Using Derivative Information in the Statistical Analysis of Computer Models

Gemma Stephenson

School of Ocean and Earth Science

University of Southampton

Thesis for the degree of

Doctor of Philosophy

July 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF OCEAN AND EARTH SCIENCE

Doctor of Philosophy

USING DERIVATIVE INFORMATION IN THE STATISTICAL ANALYSIS OF COMPUTER MODELS

by Gemma Stephenson

Complex deterministic models are an important tool for studying a wide range of systems. Often though, such models are computationally too expensive to perform the many runs required. In this case one option is to build a Gaussian process emulator which acts as a surrogate, enabling fast prediction of the model output at specified input configurations. Derivative information may be available, either through the running of an appropriate adjoint model or as a result of some analysis previously performed. An emulator would likely benefit from the inclusion of this derivative information. Whether further efficiency is achieved, however, depends on the relation between the computational cost of obtaining the derivatives and the value of the derivative information in the emulator. In our examples we see that derivatives are more valuable in models which have shorter correlation lengths and emulators without derivatives generally tend to require twice as many model runs as the emulators with derivatives to produce a similar predictive performance. We conclude that an optimal solution is likely to be a hybrid design consisting of adjoint runs in some parts of the input space and standard model runs in others.

The knowledge of the derivatives of complex models can add greatly to their utility, for example in the application of sensitivity analysis or data assimilation. One way of generating such derivatives, as suggested above, is by coding an adjoint model. Despite automatic differentiation software, this remains a complex task and the adjoint model when written is computationally more demanding.

We suggest an alternative method for generating partial derivatives of complex model output, with respect to model inputs. We propose the use of a Gaussian process emulator which, as long as the model is suitable for emulation, can be used to estimate derivatives even without any derivative information known a priori. We present encouraging results which show how an emulator of derivatives could reduce the demand for writing and running adjoint models. This is done with the use of both toy models and the climate model C-GOLDSTEIN.

Contents

List of Figures

List of Tables

Mathematical Nomenclature

$\tilde{}$	The tilde symbol placed over a letter denotes derivative information and function output combined.
$\eta(.)$	A complex model/simulator.
$\tilde{\eta}(.)$	An adjoint model.
n	Number of locations in a design.
$\tilde{n}$	Total number of derivatives and responses in a design.
n'	Number of points in a design for validation.
$\mathbf{x}$	Point, i.e location, in the input space of the simulator.
$(\mathbf{x}, d)$	Location in the input space of the simulator and whether a derivative of the response of the simulator is to be evaluated here (depending on the value of d).
$x_i^{(k)}$	Input k at point i; an input is denoted by a superscript on x and a subscript on x refers to a point in the input space.
D	Design, comprising an ordered set of points, i.e locations, in an input space.
$\tilde{D}$	Design, explaining at which locations in the input space the response is required and at which derivatives are required.
y	Simulator output: $\eta(\mathbf{x}) = y$.
Y	Simulator multi outputs.
$\tilde{y}$	Adjoint output: $\tilde{\eta}(\mathbf{x}, d) = \tilde{y}$, the derivative of $\eta(\mathbf{x})$ with respect to input d ($d \in \{1, \ldots, p\}$). When $d = 0$ we have $\tilde{\eta}(\mathbf{x}, 0) = \eta(\mathbf{x})$.
$\tilde{Y}$	Adjoint multi-outputs.
p	Number of inputs.
q	Number of basis functions.
r	Number of outputs.
$\mathbf{h}(\cdot)$	Vector of basis functions.
$c(\cdot, \cdot)$	Correlation function.
$\mathbf{t}(\cdot)$	Correlation between training data and a point/derivative we are predicting.
A	Matrix of correlations between points.
$\tilde{A}$	Matrix of correlations between points, between derivatives and points, and between derivatives themselves.
Θ	Diagonal matrix of smoothness parameters $\theta\{i\}$ which are hyperparameters of a correlation function.

β	Hyperparameters of a mean function.
B	Hyperparameters of a mean function in the multi-output setting.
σ^2	Hyperparameter which scales a covariance function.
Σ	Covariances between outputs in the multi-output setting.
FD	Finite differences
AD	Automatic differentiation

Acknowledgements

Firstly, I would like to thank my supervisor, Peter Challenor, who has provided me with much encouragement, guidance and support. I am grateful to my co-supervisors: Jeremy Oakley, who has remained a valuable and reliable source of advice since I left The University of Sheffield, and Bob Marsh who provided great help in running the C-GOLDSTEIN model. I'd like to thank Neil Edwards and Dan Zachary for providing the C-GOLDSTEIN adjoint model and also Jeff Blundell for all his time and effort helping me with the computing side; it was very much appreciated.

My experience as a PhD student benefited immensely from time spent at Los Alamos National Laboratory, I have Jim Gattiker to thank for that and for all his continued advice and support. I'm also grateful to Jeff Favorite for providing the radiation transport model data. Thanks also to Tony O'Hagan and all members of the MUCM team, past and present, for their useful comments.

Finally, I'd like to thank Dave Poulter for all his help and for showing much patience and understanding.

Declaration of Authorship

I, **Gemma Stephenson**, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

Using Derivative Information in the Statistical Analysis of Computer Models

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Signed:

Dated:

Chapter 1

Introduction

1.1 Complex Models

Complex models are used as representations of real-world phenomena. The models simulate real-world systems and are thus used in many different areas. Climate models, for example, involve solving many ODEs (ordinary differential equations) and PDEs (partial differential equations) and can take an appreciable amount of computing time to run. Complex models are also used to simulate processes in oil reservoirs to investigate, for example, the geological structure of the reservoir and the distribution of hydrocarbons in the reservoir. Models similar to those just described are required because conducting physical experiments is expensive and not always practical, or even possible in some cases. There are multiple applications of these models; for example to predict how the real world system may behave in the future or to find which input parameters maximise a particular output. These types of analyses, as well as the models themselves, are important for further applications such as policy-making.

Complex models are written as computer codes and referred to as simulators. We refer to a model, or code, *run* as evaluating the simulator at a single input value. Performing an ensemble of model runs, at a number of different input values, is given the term *computer experiment.* We express the simulator by the function $y = \eta(\mathbf{x})$, where $\mathbf{x}$ are the model inputs. Depending on the model, the output could be either a scalar or a vector. We consider both cases in this thesis: in Chapters 3, 4 and 5 we concentrate on just univariate output and in Chapter 6 we investigate multi-output emulators. The simulators are deterministic, for each time they are run with the same inputs they produce the same output.

1.2 Uncertainty in complex models

Model users need to know how much they can trust simulator outputs and therefore it is necessary to quantify the uncertainty in a model. The true values of model inputs may be unknown if they are properties of the real system which need to be observed. To observe such properties is expensive, not always practical and when the values of the inputs can be obtained, it is likely that some form of measurement error will have occurred. We need to quantify the uncertainty in the inputs as this will relate to how uncertain we are that the simulator output matches reality. We denote the true value of the inputs by $\mathbf{X}$ and the true output is then $Y = \eta(\mathbf{X})$. The purpose of uncertainty analysis is to quantify the uncertainty in model outputs caused by uncertainty in the inputs.

Another source of uncertainty is in the model itself. Even if the true values of the inputs are known, running the model at these points will not produce an output which matches exactly the observation of the real-world system. We refer to the difference between the true system value and the simulator output at the best input as the model discrepancy. Learning about model discrepancy enables calibration, which is the process of modifying inputs and reducing uncertainty in inputs by using physical observations, z_i, of the real system. Such values will be observed with a measurement error and Kennedy and O'Hagan (2001) propose the following model to link model output to reality:

$$z_i = \text{true process} + e_i = \rho\,\eta(\mathbf{x}_i, \boldsymbol{\omega}) + \delta(\mathbf{x}_i). \tag{1.1}$$

Here, e_i are the observation errors, taken to be independent and normally distributed, ρ denotes an unknown regression parameter and $\eta(\mathbf{x}_i, \boldsymbol{\omega})$ is the computer code output. The inputs to the simulator are split into calibration inputs, $\boldsymbol{\omega}$, and variable inputs, $\mathbf{x}$. The calibration inputs are unknown but fixed and observational data are required to estimate them. The variable inputs are assumed to be known for the observations used for calibration, for example $\mathbf{x}$ might be a physical location. Once the calibration inputs have been estimated, the variable inputs might be varied to predict the model output at other input points. Finally, $\delta(\mathbf{x}_i)$, is the model discrepancy function and this is independent of the code output. Hence through this function, uncertainty about how well $\eta(.)$ models the true process can be described. Higdon *et al.* (2008) extend this approach to a multivariate setting using a fully Bayesian analysis. In their example, 36 model runs result in 18720 output values and this would not otherwise have been tractable. Further development of the Kennedy and O'Hagan (2001) approach to calibration is included in the 6 step framework to validating a computer model by Bayarri *et al.* (2007); while Hang *et al.* (2009) also perform calibration though in an approach which takes into account, and estimates simultaneously, parameters without an exact physical representation in the real system.

As we discuss further in Chapter 2, it may be of interest to learn how sensitive the model output is to its various inputs. The process of evaluating how the output of a model is modified by changes in the inputs is referred to as sensitivity analysis.

Complex models tend to take an appreciable amount of computing time to run and in this sense they are expensive to execute. This is partly due to the high number of dimensions a simulation of a real-world system can require. Performing analyses such as sensitivity and uncertainty analysis can require many runs of the simulator and this quickly becomes impractical with a computationally expensive model.

A practical solution to efficiently obtain the required output, is to build a statistical approximation of the simulator. Through this statistical approximation we can build a complete probability distribution for $\eta(\cdot)$; where the mean of this distribution provides an approximate value to the simulator output and the standard deviation of the distribution describes how close it expects this mean value to be to the true simulator output. If we can approximate the simulator well enough, any runs required for analyses such as sensitivity analysis can be obtained by this method. We call this statistical approximation of the simulator, an emulator. Emulators are described in detail in Chapter 3.

1.3 Models

In this section we describe two complex models of real-world systems that are analysed throughout this thesis. The purpose of this thesis, however, is to investigate the use of derivative information in the emulation of complex models and which models we choose to demonstrate with, is in some sense arbitrary. As such, only limited detail of the science behind the models and the inner-working of the models is presented here. The reader is referred to appropriate references for both models should further detail be of interest.

1.3.1 C-GOLDSTEIN climate model

The C-GOLDSTEIN software encodes a computationally fast Earth System Model (ESM) developed by N. R. Edwards and R. Marsh (Marsh *et al.*, 2002). It comprises three coupled model components:

- The Global Ocean-Linear Drag Salt and Temperature Equation Integrator (GOLDSTEIN) ocean model based on the version of GOLDSTEIN presented by Edwards and Shepherd (2002).
- An Energy Moisture Balance Model (EMBM), which is an atmospheric model relating all components of the atmospheric system to just 3 variables (surface air temperature, sea surface temperature and specific humidity). The resulting model is then simple and efficient; and when coupled to an ocean model, a good simulation of climate can be achieved (Fanning and Weaver, 1996). The EMBM used in C-GOLDSTEIN is based on the UVic Earth System Climate Model developed by Weaver *et al.* (2001).
- A simple sea ice model also based on Weaver *et al.* (2001).

Details of the model computational components, variable parameters, input and output data are given in Marsh *et al.* (2002) along with a brief description of installation and execution procedure. Table 1.1 presents the input parameters of the model. The behaviour of the ocean, atmosphere and cryosphere in the model are altered by the parameters in the first, second and third sections of Table 1.1 respectively. The two parameters in Table 1.1: the e-folding timescale of Carbon removal and the sensitivity of the Greenland ice sheet melt to warming, are only required if the model is to simulate a future climate.

The C-GOLDSTEIN model consists of differential equations derived from the laws of physics and chemistry. The physics is simplified though and the version we use is implemented on a 36×36 grid. Longitudinal resolution is 10° and latitudinal resolution is approximately 3° at the equator, increasing to nearly 20° at the poles. This is a relatively low resolution; for example HadCM3, the Hadley Centre

Input	Default Value
Wind stress scale	2.00
Ocean horizontal diffusivity (m^2s^{-1})	2000.
Ocean vertical diffusivity (m^2s^{-1})	1e-4.
Ocean drag coefficient (days)	2.5
Initial humidity over ocean	0.
Atlantic-to-Pacific freshwater flux adjustment values (Sv)	-0.03, 0.17, 0.18, 0
Scaling factor for Atlantic to Pacific moisture flux	1
Atmospheric heat diffusivity (m^2s^{-1})	5.0e6
Atmospheric moisture diffusivity (m^2s^{-1})	1.0e6
Width of atmospheric heat diffusivity profile (rad)	1.0
Slope of atmospheric heat diffusivity profile	0.1
Zonal heat advection factor	0.
Meridional heat advection factor	0.
Zonal moisture advection factor	0.4
Meridional moisture advection factor	0.4
Scales co2 concentration relative to 350ppm	1.0
Specifies a compound annual % rate of increase	0.0
Climate sensitivity (Wm^{-2})	5.77
Solar Constant (Wm^{-2})	1368
Threshold Relative Humidity above which precipitation occurs	0.85
e-folding timescale of Carbon removal (years)	150.
Sea ice diffusivity (m^2s^{-1})	2000.
Sensitivity of Greenland Ice Sheet melt to warming ($Sv\,^{\circ}C^{-1}$)	0.01

Table 1.1: Input parameters for C-GOLDSTEIN

Coupled Model (version 3), has a resolution of 2.5°(latitude) × 3.75°(longitude) which results in a grid of 96 × 73 cells. In our version of C-GOLDSTEIN there are 100 timesteps per year and 8 ocean levels with a maximum depth of 5km. The model is generally run for a spinup phase of 4000 years at the end of which, the model is expected to have reached equilibrium. There are many outputs of the model; we illustrate one of the outputs, surface air temperature, at the year 2000 under default parameter settings in Figure 1.1.

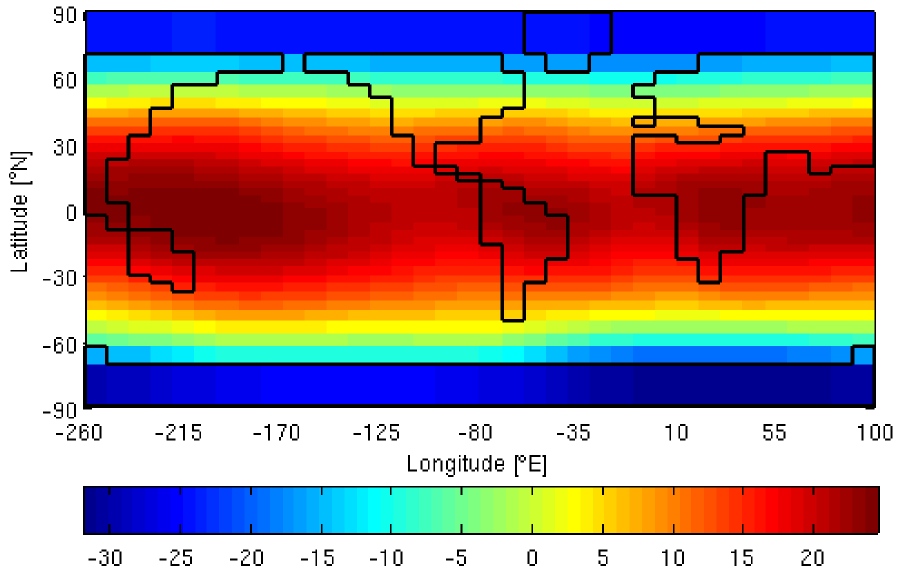


Figure 1.1: Surface air temperature output from the C-GOLDSTEIN model, run to AD2000 at default parameter settings.

The reduced physics and coarse grid in the C-GOLDSTEIN model help to make it computationally fast: using a single core of a Sun Linux workstation a spinup run of 4000 model years requires approximately 1 hour. In comparison, HadCM3 requires approximately one day to simulate 10 model years using 16 cores on HECToR (Cray XT4), the national supercomputer in the UK. On a single workstation core integration speed would be around 1 model year per day. It should be noted that a typical HadCM3 spinup run is only of order a few hundred simulation years, which is sufficient to eliminate the strong surface drifts but a long way short of reaching equilibrium in the deep ocean (Gordon *et al.*, 2000). We compare the runtime of these two models only to explain what 'computationally fast' in this situation means. C-GOLDSTEIN and HadCM3 should not be directly compared without taking further differences between the models, for example the resolution, into consideration.

For further detail of the steps required to build, configure, execute and utilise the C-GOLDSTEIN model, see Appendix B.

1.3.2 The radiation transport model

This model calculates the measured radiation signature of a gamma-ray-emitting and neutron-multiplying cylinder. The setup of the model is illustrated in Figure 1.2. The inner cylinder is the gamma-ray source and it is shielded by the outer

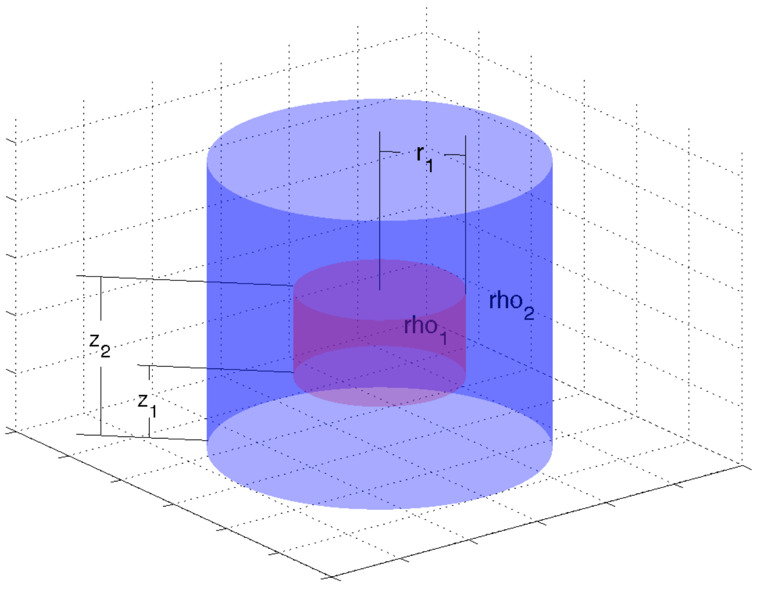


Figure 1.2: An illustration of the material configuration used in the radiation transport model and the five inputs varied in this work.

material. There are five input variable parameters, shown in Figure 1.2, and given in Table 1.2 along with the range of values we assigned to them. These ranges cover reasonable configurations, as determined by the modeling domain expert. The model generates a radiation signature which consists of five outputs: the unscattered fluxes of four gamma-ray lines and the effective neutron multiplication factor. The gamma-ray lines are gamma-rays of different energy levels, 4 being chosen as appropriate by the model expert. The effective neutron multiplication factor is the average number of neutrons from one nuclear fission that cause another fission. In the real system that we are modelling these outputs are measured at a point external to the cylinders, with the purpose of identifying the inner material based on its measured radiation signature. For our analysis, therefore, we will ultimately be interested in inverting the model, that is, finding what configurations are consistent with a measured signature.

Input	Name	Description	Range	Units
Input 1	r_1	Inner radius	0.2 - 19.8	cm
Input 2	ρ_1	Inner density	2 - 20	g/cm^3
Input 3	ρ_2	Outer density	2 - 20	g/cm^3
Input 4	z_1	Distance to base of material	0.2 - 9.8 (with the condition that $z_2 > z_1$)	cm
Input 5	z_2	Distance to top of material	0.4 - 9.8	cm
-	-	Outer radius	Fixed at 20	cm

Table 1.2: Inputs of the radiation transport model

The radiation transport model comprises of two parts: the PARTISN model and a ray tracing code. PARTISN is a computer program package which numerically solves the neutral-particle Boltzman transport equation. See Alcouffe (2001) for detail regarding PARTISN. The radiation transport model uses PARTISN for the neutron multiplication calculations, which are done in 30 energy groups. Two PARTISN calculations are done for each geometry, a forward and an adjoint, and then the results are combined in integrals over the phase space (space, angle, and energy) to give the derivatives. For the gamma-ray calculations, a ray-tracing code is employed Favorite and Bledsoe (2008); Favorite *et al.* (2009). The forward and adjoint phase-space integrals are computed directly, in a single calculation.

The radiation transport model has a mathematical adjoint and we use this property to compute the adjoint-based Jacobian matrix of the response signature with respect to five variable parameters in the test configuration. Automatic differentiation, as described in Chapter 2, is not used in the coding of the adjoint model; the relevant equations are linear and their adjoints written analytically. The adjoint to the radiation transport model requires approximately twice the amount of computing time of the model alone; thus we can obtain function output and partial derivatives of all five outputs with respect to the five inputs at about twice the cost of obtaining just the function output. We discuss adjoint models further in Chapter 2.

A Latin hypercube sample (LHS), created by Jim Gattiker, determined at which input points the model was evaluated. The 100 point LHS was generated across 5 input dimensions and the points scaled linearly from [0,1] to appropriate values using the ranges supplied in Table 1.2. We require that Input 5 > Input 4 and so it was ensured that the values for Inputs 4 and 5 lie in the upper triangle of a 2-dimensional scatterplot between these 2 inputs. All the 2-dimensional scatterplots for this design are shown in Figure 1.3. This is the only data we have available and was provided by Jeff Favorite.

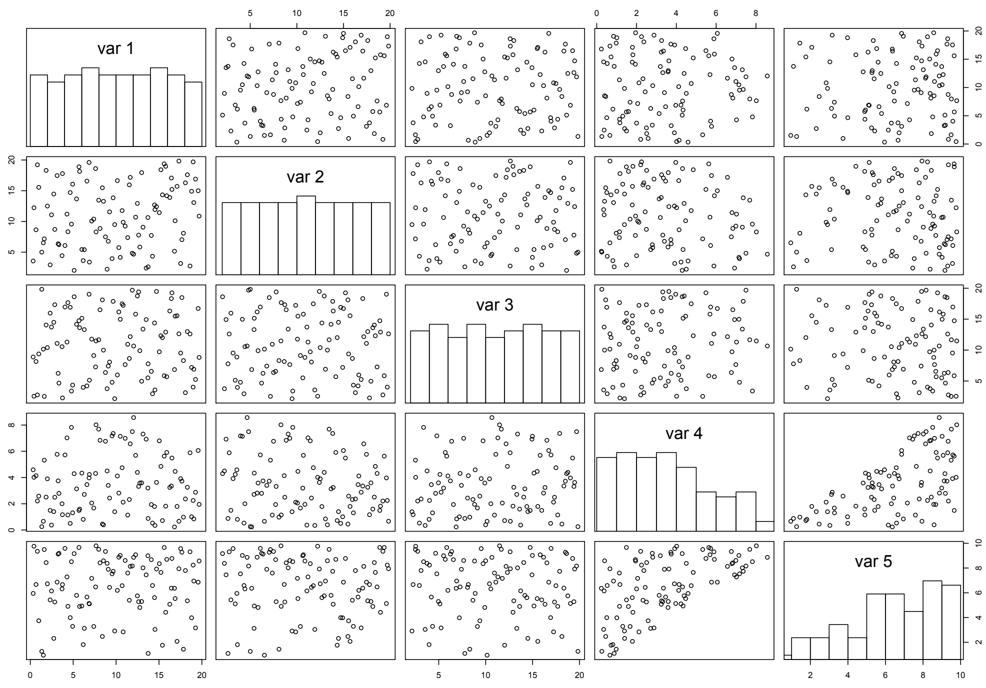


Figure 1.3: Two-dimensional scatterplots of the inputs.

1.4 Layout of thesis

In this chapter the need for managing uncertainty in complex models has been discussed and two models of real world systems which we use throughout this thesis, have been presented. In Chapter 2 we discuss derivatives of complex models, including motivation for why derivatives of these models are useful and various methods for obtaining them. The computational cost of generating derivatives is an important factor here and is therefore also discussed.

Gaussian process emulators are described in Chapter 3, along with a simple one-dimensional example to illustrate how an emulator can be used as a surrogate to a complex model. We then extend the standard Gaussian process emulator methodology in Chapter 4. This chapter looks at including derivative information in a Gaussian process emulator, in an attempt to more efficiently emulate the function output of a complex model.

In Chapter 5, the ideas of Chapters 2, 3 and 4 are all brought together and we show how we can build a Gaussian process emulator, with or without derivative information, to emulate the derivatives of a complex model. Chapter 6 is concerned with multi-output emulators and we investigate whether the performance of such an emulator is superior to independent emulators, built for each output of a model either with or without derivative information.

We conclude with Chapter 7; the ideas and results of this thesis are summarised in this chapter and, where relevant, scope for further work discussed.

We also include 2 appendices. The first, Appendix A, presents the MUCM toolkit pages that the author has written as a result of the work produced in this thesis. The second, Appendix B, is a practical guide to running the C-GOLDSTEIN model, which itself is described in Section 1.3.1. It should be noted, however, that the guide in Appendix B, which is written by the author, is not an official manual and not intended to be a complete user guide.

Chapter 2

Derivatives of complex models

2.1 Introduction

Complex models are an important tool for studying a wide range of systems. These models, which are written as computer codes and referred to as simulators, tend to take an appreciable amount of computing time to run and in this sense they are expensive to execute. Various analyses that model users may wish to perform rely on knowledge of derivatives of the simulator output with respect to the model inputs. Sensitivity analysis, for example, is the process of evaluating how the output of a model is modified by changes in the inputs. Saltelli *et al.* (2000) provide a number of different objectives of sensitivity analysis, for example investigating interactions between inputs. Sensitivity analysis may also reveal the output of the model is only strongly affected by some of the inputs and thus calibration of some input parameters will yield only marginal benefits. Obtaining observations required for calibration is likely to be expensive and so sensitivity analysis may provide a means to reducing this expense. Sensitivity analysis can be either local or global. Local sensitivity analysis is the process of calculating partial derivatives with respect to one input variable to investigate how the output is affected by small changes in that input. Global sensitivity analysis studies the effect on the output as a result of larger changes in the inputs; all inputs may be varied simultaneously in global sensitivity analysis and the whole range of interest of each inputs is generally studied.

Other tasks which make use of derivative information include data assimilation, which combines observations of the state of a system and a computer model of the system, and optimisation problems. If we wish to find which sets of input values results in either a maximum or a minimum output, then knowledge of the gradient of the function may result in a more efficient search. The information derivatives provide may also prove valuable in model development.

There are various methods to calculating derivatives. Analytical differentiation provides exact derivatives and would be the preferred choice. Sometimes this isn't practical though or the closed form expression of the simulator may be unknown. An alternative method is therefore required and in this chapter we discuss two of the main approaches of producing derivatives when analytical differentiation is not possible: finite differences, in Section 2.2, and adjoint models, in Section 2.3. We include a practical example demonstrating how a simple adjoint of a toy model could be built and in Section 2.3.4 we compare the two methods of generating derivatives of the climate model, C-GOLDSTEIN.

2.2 Finite differences

A derivative is defined by:

$$\frac{\partial\,\eta(x)}{\partial x} = \lim_{\epsilon\to 0}\frac{\eta(x+\epsilon)-\eta(x)}{\epsilon}, \tag{2.1}$$

and, if analytical differentiation in not an option, a common approach to calculating derivatives is the finite differences, FD, method. This approach estimates a derivative by fixing ϵ from equation (2.1) at a small value so we have:

$$\frac{\partial\,\eta(x)}{\partial x} \approx \frac{\eta(x+\epsilon)-\eta(x)}{\epsilon}, \tag{2.2}$$

for small ϵ. If a model has p multiple inputs then partial derivatives can be obtained by perturbing each input by the amount ϵ in turn. We therefore require $p+1$ model runs to obtain the partial derivatives w.r.t all inputs at one point in the input space. Clearly only approximate derivatives are calculated in practice here, as truncation and cancellation errors arise for very small ϵ. The truncation error is evident from the Taylor series:

$$\eta(x+\epsilon) = \eta(x) + \epsilon\frac{\partial\eta(x)}{\partial x} + \frac{\epsilon^2}{2!}\frac{\partial^2\eta(x)}{\partial x^2} + \frac{\epsilon^3}{3!}\frac{\partial^3\eta(x)}{\partial x^3} + \dots\ . \tag{2.3}$$

A simple rearrangement of (2.3) gives:

$$\frac{\partial\eta(x)}{\partial x} = \frac{\eta(x+\epsilon)-\eta(x)}{\epsilon} - \frac{\epsilon}{2!}\frac{\partial^2\eta(x)}{\partial x^2} - \frac{\epsilon^2}{3!}\frac{\partial^3\eta(x)}{\partial x^3} - \dots\ , \tag{2.4}$$

and we can therefore see that a truncation error of size

$$\frac{\epsilon}{2!}\frac{\partial^2\eta(x)}{\partial x^2} + \frac{\epsilon^2}{3!}\frac{\partial^3\eta(x)}{\partial x^3} + \dots\ , \tag{2.5}$$

occurs if we undertake the FD approach in this way. The centre divided differences approach, given as:

$$\frac{\partial\,\eta(x)}{\partial x} \approx \frac{\eta(x+\epsilon) - \eta(x-\epsilon)}{2\,\epsilon}, \tag{2.6}$$

is an improvement in accuracy, as noting that the Taylor series for $\eta(x-\epsilon)$ is:

$$\eta(x-\epsilon) = \eta(x) - \epsilon\frac{\partial\eta(x)}{\partial x} + \frac{\epsilon^2}{2!}\frac{\partial^2\eta(x)}{\partial x^2} - \frac{\epsilon^3}{3!}\frac{\partial^3\eta(x)}{\partial x^3} + \dots\ ,$$

this then gives:

$$\frac{\partial\eta(x)}{\partial x} = \frac{\eta(x+\epsilon) - \eta(x-\epsilon)}{2\,\epsilon} - \frac{\epsilon^2}{3!}\frac{\partial^3\eta(x)}{\partial x^3} - \frac{\epsilon^4}{5!}\frac{\partial^5\eta(x)}{\partial x^5} - \dots\ , \tag{2.7}$$

and we now have a truncation error of size

$$\frac{\epsilon^2}{3!}\frac{\partial^3\eta(x)}{\partial x^3} + \frac{\epsilon^4}{5!}\frac{\partial^5\eta(x)}{\partial x^5} + \dots\ ,$$

which will be smaller than (2.5). Now, however $2\,p+1$ runs of the simulator are required to learn the function output and all the partial derivatives at one location in the input space.

A partial derivative generated by either (2.2) or (2.6) is sensitive to the value of ϵ; therefore multiple model runs with varying ϵ are often required to determine an appropriate value for ϵ. If we have a stationary model such that the smoothness of the model is consistent over the design space, then this overhead is only expected to occur once per model input dimension. However, as we see in Chapter 5, for the C-GOLDSTEIN model a constant value for ϵ across the design space for one input dimensions yields wildly inaccurate derivatives. Searching for appropriate values of ϵ of course adds to the overall computational expense of generating derivatives by a finite differences method.

There are methods that can be undertaken to improve on the accuracy of derivatives calculated using the FD approach. Press *et al.* (1992) Chapter 5, for example, recommend that ϵ is chosen such that x and $x+\epsilon$ differ by a number which is exactly representable in binary. An 'exact' number for ϵ, as interpreted by the computer the complex model is running on, reduces the round off error in the FD approach. The NAG fortran library routine, D04AAF, can be employed to provide FD generated derivatives assuming the simulator is written in the fortran language. To generate derivatives at the point x, the routine requires 21 evaluations of the simulator: $\eta(x)$ and $\eta(x \pm (2i-1)\epsilon)$, where $i = 1, 2, \ldots, 10$. An FD table is then built up consisting of approximately 200 entries using an extension of the Neville algorithm, which is described by Lyness and Moler (1969). Derivatives up to the order 14 are then returned but the routine warns that derivatives with increasing order become less reliable. While more sophisticated formulae are employed in this NAG routine than simply adopting (2.2) or (2.6) to generate FD derivatives, with many complex models performing 21 simulator runs to generate the derivative w.r.t to one input at one point in the input space is likely to be computationally too expensive. We therefore do not provide further detail here of the Neville algorithm and FD table generated by the routine, but refer the reader to Numerical Algorithms Group (2006).

In summary, FD methods do not give rise to exact values of derivatives and the number of simulator runs required to employ these methods means the approach is computationally very expensive. Assuming ϵ has been assigned an appropriate value though, reasonably accurate derivatives can be generated and therefore comparison with other methods of generating derivatives is meaningful and informative.

2.3 Adjoints

An alternative approach to FD is to build the adjoint of the simulator to generate the derivatives. An adjoint model, written either by hand or with the application of a compiler that automatically differentiates the code (automatic differentiation), produces partial derivatives in addition to the standard output of the model. Adjoint models were first written for the purpose of sensitivity analysis in reactor physics problems, (Hall and Caucui, 1982), and have become popular in more fields, in particular climate science, as the availability of automatic differentiation software has increased. Marotzke *et al.* (1999) present a summary of some of the early adjoints of general circulation models.

An adjoint model, which encompasses the required adjoint equations to generate the derivatives, is computationally more expensive to run than the standard model. Efficiency is achieved in comparison to the FD method to calculating derivatives though because where computing partial derivatives by FD requires at least two runs of the simulator for each input parameter, the corresponding derivatives can all be generated by one single run of an appropriate adjoint model. Roh *et al.* (1999) show that in their example, automatic differentiation requires half the computational time that the method of divided differences needs to produce the relevant derivatives. Adjoint models therefore are generally preferred to the finite differences method for generating derivatives as they are much more efficient and produce more accurate derivatives as cancellation and truncation errors are reduced. In this section we describe the adjoint method and demonstrate how automatic differentiation works with a simple one dimensional toy model.

2.3.1 Adjoint method

An outline of the adjoint method is given here following the format of Hascoët *et al.* (2005). We have a model, $\eta(\cdot)$, which takes inputs, $\mathbf{x}$, and returns outputs, $\mathbf{y}$. There are p total inputs to the model and r total outputs. If we require the partial derivatives of all the outputs of the model with respect to all the inputs, then the Jacobian matrix is required:

$$\text{Jacobian} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_r}{\partial x_1} & \cdots & \frac{\partial y_r}{\partial x_p} \end{bmatrix}. \tag{2.8}$$

The model, $\eta(\cdot)$, is complex and therefore likely to be composed of multiple subroutines. The subroutines themselves are then generally made up of smaller routines and eventually we are left with elementary functions on individual lines of code, which, assuming they are differentiable, can be differentiated by applying the chain rule. We denote these elementary functions by $f_1, \ldots, f_K$ where f_1 is the first function executed when we run $\eta(\cdot)$ and f_K is the last. We can therefore express the model $\eta(\cdot)$ in the following way:

$$\eta(\mathbf{x}) = f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{x}), \tag{2.9}$$

and so we have K total elementary functions which make up the model. Function f_1 will act on, and therefore likely effect, the input variables, $\mathbf{x}$ and so we denote $\mathbf{z}_k$ to be the state vector: the vector of all variables values after the first k functions have been executed. In this way, we set $\mathbf{x} = \mathbf{z}_0$, the $\mathbf{x}$ vector remains fixed and we have $\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$. We use this notation for the description of the method only, in practise $\mathbf{z}_k$ is not stored but overwritten by $\mathbf{z}_{k+1}$; this is important in particular for the reverse mode which we discuss later in this section.

To differentiate the whole model with respect to $\mathbf{x}$ and obtain $\eta'(\mathbf{x})$, the chain rule can then be applied to (2.9) as follows:

$$\eta'(\mathbf{x}) = f'_K(\mathbf{z}_{K-1}) \,.\, f'_{K-1}(\mathbf{z}_{K-2}) \,.\, \dots \,.\, f'_1(\mathbf{z}_0 = \mathbf{x}). \tag{2.10}$$

Each $f'_k(\mathbf{z}_{k-1})$ is an intermediate Jacobian matrix of partial derivatives. The resulting, final, Jacobian matrix, $\eta'(\mathbf{x})$, is clearly very large as it contains all the derivatives of all the outputs, with respect to all the inputs. Often however, a model user may only be interested in the derivatives of a subset of outputs with respect to the most active variables; in this case the model can be differentiated so that only specific elements of the Jacobian matrix are returned and this results in a more efficient differentiated model.

If we require the derivatives of all r outputs with respect to one of the inputs, i, i.e. the i^{th} column of the full Jacobian matrix (2.8), we can apply the tangent mode:

$$\begin{aligned}
\frac{\partial \mathbf{y}}{\partial x^{(i)}} &= \eta'(\mathbf{x} = \mathbf{z}_0)\,\frac{\partial \mathbf{x}}{\partial x^{(i)}}, \\
&= f'_K(\mathbf{z}_{K-1}) \,.\, f'_{K-1}(\mathbf{z}_{K-2}) \,.\, \dots \,.\, f'_1(\mathbf{z}_0)\,\frac{\partial \mathbf{x}}{\partial x^{(i)}} \qquad (2.11)
\end{aligned}$$

Clearly multiplying matrices by vectors is computationally cheaper than multiplying matrices by matrices and so equation (2.11) must be computed from right to left. This is straightforward as the state of the model after the first function, f_1, is required before the state of the model after the second function, f_2, etc. This is known as the tangent linear model and can be applied alongside the running of the standard calculations in $\eta(\cdot)$. Due to this, the tangent mode is also known as the forward mode. If we require the derivatives of all r outputs with respect to multiple inputs, the tangent mode can be applied in multiple directions and hence is called the tangent multidirectional mode. This can be done in one source transformation of the original code. The computational cost of the tangent mode is therefore proportional to the number of inputs we require derivatives with respect to.

An alternative method to the tangent linear model is the adjoint method. An adjoint model is defined as the transpose of the Jacobian matrix, (Marotzke *et al.*, 1999). If we require the derivatives of one output, j, with respect to all

p inputs, i.e the j^{th} row of the full Jacobian matrix (2.8), then we can use the adjoint method and transpose of $\eta'(\mathbf{x})$:

$$\begin{aligned}\frac{\partial y^{(j)}}{\partial \mathbf{x}} &= (\eta'(\mathbf{x}=\mathbf{z}_0))^T \, \frac{\partial \mathbf{y}}{\partial y^{(j)}}, \\ &= (f_1'(\mathbf{z}_0))^T (f_2'(\mathbf{z}_1))^T \ldots (f_K'(\mathbf{z}_{K-1}))^T \, \frac{\partial \mathbf{y}}{\partial y^{(j)}}. \end{aligned} \tag{2.12}$$

As with the tangent mode, computing (2.12) from right to left is much more efficient than left to right but now the state of the model after the second function, f_2 is required before the state of the model after the first function, f_1. Hence this method is called the reverse mode. We have used the notation $\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$ but, as discussed earlier in this section, in the model itself, executing f_k will cause $\mathbf{z}_k$ to overwrite $\mathbf{z}_{k-1}$. Now to calculate the derivatives, as in (2.12), the first step is to calculate $(f_K'(\mathbf{z}_{K-1}))^T$ and to do this we require the state vector after $K-1$ functions have been executed: $\mathbf{z}_{K-1}$. To generate this we therefore need to execute $\mathbf{z}_{k-1} = f_{K-1} \circ \ldots \circ f_1(\mathbf{z}_0)$. We would expect that the function output, $\mathbf{y}$ will be required in addition to the derivatives so there is no additional computational expense executing these functions here. The next step in the calculation of (2.12) requires $\mathbf{z}_{K-2}$ and although this has already been calculated, it has since been overwritten by $\mathbf{z}_{K-1}$. Therefore either the standard f functions must be computed again, up to f_{K-2}, or when $\mathbf{z}_{K-2}$ was first calculated it could be stored such that it can be recalled when required. This can then be repeated for each $\mathbf{z}_k$ in $k \in \{1, \ldots, K\}$. If the latter option is chosen then the computational time is less but memory and storage requirements are greater and can in some cases cause problems. Regardless though of whether the forward or reverse mode is adopted, the computing resource required to run such a model is greater than its corresponding simulator which is the standard version of the model.

Both modes can be applied through the use of automatic differentiation assuming that the model, described by a computer program, is written in a high-level programming language such as Fortran. For a comprehensive account of automatic differentiation see Griewank (2003). Both the tangent and reverse mode calculate partial derivatives with respect to model inputs and so throughout the remainder of this document, an adjoint model will simply refer to a differentiated model. When such an adjoint model is run, the partial derivatives with respect to the model inputs in addition to the standard model output are produced.

2.3.2 Calculating adjoints

As discussed in Section 2.3.1, an adjoint model provides an efficient means of evaluating derivatives. The time taken, however, to develop the adjoint for a model is far from trivial. Of writing adjoints by hand, Marotzke *et al.* (1999) warn that coding an adjoint is an arduous task and is prone to errors. Due to this, before Automatic Differentiation software, adjoint models were a rare commodity! There are now, however, various automatic differentiation tools which aid in the construction of adjoints. One such tool is TAPENADE, which is a web-based software developed by Hascoët and Pascual (2004). A subroutine which requires differentiating is loaded into TAPENADE. The user must then specify the dependent variables, i.e. those which require differentiating and the independent variables, those which require differentiating with respect to. Finally the user chooses a mode of differentiation: tangent mode, tangent multidirectional mode or reverse mode. For more detail see Hascoët *et al.* (2005). In the next Section, 2.3.3, we use a simple example to outline how TAPENADE operates and build an adjoint of a toy model.

2.3.3 Building the adjoint of a toy model

We demonstrate the adjoint method and the use of the Automatic Differentiation software, TAPENADE, with the following model:

$$\eta(x) = x + \cos(x) + 2\sin(x),$$

which is Toy Model 2 in Chapter 4.

To build an adjoint with TAPENADE, the model must be coded in either the FORTRAN language (F77 or F95) or C. We choose to write Toy Model 2 in Fortran and produce the following source code:

```
Program ToyModel2
Implicit none
real*8 x, eta

open(1,file='Inputs')
read(1,*) x
```

```
      call sub1(x,eta)
      print*, 'x = ',x , 'eta = ',eta
      end

      subroutine sub1(x,eta)
      Implicit none
      real*8 x, eta

      eta = x + cos(x) + 2*sin(x)

      end
```

The source code is entered into TAPENADE and the following options taken:

- Name of top routine: ToyModel2
- Dependent output variables: eta
- Independent input variables: x
- Differentiate in: Tangent Mode.

We then obtain the following TAPENADE output for Toy Model 2:

```
C        Generated by TAPENADE     (INRIA, Tropics team)
C  Tapenade 2.2.4 (r2308) - 03/04/2008 10:04
C
C  Differentiation of sub1 in forward (tangent) mode:
C   variations  of output variables: eta
C   with respect to input variables: x
C
      SUBROUTINE SUB1_D(x, xd, eta, etad)
      IMPLICIT NONE
C
      REAL*8 x, eta
      REAL*8 xd, etad
      INTRINSIC COS
      INTRINSIC SIN
C
```

```
      etad = xd - xd*SIN(x) + 2*xd*COS(x)
      eta = x + COS(x) + 2*SIN(x)
      END
```

We can then combine the TAPENADE output with the original source code to produce an adjoint of the Toy Model:

```
C        Generated by TAPENADE     (INRIA, Tropics team)
C  Tapenade 2.2.4 (r2308) - 03/04/2008 10:04
C
C  Differentiation of sub1 in forward (tangent) mode:
C   variations  of output variables: eta
C   with respect to input variables: x
C

      Program ToyModel2
      Implicit none
      real*8 x, eta, xd, etad

      open(1,file='Inputs')
      read(1,*) x, xd

      call SUB1_D(x, xd, eta, etad)
      print*, 'x = ',x, 'eta = ',eta, "etad = ", etad

      end

      SUBROUTINE SUB1_D(x, xd, eta, etad)
      IMPLICIT NONE
C
      REAL*8 x, eta
      REAL*8 xd, etad
      INTRINSIC COS
      INTRINSIC SIN
C
      etad = xd - xd*SIN(x) + 2*xd*COS(x)
      eta = x + COS(x) + 2*SIN(x)
```

```
    END
```

The compilation of this adjoint source code and then running the subsequent executable generates the required function output and derivatives of Toy Model 2.

2.3.4 The C-GOLDSTEIN adjoint

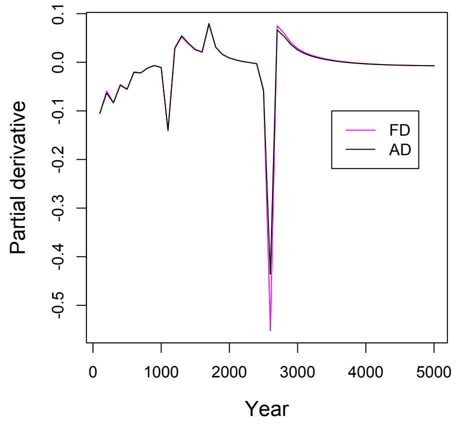
In addition to the intermediate complexity climate model, C-GOLDSTEIN, introduced in Chapter 1, there also exists the adjoint of the model, written by N. R. Edwards and D. Zachary. The adjoint model of C-GOLDSTEIN employs Automatic Differentiation, specifically, the multi-directional forward mode of TAPENADE, to produce derivative information for 12 of the input parameters, in addition to the standard model output. These 12 parameters are listed in Table 2.1. Zachary (2004) gives more detail about TAPENADE and its application to C-GOLDSTEIN.

	Input
1	Wind stress scale
2	Ocean horizontal diffusivity
3	Ocean vertical diffusivity
4	Ocean drag coefficient
5	Atmospheric heat diffusivity
6	Atmospheric moisture diffusivity
7	Zonal heat advection factor
8	Zonal moisture advection factor
9	Sea ice diffusivity
10	Atlantic-to-Pacific freshwater flux adjustment
11	Width of atmospheric heat diffusivity profile
12	Slope of atmospheric heat diffusivity profile

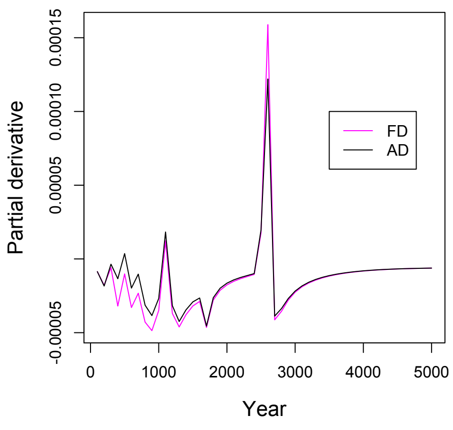
Table 2.1: Input parameters with derivatives

Further work on the C-GOLDSTEIN adjoint was required to produce a workable, appropriate model suitable to the needs of this thesis. This included differentiation of a further subroutine, to produce the derivatives of global mean air temperature. Ideally we would then validate the adjoint, especially in light of the further work performed on the model, by comparing the partial derivatives of the global mean air temperature with respect to the 12 input parameters as calculated by the C-GOLDSTEIN adjoint, to exact analytical derivatives. It is not possible to obtain analytical derivatives of the C-GOLDSTEIN model though and so instead we compare the adjoint with derivatives produced by a finite differences (FD) method. We recognise, as discussed in Section 2.2, that FD methods do not produce exact derivatives; if there is general agreement between the two methods though, we can be confident that we have a reliable adjoint model. We employ the centre divided difference equation (2.6) for the FD method with an epsilon value of 1×10^{-7}. The models are run for a long 'spinup' phase 500,000 timesteps (5000 years) at default parameter settings and the derivatives examined every 100 years. Figures 2.1 and 2.2 show how the partial derivatives compare in the 2 methods. The adjoint derivatives with respect to 11 of the parameters appear to be in general agreement with the FD method. There are peaks in some parts of the plots which are due to the nature of the variation of the parameters before the model run reaches equilibrium. For some of the inputs, at these peaks, the adjoint slightly under estimates the derivative compared to the FD value, for example in Figure 2.1a. However, there is agreement at the peak for some of the inputs, for example in Figure 2.1f.

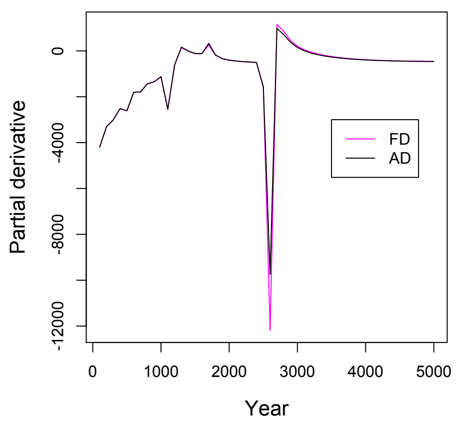
We see in Figure 2.1d that there is much conflict between adjoint produced derivatives with respect to the ocean drag coefficient, and the corresponding FD derivatives. The y-axis scale in Figure 2.1d of the value of the partial derivative is clearly inappropriate for the FD estimates. We therefore investigate the conflict in the derivative estimates w.r.t this parameter by using separate scales. The derivatives generated by the adjoint, illustrated in Figure 2.3a, appear to be a reflection in the x axis of the derivatives generated by finite differences, shown in Figure 2.3b. In Figure 2.3c we see that if we plot the adjoint generated derivatives multiplied by -1, a similar pattern is observed to the FD estimates but with a vastly different scale. There is no common factor between the estimates at each year though to explain any difference in magnitude. We attempt to verify the FD results by performing an additional 6 C-GOLDSTEIN runs with $\epsilon = \{1 \times 10^{-10},\ 1 \times 10^{-5},\ 1 \times 10^{-2}\}$ and calculating the estimates of the derivatives which result from each of these values of ϵ. The results are shown in Figure 2.3d. We see a peak at year = 400 when we have $\epsilon = 1 \times 10^{-5}$, which is in conflict with



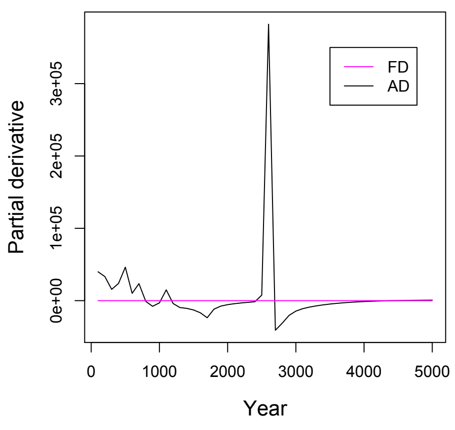
(a) Wind stress scale



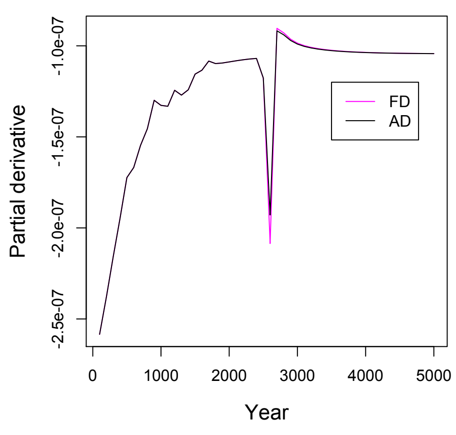
(b) Ocean horizontal diffusivity



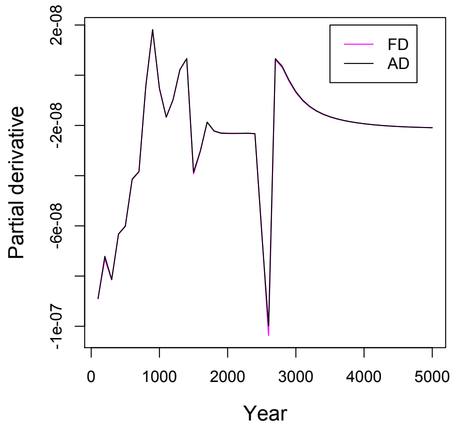
(c) Ocean vertical diffusivity



(d) Ocean drag coefficient

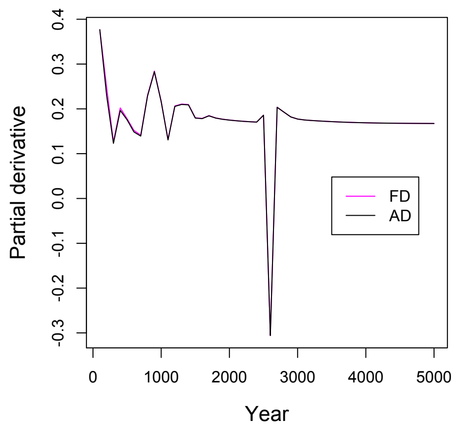


(e) Atmospheric heat diffusivity

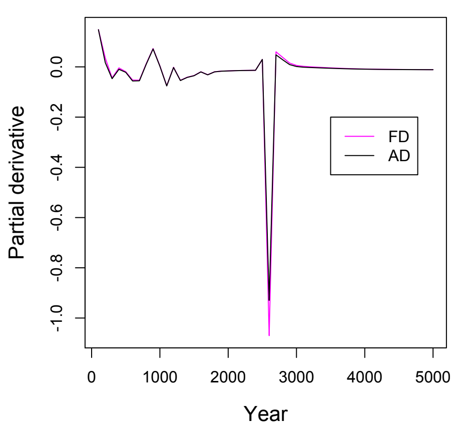


(f) Atmospheric moisture diffusivity

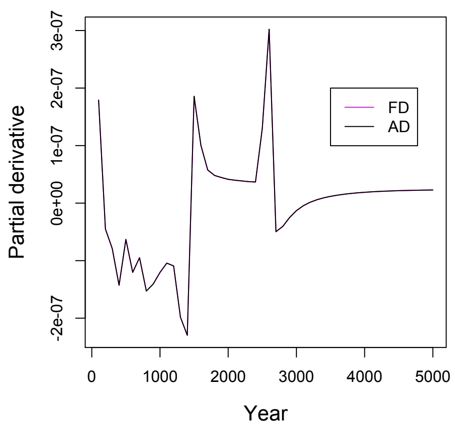
Figure 2.1: Partial derivatives of global mean air temperature with respect to parameters 1 - 6.



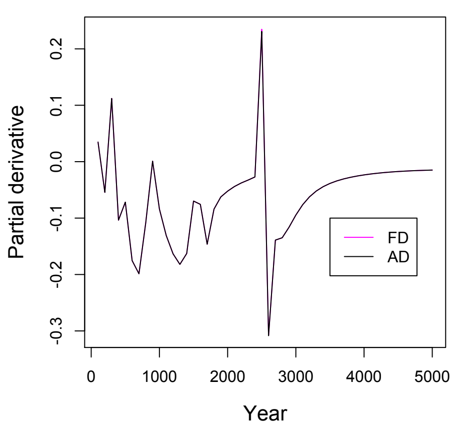
(a) Zonal heat advection factor



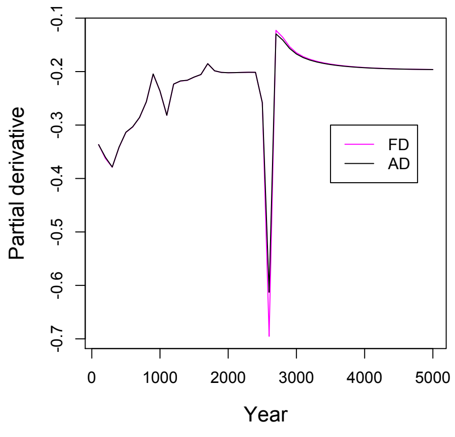
(b) Zonal moisture advection factor



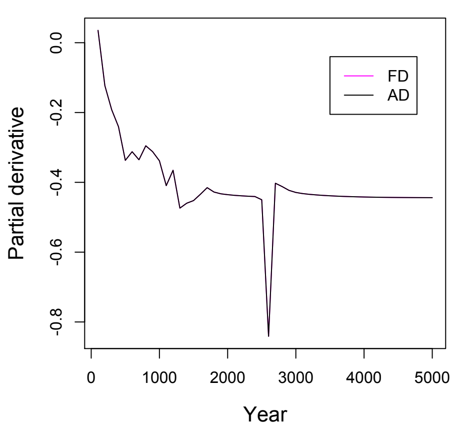
(c) Sea ice diffusivity



(d) Freshwater flux adjustment



(e) Width of atmos heat diffusivity profile



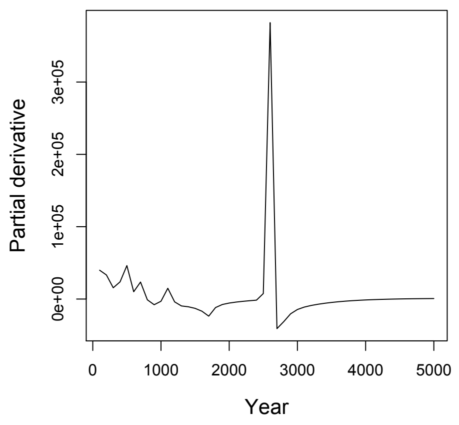
(f) Slope of atmos heat diffusivity profile

Figure 2.2: Partial derivatives of global mean air temperature with respect to parameters 7 - 12.

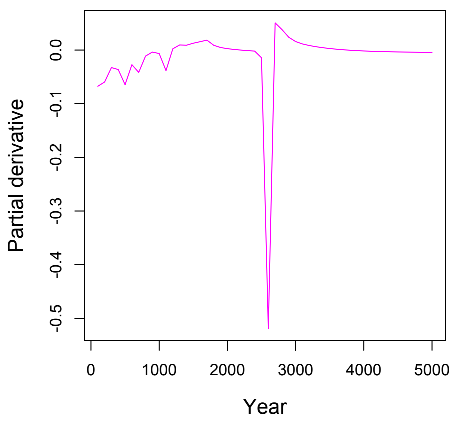
all other values of ϵ. In addition to this the derivatives when $\epsilon = 1 \times 10^{-2}$ have a slightly different pattern for the first half of the run, though are then consistently a little higher for the second half. Small differences in FD estimates with different ϵ values are to be expected and we see in Figure 2.3d, overall quite consistent results. If we assume, therefore, that the FD estimates are at least the right sign and approximate magnitude, then from Figure 2.3c it would appear that the likely problem is an errant minus sign in the adjoint code. Discussion with the original adjoint authors has identified two subroutines in the model, which are believed to be the most likely source of any possible error in the derivatives w.r.t the ocean drag coefficient. Despite further investigation, however, the conflict between adjoint and FD methods for this parameter remains unresolved.

Any analysis of the C-GOLDSTEIN model or adjoint is generally performed after the model has run to equilibrium and we can see in all panels, apart from ocean drag coefficient, of Figures 2.1 and 2.2 that there is good agreement in derivative estimates once the model has run for approximately 4000 years and reached equilibrium. This is explored further in Table 2.2 which gives a comparison of the derivative estimates at the end of the model run. All values apart from the ocean drag coefficient are close to 1 and therefore an acceptable performance is generally achieved by the adjoint for the default parameter settings. We recognise that while this provides confidence in the adjoint at the default parameter settings for 11 of the inputs, we haven't examined the performance of the adjoint elsewhere in the input space. A full scale investigation across the whole range of the inputs parameters was not practical however. If unusual adjoint behaviour is observed during subsequent analysis this can be investigated further at that time. For the purpose of this thesis, we therefore consider the C-GOLDSTEIN adjoint to be a model which produces the derivatives of global mean air temperature with respect to 11 of the input parameters.

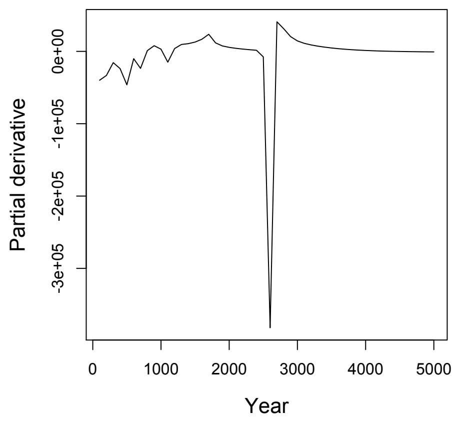
The additional computational cost of executing the C-GOLDSTEIN adjoint opposed to the the standard C-GOLDSTEIN model is far from negligible. Employing facilities (Sun Linux workstation) available at the National Oceanography Centre, Southampton, NOCS, a C-GOLDSTEIN standard model spinup run of 4000 years requires approximately 1 hour. A corresponding C-GOLDSTEIN adjoint run requires approximately 18 hours. The authors of the original C-GOLDSTEIN adjoint do report though that there is scope available for improving the efficiency of the adjoint model, (Neil Edwards, pers. comm.).



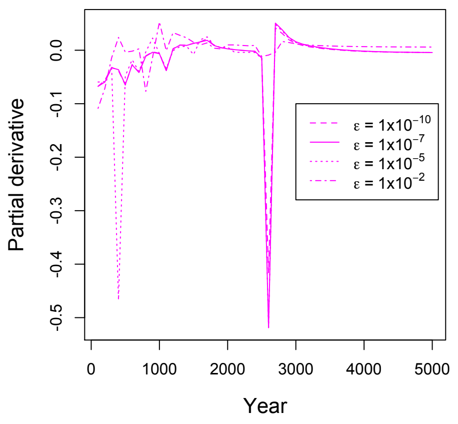
(a) Derivatives generated by the adjoint.



(b) Derivatives generated by finite differences with $\epsilon = 1 \times 10^{-7}$.



(c) Derivatives generated by the adjoint multiplied by -1.



(d) Derivatives generated by the finite differences with varying values of ϵ.

Figure 2.3: Partial derivatives of global mean air temperature with respect to the ocean drag coefficient input parameter.

C-GOLDSTEIN Parameter	AD/FD
Wind stress scale	1.010685
Ocean horizontal diffusivity	0.996329
Ocean vertical diffusivity	1.003598
Ocean drag coefficient	-162625
Atmospheric heat diffusivity	1.000126
Atmospheric moisture diffusivity	0.999667
Zonal heat advection factor	0.999990
Zonal moisture advection factor	1.010088
Sea ice diffusivity	0.994504
Atlantic-to-Pacific freshwater flux adjustment	0.999277
Width of atmos. heat diffusivity profile	1.000322
Slope of atmos. heat diffusivity profile	1.000007

Table 2.2: Relative comparison of adjoint produced derivatives with those from an FD method at year 5000.

2.4 Conclusions

Derivative information is potentially useful to many model users and as noted in Section 2.3, adjoints are generally preferred over the finite differences method to generating derivatives. Despite AD software, though, coding an adjoint model remains a complex task and the model when written, computationally expensive to execute. There may be situations when model users are unwilling, or perhaps unable, to allocate the initial time and resources required to code the adjoint of a complex model. Even if the adjoint model already exists, the additional computational expense required to run this model rather than the standard version is completely non-negligible. In Chapter 5 we suggest an alternative method for generating partial derivatives of complex model output, with respect to model inputs.

Chapter 3

Emulators

3.1 Introduction

Often, it is useful for a model user to perform analyses such as sensitivity and uncertainty analysis. As discussed in Section 1.2 of Chapter 1, the purpose of uncertainty analysis is to quantify the uncertainty in model outputs caused by uncertainty in inputs. This is done by treating the true inputs, X, as a random variable. Then, given a probability distribution, say G, for this random vector, uncertainty analysis is the study of the resulting probability distribution of the random output $Y = \eta(X)$. This output distribution is called the uncertainty distribution.

A traditional approach to uncertainty analysis is to use Monte Carlo methods. This involves sampling values of X from G and running the simulator, $\eta(\cdot)$, at these points. To estimate the mean of Y we would take the mean of this sample of outputs. To achieve acceptable accuracy, a standard sample size for this approach could be in excess of tens of thousands. The sample size is, of course, problem dependent but with a computationally expensive model it soon becomes impractical to generate the number of runs that would be required to perform this method of uncertainty analysis. Monte Carlo methods are discussed in detail by Mackay (2002). In particular the Metropolis-Hastings method, which we adopt in a small study in Chapter 6, is reviewed.

To provide efficiency in comparison to Monte Carlo methods with the simulator, an emulator can be built which acts as a statistical approximator to the simulator. An emulator encapsulates our beliefs about the simulator through a complete probability distribution. This is updated after a small number of simulator runs and the resulting probability distribution at an unknown input, provides an approximation to the true simulator output at that point. We usually take the mean of the distribution as a point estimate with uncertainty about that mean described by the standard deviation, though we do have a complete probability distribution available. To perform uncertainty analysis, we could take a sample of X from G, as with the Monte Carlo method, but evaluate the emulator mean at the required points rather than running the simulator. Far fewer runs of the simulator would be required to build an emulator, than we would need to obtain a sample of sufficient size to estimate the mean of Y by Monte Carlo methods without an emulator. It is this approach we take in Chapter 6 when we undertake a small calibration study on the radiation transport model, the model itself described in Chapter 1. With the emulator approach, however, under certain conditions it is not even necessary to take a sample to estimate the mean of the uncertainty distribution as Haylock and O'Hagan (1996) derive the distribution of the mean of Y.

Uncertainty analysis is just one application that model users may wish to perform. Once an emulator has been built and validated though, it can be used as a surrogate to the simulator in any further analysis.

Statistical emulators were developed in the 1980's from the work in Design and Analysis of Computer Experiments (DACE). For example, Sacks *et al.* (1989) use a non-Bayesian framework for approximating an unknown deterministic computer model and a Bayesian approach to predicting a computer model at untested inputs is given in Currin *et al.* (1991). Gaussian processes have become a popular choice for building emulators and a detailed discussion of Gaussian process emulation is given in Santner *et al.* (2003). Gaussian processes are also common in the field of machine learning, see Rasmussen and Williams (2006) for more detail.

In this chapter we present a review of Gaussian process emulators; for further detail and discussion see, for example, Kennedy and O'Hagan (2001) and O'Hagan (2006). In Section 3.2 we define a Gaussian process and describe the methodology required to build a Gaussian process emulator. This is then illustrated with a simple one dimensional example in Section 3.2.4. The important topic of validation is discussed in Section 3.3 and we conclude this chapter with a brief overview of some variants of standard emulators.

3.2 Gaussian process emulators

To build an emulator we first consider the model, $\eta(\cdot)$, as an unknown function, as until the model is run the output values are unknown. We require that $\eta(\cdot)$, be a smooth function so that if we know the output of the model at $\mathbf{x}_i$, we have some idea what the output of the model might be at $\mathbf{x}_j$ for $\mathbf{x}_i$ close to $\mathbf{x}_j$. We choose a Gaussian process to represent the model due to its flexibility, so that it can adapt to the shape of $\eta(\cdot)$, and convenience. In the Bayesian framework we begin by specifying our prior beliefs about the computer model and then run the simulator at the inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to obtain

$$\mathbf{y}^T = \{y_1 = \eta(\mathbf{x}_1), \ldots, y_n = \eta(\mathbf{x}_n)\}, \tag{3.1}$$

which we refer to as the *training data*. Using $\mathbf{y}$ we can derive the distribution of $\eta(\cdot)$ conditional only on the training data. The mean of this distribution acts as a fast approximation to $\eta(\cdot)$ but as we have the full distribution around the mean, we can also report how close we expect the emulator output to be to the simulator output. In the remainder of this section we look at the process of building an emulator in more detail.

3.2.1 The Gaussian process model

We choose to describe the uncertainty about the simulator output by a Gaussian process. Formally, a Gaussian process is an infinite collection of variables, where every finite subset of the variables has a multivariate normal distribution. We therefore make the assumption, a priori, that uncertainty about simulator outputs can be represented by joint normal distributions. If this is unlikely to be reasonable, a solution could be to transform the output. We apply log transformations to simulator outputs in Chapter 6 and are successful in the subsequent emulation of the transformed outputs.

A Gaussian process is defined by its mean and covariance functions so these must now be described. We specify the prior mean as:

$$E[\eta(\mathbf{x})|\,\boldsymbol{\beta}] = \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}, \tag{3.2}$$

where $\mathbf{h}(\mathbf{x})^T$ is a $1 \times q$ vector of known functions of $\mathbf{x}$ and $\boldsymbol{\beta}$ is a $q \times 1$ vector comprising of unknown coefficients. We choose the form of $\mathbf{h}(.)$ based on our prior beliefs about $\eta(\cdot)$. For example, if we believe $\eta(\mathbf{x})$ to be approximately linear in $\mathbf{x}$, then choosing $\mathbf{h}(\mathbf{x})^T = (1 \;\; \mathbf{x})$ would be appropriate.

Next we define $\sigma^2 c(\mathbf{x}_i, \mathbf{x}_j)$ to be the covariance between $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$, where σ^2 is an unknown scale parameter and $c(.,.)$ is a known correlation function. We require that $\eta(\cdot)$, be a continuous function of its inputs and consequently when two points, $\mathbf{x}_i$ and $\mathbf{x}_j$ are close there should be a high correlation between the corresponding $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$. As the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ increases, the correlation should decrease. A common form of covariance function is the Gaussian form:

$$c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta \, (\mathbf{x}_i - \mathbf{x}_j)\right\}, \tag{3.3}$$

where Θ is a diagonal matrix of positive roughness parameters,

$$\Theta = \begin{pmatrix} \theta\{1\} & 0 & \dots & 0 \\ 0 & \theta\{2\} & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & \theta\{p\} \end{pmatrix},$$

and p is the number of input dimensions. The parameters, $\theta\{k\}$ for $k \in \{1, \dots, p\}$, describe how 'rough' the model is in each input dimension. This is because the correlation between $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$ depends on the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ and this distance is rescaled as a result of Θ. The matrix, Θ, need not be diagonal, however such a form is convenient and in practise off-diagonal elements would be difficult to estimate.

3.2.2 Design

The next step is to create a design, D, which consists of an ordered set of points in the input space, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, at which the simulator is to be run to create the training data. The design of computer experiments is a large and active field. Here we introduce only a small subset of some of the design options available to computer modellers. The objective is to gain as much information as possible from the training data, as the more we learn about $\eta(\cdot)$ the better the emulator can approximate it; a space-filling design is therefore a popular choice.

Designs can be generated based on various criteria. For example, a *maximin* design is one which selects points such that the minimum distance between any two design points is maximised. McKay *et al.* (1979) compare three methods for choosing designs for Monte Carlo studies, where a deterministic computer code is modelling a real system. The three methods are simple random sampling, stratified sampling and Latin hypercube sampling, which we now describe. Suppose we have p input dimensions, with $\mathbf{x} = (x^{(1)}, \ldots, x^{(p)})$, and we need n design points at which to run the model, so we require

$$\begin{aligned}
\mathbf{x}_1 &= (x_1^{(1)}, \ldots, x_1^{(p)}), \\
\mathbf{x}_2 &= (x_2^{(1)}, \ldots, x_2^{(p)}), \\
&\vdots \\
\mathbf{x}_n &= (x_n^{(1)}, \ldots, x_n^{(p)}).
\end{aligned}$$

A Latin hypercube sample ensures that the points are spread evenly over the range of each input dimension. This is achieved by first dividing the domain of $x^{(j)}$, $(j \in \{1 \ldots p\})$, into n intervals of equal marginal probability. Then one value is sampled from each interval resulting in n values in each dimension. To obtain $\mathbf{x}_i$, we sample from one of the n values without replacement in each dimension. The process is repeated until we have $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$. McKay *et al.* (1979) assess the three methods by comparing estimators of the mean, variance and distribution function of the output. They conclude that Latin hypercube sampling is preferred.

Morris and Mitchell (1995) suggest combining a Latin hypercube with the maximin criteria to produce a maximin Latin hypercube sample. The resulting design therefore is a Latin hypercube where the minimum distance between any two points is maximised. In practise though, it is computationally very expensive to produce such a design for large n and p. A compromise is often sought where, rather than searching for the optimal design, a number of Latin hypercube samples are generated and the one with the maximum minimum distance between points is selected. We mirror this popular choice of design and frequently generate maximin Latin hypercube samples throughout this thesis.

There are other design options in addition to the Latin hypercube. For example, some non-random sequences, such as the Sobol' sequence, are found to have space-filling qualities. A design of size n can be generated by taking the first n points in the sequence; if subsequent runs are required the sequence can simply be extended and the additional points will continue to fill the input space. This is an advantage over Latin hypercube designs as if additional points to an initial Latin hypercube sample are necessary, the entire design would have to be re-computed to generate a bigger Latin hypercube. Alternatively, having created an initial design, areas where the uncertainty is greatest could be identified and then further points placed in such regions. This is an example of sequential design. See Santner *et al.* (2003) for further discussion of Sobol' sequences and design of computer experiments in general.

3.2.3 Building an emulator

From Sections 3.2.1 and 3.2.2 above, we have training data, $\mathbf{y}$, and since we have represented $\eta(\cdot)$ by a Gaussian process, the density of $\mathbf{y}$ conditional on $\boldsymbol{\beta}$ and σ^2 is:

$$\mathbf{y}|\,\boldsymbol{\beta},\sigma^2,\theta \sim N(H\boldsymbol{\beta},\sigma^2 A), \tag{3.4}$$

where $H = [\mathbf{h}(\mathbf{x}_1),\ldots,\mathbf{h}(\mathbf{x}_n)]^{\mathrm{T}}$ and A is the $n \times n$ matrix of correlations of the training data: $A = c(D,D)$. Standard normal theory, detailed by Oakley (1999), shows that:

$$\eta(\cdot)|\,\boldsymbol{\beta},\sigma^2,\mathbf{y} \sim N(m^*(.),\sigma^2 c^*(.\,,.)), \tag{3.5}$$

where

$$\begin{aligned} m^*(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} + \mathbf{t}(\mathbf{x})^T A^{-1}(\mathbf{y} - H\boldsymbol{\beta}), \\ c^*(\mathbf{x}_i,\mathbf{x}_j) &= c(\mathbf{x}_i,\mathbf{x}_j) - \mathbf{t}(\mathbf{x}_i)^T A^{-1}\mathbf{t}(\mathbf{x}_j), \\ \mathbf{t}(\mathbf{x})^T &= c(\mathbf{x},D) \\ &= \{c(\mathbf{x},\mathbf{x}_1),\ldots,c(\mathbf{x},\mathbf{x}_n)\}, \\ \mathbf{y}^T &= \eta(D) = \{\eta(\mathbf{x}_1),\ldots,\eta(\mathbf{x}_n)\}. \end{aligned}$$

We now want to obtain the distribution of $\eta(\cdot)|\,\mathbf{y}$ unconditional on $\boldsymbol{\beta}$ and σ^2. If we have genuine prior information about $\boldsymbol{\beta}$ and σ^2, then this can be utilised when specifying the prior distributions for them. In many situations though, there is enough information in the training data about these parameters and it suffices to specify a weak prior for both $\boldsymbol{\beta}$ and σ^2:

$$p(\boldsymbol{\beta},\sigma^2) \propto \sigma^{-2}. \tag{3.6}$$

Applying Bayes Theorem with (3.6) and (3.4) results in a joint Normal Inverse Gamma posterior distribution for $(\boldsymbol{\beta}, \sigma^2)$:

$$f(\boldsymbol{\beta}, \sigma^2|\mathbf{y}, \theta) \propto {\sigma^2}^{\frac{n+2}{2}} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}})^T H^T A^{-1} H(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}) + (n-q-2)\widehat{\sigma^2}\right\}, \tag{3.7}$$

where

$$\begin{aligned} \widehat{\beta} &= \left(H^{\mathrm{T}} A^{-1} H\right)^{-1} H^{\mathrm{T}} A^{-1} \mathbf{y}, & (3.8)\\ \widehat{\sigma}^2 &= (n-q-2)^{-1} \mathbf{y}^{\mathrm{T}} \left\{A^{-1} - A^{-1} H \left(H^{\mathrm{T}} A^{-1} H\right)^{-1} H^{\mathrm{T}} A^{-1}\right\} \mathbf{y}. & (3.9) \end{aligned}$$

From this we see that:

$$\boldsymbol{\beta}|\sigma^2, \mathbf{y} \sim N\left(\widehat{\boldsymbol{\beta}}, \sigma^2 (H^T A^{-1} H)^{-1}\right), \tag{3.10}$$

and integrating (3.7) with respect to $\boldsymbol{\beta}$ gives us:

$$\sigma^2|\,\mathbf{y}, \theta \sim \text{InvGam}\left(\frac{n-q}{2}, \frac{(n-q-2)\widehat{\sigma}^2}{2}\right). \tag{3.11}$$

Now if we take the product of (3.5) and (3.10) and then integrate out $\boldsymbol{\beta}$, it results in:

$$\eta(\cdot)|\,\sigma^2, \mathbf{y} \sim GP(m^{**}(.), \sigma^2 c^{**}(.\,,.)), \tag{3.12}$$

where

$$\begin{aligned} m^{**}(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T \widehat{\boldsymbol{\beta}} + \mathbf{t}(\mathbf{x})^T A^{-1}(\mathbf{y} - H\widehat{\boldsymbol{\beta}}), & (3.13)\\ c^{**}(\mathbf{x}_i, \mathbf{x}_j) &= c(\mathbf{x}_i, \mathbf{x}_j) \;-\; \mathbf{t}(\mathbf{x}_i)^{\mathrm{T}} A^{-1} \mathbf{t}(\mathbf{x}_j) \;+ \\ & \quad \left(\mathbf{h}(\mathbf{x}_i)^{\mathrm{T}} - \mathbf{t}(\mathbf{x}_i)^{\mathrm{T}} A^{-1} H\right) \left(H^{\mathrm{T}} A^{-1} H\right)^{-1} \left(\mathbf{h}(\mathbf{x}_j)^{\mathrm{T}} - \mathbf{t}(\mathbf{x}_j)^{\mathrm{T}} A^{-1} H\right)^{\mathrm{T}}. & (3.14) \end{aligned}$$

Finally we must integrate out σ^2 after combining (3.11) and (3.12) and we are left, conditional on θ, with a t process with $n-q$ degrees of freedom. The posterior mean is $m^{**}(\mathbf{x})$ and can be used as a fast approximation of $\eta(\mathbf{x})$. The posterior covariance between $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$ is $\hat{\sigma}^2 c^{**}(\mathbf{x}_i, \mathbf{x}_j)$. The posterior mean is comprised of two parts. The first part corresponds to the prior mean (3.2), where $\hat{\boldsymbol{\beta}}$ is the expected value of $\boldsymbol{\beta}$, based on the training data. At the points $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the values of $\eta(\mathbf{x}_i), i \in \{1, \ldots, n\}$ are known as they were evaluated directly to produce the training data. The second part of the emulator, $\mathbf{t}(\mathbf{x})^T A^{-1}(\mathbf{y} - H\hat{\boldsymbol{\beta}})$ has the effect of ensuring that $m^{**}(\mathbf{x}_i) = \eta(\mathbf{x}_i)$, for $i \in \{1, \ldots, n\}$.

The values of θ are unknown and if we combine (3.4) with a prior for θ, $p(\theta)$, and (3.6) we obtain the posterior density of $\boldsymbol{\beta}, \sigma^2$ and θ:

$$f(\boldsymbol{\beta}, \sigma^2, \theta|\mathbf{y}) = p(\theta)\frac{|A|^{\frac{1}{2}}}{(\sigma^2)^{\frac{1}{2}(n+2)}(2\pi)^{\frac{p}{2}}}\exp\left[-(\mathbf{y}-H\boldsymbol{\beta})^T\frac{A^{-1}}{2\sigma^2}(\mathbf{y}-H\boldsymbol{\beta})\right]. \quad (3.15)$$

If we then integrate out $\boldsymbol{\beta}$ and σ^2 the result is:

$$f(\theta|\mathbf{y}) \propto p(\theta) \times (\widehat{\sigma}^2)^{-(n-q)/2}|A|^{-1/2}|H^{\mathrm{T}}A^{-1}H|^{-1/2}. \quad (3.16)$$

We cannot analytically integrate out Θ, the matrix which is comprised of the smoothness parameters, from the posterior distribution. The simplest option is to fix θ, having estimated its value, and then use this estimate as if it were the true value. One method of estimating Θ, and one we regularly use throughout this thesis, is to ascertain which values of Θ maximise the likelihood. It is common to first apply a log transformation on the smoothness parameters and then optimise the resulting function. Aside from the simulator runs, this is computationally the most expensive part of building an emulator. Andrianakis and Challenor (2009) derive the derivatives of the log likelihood function such that a gradient-based optimisation algorithm can be employed, which is expected to be more efficient. They also propose a function which limits the search space such that very large or very small values of θ cannot be sampled. This is because the likelihood can be very flat and then the 'maximum' is found at an inappropriately large value of θ. Fixing the smoothing parameters at point estimates ignores the uncertainty in θ but Kennedy and O'Hagan (2001) find this approach to be adequate. If we do not have large number of simulator runs to build an emulator though, we may wish to account for the uncertainty in θ and Andrianakis and Challenor (2009) show how this can be done using Monte Carlo integration.

3.2.4 Example

Here we demonstrate the methodology through a 1-dimensional example. The true function is

$$\eta(x) = x + \cos(x) + \sin(x),$$

and we choose a linear form for the prior mean, so $\mathbf{h}(\mathbf{x})^T = (1 \;\; x)$ and $q = 2$. The covariance function is $\mathrm{c}(x_i, x_j) = \exp\{-\theta\,(x_i - x_j)^2\}$ and we decide to run the simulator at the following, $n = 5$, design points:

$$x_1 = -4.10, x_2 = -1.80, x_3 = 0.80, x_4 = 1.90, x_5 = 4.20.$$

This yields the subsequent training data,

$$\mathbf{y}^T = (-3.857,\ -3.001,\ 2.214,\ 2.523,\ 2.838),$$

which is shown in Figure 3.1.

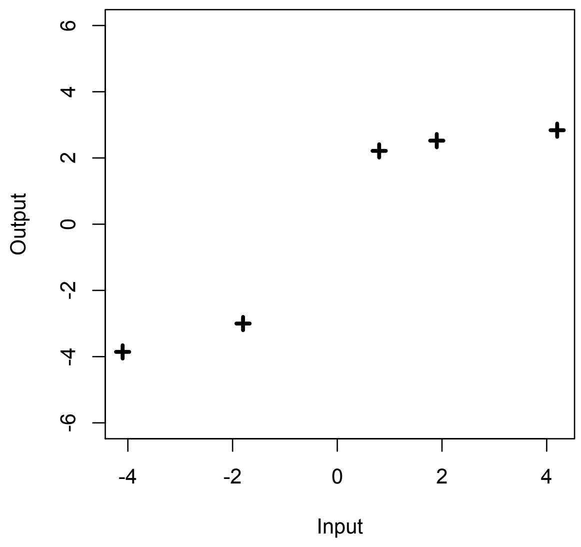


Figure 3.1: Training data for a 1-d example of a standard Gaussian process emulator.

We then follow the method described in Section 3.2.3 and derive the distribution of $\eta(x)$ conditional only on $\mathbf{y}$. Maximum likelihood estimation is employed to estimate the roughness parameter, θ. To test the emulator we evaluate the posterior mean and standard deviation at a number of new input points and as here the simulator is computationally cheap, we can also run the simulator at these points to evaluate the predictive performance of the emulator. The results are shown in Figure 3.2. The posterior mean is the red, dashed line and the red, dotted lines show the value of two standard deviations above and below the mean. For comparison the solid black line shows the true simulator output at these points.

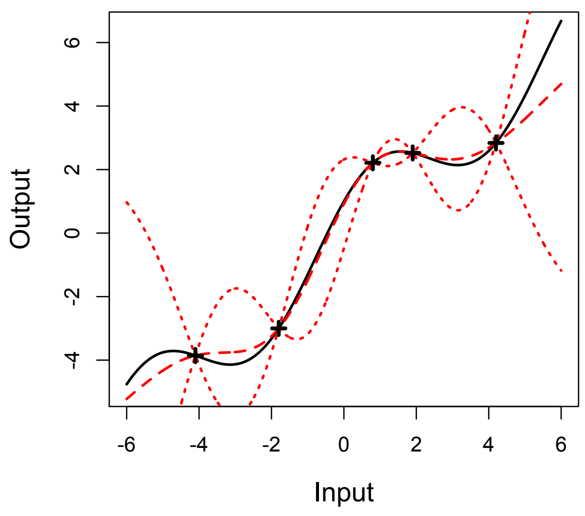


Figure 3.2: Emulator built with function output only.

In this example the posterior mean is close to the true value of the simulator; the predictions become worse and uncertainty much greater though, once it is forced to extrapolate. At the 5 design points the uncertainty pinches in to zero as expected, as the true value of the simulator at these points is known. The uncertainty becomes more appreciable the further away from a design point we predict at, how quickly the uncertainty grows between design points depends on the roughness parameter, θ. If we increase the number of design points we would expect the predictions to become closer to the simulator output and the uncertainty to decrease.

3.3 Validation

Having built an emulator, as in Section 3.2, it is important to validate it. Figure 3.2 in Section 3.2.4 shows the performance of an emulator, but in general such plots would not be possible as if it were computationally feasible to execute the simulator at every input point as in Figure 3.2.4, then there would be no need to build an emulator! More likely, we will only be able to afford, computationally, a small validation dataset and wish to use it to formally validate the emulator.

Bastos and O'Hagan (2009) introduce a number of diagnostics for validating a Gaussian process emulator. These include examining standardised prediction errors given by:

$$\frac{(\mathbf{y}_i' - m^{**}(\mathbf{x}_i'))}{\sqrt{(}c^{**}(\mathbf{x}_i', \mathbf{x}_i')))},$$

for $i \in 1 \ldots n'$, where $\mathbf{y}_i'$ are the realisations of the simulator at the validation design points, $\mathbf{x}_i'$, and n' is the number of validation points. Assuming the training data is large enough, the standardised prediction errors can be considered to be standard, normally distributed and therefore if more than about 5% of the errors lie outside $[-2, 2]$, this would imply conflict between the simulator and emulator. The location and sign of the errors could give an indication as to why there is conflict; for example if there are large errors, all of the same sign and in the same part of the input space this could mean that either the value of β, or the form of the mean function is inappropriate.

One problem with the standardised prediction errors as a diagnostic is that the errors are correlated. Independent standardised errors can be generated by:

$$G^{-1}(\mathbf{y}' - m^{**}(\mathbf{x}')),$$

where G is a matrix such that $c^{**}(\mathbf{x}', \mathbf{x}') = G\,G^T$. Bastos and O'Hagan (2009) recommend permuting the validation data set in order of variance, and then applying a pivoted Cholesky decomposition to obtain G. The result is independent pivoted Cholesky errors where each error is associated with a unique validation point, but in a different order to that of the validation dataset. Groups of unusually large or small errors in various parts of the sequence then imply possible problems with parameter estimates in the emulator.

Another diagnostic suggested by Bastos and O'Hagan (2009) is the Mahalanobis distance, given by:

$$(\mathbf{y}' - m^{**}(\mathbf{x}'))^T (c^{**}(\mathbf{x}', \mathbf{x}'))^{-1} (\mathbf{y}' - m^{**}(\mathbf{x}')). \tag{3.17}$$

This diagnostic gives a measure of overall fit and should be compared to its reference distribution, which is the F distribution with n' and $(n - q)$ degrees of freedom. This is because (3.17) is a quotient of 2 chi-square random variables; a proof of the result is given in Bastos and O'Hagan (2009). The mean and variance are n' and $\frac{2\,n'(n'+n-q-2)}{n-q-4}$ respectively. Unusually large or small values of the observed Mahalanobis distance in comparison with the theoretical distribution indicate a problem with the emulator.

If no validation data are available and there isn't enough training data to hold back a subset of runs for validation, then the diagnostics as described above cannot be employed. In this situation one option is the *leave one out* (LOO) cross validation diagnostic, proposed by Craven and Wahba (1979): we omit one observation from the training data, $y_i = \eta(\mathbf{x}_i)$ and the corresponding inputs, $\mathbf{x}_i$. We can then build an emulator with the remaining $n-1$ observations which yields the posterior distribution of $\eta(\cdot)$ given the $n-1$ observations. Using the posterior mean, $m^{**}(\cdot)$, we can then predict the response at the point which was left out and compare the prediction of the emulator, $m^{**}(\mathbf{x}_i)$, along with the associated measure of uncertainty, $c^{**}(\mathbf{x}_i, \mathbf{x}_i)$, with the true response, y_i. This is then repeated for $i \in 1 \ldots n$. In such a way therefore, LOO is a diagnostic which uses only the training data to assess the predictive performance of the emulator. In practise, the value of the smoothness parameters, θ, are usually estimated with all n observations and these estimates are then plugged in to each emulator built with $n-1$ observations in the LOO diagnostic.

In Chapters 4, 5 and 6 we make use of all the diagnostic measures described above to validate various emulators.

3.4 Emulator variants

The methodology described in Section 3.2 gives the Gaussian process approach to building an emulator but there are alternatives. A Bayes-Linear emulator for example, does not provide a full probability distribution for the outputs of a complex model; instead, just means and variances are given. This can make computations tractable even with very large datasets. As briefly discussed in Section 4.6 of Chapter 4, Killeya and Goldstein (2007) use a Bayes-Linear approach in their emulation of a compartmental model of plankton cycles. A comprehensive account of Bayes-Linear theory is given by Goldstein and Woof (2007).

There are numerous variants to the methods already discussed in this chapter which allow for emulation in more complex situations. For example, emulating time series output of a dynamic model, combining multiple emulators and building an emulator where we have multiple simulators of the same real world process.

For further detail on these and other emulator variants see the MUCM toolkit at *http://mucm.aston.ac.uk/MUCM/MUCMToolkit*.

Chapter 4

Emulating function output

4.1 Introduction

It has been recognised for some time that derivative information has the potential to improve on emulation of complex models and in some cases to reduce computational expense. The benefit of observing derivatives in modelling nonlinear, dynamic systems is discussed in Leith *et al.* (2002), Solak *et al.* (2003) and Azman and Kocijan (2005). Morris *et al.* (1993) extend the work of Currin *et al.* (1991) by considering how derivatives can be used in Gaussian process emulation and the benefit of observing derivatives in compartmental models is discussed in Killeya (2004).

In this chapter we adopt a similar approach to Morris *et al.* (1993) and investigate the use of derivative information in Gaussian process emulators, with an objective of predicting the function output of a complex model. We expand on work done previously by comparing the benefit of learning derivatives with the computational cost of obtaining them; in such a way we can then directly compare emulators with and without derivatives. The purpose of this work is to provide an answer to the question: *is it more efficient to build an emulator with derivative information?* Clearly an accurate response to such a question would be model-dependent. We tackle the issue, therefore, by looking at a number of toy models with varying levels of smoothness (Section 4.4), and by a detailed investigation into a real complex model of which the adjoint model exists and can be run to produce derivatives (Section 4.5). We begin this chapter though by detailing the methodology required to include derivative information when building a Gaussian process emulator.

4.2 Gaussian process emulators with derivatives

In this section we detail the methodology to build an emulator with derivative information in addition to the function output. We adopt the same approach as in Chapter 3, which therefore gives rise to some repetition between the following Sections and Chapter 3. This is included for completeness.

4.2.1 The Gaussian process model

O'Hagan (1992) shows how Gaussian processes can be used to model derivatives of a simulator, $\eta(\cdot)$, assuming $\eta(\cdot)$ is differentiable everywhere. The derivatives of a Gaussian process remain a Gaussian process and this allows us to adopt a similar approach to that of the standard setup as discussed in detail in Chapter 3.

As in the conventional case, we first consider the model or simulator, $\eta(\cdot)$, as an unknown function, as until the model is run the output values are unknown. We then need to choose mean and covariance functions. As in Chapter 3 we define the mean function to be $E[\eta(\mathbf{x})|\,\boldsymbol{\beta}] = \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}$ where $\mathbf{h}(\mathbf{x})^T$ is a $1 \times q$ vector of known, regressor functions of $\mathbf{x}$ and $\boldsymbol{\beta}$ is a $q \times 1$ vector comprising of unknown coefficients. As we are including derivative information in the training data, we should ensure $\mathbf{h}(\cdot)$ is differentiable across the range of $\mathbf{x}$ we are interested in; at a minimum, $\mathbf{h}(\cdot)$ must be differentiable at every design point that we have derivative information at. This will then lead to the derivative of the mean function: $E\left[\frac{\partial}{\partial x^{(d)}}\eta(\mathbf{x})\middle|\,\boldsymbol{\beta}\right] = \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}$, where $x^{(d)}$ refers to input d at point $\mathbf{x}$. We bring these functions together in $\tilde{\mathbf{h}}(\mathbf{x}, d)$, which is defined as:

$$\tilde{\mathbf{h}}(\mathbf{x}, d)^T\boldsymbol{\beta} = \begin{cases} \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} & \text{for } d = 0 \\ \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} & \text{for } d \neq 0\ . \end{cases}$$

We have the location in the input space represented by $\mathbf{x}$ and the value of d determines whether or not we are including a derivative at that point.

Similarly to $\mathbf{h}(\cdot)$, the covariance function should be differentiable for the relevant range and at a minimum, at every design point that we have derivative information at. We define $\sigma^2 c(\mathbf{x}_i, \mathbf{x}_j)$ to be the covariance between $\eta(\mathbf{x}_i)$ and

$\eta(\mathbf{x}_j)$ where σ^2 is an unknown scale hyperparameter and $c(\cdot,\cdot)$ is the correlation function. The correlation $c\left(\mathbf{x}_i, \frac{\partial \mathbf{x}_j}{\partial x_j^{(k)}}\right)$, which is the correlation between a point, $\mathbf{x}_i$ and a derivative w.r.t input k at $\mathbf{x}_j$, (denoted by $x_j^{(k)}$), is $\frac{\partial}{\partial x_j^{(k)}} c(\mathbf{x}_i, \mathbf{x}_j)$. The correlation between a derivative w.r.t input k at $\mathbf{x}_i$, (denoted by $x_i^{(k)}$) and a derivative w.r.t input l at $\mathbf{x}_j$, (denoted by $x_j^{(l)}$), is $\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(l)}} c(\mathbf{x}_i, \mathbf{x}_j)$ and so the correlation function should actually be twice differentiable. Proof of the correlations involving derivatives is given in Papoulis (1991), Chapter 10. We bring these functions together in $\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\}$ which is defined as:

$$\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} = \begin{cases} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = d_j = 0 \\ \frac{\partial}{\partial x_i^{(d_i)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i \neq 0 \text{ and } d_j = 0 \\ \frac{\partial^2}{\partial x_i^{(d_i)} \partial x_j^{(d_j)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i, d_j \neq 0\,. \end{cases}$$

As a result of the smoothness of $\eta(\cdot)$, when two points, $\mathbf{x}_i$ and $\mathbf{x}_j$ are close there should be a high correlation between the corresponding $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$, and as the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ increases, the correlation should decrease. The smoothness property of the model therefore implies that the output will be similar for inputs close together.

4.2.2 Choice of correlation function

To calculate the correlation between two derivatives we differentiate the correlation function twice, thus we should take the differentiability of the function into consideration when selecting a correlation function. The most common approach, and the one we use regularly throughout this thesis, is to define the correlation function to have the Gaussian form: $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\}$, where Θ is a diagonal matrix of positive smoothness parameters, $\theta\{k\}$ with $k \in \{1, \ldots, p\}$ and p is the number of inputs. Note that alternative notation also popular in the literature is to replace the ‘smoothness’ parameter, $\theta\{k\}$, with a correlation length, $l\{k\}$, such that $l\{k\} = \frac{1}{\sqrt{\theta\{k\}}}$. Throughout this thesis though we adopt the ‘smoothness’ parameter, $\theta\{k\}$, notation. The correlation then between a point, $\mathbf{x}_i$, and a derivative w.r.t input k at point j, $x_j^{(k)}$, is:

$$\frac{\partial}{\partial x_j^{(k)}} c(\mathbf{x}_i, \mathbf{x}_j) = 2\,\theta\{k\} \left(x_i^{(k)} - x_j^{(k)}\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\},$$

the correlation between two derivatives w.r.t input k but at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(k)}} c(\mathbf{x}_i, \mathbf{x}_j) = \left(2\,\theta\{k\} - 4\,\theta^2\{k\} \left(x_i^{(k)} - x_j^{(k)}\right)^2\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\},$$

and finally the correlation between two derivatives w.r.t inputs k and l, where $k \neq l$, at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(l)}} c(\mathbf{x}_i, \mathbf{x}_j) = 4\,\theta\{k\}\,\theta\{l\} \left(x_j^{(k)} - x_i^{(k)}\right) \left(x_i^{(l)} - x_j^{(l)}\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\}.$$

How the correlations vary as the distance between points changes is shown for the Gaussian covariance function in Figure 4.1. Killeya (2004) look further at the covariances involving derivatives and these are shown, for the Gaussian covariance function with varying levels of smoothness controlled by the parameter θ, in Figure 4.1 as well.

The Gaussian correlation function, described above, belongs to a class of correlation functions known as the exponential power form, used by Sacks *et al.* (1989), and given by:

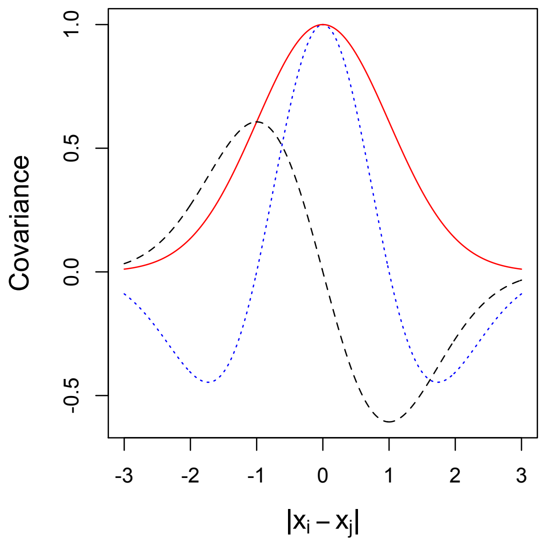
$$c(\mathbf{x}_i, \mathbf{x}_j) = \prod_{k=1}^{p} \exp\left(-\theta\{k\} \left| x_i^{(k)} - x_j^{(k)} \right|^{\rho}\right),$$

where $0 < \rho \leq 2$ and ρ can also vary with k if required. We are not able to make use of this flexibility though and must set $\rho = 2$ for all k, which results in the Gaussian form. This is because if we set $\rho = 1$ we are left with the exponential covariance function which is not differentiable, and neither are the resulting functions when $\rho < 1$. Finally, for $1 < \rho < 2$, the resulting function is only once differentiable.

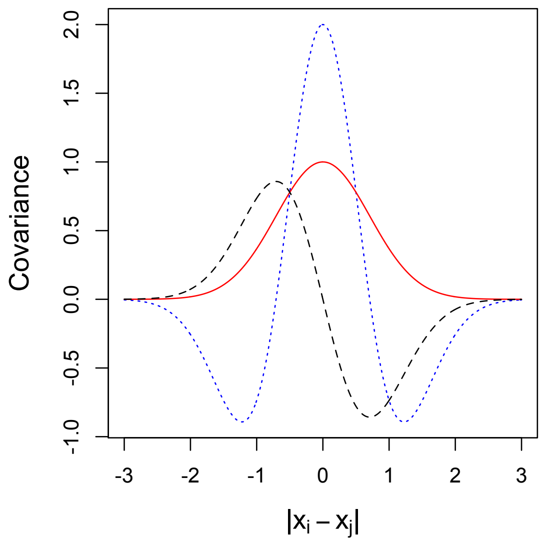
The Matérn class of correlation functions are an alternative to the exponential power functions. Stein (1999) recommends the use of this form of correlation function due to its flexibility, yet still with a manageable number of parameters. The form of a Matérn correlation function is:

$$c(\mathbf{x}_i, \mathbf{x}_j) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)}\right)^{\nu} K_{\nu} \left(\sqrt{2\nu(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)}\right), \tag{4.1}$$

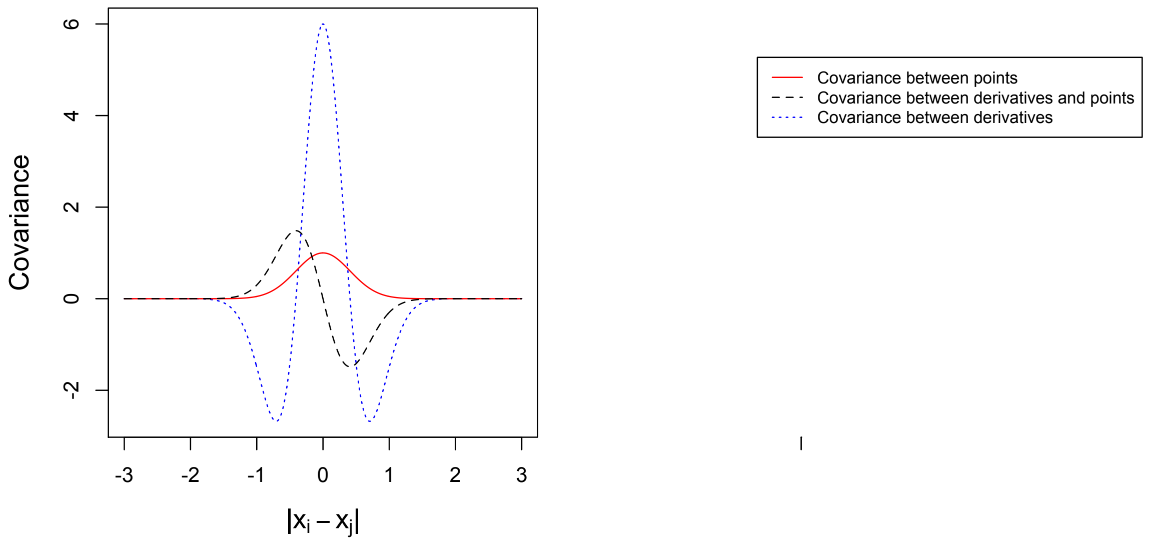
where K_ν is a modified Bessel function. The parameter ν must be positive and is often half-integer, resulting in a much simpler form of (4.1). The value assigned to ν controls the differentiability of the function and if we let $\nu \to \infty$ the result



(a) $\theta = 0.5$



(b) $\theta = 1$



(c) $\theta = 3$

Figure 4.1: Gaussian covariance function shown as the distance between points varies.

is the Gaussian form, while setting $\nu = \frac{1}{2}$ gives rise to the exponential correlation function. Rasmussen and Williams (2006) suggest that $\nu = \frac{3}{2}$ and $\nu = \frac{5}{2}$ are most interesting in the subject of machine learning but again for the purpose of this thesis we must ensure twice differentiability. The Matérn correlation function with $\nu = \frac{3}{2}$ is only once differentiable but if we set $\nu = \frac{5}{2}$ the resulting correlation function is twice differentiable and therefore is a valid alternative to the Gaussian form in this work. See Rasmussen and Williams (2006) for further discussion of correlation functions.

The correlation functions discussed in this section are summarised in Table 4.1, where we adopt the notation $\mathcal{Q} = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\Theta(\mathbf{x}_i - \mathbf{x}_j)$.

Correlation function	Formula	Differentiable?
Gaussian (Exponential power form with $\rho = 2$)	$\exp(-\mathcal{Q})$	Infinitely
Exponential (Exponential power form with $\rho = 1$)	$\prod_{k=1}^{p} \exp\left(-\theta\{k\}\left\lvert x_i^{(k)} - x_j^{(k)}\right\rvert\right)$	No
Matérn with $\nu = \frac{3}{2}$	$\left(1 + \sqrt{3\mathcal{Q}}\right)\exp\left(-\sqrt{3\mathcal{Q}}\right)$	Once
Matérn with $\nu = \frac{5}{2}$	$\left(1 + \sqrt{5\mathcal{Q}} + \frac{5\mathcal{Q}}{3}\right)\exp\left(-\sqrt{5\mathcal{Q}}\right)$	Twice

Table 4.1: A summary of some correlations functions.

4.2.3 Design

We now need to create a design which consists of a set of points in the input space at which the simulator or adjoint is to be run to create the training sample. A design for the standard case results in an ordered set of n points: $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where each $\mathbf{x}$ is a location in the input space. Here though, we need a design which in addition to specifying the location of the inputs, also determines at which points we require function output and at which points we require first derivatives. We arrange this information in the design $\tilde{D} = \{(\mathbf{x}_k, d_k)\}$, where $k = \{1, \ldots, \tilde{n}\}$ and $d_k \in \{0, 1, \ldots, p\}$. We have $\mathbf{x}_k$ which refers to the location in the design and d_k determines whether at point $\mathbf{x}_k$ we require function output or a first derivative w.r.t one of the inputs. Each $\mathbf{x}_k$ is not distinct as we may have a derivative and the function output at point $\mathbf{x}_k$, or we may require a derivative w.r.t several inputs at point $\mathbf{x}_k$. The simulator, $\eta(\cdot)$, or the adjoint or derivative, of the simulator, $\tilde{\eta}(\cdot)$, (depending on the value of each d), is then run at each of the input configurations. This results in our training data: $\tilde{\mathbf{y}} = \tilde{\eta}(\tilde{D})$, a vector of length $\tilde{n}$.

Morris *et al.* (1993) look at optimal design when derivatives are observed in addition to the model response at all the locations $\mathbf{x}_i$. Through an example they compare 4 designs, each of size $n = 10$. The example model has 8 input dimensions so $\tilde{n} = 90$. The designs are as follows:

1. *Latin hypercube design.* For more detail on Latin hypercube samples see Chapter 3.

2. *Maximin design.* Maximin designs tend to favour the corners of the design space and for this reason an initial design, D_{initial}, is created where each input dimension is assigned the value 0 or 1 with equal probability. The measure ϕ_{initial} is then calculated for this design where,

$$\phi = \frac{1}{\underline{d}(D)} \left[\sum_{i=1}^{n-1} \sum_{i+1}^{n} \left(\frac{\underline{d}(D)}{d_{ij}(D} \right)^{1000} \right]^{1/1000}.$$

We have d_{ij} which is the distance between sites i and j in the design, D, and $\underline{d}(D)$ is the minimum distance between any two sites in D. All distances are Euclidean. This measure is chosen instead of simply calculating $\underline{d}(D)$ because in addition to maximising the minimum distance, this design aims to minimise the number of pairs of input sites which are separated by the minimum distance. An input site, $\mathbf{x}_i$, is then randomly selected from D_{initial} and each input dimension of $\mathbf{x}_i$ changed from 0 to 1 (or vice versa, depending on its initial value) with probability 0.3, to produce $\mathbf{x}_{\text{cand}}$. The measure ϕ is recalculated and if $\phi_{\text{cand}} < \phi_{\text{initial}}$ then the candidate input is accepted. The algorithm continues for a specified number of iterations.

3. *Maximin Latin hypercube design.* This design is produced by generating a random Latin hypercube as an initial design and ϕ_{initial}, as defined in design (2.), calculated. A candidate new design, D_{cand} is generated by exchanging

2 entries of a randomly chosen column and ϕ_{cand} is then calculated. As in design (2.), the candidate design is accepted if $\phi_{\text{cand}} < \phi_{\text{initial}}$ and the algorithm continues in this way.

Note that this method differs to how we generate maximin Latin hypercube designs later in this chapter and also in Chapters 5 and 6. The method we adopt is to generate a number of random Latin hypercube samples and then select the design which has the maximum minimum distance.

4. *Modified maximin design.* This design is generated by take the resulting maximin design of (2.), which has five 0s and five 1s in each column. The 0's within each of column are then randomly replaced with the values $0, \frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{4}{9}$. Similarly the 1's are replaced with $\frac{5}{9}, \frac{6}{9}, \frac{7}{9}, \frac{8}{9}, 1$.

Morris *et al.* (1993) compare the designs described above via the prediction error of an emulator resulting from each design and find that the two 'compromise' designs, numbers (3.) and (4.), appear to be superior.

The choice of the sample size, n, for the standard case is discussed in Loeppky *et al.* (2009) and as a rule of thumb, $n = 10p$, where p is the number of inputs is recommended. There is not, however, such a guide for what $\tilde{n}$ might be. If we choose to obtain function output and the first derivatives w.r.t to all inputs at every location in the design, then we would expect that fewer than $10p$ locations would be required; how many fewer though, is difficult to estimate. This is discussed further in Section 4.7 but without any formal results.

4.2.4 Building the emulator

From Sections 4.2.1 and 4.2.3 above, we have training data, $\tilde{\mathbf{y}}$, and since we have represented $\eta(\cdot)$ by a Gaussian process then the density of $\tilde{\mathbf{y}}$ conditional on $\boldsymbol{\beta}$ and σ^2 is:

$$\tilde{\mathbf{y}}|\,\boldsymbol{\beta}, \sigma^2, \theta \sim N(\tilde{H}\boldsymbol{\beta}, \sigma^2\tilde{A}), \tag{4.2}$$

where $\tilde{H} = [\tilde{h}(\mathbf{x}_1, d_1), \dots, \tilde{h}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})]^{\mathrm{T}}$ and $\tilde{A}$ is the $\tilde{n} \times \tilde{n}$ matrix of correlations between points, between points and derivatives and between derivatives themselves, in the training data: $\tilde{A} = \tilde{c}(\tilde{D}, \tilde{D})$. Our interest in this chapter is emulating model response, so we wish to update the distribution of $\eta(.)$ conditional on the observations in the training data which consist of model response and derivatives. We begin this process by partitioning in the following way:

$$\begin{pmatrix} \eta(\cdot) \\ \tilde{\mathbf{y}} \end{pmatrix},$$

with mean $= \begin{pmatrix} \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} \\ \tilde{H}\boldsymbol{\beta} \end{pmatrix}$, and covariance $= \begin{pmatrix} \sigma^2 c(\mathbf{x},\mathbf{x}) & \tilde{\mathbf{t}}(\mathbf{x})^T \\ \tilde{\mathbf{t}}(\mathbf{x}) & \sigma^2\tilde{A} \end{pmatrix}$, where $\tilde{\mathbf{t}}(\mathbf{x})$ consists of the correlations between the point we are emulating the model response at, and the training data:

$$\begin{aligned} \tilde{\mathbf{t}}(\mathbf{x})^T &= \text{Corr}\left[\eta(\mathbf{x}), \tilde{\eta}(\mathbf{x}_1, d_1)\right], \ldots, \text{Corr}\left[\eta(\mathbf{x}), \tilde{\eta}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})\right] \\ &= \left[\tilde{c}\{(\mathbf{x},0),(\mathbf{x}_1,d_1)\}, \ldots, \tilde{c}\{(\mathbf{x},0),(\mathbf{x}_{\tilde{n}},d_{\tilde{n}})\right], \\ &= \tilde{c}\{(\mathbf{x},0),\tilde{D}\}. \end{aligned} \tag{4.3}$$

We have $\tilde{\mathbf{t}}(\mathbf{x},d)^T = \tilde{\mathbf{t}}(\mathbf{x})^T$, as $d = 0$ because we want to emulate function output here. We still require the tilde, ($\tilde{}$), symbol, so we have $\tilde{\mathbf{t}}(\mathbf{x})^T$ rather than $\mathbf{t}(\mathbf{x})^T$ because we must include the correlations between the derivatives in the training data and the point where we are predicting the function output. We can now use standard techniques of conditioning multivariate normal distributions to give:

$$\eta(\cdot)|\,\boldsymbol{\beta}, \sigma^2, \tilde{\mathbf{y}} \sim N(m^*(.), \sigma^2 c^*(.\,,.)), \tag{4.4}$$

where

$$\begin{aligned} m^*(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} + \tilde{\mathbf{t}}(\mathbf{x})^T\tilde{A}^{-1}(\tilde{\mathbf{y}} - \tilde{H}\boldsymbol{\beta}), \\ c^*(\mathbf{x}_i,\mathbf{x}_j) &= c(\mathbf{x}_i,\mathbf{x}_j) - \tilde{\mathbf{t}}(\mathbf{x}_i)^T\tilde{A}^{-1}\tilde{\mathbf{t}}(\mathbf{x}_j), \\ \tilde{\mathbf{y}}^T &= \tilde{\eta}(\tilde{D}) = \{\tilde{\eta}(\mathbf{x}_1,d_1), \ldots, \tilde{\eta}(\mathbf{x}_{\tilde{n}},d_{\tilde{n}})\}. \end{aligned}$$

We now want to obtain the distribution of $\eta(\cdot)|\,\tilde{\mathbf{y}}$ unconditional on $\boldsymbol{\beta}$ and σ^2. If we have genuine prior information about $\boldsymbol{\beta}$ and σ^2, then this can be utilised when specifying the prior distributions for them. In many situations though, there is enough information in the training data about these parameters and it suffices to specify a weak prior for both $\boldsymbol{\beta}$ and σ^2:

$$p(\boldsymbol{\beta}, \sigma^2) \propto \sigma^{-2}. \tag{4.5}$$

Applying Bayes Theorem with (4.5) and (4.2) results in a joint Normal Inverse Gamma posterior distribution for $(\boldsymbol{\beta}, \sigma^2)$:

$$f(\boldsymbol{\beta}, \sigma^2|\tilde{\mathbf{y}}, \Theta) \propto {\sigma^2}^{\frac{\tilde{n}+2}{2}} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}})^T\tilde{H}^T\tilde{A}^{-1}\tilde{H}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}) + (\tilde{n} - q - 2)\widehat{\sigma^2}\right\}, \tag{4.6}$$

where

$$\begin{aligned}\widehat{\beta} &= \left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{\mathbf{y}}, & (4.7)\\ \widehat{\sigma}^2 &= (\tilde{n}-q-2)^{-1}\tilde{\mathbf{y}}^{\mathrm{T}}\left\{\tilde{A}^{-1}-\tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\right\}\tilde{\mathbf{y}}. & (4.8)\end{aligned}$$

From this we see that:

$$\boldsymbol{\beta}|\sigma^2,\tilde{\mathbf{y}},\Theta \sim N\left(\widehat{\boldsymbol{\beta}},\sigma^2(\tilde{H}^T\tilde{A}^{-1}\tilde{H})^{-1}\right), \qquad (4.9)$$

and integrating (4.5) with respect to $\boldsymbol{\beta}$ gives us:

$$\sigma^2|\,\tilde{\mathbf{y}},\Theta \sim \mathrm{InvGam}\left(\frac{\tilde{n}-q}{2},\frac{(\tilde{n}-q-2)\widehat{\sigma}^2}{2}\right). \qquad (4.10)$$

Now if we take the product of (4.4) and (4.9) and then integrate out $\boldsymbol{\beta}$, it results in:

$$\eta(\cdot)|\sigma^2,\tilde{\mathbf{y}},\Theta \sim GP(m^{**}(.),\sigma^2c^{**}(.\,,.)), \qquad (4.11)$$

where

$$\begin{aligned}m^{**}(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T\widehat{\boldsymbol{\beta}}+\tilde{\mathbf{t}}(\mathbf{x})^T\tilde{A}^{-1}(\tilde{\mathbf{y}}-\tilde{H}\widehat{\boldsymbol{\beta}}), & (4.12)\\ c^{**}(\mathbf{x}_i,\mathbf{x}_j) &= c(\mathbf{x}_i,\mathbf{x}_j)\,-\,\tilde{\mathbf{t}}(\mathbf{x}_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{\mathbf{t}}(\mathbf{x}_j)\,+\\ &\left(\mathbf{h}(\mathbf{x}_i)^{\mathrm{T}}-\tilde{\mathbf{t}}(\mathbf{x}_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\left(\mathbf{h}(\mathbf{x}_j)^{\mathrm{T}}-\tilde{t}(\mathbf{x}_j)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{\mathrm{T}}. & (4.13)\end{aligned}$$

Finally we must integrate out σ^2 after combining (4.10) and (4.11) and we are left, conditional on Θ, with a t process with $\tilde{n}-q$ degrees of freedom. The posterior mean function is $m^{**}(\cdot)$ and posterior covariance function is $c^{**}(\cdot,\cdot)$.

The values of Θ are unknown and if we combine (4.2) with a prior for Θ, $p(\Theta)$, and (4.5) we obtain the posterior density of $\boldsymbol{\beta},\sigma^2$ and Θ:

$$f(\boldsymbol{\beta},\sigma^2,\Theta|\tilde{\mathbf{y}}) = p(\Theta)\frac{|\tilde{A}|^{\frac{1}{2}}}{(\sigma^2)^{\frac{1}{2}(\tilde{n}+2)}(2\pi)^{\frac{p}{2}}}\exp\left[-(\tilde{\mathbf{y}}-\tilde{H}\boldsymbol{\beta})^T\frac{\tilde{A}^{-1}}{2\sigma^2}(\tilde{\mathbf{y}}-\tilde{H}\boldsymbol{\beta})\right]. \qquad (4.14)$$

If we then integrate out $\boldsymbol{\beta}$ and σ^2 the result is:

$$f(\Theta|\tilde{\mathbf{y}}) \propto p(\Theta)\times(\widehat{\sigma}^2)^{-(\tilde{n}-q)/2}|\tilde{A}|^{-1/2}|\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}|^{-1/2}. \qquad (4.15)$$

It is not possible to handle Θ analytically and so the simplest option is to fix Θ, having estimated its value, and then use this estimate as if it were the true value. This ignores the uncertainty in Θ but Kennedy and O'Hagan (2001) find this approach to be adequate. We discuss methods for dealing with Θ further in Chapter 3.

Since the derivatives of a Gaussian process remain a Gaussian process we can validate an emulator built with derivative information using the same approaches to that of a standard emulator. Gaussian process validation is discussed more fully in Chapter 3.

4.2.5 Example

In Chapter 3 the methodology for a standard Gaussian process emulator is illustrated with a 1-dimensional example. This is covered in detail in Section 3.2.4. Here, we repeat that example but include derivative information in the training data and follow the methods described in Section 4.2.4. The true function is

$$\eta(x) = x + \cos(x) + \sin(x),$$

and we choose a linear form for the prior mean, so

$$\tilde{\mathbf{h}}(x,d)^T = \begin{cases} (1\ x) & \text{for } d = 0 \\ (1) & \text{for } d \neq 0\,, \end{cases}$$

and $q = 2$. The covariance function is $\mathrm{c}(x_i, x_j) = \exp\{-\theta\,(x_i - x_j)^2\}$. In Section 3.2.4 we choose the design:

$$D = \{-4.1, -1.8, 0.8, 1.9, 4.2\}\,,$$

and so $n = 5$. This yields the training data:

$$\mathbf{y}^T = (-3.857,\ -3.001,\ 2.214,\ 2.523,\ 2.838).$$

Here, we choose the same location of design points and decide to evaluate the partial derivative of $\eta(x)$ w.r.t x at all 5 points, which gives us the $\tilde{n} = 10$ design:

$$\begin{aligned} \tilde{D} &= \{(x_1, d_1), (x_2, d_2), \ldots, (x_{10}, d_{10})\} \\ &= \{(-4.1, 0), (-1.8, 0), (0.8, 0), (1.9, 0), (4.2, 0), \\ &\qquad (-4.1, 1), (-1.8, 1), (0.8, 1), (1.9, 1), (4.2, 1)\}. \end{aligned}$$

We then evaluate the simulator and the partial derivatives of the simulator at the design points to give:

$$\tilde{\mathbf{y}}^T = (-3.857,\ -3.001,\ 2.214,\ 2.523,\ 2.838, -0.393, 1.747, 0.979, -0.270, 1.381).$$

As in the example without derivative information, we adopt the Gaussian covariance function which in one dimension is:

$$\tilde{c}\{(x_i, d_i), (x_j, d_j)\} = \begin{cases} \exp\{-\theta\,(x_i - x_j)^2\} & \text{Case 1} \\ 2\,\theta\,(x_j - x_i)\exp\{-\theta\,(x_i - x_j)^2\} & \text{Case 2} \\ (2\,\theta - 4\,\theta^2\,(x_i - x_j)^2)\exp\{-\theta\,(x_i - x_j)^2\} & \text{Case 3} \end{cases}$$

where Case 1 is for $i, j \in \{1, \ldots, 5\}$ and therefore is the correlation between points, Case 2 is for $i \in \{6, \ldots, 10\}, j \in \{1, \ldots, 5\}$ which refers to the correlation between points and derivatives, and Case 3 is for $i, j \in \{6, \ldots, 10\}$ which is the correlation between derivatives. Each Case provides sub-matrices of correlations and we arrange them as follows:

$$\tilde{A} = \left[\begin{array}{ccc} \uparrow & \longleftarrow\ n\ \longrightarrow & \longleftarrow\ n\ \longrightarrow \\ n & \text{Case 1} & \text{Case 2} \\ \downarrow & & \\ \uparrow & & \\ n & -(\text{Case 2}) & \text{Case 3} \\ \downarrow & & \end{array}\right]. \tag{4.16}$$

The $\tilde{n} \times \tilde{n}$ matrix, $\tilde{A}$, is symmetric and within $\tilde{A}$ we have symmetric sub-matrices, Case 1 and Case 3. Case 1 is a 5×5 matrix and is the same as in the example in Section 3.2.4. Since we are including the derivative at each of the 5 design points, Case 2 and 3 sub-matrices are also of size 5×5.

Following the method described in Section 4.2.4, we now derive the distribution of $\eta(x)$ conditional only on $\tilde{\mathbf{y}}$. Maximum likelihood estimation is employed to estimate the smoothness parameter, θ. To test the emulator we run it at a number of new input points and as here the simulator is computationally cheap, we can also run the simulator at these points to evaluate the predictive performance of the emulator. Figure 4.2 shows the performance of this emulator. The posterior mean is the blue, dashed line and the blue, dotted lines show the value of two standard deviations above and below the mean. In order to assess the emulator, the true simulator output at these points is shown by the solid black line. We can see from Figure 4.2 that in this example, the posterior mean is very close to the true value of the simulator and the uncertainty is very small.

For comparison, we repeat Figure 3.2 from the same example but without derivatives (see Section 3.2.4 for a full discussion of this example). The posterior mean of the emulator without derivatives is the red, dashed line and the red, dotted lines show the value of two standard deviations above and below the

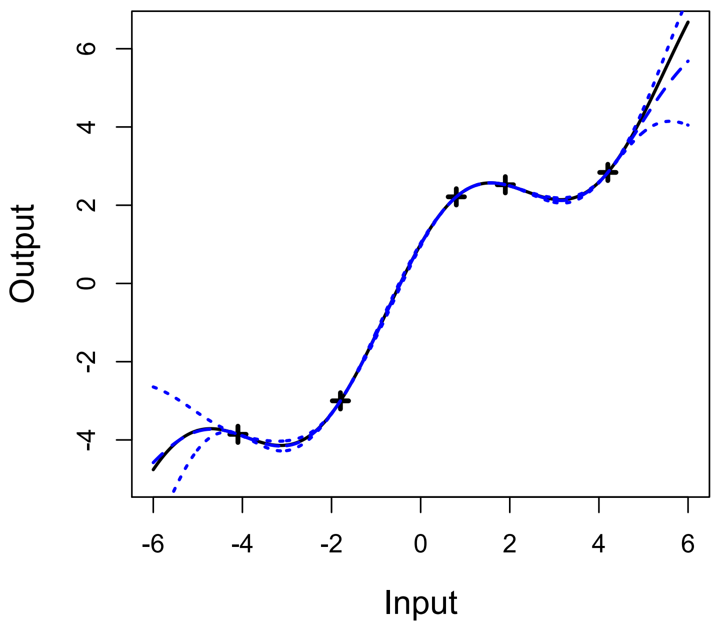


Figure 4.2: Emulator with 5 design points and derivative information

mean. It is clear if we compare Figure 3.2 with Figure 4.2 that when we include the 5 derivatives in the training data in addition to the 5 response values, the performance of the resulting emulator improves.

The performance of the emulator with derivatives (Figure 4.2), while very good, is such that it is difficult to identify precisely how and where the derivative information is having an effect. Due to this we now repeat the example but remove two of the locations from the design and the subsequent derivatives: $(x_2, d_2), (x_4, d_4), (x_7, d_7)$ and (x_9, d_9). Figure 4.3 illustrates the performance of this emulator. The blue dashed line, representing the posterior mean, is close to the true simulator output which is given by the black, solid line. The uncertainty, as in the previous figures, is shown by the blue, dotted lines. The uncertainty reduces to zero at design points, as expected, but whereas in Figure 3.2 the uncertainty becomes appreciable once we start predicting away from a design point, here the uncertainty remains very small for predictions close to the design points. It is the derivative information in the model which allows for this reduced uncertainty. This allows for quite accurate extrapolation in areas quite close to the last training data point. The true value of the simulator at $x = 5$, for example,

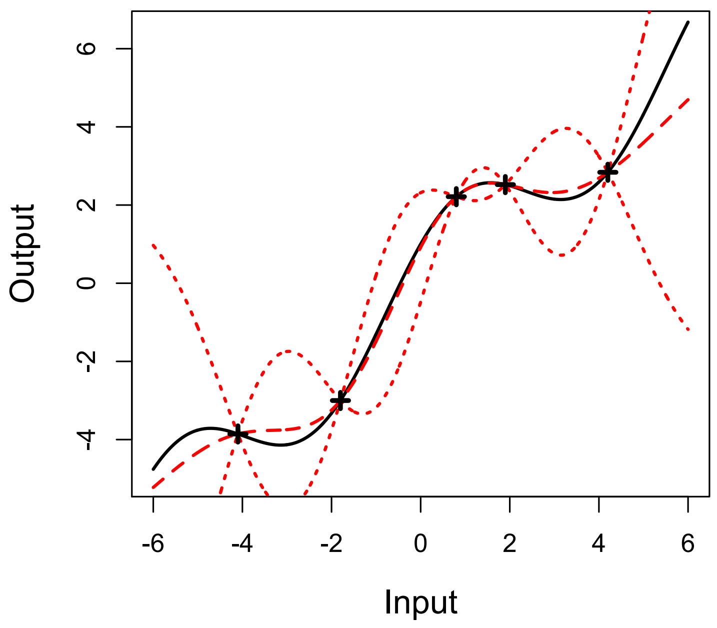


Figure 3.2 repeated. Emulator built with function output only.

is 4.325 and the posterior mean of the emulator without derivatives at $x = 5$ is 3.596 with standard deviation 1.370. The corresponding values from the emulator with derivatives are 4.162 with standard deviation 0.308. If we are particularly interested in the the input region around $x = 5$ therefore then the emulator with derivatives would clearly be preferred. If we could afford more simulator runs though, adding an extra design point in this region would vastly improve both emulators. In high dimensional input space, however, and with computationally very expensive models this isn't always practical.

We can observe from comparing Figures 4.3 and 3.2 that these emulators are producing a similar predictive performance. Which emulator has the lowest uncertainty though, is harder to estimate without formally integrating over the region. In Sections 4.4 and 4.5 we look further into this and attempt to quantify what benefit there may be in employing derivatives in the emulation of complex, computer models.

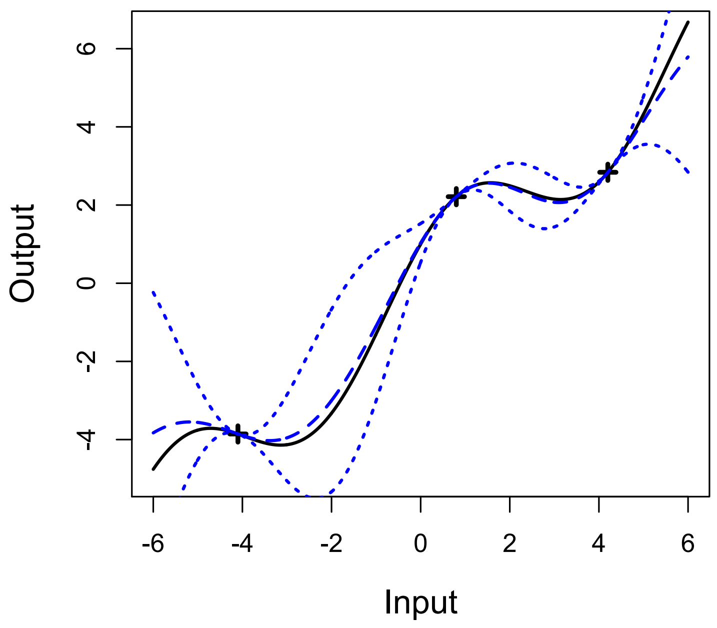


Figure 4.3: Emulator with 3 design points and derivative information.

4.3 Computational cost of obtaining and applying derivative information

It is necessary to consider the computational cost of using derivative information in computer experiments as it must be determined at which point the costs outweigh the benefits. For example, if generating the derivatives of the model increases the computational cost substantially, this extra computing time may be better spent evaluating the model at more input points instead.

In Chapter 2 various techniques for computing derivatives are reviewed, but any cost factor given is model specific. Though it is generally agreed an adjoint provides the most efficient method of obtaining derivatives, there is not a general rule for the additional computational cost required to run an adjoint model. One reason for this is that the computational cost depends greatly on how efficiently the adjoint is written.

In the situation where the derivatives are already available, perhaps as a results of some sensitivity analysis, it is a matter of considering just the additional computational cost of including this information when building the emulator.

The main source of additional cost comes from the inversion of the covariance matrix, $\tilde{A}$, as this matrix is now larger in dimension. $\tilde{A}$ must include the covariances between all partial derivatives and also the covariances between the partial derivatives and the function output. The size of $\tilde{A}$ depends on how many derivatives we include, but if a partial derivative w.r.t each input is included at every location in the design, this results in a matrix of size $n(p+1) \times n(p+1)$, where p is the number of input dimensions. More generally, we denote d to be the total number of derivatives in a design. Comparing $\tilde{A}$ to the prior covariance matrix when no derivative information is included, A, we see that $\tilde{A}$ has therefore increased in dimension by a factor of d. Estimating the smoothing parameters from the training data requires A to be inverted at each step. Therefore, approximately, the cost of including derivative information is increased by a factor of d^3 (Morris *et al.*, 1993). However the computational cost of building the emulator, once training data has been obtained, is relatively small when compared to the computational cost of running the simulator or adjoint to generate the training data.

4.4 Emulating toy models with derivative information

4.4.1 The models

To investigate the value of derivative information in Gaussian process emulation, 5 toy models are chosen with varying levels of smoothness.

1. The first toy model is 1-dimensional and *very* smooth. The simulator is $\eta_1(x) = x^3 + 200$, and we are interested in emulating $\eta_1(.)$ over the input region $[-5, 5]$.

2. Toy model 2 is 1-dimensional and *quite* smooth. Here, $\eta_2(x) = x + \cos(x) + 2\sin(x)$ and we would like to emulate $\eta_2(.)$ over the input region $[-6, 6]$.

3. The simulator for the third model is $\eta_3(x) = \frac{x}{3} + \sin(x)$. This model, also in just 1-dimension, is less smooth and $[-6, 6]$ is the input space we will cover when looking at this model.

4. Toy model 4 is 1-dimensional and very rough. The simulator is given by the equation $5\sin(x^2)+1$ and we are interested in emulating this function over the input region $[1, 10]$.

5. The final simulator is a model of the flow of water through a borehole in m^3/yr,

$$\text{flow rate} = \frac{2\pi T_u(H_u - H_l)}{\ln(\frac{r}{r_w})\left[1 + \frac{2LT_u}{\ln(\frac{r}{r_w})r_w^2 K_w} + \frac{T_u}{T_l}\right]}. \tag{4.17}$$

   This is the model chosen by Morris *et al.* (1993) to demonstrate their methodology for including derivative information in the analysis of computer models. There are eight inputs and these are shown in Table 4.2 along with a range of values that each input takes. However, we keep 5 of the inputs fixed and vary 3, namely r_w, K_w and L. The 5 fixed inputs are set at the lower bound of their range.

Input	Name in model	Description	Range	Units
Input 1	r_w	radius of borehole	0.05 – 0.15	m
Input 2	r	radius of influence	100 – 50,000	m
Input 3	T_u	transmissivity of upper aquifer	63,070 – 115,600	m^2/yr
Input 4	H_u	potentiometric head of upper aquifer	990 – 1,110	m
Input 5	T_l	transmissivity of lower aquifer	63.1 – 116	m^2/yr
Input 6	H_l	potentiometric head of lower aquifer	700 – 820	m
Input 7	L	length of borehole	1,120 – 1,680	m
Input 8	K_w	hydraulic conductivity of borehole	1,500 – 15,000	m/yr

Table 4.2: Inputs for borehole model

All of the 1-dimensional toy models over their respective input regions are shown in Figure 4.4 and are emulated employing the methodology of Chapter 3. Where derivatives are included, the additional information is implemented as in the methodology of Section 4.2. We choose as a linear form for the prior mean in all emulation here and generate a maximin latin hypercube to choose where to evaluate the simulator runs.

4.4.2 Method of comparison

The emulators will be compared by examining their predictive performances. As the simulators here are not complex models, it is possible to run them at all points the emulator is predicting the output at. An average prediction error is then determined by looking at the difference between the value the emulator predicts, and the true value at that point as given by the simulator. The following expression will be used:

$$\text{Prediction Error} = \frac{1}{n'}\sum \frac{(|\text{True Value} - \text{Predicted Value}|)}{|\text{True Value}|}, \qquad (4.18)$$

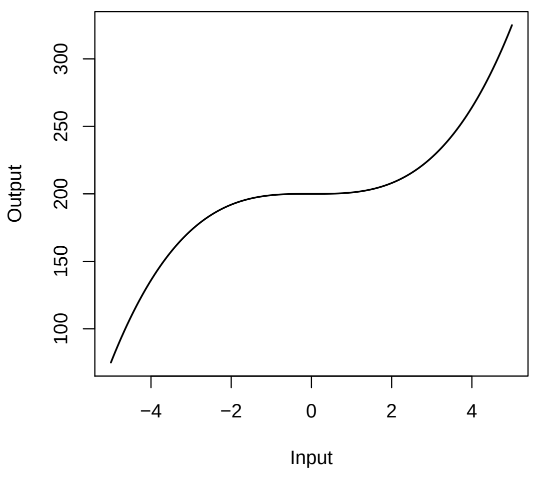
where n' is the number of points the emulators are predicting the function output at. We choose to divide by the true value so as to normalise the error and thus enable comparison of emulators of different outputs, across different ranges. Although this does also have the effect of increasing the error for areas of the input space which result in a small output value, the resulting error diagnostic still provides a good indication of the performance of the various emulators.

It is difficult to quantify exactly the computational cost of obtaining derivatives. This is discussed in Section 4.3 but no cost factor is given. Therefore, in order to investigate the value of derivative information in toy models, we will analyse how many extra runs are required for an emulator without derivatives, to give a comparable prediction error to that of an emulator built with derivative information.

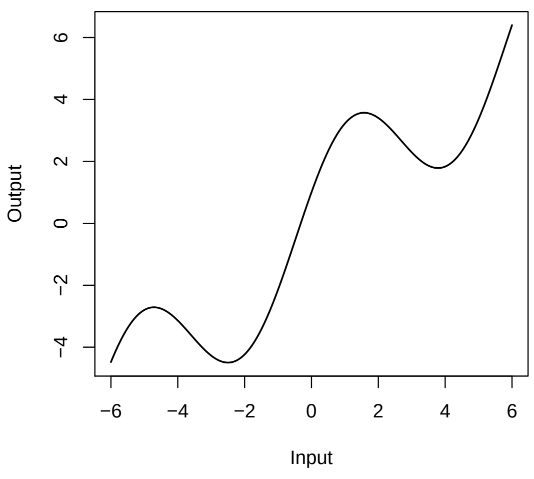
We will also investigate how the uncertainty differs across the emulators. This will be done by looking at the average standard deviation of the emulators over the specified input region. Specifically, we estimate this by the following statement

$$\text{Mean Standard Deviation} = \frac{1}{n'}\sum \hat{\sigma}\sqrt{\mathrm{c}^{**}(x_i, x_i)}, \qquad (4.19)$$

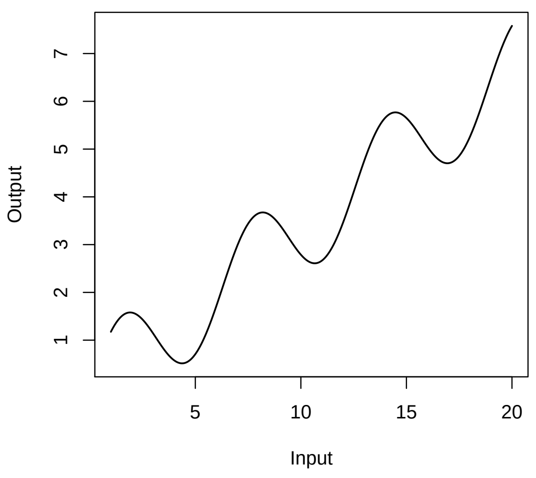
where $i \in (1, \ldots, n')$.



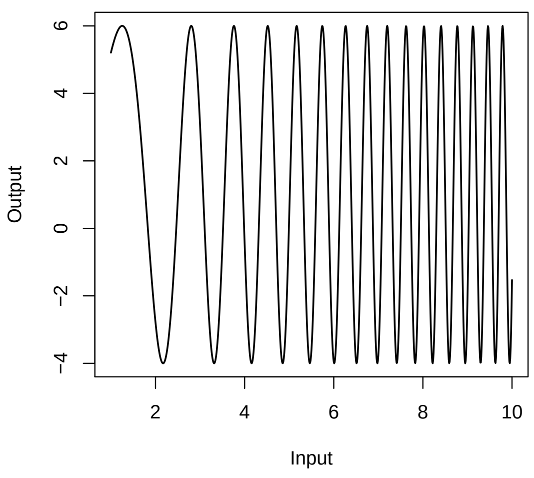
(a) Toy Model 1



(b) Toy Model 2



(c) Toy Model 3



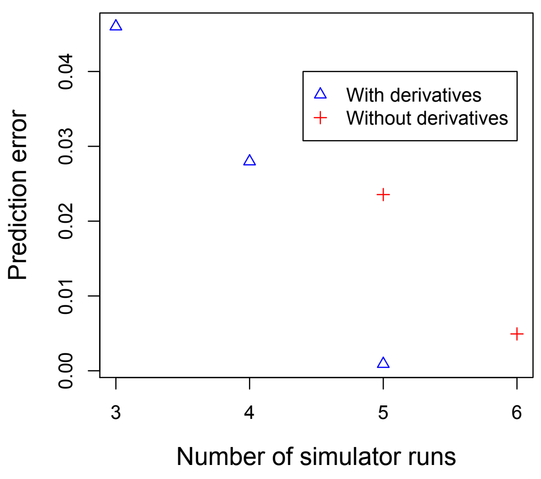
(d) Toy Model 4

Figure 4.4: Toy Models 1 - 4

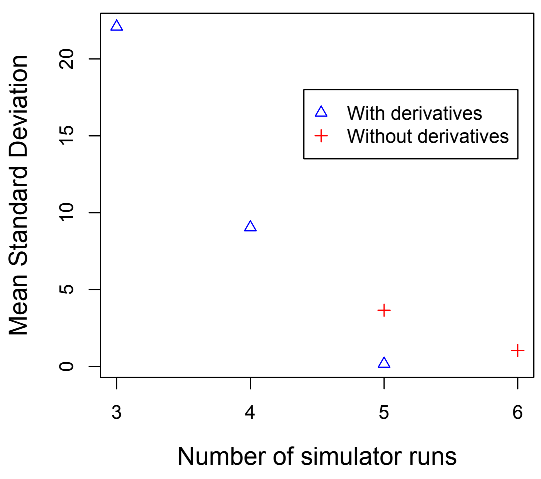
4.4.3 Results

To compare the differences in the predictive performance and in the uncertainty between emulators, Figures 4.5 - 4.7 have been produced. Part (a) of these figures show the prediction error, as calculated by (4.18) for emulators built with an increasing number of simulator runs. The measure of uncertainty across emulators built with a growing number of design points, as evaluated by (4.19), are shown in part (b). Throughout Figures 4.5 - 4.7, blue triangles represent the performance of emulators built with derivative information while red crosses display the equivalent for emulators built only with function output.

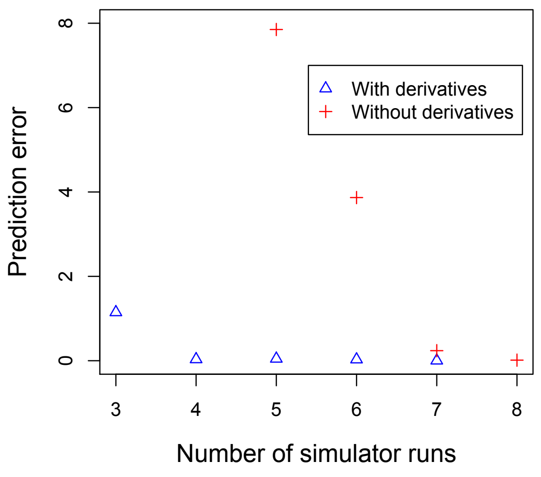
Figures 4.5 - 4.7 clearly show that for all the toy models tested here, the posterior mean of emulators built with the additional information of derivatives provide a closer approximation to the relevant simulator. The mean standard deviation is also reduced for that of emulators with simulator derivatives. The prediction error and the uncertainty for the rough simulator, as shown in Figures 4.7a and 4.7b, follow a less strict trend than the other toy models. This may be due to the difficulty in estimating the smoothness parameter, θ, for this model. We therefore fix this parameter at an appropriate value and rebuild the emulators, with and without derivatives. Due to the roughness of toy model 4 it is necessary to select a very high value for θ and the results from the previous emulation show 2000 is approximately suitable. Figures 4.7c and 4.7d shows the resulting comparisons of the prediction error and uncertainty with $\theta = 2000$. The derivative information appears to have no effect in the emulators built with a small number of simulator runs. As the number of design points is increased beyond 19 the prediction error for the emulators built with derivative information decreases much more quickly than for the corresponding emulators built only on function output. A similar pattern is observed in Figure 4.7d which shows how the uncertainty is affected.



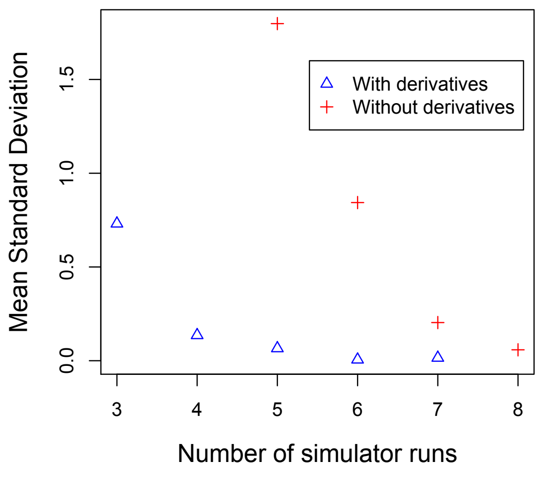
(a) Prediction error for Toy Model 1



(b) Uncertainty for Toy Model 1

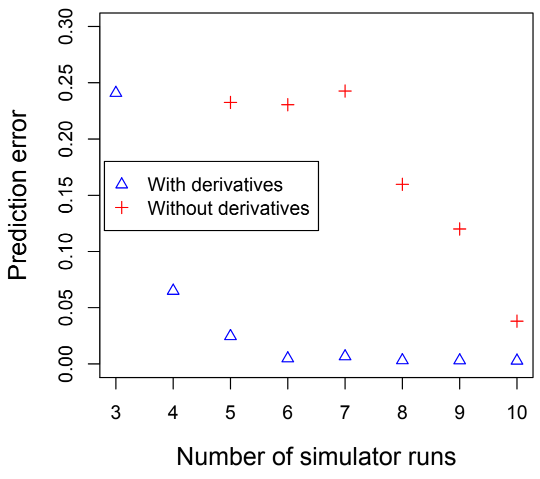


(c) Prediction error for Toy Model 2

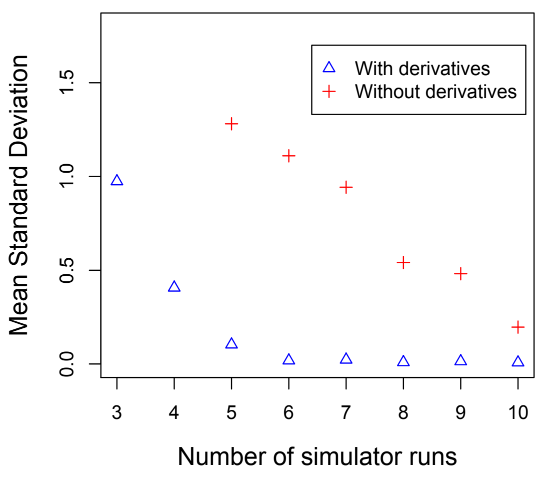


(d) Uncertainty for Toy Model 2

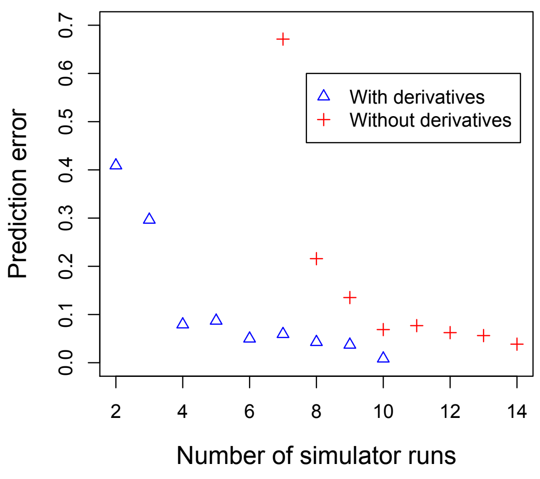
Figure 4.5: Performance of emulators built with varying numbers of runs of toy models 1 and 2



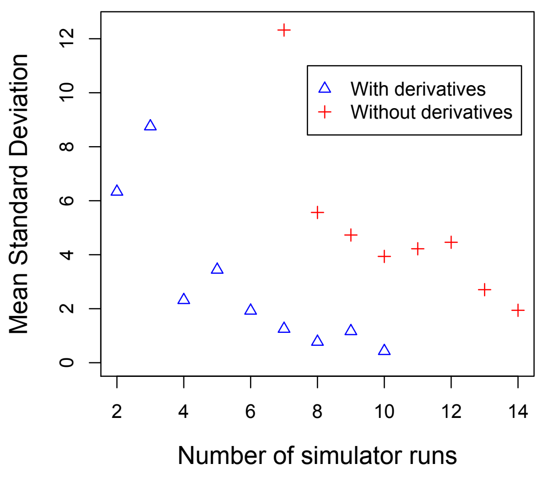
(a) Prediction error for Toy Model 3



(b) Uncertainty for Toy Model 3

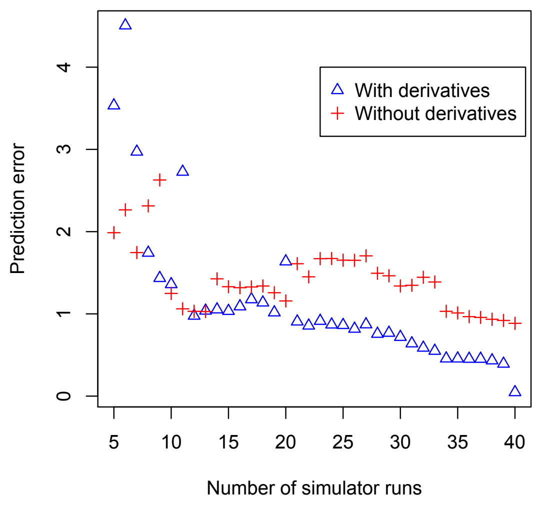


(c) Prediction error for Toy Model 5

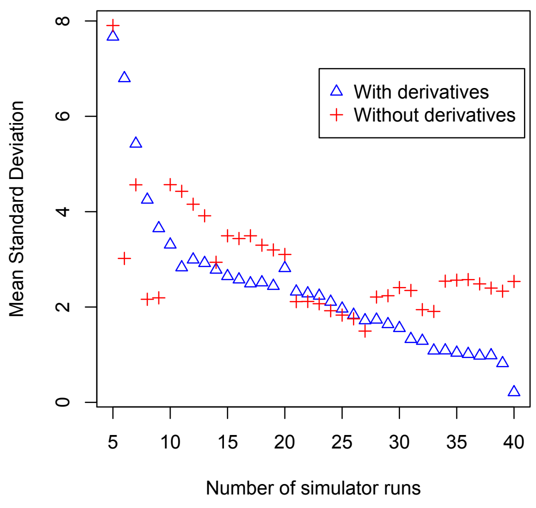


(d) Uncertainty for Toy Model 5

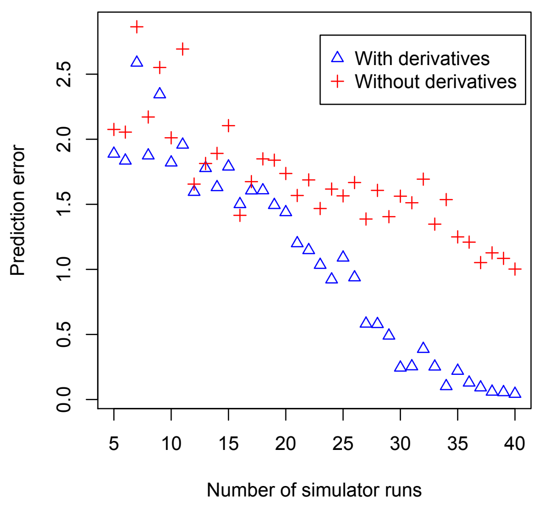
Figure 4.6: Performance of emulators built with varying numbers of runs of toy models 3 and 5



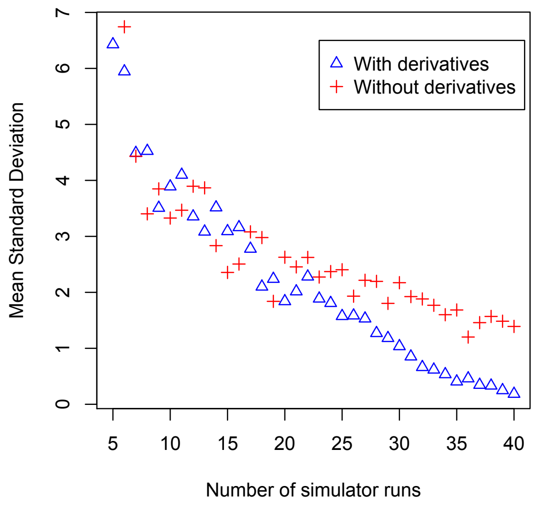
(a) Prediction error for Toy Model 4



(b) Uncertainty for Toy Model 4



(c) Prediction error for Toy Model 4 with θ fixed at 2000

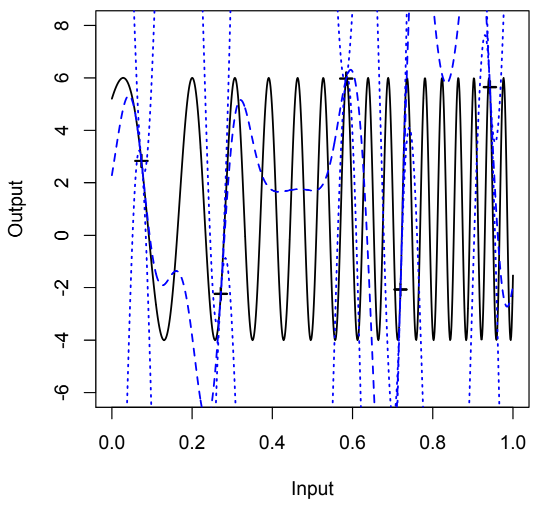


(d) Uncertainty for Toy Model 4 with θ fixed at 2000

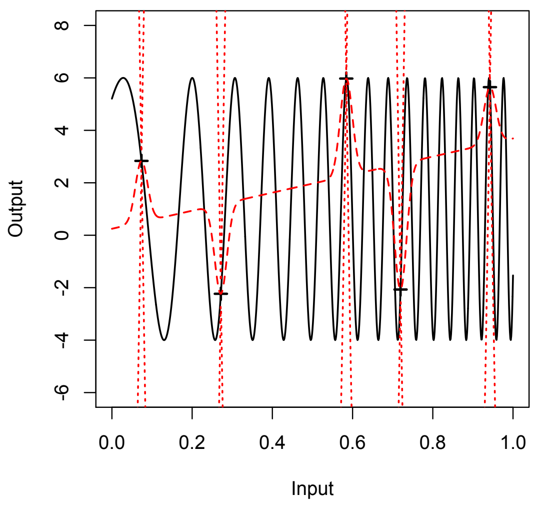
Figure 4.7: Performance of emulators built with varying numbers of runs of toy model 4

Figures 4.5 - 4.7, while providing a good indication of the effect of derivatives only provide a scalar to assess how valid any one emulator might be. To illustrate what a particular prediction error signifies, we plot the performance of 4 of the emulators of toy model 4 in Figure 4.8. Emulators built with and without derivatives when we have only 5 input points, are shown in Figures 4.8a and 4.8b respectively. We see for the emulator with derivatives (Figure 4.8a) extended good prediction with low uncertainty at and around each design point, the emulator benefiting from the derivative information at these points. As soon as we as move further away from the design points though, the emulator completely fails to capture the behaviour of the simulator and this is summarised by a prediction error of 3.534. The effect of the derivatives is very clear when we compare with an emulator built only with model response, shown in Figure 4.8b. For points close to design points the emulator does not predict well, deviating straight back to the linear part of the posterior mean due to a very high smoothness parameter. Clearly the emulator does not capture the behaviour of the function, but due to the nature of the simulator the prediction error is slightly misleading in its relatively small value of 1.987. The performance of emulators built with 40 points, shown in Figures 4.8c and 4.8d, is much better. The emulator with derivatives now captures the behaviour of the simulator across the whole input space and a prediction error of 0.04655 is achieved. The emulator without derivatives is also much improved, with only a small number of areas where the behaviour of the simulator is not captured. The prediction error for this emulator is 0.8849.

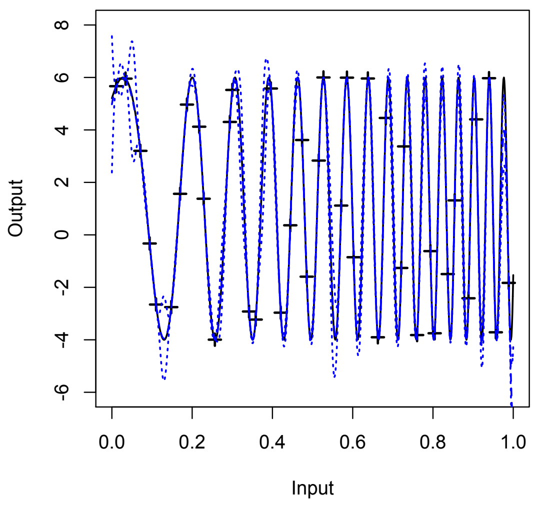
As discussed in Section 4.4.2, however, the computational cost of obtaining derivative information must be taken into consideration. Table 4.3 shows how many extra runs are required for the emulators without derivatives, to achieve similar prediction errors to the emulators with derivative information and Table 4.4 displays the equivalent information for the mean standard deviation. We can see from Tables 4.3 and 4.4, for toy models 2, 3 and 4, an emulator without derivative information requires approximately twice as many simulator runs as an emulator with derivative information to achieve similar accuracy. Though it should be noted that in the case of toy model 4, if less than approximately 20 simulator runs are performed then the emulators perform similarly, regardless of derivative information.



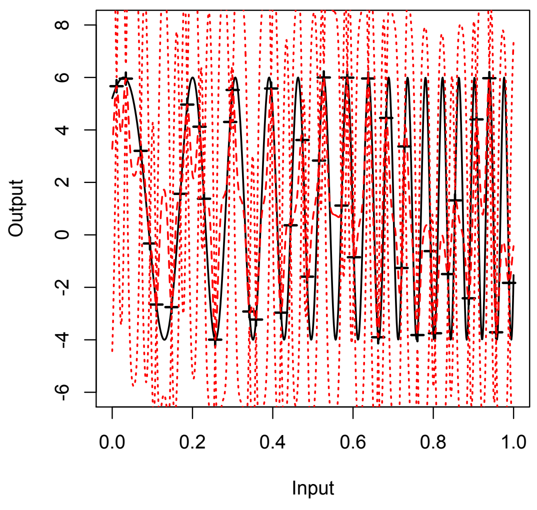
(a) Emulator with 5 input points and derivatives included. The prediction error is 3.534 with mean standard deviation 7.670.



(b) Emulator with 5 input points and no derivatives included. The prediction error is 1.987 with mean standard deviation 7.905.



(c) Emulator with 40 input points and derivatives included. The prediction error is 0.04655 with mean standard deviation 0.2117.



(d) Emulator with 40 input points and no derivatives included. The prediction error is 0.8849 with mean standard deviation 2.536.

Figure 4.8: Illustration of the performances of emulators with low and high prediction errors for toy model 4.

Model	Derivatives	Prediction Error	Design Points
1	Yes	0.0280	4
1	No	0.0236	5
1	Yes	0.0009	5
1	No	0.0049	6
2	Yes	0.0336	4
2	No	0.0135	8
3	Yes	0.0247	5
3	No	0.0381	10
4	Yes	0.9060	21
4	No	0.8850	40
4 (fixed b)	Yes	1.0339	23
4 (fixed b)	No	1.0024	40
5	Yes	0.0372	9
5	No	0.0385	14

Table 4.3: Comparison of predictive performance of different emulators

Model	Derivatives	Mean Standard Deviation	Design Points
1	Yes	0.1766	5
1	No	1.046	6
2	Yes	0.0659	5
2	No	0.0582	8
3	Yes	0.1033	5
3	No	0.1966	10
4	Yes	2.515	18
4	No	2.536	40
4 (fixed b)	Yes	1.271	28
4 (fixed b)	No	1.391	40
5	Yes	1.924	6
5	No	1.941	14

Table 4.4: Comparison of the uncertainty of different emulators

4.5 Emulation of radiation transport model with derivative information

The radiation transport model, as described in Section 1.3.2 of Chapter 1, calculates the measured radiation signature of a gamma-ray-emitting and neutron-multiplying cylinder. There are 5 inputs which we vary and we choose to investigate just one of the 5 outputs of this model: the neutron multiplication. This output is chosen as there may be correlations between the 4 gamma-ray outputs and as such these outputs are more appropriately investigated in Chapter 6.

A 100 point Latin hypercube is generated across the 5 input dimensions and the points scaled linearly from [0,1] to appropriate values using the ranges supplied in Table 1.2. We require that Input 5 > Input 4 and so ensure that the values for Inputs 4 and 5 lie in the upper triangle of a 2-dimensional scatterplot between these 2 inputs.

4.5.1 Exploratory data analysis

The neutron multiplication factor can be represented as either k_{eff} or its inverse and the radiation transport model produces this output in the inverse form. We illustrate the data, run at the input configurations from the LHS in Figure 4.9.

Since the neutron multiplication factor can be represented as k_{eff} and due to the spread of the data shown in Figure 4.9 we apply the inverse transformation on output 5, resulting therefore in the neutron multiplication factor as k_{eff}. The transformed data is shown in Figure 4.10 and provides good justification to emulate the transformed response rather than the native model response. The partial derivatives are transformed according to the chain rule: let $z_5 = \frac{1}{y_5}$, then the required derivatives of the transformed output, z_5, w.r.t to the 5 inputs, $\mathbf{x}$, are given by:

$$\begin{aligned}\frac{\partial z_5}{\partial \mathbf{x}} &= \frac{\partial z_5}{\partial y_5} \times \frac{\partial y_5}{\partial \mathbf{x}} \\ &= -\frac{1}{y_5^2} \times \frac{\partial y_5}{\partial \mathbf{x}}. \qquad (4.20)\end{aligned}$$

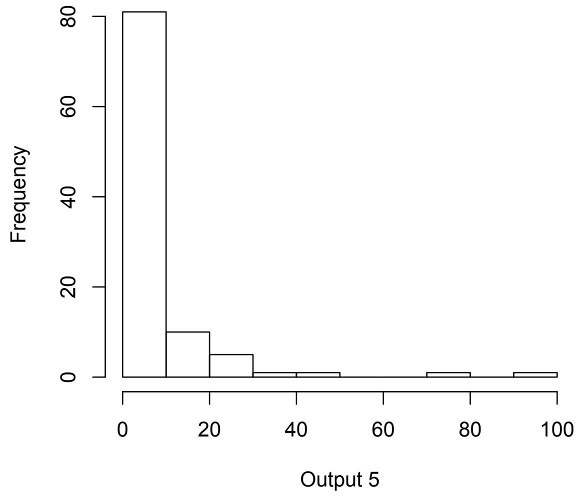


Figure 4.9: Output 5 (=1/ k_{eff}) from radiation transport model run at 100 point LHS

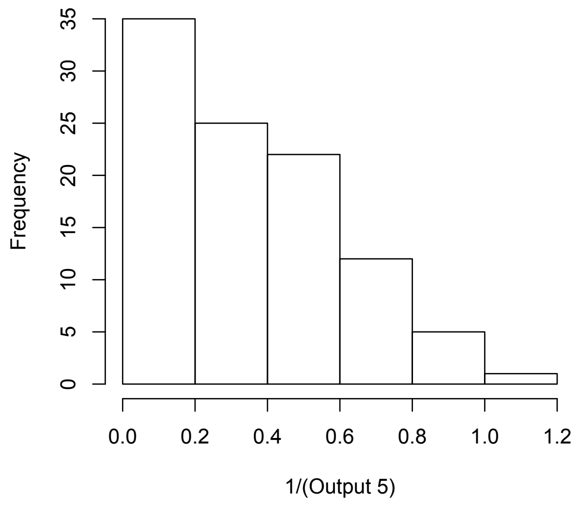
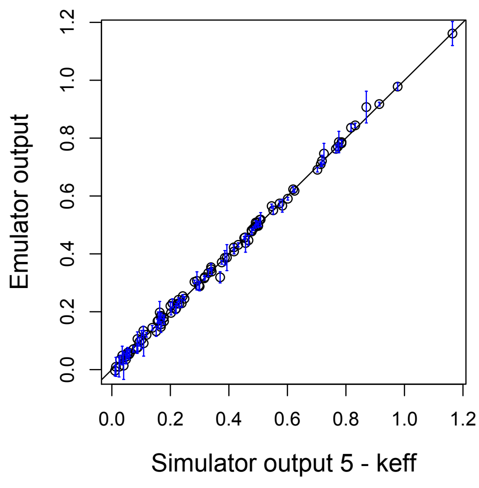


Figure 4.10: 1/Output 5 (= k_{eff}) from radiation transport model run at 100 point LHS

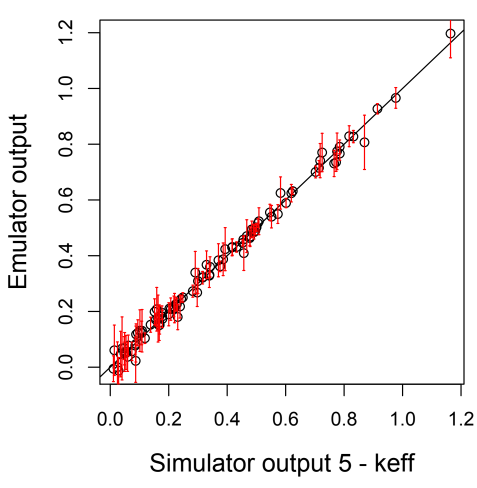
4.5.2 Initial analysis

We want to build emulators, with and without the derivative information, to assess the value of the derivatives in this example. We require a covariance function which is twice differentiable and choose $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta\, (\mathbf{x}_i - \mathbf{x}_j)\right\}$. A linear form for the prior mean is selected and the smoothness parameters, $\theta\{i\}$ are estimated through maximum likelihood.

To begin, we build emulators with the information from all 100 runs. As we do not have a separate diagnostic experiment we adopt a *leave one out* method to assess the emulators: the information from the first run is left out of the training data and an emulator built with the remaining 99 runs. We then use this emulator to predict the observation which was left out and this is repeated for all 100 observations. Comparisons between the values predicted by the emulators and the true values are shown in Figure 4.11.



(a) With derivative information



(b) No derivative information

Figure 4.11: Leave one out results for the prediction of k_{eff}

We can see from Figure 4.11 that both emulators are performing well. The emulator with derivatives would appear to be slightly superior, with better predictions and less uncertainty, than the emulator without derivatives. The adjoint of the radiation transport model requires approximately twice the computational time to run than the standard simulator, (Jeffery Favorite, pers. comm.). The question therefore is whether or not the emulator with derivatives in Figure 4.11a is performing twice as well as the emulator without derivatives in Figure 4.11b.

4.5.3 Comparison of emulators with increasing data

To attempt to answer the question posed at the end of Section 4.5.2, we first investigate the performances of the emulators with a reduced amount of the training data. We build emulators, similarly to those described in Section 4.5.2 but now with only 60 training runs. No further runs of the model are available and so we take the first 60 runs as indexed in our LHS as training data, and the remaining 40 runs are used for validation. The performances of the subsequent emulators are shown in Figure 4.12 where the emulators built with 60 training runs have been used to predict the response at the remaining 40 input sites. We see in Figure 4.12 that both emulators appear to be performing quite well.

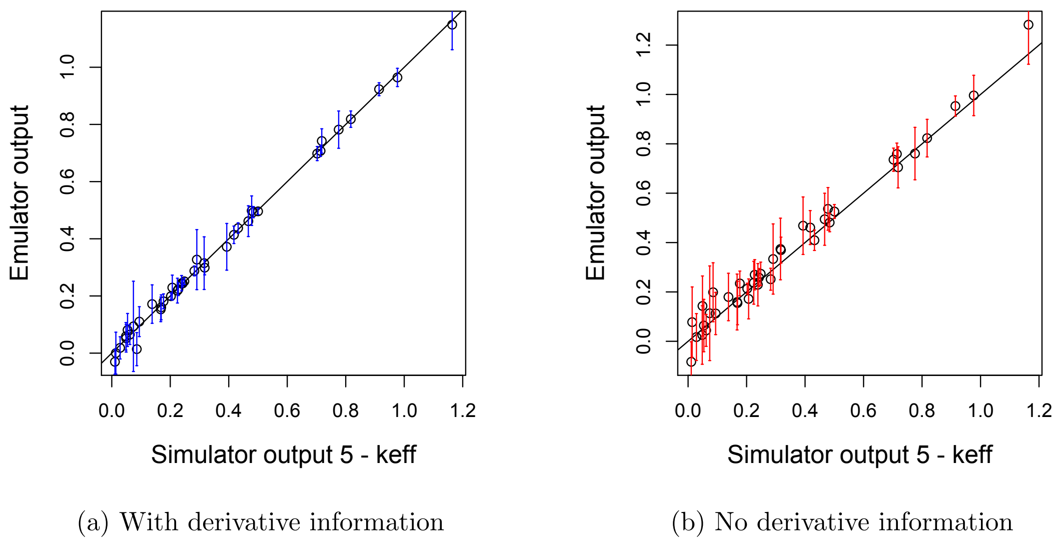
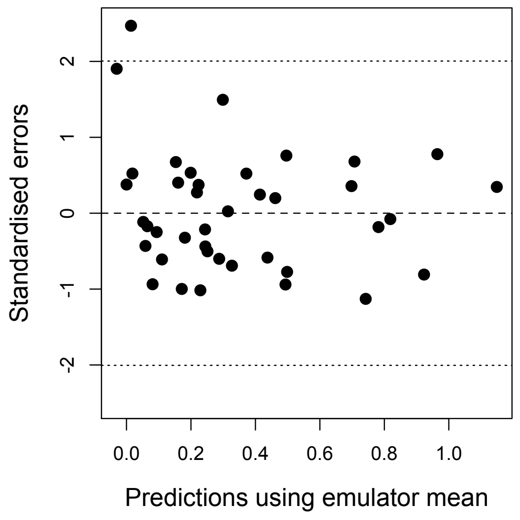
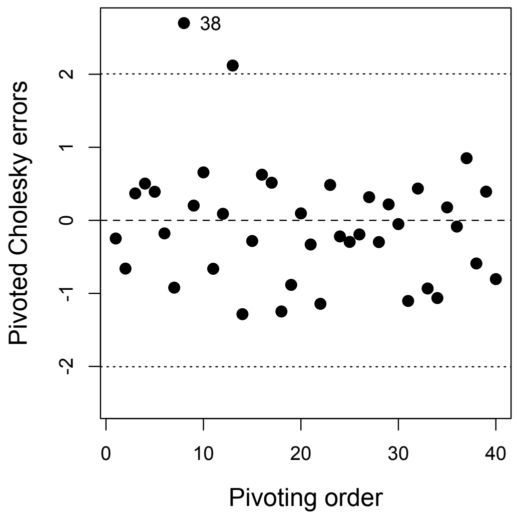


Figure 4.12: Diagnostic plots of the emulators built with $n = 60$ training runs and predicting at $n' = 40$ validation points.

We adopt further diagnostic measures as given by Bastos and O'Hagan (2009) and calculate the Mahalanobis distance and error plots for each emulator. These diagnostics are discussed further in Section 3.3. Figure 4.13 shows the error plots for the emulator built with $n = 60$ responses and the corresponding 300 derivatives. Standardised errors are plotted against the posterior mean of the emulator in Figure 4.13a and in Figure 4.13b we see the pivoted Cholesky errors. Most of the errors lie between -2 and 2 and show no clear pattern of large or very small values. The observed Mahalanobis distance for this emulator with derivatives is 26.7, which should be compared to the theoretical mean of $n' = 40$ with standard deviation 12.13. The observed Mahalanobis distance is a little low but is within 2 standard deviations of the theoretical mean and as the error plots show mostly no conflict between emulator and simulator, with only one outlier, we conclude that this emulator is valid.



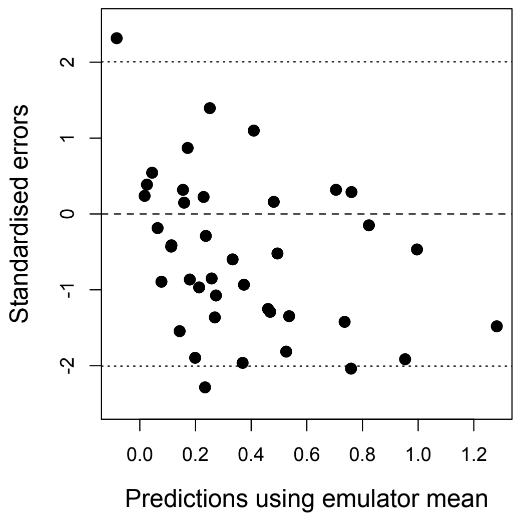
(a)



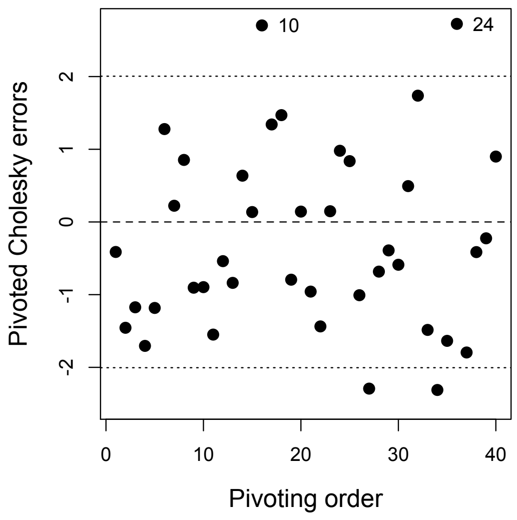
(b)

Figure 4.13: Further diagnostic plots of the emulator, built with derivative information, with $n = 60$ training runs and predicting at $n' = 40$ validation points.

The corresponding emulator built with just the simulator response at the $n = 60$ input sites produces similar validation results. The standardised errors for this emulator can be seen in Figure 4.14a and the pivoted Cholesky errors in Figure 4.14b. Again we see most of the errors fall in the required region of $[-2, 2]$ with just a couple of outliers. The observed Mahalanobis distance is 64.8, which is little high but overall there is not too much evidence of conflict between the simulator and the emulator to warrant further runs.



(a)



(b)

Figure 4.14: Further diagnostic plots of the emulator, built without derivative information, with $n = 60$ training runs and predicting at $n' = 40$ validation points.

Now we have established that emulators built with $n = 60$ runs provide satisfactory results we can continue to hold back 40 runs for validation and comparison purposes.

We now wish to determine how the derivatives impact the performance of the emulators. We can do this by reducing the data further, and then investigating how the performance of the emulators vary when we add further runs with just the response, and when we add further runs with the derivative information in addition to the response. We do this for an increasing number of runs up to $n = 60$ and use the $n' = 40$ runs that were held back to calculate the prediction error and a measure of uncertainty for each emulator. We choose a different method for calculating the prediction error to the error diagnostic applied in Section 4.4. In Section 4.4 we adopted equation (4.18) to calculate the prediction error which is repeated here:

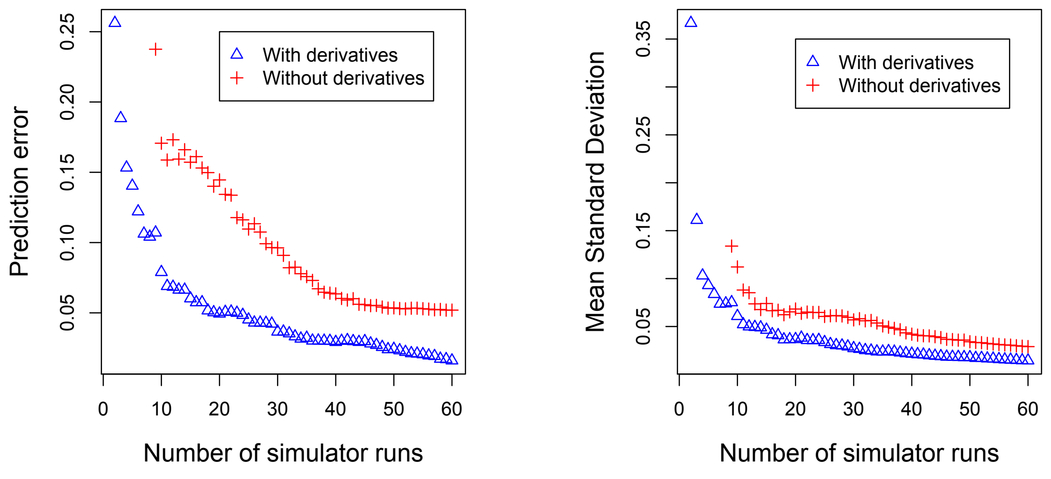
$$\text{Prediction Error} = \frac{1}{n'} \sum \frac{(|\text{True Value} - \text{Predicted Value}|)}{|\text{True Value}|}.$$

We chose to divide by the true value so as to normalise the error and thus enable comparison of emulators of different outputs, across different ranges. This does, however, also have the effect of increasing the error for areas of the input space which result in a small output value. For this reason we do not adopt equation (4.18) but use the root mean squared error as follows:

$$\begin{aligned} \text{Prediction error} &= \sqrt{\frac{1}{n'}\sum(m^{**}(\mathbf{x}'_i) - \eta(\mathbf{x}'_i))^2}, \\ \text{Mean standard deviation} &= \frac{1}{n'}\sum \hat{\sigma}\sqrt{\mathrm{c}^{**}(x_i, x_i)}, \end{aligned} \tag{4.21}$$

where $i \in \{61, \ldots, 100\}$ and $n' = 40$. The measure of uncertainty, is calculated as in Section 4.4. Now we want to build emulators with increasing amounts of training data; starting with $n = 2$, in the case where we include derivatives, and with $n = 9$, in the case where we do not include derivatives, up to $n = 60$ for both cases. Since we have truncated a 100 point LHS to provide 60 input sites to select from, any resulting design is unlikely to be optimal. To account for this we randomly permute the order of the 60 training runs. We then take the first 2 runs as indexed by this permutation and build an emulator with derivatives, calculating the prediction error and the measure of uncertainty for this emulator. Next, we take the first 3 runs as indexed by this permutation and build another emulator with derivatives. We continue in this way, building emulators without derivatives in addition from $n = 9$ onwards, until $n = 60$. We then randomly permute the order of the 60 training runs again, to generate a second permutation and repeat the process. This is repeated 20 times and we then calculate the mean prediction error and measure of uncertainty across the 20 permutations for each value of n. Clearly for $n = 60$ there is no need to permute the order as we only have the first 60 runs as indexed by the original LHS to choose from and the order they are in the training data is not important.

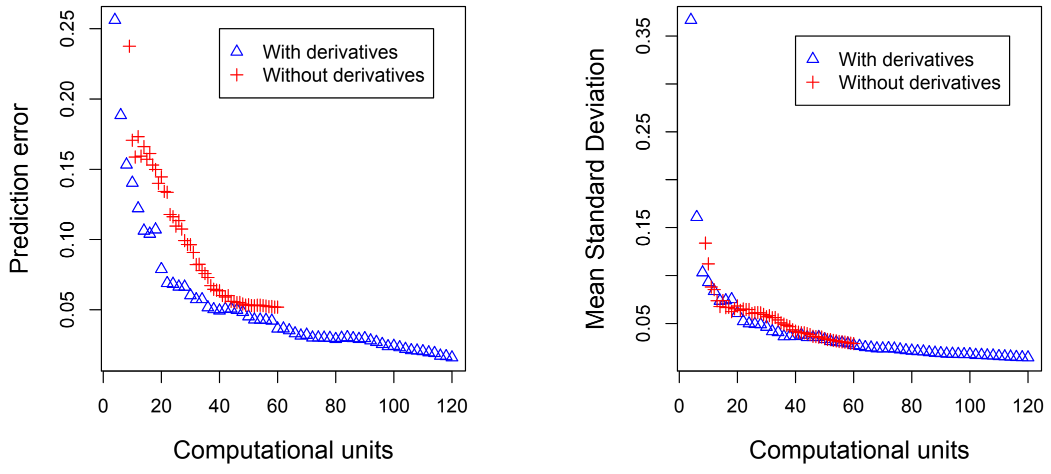
The results of the experiment described above are shown in Figure 4.15. Both the prediction error and the measure of uncertainty for the emulators with derivatives are consistently lower for all n than the emulators without derivatives. The prediction error decreases as we add to the training data for both emulators, as expected. The trend of the prediction error for the emulators with derivatives is still decreasing, even at $n = 60$. In comparison to this, the prediction error of the emulators without derivatives appear to have reached a plateau by $n = 45$. It may be the case, therefore, that by adding even more runs to the training data an emulator without derivatives may still not be able to produce a similar prediction error to that of an emulator with derivatives. The emulators built with 60 input points, with and without derivatives, have prediction errors of 0.019 and 0.047 respectively. The diagnostics of these emulators were shown earlier in this section, in Figures 4.12 to 4.14, and therefore provide an illustration of what prediction errors of these values signify.



(a) Comparison of prediction error. (b) Comparison of standard deviation.

Figure 4.15: Performance of emulators, with and without derivatives, built with increasing numbers of simulator or adjoint runs.

Figure 4.15, while providing some insight into how effective the derivatives are, does not take into consideration the extra computational expense required to generate the derivatives. The adjoint of the radiation transport model requires approximately twice the amount of computing time to run than the standard model. We therefore now compare the emulators, not by number of model runs but by computational time taken to build the emulator. For simplicity we assume each simulator run (standard version of the radiation transport model) requires one computational unit, and therefore each adjoint run requires 2 computational units. The adjoint model returns the model response and the partial derivatives w.r.t all 5 inputs. The cost to calculate model response alone at one input configuration is therefore 1 unit and the cost to generate model response and the partial derivatives w.r.t the 5 inputs at the specified input configuration is 2 units. We do not account for any extra computational time to build the emulator outside of the simulator or adjoint run time. As discussed in Section 4.3, including derivatives adds a small amount of computational expense in building the emulator, even if the derivatives themselves are already available. This is due mainly to the inversion of the covariance matrix, $\tilde{A}$, which is now bigger in size. This extra computational cost, though, is expected to be negligible in comparison to the adjoint model run time, and as such is ignored in this experiment. The results, accounting for the cost of the derivatives, are shown in Figure 4.16.



(a) Comparison of prediction error. (b) Comparison of standard deviation.

Figure 4.16: Performance of emulators, with and without derivatives, built with an increasing amount of computational expense.

We can now directly compare, in Figure 4.16, emulators built with and without derivative information. The emulators with derivatives perform consistently better than the emulators without derivatives, achieving a lower prediction error regardless of how many computational units are spent. As more runs are added to the training data, the difference in prediction error between emulators is reduced. The minimum difference is observed at 46 computational units, with prediction errors of 0.049975 and 0.051101 for emulators with and without derivatives respectively. The mean standard deviation is very similar for both emulators. This implies that for the k_{eff} output of the radiation transport model it is more efficient to run the adjoint of the model and include derivative information when building an emulator.

4.6 Applications of derivative information in computer models

In Chapter 2 we outlined some of the analyses which can benefit from derivative information. In addition to these we have shown in Section 4.2 how derivatives can be included in a Gaussian process emulator and further investigation into the benefit of emulators with derivatives is given in Sections 4.4 and 4.5.

Killeya and Goldstein (2007) utilise derivative information in the analysis of compartmental models. A Bayes Linear approach is adopted rather than implementing a fully Bayesian analysis. A Bayes-Linear emulator does not provide a full probability distribution for the outputs of a complex model, instead, just means and variances are given. Details of the Bayes Linear approach to emulating complex models is given by Craig *et al.* (1997). The methodology of Killeya and Goldstein (2007) is demonstrated by a compartmental model of plankton cycles. The first-order input derivatives are calculated analytically by hand and are employed in a sensitivity analysis. The sensitivity analysis is performed by plotting derivatives and considering the sample mean and variance. Input variables whose derivatives have a mean around zero and a small variance may be regarded as inactive. If an input variable has derivatives with a high variance, however, it may be termed an *active variable*. In this way a subset of the input variables are selected and this enables the input dimensionality to be reduced. Having determined which of the inputs are active variables they continue to investigate how the inclusion of derivatives affects the emulator uncertainty. Specifically, 11^4 input points are chosen at which to evaluate the standard deviation of the emulators; the mean value of the standard deviation, taken across this space, and the maximum value are recorded. The procedure is repeated for emulators, built with and without derivative information, with varying numbers of simulator runs. The mean standard deviation produced from 25 runs of the emulator without derivative information, is comparable to the mean standard deviation produced from 6 runs of the emulator with derivatives for the first model output and 8 runs for the second model output. The expected increase in the cost of generating the derivatives, as reported by the authors, is by a factor of 1.8. Therefore, we can consider

25 runs of the emulator without derivatives to be similar in computational cost to approximately 14 runs of the emulator with derivatives. This shows that to achieve comparable uncertainty, in this example, it is computationally cheaper to include derivative information in the building of an emulator. Killeya and Goldstein (2007) also evaluate the additional reduction in mean variance caused by including derivatives, for a given set of runs. In their example for the first output variable, the maximum additional reduction in variance is 44.5% and for the second, 31.2%. Thus it is shown that uncertainty about the model is reduced substantially, if derivative information is included when building an emulator. They do not compare, however, the prediction error in the emulator means.

The benefit of observing derivatives in modelling nonlinear dynamic systems is discussed in Leith *et al.* (2002), Solak *et al.* (2003) and Azman and Kocijan (2005). Their work centres around combining linear local models with Gaussian process models. They use derivative observations as a means to decrease the number of data points in areas close to the equilibrium of the system. It is often the case in nonlinear dynamic systems that more experimental data is available at points close to where the system is in equilibrium, than at points further away from this state. However, the global dynamics of the system are not fully explained by just the data at points around equilibrium; information about the dynamics further away from equilibrium is also required. As explained by Azman and Kocijan (2005), derivative observations around an equilibrium point can be thought of as observations of a local linear model at this point. Therefore from the plentiful data around equilibrium a linear model can be fitted and the coefficients of the linear model are the partial derivatives of the function. A Gaussian process model is then built with the sparse data away from equilibrium and the derivatives around equilibrium. Fitting a Gaussian process model using derivatives to learn about the function at equilibrium rather than the output reduces the number of data points and hence computational expense.

4.7 Conclusions

We know from Chapter 3 that if we have a complex model or simulator which is computationally too expensive to run at every input configuration required, then a solution to this may be to build a Gaussian process emulator. Having built an emulator, using only a small number of simulator runs, the emulator can then be used as a surrogate to the simulator providing an efficient estimate of the simulator output at an unknown input configuration with an associated level of uncertainty. In this chapter we have investigated whether further efficiency in a Gaussian process emulator can be achieved if we include derivative information when building the emulator.

We have shown that in most of the 1-dimensional toy models of Section 4.4, to obtain similar levels of prediction error and uncertainty, an emulator without derivatives requires approximately twice as many model runs as the emulator with derivative information. This suggests that if the computational expense of running the simulator and generating the derivatives is less than twice that of running the model alone, then derivatives provide a more efficient use of computing time. With the exclusion of the very smooth model (toy model 1), the level of smoothness of the simulator didn't appear to have any effect on the value of the derivative information. The very smooth model, however, showed that an emulator without derivatives could achieve a similar level of prediction error with only one more simulator run. The 3-dimensional example (toy model 5), which involves modelling the flow of water through a borehole, provided fairly similar results to those of the 1-dimensional models. The emulator with derivatives required approximately two thirds as many runs as the emulator without derivatives to produce a similar prediction error. Whereas, to obtain similar levels of uncertainty, the emulator with derivatives required just under half as many simulator runs as the emulator without derivatives.

While valuable indicators, toy models are fabricated examples and their usefulness depends on their similarity in behaviour to real complex models. We therefore include a detailed study of the radiation transport model in Section 4.5, which is a complex model run and analysed by Jeffery Favorite and colleagues at Los Alamos National Laboratory. The comparison of emulators of this model built with and without derivatives showed that with the same computational expense spent, the emulators with derivatives all performed better than the emulators without derivatives. To compare emulator performance we examined their prediction error and measure of uncertainty. With less computational expense available, the derivatives appear to have a large effect, achieving considerably lower prediction errors than the emulators without derivatives. The difference in prediction error between the emulators reduces as the computational expense increases though and while emulators with derivatives are still achieving lower prediction errors, the difference is marginal. This may suggest, therefore, that if an adjoint model is available we could produce a more efficient emulator with adjoint runs rather than simulator runs. If an adjoint model does not yet exist though, the added time and expense required to build a working adjoint is non-negligible and therefore it is unlikely that derivative information would then provide efficiency. It should also be noted though that the prediction error of the emulators with derivatives continue to decrease as we increase the computational cost further; while it would appear that the prediction error of the emulators without derivatives have reached a plateau. If more data were available it would be interesting to investigate whether the prediction error of the emulators without derivatives remains in a plateau or decreases in line with the emulators with derivatives.

Chapter 5

Emulating model derivatives

5.1 Introduction

Complex models are an important tool for studying a wide range of systems and as in previous chapters, we refer to a complex model, written as a computer code, as a simulator. As discussed in Chapter 1, derivative information about the simulator output with respect to the model inputs is potentially useful to many model users, for example in the application of sensitivity analysis or data assimilation. One way of generating such derivatives, discussed in Chapter 2, is by coding an adjoint model which, despite automatic differentiation software, remains a complex task and the model when written is computationally expensive to execute. There may be situations when model users are unwilling, or perhaps unable, to allocate the initial time and resources required to code the adjoint of a complex model. For example without access to the source code, an adjoint model cannot be written. In the situation that the adjoint model already exists, the additional computational expense required to run this model in place of the standard version is completely non-negligible.

In this chapter we suggest an alternative method for generating partial derivatives of complex model output, with respect to model inputs: we propose the use of a Gaussian process emulator. As with emulators of function output, we run the simulator at a small number of input points to generate the training data. An emulator of model derivatives can then be built, conditional on this data. Although derivative information is not necessary, if it is available at some or all of the points in the training data, it can be included and the resulting emulator of model derivatives is expected to improve. We detail the methodology of this approach in Section 5.2; firstly the situation where we only have function output is presented and then in Section 5.2.2 we give the general case for emulating first derivatives and model response, with or without derivative information. We then proceed by demonstrating the approach in Section 5.3 with firstly a one-dimensional toy model, secondly the 8-dimensional borehole model used by Morris *et al.* (1993) and finally with the intermediate complexity climate model, C-GOLDSTEIN. The methodology can easily be extended to include higher derivatives and this is briefly explored in Section 5.4.

5.2 Emulating model derivatives with Gaussian processes

The derivatives of a posterior Gaussian process remain Gaussian processes with mean and covariance functions obtained by the relevant derivatives of the posterior mean and covariance functions, (O'Hagan, 1992). This can be applied to any Gaussian process emulator and in Section 5.2.1 we apply it to the standard Gaussian process emulator as described in Chapter 3. We then show, in Section 5.2.2, how this can be done when there are derivatives available to incorporate in the training data, in addition to model response. This is the same type of emulator as is presented in Chapter 4, for emulating model response. Since the derivatives of a Gaussian process remain a Gaussian process we can validate an emulator of derivatives using the same approaches to that of a standard emulator. Gaussian process validation is discussed more fully in Chapter 3. For further detail on emulators of derivatives see the *Generic Emulate Derivatives* thread in the MUCM toolkit at *http://mucm.aston.ac.uk/MUCM/MUCMToolkit.*

5.2.1 Emulating derivatives with only model response

If we cannot afford to run an adjoint model, or an appropriate adjoint model does not exist, we can still emulate model derivatives as follows. We require the simulator, $\eta(\cdot)$, to be a smooth function and, as in Chapter 3, begin by describing the uncertainty about the simulator output by a Gaussian process. O'Hagan (1992) shows that the derivatives of a Gaussian process remain a Gaussian process and thus we can model the derivatives of $\eta(\cdot)$, assuming $\eta(\cdot)$ is differentiable everywhere, by a Gaussian process. A Gaussian process is defined by its mean and covariance functions so these must now be described.

We specify the prior mean for the function output as

$$\mathrm{E}\left[\eta(\mathbf{x})|\,\boldsymbol{\beta}\right] = \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}, \tag{5.1}$$

and therefore the prior mean for the derivative is:

$$\begin{aligned} \mathrm{E}\left[\frac{\partial}{\partial x^{(d)}}\eta(\mathbf{x})\middle|\,\boldsymbol{\beta}\right] &= \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} \\ &= \tilde{\mathbf{h}}(\mathbf{x},d)^T\boldsymbol{\beta}, \text{ for } d \neq 0. \end{aligned} \tag{5.2}$$

We have the location in the input space represented by $\mathbf{x}$ and the value of d refers to which input the derivative is with respect to. For example, $(\mathbf{x}_i, d = j)$ refers to the derivative at location i in the input space w.r.t input j. We cannot have $d = 0$ as this refers to the model response and here we are emulating model derivatives. The vector $\mathbf{h}(\mathbf{x})^T$, of length $1 \times q$, comprises known, differentiable functions of $\mathbf{x}$; and $\boldsymbol{\beta}$ is a $q \times 1$ vector of unknown coefficients. We can choose the form of $\mathbf{h}(.)$ based on our prior beliefs about $\eta(\cdot)$. For example, if we believe $\eta(\mathbf{x})$ to be approximately linear in $\mathbf{x}$, then choosing $\mathbf{h}(\mathbf{x})^T = (1 \;\; \mathbf{x})$ would be appropriate and this would lead to a constant prior mean for the derivative.

The covariances between $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$ are defined, for some twice differentiable covariance function as:

$$\mathrm{Cov}\left[\eta(\mathbf{x}_i), \eta(\mathbf{x}_j)\right] = \sigma^2 c(\mathbf{x}_i, \mathbf{x}_j). \tag{5.3}$$

The covariances between derivatives, therefore, are:

$$\begin{aligned} \mathrm{Cov}\left[\frac{\partial}{\partial x^{(d_i)}}\eta(\mathbf{x}_i), \frac{\partial}{\partial x^{(d_j)}}\eta(\mathbf{x}_j)\right] &= \sigma^2\frac{\partial^2}{\partial x_i^{(d_i)}\partial x_j^{(d_j)}}c(\mathbf{x}_i, \mathbf{x}_j) \\ &= \tilde{c}\{(\mathbf{x}_i, d), (\mathbf{x}_j, d)\}, \text{ for } d \neq 0. \end{aligned} \tag{5.4}$$

A common form of correlation function is the infinitely differentiable Gaussian form $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\Theta(\mathbf{x}_i - \mathbf{x}_j)\}$, where Θ is a diagonal matrix of positive smoothness parameters, $\theta\{k\}$ with $k \in \{1, \ldots, p\}$ and p is the number of inputs.

We assume that our prior information about $\boldsymbol{\beta}$ and σ^2 will be weak, and so for the prior distribution use:

$$p(\boldsymbol{\beta}, \sigma^2) \propto \frac{1}{\sigma^2}. \tag{5.5}$$

In summary our prior beliefs take the form:

$$\eta(\mathbf{x})|\boldsymbol{\beta}, \sigma^2, \Theta \sim GP(\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}, \sigma^2 c(\mathbf{x}, \mathbf{x})) \tag{5.6}$$

$$\tilde{\eta}(\mathbf{x}, d)|\boldsymbol{\beta}, \sigma^2, \Theta \sim GP(\tilde{\mathbf{h}}(\mathbf{x}, d)^T\boldsymbol{\beta}, \sigma^2\tilde{c}\{(\mathbf{x}, d), (\mathbf{x}, d)\}), \text{ for } d \neq 0. \tag{5.7}$$

We are only concerned here with emulating derivatives and therefore require that $d \neq 0$, the general case where $d \in \{0, 1, 2 \ldots, p\}$ is covered in Section 5.2.2. Although $\eta(\mathbf{x}) = \tilde{\eta}(\mathbf{x}, d = 0)$, i.e (5.6) is (5.7) when $d = 0$, we keep the two separate in this section for reasons which will soon become clear.

The next stage is to create a design which consists of a set of points in the input space at which the simulator is to be run to create the training data. As here we are emulating derivatives, not function output, a slightly different design question is posed. It may be that an optimal design for emulating derivatives, with only model response in the training data, has different properties to that of the standard problem of emulating model response with model response. We touch on this subject further in Section 5.3.1 but without any formal results. Having specified the location of the design points we arrange this information in $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. The simulator is then run at each of the input configurations and this results in our training data of model response:

$$\begin{aligned} \mathbf{y} &= \{\eta(\mathbf{x}_1), \eta(\mathbf{x}_2), \ldots, \eta(\mathbf{x}_n)\} \\ &= \eta(D), \end{aligned} \tag{5.8}$$

a vector of length n.

The process of deriving the posterior distribution is similar to that for a standard emulator. We begin by writing the distribution of the training data, $\mathbf{y}$, conditional on the parameters β and σ^2. The training data consists of model response only and so from (5.6) and (5.8) we get:

$$\mathbf{y}|\,\boldsymbol{\beta}, \sigma^2, \Theta \sim N(H\boldsymbol{\beta}, \sigma^2 A), \tag{5.9}$$

where $H = [\mathbf{h}(\mathbf{x}_1), \ldots, \mathbf{h}(\mathbf{x}_n)]^{\mathrm{T}}$ and A is the $n \times n$ matrix of correlations of the training data: $A = c(D, D)$. Now we wish to update the distribution of $\tilde{\eta}()$ and partition in the following way:

$$\begin{pmatrix} \tilde{\eta}(\mathbf{x}, d) \\ \mathbf{y} \end{pmatrix},$$

with mean $= \begin{pmatrix} \tilde{\mathbf{h}}(\mathbf{x}, d)^T \boldsymbol{\beta} \\ H\boldsymbol{\beta} \end{pmatrix}$, and covariance $= \begin{pmatrix} \sigma^2 \tilde{c}\{(\mathbf{x}, d), (\mathbf{x}, d)\} & \tilde{t}(\mathbf{x}, d)^T \\ \tilde{t}(\mathbf{x}, d) & \sigma^2 A \end{pmatrix}$,
where

$$\begin{aligned} \tilde{\mathbf{t}}(\mathbf{x}, d)^T &= \operatorname{Corr}\left[\frac{\partial}{\partial x^{(d)}} \eta(\mathbf{x}), \eta(\mathbf{x}_1)\right], \ldots, \operatorname{Corr}\left[\frac{\partial}{\partial x^{(d)}} \eta(\mathbf{x}), \eta(\mathbf{x}_n)\right] \\ &= \frac{\partial}{\partial x^{(d)}} c(\mathbf{x}, \mathbf{x}_1), \ldots, \frac{\partial}{\partial x^{(d)}} c(\mathbf{x}, \mathbf{x}_n) \\ &= \tilde{c}\{(\mathbf{x}, d), D\}. \end{aligned} \tag{5.10}$$

We can now use standard techniques of conditioning multivariate normal distributions and this gives:

$$\tilde{\eta}(\mathbf{x}, d) | \, \boldsymbol{\beta}, \sigma^2, \Theta, \mathbf{y} \sim N(\tilde{m}^*(\mathbf{x}, d), \sigma^2 \tilde{c}^*\{(\mathbf{x}, d), (\mathbf{x}, d)\}), \tag{5.11}$$

where

$$\begin{aligned} \tilde{m}^*(\mathbf{x}, d) &= \tilde{\mathbf{h}}(\mathbf{x}, d)^T \boldsymbol{\beta} + \tilde{\mathbf{t}}(\mathbf{x}, d)^T A^{-1}(\mathbf{y} - H\boldsymbol{\beta}), \\ \tilde{c}^*\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} &= \tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} - \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^T A^{-1} \tilde{\mathbf{t}}(\mathbf{x}_j, d_j), \end{aligned}$$

and we are only interested in the case here where $d \neq 0$.

The next part to building an emulator of derivatives with only function output in the training data is the same as for the standard emulator: we apply Bayes Theorem to (5.5) and (5.9), and this results in a joint Normal Inverse Gamma posterior distribution for $(\boldsymbol{\beta}, \sigma^2)$:

$$f(\boldsymbol{\beta}, \sigma^2 | \mathbf{y}, \Theta) \propto {\sigma^2}^{\frac{n+2}{2}} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}})^T H^T A^{-1} H (\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}) + (n - q - 2)\widehat{\sigma^2}\right\}, \tag{5.12}$$

where

$$\widehat{\beta} = \left(H^{\mathrm{T}} A^{-1} H\right)^{-1} H^{\mathrm{T}} A^{-1} \mathbf{y}, \tag{5.13}$$

$$\widehat{\sigma^2} = (n - q - 2)^{-1} \mathbf{y}^{\mathrm{T}} \left\{A^{-1} - A^{-1} H \left(H^{\mathrm{T}} A^{-1} H\right)^{-1} H^{\mathrm{T}} A^{-1}\right\} \mathbf{y}. \tag{5.14}$$

From this we see that:

$$\boldsymbol{\beta}|\sigma^2, \mathbf{y} \sim N\left(\widehat{\boldsymbol{\beta}}, \sigma^2(H^T A^{-1} H)^{-1}\right), \tag{5.15}$$

and integrating (5.12) with respect to $\boldsymbol{\beta}$ gives us:

$$\sigma^2|\,\mathbf{y}, \Theta \sim \text{InvGam}\left(\frac{n-q}{2}, \frac{(n-q-2)\widehat{\sigma}^2}{2}\right), \tag{5.16}$$

as in the building of a standard emulator as given in Chapter 3.

Now to find the distribution of the derivatives of the simulator, conditional only on the function output in the training data the next step is to take the product of (5.11) and (5.15) and then integrate out $\boldsymbol{\beta}$. This results in:

$$\tilde{\eta}(\mathbf{x}, d)|\sigma^2, \mathbf{y}, \Theta \sim GP\left(\tilde{m}^{**}(\mathbf{x}, d), \sigma^2\tilde{c}^{**}\{(\mathbf{x}, d)\,, (\mathbf{x}, d)\}\right), \tag{5.17}$$

where for $d \neq 0$:

$$\begin{aligned} \tilde{m}^{**}(\mathbf{x}, d) &= \tilde{\mathbf{h}}(\mathbf{x}, d)^T\widehat{\boldsymbol{\beta}} + \tilde{\mathbf{t}}(\mathbf{x}, d)^T A^{-1}(\mathbf{y} - H\widehat{\boldsymbol{\beta}}), & (5.18) \\ \tilde{c}^{**}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} &= \tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} \;-\; \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^{\mathrm{T}} A^{-1}\tilde{\mathbf{t}}(\mathbf{x}_j, d_j) \\ &\quad + \left(\tilde{\mathbf{h}}(\mathbf{x}_i, d_i)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^{\mathrm{T}} A^{-1} H\right)\left(H^{\mathrm{T}} A^{-1} H\right)^{-1} \\ &\quad \times \left(\tilde{\mathbf{h}}(\mathbf{x}_j, d_j)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_j, d_j)^{\mathrm{T}} A^{-1} H\right)^{\mathrm{T}}. & (5.19) \end{aligned}$$

Finally integrating out σ^2 leaves us with, conditional on Θ, a t process with $n-q$ degrees of freedom. The posterior mean is $\tilde{m}^{**}(\mathbf{x}, d)$ and can be used as a fast approximation to the derivative of $\eta(\mathbf{x})$ with respect to input d. The posterior covariance between the derivative of $\eta(\mathbf{x}_i)$ with respect to input d and the derivative of $\eta(\mathbf{x}_j)$ with respect to input d is $\widehat{\sigma}^2\tilde{c}^{**}(\mathbf{x}, d)$ where
$\widehat{\sigma}^2 = (n-q-2)^{-1}\mathbf{y}^{\mathrm{T}}\left\{A^{-1} - A^{-1}H\left(H^{\mathrm{T}}A^{-1}H\right)^{-1}H^{\mathrm{T}}A^{-1}\right\}\mathbf{y}$.

The covariance function includes a matrix of roughness parameters, Θ. As when emulating response, we cannot analytically integrate out Θ from the posterior distribution. In this chapter, as throughout this thesis, we choose to fix θ, having estimated its value from the training data.

5.2.2 The general case: emulating model response and derivatives

If we can afford to run an adjoint model, or derivative information has already been generated perhaps as a result of some sensitivity analysis, we can include that information when building an emulator of derivatives. This is a similar framework to that of Chapter 4, the difference is that in this chapter our goal is to emulate derivatives, not model response. Due to similarities in the setup with Chapter 4 and with parallel objectives to Section 5.2.1, in this section we present the methodology for where we have derivatives and model response with which to emulate model derivatives within the general case. So far, in this thesis we have separated emulating model response and derivatives, with and without derivative information in the training data, and in this section we bring these situations together and present the general case for emulating model response and derivatives, with or without derivatives in the training data. As this section is effectively a summary of methods described earlier in this thesis some repetition is inevitable.

We begin by describing the uncertainty about the simulator output by a Gaussian process. Assuming $\eta(\cdot)$ is differentiable everywhere, we can proceed by modelling the derivatives of $\eta(\cdot)$ also by a Gaussian process:

We specify the prior mean for the function output as

$$\mathrm{E}\left[\eta(\mathbf{x})|\,\boldsymbol{\beta}\right] = \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}, \tag{5.20}$$

and the prior mean for the derivative is:

$$\mathrm{E}\left[\frac{\partial}{\partial x^{(d)}}\eta(\mathbf{x})\middle|\,\boldsymbol{\beta}\right] \;=\; \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}. \tag{5.21}$$

We bring these functions together in $\tilde{\mathbf{h}}(\mathbf{x}, d)$, which is defined as:

$$\tilde{\mathbf{h}}(\mathbf{x}, d)^T\boldsymbol{\beta} = \begin{cases} \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} & \text{for } d = 0 \\ \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta} & \text{for } d \neq 0 \end{cases},$$

where $d \in \{0, 1, \ldots, p\}$. We have the location in the input space represented by $\mathbf{x}$ and the value of d determines whether or not we are interested in the derivative at that point. In this way $(\mathbf{x}_i, d = 0)$, for example, would refer to the model response at point i in the input space while $(\mathbf{x}_j, d = 1)$ refers to the derivative w.r.t input 1 at point j in the input space. The vector $\mathbf{h}(\mathbf{x})^T$, of length $1 \times q$, comprises known, differentiable functions of $\mathbf{x}$; and $\boldsymbol{\beta}$ is a $q \times 1$ vector of unknown coefficients. We choose the form of $\mathbf{h}(.)$ based on our prior beliefs about $\eta(\cdot)$.

The covariances between $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$ are defined, for some twice differentiable covariance function as:

$$\mathrm{Cov}\left[\eta(\mathbf{x}_i), \eta(\mathbf{x}_j)\right] = \sigma^2 c(\mathbf{x}_i, \mathbf{x}_j), \tag{5.22}$$

and covariances between derivatives are:

$$\mathrm{Cov}\left[\frac{\partial}{\partial x^{(d_i)}}\eta(\mathbf{x}_i), \frac{\partial}{\partial x^{(d_j)}}\eta(\mathbf{x}_j)\right] = \sigma^2 \frac{\partial^2}{\partial x_i^{(d_i)} \partial x_j^{(d_j)}} c(\mathbf{x}_i, \mathbf{x}_j). \tag{5.23}$$

As we are interested in both the model response and derivatives we require the correlations between points, between derivatives and points and also between derivatives themselves. These correlations are all incorporated in $\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\}$:

$$\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} = \begin{cases} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = d_j = 0 \\ \frac{\partial}{\partial x_i^{(d_i)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i \neq 0 \text{ and } d_j = 0 \\ \frac{\partial^2}{\partial x_i^{(d_i)} \partial x_j^{(d_j)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i, d_j \neq 0\,. \end{cases}$$

A common form of correlation function is the infinitely differentiable Gaussian form $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\Theta(\mathbf{x}_i - \mathbf{x}_j)\}$, where Θ is a diagonal matrix of positive smoothness parameters, $\theta\{k\}$ with $k \in \{1, \ldots, p\}$ and p is the number of inputs. While the Gaussian form is a popular choice of correlation function it may not be suitable for every simulator and we discuss alternative correlation functions in Chapter 4.

We assume that our prior information about $\boldsymbol{\beta}$ and σ^2 will be weak, and so for the prior distribution use

$$p(\boldsymbol{\beta}, \sigma^2) \propto \frac{1}{\sigma^2}. \tag{5.24}$$

In summary our prior beliefs take the form:

$$\tilde{\eta}(\mathbf{x}, d)|\boldsymbol{\beta}, \sigma^2, \Theta \sim GP(\tilde{\mathbf{h}}(\mathbf{x}, d)^T\boldsymbol{\beta}, \sigma^2\tilde{c}\{(\mathbf{x}, d), (\mathbf{x}, d)\}), \tag{5.25}$$

where $d \in \{0, 1, \ldots, p\}$.

The next stage is to create a design which consists of a set of points in the input space at which the simulator or adjoint is to be run to create the training data. We are not restricted to a design which has either model response at every point or all first derivatives at a point. Expert knowledge may help to identify at which points in the design space model response would be beneficial and at which point the derivatives w.r.t to various inputs are most informative. Having specified the location of the design points and determined at which points we require function output and at which points we require first derivatives, we arrange this information in $\tilde{D} = \{(\mathbf{x}_k, d_k)\}$. We have $\mathbf{x}_k$ which refers to the location in the design and d_k determines whether at point $\mathbf{x}_k$ we require function output or a first derivative w.r.t one of the inputs. The simulator, $\eta(\cdot)$, or the adjoint of the simulator, $\tilde{\eta}(\cdot)$, (depending on the value of each d), is then run at each of the input configurations. This results in our training data:

$$\begin{aligned} \tilde{\mathbf{y}} &= \{\tilde{\eta}(\mathbf{x}_1, d_1), \tilde{\eta}(\mathbf{x}_2, d_2), \ldots, \tilde{\eta}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})\} \\ &= \tilde{\eta}(\tilde{D}), \end{aligned} \tag{5.26}$$

a vector of length $\tilde{n}$.

We begin the process of deriving the posterior process by writing the distribution of the training data, $\tilde{\mathbf{y}}$, conditional on the parameters β and σ^2. The training data can consist of derivatives and model response and so from (5.25) and (5.26) we get:

$$\tilde{\mathbf{y}} |\, \boldsymbol{\beta}, \sigma^2, \Theta \sim N(\tilde{H}\boldsymbol{\beta}, \sigma^2 \tilde{A}), \tag{5.27}$$

where $\tilde{H} = [\tilde{\mathbf{h}}(\mathbf{x}_1, d_1), \ldots, \tilde{\mathbf{h}}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})]^{\mathrm{T}}$ and $\tilde{A}$ is the $\tilde{n} \times \tilde{n}$ matrix of correlations between points, between points and derivatives and between derivatives themselves, in the training data: $\tilde{A} = \tilde{c}(\tilde{D}, \tilde{D})$. Now we wish to update (5.25), the distribution of $\tilde{\eta}()$, and we partition in the following way:

$$\begin{pmatrix} \tilde{\eta}(\mathbf{x}, d) \\ \tilde{\mathbf{y}} \end{pmatrix},$$

with mean $= \begin{pmatrix} \tilde{\mathbf{h}}(\mathbf{x}, d)^T \boldsymbol{\beta} \\ \tilde{H}\boldsymbol{\beta} \end{pmatrix}$, and covariance $= \begin{pmatrix} \sigma^2 \tilde{c}\{(\mathbf{x}, d), (\mathbf{x}, d)\} & \tilde{t}(\mathbf{x}, d)^T \\ \tilde{t}(\mathbf{x}, d) & \sigma^2 \tilde{A} \end{pmatrix}$,
where $\tilde{\mathbf{t}}(\mathbf{x}, d)^T$ consists of the correlations of the derivative we are emulating and the training data:

$$\begin{aligned} \tilde{\mathbf{t}}(\mathbf{x}, d)^T &= \text{Corr}\left[\tilde{\eta}(\mathbf{x}, d), \tilde{\eta}(\mathbf{x}_1, d_1)\right], \ldots, \text{Corr}\left[\tilde{\eta}(\mathbf{x}, d), \tilde{\eta}(\mathbf{x}_n, d_n)\right] \\ &= \tilde{c}\{(\mathbf{x}, d), \tilde{D}\}. \end{aligned} \tag{5.28}$$

We can now use standard techniques of conditioning multivariate normal distributions to give

$$\tilde{\eta}(\mathbf{x}, d) |\, \boldsymbol{\beta}, \sigma^2, \Theta, \tilde{\mathbf{y}} \sim N(\tilde{m}^*(\mathbf{x}, d), \sigma^2 \tilde{c}^*\{(\mathbf{x}, d), (\mathbf{x}, d)\}), \tag{5.29}$$

where

$$\begin{aligned} \tilde{m}^*(\mathbf{x}, d) &= \tilde{\mathbf{h}}(\mathbf{x}, d)^T \boldsymbol{\beta} + \tilde{\mathbf{t}}(\mathbf{x}, d)^T \tilde{A}^{-1}(\tilde{\mathbf{y}} - \tilde{H}\boldsymbol{\beta}), \\ \tilde{c}^*\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} &= \tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} - \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^T \tilde{A}^{-1} \tilde{\mathbf{t}}(\mathbf{x}_j, d_j), \end{aligned}$$

and d could take any value in $\{0, 1, \ldots, p\}$. This, (5.29), is the joint distribution of $\eta(\cdot)$ and its derivatives, conditional on the parameters, $\boldsymbol{\beta}, \sigma^2$ and Θ.

The next part of building our general emulator is the same as building a standard emulator, as given in Chapter 3 and repeated here. We apply Bayes Theorem with (5.24) and (5.27) and this results in a joint Normal Inverse Gamma posterior distribution for $(\boldsymbol{\beta}, \sigma^2)$:

$$f(\boldsymbol{\beta}, \sigma^2|\tilde{\mathbf{y}}, \Theta) \propto {\sigma^2}^{\frac{\tilde{n}+2}{2}} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}})^T \tilde{H}^T \tilde{A}^{-1} \tilde{H}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}) + (\tilde{n} - q - 2)\widehat{\sigma^2}\right\}, \tag{5.30}$$

where

$$\widehat{\beta} = \left(\tilde{H}^{\mathrm{T}} \tilde{A}^{-1} \tilde{H}\right)^{-1} \tilde{H}^{\mathrm{T}} \tilde{A}^{-1} \tilde{\mathbf{y}}, \tag{5.31}$$

$$\widehat{\sigma^2} = (\tilde{n} - q - 2)^{-1} \tilde{\mathbf{y}}^{\mathrm{T}} \left\{\tilde{A}^{-1} - \tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1} \tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\right\}\tilde{\mathbf{y}}. \tag{5.32}$$

From this we see that:

$$\boldsymbol{\beta}|\sigma^2, \tilde{\mathbf{y}} \sim N\left(\widehat{\boldsymbol{\beta}}, \sigma^2(\tilde{H}^T \tilde{A}^{-1} \tilde{H})^{-1}\right), \tag{5.33}$$

and integrating (5.30) with respect to $\boldsymbol{\beta}$ gives us:

$$\sigma^2|\, \tilde{\mathbf{y}}, \theta \sim \text{InvGam}\left(\frac{\tilde{n} - q}{2}, \frac{(\tilde{n} - q - 2)\widehat{\sigma^2}}{2}\right), \tag{5.34}$$

as in the building of a standard emulator.

Now to find the joint distribution of $\eta(\cdot)$ and the derivatives of $\eta(\cdot)$, conditional only on the training data, the next step is to take the product of (5.29) and (5.33) and then integrate out $\boldsymbol{\beta}$. This results in:

$$\tilde{\eta}(\mathbf{x}, d)|\sigma^2, \mathbf{y}, \Theta \sim GP\left(\tilde{m}^{**}(\mathbf{x}, d), \sigma^2 \tilde{c}^{**}\{(\mathbf{x}, d)\,, (\mathbf{x}, d)\}\right), \tag{5.35}$$

where:

$$\tilde{m}^{**}(\mathbf{x}, d) = \tilde{\mathbf{h}}(\mathbf{x}, d)^T \widehat{\boldsymbol{\beta}} + \tilde{\mathbf{t}}(\mathbf{x}, d)^T \tilde{A}^{-1}(\tilde{\mathbf{y}} - \tilde{H}\widehat{\boldsymbol{\beta}}), \tag{5.36}$$

$$\begin{aligned} \tilde{c}^{**}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} &= \tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} \;-\; \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^{\mathrm{T}} \tilde{A}^{-1} \tilde{\mathbf{t}}(\mathbf{x}_j, d_j) \\ &\quad + \left(\tilde{\mathbf{h}}(\mathbf{x}_i, d_i)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_i, d_i)^{\mathrm{T}} \tilde{A}^{-1} \tilde{H}\right) \left(\tilde{H}^{\mathrm{T}} \tilde{A}^{-1} \tilde{H}\right)^{-1} \\ &\quad \times \left(\tilde{\mathbf{h}}(\mathbf{x}_j, d_j)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_j, d_j)^{\mathrm{T}} \tilde{A}^{-1} \tilde{H}\right)^{\mathrm{T}}. \end{aligned} \tag{5.37}$$

Finally integrating out σ^2 leaves us with, conditional on Θ, a t process with $\tilde{n}-q$ degrees of freedom. The posterior mean is $\tilde{m}^{**}(\mathbf{x},d)$ and can be used as a fast approximation to the derivative of $\eta(\mathbf{x})$ with respect to input d if $d \neq 0$, and as a fast approximation to $\eta(\mathbf{x})$ if $d=0$. The posterior covariance between the derivatives of $\eta(\mathbf{x}_i)$ and $\eta(\mathbf{x}_j)$ with respect to input d, or between the model response $\eta(\mathbf{x}_i)$, and $\eta(\mathbf{x}_j)$, depending on the value of d, is $\widehat{\sigma}^2\tilde{c}^{**}(\mathbf{x},d)$ where $\widehat{\sigma}^2 = (\tilde{n}-q-2)^{-1}\tilde{\mathbf{y}}^{\mathrm{T}}\left\{\tilde{A}^{-1} - \tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\right\}\tilde{\mathbf{y}}$.

The covariance function includes a matrix of roughness parameters, Θ. As when emulating response, we cannot analytically integrate out Θ from the posterior distribution. In this chapter, as throughout this thesis, we choose to fix θ, having estimated its value from the training data. Further discussion of methods to estimate Θ are given in Chapter 3.

5.3 Examples

5.3.1 Toy Example

We illustrate the methods described in Section 5.2 with a 1-dimensional toy model example. The true model is

$$\eta(x) = \frac{x}{3} + \sin(x),$$

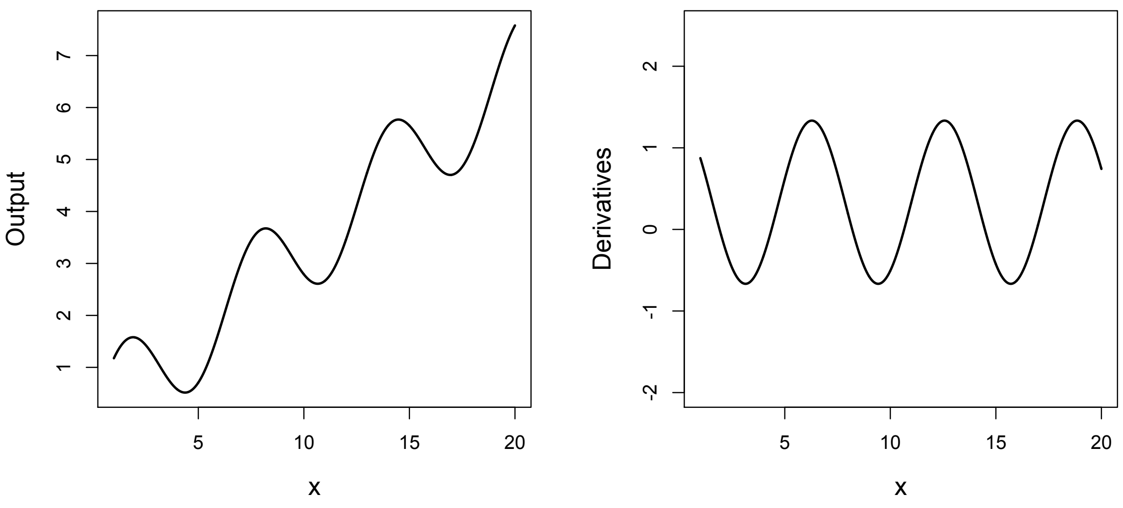
and we are interested in this function over the input region $[1, 20]$. The true function is shown in Figure 5.1a and the corresponding true derivatives to be emulated, $\frac{\partial}{\partial x}\eta(x) = \frac{1}{3} + \cos(x)$, are shown in Figure 5.1b.

We take a linear form for the prior mean of the model response, and set $\mathbf{h}(x)^T = (1\ x)$, which gives a constant prior mean for the derivative: $\frac{\partial}{\partial x}\mathbf{h}(x)^T = (0\ 1)$, and we have $q=2$. We choose the Gaussian form for the correlation function:

$$\mathrm{c}(x_i, x_j) = \exp\{-\theta\,(x_i - x_j)^2\},$$

and so the correlation between a derivative, $\tilde{\eta}(x_i, 1)$ and a point, $\tilde{\eta}(x_i, 0)$ becomes:

$$\frac{\partial}{\partial x_i}c(x_i, x_j) = 2\theta\,(x_j - x_i)\exp\{-\theta\,(x_i - x_j)^2\}, \tag{5.38}$$



(a) Toy Model response (b) Derivatives of Toy Model output

Figure 5.1: Toy Model

and the correlation between two derivatives, $\tilde{\eta}(x_i, 1)$ and $\tilde{\eta}(x_j, 1)$ is:

$$\frac{\partial^2}{\partial x_i \partial x_j} c(x_i, x_j) = \left(2\theta - 4\theta^2 (x_i - x_j)^2\right) \exp\{-\theta (x_i - x_j)^2\}. \tag{5.39}$$

We begin by choosing $n = 6$ design points with a maximin Latin hypercube sample, which in one dimension is simply the set of equidistant points. We scale the inputs to be in the unit range and also include in the training data the first derivative at each of the 6 points. This is results in the design,

$$\tilde{D} = \{(x_1, 0), \ldots, (x_6, 0), (x_1, 1), \ldots, (x_6, 1)\}.$$

Evaluating the simulator, and the derivative of the simulator where appropriate, at the points in $\tilde{D}$ results in the training data: $\tilde{\mathbf{y}} = \tilde{\eta}(\tilde{D})$, a vector of length $\tilde{n} = 12$. We now consider $\frac{\partial}{\partial x}\eta(x)$ as an unknown function and, following the methodology of Section 5.2, wish to derive the posterior distribution of the derivative of $\eta(x)$. We estimate the smoothness parameter, θ, by maximum likelihood estimation and obtain a value of 61.3. The estimated variance is $\hat{\sigma}^2 = 1.40$. We now attempt to predict the derivatives of the true function at a number of new input sites. We can see the performance of this emulator in Figure 5.2. The posterior mean is the dashed line and two standard deviations above and below this mean are shown by the dotted lines. The solid line reveals the true derivatives of the simulator at these points and the location of the n input sites for the training data are shown as crosses. Figure 5.2 shows that in this example, with derivatives in the training data in addition to the response, we can emulate the derivatives very well. Similarly to emulators of function output, at each design point where a derivative is known (shown by a cross) the uncertainty pinches in to zero. As expected the uncertainty grows as we predict a derivative away from a design point; as we reach halfway between design points though, we see the uncertainty reducing again, although not reaching zero. This is because we know the response of the function at design points as well as the derivative and this information allows the emulator to predict more accurately and with less uncertainty between design points.

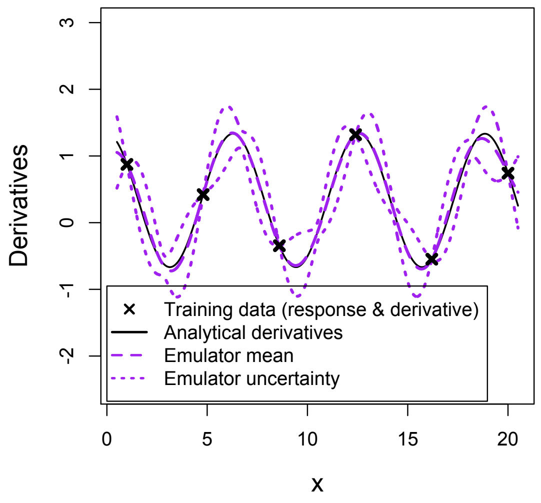


Figure 5.2: Emulating derivatives based on the function output and derivatives at 6 points.

The performance of the emulator in Figure 5.2 is very encouraging but if an adjoint to the simulator does not exist and derivative information is not available by any other method, then the training data for the emulator would consist of function output alone. Figure 5.3 shows the performance of the emulator built exactly as before but now with only the response of the simulator at the 6 points in the training data. The location of the training data is shown by crosses at the bottom of the plot; clearly these do not lie on the curve as the derivative is no longer included at these points. It is evident from Figure 5.3 that the emulator

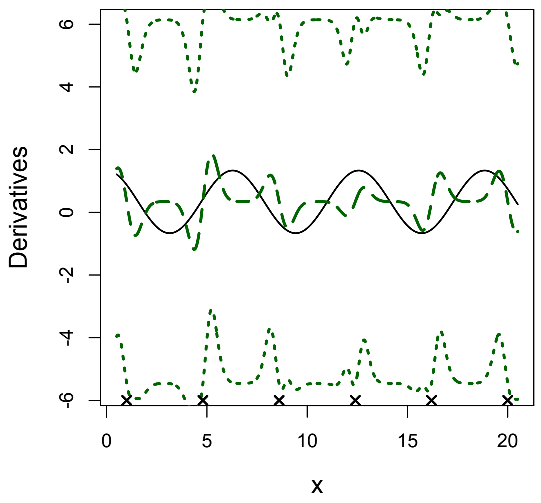
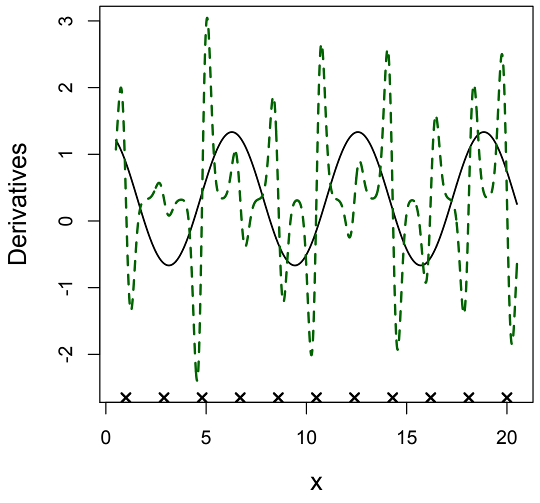
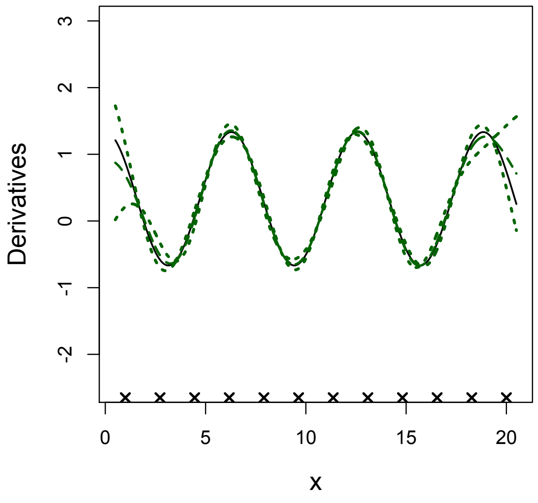


Figure 5.3: Emulating derivatives based on the function output at 6 points. Note that in comparison with Figure 5.2 a different scale on the y-axis has been used to show the much larger uncertainty in this plot.

is performing very badly; clearly there is not enough information in the model response at 6 points for the mean of the emulator to provide accurate predictions. As with standard emulation though, we would expect the performance of an emulator to improve with more runs of the simulator. Figure 5.4 shows the performance of an emulator, built similarly to that of Figure 5.3 but with $n = 11$ and $n = 12$ runs of $\eta(x)$. We adopt a maximin Latin hypercube sample for each emulator built which therefore results in 11 and 12 equidistant points for the emulators in Figures 5.4a and 5.4b respectively. In comparison to the emulator built with just 6 runs we see an improvement in the prediction of the derivatives in Figure 5.4b and reduced uncertainty across the whole input region. The emulator built with $n = 11$ runs, Figure 5.4a, is performing very badly though and despite the training data having just one less simulator run, the difference between the two emulators of Figures 5.4a and 5.4b is considerable. An explanation for the contrasting performances is the estimates of the smoothness parameter, θ. The maximum likelihood estimate for θ we obtain when building the emulator with $n = 11$ runs is 2889. The corresponding estimate when we have $n = 12$ runs



(a) Emulator built with $n = 11$ points.



(b) Emulator built with $n = 12$ points.

Figure 5.4: Emulating derivatives based on the function output alone.

is 36.4. This is a stark contrast brought about by just one extra design point, though it should be noted that due to the use of equidistant points for each emulator the two sets of design points are distinct. To investigate this further we build an emulator with the $n = 11$ equidistant points but fix the smoothness parameter at the value as estimated with the $n = 12$ equidistant points, which is 36.4. The resulting emulator is shown in Figure 5.5 and we now see a vastly improved performance, very similar to the emulator with $n = 12$ equidistant points. This confirms that the estimate of θ with $n = 11$ equidistant points is the cause of the poor performance of the emulator shown in Figure 5.4a.

Since we know that $n = 12$ equidistant points results in a sensible estimate of θ, we continue to attempt to build a valid emulator with $n = 11$ by selecting the $n = 12$ equidistant points and removing one point to result in a different emulator with $n = 11$. The maximum likelihood estimate for θ is now 50.1 and the performance of the resulting emulator is shown in Figure 5.6a. We can see from Figure 5.6a that while the uncertainty is slightly higher around $x = 6$ compared to Figure 5.5, the overall performance of the emulators is very similar with very good prediction of the derivatives. We can also see in Figure 5.6a the uncertainty pinching in between design points. Although very small, the

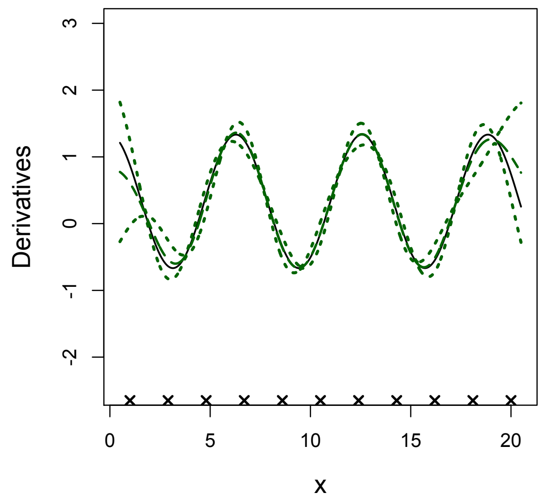
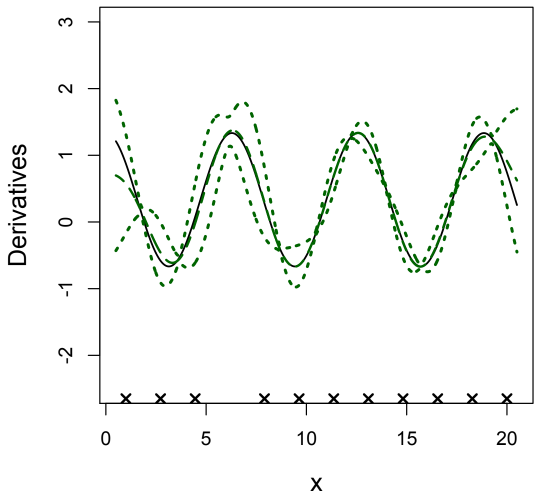


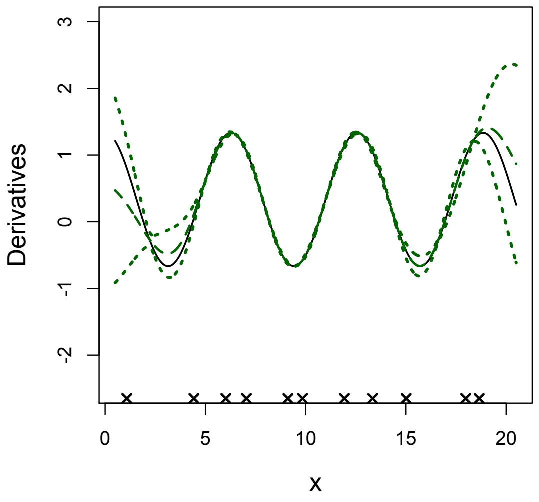
Figure 5.5: Emulating derivatives based on the function output at $n = 11$ points but θ fixed at 36.4

uncertainty does not quite reach zero at these points. For example between the 5th and 6th design points at $x = 10.46$, the standard deviation of the emulator is 0.016 (with mean -0.176). In contrast, the uncertainty is at its largest at design points. This is because without any derivative information known, at a design point the emulator is least certain of the gradient of the function at that point. The improvement and more appropriate estimate of θ is due to the spread of the design points. When all points are exactly the same distance apart, despite being optimally space filling, it becomes very difficult to estimate the smoothness parameter. Due to this we now adapt our design: a space filling element is still required and therefore a Latin hypercube sample is still appropriate but we remove the maximin criteria and instead, impose the condition that the distances between points is distinct. An emulator, built with $n = 11$ runs and this adapted LHS, is shown in Figure 5.6b. The resulting emulator performs very well and it would appear that 11 simulator runs might not all be required to build a valid emulator of derivatives for this model.

Since a real complex model would be computationally expensive to run and we would like to be able to build a valid emulator with small n we now try building emulators with the adapted LHS design but for $n = 9$ and 10. The performances of the resulting emulators can be seen in Figure 5.7. From this we see that while



(a) Emulator built with one point removed from 12 equidistant points

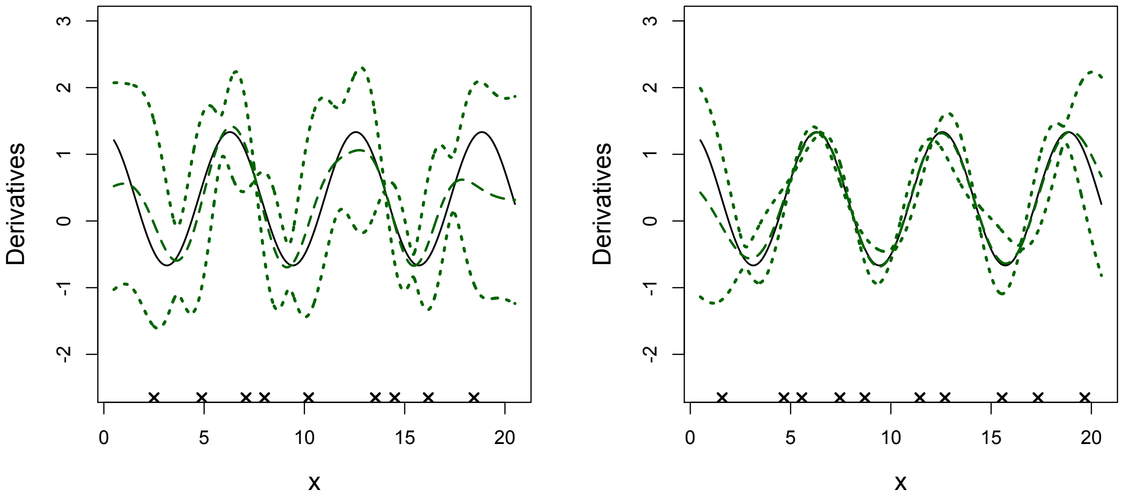


(b) Emulator built with a non-maximin LHS

Figure 5.6: Emulating derivatives based on the function output at $n = 11$ points.

the predictions from the emulator built with $n = 9$ points, match the analytical derivatives in some areas, to provide accurate predictions with low uncertainty across the whole of the specified input region, $n = 10$ runs are required.

In summary, although this is only a toy example it demonstrates how the method can work. With $n = 6$ runs and including the derivative at each point we can accurately emulate the derivatives of this model. If we don't have derivative information to include in the training data however, similar results can still be achieved by including the model response at just a few additional points. This is demonstrated in Figure 5.7b, where with just function output at $n = 10$ points the emulator is performing just as well as the emulator with $n = 6$ points and 6 derivatives, as shown in Figure 5.2.



(a) Emulator built with $n = 9$ (b) Emulator built with $n = 10$

Figure 5.7: Emulating derivatives with just function output and with the adapted LHS design.

5.3.2 Borehole Model

The simulator of this example is a model of the flow of water through a borehole in m^3/yr,

$$\text{flow rate} = \frac{2\pi T_u(H_u - H_l)}{\ln(\frac{r}{r_w})\left[1 + \frac{2LT_u}{\ln(\frac{r}{r_w})r_w^2 K_w} + \frac{T_u}{T_l}\right]}.$$

This is one of the models chosen to demonstrate the value of derivative information when emulating function output in Chapter 4. It is also the model chosen by Morris *et al.* (1993) to demonstrate their methodology for including derivative information in the analysis of computer models. There are eight inputs, shown along with a range of values that each input takes in Chapter 4, Table 4.2 and repeated here for completeness.

We vary all 8 input dimensions and build an emulator from 80 training runs chosen by generating a Latin hypercube sample. As in Example 5.3.1 we adopt a linear form for the prior mean and choose the covariance function, $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta (\mathbf{x}_i - \mathbf{x}_j)\right\}$. The correlation then between a point, $\mathbf{x}_i$, and a

Input	Name in model	Description	Range	Units
Input 1	r_w	radius of borehole	0.05 – 0.15	m
Input 2	r	radius of influence	100 – 50,000	m
Input 3	T_u	transmissivity of upper aquifer	63,070 – 115,600	m^2/yr
Input 4	H_u	potentiometric head of upper aquifer	990 – 1,110	m
Input 5	T_l	transmissivity of lower aquifer	63.1 – 116	m^2/yr
Input 6	H_l	potentiometric head of lower aquifer	700 – 820	m
Input 7	L	length of borehole	1,120 – 1,680	m
Input 8	K_w	hydraulic conductivity of borehole	1,500 – 15,000	m/yr

Table 4.2 Inputs for borehole model

derivative w.r.t input k at point j, $x_j^{(k)}$, is:

$$\frac{\partial}{\partial x_j^{(k)}} c(\mathbf{x}_i, \mathbf{x}_j) = 2\,\theta\{k\} \left(x_i^{(k)} - x_j^{(k)}\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\},$$

the correlation between two derivatives w.r.t input k but at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(k)}} c(\mathbf{x}_i, \mathbf{x}_j) = \left(2\,\theta\{k\} - 4\,\theta^2\{k\} \left(x_i^{(k)} - x_j^{(k)}\right)^2\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\},$$

and finally the correlation between two derivatives w.r.t inputs k and l, where $k \neq l$, at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(l)}} c(\mathbf{x}_i, \mathbf{x}_j) = 4\,\theta\{k\}\,\theta\{l\} \left(x_j^{(k)} - x_i^{(k)}\right) \left(x_i^{(l)} - x_j^{(l)}\right) \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \Theta (\mathbf{x}_i - \mathbf{x}_j)\}.$$

We choose a 20 point validation data set also from a Latin hypercube sample and analytically differentiate the simulator w.r.t each input. This generates 160 validation derivatives. We order the validation derivatives such that 1 - 20 are the derivatives w.r.t input 1, 21 - 40 are the derivatives w.r.t input 2 etc and

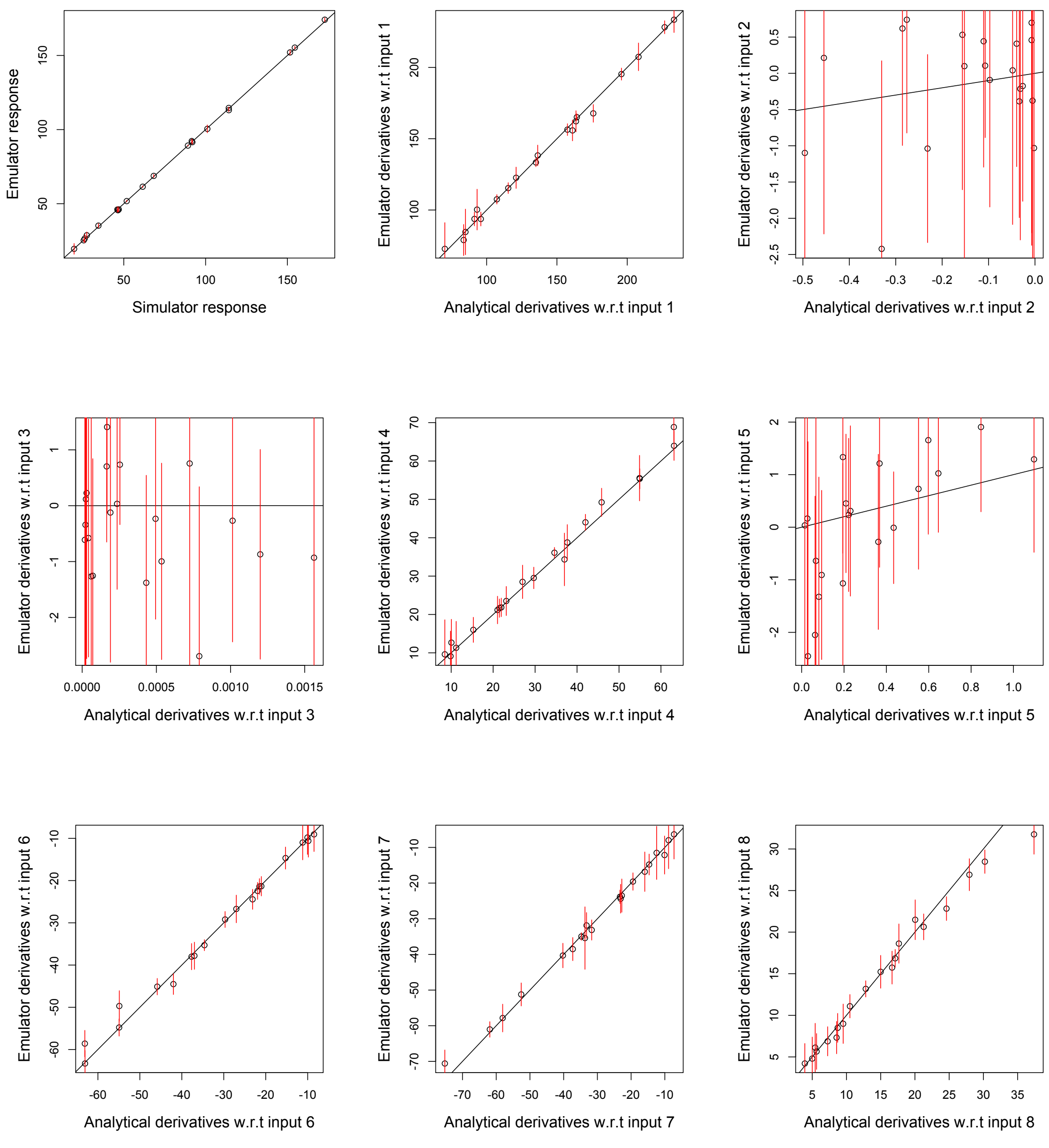


Figure 5.8: Emulated derivatives at the validation data points plotted against corresponding analytical derivatives.

we apply the diagnostic measures as given in Bastos and O'Hagan (2009). The results are summarised below and in Figures 5.8 and 5.9.

We plot the emulator mean and 2 standard deviations either side of the mean against the simulator output, at the validation input points, in the top left panel of Figure 5.8. From this it is clear that the response, the flow of water through a borehole, is emulated very well with low uncertainty. The emulated derivatives are plotted against the analytical derivatives w.r.t each of the inputs in the remaining panels of Figure 5.8. The predictions of the derivatives w.r.t inputs 1, 4, 6, 7 and 8 all match well against the analytical derivatives. There does however appear to be some conflict between the emulator and the analytical derivatives w.r.t. inputs 2, 3 and 5. The plots here of emulated derivatives against true derivatives do not see the points cluster around the $y = x$ line; the uncertainty however is appropriate with the emulator mean at most points lying within 2 standard deviations of the truth. The magnitude of the true derivatives for these inputs are the lowest of all the derivatives. The response therefore, would appear to be less sensitive to these three inputs and consequently, with function output alone, it is difficult to accurately estimate the value of these derivatives.

Further validation is required and we calculate the observed Mahalanobis distance, obtaining a value of 128. This diagnostic gives a measure of overall fit and should be compared to its reference distribution, which is the scaled F distribution with mean 160 and standard deviation 33. The observed Mahalanobis distance is within one standard deviation of its theoretical mean and so suggests a good level of overall performance of the emulator. We continue by calculating individual standardised errors and these are shown in the left panel of Figure 5.9. If the emulator is a good approximation to the analytical derivatives 95% of points would lie between -2 and $+2$, and with only a small number of outliers, this is seen in Figure 5.9. The outliers, as identified in Table 5.1, are all at different validation points but refer to derivatives w.r.t. 4th and 7th input only. In contrast there are no outliers for any of the derivatives w.r.t inputs 2, 3, or 5.

The panel on the right of Figure 5.9 shows pivoted Cholesky errors. Groups of unusually large or small errors in various parts of the sequence imply possible problems with parameter estimates in the emulator. In Figure 5.9 there is no obvious pattern to the errors though and apart from the small number of outliers most are within $[-2, 2]$ as required.

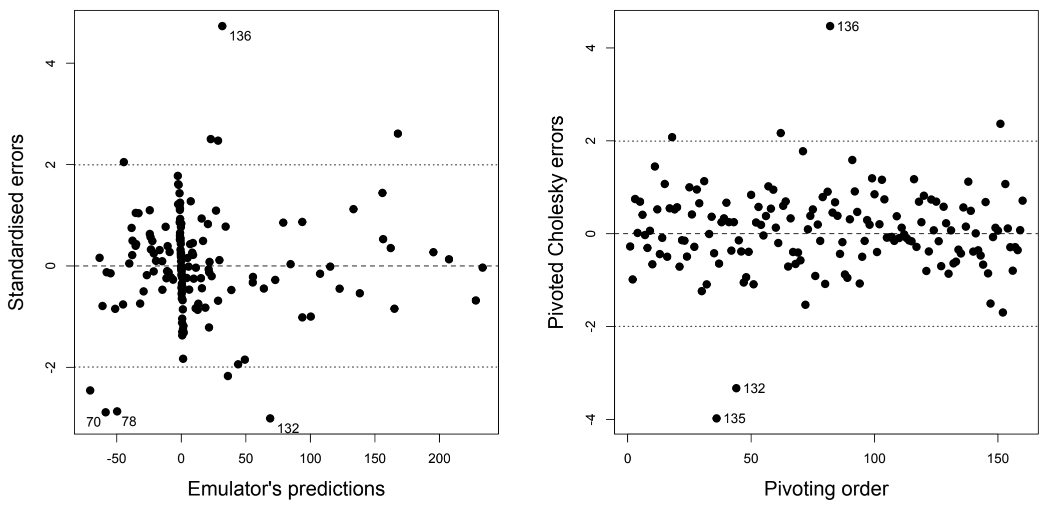


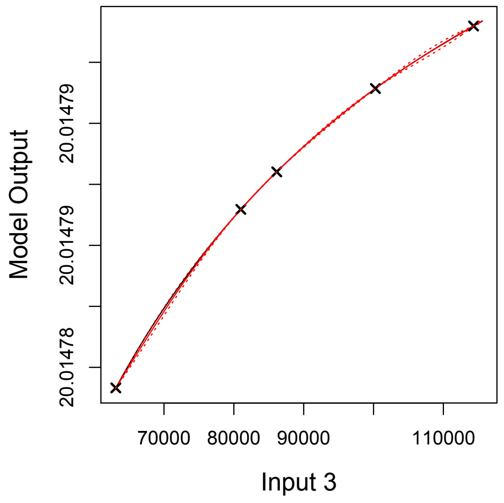
Figure 5.9: Diagnostic plots for the validation of the derivative emulator

Validation Index	Validation input point	Derivative w.r.t
70	10	Input 4
78	18	Input 4
132	12	Input 7
136	16	Input 7
135	15	Input 7

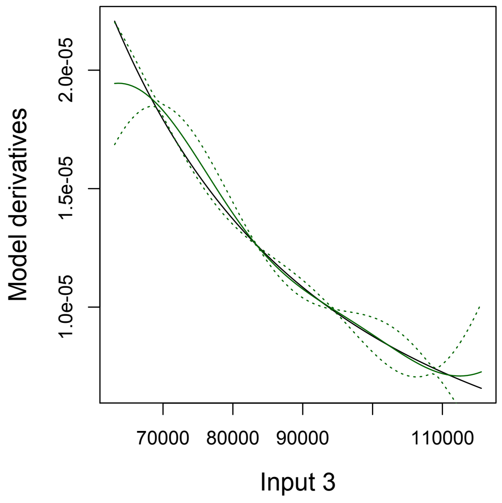
Table 5.1: Outliers shown in Figure 5.9.

We can therefore emulate the derivatives of the borehole model well though estimates of derivatives w.r.t inputs 2,3 and 5 should be considered carefully. An explanation for the relative poor emulation of the derivatives w.r.t inputs 2, 3 and 5 is that the model output is least sensitive to these inputs. Due to this, however, a model user may not be concerned with the values of these derivatives; any optimisation for example is unlikely to benefit much from this derivative information. To confirm that this is the case, and as the borehole model is a function which can be evaluated very cheaply, we can vary each of the inputs 2, 3 and 5 in turn while keeping the remaining inputs fixed at their lower limit. We choose an $n = 5$ point design, and select the design points from a Latin hypercube sample. Due to the results of Section 5.3.1, though, we ensure that none of the points are equidistant. We then calculate the model response and its derivatives at 100 points spread evenly across the input space and generate the emulator mean and standard deviation at each point to compare.

The results from this experiment applied to inputs 3 and 5 are shown in Figures 5.10 and 5.11 respectfully. We see that in just one dimension we can

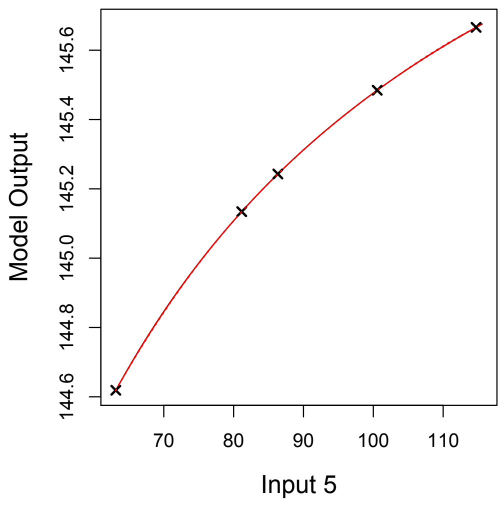


(a) Emulating the model response.

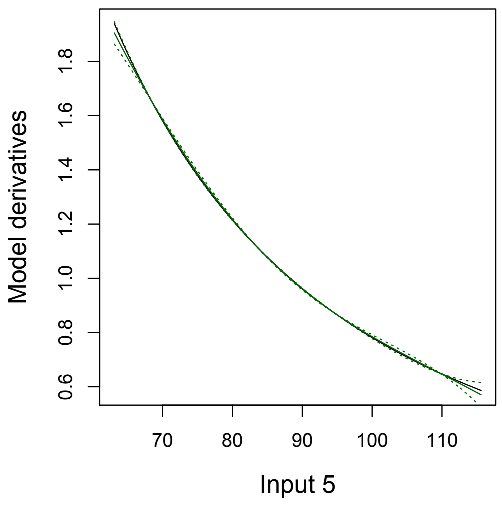


(b) Emulating the derivatives w.r.t input 3.

Figure 5.10: Emulating the borehole model and its derivatives in one dimension. Input 3 is varied and the remaining 7 inputs are fixed at their lower limit.



(a) Emulating the model response.



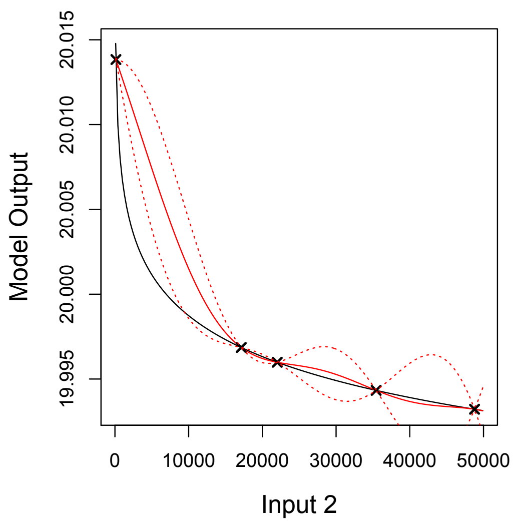
(b) Emulating the derivatives w.r.t input 5.

Figure 5.11: Emulating the borehole model and its derivatives in one dimension. Input 5 is varied and the remaining 7 inputs are fixed at their lower limit.

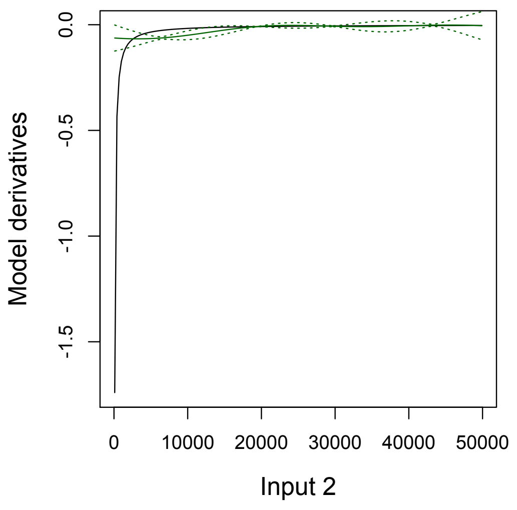
emulate the derivatives w.r.t to these inputs very well. It also is clear how little the model output is affected by changes in these inputs. The y-axis ranges in Figure 5.10a from 20.01478 to 20.01480 and in Figure 5.11a from 144.6176 to 145.6759. Comparing this to the range of the model output when we vary all 8 inputs, shown in Figure 5.8 (12.35903 to 198.5579), it is evident that inputs 3 and 5 do have very little effect on the output and explains why it is hard to achieve accurate estimates of the derivatives w.r.t these inputs with an emulator.

The performance of the emulator built varying only input 2, shown in Figure 5.12, provides less satisfactory results. The function appears to be non-stationary. We see small changes in the input causing relatively large changes in the output, for approximately input $2 < 10000$, however for the rest of the input space the change in output is smoother. We can see from Figure 5.12a that as 4 of the design points are situated in the ‘smoother’ region of the input space the resulting estimate of the smoothness parameter is appropriate for that region and the emulator then suffers in the ‘rougher’ region.

We tackle this problem by ‘warping’ the input space so as to obtain a stationary function. A log transformation is applied to input 2 and the subsequent



(a) Emulating the model response.



(b) Emulating the derivatives w.r.t input 2.

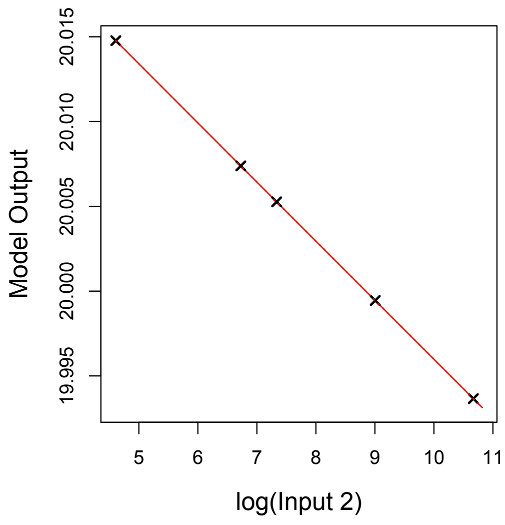
Figure 5.12: Emulating the borehole model and its derivatives in one dimension. Input 2 is varied and the remaining 7 inputs are fixed at their lower limit.

emulation of the model response and the derivatives w.r.t the log of input 2 are shown in Figures 5.13a and 5.13b respectively. We now have a stationary function, both the model response and the derivatives of which we are able to emulate very well. Transforming the emulator's predictions back to the meaningful scale of input 2 results in Figures 5.13c and 5.13d.

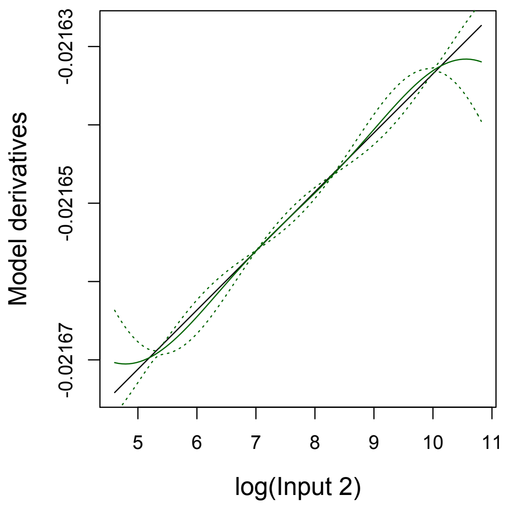
Applying a log transformation to input 2 when varying all 8 input parameters makes very little difference to the performance of the resulting emulator. This is because the effect of input 2 is too small in comparison to the the other inputs and so there is not enough signal for the emulator to accurately predict the derivatives w.r.t to this input.

5.3.3 C-GOLDSTEIN

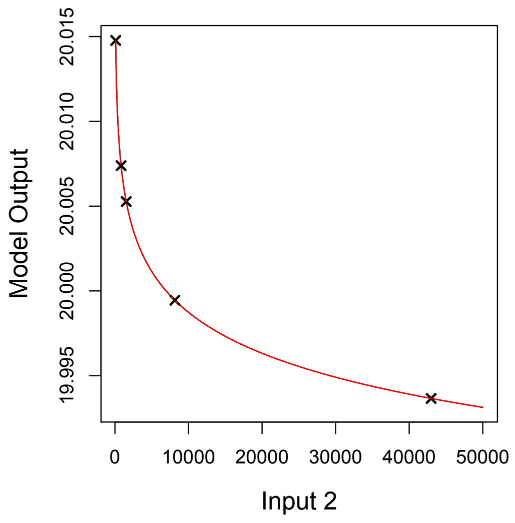
As described in Chapter 1, the C-GOLDSTEIN software encodes a computationally fast Earth System Model (ESM) developed by R. Marsh and N. R. Edwards. In addition to C-GOLDSTEIN, there also exists the adjoint of the model, described in Chapter 2, which was written by D. Zachary and N. R. Edwards. The



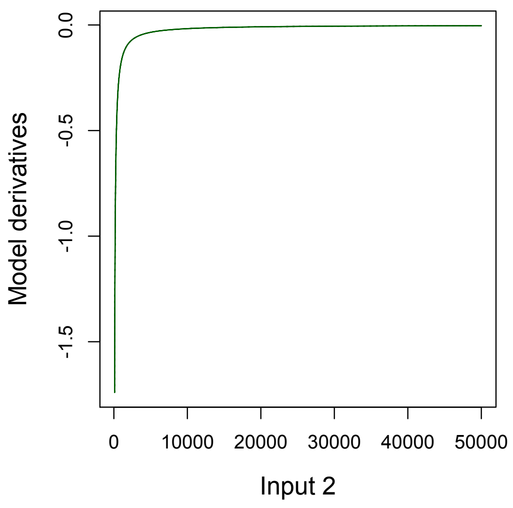
(a) Emulating the model response after a log transformation has been applied to input 2.



(b) Emulating the derivatives w.r.t to the log of input 2.



(c) Emulating the model response with the log of input 2 and then transforming back to meaningful scale.



(d) Emulating the derivatives w.r.t to the log of input 2 and then transforming back to obtain the derivatives w.r.t input 2.

Figure 5.13: Emulating the borehole model and its derivatives in one dimension. Input 2 is transformed and varied while the remaining 7 inputs are fixed at their lower limit.

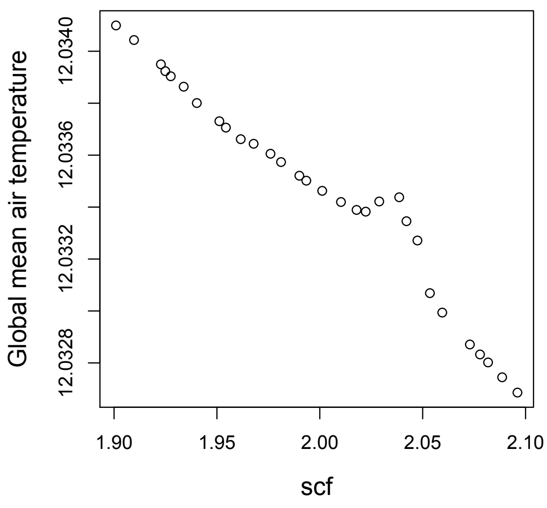
adjoint employs Automatic Differentiation to produce partial derivatives with respect to the inputs, in addition to the standard model output.

The C-GOLDSTEIN model produces many outputs and we choose global mean air temperature to study here. The adjoint model has been appropriately adapted to generate the derivatives of global mean air temperature with respect to 12 of the input parameters. For the purpose of this example we choose to vary just one of the 12 input parameters, scf, which is the wind stress scale. While thought to have some effect on global mean air temperature this input parameter is chosen primarily due to the performance of the adjoint. All finite difference (FD) experiments undertaken in an attempt to validate the adjoint, and shown in Chapter 2, have shown consistently good agreement between the adjoint and FD for this parameter. In an initial investigation, therefore this parameter is thought to be suitable.

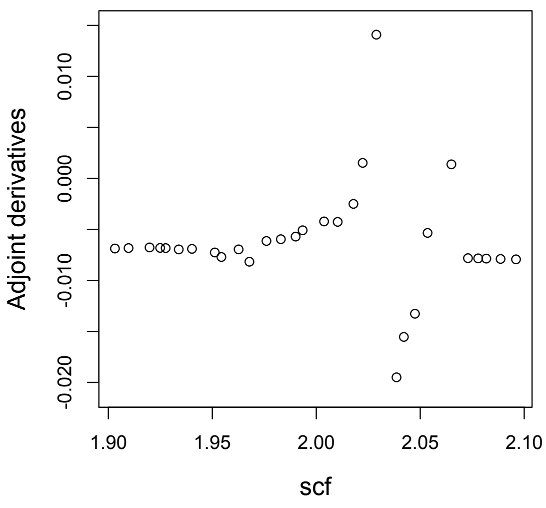
We build all the emulators in these experiments as in Section 5.2.1 with function output only and with a linear form for the prior mean. The Gaussian form for the covariance function is again adopted and validation data is acquired by running the adjoint model.

In the one-dimensional toy example of Section 5.3.1 we see that 10 simulator runs, without derivatives in the training data, are required for good emulation of the derivatives. As a starting point in this one-dimensional C-GOLDSTEIN example, therefore, we choose to run the C-GOLDSTEIN adjoint at 10 points to provide training data for the emulator, and a further 20 points to provide validation data. We only include the function output in the training data so this provides 30 validation derivatives in total. Ideally more validation derivatives would be available, however $n' = 30$ is a large enough sample to make initial judgements about the performance of our emulator and taking into consideration the computational expense of the adjoint, this seemed reasonable. Figures 5.14a and 5.14b show the how the global mean air temperature and the partial derivatives of the temperature with respect to wind stress scale vary for all 30 runs.

We begin by emulating the air temperature at the validation points. Figure 5.15a shows the performance of the emulator (in red) and the points included in the training data are in bold. We can see that the emulator performs well across most of the design space. We therefore proceed by emulating the derivatives at all 30 points and Figure 5.15b compares the emulator with the adjoint output. There



(a) Global mean air temperature



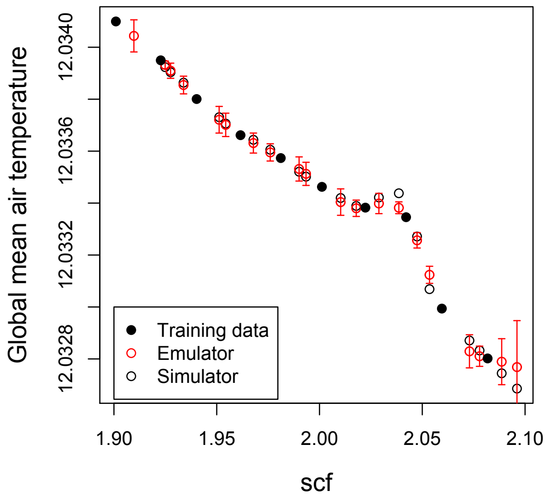
(b) Partial derivatives

Figure 5.14: C-GOLDSTEIN adjoint output at 30 points.

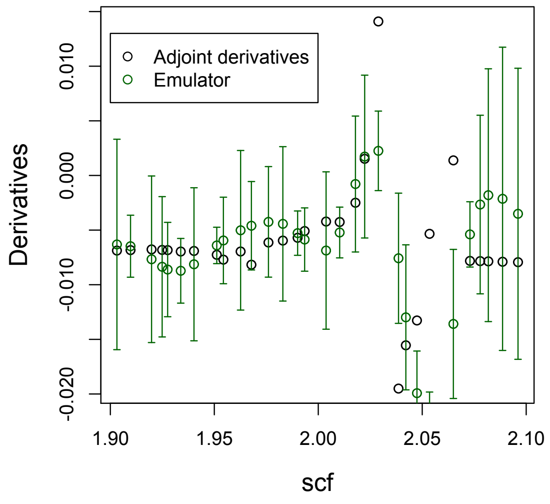
is a small area of the input space where the emulator mean matches the adjoint derivatives quite well. The emulator performs badly overall though, particularly for scf $\in \{2.02, 2.10\}$.

We attempt to improve on this by performing an additional 50 runs of C-GOLDSTEIN across the whole input space, in order to gain a better understanding of the function and provide more validation data for the standard emulator. We then restrict the training data to 11 points between 2.02 and 2.10 and build a second emulator for function output and derivatives. Figure 5.16 compares the air temperature of the emulator with the validation data. The black line is C-GOLDSTEIN output and the red line is the posterior mean evaluated at the validation points. The output is shown by a solid line but consists of 36 points. With this training data we now see the emulator is performing very well across this narrow input region.

No further runs of the adjoint are performed due to the computational expense. Figure 5.17a shows where the validation data for the derivatives is located in the input space and Figure 5.17b shows the performance of the emulator here. We also include estimates of the relevant derivatives using the finite differences approach with $\epsilon = 1e-5$. Figure 5.17b shows much less conflict between the em-



(a) Emulating function output



(b) Emulating derivatives

Figure 5.15: Emulating with the model response at $n = 10$ points.

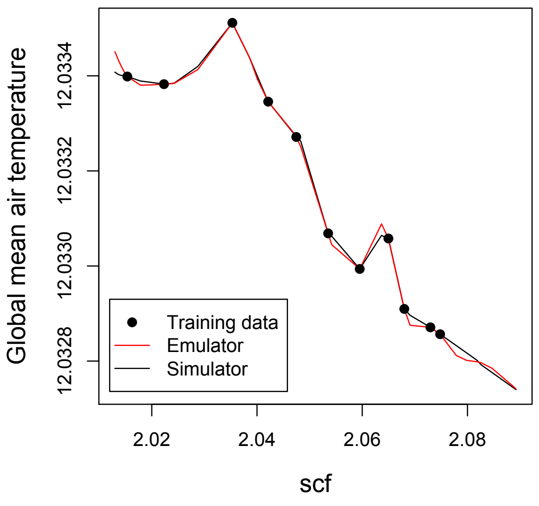
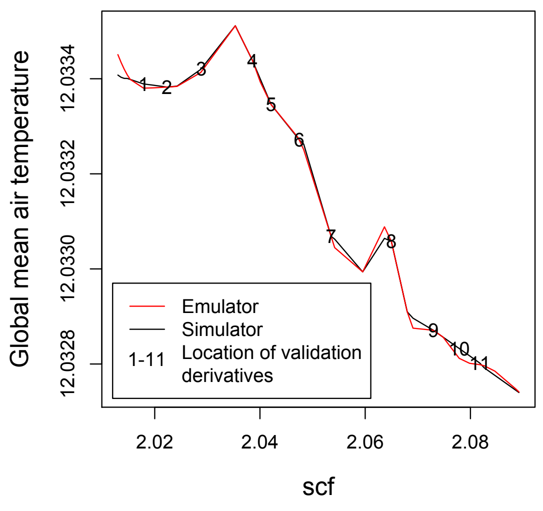
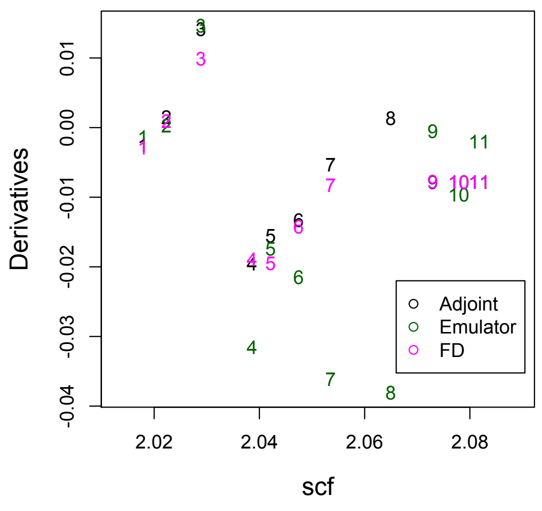


Figure 5.16: Emulating air temperature in the narrowed input space with the model response at $n = 11$ points. The true simulator output, shown here as a solid black line, consists of 36 points.

ulator and the adjoint than before (in Figure 5.15b), the emulator even matches the adjoint at the high peak at derivative point 3. The emulator however, still performs very badly at derivative points 7 and 8. There is not a clear explanation for this as Figure 5.17a shows we can emulate the function output well at these points. The finite differences estimate at point 7 is much closer to the adjoint value than the emulator, but at point 8 is so large (0.36) that it is not visible with the current scale on the y-axis.



(a) Emulating function output

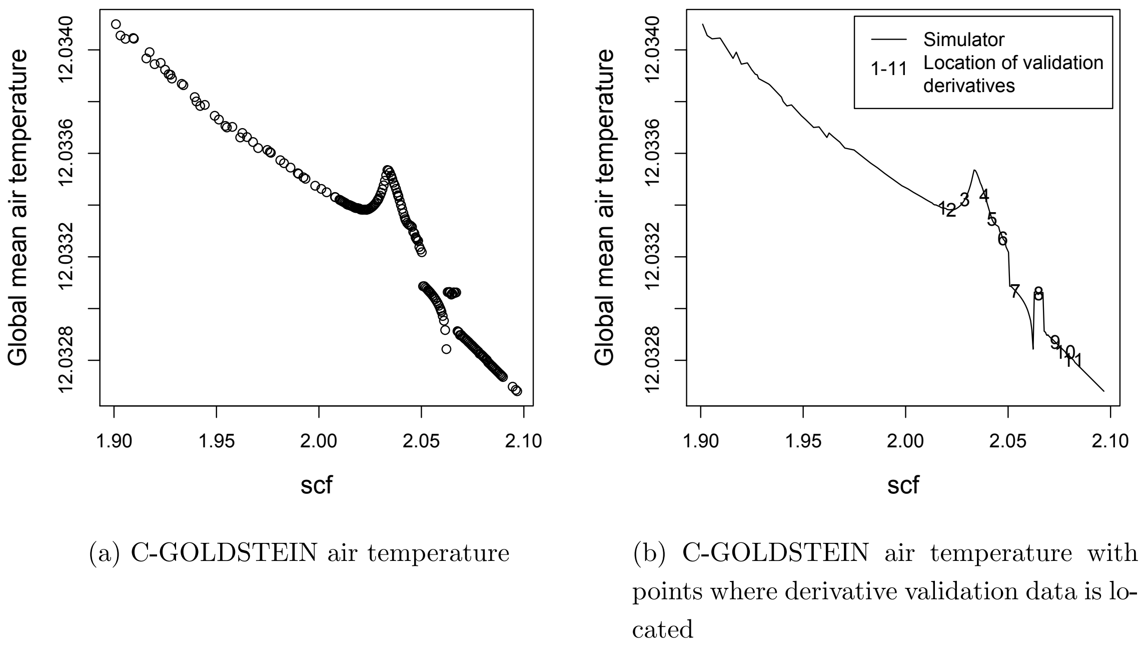


(b) Emulating the derivatives

Figure 5.17: Emulating in the narrowed input space with the model response at $n = 11$ points. For location of the training data see Figure 5.16.

We now choose to run C-GOLDSTEIN intensively in this narrowed region. The resulting data set has a further 134 points in the narrowed region and 214 points altogether. The augmented data set, shown in Figure 5.18a, is quite alarming as we had previously believed C-GOLDSTEIN to be a smooth, continuous function. Figure 5.18b shows the points where the derivative validation data is located and it is now clear why in Figure 5.17b, the emulator behaved poorly at point 8 and to a lesser extent at point 7.

The data in Figure 5.18a can be explained by examining the difference in surface air temperature between neighbouring discontinuous points. This is shown for the last apparent discontinuity at scf = 2.067 in Figure 5.19. The difference



(a) C-GOLDSTEIN air temperature

(b) C-GOLDSTEIN air temperature with points where derivative validation data is located

Figure 5.18: Augmented data set which consists of a total of 214 points.

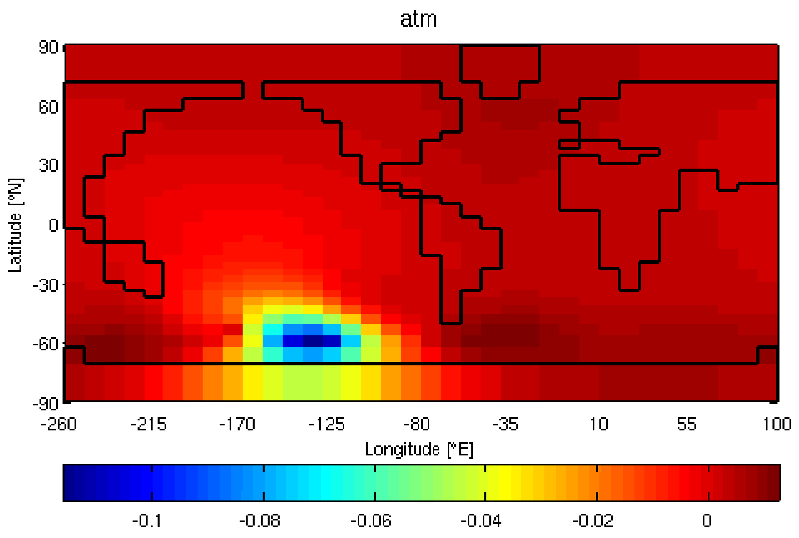


Figure 5.19: The difference in surface air temperature between the 2 points at the last "discontinuity" in Figure 5.18.

in surface air temperature between the points scf = 2.067 and scf = 2.0676 is pretty much zero everywhere apart from one patch in the southern ocean and this explains the relatively large difference in global mean air temperature. The most likely cause for this is changes in ocean convection: when a threshold is reached there is a switch in the model which can cause small jumps as seen in Figure 5.18a. The magnitude of the differences in global mean air temperature is very small, the largest in Figure 5.18a being 2.2e-04°C at scf = 2.0622. For comparison, the difference in global mean air temperature between scf = 2.0124 and scf = 2.013, a smooth part of the model, is 2.5e-06°C. While even the largest difference in temperature at a discontinuous point is too small to have an effect on the climate, the corresponding derivatives around these points are very large in magnitude, and don't exist at the exact point of discontinuity. Building an emulator capable of accurately predicting derivatives at such points, so large in magnitude relative to most of the derivatives in the input space, becomes an impossible task. As we see from Figures 5.15b and 5.17b, the emulator smooths out the derivatives in these areas, the result being estimates of derivatives, which are more appropriate at first glance. That is, only when we 'zoom in' and perform many model runs do we see the explanation for the values of the adjoint-generated derivatives. It could be argued that as the real world system is not expected to fluctuate with such sharp changes of gradient, the emulator is actually providing estimates which are closer to reality than the simulator and adjoint! The difference in global mean air temperature is too small to be of any consequence to climate scientists however, and therefore the exact derivative at and around the points of discontinuity may not be required. A smoothed over emulator result, on the other hand, could be appropriate for subsequent analysis. This is discussed further in Section 5.5.

5.4 Integrals and higher derivatives

The general case which applies to the emulation of both model response and first derivatives, with or without first derivative information in the training data, is presented in Section 5.2.2. This methodology can be extended if second or third derivatives are available, or if we wanted to emulate higher derivatives. The key concept is that derivatives of Gaussian processes remain Gaussian processes with mean and covariance functions given by the relevant derivatives of the original functions of the Gaussian processes. Consider our prior distribution for the first derivatives of $\eta(\cdot)$, (5.25) with $d \neq 0$. If we wanted to emulate the second derivatives of $\eta(\cdot)$ then (5.25) becomes

$$\frac{\partial^2}{\partial x^2}\eta(\mathbf{x})|\boldsymbol{\beta}, \sigma^2, \Theta \sim GP\left(\frac{\partial^2}{\partial x^2}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}, \sigma^2\frac{\partial^4}{\partial x_i^2 \partial x_j^2}c(\mathbf{x}_i, \mathbf{x}_j)\right). \tag{5.40}$$

Clearly this requires a 4 times differentiable covariance function though and a twice differentiable mean function. In general higher derivatives of $\eta(\cdot)$ can be modelled by Gaussian processes with mean:

$$\mathrm{E}\left\{\frac{\partial^u}{\partial x^u}\eta(\mathbf{x})\right\} = \frac{\partial^u}{\partial x^u}\mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}$$

and covariance between two higher derivatives given by:

$$\mathrm{Cov}\left\{\frac{\partial^u}{\partial x_i^u}\eta(\mathbf{x}), \frac{\partial^v}{\partial x_j^v}\eta(\mathbf{x}_j)\right\} = \frac{\partial^{u+v}}{\partial x_i^u \partial x_j^v}c(\mathbf{x}_i, \mathbf{x}_j),$$

for $u, v \in \mathbb{Z}$ and assuming that $\mathbf{h}(\cdot)$ is u times differentiable and $c(\cdot)$ is $(u+v)$ times differentiable. The posterior process is then derived similarly to Section 5.2.2.

The methodology can also be adapted to emulate integrals of functions, if this were required by a model user. This is akin to the special case of emulating the model response when we only have derivatives in the training data; such an emulator could be built using the setup and methods described in Section 5.2.2. Suppose now we are interested in the integral $I = \int_\chi \eta(\mathbf{x})d\mathbf{x}$ where χ represents the input space. O'Hagan (1991) uses a Gaussian process to make inferences about I in Bayes-Hermite quadrature. The Gaussian process emulator methodology applied to I with just model response in the training data results in:

$$I|\mathbf{y} \sim GP\left(\int_\chi m^{**}(\mathbf{x})d\mathbf{x}, \sigma^2\int_\chi\int_\chi c^{**}(\mathbf{x}_i, \mathbf{x}_j)d\mathbf{x}_i d\mathbf{x}_j\right).$$

Similarly this could be extended to include derivatives in the training data and Oakley (1999) briefly looks at the effect of observing derivatives in quadrature.

5.5 Conclusions

Adjoint models are becoming increasingly common and valuable but for complex models they remain difficult to write and expensive in computing resource and time to run. In this chapter we have adapted existing methodology to emulate model derivatives and demonstrated the methodology in 3 examples.

The one dimensional toy example, in Section 5.3.1, shows the potential the Gaussian process emulator approach has: with derivative information in our emulator we can emulate the model derivatives very well. In addition to this, given just a few extra simulator runs we can accurately emulate the model derivatives with function output alone. The design problem for building an emulator of derivatives is briefly investigated with the one dimensional toy model and we find that an optimal space-filling design in some cases, produces invalid emulators. This is because all the points are exactly the same distance apart, making it very difficult to estimate the smoothness parameter in the emulator. This may not be of serious concern with real complex models though, as in large dimensional space we are very unlikely to get multiple points with exactly the same distances between them. If we do have enough training data, in higher dimensional input space, such that this happens we would likely encounter numerical problems when building the emulator. Even if the emulator is built without any such hindrance though, the problem of the resulting poor performance of the emulator can likely be resolved by removing a small number of the observations, as we saw in Figure 5.6a. The effect on the emulation of derivatives when we have equidistance points does, however, motivate the need to investigate designs with varying distances between points to better estimate smoothness parameters. In support of this, we see the uncertainty reducing between design points which also implies perhaps space-filling isn't necessarily optimal when emulating derivatives.

The second example, in Section 5.3.2, shows that in an 8-dimensional model we can emulate very well the derivatives with respect to 5 of the inputs. The performance of the emulator is not as good with respect to inputs 2, 3 and 5 but the emulator is appropriately confident about these predictions. In a one-dimensional setting, individual emulators for each of these inputs perform well, although with a real complex model such an experiment would unlikely be possible. The model response, however, is much less sensitive to inputs 2, 3 and 5 which possibly makes it less important that these derivatives are accurately emulated. The derivatives w.r.t these inputs are relatively very small in magnitude and thus the information they provide is unlikely to have a large effect on any further analysis.

The final example, the emulation of the partial derivatives of C-GOLDSTEIN, has had mixed success. Firstly, perhaps it should be noted that the function of the derivatives, as shown in Figure 5.14b, is not a function we would confidently claim to be able to accurately emulate. We require that the simulator is a smooth function of its inputs and therefore also that the partial derivatives are a smooth function of the inputs. This is a key requirement as it accounts for the efficiency of an emulator. As discussed in Chapter 3, if a model is smooth then knowledge of the output at $\mathbf{x}_i$ tells us something about the output at $\mathbf{x}_j$ for $\mathbf{x}_i$ close to $\mathbf{x}_j$. Monte Carlo methods do not make use of this extra information and as such the emulator approach is more efficient. If there are areas of the input space where a model does not respond smoothly to changes in the inputs, i.e the model is non-stationary, standard emulation is unlikely to be appropriate. There are options that could be investigated in this situation though. For example, Gramacy and Lee (2008) split the output into regions which have a similar level of smoothness. Other options include 'warping' the input space such that we have a stationary function on this transformed space, as demonstrated in Section 5.3.2, or another option is to select a non-stationary covariance function. In our C-GOLDSTEIN example, we split the output into regions, and emulate the derivatives of the 'rough' patch separately. With this method and given enough training data, we can emulate the derivatives of this model quite well. This is relatively straightforward in one-dimension as even with limited observations of the function, it is clear which areas we need to perform further runs, to 'zoom in'. In higher dimensional space though, identifying the location and cause of such 'rough' regions is much harder. Even if this can be done, as in the example illustrated in Section 5.3.3, many simulator runs will be required, and it is unlikely that emulation will be an efficient alternative to an adjoint model. Of course, an adjoint to the required model might not exist, so the question becomes about whether emulation is more efficient than the many runs a finite differences experiment would entail.

The difference between FD and adjoint estimated derivatives, as shown in Figure 5.17b, is in contrast with the validation results in Chapter 2, where good agreement is apparent. It is clear, however, from Figure 5.18, that the adjoint derivatives are accurate here and it is likely that if FD runs are performed with a more appropriate value of ϵ, the resulting derivatives would be in closer agreement to the adjoint. This highlights the importance and difficulty in selecting a suitable value of ϵ when undertaking FD experiments.

We've just described how the derivatives from the adjoint model, at and around the points of discontinuity, appear to be accurate and if the purpose is to investigate whether the model 'misbehaves' at any inputs points, then relatively large derivative values are very informative of this. This is assuming of course that the model is expected to be smooth. It could be argued though that the adjoint-generated derivatives are actually misleading for other types of analysis. Suppose, for example, we wish to run some optimisation algorithm which makes use of gradient information. Inputing partial derivatives which, while accurate at a precise level are unrepresentative of the trend, seen for example at point 8 in Figure 5.18b, will cause problems for the optimisation algorithm. For example, the algorithm will be less efficient as it will necessarily spend time searching in the wrong part of the input space. In addition to this there is a greater chance that a local maximum or minimum is returned, rather than the global optimum which we are interested in. In this situation, emulated derivatives could actually provide more meaningful estimates of the gradient which would in turn assist the optimisation algorithm.

In summary, we have suggested an alternative approach to the efficient evaluation of model derivatives. While this is encouraging, validation derivatives will still be required and if an adjoint model does not exist or is too expensive to execute, then other techniques to produce validation derivatives will have to be employed. Derivatives generated by the finite differences approach could be an option, though this requires multiple simulator runs and numerical and approximation errors can cause inaccurate derivatives. This complicates the validation process as we need to be confident that validation derivatives are accurate enough to be used as a diagnostic: it is possible that conflict between emulated derivatives and those generated by finite differences may be in part due to an inappropriate choice of ϵ.

Chapter 6

Use of derivatives in multivariate emulation

6.1 Introduction

Complex computer models are an important tool for studying a wide range of systems and such models tend to have many outputs. The standard approach to emulating multiple outputs of a complex model is to build an independent emulator for each output; such an approach, however, ignores any possible correlation between outputs. There are various approaches to emulating multivariate response; for example, Conti and O'Hagan (2010) emulate multiple outputs of a complex model with the use of a separable covariance structure, that is treating covariances between model inputs and model outputs separately. Urban and Fricker (2009) adopt a similar framework in the multivariate emulation of a climate model. One of the difficulties in emulating multivariate response is handling large datasets; we have n model runs and r model outputs which results in an array of size $n \times r$. For large r, one approach is to use principle component analysis to reduce the dimension of the output space. Alternatively, Rougier (2008) introduces the outer-product emulator of which the Conti and O'Hagan (2010) approach is a special case. The outer-product emulator provides computational efficiency by utilising a separable structure in the mean function as well as in the covariance function. The outer-product emulator is demonstrated further in Rougier *et al.* (2009).

As discussed in Chapter 2, it is possible to obtain derivatives of model outputs with respect to model inputs, for example through the adjoint of the model. The use of derivative information when building univariate emulators is investigated in Chapter 4 and here we explore the value of derivatives in multivariate emulation. We adapt the separable covariance approach of Conti and O'Hagan (2010) to allow for the inclusion of derivative information and the resulting methodology is given in Section 6.2. We then demonstrate this method with the use of the radiation transport model, which is the PARTISN model for estimation of radiation measurements from specified material configurations. The model produces 5 outputs and it is reasonable to expect correlation between these outputs. See Section 1.3.2 for further detail about the radiation transport model. We continue in this chapter by first building independent emulators, with and without derivatives, for each of the 5 outputs in order to compare with the multivariate approach. A multivariate emulator is then built, again with and without derivatives, in Section 6.3.3 and results showing the value of including the derivative information, in both univariate and multivariate situations are given in Section 6.3.4. In this way we attempt to answer the following question specifically for the radiation transport model: *should we build 5 independent emulators or one multi-output emulator and, given the type of emulator, would it be more efficient to include derivative information?* We then conclude this chapter with a small calibration study using one of the emulators built in Section 6.3.

6.2 Multivariate emulation with separable covariance and derivatives

To build a multivariate emulator with derivative information we follow the framework of Conti and O'Hagan (2010) for the multivariate aspect and Morris *et al.* (1993) for the inclusion of derivatives. As in Chapter 4 there are $\mathbf{x}$ input configurations and p input dimensions. We are now interested in emulating r multiple outputs of the simulator.

As with univariate emulation, we model $\eta(\cdot)$ with a Gaussian process, though as we are interested in the r outputs of $\eta(\cdot)$, we require an r-variate Gaussian process. We now need to specify mean and covariance functions and begin by defining the mean function to be $E[\eta(\mathbf{x})|\, B] = \mathbf{h}(\mathbf{x})^T B$, where $\mathbf{h}(\mathbf{x})^T$ is a $1 \times q$ vector of known, differentiable functions of $\mathbf{x}$ and B is a $q \times r$ matrix of unknown coefficients, β. Each output therefore shares the function $\mathbf{h}(\mathbf{x})^T$ but has its own $(\beta_1, \ldots, \beta_q)$. As we are including derivative information in the training data we must ensure $\mathbf{h}(\cdot)$ is differentiable. This will then lead to the derivative of the mean function: $E\left[\frac{\partial}{\partial x^{(d)}}\eta(\mathbf{x}) \middle| B\right] = \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T B$, where $x^{(d)}$ refers to input d at point $\mathbf{x}$. We bring these functions together in $\tilde{\mathbf{h}}(\mathbf{x}, d)$, which is defined as:

$$\tilde{\mathbf{h}}(\mathbf{x}, d)^T B = \begin{cases} \mathbf{h}(\mathbf{x})^T B & \text{for } d = 0 \\ \frac{\partial}{\partial x^{(d)}}\mathbf{h}(\mathbf{x})^T B & \text{for } d \neq 0 \,. \end{cases}$$

The value of d determines whether a partial derivative is to be evaluated at that point or the model response, and can therefore take any value in $\{0, 1, \ldots, p\}$. For example, $\{(\mathbf{x}_1, d = 0), (\mathbf{x}_2, d = 3)\}$ refers to the model response at location $\mathbf{x}_1$ and the partial derivative w.r.t the third input at location $\mathbf{x}_2$.

As in Chapter 4 a correlation function, $c(\mathbf{x}_i, \mathbf{x}_j)$, which is twice differentiable is required since we require the correlations between points and derivatives, and between derivatives themselves. The correlation then between a point, $\mathbf{x}_i$ and a derivative w.r.t input k at $\mathbf{x}_j$, (denoted by $x_j^{(k)}$) is $\frac{\partial}{\partial x_j^{(k)}}c(\mathbf{x}_i, \mathbf{x}_j)$. The correlation between a derivative w.r.t input k at $\mathbf{x}_i$, (denoted by $x_i^{(k)}$), and a derivative w.r.t input l at $\mathbf{x}_j$, (denoted by $x_j^{(l)}$), is $\frac{\partial^2}{\partial x_i^{(k)} \partial x_j^{(l)}}c(\mathbf{x}_i, \mathbf{x}_j)$. These are combined in $\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\}$:

$$\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} = \begin{cases} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = d_j = 0 \\ \frac{\partial}{\partial x_i^{(d_i)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = 0 \text{ and } d_j \neq 0 \\ \frac{\partial^2}{\partial x_i^{(d_i)} \partial x_j^{(d_j)}} c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i, d_j \neq 0 \,, \end{cases}$$

Note that a superscript on x refers to the input dimension and a subscript on x refers to the location of that point in the design space.

A common choice of correlation function, and one we regularly choose in this thesis, is the Gaussian form: $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta \, (\mathbf{x}_i - \mathbf{x}_j)\}$, where the diagonal matrix, Θ, consists of positive smoothness parameters, θ as in Section 4.2.1. The correlations between points and derivatives, and between derivatives themselves are given for this function as:
$\tilde{c}\{(\mathbf{x}_i, d_i), (\mathbf{x}_j, d_j)\} =$

$$\begin{cases} 2\,\theta\{k\} \left(x_i^{(k)} - x_j^{(k)}\right) c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = 0 \text{ and } d_j = k \\ \left(2\,\theta\{k\} - 4\,\theta^2\{k\} \left(x_i^{(k)} - x_j^{(k)}\right)^2\right) c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = d_j = k \\ 4\,\theta\{k\}\,\theta\{l\} \left(x_j^{(k)} - x_i^{(k)}\right)\left(x_i^{(l)} - x_j^{(l)}\right) c(\mathbf{x}_i, \mathbf{x}_j) & \text{for } d_i = k, d_j = l \end{cases}$$

where k, l could take any value in $\{1, \ldots, p\}$. Covariances between outputs are given by the $r \times r$ matrix, Σ. We then combine $\tilde{c}(.,.)$ and Σ resulting in a separable covariance structure.

As with univariate emulation we have a design, $\tilde{D} = \{(\mathbf{x}_k, d_k)\}$, where $k = \{1, \tilde{n}\}$ and $d_k \in \{0, 1, \ldots, p\}$. The location in the design is given by $\mathbf{x}_k$ and d_k determines whether at point $\mathbf{x}_k$, a first partial derivative or function output is required. Running the simulator, $\eta(\cdot)$, or the adjoint of the simulator, $\tilde{\eta}(.)$, at each of the input configurations in $\tilde{D}$ results in the training data, $\tilde{Y}$, a $\tilde{n} \times r$ matrix. Since we have represented $\eta(\cdot)$ by an r-variate Gaussian process then $\tilde{Y}$ conditional on B and Σ follows the matrix-Normal distribution:

$$\tilde{Y} | \, B, \Sigma, \theta \sim N_{\tilde{n} \times r}(\tilde{H}B, \Sigma, \tilde{A}), \tag{6.1}$$

where $\tilde{H} = [\tilde{h}(\mathbf{x}_1, d_1), \ldots, \tilde{h}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})]^{\mathrm{T}}$, $\tilde{A}$ is the $\tilde{n} \times \tilde{n}$ matrix of correlations between points, between derivatives and points and between derivatives themselves: $\tilde{A} = \tilde{c}(\tilde{D}, \tilde{D})$ and Σ is the $r \times r$ matrix of covariances between outputs. The exact form of $\tilde{c}(\cdot, \cdot)$ depends on where derivatives are included.

As in the univariate case, standard normal theory shows that:

$$\tilde{\eta}(\cdot) | \, B, S, \tilde{Y} \sim GP_r(\tilde{m}^*(.), \tilde{c}^*(.,.)\Sigma), \tag{6.2}$$

where

$$\begin{aligned}
\tilde{m}^*(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T B + \tilde{\mathbf{t}}(\mathbf{x})^T \tilde{A}^{-1}(\tilde{Y} - \tilde{H}B), && (6.3)\\
\tilde{c}^*(\mathbf{x}_i, \mathbf{x}_j) &= c(\mathbf{x}_i, \mathbf{x}_j) - \tilde{\mathbf{t}}(\mathbf{x}_i)^T \tilde{A}^{-1} \tilde{\mathbf{t}}(\mathbf{x}_j), && (6.4)\\
\tilde{\mathbf{t}}(\mathbf{x})^T &= \tilde{c}\{\tilde{D}, (\mathbf{x}, 0)\} \\
&= [\tilde{c}\{(\mathbf{x}, 0), (\mathbf{x}_1, d_1)\}, \ldots, \tilde{c}\{(\mathbf{x}, 0), (\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})] \,, && (6.5)\\
\tilde{Y}^T &= \tilde{\eta}(\tilde{D}) = \{\tilde{\eta}(\mathbf{x}_1, d_1), \ldots, \tilde{\eta}(\mathbf{x}_{\tilde{n}}, d_{\tilde{n}})\}. && (6.6)
\end{aligned}$$

Note that in equation (6.3) we have $\mathbf{h}(\mathbf{x})^T \hat{B}$ and not $\tilde{\mathbf{h}}(\mathbf{x}, d)^T \hat{B}$. This is because here we are emulating the multiple output response and not the derivatives; we therefore require $d = 0$ and $\tilde{\mathbf{h}}(\mathbf{x}, 0)^T \hat{B} = \mathbf{h}(\mathbf{x})^T \hat{B}$. Similarly, in equation (6.5) we have $\tilde{\mathbf{t}}(\mathbf{x}, 0)^T = \tilde{\mathbf{t}}(\mathbf{x})^T$ as we are emulating model response and require $d = 0$. The tilde symbol, $(\tilde{\ })$, is still required as $\tilde{\mathbf{t}}(.)$ includes the correlations between the derivatives in the training data and the point we are predicting the model response at.

We now want to obtain the distribution of $\eta(\cdot)|\tilde{Y}$ unconditional on B and Σ. As in the univariate case we specify a weak prior for both B and S:

$$p(B,\Sigma) \propto |\Sigma|^{-\frac{r+1}{2}} \tag{6.7}$$

Applying Bayes Theorem with (6.7) and (6.1) results in the following posterior distribution for (B,Σ):

$$\begin{aligned} f(B,\Sigma|\tilde{Y},\theta) \quad \propto \quad & |A|^{-\frac{r}{2}}|\Sigma|^{-\frac{\tilde{n}+r+1}{2}} \times \\ & \exp\left\{-\frac{1}{2}\mathrm{Tr}[\tilde{Y}^T G\tilde{Y}\Sigma^{-1} + (B-\hat{B})^T\tilde{H}^T\tilde{A}^{-1}\tilde{H}(B-\hat{B})\Sigma^{-1}]\right\}, \end{aligned} \tag{6.8}$$

where

$$\begin{aligned} \hat{B} &= (\tilde{H}^T\tilde{A}^{-1}\tilde{H})^{-1}\tilde{H}^T\tilde{A}^{-1}\tilde{Y}, & (6.9)\\ G &= \tilde{A}^{-1} - \tilde{A}^{-1}\tilde{H}(\tilde{H}^T\tilde{A}^{-1}\tilde{H})^{-1}\tilde{H}^T\tilde{A}^{-1}. & (6.10) \end{aligned}$$

From this we see that:

$$B|\Sigma,\tilde{Y} \sim N_{q\times r}\left(\hat{B},\Sigma,(\tilde{H}^T\tilde{A}^{-1}\tilde{H})^{-1}\right). \tag{6.11}$$

Now if we take the product of (6.2) and (6.11) and then integrate out B, it results in:

$$\tilde{\eta}(\cdot)|\Sigma,\theta,\tilde{Y} \sim GP_r(\tilde{m}^{**}(\cdot),\tilde{c}^{**}(\cdot,\cdot)\Sigma), \tag{6.12}$$

where

$$\begin{aligned} \tilde{m}^{**}(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T\hat{B} + \tilde{\mathbf{t}}(\mathbf{x})^T\tilde{A}^{-1}(\tilde{Y}-\tilde{H}\hat{B}), & (6.13)\\ \tilde{c}^{**}(\mathbf{x}_i,\mathbf{x}_j) &= c(\mathbf{x}_i,\mathbf{x}_j) - \tilde{\mathbf{t}}(\mathbf{x}_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{\mathbf{t}}(\mathbf{x}_j) + \\ &\quad \left(\mathbf{h}(\mathbf{x}_i)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\left(\mathbf{h}(\mathbf{x}_j)^{\mathrm{T}} - \tilde{\mathbf{t}}(\mathbf{x}_j)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{\mathrm{T}}. & (6.14) \end{aligned}$$

Finally we must integrate out Σ and we are left, conditional on θ, with a r-variate T process:

$$\tilde{\eta}(\mathbf{x})|\theta,\tilde{Y} \sim \tau(\tilde{m}^{**}(\mathbf{x}),\tilde{c}^{**}(\mathbf{x}),\hat{\Sigma};\tilde{n}-q), \tag{6.15}$$

where

$$\hat{\Sigma} = \frac{1}{\tilde{n}-q}(\tilde{Y}-\tilde{H}\hat{B})^T\tilde{A}^{-1}(\tilde{Y}-\tilde{H}\hat{B}). \tag{6.16}$$

The posterior mean function is $\tilde{m}^{**}$ and posterior covariance function is $\tilde{c}^{**}$. For detail of the derivation without derivative information see Conti and O'Hagan (2010).

We again cannot integrate out Θ from (6.15) so could instead estimate the parameters by maximum likelihood as we do in Chapter 4. Conti and O'Hagan (2010) give the likelihood function in the multi-output case and with the inclusion of derivative information this becomes:

$$f(\Theta|\tilde{Y}) \propto |\tilde{A}|^{-\frac{r}{2}}|\tilde{H}^T\tilde{A}^{-1}\tilde{H}|^{-\frac{r}{2}}|\tilde{Y}^T G\tilde{Y}|^{\frac{\tilde{n}-q}{2}},$$

with G as defined in (6.10). Other options and further detail about the handling of Θ are given in Chapter 3.

As in Chapter 4 we can now use $\tilde{m}^{**}(\mathbf{x})$ as a fast approximation to $\eta(\cdot)$ and uncertainty about the emulator is given by $\tilde{c}^{**}(\mathbf{x},\mathbf{x})\hat{\Sigma}$.

6.3 Emulating multiple outputs of the radiation transport model with derivatives

The radiation transport model, as described in Section 1.3.2 of Chapter 1, calculates the measured radiation signature of a gamma-ray-emitting and neutron-multiplying cylinder. There are 5 inputs and in Chapter 4 just output 5, the neutron multiplication factor, is investigated. Here we study all 5 outputs of the model as we believe a priori that there maybe correlations between the outputs.

The data available is that resulting from the 100 point Latin hypercube sample as in Chapter 4.

6.3.1 Exploratory data analysis

We illustrate all 5 outputs from the radiation transport model, run at the input configurations from the LHS, in Figure 6.1. In Chapter 4 we transformed output 5 from the neutron multiplication factor inverse, $\frac{1}{k_{eff}}$, to k_{eff} but include the original data representation in Figure 6.1 for completeness.

Due to the spread of the data shown in Figure 6.1, we apply log transformations to the outputs 1 - 4 (the gamma-line outputs). The transformed data is

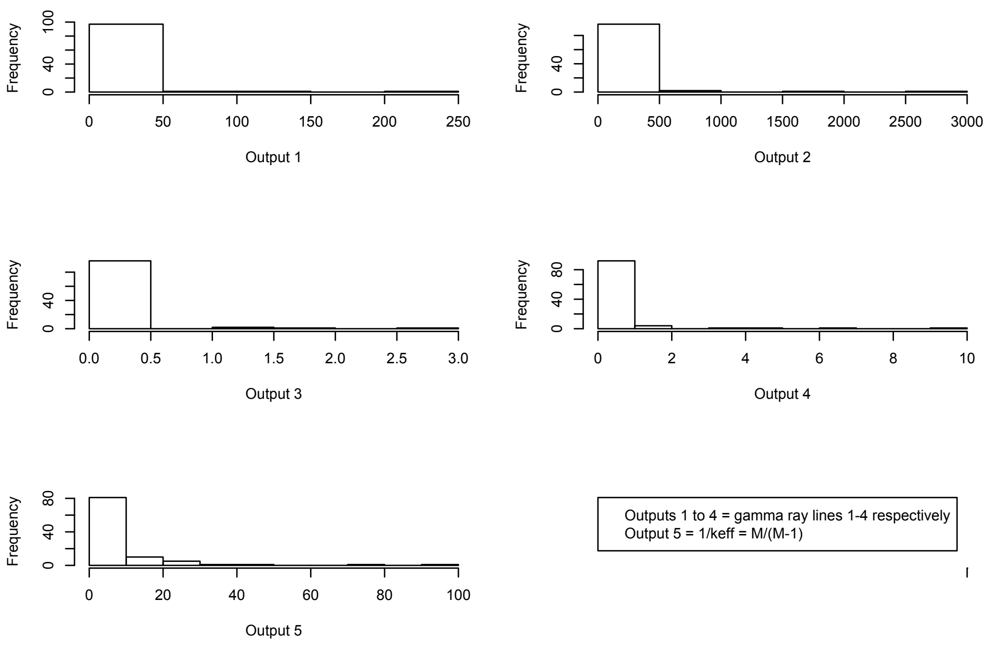


Figure 6.1: Histogram showing the output from the radiation transport model.

shown in Figure 6.2 and provides good justification to emulate the transformed responses rather than the native responses.

As with the derivatives of output 5, the partial derivatives of outputs 1 - 4 require transforming according to the chain rule: let $z_1 = log(y_1)$, then the required derivatives of the transformed output, z_1, w.r.t to the 5 inputs, $\mathbf{x}$, are given by:

$$\begin{aligned}\frac{\partial z_1}{\partial \mathbf{x}} &= \frac{\partial z_1}{\partial y_1} \times \frac{\partial y_1}{\partial \mathbf{x}} \\ &= \frac{1}{y_1} \times \frac{\partial y_1}{\partial \mathbf{x}}. \qquad (6.17)\end{aligned}$$

These transformations, repeated for outputs 2 - 4, are reasonable in the domain where the gamma-ray activity of the material is reasonably expected to be respond exponentially to the physical configuration.

Multivariate emulation may be more appropriate if there is correlation between outputs. Figure 6.3 shows 2-dimensional scatter plots of the outputs and a very strong relationship between the first 4 outputs is clear. There also appears to be some, albeit weaker, correlation between output 5 and the first 4. Multivariate emulation, therefore is an appropriate approach to test in this situation. In

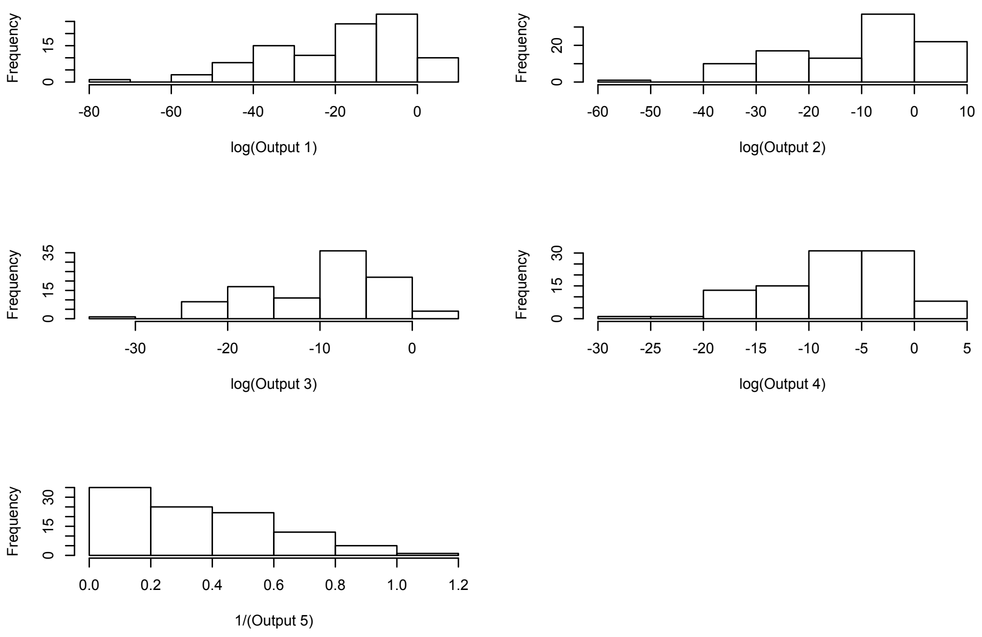


Figure 6.2: Histogram showing the transformed responses.

addition to this we see in Chapter 4 the high value of the derivative information in the independent emulation of output 5; it would be interesting therefore to see if derivatives are also required for more efficient emulation of the first 4 outputs independently, and also in a multivariate setting.

We begin, as when we emulate output 5 in Chapter 4, by building independent, univariate emulators for each output with the information from all 100 runs. We adopt a *leave one out* method to assess the emulators: the information from the first run is left out of the training data and an emulator built with the remaining 99 runs. We then use this emulator to predict the observation which was left out and this is repeated for all 100 observations. Comparisons between the values predicted by the emulators and the true values for output 1 are shown in Figure 6.4. The posterior mean of the emulator at each input configuration is used as the prediction and we plot 2 standard deviations above and below each mean value to assess the measure of uncertainty in the emulator. It is clear from Figure 6.4 that both the emulator with, and without, derivatives is performing very well. The emulators of outputs 2 - 4 perform similarly and the corresponding plots are therefore omitted here. The performances of the emulators of output 5 are

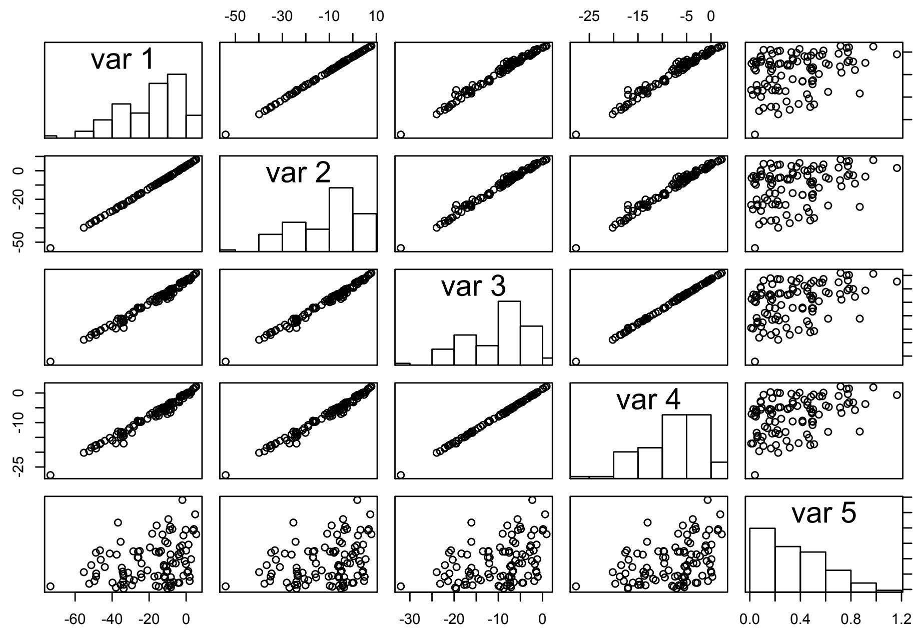
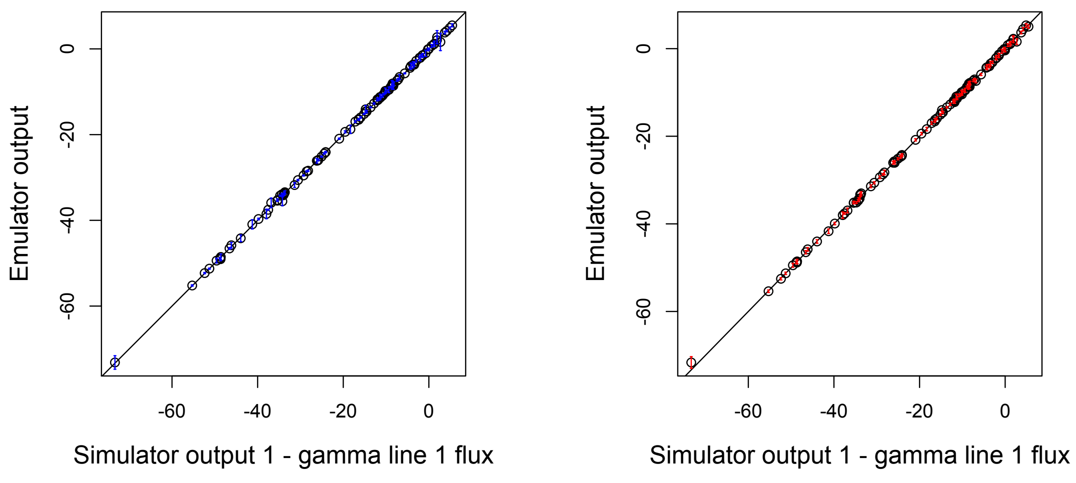


Figure 6.3: Scatterplot showing the correlation between the outputs of the model.

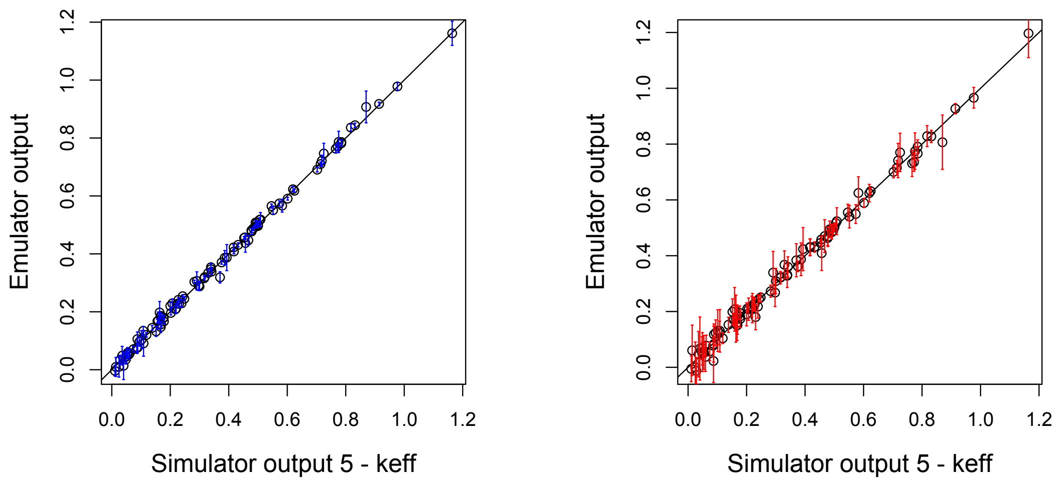


(a) With derivative information

(b) No derivative information

Figure 6.4: Leave one out results for the prediction of output 1.

illustrated in Chapter 4, (Figure 4.11) and repeated here for completeness. The



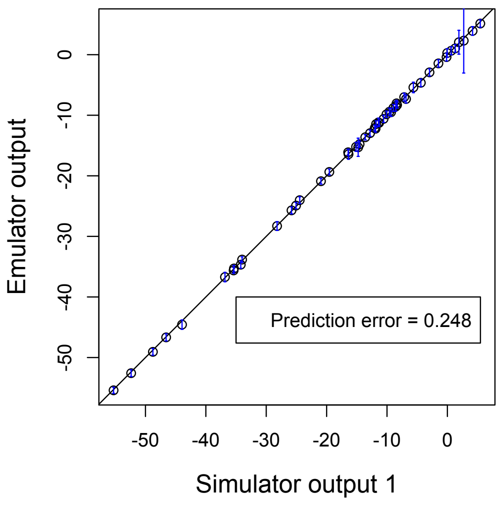
(a) With derivative information

(b) No derivative information

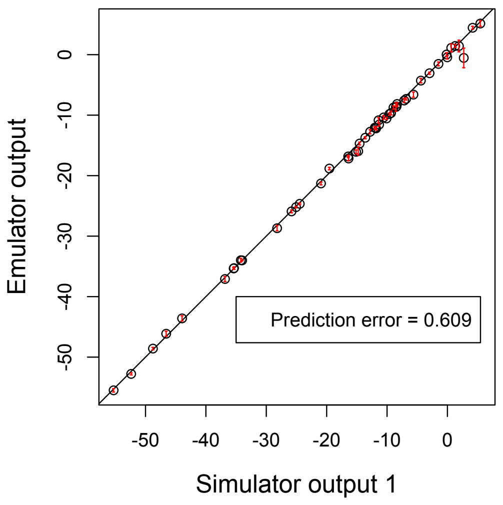
Figure 4.11: Leave one out results for the prediction of output 5.

emulator of output 5 with derivatives, Figure 4.11a, would appear to be slightly superior, with better predictions and less uncertainty, than the emulator of output 5 without derivatives, Figure 4.11b. The performances of both emulators though, while not quite as good as those of outputs 1 - 4, are still satisfactory.

Since all outputs have been shown to be well emulated we can afford to reduce the number of runs we use to build the emulators with, and therefore leave data to be used as validation. Emulators are built as before but now we follow the general rule of thumb as given by Loeppky *et al.* (2009), and use training data of size $n = 10p = 50$. We attempt to choose the $n = 50$ most space filling points from our 100 point LHS by adopting the maximin criteria: we wish to select 50 points from the 100 available such that the minimum distance between points is maximised. Calculating the minimum distance of every 50 point combination is not practical as there are $\binom{100}{50} \approx 1 \times 10^{29}$ combinations altogether. We therefore randomly select a $100,000$ member subset of 50 point combinations and calculate the minimum distance between the points of each of these combinations. The combination with the largest minimum distance is then selected as training data, allowing the remaining 50 input sites to be used for prediction. Performances of the emulators of outputs 1, 3 and 5 built with such training data and predicting at the remaining input sites are shown in Figures 6.5 to 6.7 respectively.



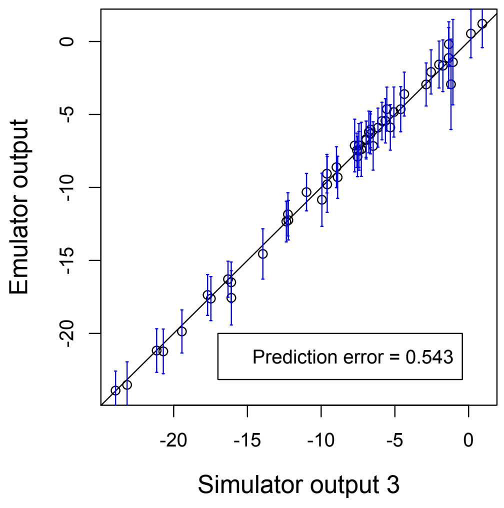
(a) With derivative information.



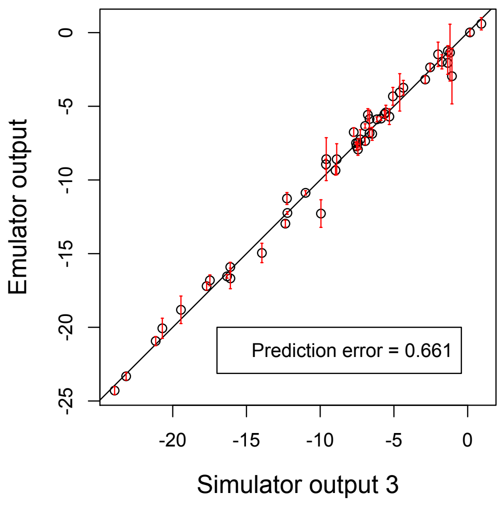
(b) No derivative information.

Figure 6.5: Diagnostics plot of the emulators of output 1 built with $n = 50$ training runs and predicting at $n' = 50$ points. The prediction error is the root mean squared error calculated as in equation (6.18) and discussed further in Section 6.3.2.

Both emulators of outputs 1 perform very well with $n = 50$ runs and the emulators of output 3 also perform well. Both emulators of outputs 2 and 4 perform similarly to those of outputs 1 and 3 respectively, and so the corresponding figures are omitted. The emulator without derivatives of output 5, Figure 6.7b, does not perform quite as well as the first 4 outputs with $n = 50$ runs. In Chapter 4 we see that to fully validate emulators of output 5, $n = 60$ runs are required. However, since the emulators for the first 4 outputs perform very well and we have some knowledge of the emulation of output 5 from Chapter 4, for the purposes of a comparison between independent and multi-output emulators $n = 50$ runs is an acceptable amount of training data. This leaves 50 runs that we can continue to hold back to compare the predictive performances of the various emulators.

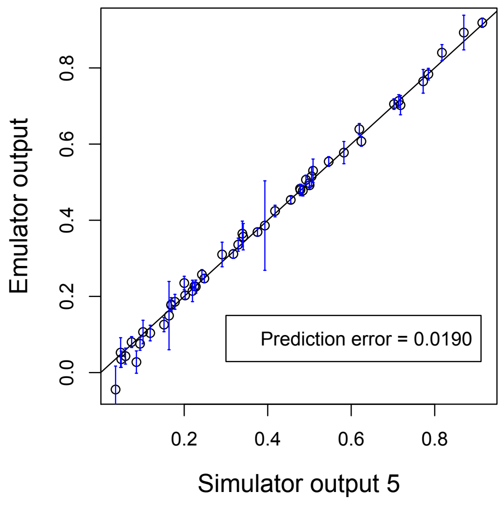


(a) With derivative information.

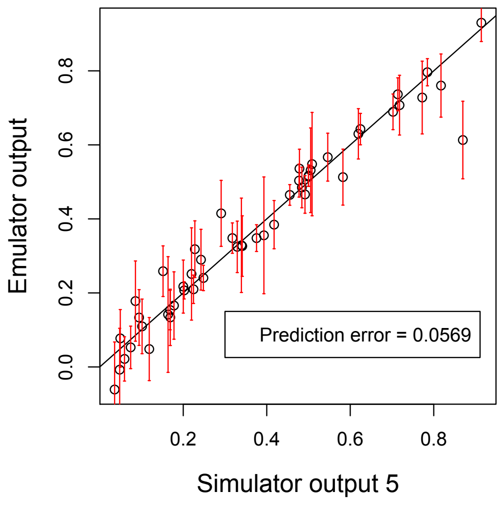


(b) No derivative information.

Figure 6.6: Diagnostics plot of the emulators of output 3 built with $n = 50$ training runs and predicting at $n' = 50$ points.



(a) With derivative information.



(b) No derivativeinformation.

Figure 6.7: Diagnostics plot of the emulators of output 5 built with $n = 50$ training runs and predicting at $n' = 50$ points.

6.3.2 Emulating each output independently

We now wish to determine how the derivatives impact the performance of independent emulators for each output to compare with the multivariate approach. We can do this by reducing the data further, and then investigating how the performance of the emulators vary when we add further runs with just the response, and when we add further runs with the derivative information in addition to the response. We adopt a similar method to that of Section 4.5 in Chapter 4 but use an increasing number of runs up to $n = 50$ of the total training data, as detailed in Section 6.3.1, and use the $n' = 50$ runs that are held back to calculate the prediction error and a measure of uncertainty for each emulator. We calculate these as in Section 4.5:

$$\begin{aligned} \text{Prediction error} &= \sqrt{\frac{1}{n'}\sum(m^{**}(\mathbf{x}_i') - \eta(\mathbf{x}_i'))^2}, && (6.18) \\ \text{Mean standard deviation} &= \frac{1}{n'}\sum \hat{\sigma}\sqrt{\mathrm{c}^{**}(x_i, x_i)}, && (6.19) \end{aligned}$$

for $i \in \{51, \ldots, 100\}$ and $n' = 50$.

We want to build emulators with increasing amounts of training data; starting with $n = 2$, in the case where we include derivatives, and with $n = 9$, in the case where we do not include derivatives, up to $n = 50$ for both cases. We have truncated a 100 point LHS to provide 50 input sites to select from. The $n = 50$ design, however, is not optimal and therefore any design resulting from further truncations will also not optimally spread the points in the space. This makes it difficult to compare the emulators built with different numbers of runs as the inclusion of one extra point may be particularly informative in some cases, and not in others. To account for this we randomly permute the order of the 50 training runs. We then take the first 2 runs as indexed by this permutation and build an emulator with derivatives, calculating the prediction error and the measure of uncertainty for this emulator. Next, we take the third run, as indexed by this permutation, in addition to the first 2 runs and build another emulator with derivatives. We continue in this way, building emulators without derivatives in addition from $n = 9$ onwards, until $n = 50$. We then randomly permute the order of the 50 training runs again, to generate a second permutation and repeat the process. This is repeated 20 times and we then calculate the mean prediction error and measure of uncertainty across the 20 permutations for each value of n.

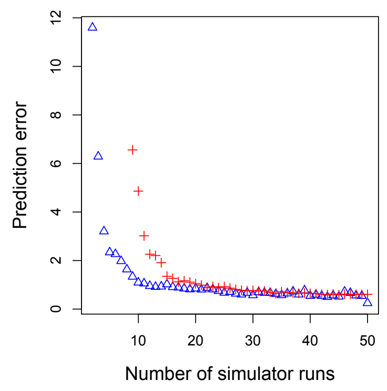
There are only 50 combinations when $n = 49$ and due to the random sampling, 3 of these permutations resulted in the same 49 points being selected. Two of these combinations were therefore discarded and random sampling continued until we had 20 unique permutations. Clearly for $n = 50$ there is no need to permute the order as we only have 50 training runs to choose from, and the order they are in the training data is irrelevant. All emulators are built with a linear prior mean and Gaussian correlation function with Θ estimated by MLE.

The results of the experiment described above are shown in Figure 6.8. The prediction error decreases for all emulators as we increase the training data, as expected. Figures 6.8a and 6.8b, show that for outputs 1 and 2 initially the emulators with derivatives achieve lower prediction errors but when $n > 20$, approximately, the emulators without derivatives have caught up and are performing similarly. Corresponding plots for outputs 3 and 4, Figures 6.8c and 6.8d, show a similar pattern to outputs 1 and 2. Here though the effect of the derivatives appears to be slightly weaker as the emulators without derivatives are achieving similar prediction errors to those with derivatives for approximately $n > 18$. Figure 6.8e, which illustrates the performance of the emulators of output 5 provides very different results. The emulators with derivatives consistently outperform corresponding emulators without derivatives for all n and the margin in prediction error between the emulators appears relatively large. This is explored further in Chapter 4.

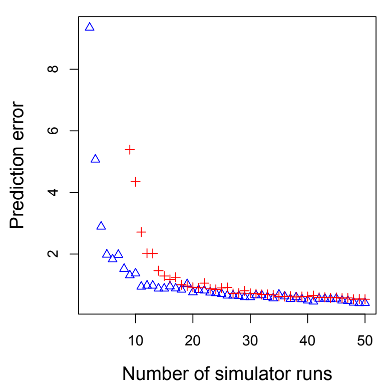
The computational expense of building the various emulators is not considered here but discussed in Section 6.3.4.

6.3.3 Five-output emulation of the radiation transport model

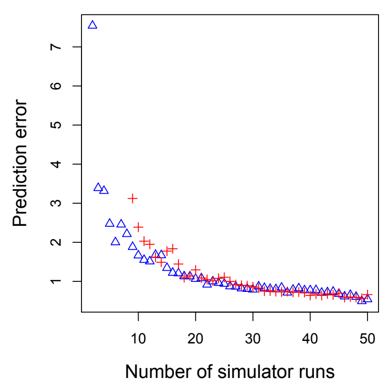
We now want to build a multi-output emulator, with and without the derivative information, to compare with the univariate approach. As in Section 6.3.2, we choose the covariance function: $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta \,(\mathbf{x}_i - \mathbf{x}_j)\right\}$ and a linear form for the prior mean is selected. We adopt the same method for comparing emulators as in Section 6.3.2 and select 50 space filling runs for total training data while holding back the remaining 50 simulator runs for prediction. Multi-output emulators are then built with increasing training data up to $n = 50$. The same permutations of the order of the training data in Section 6.3.2 are used



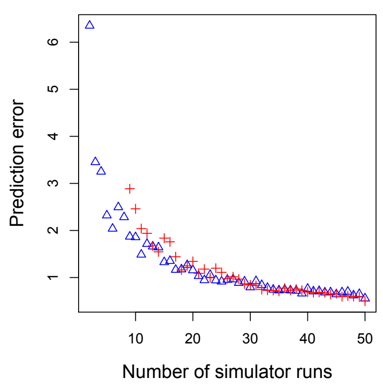
(a) Output 1



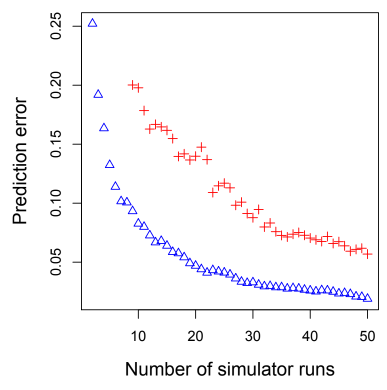
(b) Output 2



(c) Output 3



(d) Output 4



(e) Output 5

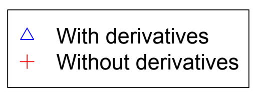


Figure 6.8: Comparison of the prediction error for independents emulators of the 5 outputs, built with varying numbers of simulator runs.

here and multiple emulators are built for each value of n. The mean prediction error for each n is then evaluated and the results can be seen in Figure 6.9.

The multi-output emulators perform similarly to the independent emulators, in particular for outputs 3 and 4. Figures 6.9c and 6.9d show that although for low n emulators with derivatives achieve a lower prediction error, once we have approximately $n > 20$ runs in the training data, the inclusion of derivative information has little effect on the performance of the resulting emulator.

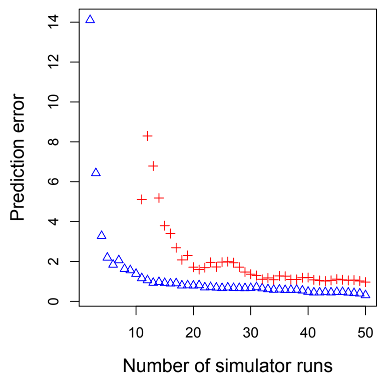
It would appear, for outputs 1 and 2 though, that the derivatives may be more valuable in a multi-output emulator than in independent emulators. Figures 6.9a and 6.9b, show the prediction error of the emulators without derivatives does not quite reach equality with the emulators built with derivative information, even with $n = 50$. In comparison, Figures 6.8a and 6.8b show that the independent emulators achieve similar prediction error regardless of derivative information for approximately $n > 20$.

Finally, in Figure 6.9e we see the performance of the multivariate emulator for output 5. Again similarity with the corresponding independent emulator is evident and we see the emulators with derivatives consistently outperforming the emulators without derivatives. The gap in prediction error between emulators is now smaller though, implying that the derivatives are perhaps less informative for this output in the multivariate setting.

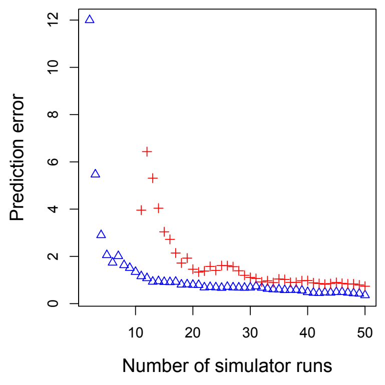
In summary, the results of this investigation are mixed and we explore this further, also with consideration of the computational expense required to build the various emulators, in Section 6.3.4.

6.3.4 Comparisons

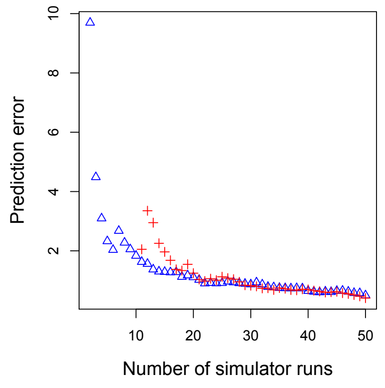
Figures 6.8 and 6.9 in Sections 6.3.2 and 6.3.3 show how the prediction error of independent and multivariate emulators, respectively, vary for increasing n. We now directly compare the univariate and multivariate approach, with and without derivative information. Figure 6.10 shows the predictive performance compared to the computational expense required to build each emulator. We know that the adjoint requires approximately twice the amount of computing time to run than the simulator. For simplicity therefore we assume each simulator run (standard version of the radiation transport model) requires one computational unit, and each time we run the adjoint of the radiation transport model we require 2 units.



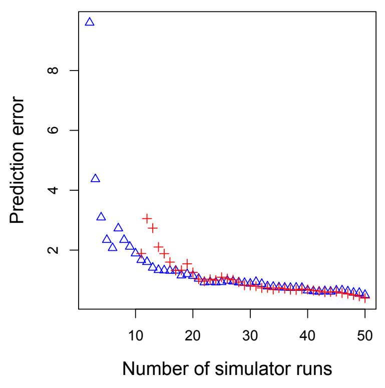
(a) Output 1



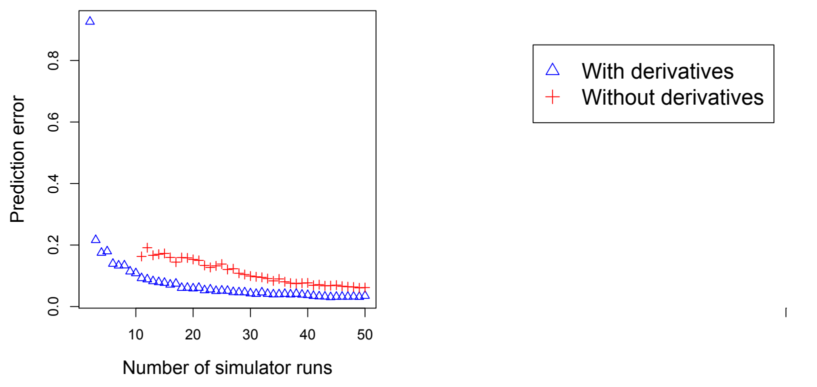
(b) Output 2



(c) Output 3



(d) Output 4

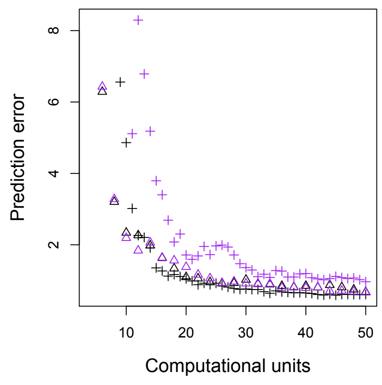


(e) Output 5

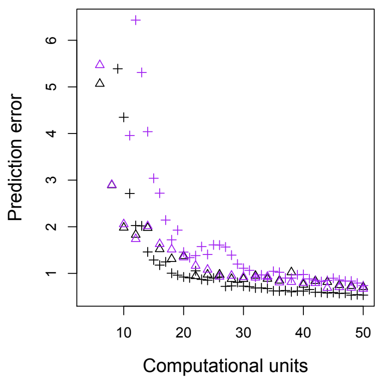
Figure 6.9: Comparison of the prediction error for multivariate emulators of the 5 outputs, built with varying numbers of simulator runs.

As an adjoint model outputs the model response in addition to the partial derivatives this means we can obtain the model response and derivatives in return for 2 computational units, or just model response in return for 1 unit. We do not account for any extra computational time to build either the independent or multivariate emulators outside of the simulator or adjoint run time. The prediction errors generated by both the independent and multi-output emulators built with a computational expense of 4 units, which equates to 2 runs with derivatives, are very large and removed from Figure 6.10. The minimum number of runs required to build an emulator without derivatives is 9, which equates to 9 computational units and is therefore not affected by the enforced range of $[5, 50]$ on the x axis. This enables a more appropriate scale on the y-axis for overall comparison of the emulators. In addition to Figure 6.10, and to enable all the data to be presented, we plot the prediction errors on a log scale. This comparison, which includes the results for emulators built with a computational expense of 4 units, is shown in Figure 6.11 and allows a closer examination of the performance of the emulators. We see from Figures 6.10a and 6.10b that for outputs 1 and 2, though there is little difference, the independent emulators without derivatives built with a minimum of 14 computational units achieve the lowest prediction errors. The emulators with derivatives perform similarly, regardless of whether the prediction is part of the multi-output emulator. The multi-output emulator without derivatives, however performs slightly worse for all n. We see a different pattern in Figures 6.10c and 6.10d though: for outputs 3 and 4 the superior emulators for all n are clearly the emulators without derivative information, and there is very little difference in prediction error between the multi-output and independent emulators. The emulators of output 5, shown in Figure 6.10e, however, provide contrasting results to those of outputs 1 to 4. Now we see the independent emulators, with derivatives, outperforming the other emulators. Derivative information is clearly very informative for this output as while the independent emulators outperform the multi-output emulators, the multi-output emulators with derivatives are still achieving lower prediction errors than the independent emulator without derivatives. The results from all 5 outputs are summarised in Table 6.1.

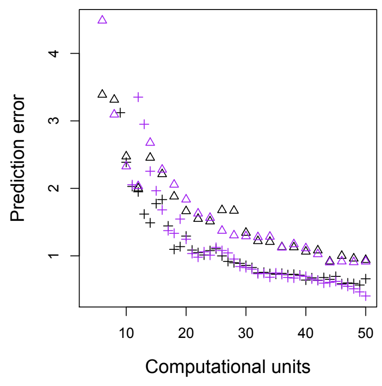
We also compare the uncertainty associated with each of the emulators in Figure 6.12. This measure of uncertainty, given by equation (6.19), is produced by taking the standard deviation of each emulator at each validation point; a mean value is then calculated for each n. The results are, overall, as we would



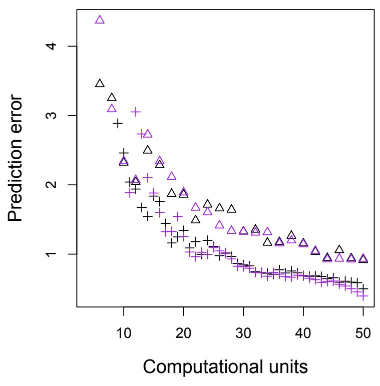
(a) Output 1



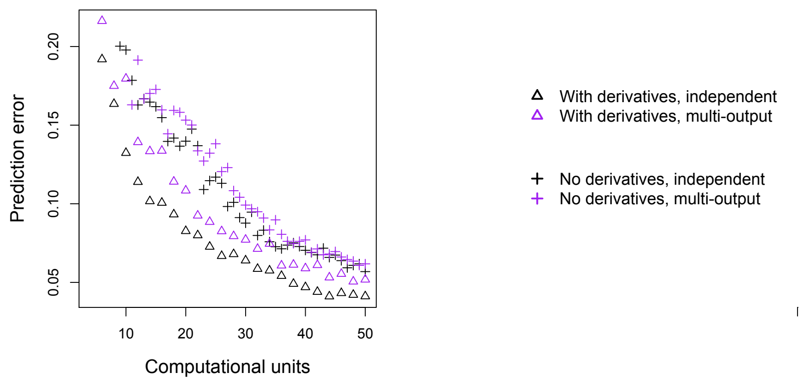
(b) Output 2



(c) Output 3

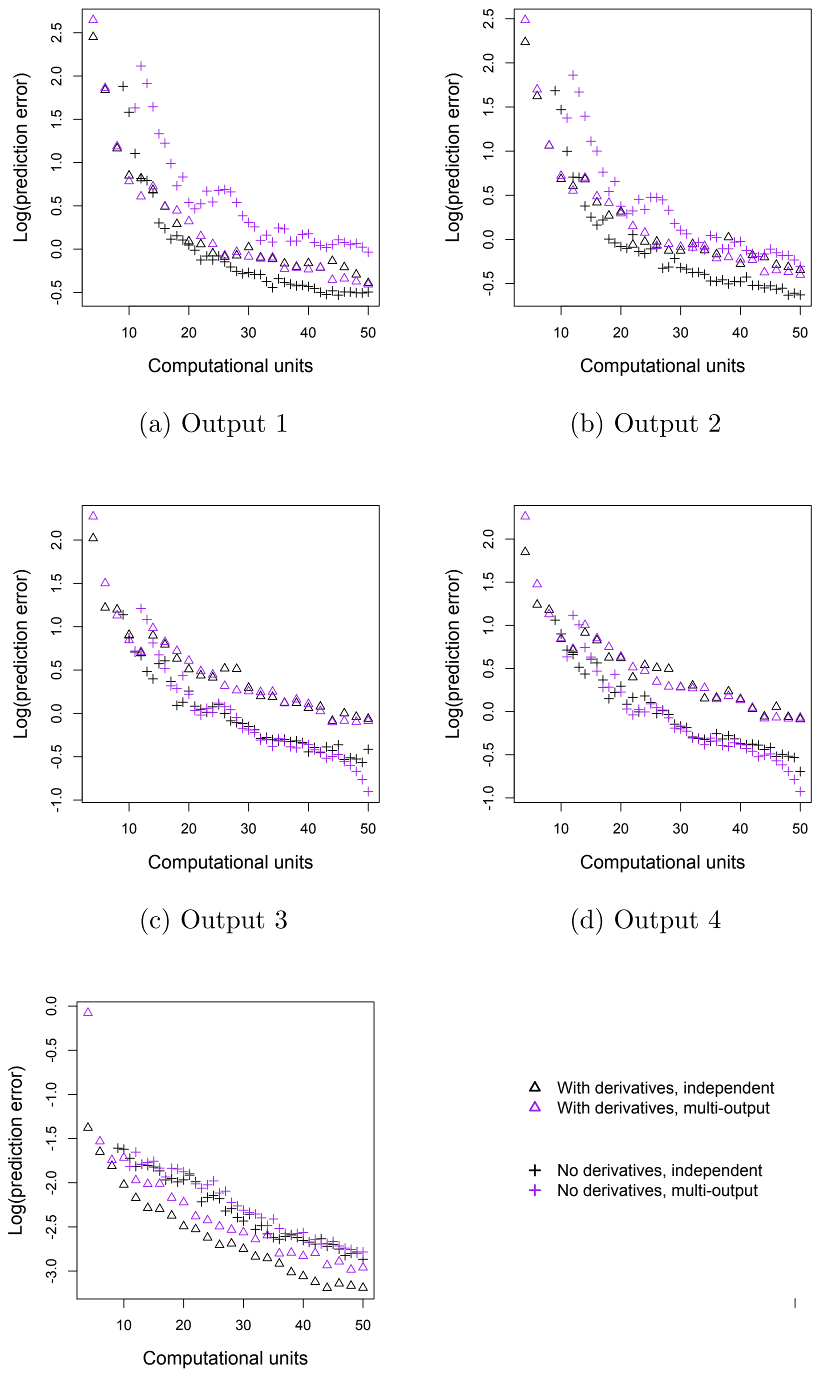


(d) Output 4



(e) Output 5

Figure 6.10: Comparison of the prediction error of the multivariate and independent emulators for all 5 outputs. The prediction errors resulting from the emulators built with 4 computational units are removed.



(a) Output 1

(b) Output 2

(c) Output 3

(d) Output 4

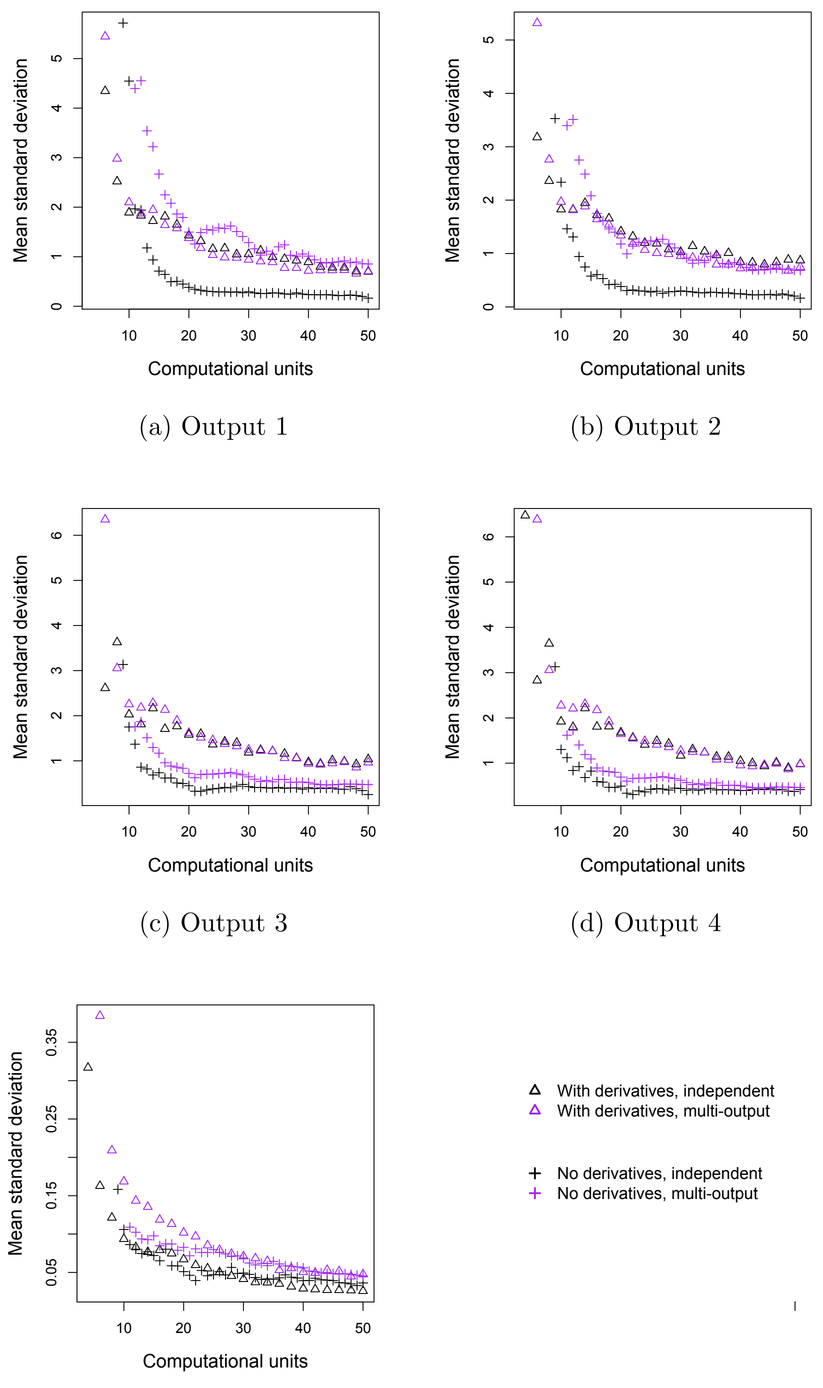
(e) Output 5

Figure 6.11: Comparison of the prediction error on the log scale of the multivariate and independent emulators for all 5 outputs. All data is presented.

expect with the emulators achieving the lowest prediction errors in Figure 6.10 also producing the lowest mean standard deviation. There is a bigger gap in the uncertainty, however, between the independent emulators without derivatives and the remaining emulators for outputs 1 and 2. These results are also summarised in Table 6.1 to give an overview of the relative performance of the emulators.

Output	Lowest prediction error		Lowest uncertainty	
	Type	Derivatives?	Type	Derivatives?
1,2	Independent	No	Independent	No
3,4	Either	No	Independent	No
5	Independent	Yes	Independent	Either

Table 6.1: Summary of best achieving emulators.



(a) Output 1

(b) Output 2

(c) Output 3

(d) Output 4

(e) Output 5

Figure 6.12: Comparison of the uncertainty of the multivariate and independent emulators for all 5 outputs.

6.4 Calibration of radiation transport model

6.4.1 Introduction

In Sections 6.3.2 and 6.3.3 we built emulators of the radiation transport model to provide an efficient method for performing further analysis on the model. We now use the emulators to perform calibration, finding what input settings are consistent with a given model response. Observational data is unavailable, so as an alternative we choose one run from the training set of 100 to represent such an observation. The 22nd run is chosen by the domain expert as an interesting setting of parameters. The inputs and outputs for this run are given in Table 6.2.

Inputs	Outputs		
		Native	Transformed
$r_1 = 7.6113$	$\gamma_1 =$	4.132718e-07	-14.6991603
$\rho_1 = 18.6040$	$\gamma_2 =$	2.308216e-04	-8.3738654
$\rho_2 = 7.2778$	$\gamma_3 =$	2.134913e-04	-8.4519145
$z_1 = 4.3119$	$\gamma_4 =$	1.652884e-03	-6.4052336
$z_2 = 6.4739$	$1/k_{eff} =$	2.039376e+00	0.4903461

Table 6.2: Inputs and outputs for run index 22, chosen to be the target output.

To do this we use a Markov chain Monte Carlo (MCMC) method with the emulators to learn which other sets of inputs are consistent with the output from run index 22. MCMC and the method we employ is discussed further in Section 6.4.2 along with the results from this applied to the radiation transport model.

6.4.2 MCMC applied to radiation transport model

If we are interested in inferences from a posterior distribution but cannot analytically derive the distribution then we can use Markov chain Monte Carlo methods. We generate a Markov chain and the stationary distribution of the chain is equal to the desired posterior distribution.

One way of constructing such a chain is by using the the Metropolis-Hastings algorithm: let $\mathbf{x}_t$ be the state of the chain a time t.

1. Choose a proposal distribution, $\mathrm{q}(\mathbf{x}_c|\mathbf{x}_t)$.

2. Generate a candidate value $\mathbf{x}_c$ from the proposal distribution.
3. Calculate the likelihood ratio and the ratio of the proposal density between the current sample, $\mathbf{x}_t$ and the candidate sample, $\mathbf{x}_c$,
4. Draw α from U(0,1).
5. If $\frac{f(\mathbf{x}_c)q(\mathbf{x}_c|\mathbf{x}_t)}{f(\mathbf{x}_0)q(\mathbf{x}_c|\mathbf{x}_t)} > \alpha$; then accept the candidate value and $\mathbf{x}_{t+1} = \mathbf{x}_c$.
   else; reject the candidate value and $\mathbf{x}_{t+1} = \mathbf{x}_t$.

To learn which other input configurations are consistent with our target output we apply MCMC methods using emulators. From the results of the investigation into the comparison of multivariate and univariate emulators, with and without derivatives, we choose to build univariate emulators for all outputs and include derivative information only in the emulator of output 5. We remove the target output from the training data and build an emulator, for each output, with the remaining 99 runs in the data set.

We apply the Metropolis-Hastings algorithm as outlined above. We take the following, approximately mid-range, points for the initial values of the inputs: $\mathbf{x}_0 = (10\ 11\ 11\ 2.5\ 5)$ and generate the first candidate value: $\mathbf{x}_c = \mathbf{x}_0 + U(0,1)$. We use the following likelihood function:

$$\exp\left\{-\sum_{i=1}^{5}\left(\frac{m_i^{**} - \mathbf{t}_i}{\mathrm{sd}\,\mathbf{t}_i}\right)^2\right\},$$

where m_i^{**} is the posterior mean of the emulator, $\mathbf{t}_i$ is the target output and sd = 0.1, which represents observational error and is scaled to the measurement. A uniform prior over the original parameter range results in a posterior distribution conforming to the likelihood.

We generated 100,000 samples from the posterior distribution. Although the radiation transport model is relatively quick to run in comparison with other complex models, performing 100,000 runs would still take an order of days. By using the emulator instead, the MCMC analysis was complete in approximately an hour.

The results are summarised in Figure 6.13, which shows two-dimensional marginal projections of the 5 dimensional space for the last 5000 iterations. Along the diagonal are histograms showing the density for each input parameter, where the bottom left panel shows input 1 through to the top right panel which shows input 5. The upper panels show contours and the lower panels show two dimensional scatter plots between the inputs, where the black points are from earlier in the chain and as the chain progresses move to the light colours. The blue star shows input 22, the observation we are comparing to. Firstly, we can see that for each input the chain covers the true input configuration of the observation which is encouraging. It appears that to achieve similar output to the observation, it is plausible for input 5 to take most values across its input range. In contrast, input 3 appears to be more tightly constrained. In addition to this we can see that there exists a tradeoff between the first and third input parameter, which represents inner radius and outer density respectively.

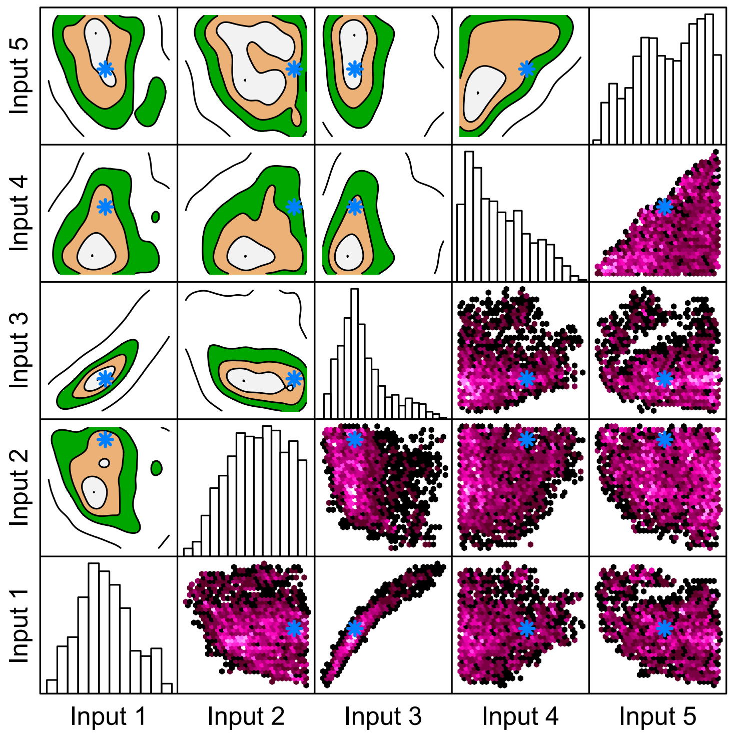


Figure 6.13: Two-dimensional plots from the MCMC results.

6.5 Conclusions

In this chapter we have investigated the use of derivative information in both univariate and multivariate emulation, through the application of the radiation transport model. We have not seen here any strong evidence in support of building a multivariate emulator, with or without derivatives, over independent emulators for each output. The impact of derivative information has been inconsistent, with the derivatives proving more informative for some outputs than others, in both the univariate and multivariate environments.

Despite investigating the use of derivative information in multivariate emulation we still chose to use univariate emulators in the calibration of the radiation transport model in Section 6.4. This is because the multivariate emulator performs similarly to the independent emulators, and as noted above, therefore provides no strong reason to choose an emulator other than univariate emulators for the calibration. Given all the data available, derivatives are not required to provide accurate emulation of outputs 1 to 4 and therefore are not used in the calibration study. We have seen here and in Chapter 4 though, that derivative information does improve the emulation of output 5 and therefore an emulator built with function output 5 and the corresponding derivatives is selected.

In summary, suppose we wish to build an emulator of the 5 outputs of the radiation transport model and have only a finite amount of computational time in which to run the simulator or adjoint. We describe this time in terms of total computational units, T. In this chapter we have attempted to answer the question: *should we build 5 independent emulators or one multi-output emulator and, given the type of emulator, would it be more efficient to include derivative information?* We have not seen any strong evidence to support choosing the multivariate option over univariate and so for this model, the answer to the first part of the question is straightforward. Whether it is more efficient to include derivatives in the independent emulators however, is much more difficult to answer. From Figures 6.10 and 6.11 we see that for outputs 1 and 2, derivative information has little effect and we could choose either to run the adjoint at $\frac{T}{2}$ input sites, or the simulator at T input sites. Emulators of outputs 3 and 4, however, clearly do not benefit enough from the inclusion of derivatives to justify the computation expense of the adjoint. In Chapter 4 we see that emulators of very smooth models seem to benefit less from the inclusion of derivatives than emulators of models with a higher degree of variability. We do not adopt a formal measure to assess smoothness in the radiation transport model, but an indication can be drawn from the values of the smoothness parameters as estimated by MLE in the emulators. All inputs were scaled to be on the unit cube hence we can combine the 5 θ values for each output in the independent emulators into one measure, for ease of comparison. We find that the mean values of θ for outputs 3 and 4, 0.285 and 0.294 respectively, are smaller than the corresponding estimates of θ for outputs 1, 2 and 5, $0.871, 2.507, 0.485$ respectively. This implies that outputs 3 and 4 are smoother functions of the inputs than outputs 1, 2 and 5 and is consistent with the conclusions made in Chapter 4. For outputs 3 and 4, therefore, we should choose to perform T runs of the simulator. In contrast, the emulator of output 5 performs much better with derivatives included, despite the computational expense of the adjoint and the conclusion for this output would be to perform $\frac{T}{2}$ adjoint runs. The adjoint model, when run at a particular input configuration, produces all the partial derivatives of the 5 outputs with respect to the 5 inputs. It is not possible to 'switch off' the derivatives of outputs 3 and 4 and thus reduce the computational expense of running the adjoint. To achieve this a new adjoint model would have to be especially coded and therefore is very unlikely to be an efficient solution to our question. We are not restricted, however, to running either the adjoint at $\frac{T}{2}$ inputs sites or the simulator at T inputs sites. We suggest, therefore, that a mixture of adjoint and simulator runs is likely to be optimal in this case. This generates an interesting design problem and scope for further work: we would need to decide how many adjoint runs and how many simulator runs to perform, and also at which input configurations to obtain derivatives in addition to the model response.

Chapter 7

Conclusions and further work

In this thesis we have investigated the use of derivative information in the statistical analysis of computer models. We began in Chapter 1 by introducing the complex models used in this work and motivated the need for managing uncertainty in complex models. In Chapter 2 we discussed derivatives of complex models. Many model users find knowledge of derivatives in complex models informative, with multiple analyses requiring, or improving from, the knowledge of derivative information. Examples of such analyses are optimisation, data assimilation and sensitivity analysis. Given the apparent value of derivatives, we then looked at exisiting methods of producing them: adjoint models are becoming increasingly popular as a more accurate and efficient means to generating derivative information than the traditional, finite differences approach. Adjoint models also have their drawbacks however, mostly in the additional computational expense required to run them and the initial time and resource that must be allocated to produce such a model. Having first discussed the standard Gaussian process emulator in Chapter 3, we then proposed a new approach to generating derivatives in Chapter 5. We showed how we can build a Gaussian process emulator, with or without derivative information, to emulate the derivatives of a complex model. We can then predict the partial derivatives at any input configuration, within the original range, and each prediction is associated with a level of uncertainty. The results were encouraging and we were able to accurately emulate derivatives even without any derivative information in the training data. This is very likely to be much more efficient than either running an appropriate adjoint model, or conducting a finite differences experiment. The emulator aprroach to prediciting derivatives, however, is not without its limitations. For example, the simulator and its derivatives must be a suitable function for emulation, which demands that the model is a smooth function of its inputs and the derivatives are therefore fully defined across the input space. In addition, a Gaussian process should be a suitable tool with which to model the simulator. We have seen that with these conditions satisfied the emulation approach to generating derivatives has the potential to be an efficient alternative to existing methods.

This thesis has also been concerned with looking at whether the inclusion of derivative information could result in a more efficient emulator of model response. In Chapter 4 we presented an investigation into this using both toy models and a real complex model: a radiation transport model. Whether further efficiency is likely to be achieved in general depends on the computational cost of obtaining the derivatives, but we did find that if the model is very smooth the derivative information is less valuable and negligible further efficiency, if any, is achieved. If the model exhibits slightly different behaviour though, such that it still responds smoothly to its inputs but with more change in output for relatively small changes in inputs, the derivatives appear to be more valuable. We also see through the use of toy models that the emulators we tested without derivatives generally tend to require twice as many model runs as the emulators with derivatives to produce a similar predictive performance. The adjoint of the radiation transport model is relatively cheap to run, requiring twice the amount of time of the standard model, and yet, in our emulation study derivatives only provided a more efficient emulator for one of the 5 outputs. If we only have a very small amount of training data then the derivatives are required to build a valid emulator of the radiation transport model, and therefore derivatives may be more valuable in an initial analysis, especially if the model experts do not have much knowledge a priori of the behaviour of the model. In conclusion, an optimal solution is likely to be a hybrid design consisting of adjoint runs in some parts of the input space and simulator runs in others. This presents an interesting design problem which we discuss further in the *Design* section later in this chapter.

Chapter 6 was concerned with multi-output emulators and we concluded that from our example, regardless of the inclusion of derivative information, the multivariate emulator was not notably superior to independent emulators built for each output. Little gain, therefore, would be achieved in building a multi output emulator in replace of univariate emulators.

We now discuss some of the findings in this thesis in more detail and speculate on how the work could be taken forward:

- *Emulation and validation of model derivatives*

  We have discussed in this chapter how an emulator of model derivatives could be a more efficient alternative to either an adjoint model or a finite differences experiment. An emulator, however, must be validated before any predicted derivatives are employed in further analysis. If an adjoint model does not exist then there is little option other than the finite differences approach, the limitations of which were discussed in Chapters 2 and 5. If we have robust diagnostics of an emulator of derivatives, which account for the uncertainty in the FD approach, then this could be a viable alternative to validation through the use of an adjoint model. We may not be able, however, to afford the further simulator runs required to generate FD validation derivatives. In this case, we could perhaps use a (previously approved) emulator of the model response to generate the output required for FD validation derivatives. The emulator of derivatives would not be guaranteed to pass with this method as validation of a (standard) emulator of model response does not necessarily imply a valid emulator of the model derivatives; there have been cases where the emulator of model response passes the diagnostics set out by Bastos and O'Hagan (2009), but the emulator of model derivatives fails. Careful thought and consideration would have to be undertaken to ensure such an approach resulted in a credible emulator of derivatives though.

  Of course if we do have some derivative information already available, rather than building an emulator of derivatives with model response alone and using the derivatives as validation data, a more sensible option would perhaps be to include the derivative information in the training data and then adopt a *'leave one derivative out'* approach. We may not be able to formally validate with this variation of LOO, but we would gain some insight into the performance of the emulator of derivatives. If this approach is adopted then it might be advisable to leave out the model response and all the partial derivatives at the 'left out' input point rather than just the derivative we are predicting. This is because we are unlikely to have simulator output at every input configuration we wish to predict a derivative at.

- *Non-stationary models*

  One of the conditions to generating derivatives, or function output, using the approach described in this work is that the model must be suitable for emulation. Adopting the methodology presented in this thesis assumes the model is a smooth, continuous function of its inputs. As we see in Chapter 5, non-stationarity in the model causes problems for the emulator. The C-GOLDSTEIN model is a smooth function of the wind stress scale input, scf, for most of the selected range but between scf = 2.02 and scf = 2.10 we have observed a 'rough' patch and at some points even discontinuities. This behaviour was not known a priori and it was through the use of an emulator of derivatives that we were able to discover and home in on the problem areas in the input space. Now of course the derivatives do not exist at the discontinuous points, and we were only able to emulate the derivatives in rest of the 'rough' patch of the function by splitting it from the smooth section and emulating it separately. Emulating non-stationary models, regardless of derivative information, is an area which requires further research. The option we took in the C-GOLDSTEIN example in Chapter 5, of splitting the simulator output into regions is an informal version of the Bayesian Treed Gaussian process developed by Gramacy and Lee (2008). Alternative options to investigate include selecting a non-stationary covariance function and 'warping' the input space such that we now have a stationary function in the resulting space to emulate with a Gaussian process in the standard approach. An example of 'warping' the input space was given in the Borehole model example in Chapter 5. Despite the flexibility of a Gaussian process it may be that a particular model simply can not be well modelled by a Gaussian process. If this is the case though, the validation diagnostics of Bastos and O'Hagan (2009) should highlight this.

- *Design*

  Assuming $\eta(\cdot)$ and the subsequent derivatives of $\eta(\cdot)$ can be modelled by a Gaussian process, and assuming we have derivatives to validate our emulator where required, a topic which also requires further research is that of design. A lot of work has been done into the design of computer experiments, but not when we wish to emulate derivatives with only the function output in the training data. As we see in Chapter 5 a purely space-filling design may not be optimal. Discussion with Max Morris on this issue has led to the (informal) conjecture that maximin distance designs are probably not D-optimal for derivative prediction, but are very close to being optimal. This is in contrast to the results seen in the 1-d toy example in Chapter 5; though as we explain in Section 5.5 of that chapter, the problem in this example was due to the maximin distance design resulting in equidistant points, and we wouldn't expect this to occur when dealing with higher dimensional input space. There is also likely to be the option of obtaining derivatives in some parts of the input space and the model response at others, the optimal, or a near-optimal, combination of derivatives and response would be ideal and therefore also provides scope for further work.

Morris *et al.* (1993) look at optimal design, for predicting model response, in the case where derivatives are available and included in the training data. They conclude that the same kind of design which is optimal without derivatives, is still optimal with the inclusion of the derivatives. They only consider, however, the situation where a derivative is observed at every input site that the model response is also evaluated at. As when we wish to emulate derivatives it maybe that the optimal solution is actually to have a hybrid design which requires derivatives at some points, response at some points and both derivatives and response combined at others. It would be interesting to investigate such a design and then re-examine the emulation, with and without derivatives, of the radiation transport model. This is similar to the situation examined by Cumming and Goldstein (2009) where there are 2 simulators modelling the same real system: one is accurate and relatively slow while the other is approximate but much faster. Cumming and Goldstein (2009) describe an approach that combines the outputs from the 2 simulators in the building of one emulator. There are of course differences between Cumming and Goldstein (2009) and our situation here; most importantly our "fast" model (the standard simulator) is not an approximation of the "slow" model (the adjoint model), instead the two models provide different types of information. Nevertheless their approach could perhaps be adapted to fit our situation. The computational cost of the C-GOLDSTEIN adjoint model is 18 times more than the standard simulator. While this cost has the potential to decrease by optimising the code further, it is still an expensive model and a mixed design of adjoint and simulator runs could be interesting to investigate.

- *Concluding thoughts*

  Where we've included derivative information in an emulator in this thesis we have assumed these derivatives to be exact within the computational precision error. It would be interesting to investigate if there is some noise on the derivative observations, how large an effect there is. If validation diagnostics could point specifically to this problem this would be useful when considering whether to include derivative information in a Gaussian process emulator.

In summary, in this thesis we have presented a new approach to generating derivatives which, assuming the model is appropriate for emulation, is likely to be much more efficient than the existing methods of calculating derivatives. Whether the inclusion of derivative information when building an emulator of model response results in a more efficient emulator, depends on the computational expense of generating the derivatives. As a result of work presented here, however, a design consisting of a mixture of derivatives at some points, model response at some points and both derivatives and model response at other points would be advised.

Bibliography

Alcouffe, R. E. (2001). Partisn calculations of 3d radiation transport benchmarks for simple geometries with void regions. *Progress in nuclear energy*, **39**, 181–190.

Andrianakis, Y. and Challenor, P. G. (2009). Parameter estimation and prediction using gaussian processes. *MUCM Technical report 09/05, University of Southampton.*

Azman, K. and Kocijan, J. (2005). Comprising prior knowledge in dynamic gaussian process models. *Proceedings of the International Conference on Computer Systems and Technologies*, IIIB.2/1–6.

Bastos, L. S. and O'Hagan, A. (2009). Diagnostics for gaussian process emulators. *Technometrics*, **51**, 425–438.

Bayarri, M. J., Berger, J. O., Paulo, R., Sacks, J., Cafeo, J. A., Cavendish, J., Lin, C.-H. and Tu, J. (2007). A framework for validation of computer models. *Technometrics*, **49**, 138–154.

Conti, S. and O'Hagan, A. (2010). Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, **140**, 640–651.

Craig, P. S., Goldstein, M., Seheult, A. H. and Smith, J. A. (1997). Pressure matching for hydrocarbon reservoirs: A case study in the use of bayes linear strategies for large computer experiments. *Case Studies in Bayesian Statistics 3*, 36–93.

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik*, **31**, 377–403.

Cumming, J. A. and Goldstein, M. (2009). Small sample bayesian designs for complex high-dimensional models based on information gained using fast approximations. *Technometrics*, **51**, 377–388.

Currin, C., Mitchell, T., Morris, M. and Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, **86**, 953–963.

Edwards, N. R. and Shepherd, J. G. (2002). Bifurcations of the thermohaline circulation in a simplified three-dimensional model of the world ocean and the effects of inter-basin connectivity. *Climate Dynamics*, **19**, 31–42.

Fanning, A. F. and Weaver, A. J. (1996). An atmospheric energy-moisture balance model: climatology, interpentadal climate change, and coupling to an ocean general circulation model. *Journal of Geophysical Research*, **101**, 15111–15128.

Favorite, J. A. and Bledsoe, K. C. (2008). Inverse transport methods using subcritical neutron multiplication. *Los Alamos Technical Report LA-UR-08-7033 (submitted to Nuclear Science and Engineering).*

Favorite, J. A., Bledsoe, K. C. and Ketcheson, D. I. (2009). Surface and volume integrals of uncollided adjoint fluxes and forward-adjoint flux products. *Nuclear Science and Engineering*, **163**, 73–84.

Goldstein, M. and Woof, D. A. (2007). *Bayes Linear Statistics: Theory and Methods*. Wiley.

Gordon, C., Cooper, C., Senior, C. A., Banks, H., Gregory, J. M., Johns, T. C., Mitchell, J. F. and Wood, R. A. (2000). The simulation of sst, sea ice extents and ocean heat transports in aversion of the hadley centre coupled model without flux adjustments. *Climate Dynamics*, **16**, 147–168.

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

Griewank, A. (2003). A mathematical view of automatic differentiation. In *Acta Numerica*, Cambridge University Press, vol. 12, 321–398.

Hall, M. C. H. and Caucui, D. G. (1982). Sensitivity analysis of a radiative-convective model by the adjoint method. *Journal of the Atmospheric Sciences*, **39**, 2038–2050.

Hang, G., Santner, T. J. and Rawlinson, J. J. (2009). Simultaneous determination of tuning and calibration parameters for computer experiments. *Technometrics*, **51**, 464–474.

Hascoët, L., Greborio, R.-M. and Pascual, V. (2005). Computing adjoints by automatic differentiation with tapenade. In B. Sportisse and F.-X. LeDimet (eds.), *Ecole INRIA-CEA-EDF "Problemes non-lineaires appliques"*. Springer.

Hascoët, L. and Pascual, V. (2004). *TAPENADE 2.1 user's guide*. Technical Report 0300, INRIA.
URL `http://www.inria.fr/rrrt/rt-0300.html`

Haylock, R. G. and O'Hagan, A. (1996). On inference for outputs of computationally expensive algorithms with uncertainty on the inputs. In J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds.), *Bayesian Statistics 5*. Oxford: University Press, 629–637.

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

IPCC (2000). *Special Report on Emissions Scenarios*. Cambridge University Press.

Johnston, W. R. (2007). Historical data relating to global climate change. *WEB Resource: http://www.johnstonsarchive.net/environment/co2table.html*.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of Royal Statistical Society. Series B (Statistical Methodology)*, **63**, 425–464.

Killeya, M. R. H. (2004). *"Thinking Inside The Box" Using Derivatives to Improve Bayesian Black Box Emulation of Computer Simulators with Applications to Compartmental Models.*. Ph.D. thesis, Department of Mathematical Sciences, University of Durham.

Killeya, M. R. H. and Goldstein, M. (2007). Exploiting compartmental structure through derivatives in bayesian emulation of computer simulators with application to the plankton cycle. *Under revision for Journal of Statistical Planning and Inference*.

Leith, D. J., Leithead, W. E., Solak, E. and Murray-Smith, R. (2002). Divide and conquer identification using gaussian process priors. *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, **1**, 624–629.

Loeppky, J. L., Sacks, J. and Welch, W. J. (2009). Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, **51**, 366–376.

Lyness, J. N. and Moler, C. B. (1969). Generalised romberg methods for integrals of derivatives. *Numer. Math*, **14**, 1–14.

Mackay, D. J. C. (2002). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, New York.

Marotzke, J., Giering, R., Zhang, K. Q., Stammer, D., Hill, C. and Lee, T. (1999). Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport sensitivity. *Journal of Geophysical Research*, **104**, 29529–29547.

Marsh, R., Edwards, N. R. and Shepherd, J. G. (2002). Development of a fast climate model (c-goldstein) for earth system science. *Southampton Oceanography Centre Internal Document*, **83**.

McKay, M. D., Beckman, R. J. and Conover, W. J. (1979). A comparison of three methods for selecting values of input in the analysis of output from a computer code. *Technometrics*, **21**, 239–245.

Morris, M. D. and Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, **43**, 381–402.

Morris, M. D., Mitchell, T. J. and Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, **35**, 243–255.

Numerical Algorithms Group (2006). *NAG Fortran Library Manual Mark 21*. The Numerical Algorithms Group Ltd, Oxford, UK. 2006.

Oakley, J. (1999). *Bayesian Uncertainty Analysis for Complex Computer Codes*. Ph.D. thesis, Department of Probability and Statistics, University of Sheffield.

O'Hagan, A. (1991). Bayes-hermite quadrature. *Journal of statistical planning and inference*, **29**, 245–260.

O'Hagan, A. (1992). Some bayesian numerical analysis. In J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds.), *Bayesian Statistics 4*. Oxford: University Press, 345–363.

O'Hagan, A. (2006). Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering and System Safety*, **91**, 1290–1300.

Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes*. Mcgraw-hill, third edn.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992). *Numerical Recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA.

Rasmussen, C. E. and Williams, C. (2006). *Gaussian processes for machine learning*. The MIT Press.

Roh, L., Bischof, C., Chang, N., Lee, K., Kanevsky, V., Nakagawa, O. and Oh, S.-Y. (1999). Fast statistical-based interconnect modeling using automatic differentiation. *Simulation of Semiconductor Processes and Devices, 1999. SISPAS '99. 1999 Conference on*, 159–162.

Rougier, J. (2008). Efficient emulators for multivariate deterministic functions. *Journal of Computational and Graphical Statistics*, **17**, 827–843.

Rougier, J., Guillas, S., Maute, A. and Richmond, A. (2009). Expert knowledge and multivariate emulation: The thermosphere-ionosphere electrodynamics general circulation model (tie-gcm). *Technometrics*, **51**, 414–424.

Sacks, J., Welch, W. J., Mitchell, T. J. and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, **4**, 409–423.

Saltelli, A., K.Chan and Scott, E. M. (2000). *Sensitivity Analysis*. John Wiley and Sons, Chichester.

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

Solak, E., Murray-Smith, R., Leithead, W. E., Leith, D. J. and Rasmussen, C. E. (2003). Derivative observations in gaussian process models of dynamic systems. *Advances in Neural Information Processing Systems 15*.

Stein, M. L. (1999). *Interpolation of spatial data - some theory for kriging*. Springer.

Urban, N. and Fricker, T. (2009). A comparison of latin hypercube and grid ensemble designs for the multivariate emulation of a climate model. *Submitted to Computers & Geosciences*.

Weaver, A. J., Eby, M., Wiebe, E. C., Bitz, C. M., Duffy, P. B., Ewen, T. L., Fanning, A. F., Holland, M. M., MacFadyen, A., Matthews, H. D., Meissner, K. J., Saenko, O., Schmittner, A., Wang, H. X. and Yoshimori, M. (2001). The uvic earth system climate model: Model description, climatology, and applications to past, present and future climates. *Atmosphere-Ocean*, **39**, 361–428.

Zachary, D. S. (2004). *Automatic Differentiation for convex optimization and its applications to an intermediate complexity global climate model*. Tech. rep., Logilab (LOGIstic LABoratory), University of Geneva.

Appendix A

MUCM toolkit

The Managing Uncertainty in Complex Models (MUCM) toolkit is an online resource with a large number of pages detailing MUCM methods. Later releases of the toolkit will build on the current version by adding further material. In this Appendix we present the MUCM toolkit pages relevant to derivative information.

A.1 Definition of adjoint

We begin with a definition of an adjoint model. This page can be found at: *http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page=DefAdjoint.html*.

Definition of term: Adjoint

An adjoint is an extension to a simulator which produces derivatives of the simulator output with respect to its inputs.

Technically, an adjoint is the transpose of a tangent linear approximation; written either by hand or with the application of automatic differentiation, it produces partial derivatives in addition to the standard output of the simulator. An adjoint is computationally more expensive to run than the standard simulator, but efficiency is achieved in comparison to the finite differences method to generating derivatives. This is because where computing partial derivatives by finite differences requires at least two runs of the simulator for each input parameter, the corresponding derivatives can all be generated by one single run of an appropriate adjoint.

A.2 Emulating with derivatives

In this section we present the pages which describe how to include derivative information in a Gaussian process emulator. The objective here is to emulate function output.

A.2.1 ThreadVariantWithDerivatives

This page describes the ingredients required to build a Gaussian process emulator with derivative information. This page can be found at:

http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page=ThreadVariantWithDerivatives.html.

Thread: Emulators with derivative information

Overview

This thread describes how we can use derivative information in addition to standard function output to build an emulator. As in the core thread for analysing the core problem using GP methods (ThreadCoreGP) the following apply:

- We are only concerned with one simulator.
- The simulator only produces one output, or (more realistically) we are only interested in one output.
- The output is deterministic.
- We do not have observations of the real world process against which to compare the simulator.
- We do not wish to make statements about the real world process.

Each of these aspects of the core problem is discussed further in page DiscCore.

However now, we also assume we can directly observe first derivatives of the simulator either through an adjoint model or some other technique. This thread describes the use of derivative information when building an emulator for the fully Bayesian approach only, thus we require the further restriction:

- We are prepared to represent the simulator as a Gaussian process.

There is discussion of this requirement in page DiscGaussianAssumption. If we want to adopt the Bayes linear approach it is still possible to include derivative information and this may be covered in a future release of the toolkit. Further information on this can be found in Killeya, M.R.H., (2004) "Thinking Inside The Box" Using Derivatives to Improve Bayesian Black Box Emulation of Computer Simulators with Applications to Compartmental Models. Ph.D. thesis, Department of Mathematical Sciences, University of Durham.

Readers should be familiar with ThreadCoreGP, before considering including derivative information.

Active inputs

As in ThreadCoreGP

The GP model

As in ThreadCoreGP the first stage in building the emulator is to model the mean and covariance structures of the Gaussian process that is to represent the simulator. As explained in the definition page of a Gaussian process (DefGP), a GP is characterised by a mean function and a covariance function. We model these functions to represent prior beliefs that we have about the simulator, i.e. beliefs about the simulator prior to incorporating information from the training sample. The derivatives of a Gaussian process remain a Gaussian process and so we can use a similar approach to ThreadCoreGP here.

The choice of the emulator prior mean function is considered in the alternatives page AltMeanFunction. However here we must ensure that the chosen function is differentiable. In general, the choice will lead to the mean function depending on a set of hyperparameters that we will denote by β.

The most common approach is to define the mean function to have the linear form $m(x) = h(x)^{\mathrm{T}}\beta$, where $h(\cdot)$ is a vector of regressor functions, whose specification is part of the choice to be made. As we are including derivative information in the training sample we must ensure that $h(\cdot)$ is differentiable. This will then lead to the derivative of the mean function: $\frac{\partial}{\partial x}m(x) = \frac{\partial}{\partial x}h(x)^{\mathrm{T}}\beta$. For appropriate ways to model the mean, both generally and in linear form, see AltMeanFunction.

The covariance function is considered in the discussion page DiscCovarianceFunction and here must be twice differentiable. Within the toolkit we will assume that the covariance function takes the form $\sigma^2 c(\cdot,\cdot)$, where σ^2 is an unknown scale hyperparameter and $c(\cdot,\cdot)$ is called the correlation function indexed by a set of correlation hyperparameters δ. The correlation then between a point, x_i, and a derivative w.r.t input k at x_j, (denoted by $x_j^{(k)}$), is $\frac{\partial}{\partial x_j^{(k)}}c(x_i,x_j)$. The correlation between a derivative w.r.t input k at x_i, (denoted by $x_i^{(k)}$), and a derivative w.r.t input l at x_j, (denoted by $x_j^{(l)}$), is $\frac{\partial^2}{\partial x_i^{(k)}\partial x_j^{(l)}}c(x_i,x_j)$. The choice of correlation function is considered in the alternatives page AltCorrelationFunction.

The most common approach is to define the correlation function to have the Gaussian form $c(x_i,x_j) = \exp\{-(x_i-x_j)^{\mathrm{T}}C(x_i-x_j)\}$, where C is a diagonal matrix with elements the inverse squares of the elements of the δ vector. The correlation then between a point, x_i, and a derivative w.r.t input k at point j, $x_j^{(k)}$, is:

$$\frac{\partial}{\partial x_j^{(k)}}c(x_i,x_j) = \frac{2}{\delta^2\{k\}}\left(x_i^{(k)} - x_j^{(k)}\right)\,\exp\{-(x_i-x_j)^{\mathrm{T}}C(x_i-x_j)\},$$

the correlation between two derivatives w.r.t input k but at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)}\partial x_j^{(k)}}c(x_i,x_j) = \left(\frac{2}{\delta^2\{k\}} - \frac{4\left(x_i^{(k)} - x_j^{(k)}\right)^2}{\delta^4\{k\}}\right)\exp\{-(x_i-x_j)^{\mathrm{T}}C(x_i-x_j)\},$$

and finally the correlation between two derivatives w.r.t inputs k and l, where $k \neq l$, at points i and j is:

$$\frac{\partial^2}{\partial x_i^{(k)}\partial x_j^{(l)}}c(x_i,x_j) = \frac{4}{\delta^2\{k\}\delta^2\{l\}}\left(x_j^{(k)} - x_i^{(k)}\right)\left(x_i^{(l)} - x_j^{(l)}\right)\exp\{-(x_i-x_j)^{\mathrm{T}}C(x_i-x_j)\}.$$

Prior distributions

As in ThreadCoreGP

Design

The next step is to create a design, which consists of a set of points in the input space at which the simulator or adjoint is to be run to create the training sample. Design options for the core problem are discussed in the alternatives page on training sample design (AltCoreDesign). Here though, we also need to decide at which of these points we want to obtain function output and at which points we want to obtain partial derivatives. This adds a further consideration when choosing a design option but as yet we don't have any specific design procedures which take into account the inclusion of derivative information.

If one of the design procedures described in AltCoreDesign is applied, the result is an ordered set of points $D = \{x_1, x_2, \ldots, x_n\}$. Given D, we would now need to choose at which of these points we want to obtain function output and at which we want to obtain partial derivatives. This information is added to D resulting in the design, $\tilde{D}$ of length $\tilde{n}$. A point in $\tilde{D}$ has the form (x, d), where d denotes whether a derivative or the function output is to be included at that point. The simulator, $f(\cdot)$, or the adjoint of the simulator, $\tilde{f}(\cdot)$, (depending on the value of each d), is then run at each of the input configurations.

One suggestion that is commonly made for the choice of the sample size, n, for the core problem is $n = 10p$, where p is the number of inputs. (This may typically be enough to obtain an initial fit, but additional simulator runs are likely to be needed for the purposes of validation, and then to address problems raised in the validation diagnostics.) There is not, however, such a guide for what $\tilde{n}$ might be. If we choose to obtain function output and the first derivatives w.r.t to all inputs at every location in the design, then we would expect that fewer than $10p$ locations would be required; how many fewer though, is difficult to estimate.

Fitting the emulator

Given the training sample of function output and derivatives, and the GP prior model, the process of building the emulator is given in the procedure page ProcBuildWithDerivsGP

The result of ProcBuildWithDerivsGP is the emulator, fitted to the prior information and training data. As with the core problem, the emulator has two parts, an updated GP (or a related process called a t-process) conditional on hyperparameters, plus one or more sets of representative values of those hyperparameters. Addressing the tasks below will then consist of computing solutions for each set of hyperparameter values (using the GP or t-process) and then an appropriate form of averaging of the resulting solutions.

Although the fitted emulator will correctly represent the information in the training data, it is always important to validate it against additional simulator runs. For the core problem, the process of validation is described in the procedure page ProcValidateCoreGP. Here, we are interested in predicting function output, therefore as in ProcValidateCoreGP we will have a validation design D' which only consists of points for function output; no derivatives are required and as such the simulator, $f(\cdot)$, not the adjoint, $\tilde{f}(\cdot)$, is run at each x'_j in D'. Then in the case of a linear mean function, weak prior information on hyperparameters β and σ, and a single posterior estimate of δ, the predictive mean vector, m^*, and the predictive covariance matrix, V^*, required in ProcValidateCoreGP, are given by the functions $m^*(\cdot)$ and $v^*(\cdot,\cdot)$ which are given in ProcBuildWithDerivsGP. We can therefore validate an emulator built with derivatives using the same procedure as that which we apply to validate an emulator of the core problem. It is often necessary, in response to the validation diagnostics, to rebuild the emulator using additional training runs which can of course, include derivatives. We hope to extend the validation process using derivatives as we gain more experience in validation diagnostics and emulating with derivative information.

Tasks

Having obtained a working emulator, the MUCM methodology now enables efficient analysis of a number of tasks that regularly face users of simulators.

Prediction

The simplest of these tasks is to use the emulator as a fast surrogate for the simulator, i.e. to predict what output the simulator would produce if run at a new point in the input space. In this thread we are concerned with predicting the function output of the simulator. The prediction of derivatives of the simulator output w.r.t the inputs, at a new point in the input space is covered in the thread ThreadGenericEmulateDerivatives. The process of predicting function output at one or more new points for the core problem is set out in ProcPredictGP. When we have derivatives in the training sample the process of prediction is the same as for the core problem, but anywhere D, t, A, e etc are required, they should be replaced with $\tilde{D}, \tilde{t}, \tilde{A}, \tilde{e}$.

For some of the tasks considered below, we require to predict the output not at a set of discrete points, but in effect the entire output function as the inputs vary over some range. This can be achieved also using simulation, as discussed in the procedure page for simulating realisations of an emulator (ProcSimulationBasedInference).

Uncertainty analysis

Uncertainty analysis is the process of predicting the simulator output when one or more of the inputs are uncertain. The procedure page on uncertainty analysis using a GP emulator (ProcUAGP) explains how this is done for the core problem. We hope to extend this procedure to cover an emulator built with derivative information in a later release of the toolkit.

Sensitivity analysis

In sensitivity analysis the objective is to understand how the output responds to changes in individual inputs or groups of inputs. The procedure page ProcVarSAGP gives details of carrying out variance based sensitivity analysis for the core problem. We hope to extend this procedure to cover an emulator built with derivative information in a later release of the toolkit.

Examples

One dimensional example

Additional Comments, References, and Links

If we are interested in emulating multiple outputs of a simulator, there are various approaches to this discussed in the alternatives page AltMultipleOutputsApproach. If the approach chosen is to build a multivariate GP emulator and derivatives are available, then they can be included using the methods described in this page combined with the methods described in the thread for the analysis of a simulator with multiple outputs (ThreadVariantMultipleOutputs). A variant thread on multiple outputs with derivatives (ThreadVariantMultipleOutputsWithDerivatives) page may be included in a later release of the toolkit.

A.2.2 ProcBuildWithDerivsGP

This page describes the procedure required to build a Gaussian process emulator with derivative information. This page can be found at:

http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page= ProcBuildWithDerivsGP.html.

Procedure: Build Gaussian process emulator with derivative information

Description and Background

The preparation for building a Gaussian process (GP) emulator with derivative information involves defining the prior mean and covariance functions, identifying prior distributions for hyperparameters, creating a design for the training sample, then running the adjoint, or simulator and a method to obtain derivatives, at the input configurations specified in the design. All of this is described in the variant thread on emulators with derivative information (ThreadVariantWithDerivatives). The procedure here is for taking those various ingredients and creating the GP emulator.

Additional notation for this page

Including derivative information requires further notation than is specified in the Toolkit notation page (MetaNotation). We declare this notation here but it is only applicable to the derivatives thread variant pages.

- The tilde symbol ($\tilde{}$) placed over a letter denotes derivative information and function output combined.
- We introduce an extra argument to denote a derivative. We define $\tilde{f}(x,d)$ to be the derivative of $f(x)$ with respect to input d and so $d \in \{0,1,\ldots,p\}$. When $d=0$ we have $\tilde{f}(x,0)=f(x)$. For simplicity, when $d=0$ we adopt the shorter notation so we use $f(x)$ rather than $\tilde{f}(x,0)$.
- An input is denoted by a superscript on x, while a subscript on x refers to the point in the input space. For example, $x_i^{(k)}$ refers to input k at point i.

Inputs

- GP prior mean function $m(\cdot)$, differentiable and depending on hyperparameters β
- GP prior correlation function $c(\cdot,\cdot)$, twice differentiable and depending on hyperparameters δ
- Prior distribution $\pi(\cdot,\cdot,\cdot)$ for β,σ^2 and δ where σ^2 is the process variance hyperparameter
- We require a design. In the core thread ThreadCoreGP the design, D, is an ordered set of points $D=\{x_1,x_2,\ldots,x_n\}$, where each x is a location in the input space. Here, we need a design which in addition to specifying the location of the inputs, also determines at which points we require function output and at which points we require first derivatives. We arrange this information in the design $\tilde{D}=\{(x_k,d_k)\}$, where $k=\{1,\ldots,\tilde{n}\}$ and $d_k \in \{0,1,\ldots,p\}$. We have x_k which refers to the location in the design and d_k determines whether at point x_k we require function output or a first derivative w.r.t one of the inputs. Each x_k is not distinct as we may have a derivative and the function output at point x_k or we may require a derivative w.r.t several inputs at point x_k.
- Output vector is $\tilde{f}(\tilde{D})$ of length $\tilde{n}$.

Outputs

A GP-based emulator in one of the forms discussed in DiscGPBasedEmulator.

In the case of general prior mean and correlation functions and general prior distribution:

- A GP posterior conditional distribution with mean function $m^*(\cdot)$ and covariance function $v^*(\cdot,\cdot)$ conditional on $\theta=\{\beta,\sigma^2,\delta\}$. If we want to emulate the derivatives rather than the function output see ProcBuildGPEmulateDerivs.
- A posterior representation for θ

In the case of linear mean function, general correlation function, weak prior information on β,σ^2 and general prior distribution for δ:

- A t process posterior conditional distribution with mean function $m^*(\cdot)$, covariance function $v^*(\cdot,\cdot)$ and degrees of freedom b^* conditional on δ
- A posterior representation for δ

As explained in DiscGPBasedEmulator, the "posterior representation" for the hyperparameters is formally the posterior distribution for those hyperparameters, but for computational purposes this distribution is represented by a sample of hyperparameter values. In either case, the outputs define the emulator and allow all necessary computations for tasks such as prediction of the simulator output, uncertainty analysis or sensitivity analysis.

Procedure

General case

We define the following arrays (following the conventions set out in MetaNotation where possible).

$\tilde{e} = \tilde{f}(\tilde{D}) - \tilde{m}(\tilde{D})$, an $\tilde{n} \times 1$ vector, where $\tilde{m}(x,0) = m(x)$, and $\tilde{m}(x,d) = \frac{\partial}{\partial x^{(d)}} m(x)$ if $d \neq 0$.

$\tilde{A} = \tilde{c}(\tilde{D},\tilde{D})$, an $\tilde{n} \times \tilde{n}$ matrix, where the exact form of $\tilde{c}(.,.)$ depends on where derivatives are included. The general expression for this is: $\tilde{c}(.,.) = \mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,d_j)\}$ and we can break it down into three cases:

- Case 1 is for when $d_i = d_j = 0$ and as such represents the covariance between 2 points. This is the same as in ThreadCoreGP and is given by:
$$\mathrm{Corr}\{\tilde{f}(x_i,0),\tilde{f}(x_j,0)\} = c(x_i,x_j).$$
- Case 2 is for when $d_i \neq 0$ and $d_j = 0$ and as such represents the covariance between a derivative and a point. This is obtained by differentiating $c(.,.)$ w.r.t input d_i:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,0)\} = \frac{\partial c(x_i,x_j)}{\partial x_i^{(d_i)}}, \text{for } d_i \neq 0.$$
- Case 3 is for when $d_i \neq 0$ and $d_j \neq 0$ and as such represents the covariance between two derivatives. This is obtained by differentiating $c(.,.)$ twice: once w.r.t input d_i and once w.r.t input d_j:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_j),\tilde{f}(x_j,d_j)\} = \frac{\partial^2 c(x_i,x_j)}{\partial x_i^{(d_i)}\partial x_j^{(d_j)}}, \text{for } d_i,d_j \neq 0$$
.
    - Case 3a. If $d_i,d_j \neq 0$ and $d_i = d_j$ we have a special version of Case 3 which gives:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,d_i)\} = \frac{\partial^2 c(x_i,x_j)}{\partial x_i^{(d_i)}, x_j^{(d_i)}}, \text{for } d_i \neq 0.$$

$\tilde{t}(x) = \tilde{c}\{\tilde{D},(x,0)\}$, an $\tilde{n} \times 1$ vector function of x. We have $d = 0$ as here we want to emulate function output. To emulate derivatives, $d \neq 0$ and this is covered in the generic thread on emulating derivatives (ThreadGenericEmulateDerivatives).

Then, conditional on θ and the training sample, the output vector $\tilde{f}(x,0) = f(x)$ is a multivariate GP with posterior mean function

$$m^*(x) = m(x) + \tilde{t}(x)^{\mathrm{T}} \tilde{A}^{-1}\tilde{e}$$

and posterior covariance function

$$v^*(x_i,x_j) = \sigma^2\{c(x_i,x_j) - \tilde{t}(x_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{t}(x_j)\}\,.$$

This is the first part of the emulator as discussed in DiscGPBasedEmulator. The emulator is completed by a second part formally comprising the posterior distribution of θ, which has density given by

$$\pi^*(\beta,\sigma^2,\delta) \propto \pi(\beta,\sigma^2,\delta) \times (\sigma^2)^{-\tilde{n}/2}|\tilde{A}|^{-1/2} \times \exp\{-\tilde{e}^{\mathrm{T}}\tilde{A}^{-1}\tilde{e}/(2\sigma^2)\}\,.$$

For the output vector $\tilde{f}(x,d)$ with $d \neq 0$ see the procedure page on building an emulator of derivatives (ProcBuildEmulateDerivsGP).

Linear mean and weak prior case

Suppose now that the mean function has the linear form $m(x) = h(x)^{\mathrm{T}}\beta$, where $h(\cdot)$ is a vector of q known basis functions of the inputs and β is a $q \times 1$ column vector of hyperparameters. When $d \neq 0$ we therefore have $\tilde{m}(x,d) = \tilde{h}(x,d)^{\mathrm{T}}\beta = \frac{\partial}{\partial x^{(d)}} h(x)^{\mathrm{T}}\beta$. Suppose also that the prior distribution has the form $\pi(\beta,\sigma^2,\delta) \propto \sigma^{-2}\pi_\delta(\delta)$, i.e. that we have weak prior information on β and σ^2 and an arbitrary prior distribution $\pi_\delta(\cdot)$ for δ.

Define $\tilde{A}$ and $\tilde{t}(x)$ as in the previous case. In addition, define the $\tilde{n} \times q$ matrix

$$\tilde{H} = [\tilde{h}(x_1,d_1),\ldots,\tilde{h}(x_{\tilde{n}},d_{\tilde{n}})]^{\mathrm{T}},$$

the vector

$$\widehat{\beta} = \left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{f}(\tilde{D})$$

and the scalar

$$\widehat{\sigma}^2 = (\tilde{n}-q-2)^{-1}\tilde{f}(\tilde{D})^{\mathrm{T}}\left\{\tilde{A}^{-1} - \tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\right\}\tilde{f}(\tilde{D})\,.$$

Then, conditional on δ and the training sample, the output vector $\tilde{f}(x,0) = f(x)$ is a t process with $b^* = \tilde{n} - q$ degrees of freedom, posterior mean function

$$m^*(x) = h(x)^{\mathrm{T}}\widehat{\beta} + \tilde{t}(x)^{\mathrm{T}}\tilde{A}^{-1}(\tilde{f}(\tilde{D}) - \tilde{H}\widehat{\beta})$$

and posterior covariance function

$$v^*(x_i,x_j) = \widehat{\sigma}^2\{c(x_i,x_j) - \tilde{t}(x_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{t}(x_j) + \left(h(x_i)^{\mathrm{T}} - \tilde{t}(x_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\left(h(x_j)^{\mathrm{T}} - \tilde{t}(x_j)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{\mathrm{T}}\}\,.$$

This is the first part of the emulator as discussed in DiscGPBasedEmulator. The emulator is formally completed by a second part comprising the posterior distribution of δ, which has density given by

$$\pi_\delta^*(\delta) \propto \pi_\delta(\delta) \times (\widehat{\sigma}^2)^{-(\tilde{n}-q)/2}|\tilde{A}|^{-1/2}|\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}|^{-1/2}\,.$$

In order to derive the sample representation of this posterior distribution for the second part of the emulator, three approaches can be considered.

1. Exact computations require a sample from the posterior distribution of δ. This can be obtained by MCMC; a suitable reference can be found below.
2. A common approximation is simply to fix δ at a single value estimated from the posterior distribution. The usual choice is the posterior mode, which can be found as the value of δ for which $\pi^*(\delta)$ is maximised. See the alternatives page on estimators of correlation hyperparameters (AltEstimateDelta).
3. An intermediate approach first approximates the posterior distribution by a multivariate lognormal distribution and then uses a sample from this distribution, as described in the procedure page ProcApproxDeltaPosterior.

Each of these approaches results in a set of values (or just a single value in the case of the second approach) of δ, which allow the emulator predictions and other required inferences to be computed.

Although it represents an approximation that ignores the uncertainty in δ, approach 2 has been widely used. It has often been suggested that, although uncertainty in these correlation hyperparameters can be substantial, taking proper account of that uncertainty through approach 1 does not lead to appreciable differences in the resulting emulator. On the other hand, although this may be true if a good single estimate for δ is used, this is not necessarily easy to find, and the posterior mode may sometimes be a poor choice. Approach 3 has not been used much, but can be recommended when there is concern about using just a single δ estimate. It is simpler than the full MCMC approach 1, but should capture the uncertainty in δ well.

References

Morris, M. D., Mitchell, T. J. and Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. Technometrics, 35, 243-255.

A.2.3 ExamVariantWithDerivatives1Dim

This page presents a 1-dimensional example where we build an emulator, with and without derivatives, and then attempt to validate the emulator. This page can be found at:

http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page=ExamVariantWithDerivatives1Dim.html.

Example: A one dimensional emulator built with function output and derivatives

Simulator description

In this page we present an example of fitting an emulator to a $p=1$ dimensional simulator when in addition to learning the function output of the simulator, we also learn the partial derivatives of the output w.r.t the input. The simulator we use is the function, $\sin(2x)+(\frac{x}{2})^2$. Although this is not a complex simulator which takes an appreciable amount of computing time to execute, it is still appropriate to use as an example. We restrict the range of the input, x, to lie in $[-5,5]$ and the behaviour of our simulator over this range is shown in Figure 1. The simulator can be analytically differentiated to provide the relevant derivatives: $2\cos(2x)+(\frac{x}{2})$. For the purpose of this example we define the adjoint as the function which when executed returns both the simulator output and the derivative of the simulator output w.r.t the simulator input.

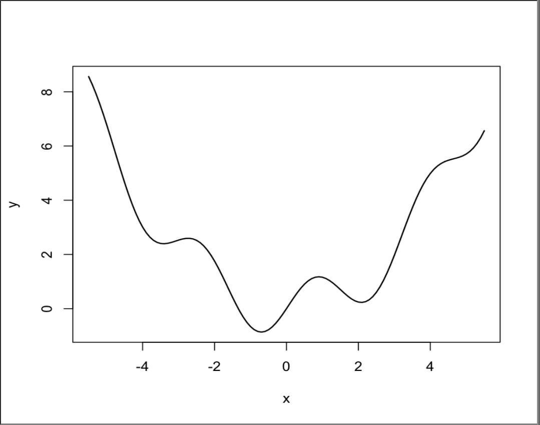


Figure 1: The simulator over the specified region

Design

We need to choose a design to select at which input points the simulator is to be run and at which points we want to include derivative information. An Optimised Latin Hypercube, which for one dimension is a set of equidistant points on the space of the input variable, is chosen. We select the following 5 design points:

$D=[-5,-2.5,0,2.5,5]$.

We choose to evaluate the function output and the derivative at each of the 5 points and scale our design points to lie in $[0,1]$. This information is added to D resulting in the design, $\tilde{D}$ of length $\tilde{n}$. A point in $\tilde{D}$ has the form (x,d), where d denotes whether a derivative or the function output is to be included at that point; for example $(x,d)=(x,0)$ denotes the function output at point x and $(x,d)=(x,1)$ denotes the derivative w.r.t to the first input at point x.

This results in the following design:

$\tilde{D}=[(x_1,d_1),(x_2,d_2),\cdots,(x_{10},d_{10})]=[(0,0),(0.25,0),(0.5,0),(0.75,0),(1,0),(0,1),(0.25,1),(0.5,1),(0.75,1),(1,1)]$

The output of the adjoint at these points, which make up our training data, is:

$\tilde{f}(\tilde{D})=[6.794,2.521,0,0.604,5.706,-41.78,-6.827,20,18.17,8.219]^{\mathrm{T}}$.

Note that the first 5 elements of $\tilde{D}$, and therefore $\tilde{f}(\tilde{D})$, are simply the same as in the core problem. Elements $6-10$ correspond to the derivatives. Throughout this example n will refer to the number of distinct locations in the design (5), while $\tilde{n}$ refers to the total amount of data points in the training sample. Since we include a derivative at all n points w.r.t to the $p=1$ input, $\tilde{n}=n(p+1)=10$.

Gaussian process setup

In setting up the Gaussian process, we need to define the mean and the covariance function. As we are including derivative information here we need to ensure that the mean function is once differentiable and the covariance function is twice differentiable. For the mean function, we choose the linear form described in the alternatives page on emulator prior mean function (AltMeanFunction), which is $h(x)=[1,x]^{\mathrm{T}}$ and $q=1+p=2$. This corresponds to $\tilde{h}(x,0)=[1,x]^{\mathrm{T}}$ and we also have $\frac{\partial}{\partial x}h(x)=[1]$ and so $\tilde{h}(x,1)=[1]$.

For the covariance function we choose $\sigma^2c(\cdot,\cdot)$, where the correlation function $c(\cdot,\cdot)$ has the Gaussian form described in the alternatives page on emulator prior correlation function (AltCorrelationFunction). We can break this down into 3 cases, as described in the procedure page for building a GP emulator with derivative information

(ProcBuildWithDerivsGP):

Case 1 is the correlation between points, so we have $d_i = d_j = 0$ and $c\{(x_i,0),(x_j,0)\} = \exp\left[-\{(x_i - x_j)/\delta\}^2\right]$.

Case 2 is the correlation between a point, x_i, and the derivatives at point x_j, so we have $d_i = 0$ and $d_j \neq 0$. Since in this example $p = 1$, this amounts to $d_j = 1$ and we have:

$$\frac{\partial}{\partial x_j} c\{(x_i,0),(x_j,1)\} = \frac{2}{\delta^2}\,(x_i - x_j)\,\exp\left[-\{(x_i - x_j)/\delta\}^2\right]$$

.

Case 3 is the correlation between two derivatives at points x_i and x_j. Since in this example $p = 1$, the only relevant correlation here is when $d_i = d_j = 1$ (which corresponds to Case 3a in ProcBuildWithDerivsGP) and is given by:

$$\frac{\partial^2}{\partial x_i \partial x_j} c\{(x_i,1),(x_j,1)\} = \left(\frac{2}{\delta^2} - \frac{4\,(x_i - x_j)^2}{\delta^4}\right) \exp\left[-\{(x_i - x_j)/\delta\}^2\right].$$

Each Case provides sub-matrices of correlations and we arrange them as follows:

$$\tilde{A} = \left(\begin{array}{c|c} \text{Case 1} & \text{Case 2} \\ \hline \text{Case 2} & \text{Case 3} \end{array}\right),$$

an $\tilde{n} \times \tilde{n}$ matrix. The matrix $\tilde{A}$ is symmetric and within $\tilde{A}$ we have symmetric sub-matrices, Case 1 and Case 3. Case 1 is an $n \times n = 5 \times 5$ matrix and is exactly the same as in the procedure page ProcBuildCoreGP. Since we are including the derivative at each of the 5 design points, Case 2 and 3 sub-matrices are also of size $n \times n = 5 \times 5$.

Estimation of the correlation length

We need to estimate the correlation length δ. In this example we will use the value of δ that maximises the posterior distribution $\pi^*_\delta(\delta)$, assuming that there is no prior information on δ, i.e. $\pi(\delta) \propto \mathrm{const}$. The expression that needs to be maximised is (from ProcBuildWithDerivsGP)

$$\pi^*_\delta(\delta) \propto (\widehat{\sigma}^2)^{-(\tilde{n}-q)/2} |\tilde{A}|^{-1/2} |\tilde{H}^{\mathrm{T}} \tilde{A}^{-1} \tilde{H}|^{-1/2}.$$

where

$$\widehat{\sigma}^2 = (\tilde{n} - q - 2)^{-1} \tilde{f}(\tilde{D})^{\mathrm{T}} \left\{\tilde{A}^{-1} - \tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\right\} \tilde{f}(\tilde{D}).$$

We have $\tilde{H} = [\tilde{h}(x_1,d_1),\ldots,\tilde{h}(x_{10},d_{10})]^{\mathrm{T}}$, where $\tilde{h}(x,d)$ and $\tilde{A}$ are defined above in section Gaussian process setup.

Recall that in the above expressions the only term that is a function of δ is the correlation matrix $\tilde{A}$.

The maximum can be obtained with any maximisation algorithm and in this example we used Nelder - Mead. The value of δ which maximises the posterior is 0.183 and we will fix δ at this value thus ignoring the uncertainty with it, as discussed in ProcBuildCoreGP. We refer to this value of δ as $\hat{\delta}$. We have scaled the input to lie in [0,1] and so in terms of the original input scale, $\hat{\delta}$ corresponds to a smoothness parameter of 10 x 0.183 = 1.83

Estimates for the remaining parameters

The remaining parameters of the Gaussian process are β and σ^2. We assume weak prior information on β and σ^2 and so having estimated the correlation length, the estimate for σ^2 is given by the equation above in section Estimation of the correlation length, and the estimate for β is

$$\hat{\beta} = \left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{f}(\tilde{D}).$$

Note that in these equations, the matrix $\tilde{A}$ is calculated using $\hat{\delta}$. The application of the two equations for $\hat{\beta}$ and $\widehat{\sigma}^2$, gives us in this example $\hat{\beta} = [4.734, -2.046]^{\mathrm{T}}$ and $\widehat{\sigma}^2 = 15.47$

Posterior mean and Covariance functions

The expressions for the posterior mean and covariance functions as given in ProcBuildWithDerivsGP are

$$m^*(x) = h(x)^{\mathrm{T}}\widehat{\beta} + \tilde{t}(x)^{\mathrm{T}}\tilde{A}^{-1}(\tilde{f}(\tilde{D}) - \tilde{H}\widehat{\beta})$$

and

$$v^*(x_i,x_j) = \widehat{\sigma}^2\{c(x_i,x_j) - \tilde{t}(x_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{t}(x_j) + \left(h(x_i)^{\mathrm{T}} - \tilde{t}(x_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)\left(\tilde{H}^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{-1}\left(h(x_j)^{\mathrm{T}} - \tilde{t}(x_j)^{\mathrm{T}}\tilde{A}^{-1}\tilde{H}\right)^{\mathrm{T}}\}.$$

Figure 2 shows the predictions of the emulator for 100 points uniformly spaced on the original scale. The solid, black line is the output of the simulator and the blue, dashed line is the emulator mean m^* evaluated at each of the 100 points. The blue dotted lines represent 2 times the standard deviation about the emulator mean, which is the square root of the diagonal of matrix v^*. The black crosses show the location of the design points where we have evaluated the function output and the derivative to make up the training sample. The green circles show the location of the validation data which is discussed in the section below. We can see from Figure 2 that the emulator mean is very close to the true simulator output and the uncertainty decreases as we get closer the location of the design points.

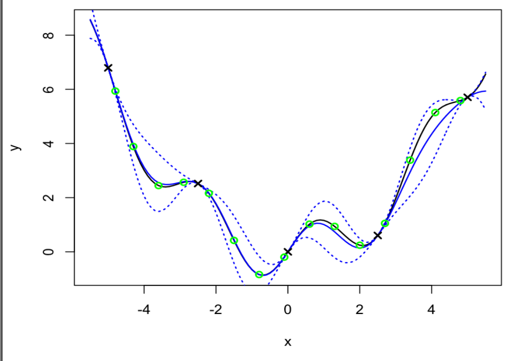


Figure 2: The simulator (solid black line), the emulator mean (blue, dotted) and 95% confidence intervals shown by the blue, dashed line. Black crosses are design points, green circles are validation points.

Validation

In this section we validate the above emulator according to the procedure page for validating a GP emulator (ProcValidateCoreGP).

The first step is to select the validation design. We choose here 15 space filling points ensuring these points are distinct from the design points. The validation points are shown by green circles in Figure 2 above and in the original input space of the simulator are:

$$[-4.8, -4.3, -3.6, -2.9, -2.2, -1.5, -0.8, -0.1, 0.6, 1.3, 2, 2.7, 3.4, 4.1, 4.8]$$

and in the transformed space:

$$D' = [x'_1, x'_2, \cdots, x'_{15}] = [0.02, 0.07, 0.14, 0.21, 0.28, 0.35, 0.42, 0.49, 0.56, 0.63, 0.7, 0.77, 0.84, 0.91, 0.98].$$

Note that the prime symbol ($'$) does not denote a derivative, as in ProcValidateCoreGP we use the prime symbol to specify validation. We're predicting function output in this example and so do not need validation derivatives; as such we have a validation design D' and not $\tilde{D}'$. The function output of the simulator at these validation points is

$$f(D') = [5.934, 3.888, 2.446, 2.567, 2.161, 0.421, -0.840, -0.196, 0.102, 0.938, 0.243, 1.050, 3.384, 5.143, 5.586]^{\mathrm{T}}$$

We then calculate the mean $m^*(\cdot)$ and variance $v^*(\cdot,\cdot)$ of the emulator at each validation design point in D' and the difference between the emulator mean and the simulator output at these points can be compared in Figure 2.

We also calculate standardised errors given in ProcValidateCoreGP as $\frac{f(x'_j)-m^*(x'_j)}{\sqrt{v^*(x'_j,x'_j)}}$ and plot them in Figure 3.

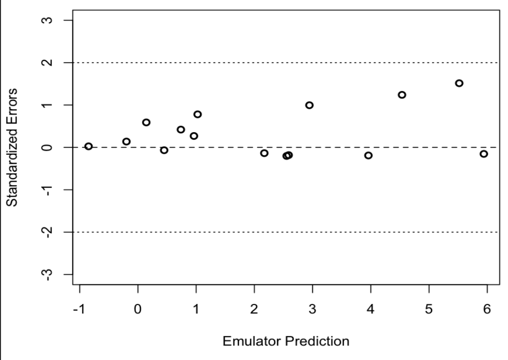


Figure 3: Individual standardised errors for the prediction at the validation points

Figure 3 shows that all the standardised errors lie between -2 and 2 providing no evidence of conflict between simulator and emulator.

We calculate the Mahalanobis distance as given in ProcValidateCoreGP:

$$M = (f(D') - m^*(D'))^{\mathrm{T}}(v^*(D', D'))^{-1}(f(D') - m^*(D')) = 6.30$$

when its theoretical mean is

$\mathrm{E}[M] = n' = 15$ and variance, $\mathrm{Var}[M] = \frac{2n'(n'+\tilde{n}-q-2)}{\tilde{n}-q-4} = 12.55^2$

We have a slightly small value for the Mahalanobis distance therefore, but it is within one standard deviation of the theoretical mean. The validation sample is small and so we would only expect to detect large problems with this test. This is just an example and we would not expect a simulator of a real problem to only have one input, but with our example we can afford to run the simulator intensely over the specified input region. This allows us to assess the overall performance of the emulator and, as Figure 2 shows, the emulator can be declared as valid.

Comparison with an emulator built with function output alone

We now build an emulator for the same simulator with all the same assumptions, but this time leave out the derivative information to investigate the effect of the derivatives and compare the results.

We obtain the following estimates for the parameters:

$\hat{\delta} = 2.537, \hat{\beta} = [82.06, 54.34]^{\mathrm{T}}$ and $\widehat{\sigma}^2 = 49615$.

Figure 4 shows the predictions of this emulator for 100 points uniformly spaced on the original scale. The solid, black line is the output of the simulator and the red, dashed line is the emulator mean evaluated at each of the 100 points. The red dotted lines represent 2 times the standard deviation about the emulator mean. The black crosses show the location of the design points where we have evaluated the function output. We can see from Figure 4 that the emulator is not capturing the behaviour of the simulator at all and further simulator runs are required.

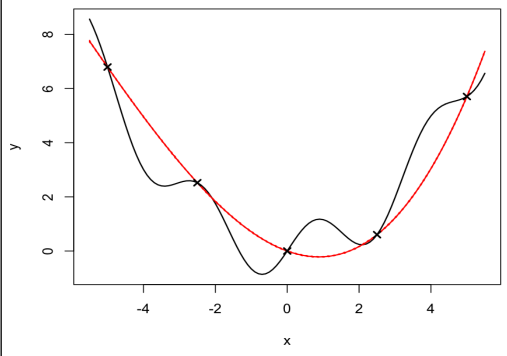


Figure 4: The simulator (solid black line), the emulator mean (red, dotted) and 95% confidence intervals shown by the red, dashed line. Black crosses are design points.

We add 4 further design points, $[-3.75, -1.25, 1.25, 3.75]$, and rebuild the emulator, without derivatives as before. This results in new estimates for the parameters, $\hat{\delta} = 0.177, \hat{\beta} = [4.32, -2.07]^{\mathrm{T}}$ and $\widehat{\sigma}^2 = 15.44$, and Figure 5 shows the predictions of this emulator for the same 100 points. We now see that the emulator mean closely matches the simulator output across the specified range.

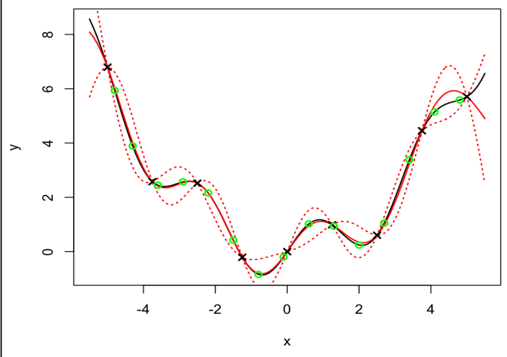


Figure 5: The simulator (solid black line), the emulator mean (red, dotted) and 95% confidence intervals shown by the red, dashed line. Black crosses are design points, green circles are validation points.

We repeat the validation diagnostics using the same validation data and obtain a Mahalanobis distance of 4.70, while the theoretical mean is 15 with standard deviation 14.14. As for the emulator with derivatives, a value of 4.70 is bit small; however the standardised errors calculated as before, and shown in Figure 6 below, provide no evidence of conflict between simulator and emulator and the overall performance of the emulator as illustrated in Figure 5 is satisfactory.

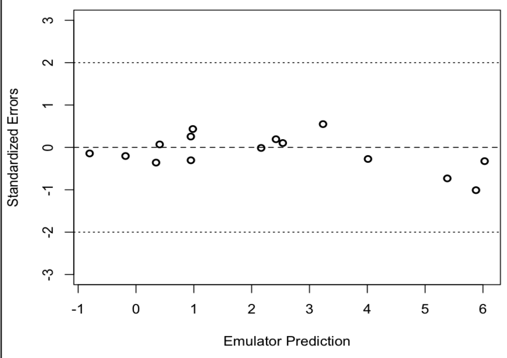


Figure 6: Individual standardised errors for the prediction at the validation points

We have therefore now built a valid emulator without derivatives but required 4 extra simulator runs to the emulator with derivatives, to achieve this.

A.3 Emulating derivatives

In this section we present the pages which describe how to emulate derivatives either with or without derivative information in the training data.

A.3.1 ThreadGenericEmulateDerivatives

This page describes the ingredients required to build an emulator of derivatives. This page can be found at:
http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page= ThreadGenericEmulateDerivatives.html.

Thread: Generic methods to emulate derivatives

Overview

This thread describes how we can build an emulator with which we can predict the derivatives of the model output with respect to the inputs. If we have derivative information available, either from an adjoint model or some other means, we can include that information when emulating derivatives. This is similar to the variant thread on emulators with derivative information (ThreadVariantWithDerivatives) which includes derivative information when emulating function output. If the adjoint to a simulator doesn't exist and we don't wish to obtain derivative information through another method, it is still possible to emulate model derivatives with just the function output.

The emulator

The derivatives of a posterior Gaussian process remain Gaussian processes with mean and covariance functions obtained by the relevant derivatives of the posterior mean and covariance functions. This can be applied to any Gaussian process emulator. The process of building an emulator of derivatives with the fully Bayesian approach is given in the procedure page ProcBuildEmulateDerivsGP. This covers building a Gaussian process emulator of derivatives with just function output, an extension of the core thread ThreadCoreGP, and a Gaussian process emulator of derivatives built with function output and derivative information, an extension of ThreadVariantWithDerivatives.

The result is a Gaussian process emulator of derivatives which will correctly represent any derivatives in the training data, but it is always important to validate the emulator against additional derivative information. For the core problem, the process of validation is described in the procedure page ProcValidateCoreGP. Although here we are interested in emulating derivatives, as we know the derivatives of a Gaussian process remain a Gaussian process, we can apply the same validation techniques as for the core problem. We require a validation design D' which consists of points where we want to obtain validation derivatives. An adjoint is then run at these points; if an appropriate adjoint does not exist the derivatives are obtained through another technique, for example finite differences. If any local sensitivity analysis has already been performed on the simulator, some derivatives may already have been obtained and can be used here for validation. Then in the case of a linear mean function, weak prior information on hyperparameters β and σ, and a single posterior estimate of δ, the predictive mean vector, m^*, and the predictive covariance matrix, V^*, required in ProcValidateCoreGP, are given by the functions $\tilde{m}^*(\cdot)$ and $\tilde{v}^*(\cdot,\cdot)$ which are given in ProcBuildEmulateDerivsGP. We can therefore validate an emulator of derivatives using the same procedure as that which we apply to validate an emulator of the core problem. It is often necessary, in response to the validation diagnostics, to rebuild the emulator using additional training runs which can of course, include derivatives. We hope to extend the validation process using derivatives as we gain more experience in validation diagnostics and emulating with derivative information.

The Bayes linear approach to emulating derivatives may be covered in a future release of the toolkit.

Tasks

Having obtained a working emulator, the MUCM methodology now enables efficient analysis of a number of tasks that regularly face users of simulators.

Prediction

The simplest of these tasks is to use the emulator as a fast surrogate for the adjoint, i.e. to predict what derivatives the adjoint would produce if run at a new point in the input space. The process of predicting function output at one or more new points for the core problem is set out in the prediction page ProcPredictGP. Here we are predicting derivatives and the process of prediction is the same as for the core problem. If the procedure in ProcBuildEmulateDerivsGP is followed, $\tilde{D}, \tilde{t}, \tilde{A}, \tilde{e}$ etc are used in replace of D, t, A, e, as required in ProcPredictGP.

Sensitivity analysis

In sensitivity analysis the objective is to understand how the output responds to changes in individual inputs or groups of inputs. Local sensitivity analysis uses derivatives to study the effect on the output, when the inputs are perturbed by a small amount. Emulated derivatives could replace adjoint produced derivatives in this analysis if the adjoint is too expensive to execute or in fact does not exist.

Other tasks

Derivatives can be informative in optimization problems. If we want to find which sets of input values results in either a maximum or a minimum output then knowledge of the gradient of the function, with respect to the inputs, may result in a more efficient search. Derivative information is also useful in data assimilation.

A.3.2 ProcBuildEmulateDerivsGP

This page describes the procedure required to build a Gaussian process emulator of derivatives. This page can be found at:

http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page= ProcBuildEmulateDerivsGP.html.

MUCM Toolkit

Managing Uncertainty in Complex Models

Procedure: Build Gaussian process emulator of derivatives

Description and Background

The preparation for building a Gaussian process (GP) emulator of derivatives involves defining the prior mean and covariance functions, identifying prior distributions for hyperparameters, creating a design for the training sample, then running the adjoint, or simulator, at the input configurations specified in the design. This is described in the generic thread on methods to emulate derivatives (ThreadGenericEmulateDerivatives). The procedure here is for taking those various ingredients and creating the GP emulator.

Additional notation for this page

Derivative information requires further notation than is specified in the Toolkit notation page (MetaNotation). As in the procedure page on building a GP emulator with derivative information (ProcBuildWithDerivsGP) we use the following additional notation:

- The tilde symbol ($\sim$) placed over a letter denotes derivative information and function output combined.
- We introduce an extra argument to denote a derivative. We define $\tilde{f}(x,d)$ to be the derivative of $f(x)$ with respect to input d and so $d \in \{0,1,\ldots,p\}$. When $d=0$ we have $\tilde{f}(x,0)=f(x)$. For simplicity, when $d=0$ we adopt the shorter notation so we use $f(x)$ rather than $\tilde{f}(x,0)$.
- An input is denoted by a superscript on x, while a subscript on x refers to the point in the input space. For example, $x_i^{(k)}$ refers to input k at point i.

Inputs

- GP prior mean function $m(\cdot)$, differentiable and depending on hyperparameters β
- GP prior correlation function $c(\cdot,\cdot)$, twice differentiable and depending on hyperparameters δ
- Prior distribution $\pi(\cdot,\cdot,\cdot)$ for β,σ^2 and δ where Σ is the process variance hyperparameter
- Design, $\tilde{D}=\{(x_k,d_k)\}$, where $k=\{1,\ldots,\tilde{n}\}$ and $d_k \in \{0,1,\ldots,p\}$. We have x_k which refers to the location in the design and d_k determines whether at point x_k we require function output or a first derivative w.r.t one of the inputs. Each x_k is not necessarily distinct as we may have a derivative and the function output at point x_k or we may require a derivative w.r.t several inputs at point x_k. If we do not have any derivative information, $d_k=0,\forall k$ and the resulting design is as in the core thread ThreadCoreGP.
- Output vector is $\tilde{f}(\tilde{D})=\tilde{f}(x_k,d_k)$ of length $\tilde{n}$. If we are not including derivatives in the training data, $d_k=0,\forall k$ and the output vector reduces to $f(D)=f(x)$ as in ThreadCoreGP.

Outputs

A GP-based emulator in one of the forms discussed in the discussion page DiscGPBasedEmulator.

In the case of general prior mean and correlation functions and general prior distribution:

- A GP posterior conditional distribution with mean function $\tilde{m}^*(\cdot)$ and covariance function $\tilde{v}^*(\cdot,\cdot)$ conditional on $\theta=\{\beta,\sigma^2,\delta\}$.
- A posterior representation for θ

In the case of linear mean function, general correlation function, weak prior information on β,σ^2 and general prior distribution for δ:

- A t process posterior conditional distribution with mean function $\tilde{m}^*(\cdot)$, covariance function $\tilde{v}^*(\cdot,\cdot)$ and degrees of freedom b^* conditional on δ
- A posterior representation for δ

As explained in DiscGPBasedEmulator, the "posterior representation" for the hyperparameters is formally the posterior distribution for those hyperparameters, but for computational purposes this distribution is represented by a sample of hyperparameter values. In either case, the outputs define the emulator and allow all necessary computations for tasks such as prediction of the partial derivatives of the simulator output w.r.t the inputs, uncertainty analysis or sensitivity analysis.

Procedure

General case

We define the following arrays (following the conventions set out in MetaNotation where possible).

$\tilde{e}=\tilde{f}(\tilde{D})-\tilde{m}(\tilde{D})$, an $\tilde{n}\times 1$ vector, where $\tilde{m}(\tilde{D})=\frac{\partial}{\partial x^{(d_k)}}m(x_k)$.

$\tilde{A}=\tilde{c}(\tilde{D},\tilde{D})$, an $\tilde{n}\times\tilde{n}$ matrix, where $\tilde{c}(.,.)$ includes the covariances involving derivatives. The exact form of $\tilde{c}(.,.)$ depends on where derivatives are included. The general expression for this is: $\tilde{c}(.,.)=\mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,d_j)\}$ and we can break it down into three cases:

- Case 1 is for when $d_i=d_j=0$ and as such represents the covariance between 2 points. This is the same as in ThreadCoreGP and is given by:
$$\mathrm{Corr}\{\tilde{f}(x_i,0),\tilde{f}(x_j,0)\}=c(x_i,x_j).$$
- Case 2 is for when $d_i\neq 0$ and $d_j=0$ and as such represents the covariance between a derivative and a point. This is obtained by differentiating $c(.,.)$ w.r.t input d_i:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,0)\}=\frac{\partial c(x_i,x_j)}{\partial x_i^{(d_i)}},\mathrm{for}\ d_i\neq 0.$$
- Case 3 is for when $d_i\neq 0$ and $d_j\neq 0$ and as such represents the covariance between two derivatives. This is obtained by differentiating $c(.,.)$ twice: once w.r.t input d_i and once w.r.t input d_j:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_j),\tilde{f}(x_j,d_j)\}=\frac{\partial^2 c(x_i,x_j)}{\partial x_i^{(d_i)}\partial x_j^{(d_j)}},\mathrm{for}\ d_i,d_j\neq 0$$
.
    - Case 3a. If $d_i,d_j\neq 0$ and $d_i=d_j$ we have a special version of Case 3 which gives:
$$\mathrm{Corr}\{\tilde{f}(x_i,d_i),\tilde{f}(x_j,d_i)\}=\frac{\partial^2 c(x_i,x_j)}{\partial x_i^{(d_i)},x_j^{(d_i)}},\mathrm{for}\ d_i\neq 0.$$

$\tilde{t}(x,d)=\tilde{c}\{\tilde{D},(x,d)\}$, an $\tilde{n}\times 1$ vector function of x. We have $d\neq 0$ as here we want to emulate derivatives. To emulate function output, $d=0$ and this is covered in ThreadCoreGP or ThreadVariantWithDerivatives if we have derivatives in the training data.

Then, conditional on θ and the training sample, the output vector $\tilde{f}(x,d)$ is a multivariate GP with posterior mean function

$\tilde{m}^*(x,d)=\tilde{m}(x,d)+\tilde{t}(x,d)^{\mathrm{T}}\tilde{A}^{-1}\tilde{e}$

and posterior covariance function

$\tilde{v}^*\{(x_i,d_i),(x_j,d_j)\}=\sigma^2\{\tilde{c}\{(x_i,d_i),(x_j,d_j)\}-\tilde{t}(x_i,d_i)^{\mathrm{T}}\tilde{A}^{-1}\tilde{t}(x_j,d_j)\}$.

This is the first part of the emulator as discussed in DiscGPBasedEmulator. The emulator is completed by a second part formally comprising the posterior distribution of θ, which has density given by

$$\pi^*(\beta,\sigma^2,\delta) \propto \pi(\beta,\sigma^2,\delta) \times (\sigma^2)^{-\tilde{n}/2}|\tilde{A}|^{-1/2} \times \exp\{-\tilde{e}^{\rm T}\tilde{A}^{-1}\tilde{e}/(2\sigma^2)\}\,.$$

For the output vector $\tilde{f}(x,0)=f(x)$ see the procedure page on building a GP emulator for the core problem (ProcBuildCoreGP) or the procedure page for building a GP emulator when we have derivatives in the training data (ProcBuildWithDerivsGP).

Linear mean and weak prior case

Suppose now that the mean function has the linear form $m(x)=h(x)^{\rm T}\beta$, where $h(\cdot)$ is a vector of q known basis functions of the inputs and β is a $q\times 1$ column vector of hyperparameters. When $d\neq 0$ we therefore have $\tilde{m}(x,d)=\tilde{h}(x,d)^{\rm T}\beta=\frac{\partial}{\partial x^{(d)}}h(x)^{\rm T}\beta$. Suppose also that the prior distribution has the form $\pi(\beta,\Sigma,\delta)\propto\sigma^{-2}\pi_\delta(\delta)$, i.e. that we have weak prior information on β and Σ and an arbitrary prior distribution $\pi_\delta(\cdot)$ for δ.

Define $\tilde{A}$ and $\tilde{t}(x)$ as in the previous case. In addition, define the $\tilde{n}\times q$ matrix

$$\tilde{H}=[\tilde{h}(x_1,d_1),\ldots,\tilde{h}(x_{\tilde{n}},d_{\tilde{n}})]^{\rm T},$$

the vector

$$\widehat{\beta}=\left(\tilde{H}^{\rm T}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\rm T}\tilde{A}^{-1}\tilde{f}(\tilde{D})$$

and the scalar

$$\widehat{\sigma}^2=(\tilde{n}-q-2)^{-1}\tilde{f}(\tilde{D})^{\rm T}\left\{\tilde{A}^{-1}-\tilde{A}^{-1}\tilde{H}\left(\tilde{H}^{\rm T}\tilde{A}^{-1}\tilde{H}\right)^{-1}\tilde{H}^{\rm T}\tilde{A}^{-1}\right\}\tilde{f}(\tilde{D})\,.$$

Then, conditional on δ and the training sample, the output vector $\tilde{f}(x,d)$ is a t process with $b^*=\tilde{n}-q$ degrees of freedom, posterior mean function

$$\tilde{m}^*(x,d)=\tilde{h}(x,d)^{\rm T}\widehat{\beta}+\tilde{t}(x,d)^{\rm T}\tilde{A}^{-1}(\tilde{f}(\tilde{D})-\tilde{H}\widehat{\beta})$$

and posterior covariance function

$$\tilde{v}^*\{(x_i,d_i),(x_j,d_j)\}=\widehat{\sigma}^2\{\tilde{c}\{(x_i,d_i),(x_j,d_j)\}-\tilde{t}(x_i,d_i)^{\rm T}\tilde{A}^{-1}\tilde{t}(x_j,d_j)+\left(\tilde{h}(x_i,d_i)^{\rm T}-\tilde{t}(x_i,d_i)^{\rm T}\tilde{A}^{-1}\tilde{H}\right)\left(\tilde{H}^{\rm T}\tilde{A}^{-1}\tilde{H}\right)^{-1}\left(\tilde{h}(x_j,d_j)^{\rm T}-\tilde{t}(x_j,d_j)^{\rm T}\tilde{A}^{-1}\tilde{H}\right)^{\rm T}\}\,.$$

This is the first part of the emulator as discussed in DiscGPBasedEmulator. The emulator is formally completed by a second part comprising the posterior distribution of δ, which has density given by

$$\pi_\delta^*(\delta)\propto\pi_\delta(\delta)\times(\widehat{\sigma}^2)^{-(\tilde{n}-q)/2}|\tilde{A}|^{-1/2}|\tilde{H}^{\rm T}\tilde{A}^{-1}\tilde{H}|^{-1/2}\,.$$

In order to derive the sample representation of this posterior distribution for the second part of the emulator, three approaches can be considered.

1. Exact computations require a sample from the posterior distribution of δ. This can be obtained by MCMC; a suitable reference can be found below.
2. A common approximation is simply to fix δ at a single value estimated from the posterior distribution. The usual choice is the posterior mode, which can be found as the value of δ for which $\pi^*(\delta)$ is maximised. See the alternatives page AltEstimateDelta for a discussion of alternative estimators.
3. An intermediate approach first approximates the posterior distribution by a multivariate lognormal distribution and then uses a sample from this distribution; this is described in the procedure page ProcApproxDeltaPosterior.

Each of these approaches results in a set of values (or just a single value in the case of the second approach) of δ, which allow the emulator predictions and other required inferences to be computed.

Although it represents an approximation that ignores the uncertainty in δ, approach 2 has been widely used. It has often been suggested that, although uncertainty in these correlation hyperparameters can be substantial, taking proper account of that uncertainty through approach 1 does not lead to appreciable differences in the resulting emulator. On the other hand, although this may be true if a good single estimate for δ is used, this is not necessarily easy to find, and the posterior mode may sometimes be a poor choice. Approach 3 has not been used much, but can be recommended when there is concern about using just a single δ estimate. It is simpler than the full MCMC approach 1, but should capture the uncertainty in δ well.

Additional Comments

We can use this procedure to emulate derivatives whether or not we have derivatives in the training data. Quantities $\tilde{A},\tilde{H},\tilde{f}(\tilde{D}),\tilde{m}(\tilde{D})$ and therefore $\tilde{e}$, above are taken from ProcBuildWithDerivsGP as they allow for derivatives in the training data, in addition to function output. In the case when we build an emulator with function output only, $d=0$ for all the training data and these quantities reduce to the same quantities without the tilde symbol (~), as defined in ProcBuildCoreGP. Then to emulate derivatives in the general case, conditional on θ and the training sample, the output vector $\tilde{f}(x,d)$ is a multivariate GP with posterior mean function

$$\tilde{m}^*(x,d)=\tilde{m}(x,d)+\tilde{t}(x,d)^{\rm T}A^{-1}e$$

and posterior covariance function

$$\tilde{v}^*\{(x_i,d_i),(x_j,d_j)\}=\sigma^2\{\tilde{c}\{(x_i,d_i),(x_j,d_j)\}-\tilde{t}(x_i,d_i)^{\rm T}A^{-1}\tilde{t}(x_j,d_j)\}\,.$$

To emulate derivatives in the case of a linear mean and weak prior, conditional on δ and the training sample, the output vector $\tilde{f}(x,d)$ is a t process with $b^*=n-q$ degrees of freedom, posterior mean function

$$\tilde{m}^*(x,d)=\tilde{h}(x,d)^{\rm T}\widehat{\beta}+\tilde{t}(x,d)^{\rm T}A^{-1}(f(D)-H\widehat{\beta})$$

and posterior covariance function

$$\tilde{v}^*\{(x_i,d_i),(x_j,d_j)\}=\widehat{\sigma}^2\{\tilde{c}\{(x_i,d_i),(x_j,d_j)\}-\tilde{t}(x_i,d_i)^{\rm T}A^{-1}\tilde{t}(x_j,d_j)+\left(\tilde{h}(x_i,d_i)^{\rm T}-\tilde{t}(x_i,d_i)^{\rm T}A^{-1}H\right)\left(H^{\rm T}A^{-1}H\right)^{-1}\left(\tilde{h}(x_j,d_j)^{\rm T}-\tilde{t}(x_j,d_j)^{\rm T}A^{-1}H\right)^{\rm T}\}\,.$$

Appendix B

G-GOLDSTEIN manual

The steps required to build, configure, execute and utilise the C-GOLDSTEIN Earth System Model are described in this Appendix. In addition to the model itself, the configuration tool `mkgoin` is described. This tool can be used to generate multiple configuration files required for computer experiments. This is not an official manual to the C-GOLDSTEIN software and is not intended to be a complete user guide.

B.1 Introduction

B.1.1 Purpose, scope and structure of this document

The purpose of this document is to provide more detailed and up-to-date guidelines on compiling and executing C-GOLDSTEIN, than available in Marsh *et al.* (2002). The reader is assumed to have only a very basic understanding of the Unix environment, in which the model operates, and the FORTRAN language in which the model is coded. Accordingly some knowledge of Earth system modelling is assumed, but only at the most basic level.

This document fundamentally consists of a collection of observations made during attempts to perform the procedures described. It is not a definitive user guide and should not be treated as such. The author did not design, and is not involved in the development of the C-GOLDSTEIN software. All code and guidelines are intended as suggestions only.

This section describes the purpose, scope and structure of this document along with a very brief introduction to the C-GOLDSTEIN software. Section

B.2 details the C-GOLDSTEIN build and installation procedures. Section B.3 contains guidelines on how to run the model as a single run, or in a batch mode using the `mkgoin` tool. A list of input variables is given along with a corresponding description. This is followed by a suggested method for running a spinup and subsequent runs of the model to the year 2000 and into the future. Finally, Section B.5 provides details on understanding the output and methods to extract required information from the output.

B.1.2 The C-GOLDSTEIN model

The C-GOLDSTEIN software encodes a computationally fast Earth System Model (ESM) developed by R. Marsh and N. R. Edwards (Marsh *et al.*, 2002). It comprises three coupled model components:

- The Global Ocean-Linear Drag Salt and Temperature Equation Integrator (GOLDSTEIN) ocean model.
- An atmospheric Energy Moisture Balance Model (EMBM).
- A simple sea ice model.

Details of the model computational components, variable parameters, input and output data are given in Marsh *et al.* (2002) along with a brief description of installation and execution procedure. This document refers to two versions of C-GOLDSTEIN:

- Version 1 is the basic model.
- Version 2 incorporates observed carbon dioxide values up to the year 2000 and has forecasting capabilities thereafter.

B.2 Build and Installation

This section outlines the procedures required to build a C-GOLDSTEIN executable.

B.2.1 Requirments

In order to build C-GOLDSTEIN the following software must be available:

- Unix operating system (SOLARIS 9, Suse 9 and Red Hat 10 are known to work)
- The Make tool (GNU is known to work)
- A text editor
- FORTRAN 77 compiler (G77, gfortran and Intel Fortran 9.0 are known to work)
- FORTRAN 90 compiler (Intel Fortran 9.0 and gfortran are known to work)

B.2.2 Building

The binary (or executable) is generated automatically within the terminal with the issuing of the following statement.

```
$> make goldstein
```

This instructs the `Make` program to perform the actions listed in the `Makefile` file. Unless your system is identical to the system on which C-GOLDSTEIN was developed (unlikely!), you will need to modify this `Makefile` in order to build C-GOLDSTEIN on your system. `Makefile` is within the `genie-cgoldstein` directory.

The correct FORTRAN compiler for your system may be selected by commenting out the other listed compilers. It is important to note that the `r8` flag, or its equivalent, is required for all compilers. Failure to follow this instruction will result in multiple underflow errors. Correct execution of the `make goldstein` command will result in the `goldstein` binary. You may have to set the executable file permission in order to perform later tasks. This may be achieved with the command

```
$> chmod o+x goldstein
```

Users familiar with Makefile constructs should have no difficulty in producing a valid executable.

B.3 Configuration

This section outlines the procedures required to configure the C-GOLDSTEIN model. Each instance of C-GOLDSTEIN execution requires an input configuration file, along with various data sets. A method for automatic generation of the multiple input configuration files needed for ensemble execution is described. The input configuration file for any run is described in Section B.3.2.

B.3.1 Ancillary data

C-GOLDSTEIN requires various ancillary input files such as `taux_u.interp` and `uncep.silo`. These should be located in the `genie-cgoldstein` directory and do not require modification by the user.

B.3.2 Inputs overview

The C-GOLDSTEIN executable requires a `goin` file, containing values for input parameters and various time-step configurations. The contents of a `goin` file may be assigned to five groups:

1. Time step input parameters - these control the duration of a run and temporal resolution of the model outputs. See Section B.3.3.
2. I/O input parameters - input/output filenames etc. See Section B.3.4.
3. Ocean input parameters - see Section B.3.5 and Table B.1.
4. Atmosphere input parameters - see Section B.3.6 and Table B.2.
5. Sea Ice input parameters - see Section B.3.7 and Table B.3.

B.3.3 Time step input parameters

These parameters control the temporal resolution of the recorded output data and the duration of the run.

`nsteps`
: This specifies the length of the run. There are 100 timesteps per year, as detailed below, therefore the value of nsteps is 100 times the required number of years.

`npstp`
: During a run, diagnostic information of the run is written to an `out` file when nsteps is at multiples of npstp.

`iwstp`
: When nsteps is at multiples of iwstp, files are saved so the run could be restarted from this point if required.

`itstp`
: When nsteps is at multiples of itstp output files are written. For example, if data were required once per year itstp would be 100.

`ianav`
: Annual averages are obtained over ianav timesteps.

`nyearr`
: timesteps per year = 100

`A/O dt ratio`
: Atmosphere to ocean timestep ratio. This has a default value of 5.

`rel`
: Velocity Relaxation (as defined by Marsh *et al.* (2002) in Appendix A) with default value 0.90.

B.3.4 I/O input parameters

These parameters dictate the location of various input and outputs.

`n/c`
: This specifies whether the run is a new or continuing run.

`output file number`
: Output files will be identified with this.

`input file name`
: If `c` was specified (a continued run), `input file name` is the restart file from which this run will continue.

B.3.5 Ocean input parameters

The following table lists the many parameters used to alter the behaviour of the ocean in the model.

Input	Input Name in Model	Default Value
Inital ocean temperature in northern hemisphere (°C)	temp0	10
Inital ocean temperature in southern hemisphere (°C)	temp1	10
Wind stress scale	scl_tau or scf	2.00
Ocean horizontal diffusivity (m^2s^{-1})	ocean diffusivity iso or diff1	2000.
Ocean vertical diffusivity (m^2s^{-1})	ocean diffusivity dia or diff2	1e-4.
Ocean drag coefficient (days)	inverse minimum drag or adrag	2.5
Initial humidity over ocean	relh0_ocean	0.
Atlantic-to-Pacific freshwater flux adjustment values (Sv)	extra1a, extra1b, extra1c, extra1d	-0.03, 0.17, 0.18, 0
Scaling factor for Atlantic to Pacific moisture flux	scl_fwf	1

Table B.1: Ocean Input Parameters for C-GOLDSTEIN

B.3.6 Atmosphere input parameters

The following table lists the parameters used to alter the behaviour of the atmosphere in the model.

Input	Input Name in Model	Default Value
Atmospheric heat diffusivity (m^2s^{-1})	atm. diff. amp. for T or diffamp(1)	5.0e6
Atmospheric moisture diffusivity (m^2s^{-1})	atm. diff. amp. for q or diffamp(2)	1.0e6
Width of atmospheric heat diffusivity profile (rad)	atm. diff. amp. dist'n width or diffwid	1.0
Slope of atmospheric heat diffusivity profile	atm. diff. amp. slope or difflin	0.1
Zonal heat advection factor	atm. advection factors for T_z or betaz1	0.
Meridional heat advection factor	atm. advection factors for T_m	0.
Zonal moisture advection factor	atm. advection factors for q_z or betaz2	0.4
Meridional moisture advection factor	atm. advection factors for q_m	0.4
Scales co2 concentration relative to 350ppm	scl_co2	1.0
Specifies a compound annual % rate of increase	pc_co2_rise	0.0
Climate sensitivity (Wm^{-2})	delf2x	5.77
Solar Constant (Wm^{-2})	solconst	1368
Inital temperature of the atmosphere	tatm	0.
Initial humidity over land	relh0_land	0
Threshold Relative Humidity above which precipitation occurs	rmax	0.85
e-folding timescale of Carbon removal (years)	t_absorb	150.

Table B.2: Atmosphere Input Parameters for C-GOLDSTEIN

B.3.7 Sea Ice input parameters

The following table lists the parameters used to alter the behaviour of sea ice in the model.

Input	Input Name in Model	Default Value
Sea ice diffusivity (m^2s^{-1})	sea-ice eddy diffusivity or diffsic	2000.
Sensitivity of Greenland Ice Sheet melt to warming ($Sv\,^{\circ}C^{-1}$)	k_gis	0.01

Table B.3: Sea Ice Input Parameters for C-GOLDSTEIN

B.4 Multiple run executions

B.4.1 The mkgoin program

Once an experimental design has been chosen the corresponding `goin` input file must be created for each instance of the model. The `mkgoin` program provides a simplified mechanism for generating multiple `goin` files.

The procedure for `mkgoin` operation is as follws:

1. Ensure the existance of the directories `../goin/` and `../goout/`.
2. Provide a `goin.std` file. This is a typical `goin` file which will provide all input parameters not being changed in the experiment.
3. Generate a design file listing the value of each parameter being changed in the experiment. Each run of the model requires a value for all variables being changed. This file is described in Section B.4.2.
4. Modify the `mkgoin.f` source, identifying the parameters to be changed in the experiment. This is further described in Section B.4.3.
5. Build and execute the `mkgoin` program to generate the multiple `goin` files.
6. Execute the C-GOLSTEIN model once for each `goin` file. This is further described in SectionB.4.4.

B.4.2 Design file

The design lists, for each model run, a value for each parameter being varied at any point in the experiment. Any parameter that remains constant throughout the experiment will be taken from the `goin.std` file. Examples of `goin.std` to generate a batch of spinup input files in version 1 or 2 of C-GOLDSTEIN are shown in Sections B.4.5 and B.4.6 respectively. Differences between `goin` files for versions 1 and 2 are explained further in Section B.4.6.

The design file itself is simply a space delimited text file enumerating each combination of input parameters required in the experiment. For example, an experiment consisting of 10 runs, varying only ocean drag coefficient, atmospheric moisture diffusivity and climate sensitivity would require a design file as shown in Figure B.1. This file must be called `Design` and located in the `genie-cgoldstein` directory. The first column consists of the values for the ocean drag coefficient, the second for atmospheric moisture diffusivity and the third for climate sensitivity.

```
3.496280 1573731 3.696902
3.182065 1530661 4.249042
4.083741 1596162 3.362610
4.290861 1637941 5.113775
3.417705 1744212 4.840546
3.758303 1650016 5.653014
3.571851 1709990 7.804285
2.994917 1589768 5.515842
3.354139 1674013 6.146303
3.790954 1618322 4.405619
```

Figure B.1: Example of a design file

Note that there is no header information. Accordingly the read routine in `mkgoin.f` must be modified for each experiment, as shown in Section B.4.3.

B.4.3 Modifying mkgoin

Depending on which input parameters are being varied, the `mkgoin.f` source code may need to be altered and recompiled. This source code is located in the `genie-cgoldstein` directory. In the previous example the parameters ocean drag coefficient, atmospheric moisture diffusivity and climate sensitivity were altered.

This would require the changes shown in Figure B.2 to be made to the line beginning `read(97,)` of `mkgoin.f`.

```
       read(97,*,iostat=iend,end=99)
      &    adrag,diffamp(2),delf2x
```

Figure B.2: Example of adapted part of mkgoin.f

The complete FORTRAN source code for `mkgoin.f` is given in Section B.7.

B.4.4 Executing C-GOLDSTEIN with mkgoin generated inputs

The `mkgoin` program will generate multiple `goin` files. The previous example will result in the files `../goin/goin.0` to `../goin/goin.9`.

The first instance of the model my be executed as follows:

```
goldstein < ../goin/goin.0 > ../goout/out.0
```

This will run C-GOLDSTEIN at the first input configuration, `goin.0`, located in the directory `goin`. At multiples of `npstp` information will be written to `out.0`, located in the directory `goout`. Output will be written to the `results/` directory as detailed in Section B.5.

B.4.5 Spinup without observed CO_2

It is recommended to spinup C-GOLDSTEIN for 4000 years. To run the model to equilibrium without incorporating observed values of carbon dioxide, version 1 of C-GOLDSTEIN must be executed. An example of a `goin` file to run such a spin up with default parameter values is given here:

```
400001 400000 20000 100 100 nsteps npstp iwstp itstp ianav
n                           new/continuing
100   5                     timesteps per year and A/O dt ratio
10 10 0.90 2.00             temp0 temp1 rel scl_tau
2000. 1e-4                  ocean diffusivites iso/dia (or horiz/vert)
2.5                         inverse minimum drag in days
5.0e6   1e6  1.0 0.1        atm. diff. amp. for T & q & dist'n width, slope
0. 0. 0.4 0.4               atm. advection factors for T_z, T_m, q_z, q_m
1.0 0.0                     scl_co2 pc_co2_rise
2000.                       sea-ice eddy diffusivity
0. 0. 0.                    tatm relh0_ocean relh0_land
-0.03 0.17 0.18             extra1a extra1b extra1c
tmp/tmp                     output file number
tmp/tmp.avg                 input file name
```

Input file for Spinup V1

B.4.6 Spinup with observed CO_2 (1800 - 2000)

To spinup C-GOLDSTEIN with observed CO_2 values from 1800 to the year 2000, version 2 of the model must be executed. This version, in addition to extended source code, contains the file `co2_historical.dat` within the `genie-cgoldstein` directory. This file consists of observed values of atmospheric carbon dioxide, as reported by Johnston (2007). Ice core observations are used pre 1958. The layout of a `goin` file for version 2 is slightly different to that for version 1, due to minor variations in the source code of the two models. A `goin` file for version 2 also requires further input parameters in addition to those existing in a `goin` file for version 1. Two additional inputs are `t_absorb` and `k_gis`, see Sections B.3.6 and B.3.7 respectively for details. An example of a `goin` file to spinup to the year 2000 is given here:

```
400001 10000 20000 100 100  nsteps npstp iwstp itstp ianav
n                           new/continuing
100  5                      timesteps per year and A/O dt ratio
10                          temp0
10                          temp1
0.9                         rel
2                           scl_tau
2000.                       horiz ocean diff.
1.e-04                      vertical ocean diff.
2.5                         adrag
5.0e6                       diffamp(1)
1e6                         diffamp(2)
1.0                         diffwid
0.1                         difflin
0.                          betaz(1)
0.                          betam(1)
0.4                         betaz(2)
0.4                         betam(2)
1.0                         scl_co2
0.0                         pc_co2_rise
5.77                        delf2x
1                           iscen1
0.85                        rmax
2000                        diffsic
0.                          tatm
0.                          relh0_ocean
0.                          relh0_land
1                           scl_fwf
0.                          extra1d
1368                        solconst
tmp/spn                     output file number
tmp/tmp.avg                 input file name
150.                        t_absorb
1e-02                       k_gis
```

Input file for Spinup V2

B.4.7 Forecast

C-GOLDSTEIN can forecast past 2000 after an appropriate spinup. Input parameters for these forecasts are based on IPCC emission scenarios (IPCC, 2000). The data for the 6 emissions scenarios detailed by IPCC (2000) are located in the `genie-cgoldstein` directory for version 2.

Having chosen an emissions scenario, (or created a new one), there are 2 methods to implement it as detailed below.

1. In the `genie-cgoldstein` directory copy the chosen emissions scenario to `emissions.dat`.

2. In the `genie-cgoldstein` directory go to `gseta.F`. Where currently `emissions.dat` is opened change the file name to the chosen scenario.

An example of a `goin` file to continue the previously spunup version 2 model run, from the `goin` file in Section B.4.6, up to the year 3000 is shown here:

```
100001 10000 20000 100 100  nsteps npstp iwstp itstp ianav
c                           new/continuing
100  5                      timesteps per year and A/O dt ratio
10                          temp0
10                          temp1
0.9                         rel
2                           scl_tau
2000.                       horiz ocean diff.
1.e-04                      vertical ocean diff.
2.5                         adrag
5.0e6                       diffamp(1)
1e6                         diffamp(2)
1.0                         diffwid
0.1                         difflin
0.                          betaz(1)
0.                          betam(1)
0.4                         betaz(2)
0.4                         betam(2)
1.0                         scl_co2
0.0                         pc_co2_rise
5.77                        delf2x
1                           iscen1
0.85                        rmax
2000                        diffsic
0.                          tatm
0.                          relh0_ocean
0.                          relh0_land
1                           scl_fwf
0.                          extra1d
1368                        solconst
ctd/000                     output file number
tmp/spn.0                   input file name
150.                        t_absorb
1e-02                       k_gis
```

Input file for a continued run

B.5 Output

B.5.1 Location

A directory called `results` must be created along with a subdirectory with a three character name for the ensemble of runs. For an ensemble of spinup runs, for example, the directory may be `results/spn`. The model will then write output data to this directory. Output data is in ascii, identified by the 3 character name given to each run in `mkgoin.f`, see Section B.4.1. The first member of an ensemble typically has the name 000 and for a run of this name, Table B.4 gives details of the output files. Temperature is measured in Celcius and salinity is measured in psu.

Output file	Description
000.n	'Restart' files which can be used as input files in continuing runs
000.t	Mean ocean temperature in various regions
000.s	Mean ocean salinity in various regions
000.airt	Global mean air temperature
000.q	Global mean specific humidity
000.opsi	Meridional overturning streamfunction
000.zpsi	Zonal overturning streamfunction
000.psi	Barotropic streamfunction
000.rho	Density
000.cost	2d array showing the frequency of convection during the run
000.relh	Relative humidity
000.fx0a	Net heat flux into atmosphere
000.pptn	Precipitation rate
000.evap	Evaporation rate
000.runoff	River runoff
000.fwfxneto	Net freshwater flux into ocean
000.fx0neto	Net heat flux into ocean from atmosphere and sea ice
000.fofy	Poleward heat flux in Atlantic, Pacific and total

Table B.4: Output files

B.5.2 Plots

Various fields and time series can be plotted from the data. A number of `matlab` programs for creating these plots are located in the `genie-cgoldstein` directory. In particular, `tstq.m` is a program to visualise temperature and salinity and `tplot` produces various time series data. One of the results from the `matlab` program `tstq.m` is a plot of surface air temperature. This output, at the year 2000 under default parameter settings is shown in Figure B.3.

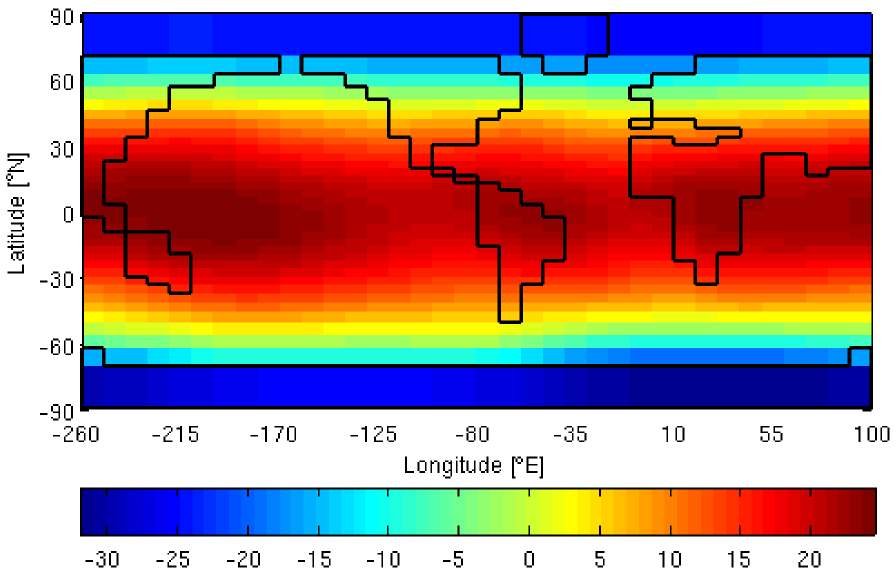
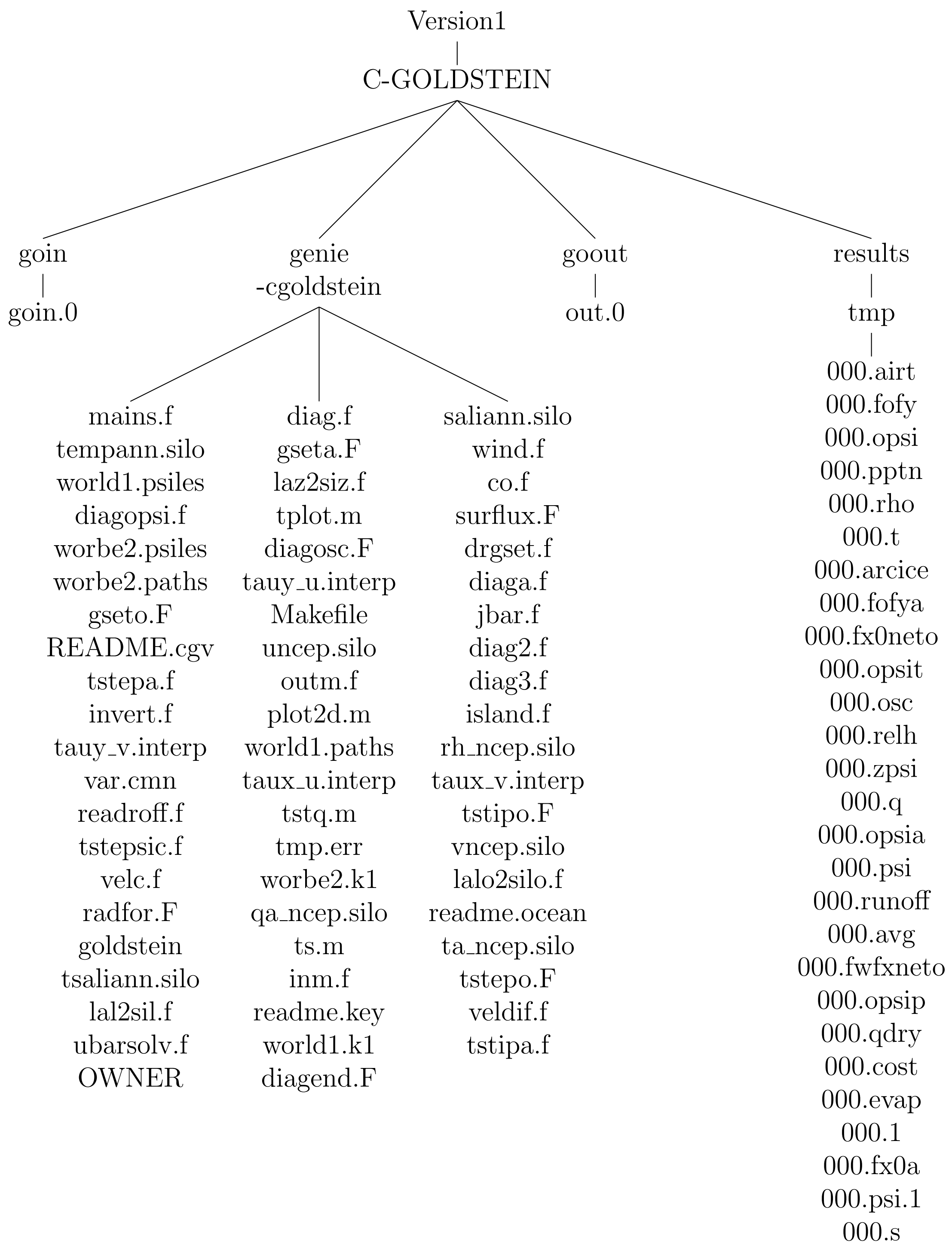


Figure B.3: Surface air temperature output from the C-GODLSTEIN model, run to AD2000 at default parameter settings.

B.6 Directory Structure

In this section we present the directory structure of the C-GOLDSTEIN model.


Version1
C-GOLDSTEIN
goin
goin.0
genie
-cgoldstein
goout
out.0
results
tmp
mains.f
tempann.silo
world1.psiles
diagopsi.f
worbe2.psiles
worbe2.paths
gseto.F
README.cgv
tstepa.f
invert.f
tauy_v.interp
var.cmn
readroff.f
tstepsic.f
velc.f
radfor.F
goldstein
tsaliann.silo
lal2sil.f
ubarsolv.f
OWNER
diag.f
gseta.F
laz2siz.f
tplot.m
diagosc.F
tauy_u.interp
Makefile
uncep.silo
outm.f
plot2d.m
world1.paths
taux_u.interp
tstq.m
tmp.err
worbe2.k1
qa_ncep.silo
ts.m
inm.f
readme.key
world1.k1
diagend.F
saliann.silo
wind.f
co.f
surflux.F
drgset.f
diaga.f
jbar.f
diag2.f
diag3.f
island.f
rh_ncep.silo
taux_v.interp
tstipo.F
vncep.silo
lalo2silo.f
readme.ocean
ta_ncep.silo
tstepo.F
veldif.f
tstipa.f
000.airt
000.fofy
000.opsi
000.pptn
000.rho
000.t
000.arcice
000.fofya
000.fx0neto
000.opsit
000.osc
000.relh
000.zpsi
000.q
000.opsia
000.psi
000.runoff
000.avg
000.fwfxneto
000.opsip
000.qdry
000.cost
000.evap
000.1
000.fx0a
000.psi.1
000.s

B.7 Source code

The complete FORTRAN source code for `mkgoin.f` is given here.

```
     program mkgoin

      real*8 tv,temp0,temp1,rel,scf,diff(2),adrag,diffamp(2),width
     .,slope,betaz(2),betam(2),scl_co2,pc_co2_rise,delf2x,diffsic
     .,tatm,relh0_ocean,relh0_land,etxra1a,extra1b,extra1c,scl_fwf
      integer nsteps,npstp,iwstp,itstp,ndta,ianav
      integer icount
      character ans,lout*3,lin*6
      character filegoin*20, filegoout*20, outfile*7, infile*7
     &,fileerr*20, filelog*20
      character*10 fmt
      character*10 fmt1
      character*10 fmt10
      character*10 fmt100

c different formats for runs 0-9, 10-99, 100+

      fmt1  ='(a13,i1.1)'
      fmt10 ='(a13,i2.2)'
      fmt100='(a13,i3.3)'

      open(1,file='goin.std',status='old')

      read(1,*)nsteps,npstp,iwstp,itstp,ianav
      read(1,'(a1)')ans
      read(1,*)tv,ndta
      read(1,*)temp0,temp1,rel,scf
      read(1,*)diff(1),diff(2)
      read(1,*)adrag
      read(1,*)diffamp(1),diffamp(2),width,slope
      read(1,*)betaz(1),betam(1),betaz(2),betam(2)
      read(1,*)scl_co2,pc_co2_rise,delf2x
      read(1,*)sea_ice
      read(1,*)tatm,relh0_ocean,relh0_land
      read(1,*)extra1a,extra1b,extra1c,scl_fwf
      read(1,'(a7)')lout
      read(1,'(a11)')lin

c  open design file

      open(97,file='Design',status='old')

c  input file
      infile='tmp/tmp.avg'

      icount = -1

      iend=0

      do while (iend.eq.0)

        read(97,*,iostat=iend,end=99)
     &      adrag,diffamp(2),scl_fwf,delf2x,diff(2)

        icount=icount+1

        if(icount.le.9) write(outfile,'(a6,i1)') 'tmp/00',icount
        if(icount.ge.10.and.icount.le.99)
     & write(outfile,'(a5,i2)') 'tmp/0',icount
        if(icount.ge.100) write(outfile,'(a4,i3)') 'tmp/',icount

c  setup correct format

        if(icount.le.9) then
          fmt=fmt1
         else if (icount.le.99) then
          fmt=fmt10
         else
          fmt=fmt100
        endif

        write(filegoin,fmt) '../goin/goin.',icount
        write(filegoout,fmt) '/goout/out.',icount

        open(2,file=filegoin,status='new')
        write(2,*)nsteps,npstp,iwstp,itstp,ianav
        write(2,'(a1)')ans
        write(2,*)tv,ndta
        write(2,*)temp0,temp1,rel,scf
        write(2,*)diff(1),diff(2)
        write(2,*)adrag
        write(2,*)diffamp(1),diffamp(2),width,slope
        write(2,*)betaz(1),betam(1),betaz(2),betam(2)
        write(2,*)scl_co2,pc_co2_rise,delf2x
        write(2,*)sea_ ice
        write(2,*)tatm,relh0_ocean,relh0_ land
        write(2,*)extra1a,extra1b,extra1c,scl_ fwf
        write(2,'(a7)')outfile
        write(2,'(a7)')infile

        open(3,file=filegoout,status='new')
        close(2)
        close(3)

99      continue
      enddo

      stop
      end
```