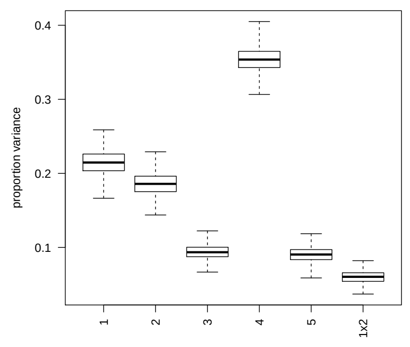


Figure 3.7: Most important main effects and interactions - Friedman function.

```
R> boxplot(sens$S[, colMeans(sens$S) > .01], las = 2,
+          ylab = 'proportion variance', range = 0)
```

As expected, we see that almost all of the variance is from the first five variables and the only strong interaction is between the first two variables.

As a final note for this example, we discuss tempering diagnostics. We would like for neighboring chains to have swap acceptance rate of somewhere around 23%. Running `bass` with `verbose = T` prints these acceptance rates every 1000 iterations. At the completion of the sampling, we can investigate acceptance rates by dividing the swap counts by the number of swap proposals, as follows.

```
R> mod$count.swap/mod$count.swap.prop
```

```
[1] 0.3621554 0.3793670 0.3779575 0.2840164 0.2185430 0.3181351
[7] 0.2764826 0.2377282
```

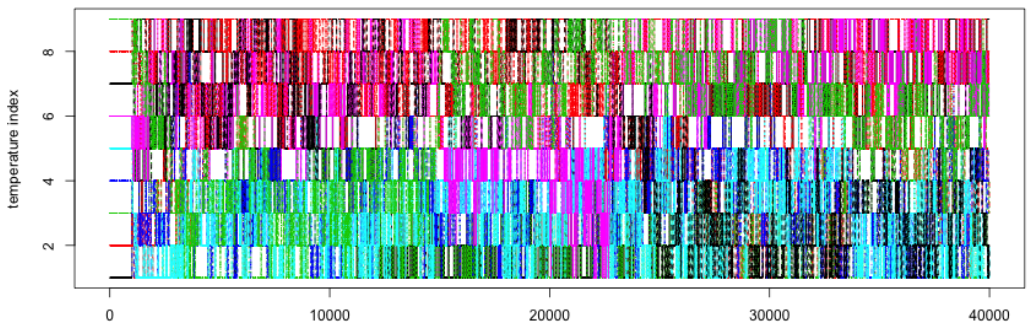


Figure 3.8: Parallel tempering diagnostics - swap trace plot.

Since we have specified nine temperatures, there are eight possible swaps, hence the eight numbers. If, for example, we wanted to increase the first acceptance rate, we would move the second temperature closer to the first.

Further analysis of swaping can be done by looking at swap trace plots.

```
R> matplot(mod$temp.val, type = 'l', ylab = 'temperature index')
```

Figure 3.8 shows the swap trace plot where y-axis values are temperature indices (1 is the true posterior and 9 is the posterior raised to the smallest power), the x-axis shows MCMC iteration and the colored lines represent the different chains. We want to see these chains mixing throughout, as we do here.

Determining whether the smallest value of the temperature ladder is small enough to allow for good mixing can be difficult. In this example, we could run the model with `temp.ladder = 8.15` and look at mixing diagnostics. One could also look at predicted versus observed plots at the different temperatures for the last MCMC iteration by executing the following code, the output of which is shown in Figure 3.9.

```
R> par(mfrow=c(3,3))
R> temp.ind <- sapply(mod$curr.list, function(x) x$temp.ind)
R> for(i in 1:length(mod$temp.ladder)) {
+   ind <- which(temp.ind == i)
```

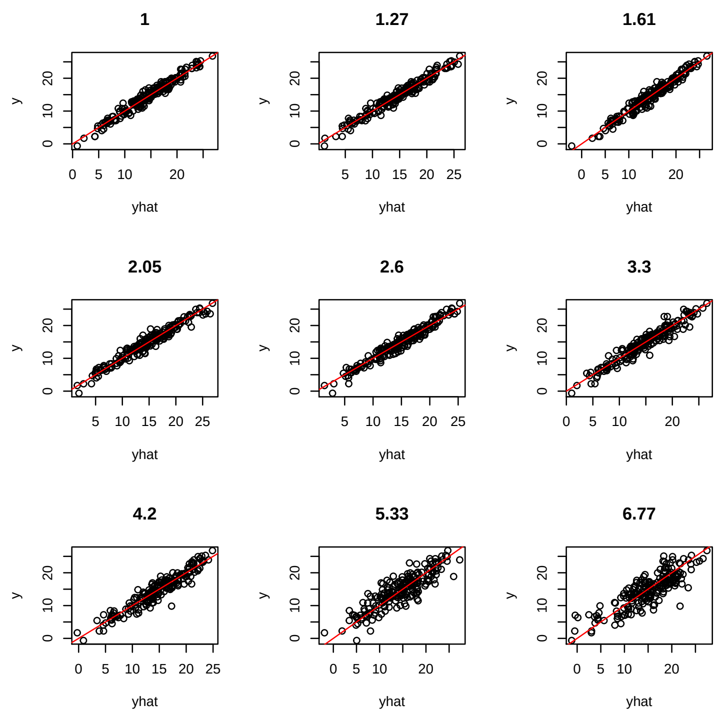


Figure 3.9: Predicted versus observed for the last MCMC iteration of the nine chains at different temperatures. The temperatures are shown above each plot.

```
+   yhat <- mod$curr.list[[ind]]$des.basis %*%
+             mod$curr.list[[ind]]$beta
+   plot(yhat, y, main = round(mod$temp.ladder[i], 2))
+   abline(a = 0, b = 1, col = 2)
+ }
```

Note that the `curr.list` object is a list with number of elements equal to the number of temperatures. This list contains the MCMC state for each chain. Since we swap temperatures rather than entire states, the chains are not in order according to temperature. We note that using the default prior for σ^2 with a temperature ladder with relatively large values can lead to instabilities when estimating σ^2. In cases where that is clearly the case, the prior for σ^2 will be automatically changed and a warning will be generated.

To demonstrate what is different when we use tempering, consider the equivalent BASS model fit without tempering.

```
R> mod.noTemp <- bass(x, y, nmcmc = 40000, nburn = 30000,
+                     thin = 10, verbose = F)
```

We compare the root mean square prediction error (RMSE) for the two models, as well as the empirical coverage of 95% probability intervals. First, the RMSE for the model fit without tempering

```
R> pred.noTemp <- predict(mod.noTemp, x.test)
R> sqrt(mean((colMeans(pred.noTemp) - fx.test)^2))

[1] 0.5303968
```

and the empirical coverage

```
R> quants.noTemp <- apply(pred.noTemp, 2, quantile,
+                         probs = c(.025, .975))
R> mean((quants.noTemp[1, ] < fx.test) &
+      (quants.noTemp[2, ] > fx.test))

[1] 0.923
```

demonstrate that the fit is quite good. When we use parallel tempering, the RMSE

```
R> sqrt(mean((colMeans(pred) - fx.test)^2))

[1] 0.4769708
```

and the empirical coverage

```
R> quants <- apply(pred, 2, quantile, probs = c(.025, .975))
R> mean((quants[1, ] < fx.test) & (quants[2, ] > fx.test))

[1] 0.949
```

are better, though not by an extreme amount. Under different seeds, we tend to see higher coverage when we use tempering and lower coverage when we do not. We also tend to get better models in terms of RMSE when we use tempering. Other benefits of tempering will be shown in later examples. Because the computational burden is currently linear in the number of temperatures, using fewer temperatures is better. Thus, for many purposes, the model without tempering may be good enough.

3.4.3 Friedman function with a categorical variable

In this example, we use data generated from a function similar to the Friedman function in the previous example but with a categorical variable included. The function, introduced in Gramacy & Taddy (2010), has

$$f(\mathbf{x}) = \begin{cases} 10\sin(\pi x_1 x_2) & x_{11} = 1 \\ 20(x_3 - 0.5)^2 & x_{11} = 2 \\ 10x_4 + 5x_5 & x_{11} = 3 \\ 5x_1 + 10x_2 + 20(x_3 - 0.5)^2 + 10\sin(\pi x_4 x_5) & x_{11} = 4 \end{cases} \tag{3.9}$$

as the mean function and standard Normal error. Again, $x_6, \ldots, x_{10}$ are unimportant. We generate 500 random uniform samples of the first 10 variables and randomly sample 500 values of the four categories of the 11th variable. The `bass` function treats input variables as categorical only if they are coded as factors.

```
R> set.seed(0)
R> f <- function(x) {
+   as.numeric(x[, 11] == 1) * (10 * sin(pi * x[, 1] * x[, 2])) +
+   as.numeric(x[ ,11] == 2) * (20 * (x[, 3] - .5)^2) +
+   as.numeric(x[, 11] == 3) * (10 * x[, 4] + 5 * x[, 5]) +
+   as.numeric(x[, 11] == 4) * (10 * sin(pi * x[, 5] * x[, 4]) +
+                    20 * (x[, 3] - .5)^2 + 10 * x[, 2] + 5 * x[, 1])
+ }
R> sigma <- 1
R> n <- 500
R> x <- data.frame(matrix(runif(n * 10), n, 10),
+                 as.factor(sample(1:4, size = n, replace = T)))
R> y <- rnorm(n, f(x), sigma)
```

We fit a model with tempering and use it for prediction, as in the previous example.

```
R> mod <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+             temp.ladder = (1 + .2)^(1:6 - 1), verbose = F)
```

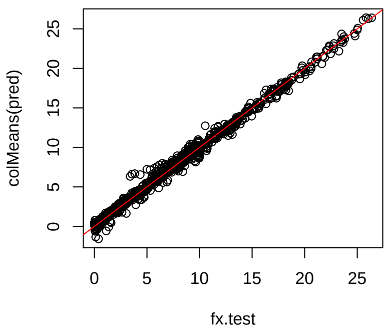


Figure 3.10: BASS prediction on test data - Friedman function with categorical predictor.

```
R> n.test <- 1000
R> x.test <- data.frame(matrix(runif(n.test * 10), n.test, 10),
+                  as.factor(sample(1:4, size = n.test, replace = T)))
R> pred <- predict(mod, x.test)
```

Plotting posterior predictive samples against true values of $f(x)$ shows that we have a good fit (Figure 3.10).

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

Sensitivity analysis is performed in the same manner.

```
R> sens <- sobol(mod)
```

Plotting the posterior distributions of the most important (explaining more than 0.5% of the variance) sensitivity indices in Figure 3.11, we see how important the categorical variable is as well as which variables it interacts with.

```
R> boxplot(sens$S[, colMeans(sens$S) > .005], las = 2,
+         ylab = 'proportion variance', range = 0)
```

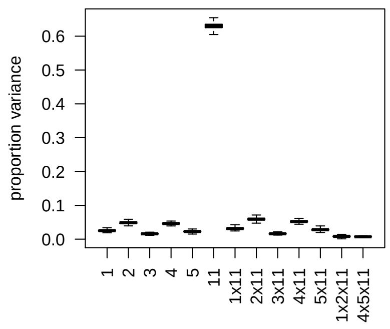


Figure 3.11: Most important main effects and interactions - Friedman function with categorical predictor.

3.4.4 Friedman function with functional response

Next, we consider an extension of the Friedman function that is functional in one variable, as in Chapter 2. We use

$$f(\mathbf{x}) = 10\sin(2\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 \tag{3.10}$$

where we treat x_1 as the functional variable. Note that we insert a factor of two into the sin function in order to increase the variability due to x_1, making the problem more challenging. We generate 500 combinations of $x_2, \ldots, x_{10}$ from a uniform hypercube. We generate a grid of values of x_1 of length 50. This ends up being 500×50 combinations of inputs, for which we evaluate f and add standard Normal error. We keep the responses in a matrix of dimension 500×50 so that each row represents a curve. The inputs are kept separate in a 500×9 matrix and a grid of length 50.

```
R> set.seed(0)
R> f<-function(x) {
+   10 * sin(2 * pi * x[, 1] * x[, 2]) + 20 * (x[, 3] - .5)^2 +
+     10 * x[, 4] + 5 * x[, 5]
+ }
R> sigma <- 1
```

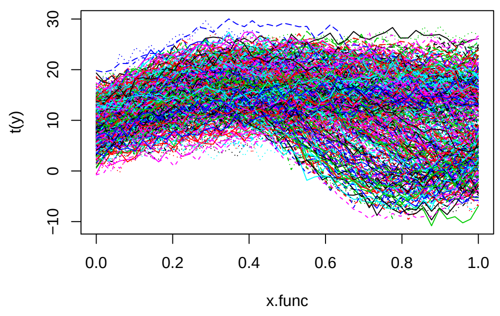


Figure 3.12: 500 Functional responses. The goal is to fit a functional nonparametric regression model and perform sensitivity analysis.

```
R> n <- 500
R> n.func <- 50
R> x.func <- seq(0, 1, length.out = n.func)
R> x <- matrix(runif(n * 9), n)
R> y <- matrix(f(cbind(rep(x.func, each = n),
+                       kronecker(rep(1, n.func), x))),
+              ncol = n.func) + rnorm(n * n.func, 0, sigma)
```

The functional data can be plotted as follows and are shown in Figure 3.12.

```
R> matplot(x.func, t(y), type='l')
```

In order for the `BASS` package to handle functional responses, each curve needs to be evaluated on the same grid. Thus, the responses must be able to be stored as a matrix without missing values.

We fit the model by specifying our matrices `x` and `y` as well as the grid `x.func`.

```
R> mod <- bass(x, y, xx.func = x.func)

MCMC Start #-- Mar 14 21:37:48 --# nbasis: 0
MCMC iteration 1000 #-- Mar 14 21:37:50 --# nbasis: 110
MCMC iteration 2000 #-- Mar 14 21:37:54 --# nbasis: 176
MCMC iteration 3000 #-- Mar 14 21:37:59 --# nbasis: 173
MCMC iteration 4000 #-- Mar 14 21:38:03 --# nbasis: 115
MCMC iteration 5000 #-- Mar 14 21:38:05 --# nbasis: 74
MCMC iteration 6000 #-- Mar 14 21:38:06 --# nbasis: 60
MCMC iteration 7000 #-- Mar 14 21:38:08 --# nbasis: 60
```

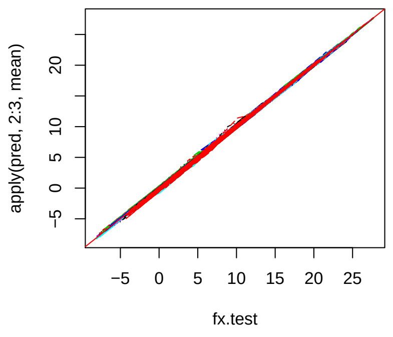


Figure 3.13: BASS prediction performance - Friedman function with functional response.

```
MCMC iteration 8000 #-- Mar 14 21:38:10 --# nbasis: 62
MCMC iteration 9000 #-- Mar 14 21:38:11 --# nbasis: 71
MCMC iteration 10000 #-- Mar 14 21:38:17 --# nbasis: 62
```

Prediction is as simple as before. If we want to predict on a different functional grid, we can specify that in the `predict` function with `newdata.func`.

```
R> n.test <- 100
R> x.test <- matrix(runif(n.test * 9), n.test)
R> pred <- predict(mod, x.test)
```

Following, we make a functional predicted versus observed plot, shown in Figure 3.13.

```
R> fx.test<-matrix(f(cbind(rep(x.func, each = n.test),
+                 kronecker(rep(1, n.func), x.test))), ncol=n.func)
R> matplot(fx.test, apply(pred, 2:3, mean), type = 'l')
R> abline(a = 0, b = 1, col = 2)
```

We will demonstrate the two methods of sensitivity analysis discussed in Section 3.3. First, we can get the Sobol' indices for the functional variable and its interactions just as we do the other variables. This is the default.

```
R> sens <- sobol(mod, mcmc.use = 1:100)
```

```
Sobol Start #-- Mar 14 21:38:31 --# Models: 25
Sobol #-- Mar 14 21:38:44 --# Model: 10
Sobol #-- Mar 14 21:38:57 --# Model: 20
Total Sensitivity #-- Mar 14 21:39:05 --#
```

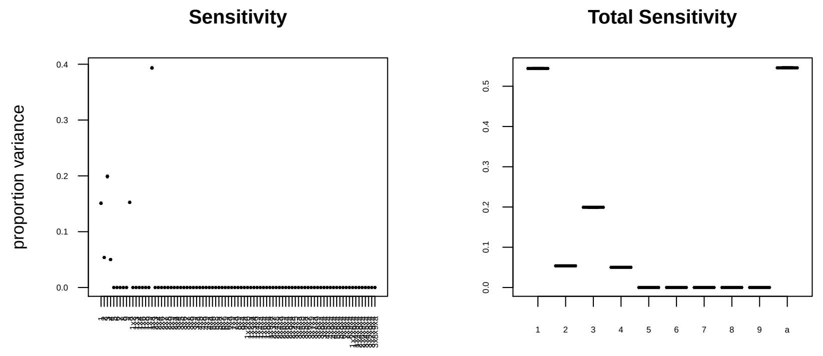


Figure 3.14: Sensitivity analysis - Friedman function with functional response.

When we plot the variance decomposition, as shown in Figure 3.14, the functional variable is labeled with the letter "a." If we had multiple functional variables, they would be labeled with different letters.

```
R> plot(sens, cex.axis = .5)
```

The other approach to sensitivity analysis is to get a functional variance decomposition. This is done by using the `func.var` parameter. If there is only one functional variable, we set `func.var = 1`. Otherwise we set `func.var` to the column of `xx.func` we want to use for our functional variance decomposition. This will be explained in more detail in a later example.

```
R> sens.func <- sobol(mod, mcmc.use = 1:100, func.var = 1)

Sobol Start #-- Mar 14 21:39:05 --# Models: 25
Sobol #-- Mar 14 21:39:16 --# Model: 10
Sobol #-- Mar 14 21:39:27 --# Model: 20
```

When we plot the variance decomposition, shown in Figure 3.15, we we get two plots.

```
R> plot(sens.func)
```

The left plot shows the posterior mean (using posterior samples specified with `mcmc.use`) of the functional sensitivity indices in a functional pie chart. The right

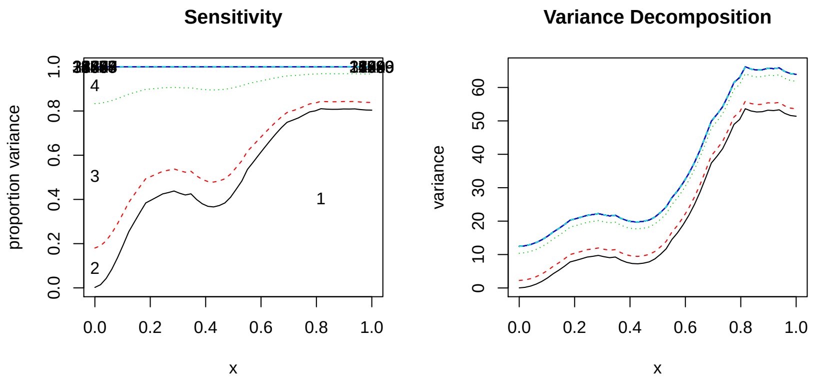


Figure 3.15: Functional sensitivity analysis - Friedman function with functional response.

plot shows the variance decomposition as a function of the functional variable. Thus, the top line in the right plot is the total variance in y as a function of x_1. The bottom line (black) is the total variance explained by the main effect of x_2 as a function of x_1. The labels in the plot on the left are the variable numbers (columns of `x`).

3.4.5 Air foil data

In this example, we consider a NASA data set, obtained from a series of aerodynamic and acoustic tests of two and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel (Lichman, 2013). The response is scaled sound pressure level, in decibels. There are five inputs: (1) Frequency, in Hertzs; (2) angle of attack, in degrees; (3) chord length, in meters; (4) free-stream velocity, in meters per second; and (5) suction side displacement thickness, in meters. The data have 1503 combinations of these inputs, some of which are collinear (variables 2 and 5 have correlation of 0.75).

```
R> dd <- read.table('https://archive.ics.uci.edu/ml/
+    machine-learning-databases/00291/airfoil_self_noise.dat')
```

We set aside 200 input combinations to use for testing.

```
R> set.seed(0)
```

```
R> test <- sample(nrow(dd), size=150)
R> x <- dd[-test, 1:5]
R> y <- dd[-test, 6]
```

We fit a BASS model using tempering.

```
R> mod <- bass(x, y, nmcmc = 20000, nburn = 10000, thin = 10,
+              temp.ladder = 1.1^(0:5), verbose = F)
```

We can predict as we have before. However, this prediction is for the mean function.

```
R> x.test <- dd[test, 1:5]
R> y.test <- dd[test, 6]
R> pred <- predict(mod, x.test)
```

Now, if we are interested in predicting actual data rather than the mean function, we can incorporate uncertainty from our estimate of σ^2. The vector `mult` below is the multiplyer we would use to get 95% prediction intervals.

```
R> mult <- 1.96 * sqrt(mod$s2)
R> q1 <- apply(pred - mult, 2, quantile, probs = .025)
R> q2 <- apply(pred + mult, 2, quantile, probs = .975)
R> mean((q1 < y.test) & (q2 > y.test))

[1] 0.9466667
```

This puts our empirical coverage where we would expect it to be. We can plot our 95% prediction intervals as follows, shown in Figure 3.16.

```
R> plot(y.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
R> segments(y.test, q1, y.test, q2, col = 'lightgrey')
```

Next, we can obtain and plot the Sobol' decomposition, shown in Figure 3.17.

```
R> sens <- sobol(mod, verbose = F)
R> plot(sens)
```

The uncertainty in the sensitivity indices in Figure 3.17 is significant and helps us to understand that there are many possible models for these data that use

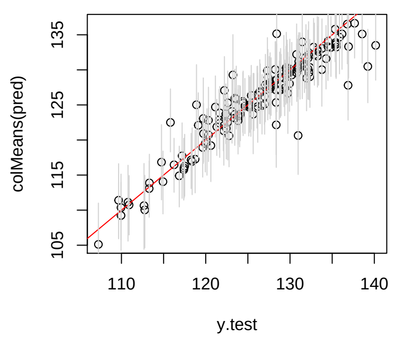


Figure 3.16: Prediction performance - air foil data.

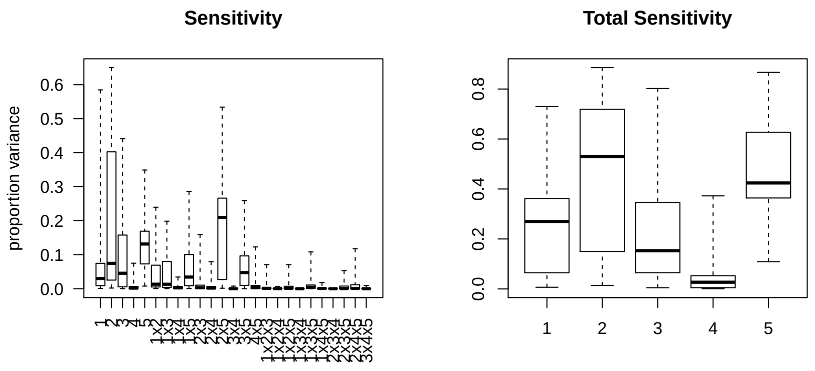


Figure 3.17: Sobol decomposition - air foil data.

different variables and interactions. The proper characterization of this uncertainty would be impossible if our RJMCMC chain was stuck in a mode. Hence, tempering is crucial in this problem. By exploring the posterior modes, tempering allows us to find not just a model that predicts well, but all the models that predict well.

3.4.6 Pollutant spill model

The final example we present is an emulation problem. The simulator is for modeling a pollutant spill caused by a chemical accident, obtained from Surjanovic & Bingham (2017). While fast to evaluate, this simulator provides a good testbed for BASS methods. The simulator has four inputs: (1) Mass of pollutant spilled at each of two locations (range $7-13$), (2) diffusion rate in the channel ($0.02-0.12$), (3) location of the second spill ($0.01-3$), and (4) time of the second spill ($30.01-30.295$). The simulator outputs a function in space (one dimension) and time that is the concentration of the pollutant.

We generate 10000 combinations of the four simulator inputs uniformly from within their respective ranges.

```
R> set.seed(0)
R> n <- 10000
R> x <- cbind(runif(n, 7, 13), runif(n, .02, .12), runif(n, .01, 3),
+             runif(n, 30.01, 30.295))
```

We specify six points in space and 20 time points. The functional grid we will pass to the `bass` function will thus have two columns, called `x.func` below.

```
R> s <- c(0, 0.5, 1, 1.5, 2, 2.5)
R> t <- seq(.3, 60, length.out = 20)
R> x.func <- expand.grid(t, s)
```

We use the `environ` function available from `http://www.sfu.ca/~ssurjano/Code/environr.html` to generate realizations of the simulator. We will model the log of the simulator output.

```
R> out <- t(apply(x, 1, environ, s = s, t = t))
R> y <- log(out + .01)
```

With this amount of data, we are presented with an extremely large model space to search through. In addition, since the data are smooth (no random noise), BASS models will tend to allocate a very large number of basis functions to try to capture the smoothness. In order to compensate for the large model space and the smoothness, we need to set an extreme prior on the number of basis functions to have a managable model. We do this by increasing `h2` by many orders of magnitude. In this example, we set `h2 = 1e250`. This results in a prior for the number of basis functions with very heavy weight near zero. Because of the large amount of data, we still get hundreds of basis functions.

Using such an extreme prior makes our multimodal posterior more peaked, and more difficult to explore. We may need hundreds of chains running at different temperatures in order to get the temperatures close enough to eachother to allow for frequent swapping. Another possibility that does not require picking a temperature ladder is to instead run multiple cold chains (at the true posterior) and allow them to swap states. This is a version of parallel hierarchical sampling introduced in Rigat & Mira (2012) that can be easily implemented by setting `temp.ladder = rep(1, n.chains)`.

```
R> mod <- bass(x, y, xx.func = x.func, nmcmc = 110000, nburn = 100000,
+       thin = 10, h2 = 1e250, save.yhat = F, temp.ladder = rep(1, 10),
+       npart.func = 1, verbose = F, maxBasis = 175)
```

Note that we specify `save.yhat = F`. By default, the `bass` function saves in-sample predictions for all MCMC samples (post burn-in and thinned). This can be a significant storage burden when we have large amounts of functional data, as we do in this

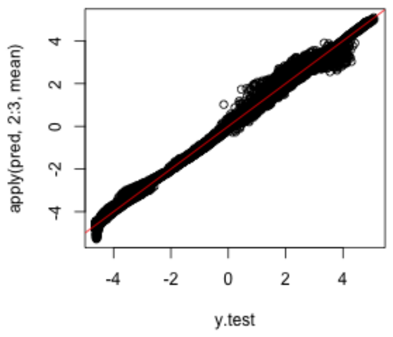


Figure 3.18: BASS prediction performance - pollutant spill model.

case. Changing the `save.yhat` parameter can relieve this. If in-sample predictions are of interest, they can be obtained after model fitting using the `predict` function.

As with the previous example, prediction here is for the mean function. Whatever error is left over (in σ^2) is inability of the BASS model to pick up high frequency signal.

```
R> n.test <- 1000
R> x.test <- cbind(runif(n.test, 7, 13),runif(n.test, .02, .12),
+                  runif(n.test, .01, 3), runif(n.test, 30.01, 30.295))
R> y.test <- log(t(apply(x.test, 1, environ, s = s, t = t)) + .01)
R> pred <- predict(mod, x.test)
```

A plot of the predicted (mean function) versus observed data is shown in Figure 3.18.

```
R> plot(y.test, apply(pred, 2:3, mean))
R> abline(a = 0, b = 1, col = 2)
```

To see what the predictions look like in space and time, consider the plots shown in Figure 3.19. These show posterior draws (in grey) of the mean function for one setting of the four inputs along with simulator output (in red).

```
R> pp <- pred[, 1, ]
R> ylim <- range(y)
R> par(mfrow=c(2, 3))
R> for(i in 1:length(s)) {
```

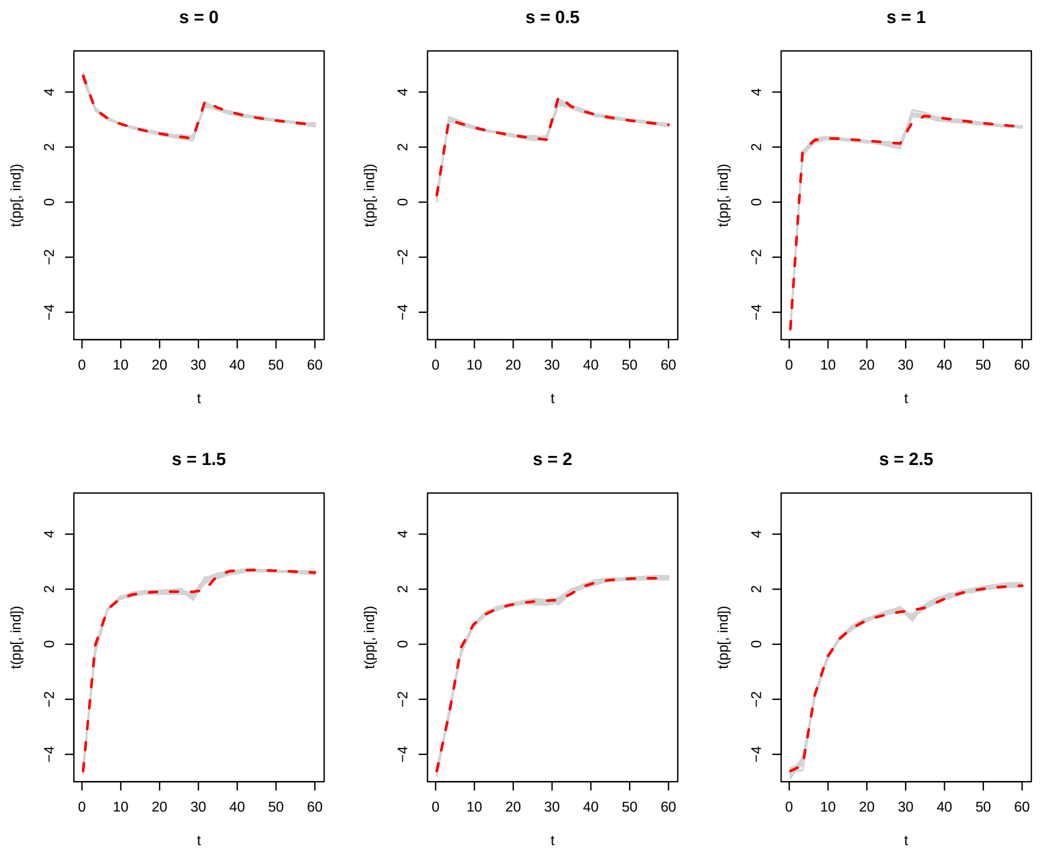


Figure 3.19: BASS prediction in space and time - pollutant spill model.

```
+   ind <- length(t) * (i - 1) + 1:length(t)
+   matplot(t, t(pp[, ind]), type = 'l', col = 'lightgrey',
+           ylim = ylim, main = paste('s =', s[i]))
+   lines(t, y.test[1, ind], col = 2, lwd = 2, lty = 2)
+ }
```

We can use the sensitivity analysis methods above, but we can get Sobol' indices as a function of either space or time. Below, we show how to get them as a function of time. We limit the models considered using `mcmc.use` to speed up computations. Since we have two functional inputs, we have two letters that can be included in these sensitivity plots (functional inputs are labeled with letters). Note that variable four is not included. This is because it did not explain any variance.

```
R> sens.func1 <- sobol(mod, mcmc.use = 1, func.var = 1,
+                      xx.func.var = t, verbose = F)
R> plot(sens.func1)
```

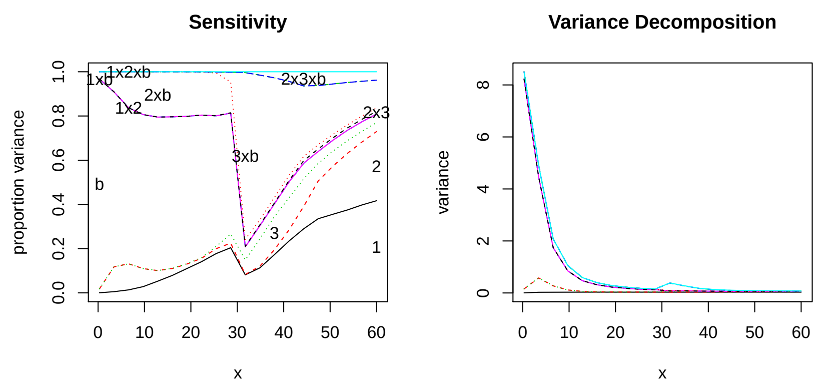


Figure 3.20: Sensitivity indices as a function of time - pollutant spill model.

3.5 Summary

Our proposed BASS framework provides a powerful general tool for nonparametric regression settings. It can be used for modeling with many continuous and categorical inputs, large sample size and functional response. It provides posterior sensitivity analyses without integration error. The MCMC approach to inference, especially using parallel tempering, yields posterior samples that can be used for probabilistic prediction. The `BASS` package makes these features accessible to users with minimal exposure. These capabilities have been demostrated with a set of examples involving different dimensions, categorical variables, functional responses, and large datasets.

Chapter 4

Inferring Atmospheric Release Characteristics in a Large Computer Experiment using Bayesian Adaptive Splines

Less than one year after California's Diablo Canyon Nuclear Power Plant became operational in 1985, a catastrophic nuclear accident occurred in Chernobyl, then part of the Soviet Union, resulting in large amounts of radioactive material being released into the atmosphere and proving fatal for many emergency responders and reactor staff. Radioactive plumes, which drifted over much of the western Soviet Union and Europe, necessitated the evacuation and long-term resettlement of many local people and have had lasting effects on public health (United Nations Scientific Committee on the Effects of Atomic Radiation, 2008).

Pacific Gas and Electric Company, which operates the Diablo Canyon Nuclear Power Plant, performed a series of experimental releases of sulfur hexafluoride, a benign gas, from the California plant soon after the 1986 Chernobyl accident. The purpose of these experiments was to gather concentration data at downwind observation sites after each release to be used for validation of established particle dispersion models in the presence of complex terrain (Thuillier, 1992). The inverse problem, to determine if the release characteristics can be inferred based on observations of the plume, was of less interest. However, the latest large-scale radioactive release that happened after an accident in Fukushima, Japan, has prompted renewed interest in the experimental release data from 1986 as a testbed for particle dispersion forward and inverse modeling (Lucas et al., 2017). Highly complex systems of this nature rarely allow for well controlled experiments, making the 1986 experimental release data a rare asset.

Particle dispersion models have improved in the past three decades, and statistical calibration of computer models has become a well developed field, especially following the work of Kennedy & O'Hagan (2001). Our interest lies in developing non-intrusive uncertainty quantification methods, specifically emulation and inverse modeling methods, suitable for use with the 1986 atmospheric release observations and many evaluations of a state-of-the-art particle dispersion model. This is a difficult task due to the complexity of both the observed and simulated data. The experimental release observations form a spatio-temporal field that includes measurement error, background noise and missing values. Each of our many evaluations of the simulator has both continuous and categorical inputs and outputs a spatio-temporal field, resulting in massive amounts of simulated data. We note that, in general, uncertainty quantification has become an essential step in much of modern scientific exploration, and many fields are consistently increasing their computational capacity, making analysis of large amounts of simulated data a relevant topic.

The primary innovation in this work that can be extended to other large computer experiments is our emulation methodology. We avoid the GP because we employ a large number of evaluations of the simulator, each with spatio-temporal output. As we have discussed, scalability is not a strength of the GP, though more scalable versions are introduced in Kaufman et al. (2011) and Gramacy & Apley (2015). Instead, we again explore BMARS. While BMARS can be used on its own for functional emulation as in Chapter 2, we opt for a different approach in this application. We use BMARS to model weights on spatio-temporal empirical orthogonal functions (EOFs) that are linearly combined to yield a spatio-temporal emulator. This has commonalities with the approach of Higdon et al. (2008), which emulates simulators with highly multivariate output by using GPs to model weights when linearly combining principle components. In our application, we also require an emulator that can handle categorical inputs. We develop a way for BMARS to incorporate categorical inputs that is true to its adaptive nature. The result is an emulator that (1) can flexibly and accurately model complicated functional response surfaces; (2) quantifies emulator uncertainty; and (3) is scalable in the number of inputs, the number of model runs and the dimension of the functional response. This is all fairly easy to reproduce via the R package BASS (Francom, 2017), introduced in Chapter 3.

Inverse modeling, or calibration, for this particular dataset presents its own set of unique challenges. In addition to emulation for simulators that are expensive to evaluate, other essential parts of calibration are building the simulator discrepancy model and the observational error model. We combine these models with a modularized Bayesian approach similar to that of Liu et al. (2009) to ensure identifiability and to simplify computations. We use a BMARS model for the discrepancy. The scalability and flexibility of the combined framework could be valuable for other computer experiments with large amounts of observational and simulation data.

We introduce our data, methods and findings as follows. In Section 4.1, we introduce the experimental release data and our simulations. In Section 4.2, we detail the construction of our emulator and evaluate the fit. In Section 4.3, we describe our calibration framework, detailing simulator discrepancy and observational error models. We also demonstrate the inversion capability of the framework on synthetic data. In Section 4.4, we show the results of our calibration technique using the Diablo Canyon plume observations and discuss the accuracy. In Section 4.5, we summarize and detail future work.

4.1 Data

As is common in computer experiments, we have two sources of data: observations and simulations (evaluations of the simulator). In this section, we describe each of these.

4.1.1 Observations

The 1986 experimental release we analyze is one of eight releases performed on different days between August 31 and September 17. We focus on a release of 146 kilograms of sulfur hexafluoride (SF_6) on September 4. Starting at 8:00AM local time, the SF_6 gas was released continuously for eight hours from a location at the base of the southmost containment unit at the Diablo Canyon Nuclear Power Plant.

Air sampling was performed automatically at 150 sites in the surrounding area. At each site, air was pumped into a tedlar bag over the course of one hour, at which point the bag was sealed and air was pumped into a new bag. Sampling was done from the hours of 7:00AM to 7:00PM, yielding 12 measurements at each site. The quantity of interest, the concentration of SF_6, is reported as an hourly average for each site. Roughly 24% of the samples are missing for unknown reasons.

The first sample at each site (the 7-8am average) was taken prior to the beginning of the experimental release, thus measuring the background level of SF_6. Investigation of the background level shows moderate amounts near the plant switchyard, where SF_6 is used for electrical insulation. A map of the background level is shown in Figure 4.1, along with the time series of the period after the release for a few sites. The time series are noisy, show orders of magnitude in variation (even in the background level) and missing values. The large background level at sites near the switchyard complicate the task of determining the controlled release characteristics based on our plume observations, since this fugitive release plays a confounding role. To minimize this confounding, for the 12 measurements at each site we subtract the site background level (the first measurement). Any negative values are truncated to zero. This is an effort to remove the background, but it makes the assumptions that the background level (1) is constant over time, (2) is small enough that it has negligible effect on down-wind measurements (i.e., it mostly disperses before reaching other sites) and (3) is not subject to large measurement error. The latter two assumptions are likely to be valid, but the first is difficult to justify. Still, in the absence of unconfounded temporal data to characterize the switchyard release, we proceed under these assumptions. For sites that are missing a background measurement (shown with slightly smaller dots in Figure 4.1), we do not subtract a background value from the measurements. While we could try to impute the background level from neighboring sites, we assume that the background level does not exhibit strong spatial correlation. Further, there are no missing background readings that we believe would be significantly large (i.e., there are no missing values very near the switchyard). Because of the many orders of magnitude difference in observations, small background errors are unlikely to have much effect on our ability to identify release characteristics. Even the largest background measurements are orders of magnitude smaller than the largest post-release measurements later in the time series.

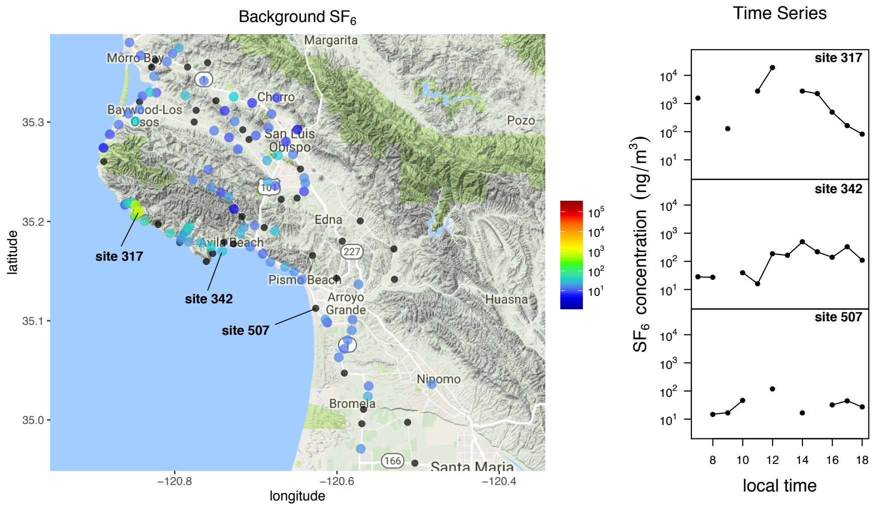


Figure 4.1: On the left, background SF$_6$ levels are shown (corresponding to 7:00 local time) for 137 of the 150 sites. Slightly smaller back dots are sites missing the 7:00 observation. On the right, observed time series for a few of the sites are shown. Lines connect adjacent observations in time, with missing values excluded.

4.1.2 Simulations

The simulator used in this work is a Lagrangian particle dispersion model called FLEXPART ("FLEXible PARTicle dispersion model") (Stohl et al., 2005). An in-depth discussion of the simulator and simulations can be found in Lucas et al. (2017), while we only introduce the simulations briefly here. The FLEXPART model requires six inputs detailing characteristics of the release (latitude, longitude, altitude, start time, duration, and amount) as well as a wind field. We use the Weather Research and Forecasting Model (WRF) to generate wind fields while varying only a few of WRF's many inputs. The inputs we vary are pre-release initialization time (9 or 15 hours), planetary boundary layer physics model used (YSU, MYJ TKE, or MYNN TKE), land surface model used (thermal diffusion, Noah, or RUC), FDDA nudging amount (none, low, or high), and type of reanalysis data used (NARR, ECMWF, or CFSR). We use a series of five nested domains for evaluating WRF, so that the innermost domain resolves winds to 300 meters. More about these parameters and nesting can be found in Skamarock et al. (2008). A combination of the five categorical variables yields a wind field, which, along with a setting of the six continuous variables detailing the release characteristics, yields a simulated plume.

We obtain an ensemble of 18000 evaluations of FLEXPART with the 11-dimensional inputs sampled from a Latin Hypercube using the ranges in Table 4.1 for the six continuous parameters. We note that 18000 model evaluations is uncommonly large in the computer experiment literature, though there are some recent applications with many evaluations (Gramacy & Apley, 2015; Kaufman et al., 2011). Our computational budget allowed for this large number of evaluations, though each is still expensive to obtain. As discussed above, this large number of simulations makes emulation difficult, since traditional emulation techniques do not scale well.

The output from one evaluation of the simulator is spatio-temporal on a grid of 400×400 spatial locations and 34 time points, as shown in Figure 4.2. The 34 time points are 30-minute averages starting at 6:00 local time, thus extending to 23:00. We use only a subset of the 160,000 spatial locations when building the emulator, though the methods we present are scalable to moderately large spatial grids. Specifically, we use the 137 spatial locations that correspond to the sites shown in Figure 4.1, which are sites where we collect measurements. We exclude 13 of the 150 measurement sites from the analysis because they fall outside the region considered in the simulations, and are far enough away from the release that they are likely to only measure background levels.

Both simulation and observation data are transformed to be on the $\log_{10}$ scale after adding a constant of 20 to all values. The log scale makes modeling the many orders of magnitude difference in concentrations easier. Adding 20, determined in consultation with field experts, minimizes the amount of attention we give to small (in this case unimportant) concentration values.

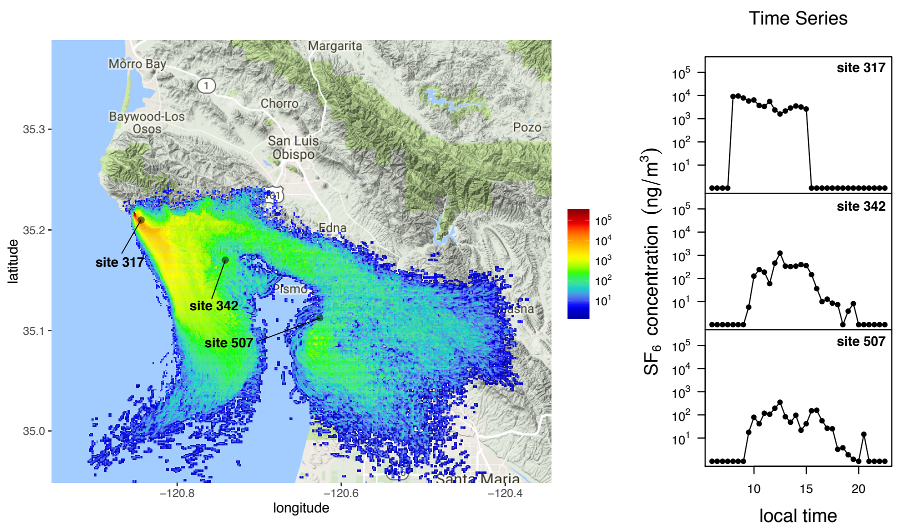


Figure 4.2: We show the simulated SF_6 plume for one of the 18000 simulations on the left (corresponding to 9:30 local time). On the right, we show simulation time series for a few of the sites.

4.2 Emulator

A usable emulator in this case needs to be able to produce a reasonably close approximation to the computer model for any possible combination of the 11 inputs. That is, given the characteristics of the release and the wind field charac-

Table 4.1: FLEXPART continuous parameter ranges

parameter	lower bound	upper bound	true value
latitude	35.1977	35.2250	35.2111
longitude	-120.8708	-120.8384	-120.8543
altitude (meters)	1	10	2
start time	7:00	9:00	8:00
duration (hours)	6	10	8
amount (kg)	10	1000	146.016

teristics, it should produce the spatio-temporal plume. Obtaining an estimate of emulation error is also a necessary task in order to allow us to propagate the error into calibration uncertainty. If this error is disregarded, we leave open the possibility of being too confident in our estimates of release and wind characteristics that could have generated the calibration data. This case also requires an emulator that can use the large number of model runs available.

Our model runs provide us with 18000 plumes in space and time. We use these plumes to obtain empirical orthogonal functions (EOFs) in space and time jointly, and we model the weights in this EOF decomposition using adaptive splines.

4.2.1 Empirical Orthogonal Functions

Let $y^c(s, t, \mathbf{x})$ denote computer model output at spatial location s, time t and 11-dimensional input setting $\mathbf{x}$. The model output is on a grid of n_s spatial locations and n_t times. We define the vector of model output for input setting $\mathbf{x}_j$ as

$$\mathbf{y}^c(\mathbf{x}_j) = (y^c(s_1, t_1, \mathbf{x}_j), y^c(s_2, t_1, \mathbf{x}_j), \ldots, y^c(s_{n_s}, t_1, \mathbf{x}_j), y^c(s_1, t_2, \mathbf{x}_j), \ldots, y^c(s_{n_s}, t_{n_t}, \mathbf{x}_j))' \tag{4.1}$$

and define the matrix of model run output $\mathbf{Y}^c = [\mathbf{y}^c(\mathbf{x}_1), \ldots, \mathbf{y}^c(\mathbf{x}_{n_x})]$ for the n_x model runs, which has dimensions $n_s n_t \times n_x$. We obtain discretized EOFs using the singular value decomposition, yielding $\mathbf{Y}^c = \mathbf{U}\mathbf{D}\mathbf{V}'$. The matrix $\mathbf{U}$ is the $n_s n_t \times n_s n_t$ matrix that has EOFs as columns and $\mathbf{D}\mathbf{V}'$ is the $n_s n_t \times n_x$ matrix of weights. To reduce the dimension in the problem, we use only the first k EOFs, where $k < n_s n_t$, resulting in the truncated decomposition $\hat{\mathbf{Y}}^c = \hat{\mathbf{U}}\hat{\mathbf{D}}\hat{\mathbf{V}}'$ where $\hat{\mathbf{U}}$ has dimension $n_s n_t \times k$ and $\hat{\mathbf{D}}\hat{\mathbf{V}}'$ has dimension $k \times n_x$. For modeling purposes, we write the non-truncated EOF decomposition as $y^c(s, t, \mathbf{x}_j) = \sum_{i=1}^{n_s n_t} K_i(s, t) w_{ij}$ where $K_i(s, t)$ is the i^{th} EOF at spatial location s and time t and w_{ij} is the corresponding weight for $\mathbf{x}_j$, specifically $(\mathbf{D}\mathbf{V}')_{ij}$.

We specify our emulator as the truncated decomposition

$$y^c(s, t, \mathbf{x}) = \sum_{i=1}^{k} K_i(s, t) w_i(\mathbf{x}) + u(s, t) \tag{4.2}$$

where $u(s, t)$ is the truncation error. We replace w_{ij} with $w_i(\mathbf{x})$ in order to allow us to predict computer model output for $\mathbf{x}$ not in our training sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_{n_x}\}$. We model the weight functions using adaptive splines with

$$w_i(\mathbf{x}) = \eta_i(\mathbf{x}) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma_i^2) \tag{4.3}$$

using the values of $\{w_{i1}, \ldots, w_{in_x}\}$ to train $w_i(\cdot)$. We discuss the form of the function $\eta_i(\cdot)$ in the next section. We assume that the weights on the EOFs are independent *a priori*.

Regarding the truncation, we need k, the number of EOFs used, to be sufficiently large to yield a suitable approximation, but small enough to be computationally feasible. Assuming a parametric distribution for the truncation error, $u(s,t)$, for which it is independent and identically distributed for all s and t is likely to improperly characterize the emulation uncertainty because of the large variation in plume characteristics in space and time. Instead we assume truncation error for a particular (s,t) combination comes from the distribution of $n_x = 18000$ truncation errors we have seen already,

$$u(s,t) \sim Unif \left\{ y^c(s,t,\mathbf{x}_j) - \sum_{i=1}^{k} K_i(s,t) w_{ij} \right\}_{j=1}^{n_x}. \tag{4.4}$$

Since the EOFs are fixed, there is little interest in the truncation error after we have chosen k, other than to make sure it is propagated during calibration. Hence, we see no value in trying to estimate a parametric distribution. The large value of n_x makes our discrete distribution for the truncation error accurate without making any limiting assumptions about distribution tails and symmetry. Further, since this distribution has no unknown parameters and because the weights $w_i(\mathbf{x})$ and $w_j(\mathbf{x})$ have no *a priori* correlation, the weights will also have no *a posteriori* correlation. This means that the adaptive spline models for $w_i(\mathbf{x})$ and $w_j(\mathbf{x})$ can be fit completely in parallel, which is a significant computational benefit.

4.2.2 Adaptive Splines

As we have discussed in Chapter 2, BMARS models can be a powerful tool for emulation for a number of reasons: (1) they are adaptive because knots and variables are only included in basis functions if they are necessary; (2) they perform implicit variable selection, as basis functions only include variables that are useful; (3) they scale well, especially when compared to Gaussian process models; (4) they can flexibly pick up localized signal when the data suggest such signal exists; (5) they yield analytical Sobol' sensitivity indices; and (6) they can be used to emulate simulators with functional response.

The model for the adaptive spline mean function $\eta_i(\mathbf{x})$, used in Equation 4.3, is a linear combination of tensor product basis functions. We drop the index as we describe this model, since it is fit independently for $i = 1, \ldots, k$. Thus, $\eta(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m B_m(\mathbf{x})$ where the basis function $B_m(\mathbf{x})$ is of the form

$$B_m(\mathbf{x}) = \begin{cases} \prod_{j=1}^{J_m} g_{jm}[s_{jm}(x_{v_{jm}} - t_{jm})]_+ & \text{if } J_m > 0 \\ 1 & \text{if } J_m = 0. \end{cases} \tag{4.5}$$

When $J_m > 0$, the basis function is a tensor product of polynomial splines where t_{jm} is a knot, v_{jm} indexes a variable, and $s_{jm} \in \{-1, 1\}$. The function $[\cdot]_+$ is defined as $\max(0, \cdot)$. The constant g_{jm} scales the basis function to have maximum of one, which helps with computational stability. Without loss of generality, if $x_{v_{jm}} \in [0, 1]$, then $g_{jm} = [(s_{jm} + 1)/2 - s_{jm}t_{jm}]^{-1}$. A basis function has J_m elements in the tensor product where each is required to involve a different variable. Hence, J_m is the degree of interaction for basis function m. The piecewise structure of the basis function will become important in the next section, where we will discuss the case when $J_m = 0$.

The unknowns associated with $\eta(\mathbf{x})$ are the number of basis functions, M, the basis function weights $\mathbf{a}$, and the basis function parameters. The value of σ^2 from Equation 4.3 is also unknown (again, we drop the index for notational simplicity). We assign priors to all of these parameters following the specifications of Chapter 2.

4.2.3 Categorical Predictors

As we have discussed, our emulator needs to be able to handle categorical inputs. This requires an extension of the traditional adaptive splines models that follows an approach similar to Friedman (1991b), but modified to fit into our Bayesian framework. Specifically, if the inputs to the simulator are $(\mathbf{x}, \mathbf{z})$ where $\mathbf{z}$ is a vector of categorical variables, we define the portion of the basis function due to the categorical variables as

$$B_m(\mathbf{z}) = \begin{cases} \prod_{l=1}^{L_m} 1(z_{u_{lm}} \in D_{lm}) & \text{if } L_m > 0 \\ 1 & \text{if } L_m = 0. \end{cases} \tag{4.6}$$

where u_{lm} indexes a categorical variable, D_{lm} is a proper subset of the categories of variable u_{lm}, and L_m is the degree of interaction of categorical variables. This approach to handling categorical variables is somewhat similar to Storlie et al. (2015) and Ma et al. (2015). We combine this portion of the basis function with Equation 4.5 so that the m^{th} basis function is $B_m(\mathbf{x}, \mathbf{z}) = B_m(\mathbf{x})B_m(\mathbf{z})$. The purpose of allowing for $L_m = 0$ or $J_m = 0$ is to allow for basis functions that involve only the continuous predictors, or only the categorical predictors, respectively. In our case study, we permit up to third-order interactions of each variable type ($J_m = 3$ and $L_m = 3$) maximally resulting in six-way interactions if the data dictates that such interactions are useful. Allowing D_{lm} to be a subset of categories rather than a single category is useful for cases when two or more categories exhibit similar behavior, as it allows us to use fewer basis functions to describe the behavior. We again use a discrete uniform prior for the categorical basis function parameters $\{L_m, (u_{lm}, D_{lm})_{j=1}^{L_m}\}_{m=1}^{M}$, and we alter the constraint discussed in Chapter 2 to apply to the new set of basis functions that include both categorical and continuous variables. That is, each resulting basis function must have at least 20 non-zero points.

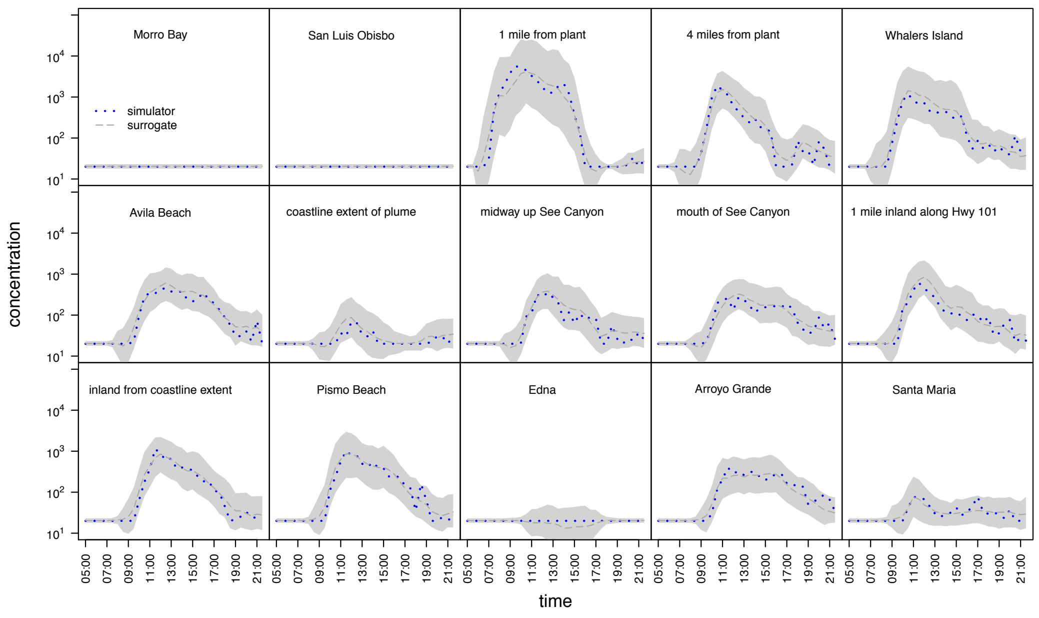
4.2.4 Performance

We demonstrate the performance of the emulator by comparing emulator and simulator output at input settings not used to fit the surrogate or to build the EOFs. We use the R package BASS (Francom, 2017), introduced in Chapter 3, to fit the BMARS models. Figure 4.3 shows the performance at 15 of the 137 spatial locations for two different holdout simulator runs. In addition to the emulator mean, posterior predictive uncertainty is shown. The prediction uncertainty varies with space and time because (1) the homoscedastic BMARS error from Equation 4.3 is multiplied by the corresponding spatio-temporal EOF in Equation 4.2, thus producing spatio-temporal noise and (2) the truncation error varies in space and time. The models corresponding to the most important EOFs use around 200 BMARS basis functions to fit the data, while the least important EOFs, which correspond to higher frequencies, use around 50 basis functions.

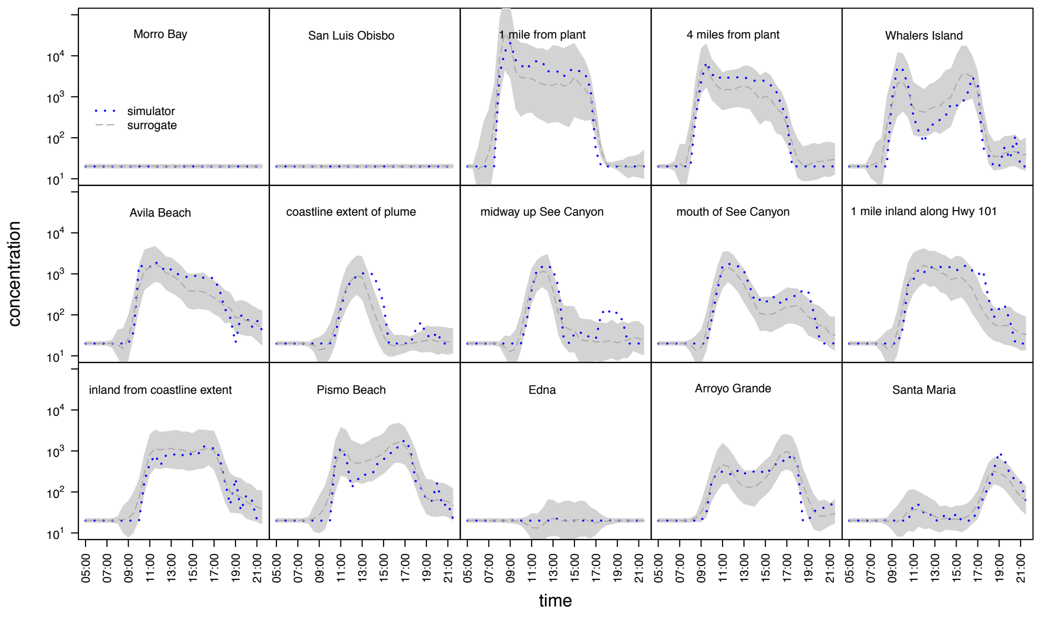
These holdout predictions demonstrate reasonable emulator performance, especially considering the complexity of this simulator. Predictions of a number of other randomly chosen holdout simulations also show good performance. By using a large number of EOFs in our emulator, we are able to capture subtle variation in the shape of the time series for different input settings. For instance, the difference in the shapes of the two time series for the Whalers Island location (top right in Figure 4.3 (a) and (b)) could be attributed to “random” variation if a less accurate emulator was used. Instead, we can attribute the difference to input parameter variation. Because the simulator is deterministic, there is technically no “random” variation, though there may be numerical variation that is difficult to attribute to any parameter.

Our efforts to build an emulator for a single location (using the 34 time points) resulted in poor emulation compared to those that utilize the spatio-temporal plume information. We also achieved better emulation performance using spatio-temporal EOFs than we did using EOFs that were separable in space and time. While separable EOFs have the benefit of easier interpretation, we have little interest in interpretation, and separability ends up being a poor assumption for this plume model.

The only case we foresee where we may be interested in interpretation of the EOFs is when doing sensitivity analysis. We can easily obtain the Sobol' decomposition for each EOF adaptive spline model, extending the work of Chapter 2 to include the categorical framework we have introduced (shown in Appendix B). However, if the EOF lacks interpretability, so too does the sensitivity analysis. Because the first EOF is generally interpretable as the average and accounts for most of the variance, sensitivity analysis of that EOF model may be interesting (shown in Figure 4.4). That sensitivity analysis shows that variable three, which is the release altitude, plays no role in the weight for the first EOF. Hence, we would expect to learn very little about that parameter during calibration, since it is also unimportant in all of the other EOF models. We note that a more interpretable functional sensitivity analysis can be performed analytically using our emulator (i.e. the Sobol' decomposition integrals are analytical), but it bears a large computational burden and will not be further explored in this chapter. A derivation of the Sobol' indices for this case is given in Appendix B.



(a) Holdout Sample 1



(b) Holdout Sample 2

Figure 4.3: Demonstration of emulator fit at 15 locations. The top plots (a) show the time series simulator output (dotted lines) for a particular set of inputs not used to train the emulator. The bottom plots show a different input setting, to demonstrate variation in the model output shape. Emulator posterior predictive means (dashed lines) and pointwise 95% probability intervals (shaded region).

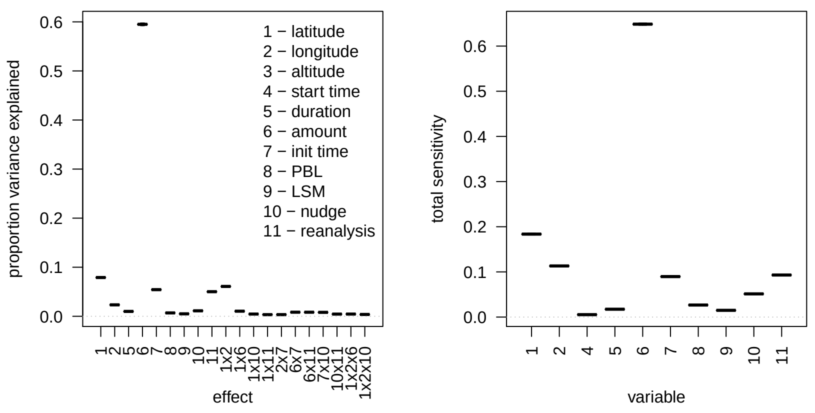


Figure 4.4: Sensitivity analysis for the adaptive spline model corresponding to the first EOF. Variable numbers 1-6 are the continuous inputs while variables 7-11 are the categorical inputs.

4.3 Calibration Model

Our strategy for calibration differs from that of Kennedy & O'Hagan (2001) in a few important ways. Rather than the GP emulator, we use BMARS. We also opt to fit the emulator before calibrating, independent of the observational data. While we do this primarily for computational reasons, it marks a general difference in strategy. A similar approach was used in Gramacy et al. (2015), with the justification that with a large number of simulations the observational data is not going to significantly influence the emulator. Others justify this strategy on philosophical grounds, because the emulator is meant only to replicate the computer model and thus should not be influenced by observational data (Liu et al., 2009).

Let $y^F(s,t)$ denote the log concentration data gathered from sensors at location s and time t. Let $\zeta(s,t)$ denote the true concentration at the same location and time. Then we set up the calibration model as follows:

$$y^F(s,t) = \zeta(s,t) + \nu, \quad \nu \sim N(0, \sigma_F^2) \tag{4.7}$$

$$\zeta(s,t) = y^c(s,t,\boldsymbol{\theta}) + \delta(s,t) \tag{4.8}$$

where ν is the observation error, σ_F^2 is the observation error variance and $y^c(s,t,\boldsymbol{\theta})$ denotes the estimated computer model output at location s, time t and input setting $\boldsymbol{\theta}$. The $\delta(s,t)$ term denotes the systematic model discrepancy. Hence, $\boldsymbol{\theta}$ is the unknown setting of the simulator that best matches reality when jointly considered with simulator discrepancy and observational error. While the observations are hourly averages, the emulator gives 30-minute averages. Thus, we average 30-minute averages in the emulator output to match the observation time scale. We also exclude emulator and discrepancy values at (s,t) where the corresponding $y^F(s,t)$ is missing.

A potential problem with the introduced modeling framework is that the observations can be produced in a number of different ways. For instance, we might get good prediction if we get as close as we can to the observations by only altering $\boldsymbol{\theta}$, and then consider $\delta(\cdot)$ to be the leftover spatio-temporal structure in combination with ν, the observational error. However, we could achieve equally good prediction by fixing $\boldsymbol{\theta}$ at a particular value and only altering $\delta(\cdot)$. These two examples represent the extremes in overfitting, but we may attain equally misleading combinations of these. Hence, we need restrictions in order to make all the terms identifiable. The most satisfying restrictions we can introduce are in the form of an informative prior for the shape of the discrepancy, rather than an informative prior for the parameters $\boldsymbol{\theta}$ (Liu et al., 2009). While there may be some applications where the shape of the discrepancy is somewhat understood, we do not have an informative prior for our discrepancy model.

Following Liu et al. (2009), we make the quantities of interest identifiable by modularizing the analysis further. Particularly, we fix the shape of the model discrepancy before trying to infer $\boldsymbol{\theta}$. We get an estimate of the model discrepancy by estimating the computer model output at the prior mean parameter settings (Bayarri et al., 2007b) and subtracting that from the observations. We then fit a BMARS model to that discrepancy as a function of s and t, effectively smoothing it out. While we fix this discrepancy shape, we allow for its influence to vary by multiplying the discrepancy by a scale factor, γ. Our prior for γ is uniform between zero and two. If γ is near zero, the influence of the discrepancy is minimal. If it is near two, then the discrepancy plays a larger role. We limit the upper bound to two because anything larger would give the discrepancy similar magnitude to the simulator output, which we hope is not the case. While we introduce a scale parameter to the discrepancy, we do not introduce an intercept parameter because doing so would likely confound our ability to learn one of the simulator parameters (release amount), which explains much of the magnitude variation. Without *a priori* knowledge of systematic magnitude discrepancy, we refrain from including an intercept (or a multiplicative discrepancy term for $y^c(\cdot)$, included in Kennedy & O'Hagan (2001)). Thus, we alter Equation 4.8 to be

$$\zeta(s,t) = y^c(s,t,\boldsymbol{\theta}) + \gamma\delta(s,t).$$

While the functional forms of $y^c(\cdot)$ and $\delta(\cdot)$ are fixed in advance, we incorporate their uncertainties in calibration by sampling their posterior predictive distributions, rather than using their mean functions.

Under this calibration framework, our likelihood is $\mathbf{y}^F|\boldsymbol{\theta},\sigma^2 \sim N(\mathbf{y}^c(\boldsymbol{\theta}) + \gamma\boldsymbol{\delta}, \sigma_F^2\mathbf{I})$ and our posterior is

$$\pi(\boldsymbol{\theta},\sigma_F^2,\gamma|\mathbf{y}^F) \propto N(\mathbf{y}^F|\mathbf{y}^c(\boldsymbol{\theta}) + \gamma\boldsymbol{\delta}, \sigma_F^2\mathbf{I}) IG(\sigma_F^2|a,b) 1(\boldsymbol{\theta} \in D) 1(\gamma \in [0,2]) \tag{4.9}$$

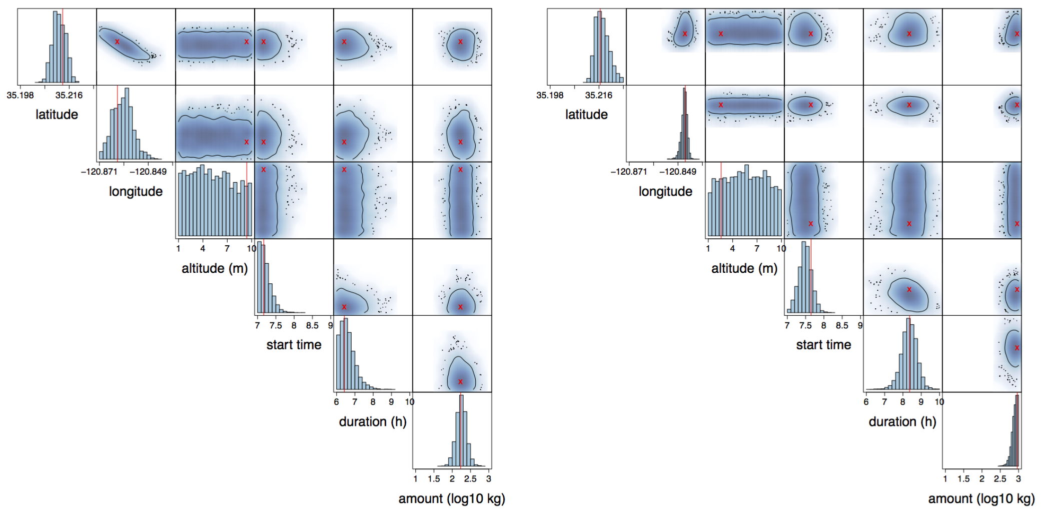
where D is the hypercube based on the prior ranges identified in Table 4.1 that also allows for all the discrete parameter combinations. We obtain samples from the posterior by using Markov chain Monte Carlo (MCMC) methods (Gelman et al., 2013). We sample σ_F^2 and γ from their Inverse Gamma and Truncated Normal full conditionals, respectively. We use the Metropolis-Hastings algorithm to sample the six continuous parameters in $\boldsymbol{\theta}$ from their joint full conditional. We sample the five categorical parameters jointly from their discrete full conditional. Specifically, there are 162 combinations of the categorical parameters. We take a sample from the emulator posterior predictive distribution for each of the 162 combinations and evaluate the posterior up to a constant. This is reweighted to produce a posterior (full conditional) probability for each setting of the categorical parameters. We then sample one of the 162 combinations according to that probability distribution.

The computational bottleneck to this algorithm is sampling the emulator posterior predictive distribution, as it requires building the BMARS basis functions for all EOF models. A single sample takes 0.2 seconds when the tasks for the 100 models are split between four cores. To obtain 162 samples, one for each categorical combination, a naive approach would require more than 30 seconds for each MCMC iteration. We overcome the bottleneck in sampling the posterior predictive 162 times by building all of the possible categorical basis functions in advance. For each EOF model and each emulator posterior sample, we obtain the basis functions used in each of the 162 predictive combinations. The resulting basis functions, which are combinations of ones and zeros, are multiplied (pointwise) by the portions of the basis functions from the continuous predictors. This allows us to obtain the 162 posterior predictive samples in less than 0.1 seconds on four cores, in contrast to the naive approach that takes more than 30 seconds. Though the categorical basis functions may seem like they would have a large memory footprint, they are combinations of ones and zeros and thus can be stored efficiently in memory.

4.3.1 Synthetic Calibration

To test our methods, we can calibrate to data where we know the truth. Particularly, we use two of the holdout model runs to perform two synthetic calibrations. That is, we treat each of the two holdout model runs as the true data. Hence, we exclude the simulator discrepancy portion of the model. The goal is to see if the parameters used to generate the model runs can be reasonably identified. One- and two-way marginal posterior distributions for the six continuous parameters are shown in Figure 4.5. These show that we are able to learn five of the six parameters well. However, we are unable to learn the altitude from these data. This is not surprising, given that our sensitivity analyses showed that altitude played little to no role in the emulator, so the output cannot constrain this particular input. Upon closer investigation, we found that the grid used in the simulations was too coarse to learn anything about the altitude parameter in the range that was specified *a priori* (see Table 4.1).

Regarding the five categorical parameters that are inputs to WRF, we are able to identify these well in the two synthetic examples (these results are not shown). Most posterior probability (86% and 87% in the two examples) is given



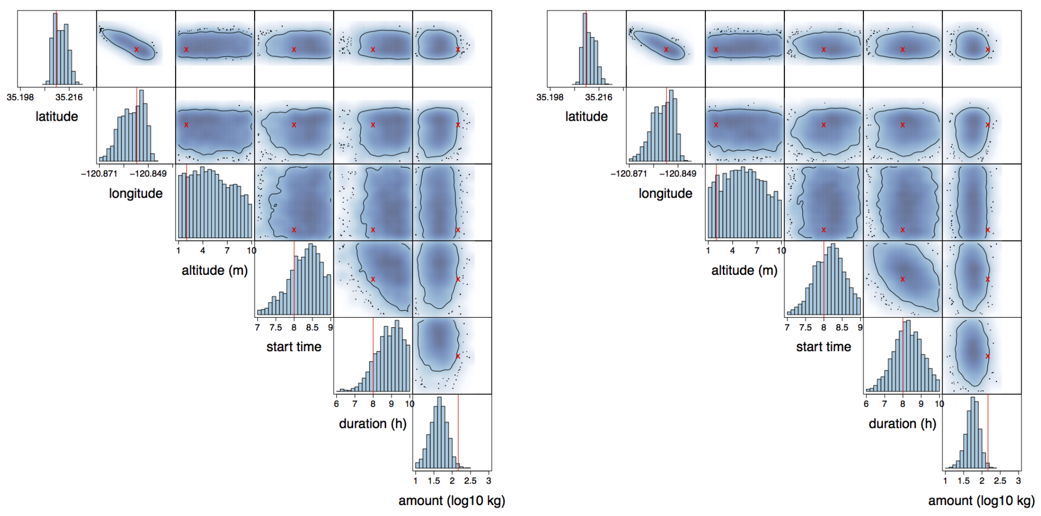
(a) Synthetic Calibration 1 (b) Synthetic Calibration 2

Figure 4.5: Marginal posterior distributions of continuous parameters in two synthetic calibration problems where we calibrate to model run output. True values are marked with vertical lines and X's. 95% contours are shown in the two-way marginal plots.

to the particular combination of five variables that are the true settings. Other synthetic calibrations using holdout data have shown that the land surface model can be difficult to identify in some cases (results not shown).

4.4 Inferring the Source of a Diablo Canyon Release

In this section, we present the results of the case study. We first discuss calibration results under two different discrepancy settings: (1) assuming no discrepancy (i.e., $\delta = 0$) and (2) using the modularized discrepancy discussed above. Figure 4.6 shows the one- and two-dimensional marginal posterior distributions of the continuous parameters for the no discrepancy and modularized discrepancy cases. The true values of the parameters are also shown in Figure 4.6, from which we see that latitude and longitude are well identified under both discrepancy models. As can be seen from its nearly uniform posterior distribution, altitude is not well identified for the reason discussed in the previous section. The differences between these cali-

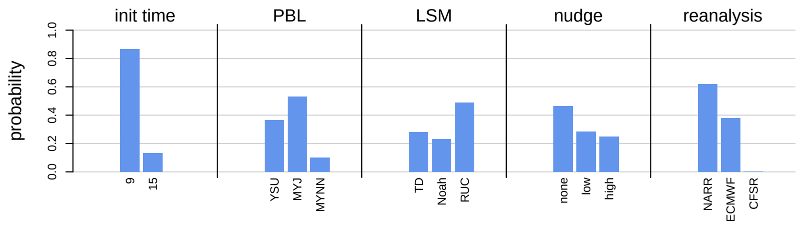


(a) Calibration Without Discrepancy (b) Calibration With Discrepancy

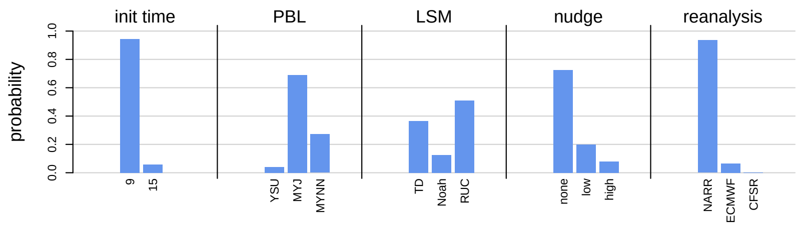
Figure 4.6: Marginal posterior distributions of continuous parameters obtained from calibrating to real data. True values are shown with vertical lines and X's. The left panel does not use discrepancy while the right panel does. 95% contours are shown in the two-way marginal plots.

brated release location parameters are negligible under the two different discrepancy models. However, the release timing parameters (start time and duration) exhibit a fairly significant shift. The discrepancy model helps to reduce the bias of the timing parameters. While both models result in a biased amount parameter, the bias is reduced when a discrepancy model is included.

The bias in the calibrated amount parameter is most likely due to the fact that the amount is highly correlated with the maximum (over space and time) of a model run. The maximum is important because we are using the log scale, so other measurements may be orders of magnitude smaller and thus would not reflect much difference in the amount. Since the emulator is more smooth than the observations, the emulator maximum tends to be smaller than that seen in the observational data. Hence, we did not have a bias in our calibrated amount under the synthetic calibrations discussed earlier, as the synthetic data are more smooth, and maximum values tend to be well predicted by the emulator.



(a) Calibration Without Discrepancy



(b) Calibration With Discrepancy

Figure 4.7: Marginal posterior distributions of categorical parameters obtained from calibrating to real data. The top panel does not use discrepancy while the bottom panel does. Unlike the continuous parameters, the true values of these parameters are unknown.

In Figure 4.7, the posterior marginal distributions of the categorical variables are shown for the two discrepancy cases. These distributions show significant change when a discrepancy model is used. The distribution of each parameter becomes less varied when the discrepancy is included. Notably, the reanalysis parameter strongly favors the North American model (NNAR) over the European model (ECMWF) when the discrepancy is included. The distribution of the planetary boundary layer (PBL) physics parameters is altered to give little mass to the YSU setting. There is also a much stronger preference for no nudging when discrepancy is included.

While the original goal of this analysis was to test the calibration methodology on an experimental release where many of the release characteristics were known, the field study was conducted over three decades ago at a time when global positioning satellites were not available. As a result, there is a discrepancy between the coordinates of the release location in the original field study documentation (UTM 695368 Easting, 3898440 Northing, and zone 10) and the qualitative description of the release site in the paper by Thuillier (1992). Our initial analysis showed that our estimate (posterior mean) of the location of the release (latitude and longitude) was biased by about 100 meters and more consistent with location described in Thuillier (1992), which indicates that the release occurred at the base of Containment Unit 2 (the southmost containment unit). The corrected release location is supported by our calibration analysis, indicating it is much more probable than the originally reported release location, as shown in Figure 4.8.

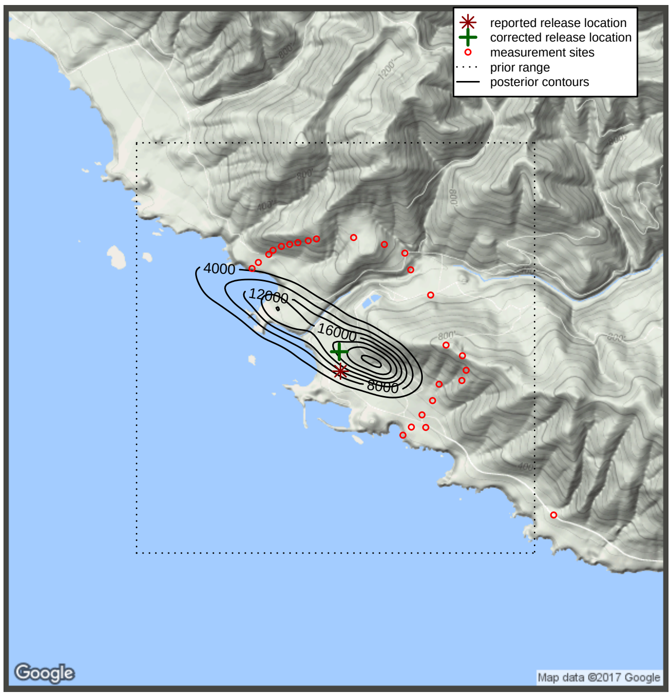


Figure 4.8: The marginal posterior distribution of the latitude and longitude parameters, as well as the reported location (see text for details).

As discussed previously, our calibration approach fixed the shape of the model discrepancy before inferring the calibration parameters. The scale parameter γ, which would inflate or deflate the effect, preferred a deflated discrepancy (95%

probability interval of (0.58, 1.02) for γ). The model discrepancy inferred from the data can be a useful tool in understanding what parts of the model could be explored further to improve predictive accuracy. We perform an exploratory data analysis of the discrepancy to try to determine its meaning. We see common shapes in groups of the discrepancy time series. When we cluster these time series, we partition our spatial field into areas where the discrepancy time series look similar. The clustered time series and the accompanying locations are shown in Figure 4.9. Clustering was performed using the fdakma R package (Parodi et al., 2015) with a K-means (but using the medians) type of algorithm that calculated the L2 distance between the first derivatives (approximated by differencing) of the time series. Thus, cluster membership was determined based on similarity of the shape, rather than amplitude of the curves. The number of clusters was selected based on visual assessment of the clustering results. The more interesting parts of our discrepancy are in clusters three and five, shown in Figure 4.9. Cluster three corresponds to locations that would be in the early path of the plume under the meteorological conditions we observe. The shape of the time series in cluster three seems to indicate a discrepancy in the timing of the plume reaching those locations, implying that the simulator's early concentrations are too high and late concentrations are too low. This matches the fact that our inference regarding the timing parameters was improved when we included this discrepancy model. As shown in Figure 4.6, excluding a discrepancy results in later inferred start time. Cluster five corresponds to the spatial locations closest to the release. These have largest discrepancy likely because of the high concentration of the plume just after the release. With only a slight perturbation of the wind conditions, the early plume predictions can be in completely different locations because the plume is so concentrated. Thus, if wind predictions are only slightly inaccurate, the early plume predictions yield a large model discrepancy.

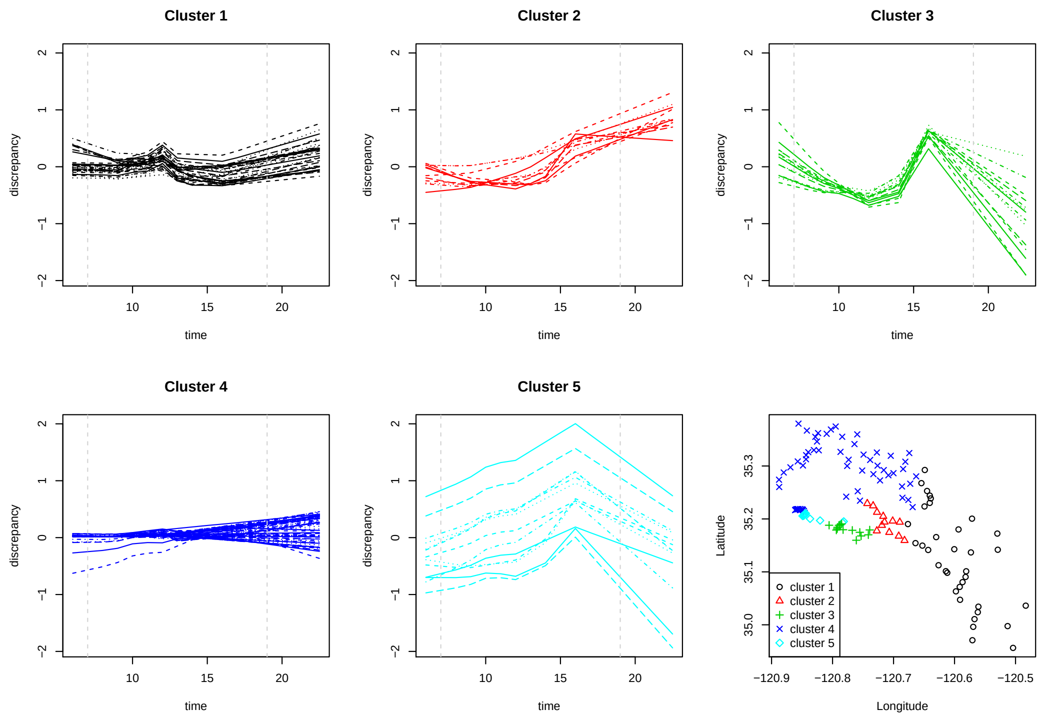
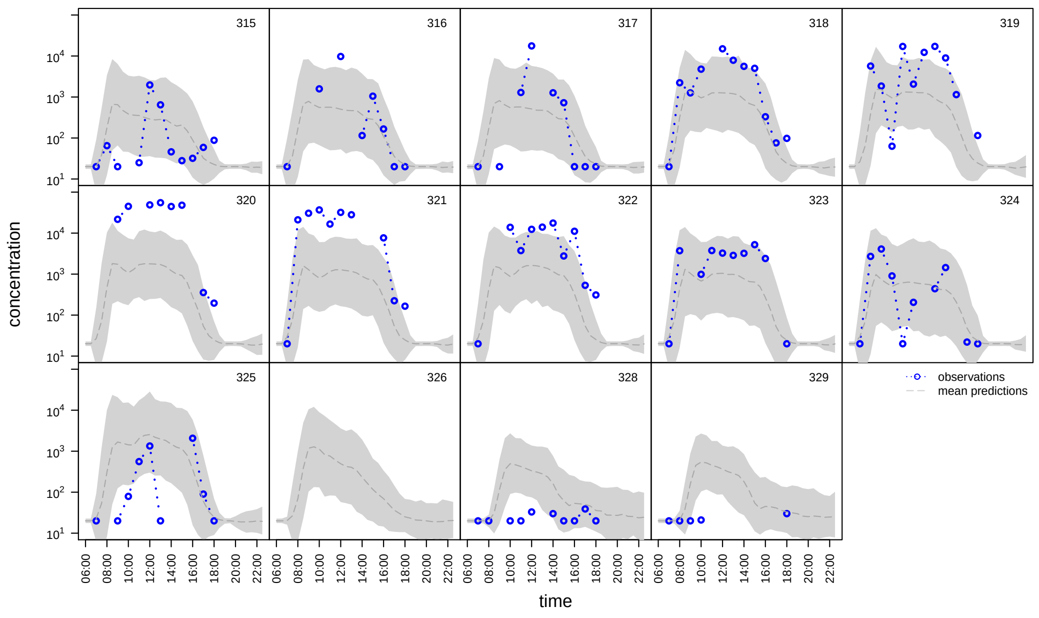


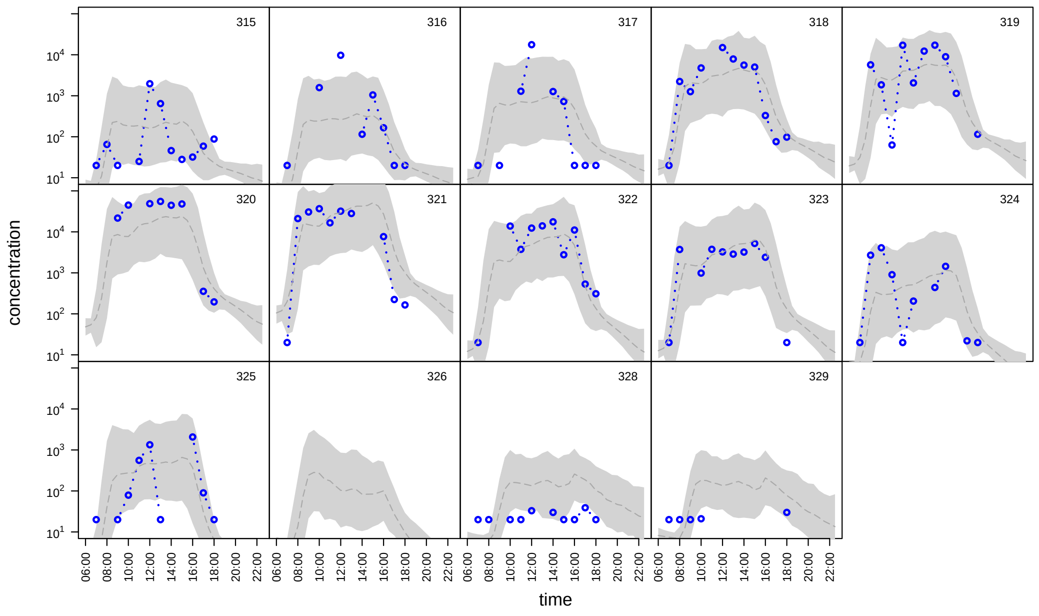
Figure 4.9: Clusters of discrepancy time series according to shape. The bottom right plot shows locations marked by cluster membership. The vertical dotted lines in the time series represent interval limits outside of which extrapolation begins.

Our calibrated predictions with and without discrepancy for the locations in cluster five are shown in Figure 4.10. These show that including discrepancy brings a significant benefit at those locations while not being overly complex in shape. These also show that the extrapolated predictions including discrepancy can be fairly inaccurate. Indeed, the curves shown in Figure 4.9 show that the temporal extrapolation is linear. As in all discrepancy functions that are motivated by data rather than scientific input, this extrapolation should not be trusted. Note that one of the locations, site 326, is missing all of the observations. Because site 326 is located such that it is not much of a spatial extrapolation (not shown), this prediction may be trustworthy.

To see the broader effect of including discrepancy on calibrated predic-



(a) Prediction Excluding Discrepancy



(b) Prediction Including Discrepancy

Figure 4.10: Calibrated prediction for the locations in cluster 5. The top panel shows prediction excluding discrepancy, while the bottom panel includes discrepancy. The 95% pointwise probability intervals do not include the observational error, which is Normal with posterior standard deviation near 0.39 (95% probability interval (0.35, 0.43) for σ_F, symmetry coincidental).

tion, we show the predicted (posterior mean) versus observed data in Figure 4.11. This shows that the discrepancy model, while simple, generally improves prediction. The improvement is most significant for large values, corresponding to the location clusters closest to the release.

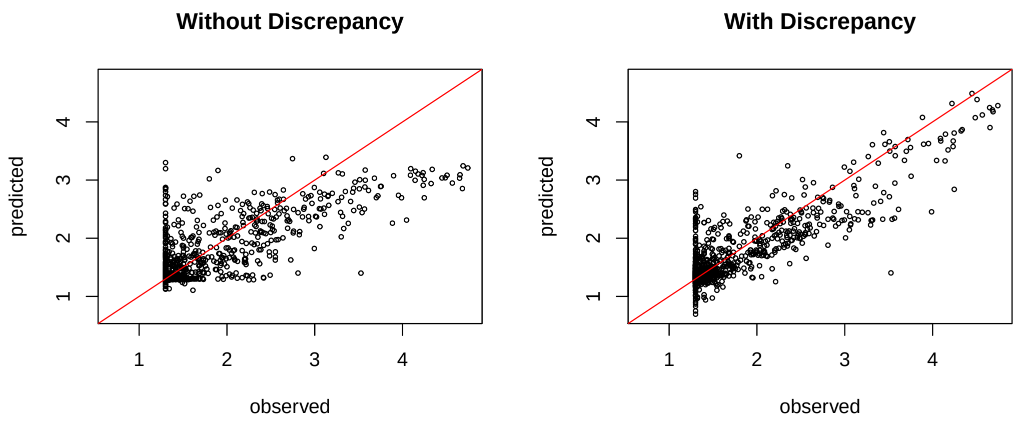


Figure 4.11: Calibrated prediction versus the observations, with and without the discrepancy in prediction.

4.5 Discussion

We have presented an analysis of a computer experiment with important applications to locating and assessing an atmospheric release. In the process, we have developed emulation methodology that can be scaled for use with large numbers of model runs when each has functional output. We have extended existing BMARS methodology to allow for categorical inputs and to model in a reduced dimension space. In building this emulator, we have detailed how to keep track of different sources of uncertainty. We have extended modular approaches to computer model calibration for use with a BMARS emulator and a BMARS discrepancy function, paying special attention to identifiability.

The immediate applications of this work are for research and development purposes rather than emergency response. In an actual emergency, there would be limited time to run WRF (in forecast mode) to get the wind fields for use in FLEXPART. However, an approach that uses weather analogues (Delle Monache et al., 2011) rather than WRF runs may be feasible. An ensemble of FLEXPART simulations could be obtained in parallel, and emulation and calibration models could be fit thereafter. In order to be useful, emulation and calibration need to be done quickly and accurately. Hence, the emulation and calibration methodology outlined here has potential to be useful. This work would also be valuable for forensic and remediation purposes, i.e., determining what happened during an atmospheric release after it has ended, or understanding what populations were exposed during a release.

A possible extension of the proposed model would be to include functional input. That is, rather than parameterizing WRF with five categorical parameters, we could include the entire spatio-temporal wind field as input for the emulator. This may be possible by decomposing wind fields onto a set of basis functions and including the basis function weights as inputs to the BMARS emulator.

Chapter 5

Functional Nonlinear Regression and Registration using Bayesian Adaptive Splines

When a computer model outputs functional data, it is sometimes the case that the functional data are misaligned. For instance, if a computer model response is a time series, and different input combinations yield time series with different time scales, the functional data are misaligned. Many of the fundamental uncertainty quantification tasks discussed in previous chapters are complicated by the misalignment, beginning with emulation.

In this chapter, we develop an emulator that can handle misaligned functional data. This is motivated by a high-energy-density physics computer model used to simulate inertial confinement fusion ignition experiments that take place on the National Ignition Facility (NIF) at Lawrence Livermore National Laboratory. The goal of these experiments is to achieve nuclear fusion by compressing and heating a fuel capsule using high-energy lasers. The simulator, called HYDRA, takes nine inputs describing the laser stimulus (shape, amount, and timing) given to the capsule as well as the density of the fuel in the capsule (Peterson et al., 2017). The simulator outputs many quantities, but we are particularly interested in the energy production rate over time. Figure 5.1 shows the log of the simulator output from five different combinations of the nine inputs. The goal is to build an emulator that can predict and quantify prediction uncertainty using nearly 25000 simulations.

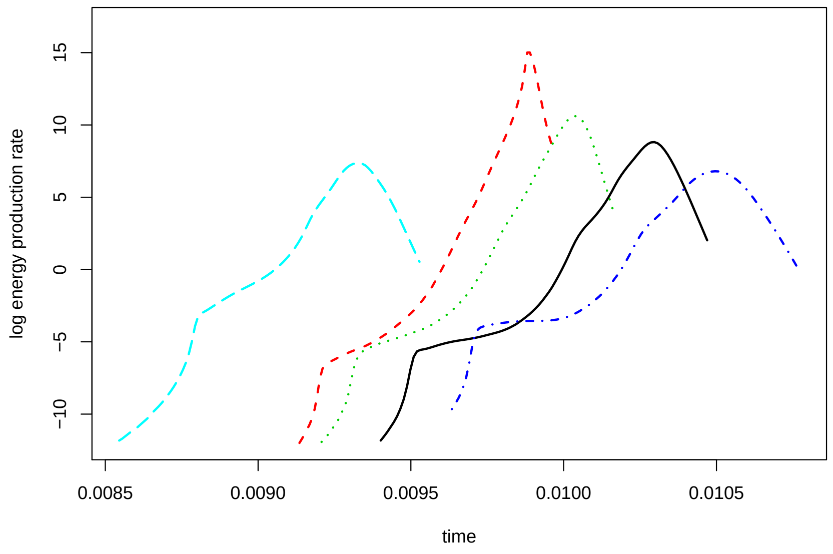


Figure 5.1: Five curves output from five runs of the HYDRA simulator with different input settings.

Approaches to emulation for models with misaligned functional responses are limited. Hung et al. (2015) build a Gaussian process emulator, including the functional response using a Kronecker covariance structure and handling the misalignment with missing data methods. Missing data methods, where the functional response is imputed for the entire range of possible functional responses, are unlikely to be useful here because of the large variability in the misalignment of the responses, as shown in Figure 5.1. Other possible approaches center on registration, which is the process of aligning functional data. When only the aligned functions are of interest, the data can be registered as a preprocessing step, as in Bayarri et al. (2007a). An emulator built from registered data would be limited to prediction of aligned curves only, unless a registration model was incorporated into the emulator.

The prospect of combining a registration model with an aligned functional data model is of interest to functional data analysis researchers more generally. For instance, Earls et al. (2017) develop a model that, for a functional dataset without covariates, uses Gaussian processes to simultaneously infer the aligned mean curve and the warping function, which is the function that transforms the unregistered data to be registered. Telesca (2015) also considers, for the purposes of inference and not prediction, a mixed model for both the warped data and the warping functions. A good deal of attention has been given to modeling of warping functions for inference purposes (Gervini & Gasser, 2004; Tucker et al., 2013), but attention has not been given to modeling warping functions with covariates, especially in a nonlinear fashion. A warping function must be non-decreasing, which can complicate modeling of warping functions. Further, joint modeling of phase and amplitude variability can lead to identifiability issues, since overfitting of phase variability by not constraining warping functions to be smooth is detrimental to the analysis of the amplitude variability.

While many methods for registration have been proposed, landmark registration is preferred when possible (Gervini & Gasser, 2004). Landmark registration consists of aligning functions according to where important functional traits (landmarks) occur. For instance, local maxima, minima, or inflection points may be considered landmarks. Landmarks can be considered realizations from the warping functions, making the shape of the warping functions easier to identify. The difficulty with landmark registration is that landmarks can sometimes be difficult to select. Hence, methods that do not depend on landmarks are more common. Recent work by Strait & Kurtek (2016) develops a method to estimate landmark placement with uncertainty. In the case of the HYDRA simulations, there are a few clear landmarks occurring in each curve. These are the start, end, global maximum, and point at which the curve changes direction between the start and global maximum. If we consider warping functions that are linear between the landmarks, Figure 5.2 shows what the aligned curves look like. For comparison, Figure 5.2 also shows what the aligned curves would look like with only the start and end points as landmarks. Variation near the excluded landmarks would require the model for the aligned curves to be more complex, and justifies the inclusion of all four landmarks since they are easily identified. Figure 5.2 shows the (inverse) warping functions using four landmarks.

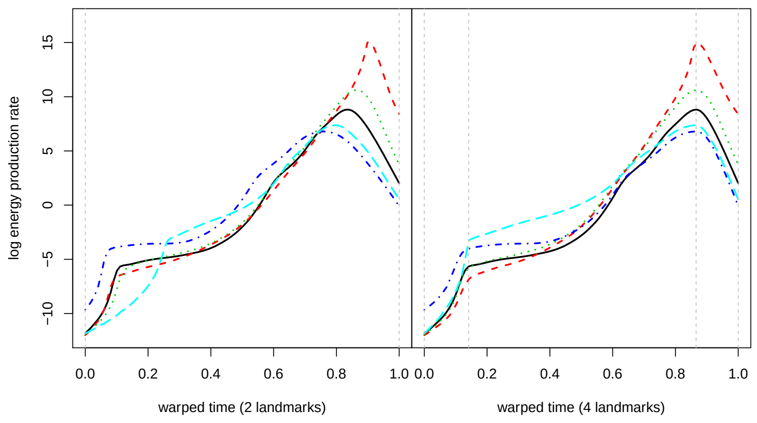


Figure 5.2: The five curves from Figure 5.1 warped in two different ways. The left panel shows landmark warping with two landmarks, while the right uses four landmarks. Dotted vertical lines indicate the location of landmarks

We propose an emulator that uses a Bayesian adaptive spline model for the

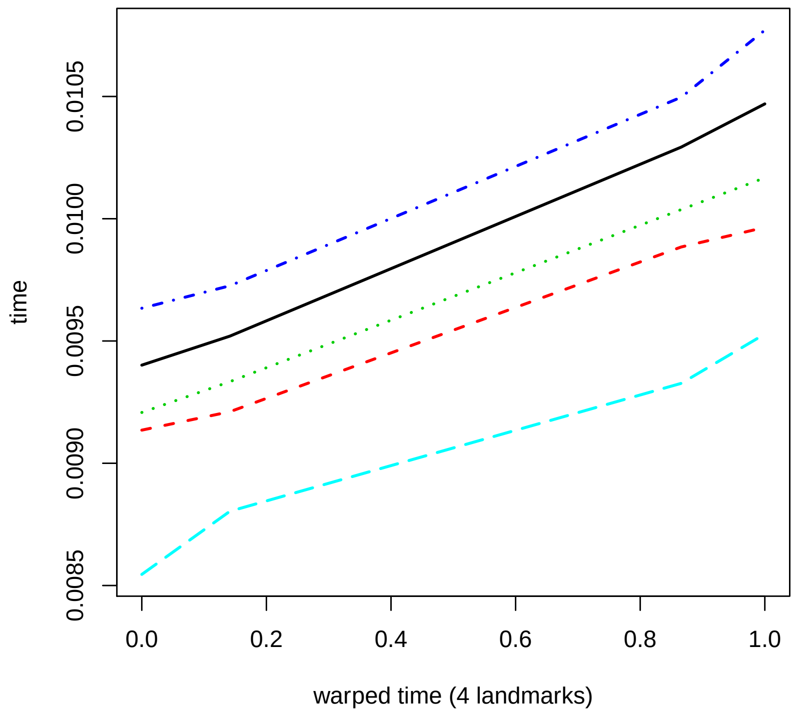


Figure 5.3: Warping functions for the five curves in Figure 5.1 that are linear between landmarks.

warped curves as well as a Bayesian adaptive spline model for the warping functions.

The most natural way to require a BMARS model to be non-decreasing in the functional variable would be to include such a constraint in the likelihood. Then, if $f(r, \mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m B_m(\mathbf{x}) B_m(r)$ is the corresponding BMARS mean function, $\frac{d}{dr} f(r, \mathbf{x}) = \sum_{m=1}^{M} a_m B_m(\mathbf{x}) B'_m(r)$, and $B'_m(r)$ is easy to obtain as the derivative of a truncated linear function. However, for a particular $f(r, \mathbf{x})$, we need to check that $\frac{d}{dr} f(r, \mathbf{x}) \geq 0$ for all possible $\mathbf{x}$. This becomes a combinatorial problem because of the dimension of $\mathbf{x}$ and the size of M. Another possible approach is to only allow non-decreasing basis functions. Because $0 \leq B_m(\mathbf{x}) \leq 1$, this simply requires that $a_m B'_m(r) \geq 0$. However, the resulting shape of the BMARS mean function, $f(r, \mathbf{x})$, is limited in this case. Yet another approach would be to work with different basis functions altogether, such as Bernstein polynomials (Chang et al., 2007). However, multivariable nonparametric regression where one of the dimensions is required to be non-decreasing is going to be a difficult problem with any basis functions, for the reasons just described. If the basis functions have a tensor product form, we will face the combinatorial problem. If the basis functions are required to be non-decreasing in the functional variable, the possible shapes of the non-decreasing functions are likely to be limited. Instead, we opt to model the warping functions in a transformed space, according to a transformation that results in monotonicity. Specifically, we model in the log derivative space.

We describe the emulator in full in Section 5.1, including possible methods of inference. In Section 5.2, we emulate the HYDRA simulator and evaluate the performance. In Section 5.3, we conclude and discuss possible extensions.

5.1 Adaptive Spline Emulator

Let $y_i(r)$ denote the output from the i^{th} computer model run for $i = 1, \ldots, n$. The output is a function of r, though r may have a different range for different model runs. We model $y_i(r)$ as the composition of two random functions,

$$y_i(r) = y_i(r^*) \circ w_i^{-1}(r). \tag{5.1}$$

Here, $y_i(r^*)$ represents the aligned data and $w_i^{-1}(r)$ is the non-decreasing warping function that transforms the aligned data to the original scale. If $r \in T$, $r^* \in T^*$, and $y_i(r) \in \mathbb{R}$, then we have that $w : T^* \to T$, $y(r) : T \to \mathbb{R}$, and $y(r^*) : T^* \to \mathbb{R}$.

Note that Equation 5.1 can be rewritten as

$$y_i(r) \circ w_i(r^*) = y_i(r^*), \tag{5.2}$$

since function composition is associative and $w^{-1} \circ w$ is the identity function.

We then model the aligned data as

$$\begin{aligned} y_i(r^*) &\sim \mathrm{N}\left(\eta_y(r^*, \mathbf{x}_i), \sigma_y^2\right) \\ \eta_y(r^*, \mathbf{x}_i) &= a_0^y + \sum_{m=1}^{M_y} a_m^y B_m^y(r^*) B_m^y(\mathbf{x}_i), \end{aligned} \tag{5.3}$$

which is the BMARS model introduced in Chapter 2. The warping functions are modeled on the log derivative scale as

$$\begin{aligned} log\left(\frac{d}{dr^*} w_i(r^*)\right) \equiv v_i(t^*) &\sim \mathrm{N}\left(\eta_v(r^*, \mathbf{x}_i), \sigma_v^2\right) \\ \eta_v(r^*, \mathbf{x}_i) &= a_0^v + \sum_{m=1}^{M_v} a_m^v B_m^v(r^*) B_m^v(\mathbf{x}_i), \end{aligned}$$

which is another BMARS model. We can rewrite w in terms of v as

$$w_i(r^*) = r_i^0 + \int_{r_0^*}^{r^*} \exp\{v_i(s)\} ds$$

where r_0^* is the infimum of R^* and r_i^0 is the constant that vanishes when taking derivatives, which can be interpreted as the start time. This integral has an analytical solution, given in Appendix C. Finally, we model

$$r_i^0 \sim \mathrm{N}\left(\eta_r(\mathbf{x}_i), \sigma_r^2\right)$$
$$\eta_r(\mathbf{x}_i) = a_0^r + \sum_{m=1}^{M_r} a_m^r B_m^r(\mathbf{x}_i),$$

which is a third BMARS model with scalar, rather than functional, response.

The warping functions $w_i(r^*)$ would be treated as latent functions if we did not have landmarks. The landmarks are realizations of these functions, which we denote as $\mathbf{w}_i = \{w_i(r_1^*), \ldots, w_i(r_L^*)\}$ where $L = 4$ is the number of landmarks. Some simplifications result from assuming that the warping functions are linear between the landmarks. We can discretize the derivative and integral given above. If we let $\mathbf{v}_i = \{v_{i1}, \ldots, v_{iL-1}\}$ where $v_{ij} = \log\left(\frac{w_i(r_{j+1}^*) - w_i(r_j^*)}{r_{j+1}^* - r_j^*}\right)$ for $j = 1, \ldots, L-1$, then $\mathbf{v}_i$ is a coarsely discretized version of the function $v_i(r^*)$, which we can model. Then, to transform to the space of w, we need only $w_{ij} = r_i^0 + \sum_{l=2}^{j} \exp\{v_{ij}\}(r_l^* - r_{l-1}^*)$, from which we obtain $w_i(r^*)$ by interpolating between the values of $\mathbf{w}_i$. Hence, the coarseness of the discretization of the derivative does not matter, and the warping functions are identifiable.

To perform inference, there are two fundamental questions that need to be addressed. First, we must consider whether the model for the warping functions should be influenced by anything other than the landmarks. The second question arises again because we have the landmarks for the observed curves, so we can use Equation 5.2 to get the corresponding aligned data. With aligned data, the second question is whether the model for the aligned data should depend on the predicted warping functions. We consider two approaches to answering these questions: the full Bayesian approach and the modularized Bayesian approach.

5.1.1 Full Bayesian Inference

Let $\mathbf{y}_i^* = \{y_i(r_1^*), \ldots, y_i(r_m^*)\}$ denote a vector of m realizations of the aligned curve i. Given $\mathbf{v}_i$, r_i^0 and $\mathbf{y}_i^*$, we can reconstruct the function $y_i(r)$ on a grid. Let η_y denote the random function (and its associated parameters) given in Equation 5.3, with η_v and η_r similarly defined. Let $\mathbf{X}$ denote the design matrix with n rows and nine columns. Then the likelihood is given by

$$f(\mathbf{y}_1^*, \ldots, \mathbf{y}_n^*, \mathbf{v}_1, \ldots, \mathbf{v}_n, r_1^0, \ldots, r_n^0 | \eta_y, \eta_v, \eta_r, \sigma_y^2, \sigma_v^2, \sigma_r^2, \mathbf{X})$$

which we rewrite as $f(y^*, v, r^0|\theta, \mathbf{X}) = f(y^*|v, r^0, \theta, \mathbf{X}) f(v|r^0, \theta, \mathbf{X}) f(r^0|\theta, \mathbf{X})$, where (v, r^0) can be used to make w. The posterior is given by

$$\pi(\theta|y^*, v, r^0, \mathbf{X}) \propto f(y^*|v, r^0, \theta, \mathbf{X}) f(v|r^0, \theta, \mathbf{X}) f(r^0|\theta, \mathbf{X}) \pi(\theta).$$

Then the full conditionals are

$$\begin{aligned}
[\eta_y] &\propto \mathrm{N}(y \circ w | \eta_y, \sigma_y^2) \pi(\eta_y) \\
[\eta_v] &\propto \mathrm{N}(y \circ w | \eta_y, \sigma_y^2)\, \mathrm{N}(v | \eta_v, \sigma_v^2) \pi(\eta_v) \\
[\eta_r] &\propto \mathrm{N}(y \circ w | \eta_y, \sigma_y^2)\, \mathrm{N}(r^0 | \eta_r, \sigma_r^2) \pi(\eta_r).
\end{aligned}$$

Hence, updates to the warping function parameters require a likelihood calculation for the warped data. Thus the warping functions depend on the landmarks and the fit of the warped data.

5.1.2 Modularized Bayesian Inference

As opposed to the full Bayesian approach, a possible modular approach makes the warping functions only depend on the landmarks, and the model for the warped data only depend on the data warped by the observed landmarks. The full conditionals would be

$$
\begin{aligned}
[\eta_y] &\propto \mathrm{N}(y^* | \eta_y, \sigma_y^2)\pi(\eta_y) \\
[\eta_v] &\propto \mathrm{N}(v | \eta_v, \sigma_v^2)\pi(\eta_v) \\
[\eta_r] &\propto \mathrm{N}(r^0 | \eta_r, \sigma_r^2)\pi(\eta_r),
\end{aligned}
$$

where the three models could be fit independently. Also, warping of the training data can be done once using the observed landmarks, and does not need to be repeated as part of the MCMC algorithm. Because of the large number of model runs, warping the training data is the most time consuming part of the full Bayesian MCMC algorithm.

5.1.3 Computation

The modular Bayesian inference can be done using the BASS package directly, where three BMARS models are fit independently using three different responses but the same input combinations. The first model has the aligned functional data, evaluated on a common grid, as the response. The second model has the log of the differences between the landmarks as the response. The third model has the first landmark as the response.

The full Bayesian inference can be done using a few relatively simple manipulations of the BASS package. In a given RJMCMC iteration, the model for the first landmarks is updated conditional on the other two models. Then the model for the log of the differences between the landmarks is updated conditional on the other two models. Finally, the warping functions resulting from these two models are used to update the aligned data, and the aligned data model is updated.

The modular approach to inference can be completed in a matter of minutes, while the full Bayesian approach requires nearly 24 hours. The major bottleneck of the full Bayesian computation is warping the training data in each MCMC iteration. If we had less training data, the difference in timing would not be as drastic.

5.2 HYDRA Emulation

5.2.1 Prediction Performance

We hold out 100 HYDRA simulations to use for evaluation of the predictive performance of the emulator. To assess uncertainty of an emulator prediction, we build a 95% pointwise probability region for an unwarped prediction based on samples from the posterior predictive distribution. These samples vary in both the amplitude and phase dimensions. To get the 95% pointwise probability region, we build a two dimensional kernel density estimate of the discretized posterior predictive samples, from which we extract the 95% contour. Figure 5.4 shows 25 predictions with uncertainty under the full Bayesian approach as well as the true curves.

The shape of the probability regions in Figure 5.4 demonstrates an aspect of our warping model that may not be desirable. Because the warping functions are modeled with additive homoskedastic error on the log derivative scale, the error becomes multiplicative and heteroskedastic after exponentiating and integrating. If we disregard the error in that space, essentially treating σ_v^2 as a nuisance parameter used only for fitting, we get the posterior predictive regions shown in Figure 5.5.

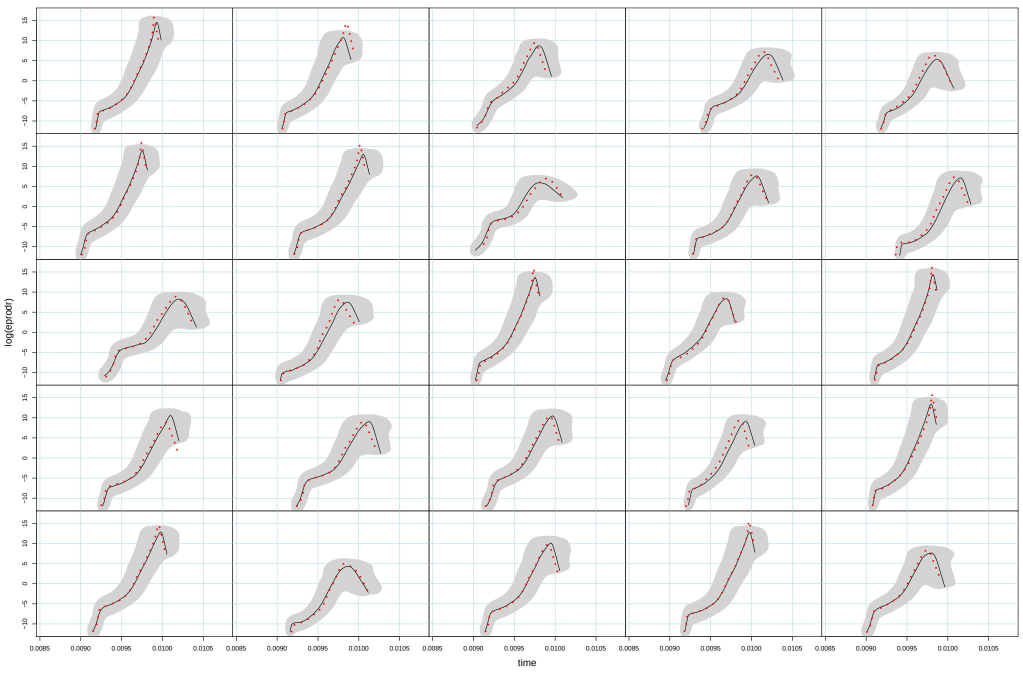


Figure 5.4: Prediction of 25 holdout simulations under the full Bayesian model. The shaded regions are 95% pointwise probability regions. The solid line is the posterior mean prediction. The dotted line is the true simulator output.

Predictions using the modular approach are shown in Figure 5.6, and in Figure 5.7 setting $\sigma_v^2 = 0$. The prediction performance is similar to the full Bayes prediction. The modular approach is preferred based on metrics like posterior predictive root mean squared error of the landmark predictions (1.94×10^{-5} versus 2.83×10^{-5}) and warped data predictions (0.54 versus 0.60). This is not surprising, since these are basically the metrics used to fit the independent pieces of the modularized model, while the full Bayes model considers them jointly.

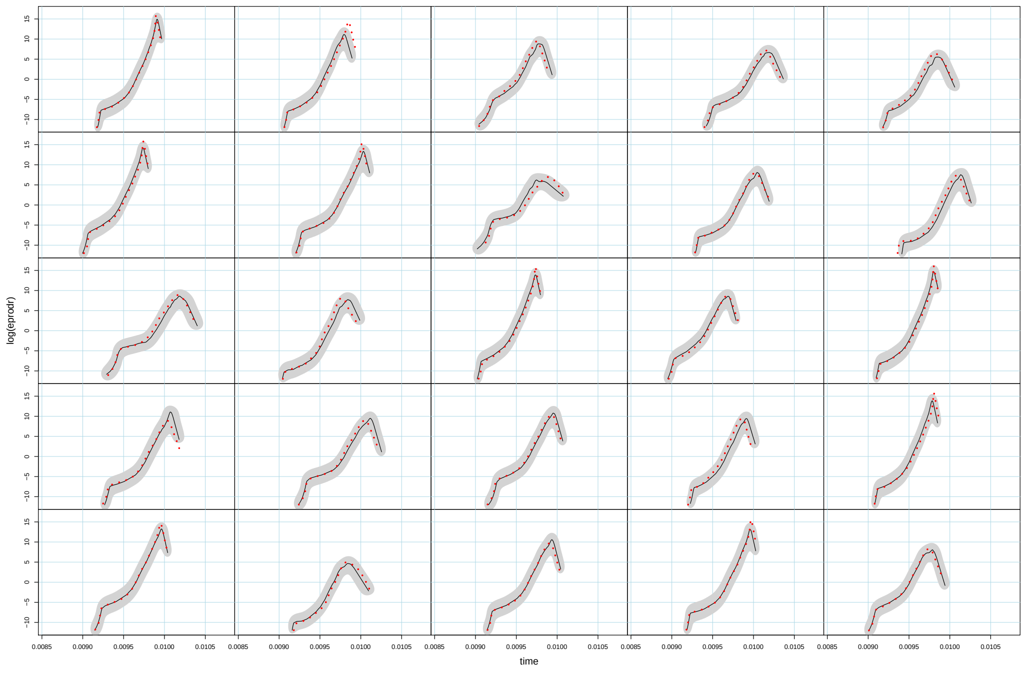


Figure 5.5: Prediction of 25 holdout simulations under the full Bayesian model while excluding variance in log derivative warping functions. The shaded regions are 95% pointwise probability regions. The solid line is the posterior mean prediction. The dotted line is the true simulator output.

5.2.2 Sensitivity Analysis

While the combination of the three BMARS models is necessary for prediction, the three pieces of the emulator can be used for sensitivity analysis in a modular way. That is, we can perform separate sensitivity analyses for the three BMARS models that make up the emulator. The most interesting sensitivity analysis is likely to be the one for the model of the aligned curves, as it explains the general differences in shape. The sensitivity analysis for the first landmark model provides an explanation for the variation in start times. The sensitivity analysis of the log derivative of the warping functions explains why some curves are longer than others.

Figure 5.8 shows the functional Sobol' decomposition of η_y, the model for

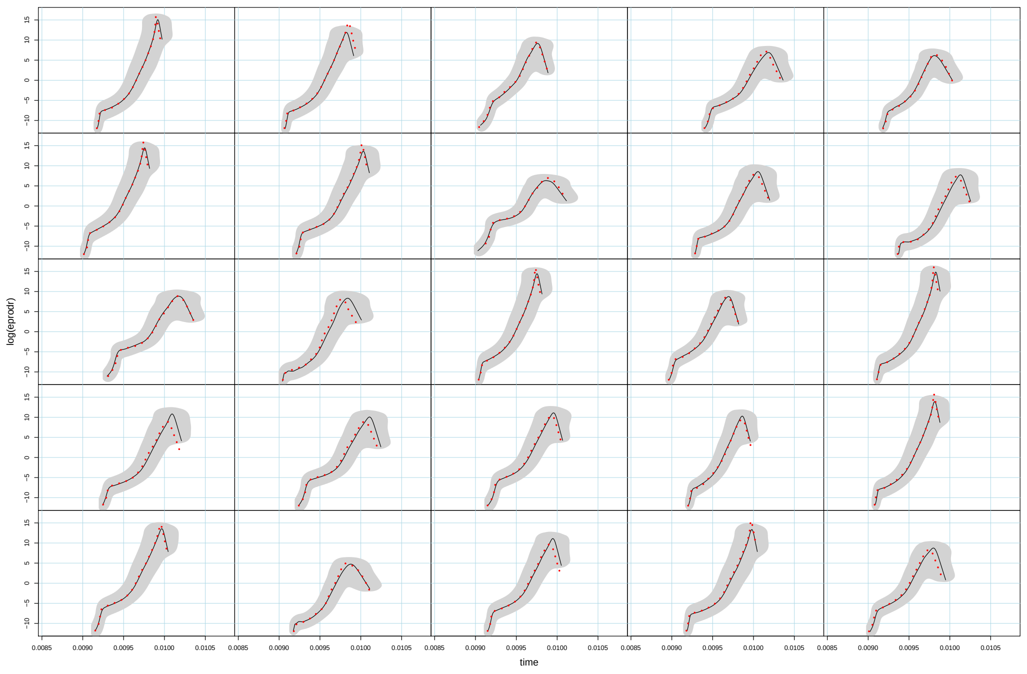


Figure 5.6: Prediction of 25 holdout simulations under the modularized Bayesian model. The shaded regions are 95% pointwise probability regions. The solid line is the posterior mean prediction. The dotted line is the true simulator output.

the warped data. These indicate that most of the variance in the early to middle time period is due to variation in input three, which controls the density of the fuel in the capsule. There is a decrease in the total variance about 75% of the way through the warped time ($r^* = 0.75$), as many of the curves overlap around that point. Variation after that point is due to changes in the shape, timing, and amount of laser stimulus to the fuel capsule.

Figure 5.9 shows the sensitivity analysis of the model for the start time, η_r, and indicates that most of the variation is due to changes in inputs six and eight, which correspond to timing variables governing the laser power. Input three is also somewhat important in explaining the start time variation.

Figure 5.10 shows the sensitivity analysis of the model for the log derivative of the warping functions, η_v. The fuel density (input 3) is again a primary source of

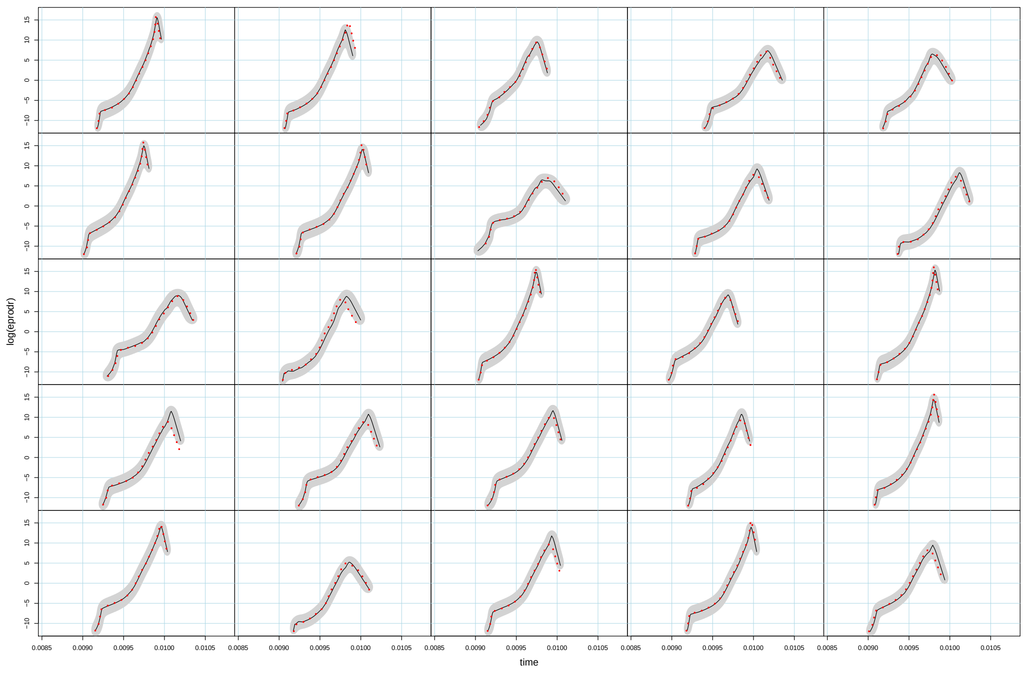


Figure 5.7: Prediction of 25 holdout simulations under the modularized Bayesian model while excluding variance in log derivative warping functions. The shaded regions are 95% pointwise probability regions. The solid line is the posterior mean prediction. The dotted line is the true simulator output.

variation in the early portion of the warped time, which corresponds to the placement of the second landmark (i.e., the difference between the first and second landmark). Variation in the placement of the third landmark is minimal, but is driven primarily by one of the timing inputs. Variation in the final landmark location is driven by a combination of shape and timing variables.

One of the more important findings of this sensitivity analysis is that we have determined which variables are unimportant. Inputs seven and nine do not explain significant variation in any of the three BMARS models that make up the emulator. These variables correspond to particular aspects of timing and shape that appear to be less influential in these ranges of parameters.

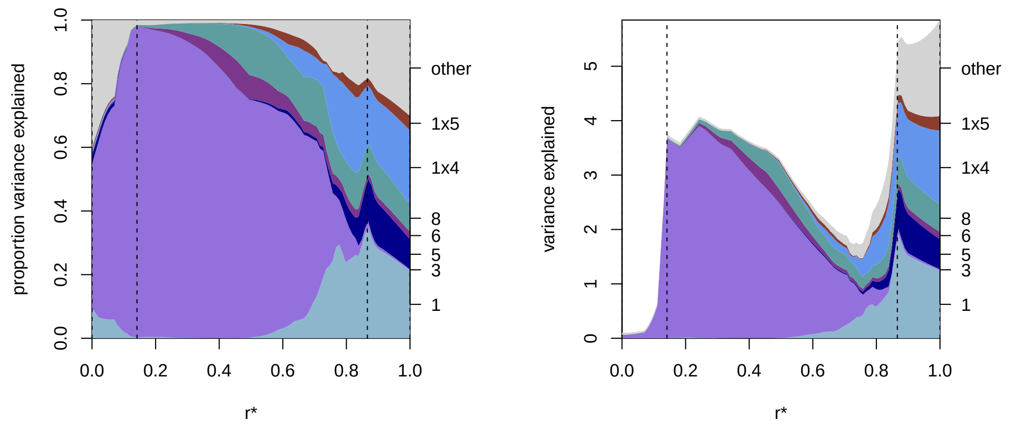


Figure 5.8: Sobol' indices for the warped data model, η_y. The left is a functional pie chart that shows the partitioned variance as a function of r^*. The variables and interactions by which it is partitioned are shown on the right axis. The right plot is the actual partitioned variance. Dotted vertical lines indicate the location of landmarks

5.3 Conclusion

We have introduced emulation methods for computer models that output misaligned functional data. We have discussed a fully Bayesian and a modularized Bayesian approach to inference. We have shown how the methods perform using a large number of multi-physics simulations from the HYDRA code. We have also demonstrated a method for performing sensitivity analysis for such a simulator.

Much of our methodology is valid because we could easily extract landmarks from the simulations. For misaligned functions like the ones we encounter here, there are likely to be at least two easily identified landmarks: the start and the end points. If those were the only landmarks we could find, the model for η_v could have scalar rather than functional response, as there would only be one slope to consider.

In cases where the functional response is misaligned but landmarks are too difficult to obtain, the full Bayesian model could be extended in such a way as to

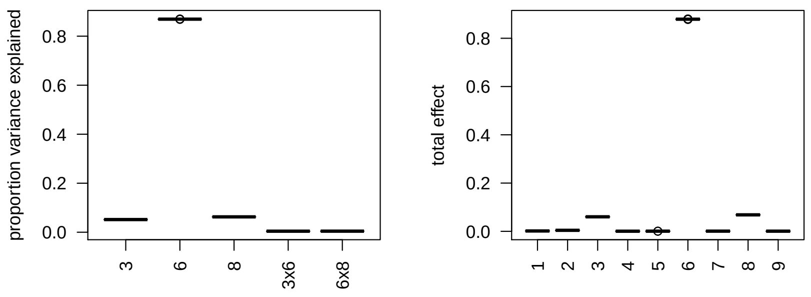


Figure 5.9: Sobol' indices for the start time model η_r. The left shows the proportion of variance explained by the most important effects. The right shows the total sensitivity indices.

propose BMARS warping functions without using landmarks to determine if they are good. Rather, the utility of a warping function would be determined completely by the warped data likelihood. This presents some challenges, as the BMARS model for the warping functions would be a latent process without data to drive it, and the latent process could be difficult to identify because of the flexibility of the model for the warped data.

When landmarks are available, there is a clear philosophical argument that the modular approach to inference is justified. The argument is that the warping functions should not be influenced by anything other than the landmarks and that the functional model in the warped space should not be influenced by anything other than the true warping functions. The benefits of the modularized approach are also apparent in the decreased computational burden.

A shortfall of the proposed methodology is the inclusion of an error term in the model for the log derivative of the warping functions. This leads to large heteroskedastic errors in predicted functions because the homoskedastic error is integrated. An approach to resolve this issue is to replace the model $v_i(t^*) \sim$

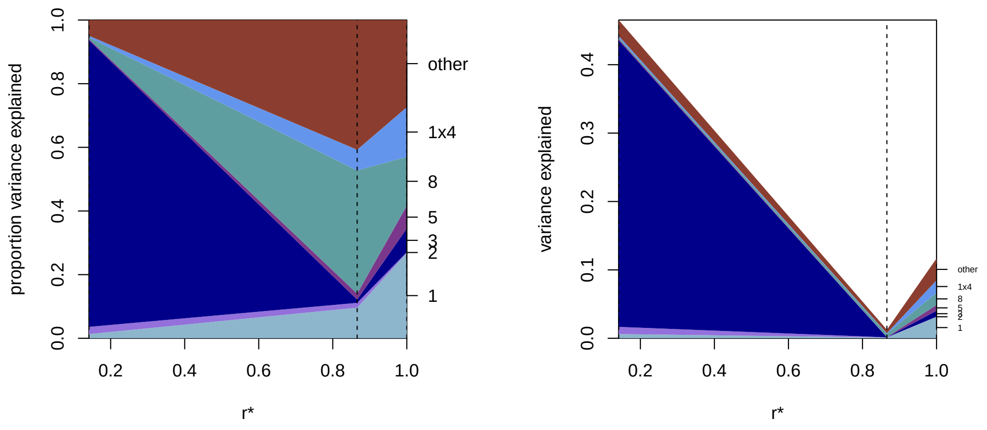


Figure 5.10: Sobol' indices for the model for the log derivative of the warping functions, η_v. The left is a functional pie chart that shows the partitioned variance as a function of r^*. The variables and interactions by which it is partitioned are shown on the right axis. The right plot is the actual partitioned variance. Dotted vertical lines indicate the location of landmarks. The plot starts at the first landmark.

$\mathrm{N}(\eta_v(r^*, \mathbf{x}_i), \sigma_v^2)$ with $w_i(t^*) \sim \mathrm{N}\left(r_i^0 + \int_{r_0^*}^{r^*} \exp\{\eta_v(s, \mathbf{x}_i)\}ds, \sigma_w^2\right)$, so that the error is not integrated. Essentially, this amounts to introducing a link function into the BMARS model. Such a nonlinear transformation of the linear BMARS mean function would complicate the calculations that are otherwise simple from having a Gaussian linear model with Gaussian coefficients. Certainly, an isotonic model for the warping functions that does not include any integration would be preferred, but more research is necessary to determine usable nonlinear isotonic functional regression models of many variables.

Chapter 6

Conclusion

We have extended the capability of Bayesian adaptive spline models for emulation, sensitivity analysis, and calibration for computer models with functional response. A key component of all of our developments has been the ability to handle large amounts of data. We have detailed how adaptive spline models for functional data are computationally tractable. We have shown that these models have closed form global sensitivity indices that can be explored functionally. We have introduced priors to minimize overfitting of various types as well as tempering to improve posterior sampling. We have introduced an alternative adaptive spline model for functional data in a reduced dimensional space, and shown how it can assimilate massive amounts of data to emulate and calibrate a complex simulator. We have introduced a way to incorporate categorical variables in Bayesian adaptive spline models. Further, we have detailed a possible way to emulate a computer model with misaligned functional response using adaptive splines to model the aligned data and warping functions.

All methodological improvements were made in an effort to emulate computer models currently being used to further modern scientific understanding. We have demonstrated that the emulators proposed can be used for important uncertainty quantification tasks. In the end, emulation is merely an application for nonlinear multiple regression. As such, the methods developed here are not limited to use for emulation. They can be utilized to build nonparametric regression models of many or few variables, with scalar or functional response.

A number of possible extensions could be pursued. Computationally, investigation of variable selection methods for choosing basis functions could allow us to go without reversible jump MCMC, as in Clyde et al. (2011). Alternative model search techniques, like those in Hans et al. (2007), could also be computationally beneficial. With regard to functional data analysis, these models could be extended to incorporate functional inputs, perhaps by summarizing the inputs on a set of basis functions as in Mondal (2012). Another extension is emulators for models that output multiple types of functional data, which is a common scenario. This could be done by representing the multiple outputs on a single set of basis functions, like empirical orthogonal functions, and modeling the weights on those basis functions using adaptive splines, similar to the approach in Chapter 4. Models for multiple scalar responses could also be of interest. Friedman (1991a) proposes a single set of basis functions with different sets of coefficients for this circumstance. Further development of these models for use in nonlinear times series analysis, as in Lewis & Stevens (1991), or with non-Gaussian likelihood, as in Holmes & Denison (2003), could make them more useful for general purpose non-parametric regression.

Appendix A

Sobol Decomposition of the Friedman Function

We obtain the Sobol' decomposition of the function

$$f(\mathbf{x}) = 10\sin(2\pi x_1 x_2) + 20\left(x_3 - \frac{1}{2}\right)^2 + 10x_4 + 5x_5$$

using the the approach where we treat x_1 as another input (augmentation approach) and where we get other sensitivity indices as a function of x_1 (functional approach).

A.1 Augmentation Approach

The overall mean is given by

$$\begin{aligned} f_0 &= \int_0^1 \cdots \int_0^1 f(\mathbf{x}) d\mathbf{x} \\ &= \underbrace{\int_0^1 \int_0^1 10\sin(2\pi x_1 x_2) dx_1 dx_2}_{a_1} + \underbrace{\int_0^1 20\left(x_3 - \frac{1}{2}\right)^2 dx_3}_{a_2 = 5/3} + \underbrace{\int_0^1 10x_4 dx_4}_{a_3 = 5} + \underbrace{\int_0^1 5x_5 dx_5}_{a_4 = 5/2} \end{aligned}$$

where

$$\begin{aligned}
a_1 &= \int_0^1 \frac{10\sin^2(\pi x)}{\pi x} dx \\
&= \frac{5}{\pi}\left[\log(2\pi) + \gamma - Ci(2\pi)\right] \\
Ci(x) &= \gamma + \log(x) + \sum_{k=1}^{\infty} \frac{(-x^2)^k}{2k(2k)!} \\
\Rightarrow a_1 &= -\frac{5}{\pi}\sum_{k=1}^{\infty} \frac{(-4\pi^2)^k}{2k(2k)!}
\end{aligned}$$

The main effects are given by

$$\begin{aligned}
f_1(x_1) &= \int_0^1 10\sin(2\pi x_1 x_2) dx_2 + a_2 + a_3 + a_4 - f_0 \\
&= \frac{10\sin^2(\pi x_1)}{\pi x_1} - a_1 \\
f_2(x_2) &= \frac{10\sin^2(\pi x_2)}{\pi x_2} - a_1 \\
f_3(x_3) &= 20\left(x_3 - \frac{1}{2}\right)^2 - a_2 \\
f_4(x_4) &= 10x_4 - a_3 \\
f_5(x_5) &= 5x_5 - a_4
\end{aligned}$$

and the interaction effect is given by

$$f_{12}(x_1, x_2) = 10\sin(2\pi x_1 x_2) - \frac{10\sin^2(\pi x_1)}{\pi x_1} - \frac{10\sin^2(\pi x_2)}{\pi x_2} + a_1.$$

Then the overall variance is given by

$$
\begin{aligned}
Var(f(\mathbf{x})) &= \int_0^1 \dots \int_0^1 f^2(\mathbf{x}) d\mathbf{x} \\
&= \int_0^1 \dots \int_0^1 \underbrace{100\sin^2(2\pi x_1 x_2)}_{50-25Si(4\pi)/2\pi} + \underbrace{400\left(x_3 - \frac{1}{2}\right)^4}_{5} + \underbrace{100x_4^2}_{100/3} + \underbrace{25x_5^2}_{25/3} \\
&+ \underbrace{400\sin(2\pi x_1 x_2)\left(x_3 - \frac{1}{2}\right)^2}_{2a_1a_2} + \underbrace{200x_4\sin(2\pi x_1 x_2)}_{2a_1a_3} + \underbrace{100x_5\sin(2\pi x_1 x_2)}_{2a_1a_4} \\
&+ \underbrace{400x_4\left(x_3 - \frac{1}{2}\right)^2}_{2a_2a_3} + \underbrace{200x_5\left(x_3 - \frac{1}{2}\right)^2}_{2a_2a_4} + \underbrace{100x_4x_5}_{2a_3a_4} d\mathbf{x} - f_0^2 \\
&= 50 - 25Si(4\pi)/2\pi + 5 + 125/3 \\
&\qquad + 2(a_1a_2 + a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4 + a_3a_4) - f_0^2
\end{aligned}
$$

where the underbraces give the quantity after integration and

$$
Si(x) = \int_0^x \frac{\sin t}{t} dt = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)(2k-1)!}.
$$

The variances for the main effects are given by

$$
\begin{aligned}
Var(f_1(x_1)) &= \int_0^1 f_1^2(x_1)dx_1 \\
&= \int_0^1 \underbrace{\frac{100\sin^4(\pi x_1)}{\pi^2 x_1^2}}_{50/\pi[2Si(2\pi)-Si(4\pi)]} + \underbrace{a_1^2 - 20a_1\frac{\sin^2(\pi x_1)}{\pi x_1}}_{a_1^2-2a_1^2=-a_1^2} dx_1
\end{aligned}
$$

$$
\begin{aligned}
Var(f_2(x_2)) &= Var(f_1(x_1)) \\
Var(f_3(x_3)) &= \int_0^1 \underbrace{400\left(x_3-\frac{1}{2}\right)^4}_{5} + \underbrace{a_2^2 - 40a_2\left(x_3-\frac{1}{2}\right)^2}_{-a_2^2} dx_3 \\
Var(f_4(x_4)) &= \int_0^1 \underbrace{100x_4^2}_{100/3} + \underbrace{a_3^2 - 20a_3x_4}_{-a_3^2} dx_4 \\
Var(f_5(x_5)) &= \int_0^1 \underbrace{25x_5^2}_{25/3} + \underbrace{a_4^2 - 10a_4x_5}_{-a_4^2} dx_5.
\end{aligned}
$$

The variance for the interaction is given by

$$
\begin{aligned}
Var(f_{12}(x_1,x_2)) &= \int_0^1\int_0^1 f_{12}^2(x_1,x_2)dx_1dx_2 \\
&= \int_0^1\int_0^1 \underbrace{100\sin^2(2\pi x_1x_2)}_{50-25Si(4\pi)/2\pi} + \underbrace{\frac{100\sin^4(\pi x_1)}{\pi^2x_1^2} + \frac{100\sin^4(\pi x_2)}{\pi^2x_2^2} + a_1^2}_{100/\pi[2Si(2\pi)-Si(4\pi)]+a_1^2} \\
&\underbrace{-2\frac{10\sin^2(\pi x_1)}{\pi x_1}10\sin(2\pi x_1x_2) - 2\frac{10\sin^2(\pi x_2)}{\pi x_2}10\sin(2\pi x_1x_2)}_{-200/\pi[2Si(2\pi)-Si(4\pi)]} \\
&+ \underbrace{2a_1 10\sin(2\pi x_1x_2)}_{2a_1^2} + \underbrace{2\frac{10\sin^2(\pi x_1)}{\pi x_1}\frac{10\sin^2(\pi x_2)}{\pi x_2}}_{2a_1^2} \\
&\underbrace{-2\frac{10\sin^2(\pi x_1)}{\pi x_1}a_1 - 2\frac{10\sin^2(\pi x_2)}{\pi x_2}a_1}_{-4a_1^2} dx_1dx_2 \\
&= 50 - 25Si(4\pi)/2\pi - 100/\pi[2Si(2\pi) - Si(4\pi)] + a_1^2
\end{aligned}
$$

A.2 Functional Approach

As a function of x_1, the main effects are

$$\begin{aligned}
f_0(x_1) &= \frac{10\sin^2(\pi x_1)}{\pi x_1} + a_2 + a_3 + a_4 \\
f_2(x_1, x_2) &= 10\sin(2\pi x_1 x_2) - \frac{10\sin^2(\pi x_1)}{\pi x_1} \\
f_3(x_1, x_3) &= f_3(x_3) \\
f_4(x_1, x_4) &= f_4(x_4) \\
f_5(x_1, x_5) &= f_5(x_5).
\end{aligned}$$

Then the variance functions are

$$\begin{aligned}
D(x_1) &= \int_0^1 \cdots \int_0^1 f^2(\mathbf{x}) d\mathbf{x}_{-1} \\
&= 50 - \frac{25\sin(4\pi x_1)}{2\pi x_1} + 5 + 125/3 + 2(a_2 + a_3 + a_4)\frac{10\sin^2(\pi x_1)}{\pi x_1} \\
&+ 2a_2a_3 + 2a_2a_4 + 2a_3a_4 - f_0^2(x_1) \\
D_2(x_1) &= \int_0^1 f_2^2(x_1, x_2) dx_2 \\
&= 50 - 25\sin(4\pi x_1)/(2\pi x_1) - \frac{100\sin^4(\pi x_1)}{\pi^2 x_1^2} \\
D_3(x_1) &= 5 - a_2^2 \\
D_4(x_1) &= 100/3 - a_3^2 \\
D_5(x_1) &= 25/3 - a_4^2.
\end{aligned}$$

Appendix B

Sobol Decomposition of the BMARS Model in EOF Space

B.1 Sobol' Decomposition

If we have a function $f(\mathbf{x})$ where $\mathbf{x} = (x_1, \ldots, x_p)$, we decompose it into

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^{p} f_i(x_i) + \sum_{i=1}^{p} \sum_{j>i} f_{ij}(x_i, x_j) + \cdots + f_{1\ldots p}(x_1, \ldots, x_p) \tag{B.1}$$

where all terms added above are orthogonal. In addition, all terms except f_0 are centered at zero. This is achieved by building each term such that

$$f_0 = \int f(\mathbf{x}) d\mathbf{x} \tag{B.2}$$

$$f_i(x_i) = \int f(\mathbf{x}) d\mathbf{x}_{-i} - f_0 \tag{B.3}$$

$$f_{ij}(x_i, x_j) = \int f(\mathbf{x}) d\mathbf{x}_{-ij} - f_i(x_i) - f_j(x_j) - f_0 \tag{B.4}$$

as so on, where integrals are over the bounds of each x_i. Without loss of generality, we assume that the bounds are zero and one.

We are interested in partitioning $Var(f(\mathbf{x}))$ into variance from each main effect and interaction. First, note that if we assume that x_i is a random variable with uniform distribution $f_i(x_i) = E(f(\mathbf{x})|x_i) - f_0$ and $f_0 = E(f(\mathbf{x}))$. Also note that $Var(f(\mathbf{x})) = E(f^2(\mathbf{x})) - E(f(\mathbf{x}))^2$. Now, squaring Equation B.1 and integrating, we obtain

$$E(f^2(\mathbf{x})) = f_0^2 + \sum_{i=1}^{p} \int f_i^2(x_i)dx_i + \sum_{i=1}^{p}\sum_{j>i} \int f_{ij}^2(x_i, x_j)dx_i dx_j + \dots$$
$$+ \int f_{1\dots p}^2(x_1, \dots, x_p)d\mathbf{x}$$

which lacks any crossproduct terms because all the terms are orthogonal. We could also write each term above as a variance, i.e., $Var(f_i(x_i)) = \int f_i^2(x_i)dx_i - 0$. We subtract 0 because $(\int f_i(x_i)dx_i)^2 = 0$. This is the case for each term. Thus,

$$E(f^2(\mathbf{x})) = f_0^2 + \sum_{i=1}^{p} Var(f(x_i)) + \sum_{i=1}^{p}\sum_{j>i} Var(f(x_i, x_j)) + \dots$$
$$+ Var(f_{1\dots p}(x_1, \dots, x_p))$$

and we have that

$$Var(f(\mathbf{x})) = \sum_{i=1}^{p} Var(f(x_i)) + \sum_{i=1}^{p}\sum_{j>i} Var(f(x_i, x_j)) + \dots + Var(f_{1\dots p}(x_1, \dots, x_p)),$$

thus decomposing the variance of f into variance due to each main effect and interaction. Note that the way we construct the main effects and interactions in Equations B.2 through B.4 is sequential, so that a two way interaction function is the effect after taking into account the two main effects.

We outline some practical considerations for obtaining the variance decomposition above, as discussed in Chen et al. (2005). First, if $\mathbf{u} \subseteq \{1, \ldots, p\}$ of size s and $\mathbf{x_u} = \{x_{u_1}, \ldots, x_{u_s}\}$, then we write the effect $f_u(\mathbf{x_u})$ (interaction if $s > 1$, main effect if $s = 1$) can be written as

$$f_u(\mathbf{x_u}) = \hat{f}_u(\mathbf{x_u}) - \sum_{\mathbf{v} \in \{P(\mathbf{u}) - \mathbf{u} - \emptyset\}} f_v(\mathbf{x_v}) - f_0$$
$$\hat{f}_u(\mathbf{x_u}) = \int f(\mathbf{x}) d\mathbf{x_{-u}}$$

where $P(\mathbf{u})$ is the power set of $\mathbf{u}$. Note that $\hat{f}_u$ denotes the non-centered version of f_u. This recursive definition gives rise (with some algebra) to a simpler formulation only in terms of the non-centered effects,

$$f_u(\mathbf{x_u}) = \sum_{\mathbf{v} \in P(\mathbf{u})} (-1)^{|\mathbf{u}| - |\mathbf{v}|} \hat{f}_v(\mathbf{x_v})$$

where we define $\hat{f}_v(\mathbf{x_v}) = f_0$ if $\mathbf{v} = \emptyset$. With some further algebra, we can see that

$$Var(f_u(\mathbf{x_u})) = \sum_{\mathbf{v} \in \{P(\mathbf{u}) - \emptyset\}} (-1)^{|\mathbf{u}| - |\mathbf{v}|} Var\left(\hat{f}_v(\mathbf{x_v})\right)$$

and it is easy to show that

$$Var\left(\hat{f}_u(\mathbf{x_u})\right) = \int \hat{f}_u^2(\mathbf{x_u}) d\mathbf{x_u} - f_0^2. \tag{B.5}$$

Thus, if we can evaluate Equation B.5 analytically, we can obtain the variance decomposition analytically.

B.2 BASS Sobol' Decomposition

The Sobol' decomposition for the BASS model can be done analytically. The details for obtaining the Sobol' decomposition and functional Sobol' decomposition when inputs are continuous are in Chapter 2. Here, we will describe the details when we have categorical inputs. We will also describe how to get functional sensitivity indices when we use the EOF approach to emulation described in Chapter 4.

First, we review (Chen et al., 2005) that when we use a tensor product basis function approach such that $f(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m \prod_{j=1}^{J_m} h_{jm}(x_{v_{jm}})$, the quantities we need to obtain are $C_i^1 = \int h_{jm}(x_i)dx$ and $C_i^2 = \int h_{jm_1}(x_i)h_{lm_2}(x_i)dx_i$ where $C_i^2 = 1$ if $v_{jm_1} \neq v_{lm_2}$. These quantities are discussed when inputs are continuous in Chapter 2. For categorical inputs we use sums rather than integrals, so that

$$C_{lm}^1 = \frac{1}{|D_i|}\sum_{z\in D_i} 1(z \in D_{lm}) = \frac{|D_{lm}|}{|D_i|}$$
$$C_i^2 = \frac{1}{|D_i|}\sum_{z\in D_i} 1(z \in D_{lm})1(z \in D_{jm}) = \frac{|D_{lm} \cap D_{jm}|}{|D_i|}$$

Now if $\mathbf{u}$ is a set of variable indices,

$$Var(\hat{f}_u(\mathbf{x_u})) = \sum_{m_1=1}^{M}\sum_{m_2=1}^{M} a_{m_1}a_{m_2}\left(\prod_{l\notin\mathbf{u}} C_{lm_1}C_{lm_2}\right)\left(\prod_{l\in\mathbf{u}} C_{lm_1m_2}^2\right) - f_0^2.$$

B.3 Functional Sobol' Decomposition - EOFs

While the functional Sobol' decomposition of a functional BASS model is described in Chapter 2, the model from Chapter 4 is slightly different. We use BASS for modeling weights for EOFs. To get a functional Sobol' decomposition in this case, we need to do a little more work.

First, consider the case where we are interested in getting Sobol' indices for the functional variables like we do the other variables. The function we are decomposing is $f(\mathbf{r}, \mathbf{x}) = \sum_{i=1}^{k} K_i(\mathbf{r}) w_i(\mathbf{x})$ where $\mathbf{r} = (s, t)$. If we define

$$\begin{aligned} \hat{w}_i^u(\mathbf{x_u}) &= \int w_i(\mathbf{x}) d\mathbf{x_{-u}} \\ K_i &= \int K_i(\mathbf{r}) d\mathbf{r} \end{aligned}$$

then we can obtain the overall mean

$$\begin{aligned} f_0 &= \int \int \sum_{i=1}^{k} K_i(\mathbf{r}) w_i(\mathbf{x}) d\mathbf{r} d\mathbf{x} = \sum_{i=1}^{k} \int K_i(\mathbf{r}) d\mathbf{r} \int w_i(\mathbf{x}) d\mathbf{x} \\ &= \sum_{i=1}^{k} K_i w_i^0 \end{aligned}$$

and the non-centered effects as

$$\begin{aligned} \hat{f}_r(\mathbf{r}) &= \sum_{i=1}^{k} K_i(\mathbf{r}) w_i^0 \\ \hat{f}_u(\mathbf{x_u}) &= \sum_{i=1}^{k} K_i \hat{w}_i^u(\mathbf{x_u}) \\ \hat{f}_{ru}(\mathbf{r}, \mathbf{x_u}) &= \sum_{i=1}^{k} K_i(\mathbf{r}) \hat{w}_i^u(\mathbf{x_u}). \end{aligned}$$

Then we need to obtain

$$\begin{aligned} Var\left(\hat{f}_r(\mathbf{r})\right) &= \int \hat{f}_r^2(\mathbf{r}) d\mathbf{r} - f_0^2 \\ &= \sum_{i=1}^{k} \sum_{j=1}^{k} w_i^0 w_j^0 \int K_i(\mathbf{r}) K_j(\mathbf{r}) d\mathbf{r} - f_0^2 \qquad \text{(B.6)} \end{aligned}$$

$$\begin{aligned} Var\left(\hat{f}_u(\mathbf{x_u})\right) &= \int \hat{f}_u^2(\mathbf{x_u})d\mathbf{x_u} - f_0^2 \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} K_iK_j \int \hat{w}_i^u(\mathbf{x_u})\hat{w}_j^u(\mathbf{x_u})d\mathbf{x_u} - f_0^2 \end{aligned} \tag{B.7}$$

$$\begin{aligned} Var\left(\hat{f}_{ru}(\mathbf{r},\mathbf{x_u})\right) &= \int\int \hat{f}_{ru}^2(\mathbf{r},\mathbf{x_u})d\mathbf{r}d\mathbf{x_u} - f_0^2 \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} \int K_i(\mathbf{r})K_j(\mathbf{r})d\mathbf{r} \int \hat{w}_i^u(\mathbf{x_u})\hat{w}_j^u(\mathbf{x_u})d\mathbf{x_u} - f_0^2 \end{aligned} \tag{B.8}$$

which simplifies in Equation B.6 and Equation B.8 when the basis is orthogonal, but not in Equation B.7. Then, the only quantity left to seek an analytical solution for is

$$\begin{aligned} \int \hat{w}_i^u(\mathbf{x_u})\hat{w}_j^u(\mathbf{x_u})d\mathbf{x_u} &= a_{0i}a_{0j} + a_{0i}(w_j^0 - a_{0j}) + a_{0j}(w_i^0 - a_{0i}) \\ &\quad + \sum_{m_i=1}^{M_i}\sum_{m_j=1}^{M_j} a_{mi}a_{mj}\left(\prod_{l\notin\mathbf{u}} C_{lmi}^1C_{lmj}^1\right)\left(\prod_{l\in\mathbf{u}} C_{lmij}^2\right). \end{aligned}$$

Appendix C

Integral of Exponentiated BMARS Model

Here we will show that the integral of an exponentiated BMARS with respect to the functional variable can be obtained analytically. The specific integral in context of the warping functions introduced in Chapter 5 is

$$w_i(r^*) = r_i^0 + \int_{r_0^*}^{r^*} \exp\left\{ a_0 + \sum_{m=1}^{M} a_m B_m(\mathbf{x}_i) B_m(u) + \epsilon_i \right\} du \tag{C.1}$$

$$= r_i^0 + \exp\{a_0 + \epsilon_i\} \int_{r_0^*}^{r^*} \exp\left\{ \sum_{m=1}^{M} a_m B_m(\mathbf{x}_i) B_m(u) \right\} du \tag{C.2}$$

where $B_m(s) = g_m[s_m(u - t_m)]_+$. Without loss of generality, assume that the basis functions are ordered such that $t_1 < t_2 < \cdots < t_M$ and that $t_j < r^* < t_{j+1}$. We can rewrite Equation C.2 as

$$\begin{aligned}
w_i(r^*) &= r_i^0 + h_0 \int_{r_0^*}^{r^*} \exp\left\{ \sum_{m=1}^{M} h_{mi} B_m(u) \right\} du && \text{(C.3)} \\
&= r_i^0 + h_0 \left[\int_{r_0^*}^{t_1} \exp\left\{ \sum_{m \in A_1} h_{mi} B_m^*(u) \right\} du + \sum_{l=2}^{j} \int_{t_{l-1}}^{t_l} \exp\left\{ \sum_{m \in A_l} h_{mi} B_m^*(u) \right\} du \right. \\
&\qquad \left. + \int_{t_j}^{r^*} \exp\left\{ \sum_{m \in A_{j+1}} h_{mi} B_m^*(u) \right\} du \right] && \text{(C.4)}
\end{aligned}$$

where $B_m^*(u) = g_m s_m(u - t_m)$. Here, $A_1 = \{m \in 1, \ldots, M : s_m = -1\}$ and $A_l = \{m \in 1, \ldots, l-1 : s_m = 1\} \cup \{m \in l, \ldots, M : s_m = -1\}$ for $l = 2, \ldots, j+1$.

To complete the integral, we have that

$$\begin{aligned}
&\int_{\alpha_1}^{\alpha_2} \exp\left\{ \sum_{m \in A} h_{mi} B_m^*(u) \right\} du \\
&= \int_{\alpha_1}^{\alpha_2} \exp\left\{ \sum_{m \in A} h_{mi} g_m s_m (u - t_m) \right\} du && \text{(C.5)} \\
&= \int_{\alpha_1}^{\alpha_2} \exp\left\{ u \sum_{m \in A} h_{mi} g_m s_m + \sum_{m \in A} h_{mi} g_m s_m t_m \right\} du && \text{(C.6)} \\
&= \left[\left(\sum_{m \in A} h_{mi} g_m s_m \right)^{-1} \exp\left\{ u \sum_{m \in A} h_{mi} g_m s_m + \sum_{m \in A} h_{mi} g_m s_m t_m \right\} \right]_{u=\alpha_1}^{u=\alpha_2} . && \text{(C.7)}
\end{aligned}$$

Bibliography

Altekar, G., Dwarkadas, S., Huelsenbeck, J. P. & Ronquist, F. (2004). Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics* **20**, 407–415.

Atchadé, Y. F., Roberts, G. O. & Rosenthal, J. S. (2011). Towards optimal scaling of Metropolis-coupled Markov chain Monte Carlo. *Statistics and Computing* **21**, 555–568.

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

Bayarri, M. J., Berger, J. O., Paulo, R., Sacks, J., Cafeo, J. A., Cavendish, J., Lin, C.-H. & Tu, J. (2007b). A framework for validation of computer models. *Technometrics* **49**.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. (1984). *Classification and Regression Trees*. CRC press.

Chakraborty, A., Mallick, B. K., Mcclarren, R. G., Kuranz, C. C., Bingham, D., Grosskopf, M. J., Rutter, E. M., Stripling, H. F. & Drake, R. P. (2013). Spline-based emulators for radiative shock experiments with measurement error. *Journal of the American Statistical Association* **108**, 411–428.

Chang, I.-S., Chien, L.-C., Hsiung, C. A., Wen, C.-C., Wu, Y.-J. et al. (2007). Shape restricted regression with random Bernstein polynomials. In *Complex datasets and inverse problems*, pp. 187–202. Institute of Mathematical Statistics.

Chen, W., Jin, R. & Sudjianto, A. (2005). Analytical variance-based global sensitivity analysis in simulation-based design under uncertainty. *Journal of Mechanical Design* **127**, 875–886.

Chipman, H. A., George, E. I. & McCulloch, R. E. (1998). Bayesian CART model search. *Journal of the American Statistical Association* **93**, 935–948.

Clyde, M. A., Ghosh, J. & Littman, M. L. (2011). Bayesian adaptive sampling for variable selection and model averaging. *Journal of Computational and Graphical Statistics* **20**, 80–101.

Delle Monache, L., Nipen, T., Liu, Y., Roux, G. & Stull, R. (2011). Kalman filter and analog schemes to postprocess numerical weather predictions. *Monthly Weather Review* **139**, 3554–3570.

Denison, D. G., Holmes, C. C., Mallick, B. K. & Smith, A. F. (2002). *Bayesian Methods for Nonlinear Classification and Regression*, volume 386. John Wiley & Sons.

Denison, D. G., Mallick, B. K. & Smith, A. F. (1998a). A Bayesian CART algorithm. *Biometrika* **85**, 363–377.

Denison, D. G., Mallick, B. K. & Smith, A. F. (1998b). Bayesian MARS. *Statistics and Computing* **8**, 337–346.

Earls, C., Hooker, G. et al. (2017). Variational Bayes for functional data registration, smoothing, and prediction. *Bayesian Analysis* **12**, 557–582.

Fog, A. (2015). ***BiasedUrn**: Biased Urn Model Distributions.* R package version 1.07.

Francom, D. (2017). ***BASS**: Bayesian Adaptive Spline Surfaces.* R package version 0.2.2.

Friedman, J. H. (1991a). Adaptive spline networks. In *Advances in Neural Information Processing Systems*, pp. 675–683.

Friedman, J. H. (1991b). Estimating functions of mixed ordinal and categorical variables using adaptive splines. Technical report, DTIC Document.

Friedman, J. H. (1991c). Multivariate adaptive regression splines. *The Annals of Statistics* pp. 1–67.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A. & Rubin, D. B. (2013). *Bayesian Data Analysis.* CRC press.

Gervini, D. & Gasser, T. (2004). Self-modelling warping functions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **66**, 959–971.

Geyer, C. J. (1991). Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface (E. M. Keramides, ed.)*, pp. 156–163. Fairfax Station, Va: Interface Foundation.

Geyer, C. J. & Thompson, E. A. (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference. *Journal of the American Statistical Association* **90**, 909–920.

Gramacy, R. B. & Apley, D. W. (2015). Local Gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics* **24**, 561–578.

Gramacy, R. B., Bingham, D., Holloway, J. P., Grosskopf, M. J., Kuranz, C. C., Rutter, E., Trantham, M., Drake, R. P. et al. (2015). Calibrating a large computer experiment simulating radiative shock hydrodynamics. *The Annals of Applied Statistics* **9**, 1141–1168.

Gramacy, R. B. & Lee, H. K. (2012). Cases for the nugget in modeling computer experiments. *Statistics and Computing* **22**, 713–722.

Gramacy, R. B. & Taddy, M. (2010). Categorical inputs, sensitivity analysis, optimization and importance tempering with `tgp` version 2, an `R` package for treed Gaussian process models. *Journal of Statistical Software* **33**, 1–48.

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82**, 711–732.

Hans, C., Dobra, A. & West, M. (2007). Shotgun stochastic search for "large p" regression. *Journal of the American Statistical Association* **102**, 507–516.

Higdon, D., Gattiker, J., Williams, B. & Rightley, M. (2008). Computer model calibration using high-dimensional output. *Journal of the American Statistical Association* **103**.

Holmes, C. & Denison, D. (2003). Classification with Bayesian MARS. *Machine Learning* **50**, 159–173.

Hung, Y., Joseph, V. R. & Melkote, S. N. (2015). Analysis of computer experiments with functional response. *Technometrics* **57**, 35–44. doi:10.1080/00401706.2013.869263.

Kaufman, C. G., Bingham, D., Habib, S., Heitmann, K. & Frieman, J. A. (2011). Efficient emulators of computer experiments using compactly supported correlation functions, with an application to cosmology. *The Annals of Applied Statistics* pp. 2470–2492.

Kennedy, M. C. & O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society. Series B, Statistical Methodology* pp. 425–464.

Kolda, T. G. (2006). *Multilinear Operators for Higher-order Decompositions.* United States. Department of Energy.

Lamboni, M., Makowski, D., Lehuger, S., Gabrielle, B. & Monod, H. (2009). Multivariate global sensitivity analysis for dynamic crop models. *Field Crops Research* **113**, 312–320.

Lev-Ari, H. et al. (2005). Efficient solution of linear matrix equations with application to multistatic antenna array processing. *Communications in Information & Systems* **5**, 123–130.

Lewis, P. A. & Stevens, J. G. (1991). Nonlinear modeling of time series using multivariate adaptive regression splines (MARS). *Journal of the American Statistical Association* **86**, 864–877.

Liang, F., Paulo, R., Molina, G., Clyde, M. A. & Berger, J. O. (2008). Mixtures of g priors for Bayesian variable selection. *Journal of the American Statistical Association* **103**.

Lichman, M. (2013). UCI machine learning repository.

Liu, F., Bayarri, M., Berger, J. et al. (2009). Modularization in Bayesian analysis, with emphasis on analysis of computer models. *Bayesian Analysis* **4**, 119–150.

Lucas, D. D., Simpson, M. D., Cameron-Smith, P. & Baskett, R. L. (2017). Bayesian inverse modeling of the atmospheric transport and emissions of a controlled tracer release from a nuclear power plant. *Atmospheric Chemistry and Physics Discussions* **2017**, 1–36. doi:10.5194/acp-2017-336.

Ma, S., Racine, J. S. & Yang, L. (2015). Spline regression in the presence of categorical predictors. *Journal of Applied Econometrics* **30**, 705–717.

Maljovec, D., Wang, B., Kupresanin, A., Johannesson, G., Pascucci, V. & Bremer, P.-T. (2013). Adaptive sampling with topological scores. *International Journal for Uncertainty Quantification* **3**.

Mondal, A. (2012). *Bayesian Uncertainty Quantification for Large Scale Spatial Inverse Problems*. Ph.D. thesis, Texas A & M University.

Neal, R. M. (1996). Sampling from multimodal distributions using tempered transitions. *Statistics and computing* **6**, 353–366.

Nott, D. J., Kuk, A. Y. & Duc, H. (2005). Efficient sampling schemes for Bayesian MARS models with many predictors. *Statistics and Computing* **15**, 93–101.

Oakley, J. E. & O'Hagan, A. (2004). Probabilistic sensitivity analysis of complex models: a Bayesian approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **66**, 751–769.

Pannell, D. J. (1997). Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural economics* **16**, 139–152.

Parodi, A., Patriarca, M., Sangalli, L., Secchi, P., Vantini, S. & Vitelli, V. (2015). *fdakma: Functional Data Analysis: K-Mean Alignment*. R package version 1.2.1.

Peterson, J., Humbird, K., Field, J., Brandon, S., Langer, S., Nora, R., Spears, B. & Springer, P. (2017). Zonal flow generation in inertial confinement fusion implosions. *Physics of Plasmas* **24**, 032702.

R Core Team (2016). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

Rigat, F. & Mira, A. (2012). Parallel hierarchical sampling: A general-purpose interacting Markov chains Monte Carlo algorithm. *Computational Statistics & Data Analysis* **56**, 1450–1467.

Sacks, J., Welch, W. J., Mitchell, T. J. & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science* pp. 409–423.

Saltelli, A. & Annoni, P. (2010). How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software* **25**, 1508–1517.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M. & Tarantola, S. (2008). *Global Sensitivity Analysis: The Primer.* John Wiley & Sons.

Saltelli, A., Tarantola, S. & Campolongo, F. (2000). Sensitivity analysis as an ingredient of modeling. *Statistical Science* pp. 377–395.

Saltelli, A., Tarantola, S., Campolongo, F. & Ratto, M. (2004). *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models.* John Wiley & Sons.

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Barker, D. M., Wang, W. & Powers, J. G. (2008). A description of the advanced research WRF version 3. Technical Report NCAR/TN-475+STR, National Center For Atmospheric Research Boulder Co Mesoscale and Microscale Meteorology Div.

Sobol', I. M. (1990). On sensitivity estimation for nonlinear mathematical models. *Matematicheskoe Modelirovanie* **2**, 112–118.

Sobol', I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and computers in simulation* **55**, 271–280.

Stohl, A., Forster, C., Frank, A., Seibert, P. & Wotawa, G. (2005). Technical note: The Lagrangian particle dispersion model FLEXPART version 6.2. *Atmospheric Chemistry and Physics* **5**, 2461–2474.

Storlie, C. B., Lane, W. A., Ryan, E. M., Gattiker, J. R. & Higdon, D. M. (2015). Calibration of computational models with categorical parameters and correlated outputs via Bayesian smoothing spline anova. *Journal of the American Statistical Association* **110**, 68–82.

Storlie, C. B., Swiler, L. P., Helton, J. C. & Sallaberry, C. J. (2009). Implementation and evaluation of nonparametric regression procedures for sensitivity analysis of computationally demanding models. *Reliability Engineering & System Safety* **94**, 1735–1763.

Strait, J. & Kurtek, S. (2016). Bayesian model-based automatic landmark detection for planar curves. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 86–94.

Stripling, H., McClarren, R., Kuranz, C., Grosskopf, M., Rutter, E. & Torralva, B. (2013). A calibration and data assimilation method using the Bayesian MARS emulator. *Annals of Nuclear Energy* **52**, 103–112.

Sudret, B. (2008). Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering & System Safety* **93**, 964–979.

Sun, Y. & Genton, M. G. (2012). Functional boxplots. *Journal of Computational and Graphical Statistics* .

Surjanovic, S. & Bingham, D. (2017). Virtual library of simulation experiments: Test functions and datasets. Retrieved January 5, 2017, from `http://www.sfu.ca/~ssurjano`.

Telesca, D. (2015). Bayesian analysis of curves shape variation through registration and regression. In *Nonparametric Bayesian Inference in Biostatistics*, pp. 287–310. Springer.

Thuillier, R. H. (1992). Evaluation of a puff dispersion model in complex terrain. *Journal of the Air & Waste Management Association* **42**, 290–297.

Tucker, J. D., Wu, W. & Srivastava, A. (2013). Generative models for functional data using phase and amplitude separation. *Computational Statistics & Data Analysis* **61**, 50–66.

United Nations Scientific Committee on the Effects of Atomic Radiation (2008). Sources and effects of ionizing radiation. UNSCEAR 2008 report to the general assembly, with scientific annex Volume II.

Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J. & Morris, M. D. (1992). Screening, predicting, and computer experiments. *Technometrics* **34**, 15–25.

Zellner, A. (1986). On assessing prior distributions and Bayesian regression analysis with g-prior distributions. *Bayesian inference and decision techniques: Essays in Honor of Bruno De Finetti* **6**, 233–243.