

Dissertation

“Using GEM-SA to explore wind speed  
dependencies in HF radar backscatter power spectra”

Edmund Ryan

Department of Statistics      University of Sheffield

Submitted in partial fulfillment of the  
requirements for the MSc in Statistics

September 2008.

## Abstract

**Abstract of:** Using GEM-SA to explore wind speed dependencies in HF radar backscatter power spectra.

**Author:** Edmund Ryan

**Date:** September, 2008.

A simulator is used to model and predict the behaviour of a complex physical system. Sensitivity analysis (SA) is sometimes carried out as a way of understanding the physical system. SA is the study of how the output of a simulator is sensitive to changes in the values of an input. Usually SA is carried out using Monte-Carlo methods, but this often requires 1000s of runs (ie 1000s of simulator inputs and corresponding outputs). If one run of the simulator is computationally expensive, we substitute the simulator for an *emulator*. An emulator is a statistical representation of in the input-output relationship of the simulator. To do SA using an emulator requires usually less than 100 runs of the simulator's computer code, which is a lot less than Monte-Carlo.

In this dissertation, we build a Gaussian process emulator using GEM-SA (an emulator building program) from a simulator with a 5-dimensional input and 512-dimensional output. After conducting various diagnostic checks to ensure that the emulator is accurately representing the simulator, we then carry out sensitivity analysis to answer the question at the centre of our investigation: How is HF radar backscatter power spectra (the simulator's output) sensitive to changes in wind speed (one of the simulator's inputs)?



## Acknowledgments

I would like to thank:

My supervisor, Dr. Jeremy Oakley, for all his guidance and support.

Professor Lucy Wyatt, from the Department of Applied Maths, for her help with all the problems I had with the simulator.

Dr Jonathan Jordan, the course director, and all the other staff from the Department of Probability and Statistics.

The other MSc students who have given me enormous help and encouragement along the way.



# Contents

<b>1</b>	<b>Introduction &amp; Aims</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.1.1	Motivation behind this dissertation . . . . .	1
1.1.2	Type of Simulator used for this dissertation . . . . .	4
1.1.3	Sensitivity Analysis . . . . .	6
1.2	Aims . . . . .	6
1.3	Overview of the dissertation . . . . .	7
<b>2</b>	<b>The Emulator</b>	<b>9</b>
2.1	Introduction . . . . .	9
2.2	What is an Emulator? . . . . .	9
2.3	Building an Emulator . . . . .	10
2.4	Gaussian Process Emulators . . . . .	11
2.5	Specifying the Prior to Posterior distributions in GEM-SA . . . . .	12
2.5.1	The Prior Distribution . . . . .	12
2.5.2	The Posterior Distribution . . . . .	14
2.5.3	How to compute $\mathbf{b}$ , the vector of roughness parameters . . . . .	15
2.5.4	Normalisation and Standardisation . . . . .	15

2.5.5	Common assumptions made in building a Gaussian Process emulator . . . . .	16
2.6	Four Key Questions . . . . .	16
2.7	Conclusion . . . . .	17
<b>3</b>	<b>Design</b>	<b>19</b>
3.1	Introduction . . . . .	19
3.2	Latin Hypercube Designs . . . . .	19
3.2.1	What is a Latin Hypercube Design? . . . . .	19
3.2.2	An example . . . . .	20
3.3	Maximin Latin Hypercube Designs . . . . .	23
3.3.1	What is a <i>Maximin</i> Latin Hypercube Design? . . . . .	23
3.3.2	Morris & Mitchell (1995) . . . . .	24
3.3.3	Literature Review . . . . .	27
3.4	Conclusion . . . . .	27
<b>4</b>	<b>Implementation of the Model</b>	<b>29</b>
4.1	Introduction . . . . .	29
4.2	Multiple Output . . . . .	29
4.2.1	What was decided for this dissertation . . . . .	29
4.2.2	What the Literature Says . . . . .	32
4.3	How to obtain the training data . . . . .	33
4.3.1	The training inputs . . . . .	33
4.3.2	The training outputs . . . . .	35
4.4	How much training data is required? . . . . .	40

4.4.1	Introduction . . . . .	40
4.4.2	What is Cross-Validation? . . . . .	40
4.4.3	How is the Cross-validation error computed for this dissertation? . . . . .	41
4.4.4	Why the Cross-Validation Error not computed for $n > 50$ .	44
4.5	Results . . . . .	44
4.5.1	Line plots of Cross-validation error verses $n$ . . . . .	44
4.5.2	Interpretation of the line plots . . . . .	45
4.6	Conclusion . . . . .	48
<b>5</b>	<b>Diagnostic Checks</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Validation Data . . . . .	52
5.3	Diagnostic Check 1 . . . . .	53
5.3.1	Plot of the emulator's output against the simulator's output	53
5.3.2	Plot of the standardised prediction errors . . . . .	53
5.4	Diagnostic Check 2 . . . . .	54
5.5	Diagnostic Check 3 . . . . .	56
5.5.1	(i) Pivoted Cholesky Decomposition . . . . .	58
5.5.2	(ii) Plot of the Pivoted Cholesky errors against the Pivoting Order . . . . .	60
5.5.3	(iii) Quantile-quantile plot . . . . .	61
5.6	Results . . . . .	61
5.6.1	Roughness parameters . . . . .	62
5.6.2	Diagnostic check 1 . . . . .	63



5.6.3	Diagnostic check 2 . . . . .	65
5.6.4	Diagnostic check 3 . . . . .	66
5.7	Conclusion . . . . .	72
<b>6</b>	<b>Sensitivity Analysis</b>	<b>77</b>
6.1	Introduction . . . . .	77
6.2	Theory and Background . . . . .	78
6.2.1	What is sensitivity analysis? . . . . .	78
6.2.2	The Sensitivity Index . . . . .	79
6.2.3	Total Effects . . . . .	80
6.2.4	How sensitivity analysis is summarised in GEM-SA . . . .	80
6.2.5	Code Uncertainty . . . . .	81
6.3	Results . . . . .	82
6.3.1	Exploratory Scatter Plots . . . . .	82
6.3.2	Main Effect Plots . . . . .	82
6.3.3	Table of variances and Total Effect . . . . .	83
6.4	Conclusion . . . . .	87
<b>7</b>	<b>Overall Conclusion and Discussion</b>	<b>89</b>
7.1	Overall Conclusion . . . . .	89
7.2	Limitations, further discussion . . . . .	90
<b>8</b>	<b>Appendix</b>	<b>93</b>
8.1	How to obtain the training outputs . . . . .	93
8.1.1	Instructions for how to use the simulator . . . . .	93
8.1.2	How to obtain the ten outputs . . . . .	96

8.1.3	Excel macro . . . . .	97
8.2	R Code for Diagnostic Check 1 . . . . .	98
8.3	R Code for Diagnostic Check 2 . . . . .	99
8.3.1	The mean function, $E[f(\mathbf{X}_i^* \mathbf{y})]$ . . . . .	99
8.3.2	The covariance matrix $V[f(\mathbf{X}_i^* \mathbf{y})]$ . . . . .	101
8.3.3	Deriving the values of $D_{MD}(\mathbf{y}^*)$ . . . . .	104
8.3.4	Deriving the distributional values for $D_{MD}(\cdot)$ (table 4) . .	104
8.4	R Code for Diagnostic Check 3 . . . . .	105
8.4.1	Figures 5.4 and 5.5 . . . . .	105
8.4.2	Figures 5.6 and 5.7 . . . . .	106
8.4.3	Plot of Pivoted Cholesky errors and Quantile-quantile plot for validation data without outliers . . . . .	107
8.5	Poster Presentation . . . . .	108
<b>9</b>	<b>Bibliography</b>	<b>109</b>

# Chapter 1

## Introduction & Aims

### 1.1 Introduction

#### 1.1.1 Motivation behind this dissertation

##### Simulators

In many fields of science, simulators are used to predict and understand the behaviour of a complex physical system. For example, the MET office weather forecasts are based on the outputs of a simulator. A simulator is a mathematical function, which we denote by  $f : \mathbb{R}^p \rightarrow \mathbb{R}^{p'}$ , where  $p$  and  $p'$  are the dimensions of the input and output spaces respectively. For the purposes of this dissertation we assume that the simulator is a deterministic model (as opposed a stochastic model) obtained by solving a system of differential equations. We define a “simulator run” as the process by which a simulator computes an output from a given input. One feature of the simulator is that if a simulator run is repeated for the same input  $\mathbf{x}$ , the resulting output  $f(\mathbf{x})$  will be the same. This is in contrast to the complex physical system the simulator is trying to mimic. This is because

in real-life, randomness means that if a physical experiment is repeated with the same input value, one is very unlikely to obtain the same result.

For example, suppose I want to investigate how far a paper airplane flies given various varying input variables, e.g. wing span, area of paper used, the weight of the paper used, etc.... If I make two identical paper planes (identical in every way, e.g. same length, wingspan, weight of paper, etc...) and then throw them, it is very unlikely that the planes will travel exactly the same distance. This is because however hard the paper plane maker tries to make the two planes identical, he will never be able to do it exactly. In the same way, the person who throws the planes will never be able to repeat a throw in precisely the same way. External factors, such as air temperature, minute wind currents, etc... are also very likely to be different during both throws. These tiny differences are big enough to cause the results to be different.

### **Simulators verses Models based on a physical experiment**

Models based on a physical experiment are also used in many areas of science. However, there are many reasons why simulators (which in general do not rely on the collection of data) are often preferred:

- The collecting of the data may become impossible or impracticable.
- There may be too many input variables.
- There may be ethical issues for why a physical experiment cannot be carried out.

- The physical experiment may be too time consuming or too costly.

## Simulators and Computer Experiments

Once a simulator has been built and represents the physical system to a sufficiently good degree of accuracy, a computer experiment is usually then carried out. A computer experiment is when we carry out runs of the simulator on some varying input setting. The purpose is to answer a question about the physical system itself. For instance, is there uncertainty in the inputs? How is the output sensitive to changes in the inputs? The areas that surround these two questions are called *Uncertainty Analysis* and *Sensitivity Analysis*, respectively. Normally Monte-Carlo methods are used to carry out these kinds of analyses. However, such methods usually require in the order of 1000s of runs of the simulator. This can be a problem if the simulator is computationally expensive. Being computationally expensive means for instance that one run of the simulator takes an excessively long time<sup>1</sup>, or one run could be financially too expensive.

For a simulator which is computationally expensive, we substitute it for an emulator. An emulator is a statistical representation of the input-output relationship of the simulator. To build an emulator usually requires less than 100 runs of the simulator (which is much less than Monte-Carlo). Sensitivity analysis, Uncertainty analysis, etc... can then be carried out directly on the emulator.

---

<sup>1</sup>For example, Sacks et al. (1989) describe a fluid dynamics computer experiment where one run of the simulator takes 20 minutes.

### 1.1.2 Type of Simulator used for this dissertation

The simulator is used in weather modelling, to give better estimates to offshore wind speeds. A more technical explanation (which is not necessary for the reader to fully understand) is given in the project brief for this dissertation. This states:

*“HF radars are located on the coast, send radio waves out to sea which are scattered by moving ocean waves and some of this scattered energy arrives at the radio receiver. After some radar signal processing we get a power spectrum of the scattered signal which is the ‘raw’ data for the oceanographic parameter extraction. The power spectrum tells us the amplitude of scatter from targets which induce different frequency shifts to the scattered signal due to their different speeds (Doppler Effect). We can describe the backscatter from ocean waves with a non-linear integral equation of the form:*

$$\sigma(\omega) = \alpha S(k_0, \theta_0) + \iint K(k, \theta, \omega) S(k, \theta) S(k', \theta') dk d\theta$$

*where  $\sigma(\omega)$  is the power spectrum at frequency,  $\omega$  (rad/sec),  $S(k, \theta)$  is the ocean wave directional spectrum at wavenumber  $k$  in direction  $\theta$ ,  $K(k, \theta, \omega)$  is a function containing all the physical processes associated with scattering to second order. The other  $k, \theta$ s are different wavenumbers and directions involved in the process.”*

The above paragraph is more of a technical description of what goes. For the

purposes of the reader, the simulator can be thought of as a ‘black box’, which computes an output when an input is entered (figure 1.1). Each input has 5 variables (or dimensions), which are (with each sub-domain): Radar frequency (6 to 30 Hz), Water depth (5 to 200 metres), Wind speed (3 to 30 m/s), Wind direction (0 to 360°), S/N ratio (-70 to -10dB). The first four inputs require no explanation. The S/N ratio is difficult to explain exactly; all that is necessary to know is that it adds a noise floor.

Figure 1.1: The simulator is essentially a ‘black box’ with a 5 dimensional input and a 512 dimensional output.

Finally, each output has 512 dimensions (or values) and is represented in the form of a power spectrum. To exemplify, the numbers at the top of the next page are part of the output produced when the the following inputs are entered into the simulator: radar frequency = 6.22Hz, water depth = 160.91m, wind speed = 10.80m/s, wind direction = 155.09°, S/N ratio = -41.60dB. The numbers in the left column are the index. The actual output (Power) are in the right column, which will change when different inputs are specified. The middle column (Doppler frequencies) are always the same whatever inputs are entered in. In essence, we say that the output consists of power values corresponding 512 fixed doppler frequencies. The output is referred to as ‘HF (High Frequency) radar backscatter power spectra’. However for the purpose of simplicity, from this point onwards, it will be referred to as ‘HF radar backscatter’.

	Doppler Frequency	Power
1	-1.89258	1.98E-02
2	-1.88516	1.78E-02
3	-1.87773	2.01E-02
4	-1.87031	1.90E-02
.	.	.
.	.	.
.	.	.
511	1.89258	2.33E-02
512	1.9	2.43E-02

### 1.1.3 Sensitivity Analysis

The title of the dissertation is: “Using GEM-SA to explore the wind-speed dependencies in HF radar backscatter power spectra”. GEM-SA (the emulator building program) stands for Gaussian Emulation Machine for Sensitivity Analysis. Therefore, the main question in this dissertation is: how is the simulator output (HF radar backscatter) sensitive to changes to the input variable wind speed? (this is referred to as sensitivity analysis). To answer this, an emulator will first be built in GEM-SA (note that the type of emulator is a ‘Gaussian emulator’). The actual sensitivity analysis will then be carried out, also in GEM-SA.

## 1.2 Aims

Following on from the previous subsection, there are three main aims to this dissertation. They are:



- (1) To build an emulator using the program GEM-SA.
- (2) To carry out diagnostic checks on the emulator to check that it is performing well, and to quantify this accuracy.
- (3) To use the emulator to answer the question of whether (and if so, to what extent) the simulator output (HF radar backscatter) is sensitive to changes in the simulator input ‘wind-speed’.

## 1.3 Overview of the dissertation

The second chapter gives an account of the theory behind the Gaussian emulator. This is followed by a chapter on the design. Next, the fourth chapter describes the practicalities of how the emulator will be built. Once the emulator is in place, the chapter that follows offers detail about how the model will be checked to ensure that it is performing well (diagnostic checks). The sixth chapter then describes in detail what sensitivity analysis is and how it will be carried out on the emulator, in order to answer the question at the heart of this dissertation: how is HF radar backscatter sensitive to changes in wind-speed? Note that chapters 4, 5 and 6 each include the relevant results as well as the background and theory.

Referring to section 1.2, chapters 2, 3 and 4 are linked to the first aim, chapter 5 to the second aim, and chapter 6 to the third aim. The final chapter, chapter 7, summarises the complete findings and presents limitations and further discussion to the dissertation.



# Chapter 2

## The Emulator

### 2.1 Introduction

The main purpose of this chapter is to give an overview of the theory behind the Gaussian emulator. This is done in sections 2.3, 2.4 and 2.5. The final section, section 2.6, poses some important questions, for example how many simulator runs are required to build an emulator? These will be answered in subsequent chapters.

### 2.2 What is an Emulator?

Recall that an emulator is a statistical model, which is a substitute for the simulator when the simulator is considered computationally expensive. Before proceeding, we assume that the simulator is specified by the mathematical function  $f(\cdot) : \chi \subset \mathbb{R}^p \rightarrow \mathbb{R}$ . The  $\chi$  represents the input space given by  $\mathbf{x} = (x_1, \dots, x_p) \in \chi_1 \times \dots \times \chi_p \subset \chi$ . The simulator output is assumed to be scalar,  $y \in \mathbb{R}$ . Then, when an input  $\mathbf{x}$  is entered into the simulator, it returns an output of the form  $y = f(\mathbf{x})$ .

Unlike the simulator, when an input is entered into the emulator a probability distribution, of what we think the true output should be, is returned. We denote the mean of the distribution by  $\hat{f}(\mathbf{x})$ . Since the entire distribution is specified, we can also describe the uncertainty of this mean value to the true value  $f(\mathbf{x})$ . O'hagan (2004) states: “*The emulator is a probability distribution of the entire function  $f(\cdot)$* ”.

## 2.3 Building an Emulator

At the end of section 1.1.1, it stated that in order to build an emulator, we usually require less than 100 runs of the simulator. We refer to the inputs and outputs which correspond to these runs as the *training data*. In general, we say that training data consist of the set of inputs  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ , and corresponding set of outputs  $y_1 = f(\mathbf{x}_1), y_2 = f(\mathbf{x}_2), \dots, y_n = f(\mathbf{x}_n)$ , the latter of which are obtained by  $n$  runs of the simulator. O'Hagan (2004) states two criteria which an emulator should satisfy:

1. *At a design point  $\mathbf{x}_i$ , the emulator should reflect the fact that we know true value of the simulator output, so  $P(f(\mathbf{x}_i) = y_i) = 1$ .*
2. *At other points, denoted by the general point  $\mathbf{x}$ , the distribution for  $f(\mathbf{x})$  should give a mean value  $\hat{f}(\mathbf{x})$  that represents a plausible interpolation or extrapolation of the training data, and the probability distribution around this mean should be a realistic expression of the uncertainty about how the simulator might interpolate/extrapolate.*

## 2.4 Gaussian Process Emulators

Currin et al. (1988) first introduced the emulator from the Bayesian point of view, and since then it has been widely used. Bayesian methods to emulation are regarded by many to be more efficient than other types. The reason is that before training data are entered into the emulator, we can specify prior beliefs that we may have about the distribution of  $f(\cdot)$ . For example, we can specify the relationship between each input variable and the output to be linear. It is rare to have precise prior information. Most of time, the prior consists of specifying what we think the shape of the true distribution of  $f(\cdot)$  should be.

Most emulators created from a Bayesian point of view use the Gaussian process to represent our prior beliefs since it is a mathematically convenient type of distribution to use. As we will see in the next section, one reason for this is because it is a conjugate prior; that is the resulting posterior distribution is also a Gaussian process. This is important, since if the emulator is used to make predictions from a specified set of inputs, the resulting outputs will take the form of a vector of mean values (which represents our best guess of the true output values) and a covariance matrix (which describes our uncertainty as well as correlations between outputs).

More formally, the Gaussian Process is defined as an extension of the multivariate Gaussian distribution to infinitely many variables (Rasmussen, 2006). The multivariate Gaussian distribution is specified by a mean vector  $\mu$  and covariance matrix  $\Sigma$ . In a similar way, when specifying the Gaussian process, a mean

function  $m(\mathbf{x}) = E(f(\mathbf{x}))$  and covariance function  $c(\mathbf{x}, \mathbf{x}') = cov(f(\mathbf{x}), f(\mathbf{x}'))$  are required. In addition, a Gaussian process has the property that  $f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)$  are jointly normal distributed for any set of  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ .

Therefore, our prior knowledge consists of three pieces of information, conditional on knowing various parameters in the GP model (described in detail in the next subsection):

- $m(\mathbf{x})$ , which is the prior expectation of  $f(\mathbf{x})$ ;
- $\sigma^2 c(\mathbf{x}, \mathbf{x}) = var(\mathbf{x})$ , which describes the uncertainty about  $f(\mathbf{x})$ ;
- $\sigma^2 c(\mathbf{x}, \mathbf{x}')$ , which describes the correlation between points  $\mathbf{x}$  and  $\mathbf{x}'$ .

## 2.5 Specifying the Prior to Posterior distributions in GEM-SA

As mentioned in chapter 1, GEM-SA is the program which we will use to build the Gaussian Process emulator. The theoretical basis behind how an emulator is constructed is now given.

### 2.5.1 The Prior Distribution

The prior distribution, represented by the Gaussian process model, takes the form:

$$f(\cdot) | \beta, \sigma^2, b \sim GP(m_0(\cdot), v_0(\cdot, \cdot)) \quad (2.1)$$

where  $f(\cdot)$  represents the simulator, as specified in the previous section. The functions  $m_0(\cdot)$  and  $v_0(\cdot, \cdot)$  represent the mean and covariance functions given by:

$$m_0(\mathbf{x}) = h(\mathbf{x})^T \beta \quad (2.2)$$

$$v_0(\mathbf{x}, \mathbf{x}') = \sigma^2 c(\mathbf{x}, \mathbf{x}'; b) \quad (2.3)$$

For equation (2.2),  $\mathbf{x}$  is a  $p \times 1$  vector of input variables (the dimension of the input space is  $p$ ). The function  $h(\cdot) : \chi \subset \mathbb{R}^p \rightarrow \mathbb{R}^q$  is a known function of the inputs, represented by a  $q \times 1$  vector. We choose  $h(\cdot)$  according to what we think its form should take. A common choice is  $h(\mathbf{x})^T = (1, \mathbf{x}^T)$  and this is what is done in GEM-SA. In other words, the relationship between each input variable and the output is expected to be linear (ie  $q=p+1$ ). The vector  $\beta$  is an unknown vector of coefficients.

For equation (2.3),  $c(\mathbf{x}, \mathbf{x}'; \mathbf{b}) = \sigma^2 \exp\{-(\mathbf{x} - \mathbf{x}')^T \mathbf{B} (\mathbf{x} - \mathbf{x}')\}$ . Here,  $\mathbf{B}$  is known as the roughness matrix, a  $p \times p$  matrix, with zeros in the off-diagonal elements and diagonal elements represented by the roughness parameters  $b_1, b_2, \dots, b_p$ . (Note that the roughness vector  $\mathbf{b}$  is given by  $\mathbf{b}^T = [b_1, b_2, \dots, b_p]$ ). These give an indication of whether the input-output relationship for each input variable, given the training data, should be linear as specified by  $h(\mathbf{x})$ . The linearity is sometimes called the smoothness of the simulator. Low values indicate that an simulator is smooth for a particular input variable (ie linearity is appropriate), whereas values approaching the maximum of 99 suggest the opposite.

In GEM-SA, weak prior distributions for  $\beta$  and  $\sigma^2$  are assumed. This is represented by  $p(\beta, \sigma^2) \propto \sigma^{-2}$  and  $b_i \sim \text{Exp}(0.01)$  for independent  $b_1, b_2, \dots, b_p$ .

Note that in other academic journals, different notation is sometimes used. For example, the roughness parameters can be written as  $\frac{1}{\psi_1}, \frac{1}{\psi_2}, \dots, \frac{1}{\psi_p}$ , where  $\psi_1, \dots, \psi_p$  are known as the correlation parameters.  $c(\mathbf{x}, \mathbf{x}'; \mathbf{b})$  can also be expressed in non-matrix form as  $c(\mathbf{x}, \mathbf{x}'; \mathbf{b}) = \exp\{-\sum_{k=1}^p b_k(x_k - x'_k)^2\}$ , where  $\mathbf{x} = (x_1, \dots, x_p)$  and  $\mathbf{x}' = (x'_1, \dots, x'_p)$ .

## 2.5.2 The Posterior Distribution

We begin by specifying the distribution of the outputs of the training data. Let the elements of  $\mathbf{y} = (y_1, \dots, y_n)$  be obtained from  $n$  runs of the simulator  $f(\cdot)$ , where  $y_i = f(\mathbf{x}_i)$ , for  $i = 1, \dots, n$ , and the  $\mathbf{x}_i$ s have been chosen according to some suitable design (the design will be discussed in the next chapter). Using equation (2.1), the joint distribution of  $\mathbf{y}$  conditional on  $\beta, \sigma^2$  and  $\mathbf{b}$  is given by the multivariate Normal distribution:

$$\mathbf{y}|\beta, \sigma^2, \mathbf{b} \sim N_n(H\beta, \sigma^2 \mathbf{A}), \quad (2.4)$$

where  $H^T = [h(\mathbf{x}_1), h(\mathbf{x}_2), \dots, h(\mathbf{x}_n)]$ , and  $\mathbf{A} = [\mathbf{A}_{i,j}]$  is an  $n \times n$  matrix with  $\mathbf{A}_{i,j} = c(\mathbf{x}_i, \mathbf{x}_j; \mathbf{b})$ .

Using standard techniques for conditioning in multivariate normal distributions, and using Bayes theorem for the posterior distribution for  $(\beta, \sigma^2)$ , it can be shown that after integrating out these hyperparameters the posterior distribution for  $f(\cdot)$  unconditional on  $\beta$  and  $\sigma^2$  is given by (see Bastos & O'Hagan, 2008)

$$f(\cdot)|\mathbf{y}, \mathbf{b} \sim t_{n-q}(m_1(\cdot), v_1(\cdot, \cdot)), \quad (2.5)$$

where  $m_1(\cdot)$  and  $v_1(\cdot, \cdot)$  represent the respective posterior mean and correlation



functions, given by:

$$\begin{aligned}
m_1(\mathbf{x}) &= h(\mathbf{x})^T \hat{\beta} + t(\mathbf{x})^T \mathbf{A}^{-1}(\mathbf{y} - \mathbf{H}\hat{\beta}), \\
v_1(\mathbf{x}, \mathbf{x}') &= \hat{\sigma}^2 [c(\mathbf{x}, \mathbf{x}'; \mathbf{b}) - t(\mathbf{x})^T \mathbf{A}^{-1} t(\mathbf{x}') \\
&\quad + (h(\mathbf{x}) - t(\mathbf{x})^T \mathbf{A}^{-1} \mathbf{H})(\mathbf{H}^T \mathbf{A}^{-1} \mathbf{H})^{-1} (h(\mathbf{x}') - t(\mathbf{x}')^T \mathbf{A}^{-1} \mathbf{H})^T]
\end{aligned}$$

with  $\hat{\beta}$  and  $\hat{\sigma}^2$  given by:

$$\begin{aligned}
\hat{\beta} &= (\mathbf{H}^T \mathbf{A}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{A}^{-1} \mathbf{y} \\
\hat{\sigma}^2 &= \frac{(\mathbf{y}^T (\mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{H} (\mathbf{H}^T \mathbf{A}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{A}^{-1}))}{n - q - 2}
\end{aligned}$$

### 2.5.3 How to compute $\mathbf{b}$ , the vector of roughness parameters

One modelling issue is the fact that the posterior distribution of  $f(\cdot)$  is conditional on the roughness parameters  $b_1, b_2, \dots, b_p$ . Therefore, these roughness parameters need to be estimated. There are various ways of doing this. A common way is to compute them solely from the data by the method of maximum likelihood. This is what GEM-SA does.

### 2.5.4 Normalisation and Standardisation

Another important consideration to note is the re-scaling of the training data that is used in GEM-SA. GEM-SA normalises all inputs (whether a training input, or an input used to make a prediction) and standardises the training outputs. Let  $\mathbf{x}_i$  be the vector of inputs from the  $i$ th input variable. Then denoting the minimum and maximum values by  $x_i^{min}$  and  $x_i^{max}$ , the normalised vector of inputs,  $\mathbf{x}_i^N$  is

written as:

$$\mathbf{x}_i^N = \frac{\mathbf{x}_i - x_i^{\min} \mathbf{1}_n}{x_i^{\max} - x_i^{\min}}$$

The training outputs (given by  $\mathbf{y} = (y_1, y_2, \dots, y_n)$ ) are standardised, by subtracting the sample mean and dividing by the sample standard deviation:

$$\mathbf{y}_i^N = \frac{\mathbf{y} - m_y \mathbf{1}_n}{\sqrt{v_y}}$$

where  $m_y$  and  $v_y$  are given by:

$$m_y = \frac{1}{n} \sum_{i=1}^n y_i \quad \text{and} \quad v_y = \frac{1}{n-1} \sum_{i=1}^n (y_i - m_y)^2$$

### 2.5.5 Common assumptions made in building a Gaussian Process emulator

It is important to note that these assumptions are about the true distribution of the simulator  $f(\cdot)$ . First of all, the covariance function  $v_0(\mathbf{x}, \mathbf{x}')$  given for the prior (equation 2.3), stationarity is assumed. Secondly, the structure of the mean and covariance in the Gaussian process is assumed to take a particular form. In particular, in treating  $h(\mathbf{x})^T = (1, \mathbf{x}^T)$ , a linear relationship is assumed to be true between each input variable and simulator output. Finally, the simulator outputs are assumed to be jointly normal. Diagnostics checks provide a useful way of checking as far as possible if these assumptions are being met.

## 2.6 Four Key Questions

Before the emulator can be built, there are a number of questions which need to be answered.

### Question 1

Before using the simulator  $f(\cdot)$  to obtain the outputs for the training data, what design should be used to choose the corresponding inputs? Chapter 3 gives reasons why a Maximin Latin Hypercube design is commonly agreed to be the best design to use.

#### Question 2

How much training data should be used in order to build the emulator? This will be answered in chapter 4.

#### Question 3

GEM-SA, the program used to build the emulator, only works when the output is scalar (though the input can be multi-dimensional). However each output using the simulator consists of 512 values. Therefore, the question arises, how do we deal with multi-dimensional output in the training data? This will also be answered in chapter 4.

#### Question 4

Finally, how do we know that the emulator is an accurate representation of the simulator? This will be answered by carrying out various diagnostic checks, which will be discussed in chapter 5.

## **2.7 Conclusion**

An emulator is a statistical representation of the input-output relationship of a simulator. It is used when the simulator is computationally expensive (e.g. one run takes too long, is too expensive, etc...). In order to build the emulator, we need to carry out  $n$  runs of the simulator. The corresponding inputs and outputs

are known as the training data. The emulator is a probability distribution of the simulator  $f(\cdot)$ , where not only are we able to give our best guess for what the true output should be for a given input, but we are also able to describe the uncertainty of that guess.

Gaussian Process Emulators are the most common type of emulators built from a Bayesian point of view. The Gaussian Process is essentially a multivariate Normal distribution, specified by a mean function  $m(\mathbf{x}) = E(f(\mathbf{x}))$  and a covariance function  $c(\mathbf{x}, \mathbf{x}') = cov(f(\mathbf{x}), f(\mathbf{x}'))$ . The prior distribution, as used in GEM-SA, is based on this form. One reason for choosing the prior to take this form is that it is a conjugate prior, that is the posterior is also a Gaussian process.

Finally, there are four questions which need to be answered before the emulator can be built: (1) What design should be used for the training inputs? (2) How much training data is needed to build a good enough emulator? (3) The simulator produces multiple output, but the emulator only accepts training output which is scalar - how will this problem be rectified? (4) How does one know whether an emulator accurately representing the emulator? These questions will be answered in the three chapters that follow.

# Chapter 3

## Design

### 3.1 Introduction

Before an emulator is built using training data, consideration needs to be given to how to choose the training inputs. It turns out that the best way to choose them is according to a Maximin Latin Hypercube Design. The layout of the chapter now follows. Section 3.2 explains what a Latin Hypercube Design is, giving an example to aid understanding. Maximin Latin Hypercube Designs are then discussed in section 3.3, focussing on the academic paper where this type of design was first introduced. A literature review is also presented in this section.

### 3.2 Latin Hypercube Designs

#### 3.2.1 What is a Latin Hypercube Design?

It is clear first of all that the points chosen to be the training inputs should represent all parts of the input space. We refer generally to such designs as *space*

*filling designs*. Obviously, this is easy to do if there is no restriction on how many training inputs to use. For example, we could scatter a large number of points all over the entire input space. However, there is a restriction since the simulator is assumed to be computationally expensive. Therefore, we require the number of inputs needs to be as small as possible, whilst still representing all portions of the input space.

If there is no prior knowledge about the input variable(s) (e.g. if there is no information about whether certain values are more likely to occur than others), then a first approach is to apply simple random sampling. An immediate problem with this kind of sampling is that it may not be representative of the sample space. Stratified random sampling is a good way round this, but unfortunately when the input is of a higher dimension than one, this can also be a problem.

The solution is to apply stratified random sampling to each variable (dimension) in the input, and then randomly permute the order. This is the basis behind selecting design points according to a *Latin Hypercube Design*. An example now follows to aid understanding.

### **3.2.2 An example**

For simplicity, let attention be restricted to two variables (dimensions) in the input, and suppose we require  $n = 3$  training inputs. Let the two variables be ‘wind direction’ and ‘depth of water’. Let the domain of wind direction be  $[0^\circ, 360^\circ)$  and suppose the water depth range goes from 10 metres to 70 metres. Now, it

is likely that the wind is predominant in one direction, but this is not known to us, therefore a uniform distribution must be assumed; that is the wind is as likely to blow in one direction at any one time compared to any other direction. In a similar way, water depth is assumed to have a uniform distribution.

Since  $n = 3$  - that is 3 input values are required to be chosen - the domain of each variable is split up into 3 sections. One value is randomly selected from each sector, for each variable. Denote the two sets of selected points by 1, 2, 3 and  $a, b, c$ . Permuting the elements of the two sets with each other gives:

(1, a)	(1, b)	(1, b)	(1, c)	(1, c)
(2, c)	(2, a)	(2, c)	(2, a)	(2, b)
(3, b)	(3, c)	(3, a)	(3, b)	(3, a)

One of the five permutations is now randomly selected. Suppose the fourth group from the left is chosen. Then the design can be represented as shown in figure 3.1. In essence, the three input values that are chosen to be entered into the simulator are  $(1, c)$ ,  $(2, a)$ , and  $(3, b)$ .

In addition to the five permutations listed above, there is a final possibility, namely  $(1, a)$ ,  $(2, b)$ ,  $(3, c)$ . However this set of pairings are ignored because the 1, 2, 3 and  $a, b, c$  are paired in ascending order and so it is likely not give information about how the two variables behave when they vary in the opposite direction. In other words, if  $(1, a)$ ,  $(2, b)$ ,  $(3, c)$  were to be represented as in figure 3.1, the three pairs would go up the positive diagonal. However, there would be large gaps in the bottom right hand corner and top left hand corner and it would be a mistake to

Figure 3.1: A Latin Hypercube design in two dimensions with  $n=3$ .

adopt a design ignoring whole areas of the design region. Such a design is therefore not space-filling, so we ignore it.

Oakley (1999) summarises the general definition of a LHD. In  $k$  dimensions:

*“Suppose  $\mathbf{x}=x_1, \dots, x_k$  and we wish to draw random values of  $\mathbf{x}$ . For  $i=1, \dots, d$  we divide the sample space of  $x_i$  into  $n$  regions of equal marginal probability. We then draw one random value of  $x_i$  from each region. Then, to obtain one random value of  $\mathbf{x}$ , we sample without replacement from the values  $x_{i_1}, \dots, x_{i_n}$  for  $i=1, \dots, k$ .”*

The key advantage of a LHD is that it ensures that all parts of the domain of each input dimension is well represented, while only requiring a small number of sampled points. It can also work with a high number of dimensions, and compared to alternative methods (for example Monte-Carlo), it is quicker and computationally cheaper to generate. The concept of a Latin Hypercube Design (LHD) was first proposed by McKay et al. (1979). Since then many papers have endorsed the



design, for example Butler (2001), Santner (2003) and Wang (2003).

### 3.3 Maximin Latin Hypercube Designs

#### 3.3.1 What is a *Maximin* Latin Hypercube Design?

A *Maximin* Latin Hypercube Design (MmLHD) is a particular class of the Latin Hypercube Design (LHD) which has been shown to produce a better design than the standard way. A fuller explanation now follows.

Recall the example from the previous subsection. There were five possible sets of pairings, namely:

(1, a)	(1, b)	(1, b)	(1, c)	(1, c)
(2, c)	(2, a)	(2, c)	(2, a)	(2, b)
(3, b)	(3, c)	(3, a)	(3, b)	(3, a)

A LHD consists of randomly selecting one of these five sets. In contrast to random selection, a MmLHD consists of selecting the optimal set based on distances between the three points (suppose each of the five sets of points were to be displayed onto grids equivalent to figure 3.1). Husslage et al. (2006) defines it as follows:

*“A k-dimensional Latin hypercube design (LHD) of n points, is a set of points  $x_i = (x_{i1}, x_{i2}, \dots, x_{ik}) \in \{0, \dots, n-1\}^k$  such that for each dimension j all  $x_{ij}$  are distinct. An LHD is called maximin when the separation distance  $\min_{i \neq j} d(x_i, x_j)$  is maximal among all LHDs of given size n, where d is a certain distance measure.”*

### 3.3.2 Morris & Mitchell (1995)

The intention of this subsection is to give a more technically thorough explanation of Maximin Latin Hypercube Designs (MmLHDs). MmLHDs were first proposed by Morris & Mitchell (1995). In this subsection, we review the methodology of the authors.

#### Setup

The authors begin by stating the setting. There are  $k$  dimensions in the input space, represented collectively by the  $k$ -vector,  $\mathbf{x}=(x^{(1)}, x^{(2)}, \dots, x^{(k)})$ . It is assumed that the simulator output is one-dimensional (ie a scalar), given as  $y$ . Since  $f(\cdot)$  represents the simulator,  $y$  can be written as a function of  $\mathbf{x}$ , ie.  $y = f(\mathbf{x})$ , where  $\mathbf{x} \in T$ , and  $T$  is defined to be the domain or region of interest. The authors limited  $T$  to be the set  $[0, 1]^k$ , which means that for each input value  $\mathbf{x}$ , each of the  $k$  variables must take values between 0 and 1, inclusively. They state that a spatial stochastic process or random function  $Y$  is defined over  $T$  as an initial expression of our uncertainty about  $y$ . The authors state that  $Y$  will be a stationary Gaussian process for which:

$$\text{corr}[Y(x_s), Y(x_t)] = R[d(x_s, x_t)]$$

In other words, the correlation between responses of two input values is a function of some distance defined between those two values. Note that the paper quite often refers to the different  $k$ -vector input values as *sites*. Two different types of distances are used in the paper. They are Rectangular distances and Euclidean

distances, given respectively as:

$$d(x_s, x_t) = \sum_{l=1}^k |x_s^{(l)} - x_t^{(l)}| \quad d(x_s, x_t) = \left[ \sum_{l=1}^k (x_s^{(l)} - x_t^{(l)})^2 \right]^{1/2} \quad (3.1)$$

Similar to other types of designs, there are often a large number (permutations) of possible LHDs which could be used. A normal LHD would just randomly select one. Contrary to this, the paper discusses choosing the LHD, based on some optimality criteria. The resulting LHD is called a *Maximin* LHD.

### Using distances between points to choose the best LHD

For a given design D, we define the *distance* list  $(d_1, d_2, \dots, d_m)$  to be one in which the elements are the distinct values of inter-site distances, sorted from the smallest to the largest. Hence  $m$  can be as large as  ${}^nC_2$  or as small as 1. We also define an *index* list  $(J_1, J_2, \dots, J_m)$ , in which  $J_j$  is the number of pairs of sites in the design separated by distance  $d_j$ . To aid understanding of this, an example now follows.

*Example.* Consider a complete factorial design with  $n = 9$  points in  $k = 2$  dimensions. Pictorially, this means we have nine points arranged in a 3 by 3 grid, labelled as  $x_1, x_2, \dots, x_9$ . Treating the distance between points as Euclidean, we consider all the possible distances for each pair of sites, ie for  $1 \leq s, t \leq 9$ , where  $s \neq t$  (where  $s$  and  $t$  are used as in (3.1)). In total,  ${}^9C_2 = 36$  possible distances were found. Taking the horizontal/vertical distance between two adjacent points to be 1 unit; the diagonal distance would be  $\sqrt{2}$ . Hence, the distance list is as follows:  
 $d_1 = d_2 = \dots = d_{12} = 1$ ,  $d_{13} = d_{14} = \dots = d_{20} = \sqrt{2}$ ,  $d_{21} = d_{22} = \dots = d_{26} = 2$ ,  
 $d_{27} = d_{28} = \dots = d_{34} = \sqrt{5}$ ,  $d_{35} = d_{36} = 2\sqrt{2}$  (note that for convenience, each of

the  $J_j$ s have been chosen such that  $d_j$  corresponds to the size of each). The index list for this example was also easily verified.

The authors stated that design D was called a *Maximin* design (ie the optimal design using the Maximin criteria) if among available it:

(1a) maximises  $d_1$  , and among designs for which this is true;

(1b) minimises  $J_1$ , and among designs for which this is true;

(2a) maximises  $d_2$ , and among designs for which this is true;

(2b) minimises  $J_2$  , and among designs for which this is true;

:

:

(ma) maximises  $d_m$ , and among designs for which this is true;

(mb) minimises  $J_m$ .

Hence, the reason for the name *maximin* comes the above definition of maximising each of the  $d_j$ s and minimising each of the  $J_j$ s. The final component of the definition of this *Maximin* criteria is to define a function which assigns a numerical value for each competing design, as a way of ranking them in order of optimality. The function is derived so that the *Mm* (Maximin) designs have the highest ranking:

$$\phi_p(D) = \left[ \sum_{j=1}^m J_j d_j^{-p} \right]^{1/p}$$

where  $p$  is a positive integer, and  $d_j$  and  $J_j$  characterize the design  $D$ . In essence, the designs that minimize  $\phi_p$  are the optimal designs, ie are the Mm designs.

### 3.3.3 Literature Review

Other authors have considered MmLHDs. They include: Van Dam et al. (2006), Jin et al. (2005) and Ye et al. (2000), among others. All of these papers however have exemplified the use of MmLHDs in contexts where either the number of dimensions is very small (less than five) or the number of k-dimensional input values are small (30 or less). This dissertation is using 5 variables and between 25 and 50 training inputs<sup>1</sup>. Therefore some of the practical techniques offered by these authors may not be deployed for the needs of this dissertation. However, there is another paper (Husslage et al., 2006) which extends the technique for finding the MmLHD for up to ten dimensions and up to 100 design points. The paper even offers a website from which these designs can be downloaded: <http://www.spacefillingdesigns.nl>. This website was used in order to identify the required MmLHD for this dissertation. The next chapter explains this further.

As a final point, it should be noted that there is literature which discredits the need for the maximin criteria for the maximin LHDs. One example is by Joseph & Hung (2007) who instead opt for finding good LHDs by minimizing the pairwise correlations from maximising inter-site distances. They refer to this design as a *Minimax* LHD.

## 3.4 Conclusion

Before the outputs of the training data can be obtained from the simulator, we need a design to know how the training inputs should be chosen from the input

---

<sup>1</sup>The next chapter will discuss why between 25 and 50 values are used.

space. Clearly, they must be chosen to represent every portion of the input space. It turns out that the best design is called a Maximin Latin Hypercube Design.

Latin Hypercube Designs (LHDs) are ones where every portion of the input space is represented. Points using this design are chosen using stratified random sampling in each dimension. The orders of the strata in each dimension are then permuted. This results in many different possible LHDs. The LHD which is used in the end is chosen by randomisation.

The *Maximin* Latin Hypercube Design (MmLHD) is an extension to the traditional LHD. The design follows exactly the same procedure, except for the final step. Instead of randomly choosing a LHD from the possible permutations, the best LHD is chosen according to an optimality criteria based on distances between the points. There is much literature about MmLHDs, with many authors advocating them.

With regard to implementation, the paper by Husslage et al. (2006) offers a website where MmLHDs can be downloaded from. It is <http://www.spacefillingdesigns.nl>.

The MmLHD used in this dissertation will use this website.

# Chapter 4

## Implementation of the Model

### 4.1 Introduction

The purpose of this chapter is threefold. First of all, section 4.2 will tackle the problem of the simulator outputs being multi-dimensional but GEM-SA only accepting training outputs which are scalar. Section 4.3 describes the practicalities of how the training inputs will be obtained from a MmLHD, and how the simulator operates in order to obtain the corresponding training outputs. Finally, sections 4.4 and 4.5 will focus on the issue of how much training data will be needed in order to build the emulator.

### 4.2 Multiple Output

#### 4.2.1 What was decided for this dissertation

Recall that in question 3 of subsection 2.6, one difficulty being faced is the fact that while the simulator output is multi-dimensional, GEM-SA (the emulator building

program) can only accept training outputs that are scalar (ie each output consisting of only one value). O'hagan (2004) states that one way to get round this is to build emulators for each value of the simulator output. However, the author also comments that there is an obvious problem if this is done: since there are very likely to be correlations between the different values of an output, by building separate emulators for each output value we are ignoring these correlations. Since the emulator program being used in this dissertation (GEM-SA) can only deal with scalar outputs, this is a problem which cannot be resolved and so is an obvious limitation to our study.

However, even if we do build emulators for each output value, since each output in our simulator consists of 512 values, this would mean building 512 emulators, which would clearly be inappropriate. These 512 values need to be condensed somehow, or represented in a different way. Various options to do this are now presented.

## Option 1

One option is to pick 10 (say) of the 512 values, evenly spaced between one another. Then, one could build 10 different emulators for each of these 10 different scalar outputs. Building 10 emulators as opposed to 512 emulators is much more reasonable. The advantage of this option is that it gives a useful summary of how the output is behaving for different different doppler frequencies<sup>1</sup>. The disadvantage is that only a very small number of the 512 values of the output are actually

---

<sup>1</sup>If the reader wishes be reminded how doppler frequency is being used in this context, see subsection 1.1.2



used; the vast majority are discarded.

## **Option 2**

Another option is to add up all of the 512 values in the output, to obtain one number. The advantage of this method is that all the values in the output are used, and so changes to any one of them will directly affect their resultant sum. Another advantage is that only one emulator is needed. However, the disadvantage of this method is that even though the sum of the values of each output may change for each input, it would be hard to know at which doppler frequencies the differences would actually be. In other words, what if the emulator performed better in one part of the output, but worse in a different part? Would there be any way of knowing this with at least some degree of certainty? Having just one value would make it difficult to answer these questions.

## **Option 3**

To overcome the problems of the first two options, a third option is considered. It is a combination of the strengths of first two and it is decided that this method will be adopted. In essence, the 512 values for each output are split up into 10 equally sized groups, ignoring the first and last values (so each would contain 51 values). Therefore, the first group includes the 2nd to the 52 values, the second group taking values going from the 53rd value to the 103rd value, etc... . The 51 values in each group are then added up, resulting in ten numbers. Finally, ten emulators are built corresponding to each of these sets of ten outputs.

## 4.2.2 What the Literature Says

It is perhaps important to note that there might be more options available to deal with multiple output in addition to the three listed in the previous subsection. For instance, we could consider reducing the dimensionality by Principal Component Analysis. However, this is difficult here since each value in the output is not a value in the sense of corresponding to 512 different variables (e.g. time, speed, or whatever). The output consists of 512 values of the ‘Power’ corresponding to 512 fixed doppler frequencies.

Literature on dealing with multiple output only discusses the theory behind it, and how emulators might be built which could deal with such type of output. None discussed the separate technique of how one might best represent multi-dimensional output as scalar output, for use in emulators which can only work with such scalar outputs (such as in GEM-SA). A lot of literature on emulators treat the output as scalar from the start (ie the simulator produces scalar outputs). This is most notably true for Oakley (2005), Gosling (2006), O’Hagan (2004), Kennedy & O’Hagan (2001).

Building emulators which can deal with multi-dimensional training output is an ongoing area of research. Such theory will not be used in this dissertation since [as mentioned above] the emulator building program being used (GEM-SA) can only work deal with training output which is scalar. Nonetheless, the reader may wish to read up on some of the research which has begun to look into this, in particular Oakley & O’Hagan (2004) and Conti & O’Hagan (2007).

## 4.3 How to obtain the training data

This section is intended to be a practical guide about how to obtain the training data. In the section that follows this one (section 4.4), it discusses how much training data will be needed. This is not known yet. For convenience, this section will assume it is  $n = 25$ . Obviously the methodology that will be described here can be adapted for any value of  $n$ .

### 4.3.1 The training inputs

The last chapter discussed the background and theoretical principles behind how the training inputs are to be selected, namely by a Maximin Latin Hypercube design (MmLHD). As stated in chapter 3, the website <http://www.spacefillingdesigns.nl> (Husslage et al., 2006) is to be used determine what Latin hypercube design (LHD) is optimal in the sense of being the Maximin Latin hypercube design. The matrix at on page 36 gives this order, as obtained from the website. The columns refer to the five variables (dimensions) of the simulator, namely: radar frequency (Hz), water depth(m), wind speed (m/s), wind direction ( $^{\circ}$ ), S/N ratio (dB).

In order to obtain the initial LHD, a program in R is written. This is shown on pages 37 and 38. Essentially, the program tells R to carry out Latin hypercube sampling, but instead of randomly choosing a permutation (which would be a classical LHD), it chooses the permutation given by the orderings in matrix **Order** (a MmLHD). Explanation is now given to what the code means.

The first paragraph of code simply tells R to split up the interval  $[0,1]$  into 25 equally spaced strata, and then to randomly select 25 numbers (from the uniform distribution), for each of the five variables, with each selected number corresponding to each strata. Essentially, the result are five sets of 25 numbers between in 0 and 1. This first step is necessary because the five domains of the five variables are different.

The second paragraph of code instructs R to take the five sets of 25 numbers between 0 and 1 (labelled as u1 to u5) and stretch out the randomly selected numbers to the different sizes and positions of the five separate variables. So for instance, x1 refers to the first variable (radar frequency). It is known that radar frequency can vary from 6Hz to 30Hz. Similarly for the remaining four variables - water depth(m), wind speed (m/s), wind direction ( $^{\circ}$ ), S/N ratio(dB) - they are known to vary according the amounts shown in the code (x2, x3, x4 and x5, respectively).

The first two lines of the third paragraph of code put the five sets of 25 numbers into a matrix of five columns (labelled as **x.matrix**). Each column now shows a stratified random sample corresponding to the domain of each variable. The third and fourth lines now enters in the matrix of orderings of a MmLHD, as given by the matrix **Order**. The numbers obtained from the website list the orders from 0 to 24. Thus, to get them to go from 1 to 25, it is necessary to add 1 to each of the elements; this is the reason for the last line of code in this paragraph.

The purpose of the fourth and fifth paragraphs of code is to put the matrix labelled as **x.matrix** in the third paragraph of code in the order as given by the matrix **Order**. The final matrix, labelled as **Inputs**, now gives the actual set of 25 inputs according the MmLHD. This final matrix is displayed after the code.

### 4.3.2 The training outputs

With the training inputs now known, we feed each one into the simulator to obtain the training outputs. The procedure of how to do this is not given in the main body of text, since every simulator is different so this procedure will not apply to other simulators. However for those readers who do wish to know this procedure, it is displayed in appendix 8.1.

$$\text{Order} = \begin{pmatrix} 1 & 22 & 13 & 10 & 18 \\ 2 & 1 & 12 & 18 & 22 \\ 3 & 11 & 25 & 11 & 13 \\ 4 & 19 & 16 & 24 & 9 \\ 5 & 12 & 2 & 17 & 7 \\ 6 & 4 & 11 & 5 & 10 \\ 7 & 23 & 18 & 8 & 3 \\ 8 & 13 & 1 & 6 & 20 \\ 9 & 16 & 7 & 22 & 23 \\ 10 & 17 & 3 & 2 & 4 \\ 11 & 2 & 14 & 21 & 8 \\ 12 & 9 & 22 & 23 & 21 \\ 13 & 5 & 17 & 7 & 24 \\ 14 & 25 & 5 & 14 & 12 \\ 15 & 20 & 20 & 3 & 17 \\ 16 & 14 & 23 & 19 & 2 \\ 17 & 7 & 24 & 4 & 6 \\ 18 & 24 & 21 & 20 & 16 \\ 19 & 3 & 4 & 16 & 19 \\ 20 & 8 & 6 & 13 & 1 \\ 21 & 10 & 8 & 1 & 14 \\ 22 & 18 & 10 & 12 & 25 \\ 23 & 15 & 9 & 25 & 11 \\ 24 & 21 & 15 & 9 & 5 \\ 25 & 36 & 19 & 15 & 15 \end{pmatrix}$$

```

# First paragraph of code

z=seq(from=0, to=1-1/25, length=25)

u1=z+runif(25,0,1/25)

u2=z+runif(25,0,1/25)

u3=z+runif(25,0,1/25)

u4=z+runif(25,0,1/25)

u5=z+runif(25,0,1/25)


# Second paragraph of code

x1=qunif(u1,6,30)

x2=qunif(u2,20,200)

x3=qunif(u3,3,30)

x4=qunif(u4,0, 359.99)

x5=qunif(u5,-70,-10)


# Third paragraph of code

x=cbind(x1,x2,x3,x4,x5)

x.matrix=matrix(x,ncol=5)

order1=c(0, 21, 12, 9, 17, 1, 0, 11, 17, 21, 2, 10, 24, 10, 12, 3,
18, 15, 23, 8, 4, 11, 1, 16, 6, 5, 3, 10, 4, 9, 6, 22, 17, 7, 2,
7, 12, 0, 5, 19, 8, 15, 6, 21, 22, 9, 16, 2, 1, 3, 10, 1, 13, 20,
7, 11, 8, 21, 22, 20, 12, 4, 16, 6, 23, 13, 24, 4, 13, 11, 14, 19,
19, 2, 16, 15, 13, 22, 18, 1, 16, 6, 23, 3, 5, 17, 23, 20, 19, 15,
18, 2, 3, 15, 18, 19, 7, 5, 12, 0, 20, 9, 7, 0, 13, 21, 17, 9, 11,

```

```
24, 22, 14, 8, 24, 10, 23, 20, 14, 8, 4, 24, 5, 18, 14, 14)
```

```
order2=matrix(order1,nrow=25,byrow=TRUE)
```

```
order=order2+1
```

```
# Fourth paragraph of code
```

```
frequency=x.matrix[,1][order[,1]]
```

```
water.depth=x.matrix[,2][order[,2]]
```

```
wind.speed=x.matrix[,3][order[,3]]
```

```
wind.direction=x.matrix[,4][order[,4]]
```

```
sn.ratio=x.matrix[,5][order[,5]]
```

```
# Fifth paragraph of code
```

```
inputs=cbind(frequency,water.depth,wind.speed,wind.direction,sn.ratio)
```

```
inputs
```



**Inputs =**

6.357335	175.58645	16.779283	129.715350	−28.84746
7.701746	12.06374	15.907135	244.204042	−19.42361
7.943341	86.40514	29.058661	148.492391	−41.11371
9.123110	146.40991	20.209660	332.917019	−49.90375
10.292044	92.28047	4.429963	235.809921	−54.46376
11.310709	32.69553	13.973434	62.758598	−46.40005
12.229158	177.41804	22.159385	100.209509	−63.39448
13.277933	100.50706	3.571228	74.734431	−23.96119
13.914233	125.75345	10.018838	299.291628	−14.96070
14.691420	133.87444	5.689417	22.091637	−61.39940
15.853295	14.30376	17.310798	286.621416	−50.84814
16.863333	69.30606	25.697441	310.406201	−19.75080
17.613613	39.25364	21.034979	91.113790	−14.28285
19.251793	197.59966	7.778173	188.194605	−43.37787
20.273551	154.75143	24.544354	36.800497	−29.51524
21.075216	110.83937	27.650724	260.627685	−66.15189
22.130946	51.94796	28.439836	46.218985	−57.43263
22.816454	187.62686	24.788154	269.026383	−33.75035
24.157058	26.56621	6.936076	222.227805	−24.96801
24.789427	63.31281	8.516051	178.295252	−68.11394
25.897803	81.16682	10.723276	4.340498	−36.60096
26.631183	143.27490	13.280647	165.420034	−11.56208
27.205603	119.29055	11.842065	337.789123	−44.29900
28.861950	164.49567	18.225690	117.051982	−59.87842
29.273608	46.99118	39.22664233	204.455063	−36.23552

## 4.4 How much training data is required?

### 4.4.1 Introduction

In order to answer the question of how much training data is needed to build the emulators we consider six possible sizes, namely  $n = 25$ ,  $n = 30$ ,  $n = 35$ ,  $n = 40$ ,  $n = 45$  and  $n = 50$ .

The reason for using these particular values of  $n$  is based on the general rule of thumb that the amount of training data required is 10 times the dimensions of the input space (Loeppky, 2008). Since the input consists of 5 dimensions, this means that 50 training inputs are required. However, this is only a rule of thumb. Could an emulator be built with less training data? To answer this question, we consider lower values of  $n$ , as given above. As discussed in section 4.2, the output corresponding to each training input will be represented by 10 values. Therefore, ten emulators are to be built corresponding to the ten sets of outputs. This is done for each of the six values of  $n$ .

For each set of outputs, we determine what value of  $n$  to use by the method of *Cross-validation*.

### 4.4.2 What is Cross-Validation?

Cross-validation can be applied to any dataset. It is a way of determining how good a model is (that is fitted to the dataset) without requiring extra data. The

steps that follow explain how it works in a general setting:

- (1) Remove one point from the data;
- (2) Fit the model (emulator) on the remaining  $n - 1$  inputs and outputs.
- (3) Use the fitted model to predict the point which was removed.
- (4) Compute the difference (or error) between the actual value of the removed point ( $\mathbf{X}_i$ ) and the predicted value ( $\mathbf{x}_i$ ). Square this difference.
- (5) Repeat steps 1 to 4 for the remaining data points.
- (6) Add up all the squares of the errors and divide by  $n$ , to obtain what is known as the *Mean Squared Error*.
- (7) Finally take the square root of the number obtained in (6). This is now known as the *Root Mean Squared Error*.

Thus, the formula for the Cross-validation Root Mean Squared Error (or C-V RMSE) can be expressed as follows:

$$\text{C-V RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\mathbf{X}_i - \mathbf{x}_i)^2}$$

#### **4.4.3 How is the Cross-validation error computed for this dissertation?**

GEM-SA, the name of the emulation program, automatically computes the C-V RMSE whenever an emulator is built. In the above seven steps, the C-V RMSE is computed by the *leave one out* method, that is leaving each point out in turn. GEM-SA can alternatively be specified to *leave out the final 20%* when computing

the C-V RMSE. In other words ‘one point’ from step 1 is replaced by ‘20%’. So in effect, we are removing 20% of the data and using the remaining 80% to predict the removed points. Like *leaving one out*, we repeat this for each 20% portion of the data. This is sometimes used when the computational time is large. Both methods should produce similar results, so it did not matter which is used. For this dissertation, the latter method was chosen.

When GEM-SA builds an emulator for a set of inputs and outputs, it is mentioned above that the C-V RMSE is automatically given. In fact, two other types of cross validation errors are given. These are: the *Cross-Validation Root Mean Relative Error* (Relative C-V RMSE) and the *Cross-Validation Root Mean Standardised Error* (Standardised C-V RMSE). The question now arises, should either of the other two measures of cross-validation be used instead? The answer is yes. Winters (2008) states that the RMSE is sufficient to use when only one model is being checked. When two or more models are being compared, he states that Standardised RMSE should be used. This makes sense because the sets of errors (each corresponding to a dataset and model), will vary differently. By standardising each set of errors, the models can be compared on an equal footing using RMSE. Since no literature for or against the use of Relative C-V RMSE could be found, it was decided that this would be ignored. Therefore, since we are comparing the efficiency of six emulators (ie  $n=25, 30, \dots, 50$ ) for each output, it is most appropriate to use the Standardised C-V RMSE. O’Hagan (2008) also specifically states, in referring to GEM-SA, that the Standardised CV-RMSE should be close to 1.

For simplicity, from this point forward, we refer to the Standardised C-V RMSE as the *cross-validation error*. The cross-validation error is to be computed for all six values of  $n$ , that is  $n = 25, n = 30, \dots, n = 50$ . The six cross-validation errors are then plotted against the corresponding values of  $n$ . This is done for each of the ten sets of emulators, corresponding to the ten outputs. Therefore, we will end up with ten cross-validation plots. It is expected that for each plot, the cross validation will be high for low values of  $n$ . However as  $n$  increases, it is expected that the error will decrease rapidly, after which it should level off. The value of  $n$  where the error begins to level will be the value of  $n$  chosen for that plot. It is decided that the same training inputs must be used for each of the ten plots. Therefore, once the values of the *optimal*  $n$  have been calculated for each of the ten emulators, the highest *optimal*  $n$  value will be the overall  $n$  used. For example, if we found that 3 emulators recorded  $n = 30$ , 5 recorded  $n = 35$  and the remaining 2 recorded  $n = 40$ , then we would choose the highest, that is  $n = 40$ .

It is important to note that in practice, we would not build emulators for all of these values of  $n$ . This is because it requires 225 runs of the simulator ( $25+30+\dots+50$ ), which is a lot and since the simulator is considered computationally expensive, this could be thought of as defeating the whole point of building an emulator in the first place (though 225 runs would still be less than Monte-Carlo). Instead we would use the general rule of thumb as previously stated of  $n = 10 \times$  the number of dimensions in the input (Loeppky, 2008). However, one line of interest in this dissertation is investigating how low the value of  $n$  can go for

the emulator to still be good enough. Hence lower values of  $n$  are considered.

#### 4.4.4 Why the Cross-Validation Error not computed for

$$n > 50$$

One key question is, even if the cross validation error for each plot does level off (as described at the end of the third paragraph of the previous subsection), how do we know that it will continue like this for  $n > 50$ . In other words, how do we know it will not just jump up to a high value at say  $n=55$ , or higher values of  $n$ . We refer once more to the general rule of thumb in choosing  $n$  ( $10 \times$  no. of dimensions of the input space) as stated in Loeppky (2008). The main purpose of that paper was to give a rough statistical justification for this rule of thumb. For the simulator used here, this means that  $n = 50$  training data are considered “good enough” to build emulators from. Hence, values of  $n$  greater than 50 need not be considered.

### 4.5 Results

#### 4.5.1 Line plots of Cross-validation error verses $n$

Table 1 shows the values of the cross-validation error for different values of  $n$ , for each of the ten outputs. Figure 4.1 shows the ten corresponding plots. We refer to each of the ten sets of training output as, *output a*, *output b*, ..., *output j*. So, *output a* refers to the part of the first portion of the output (ie the sum of the 2nd to the 52nd values), *output b* to the second portion of the the output (ie the sum of the 53rd to the 103rd values), etc... .

<b>n</b>	25	30	35	40	45	50
<b>Output a</b>	5.82	3.19	0.50	0.59	0.68	0.96
<b>Output b</b>	6.25	2.57	1.25	0.67	1.25	1.48
<b>Output c</b>	4.27	6.83	2.09	0.57	0.61	1.37
<b>Output d</b>	5.41	1.25	1.93	2.64	0.99	1.65
<b>Output e</b>	4.88	1.95	1.38	0.58	1.32	1.24
<b>Output f</b>	3.87	1.58	1.10	1.98	1.30	1.02
<b>Output g</b>	2.05	4.39	2.28	1.43	2.08	0.50
<b>Output h</b>	4.88	3.11	1.50	0.66	1.15	1.24
<b>Output i</b>	6.94	4.41	1.16	1.11	0.93	0.82
<b>Output j</b>	8.54	2.83	0.18	0.75	0.79	0.82

Table 1: Cross Validation Errors for different values of  $n$ , for each of the ten outputs.

### 4.5.2 Interpretation of the line plots

As stated towards the end of section 4.4, each line plot was expected to have a high cross-validation error for low values of  $n$ , getting smaller as  $n$  increases. Only the line plot for output i (figure 4.1(i)) takes this precise form. Though, most of the line plots take this form in an approximate way. The two exceptions to this were plots of output c and output g, where the cross-validation error for  $n=25$  is less than that of one or more subsequent values of  $n$ .

Recall, that the cross-validation error being used here is the Cross-validation Root Mean squared Standardised Error. This means that values close to 1 indicate that

Figure 4.1: Line plot of cross-validation error verses  $n$  where the training output is (a) output a, (b) output b, (c) output c, (d) output d, (e) output e, (f) output f, (g) output g, (h) output h, (i) output i, (j) output j



the emulator for that particular value of  $n$  is good. It is clear from most of the line plots that when the cross-validation error settles close to 1, it tends to deviate up and down close to 1. To differentiate between deviations close to 1 and deviations not close to 1, it was decided that a cross-validation error which was 1 after rounding to 1 significant figure, meant that the emulator for that value of  $n$  was good enough. For this reason, two horizontal dashed lines with equations  $y = 0.5$  and  $y = 1.5$  were drawn for each plot, since a cross-validation error in the range  $[0.5, 1.5)$  would round to 1, to 1 significant figure.

So for example, for output a (figure 4.1(a)), at  $n = 35$  the cross-validation error is close to 1 and remains close for the three subsequent values of  $n$ . This is because the corresponding points on the plot remain between the two dashed lines, even though there is a slight increase in the cross-validation. Therefore, we say that for an emulator (for a particular output) to be good enough, the cross validation error and all subsequent values must lie in the range  $[0.5, 1.5)$ . While we are orthodox in this manner when cross-validation errors are 1.5 or greater, it is less important if the error is less than 0.5. In fact, this happened only one occasion in output j (figure 4.1(j)) at  $n=35$ , but the cross-validation errors for all subsequent values of  $n$  were close to 1.

Seven of the outputs show that this optimal value of  $n$  is at 35 or 40. As for the three remaining outputs, outputs f and g give optimal  $n$  to be 45 and 50 respectively. For output d, the cross-validation error at  $n=50$  is 1.65, which does not round to 1 to one significant figure. However, it is taken for granted that an em-

ulator built with  $n = 50$  will be good enough (see subsection 4.4.4 for explanation).

Therefore a decision must be made as to what value of  $n$  to choose. With all outputs but two showing the optimal  $n$  to be 35, 40 or 45, is this good enough reason to choose a value of  $n$  less than 50? The answer is no. Outputs d and g, although in the minority, cannot be excluded because the optimal  $n$  is based on all the cross-validation being low enough for all the outputs, not just most.

Therefore,  $n = 50$  is the smallest number of training data required to build an emulator for each of the ten outputs.

## 4.6 Conclusion

In this chapter, we have discussed various options to deal with the problem of the simulator output being multi-dimensional while GEM-SA (the emulator building program) only accepting training outputs which are scalar. It is decided that the best way to deal with this is to split the 512-value simulator output values into 10 equally sized groups, excluding the first and the last values. Then, we say that *output a* consists of the sum of the 2nd to the 52nd values, *output b* consists of the sum of the 53rd to the 103rd values, etc... . Explanation has also been given as to the practicalities of how the training inputs are obtained from a Maximin Latin Hypercube design. In addition, we have also discussed how to conduct runs of the simulator in order to obtain the training outputs.

Finally, the method of cross-validation was used to determine how much training

data was required to build the ten emulators corresponding to each of the ten outputs. Cross-validation is a way of determining how well a model is performing without requiring additional data (e.g. additional runs of the simulator). As  $n$  (the amount of training data) increases, it was expected that the cross-validation error should decrease. Corresponding plots were drawn to represent this information. Most of the plots approximately showed the expected behaviour. For each plot, we chose  $n$  where the line began to flatten. For seven of the plots, this was  $n = 35$  or  $n = 40$ . For one it was  $n = 45$  and for two  $n = 50$ . Since  $n = 50$  is the highest, we require 50 training data to build the 10 emulators.



# Chapter 5

## Diagnostic Checks

### 5.1 Introduction

Diagnostic checks are an important part of any model fitting process. Emulators are no exception. The intention is that the emulator will represent the simulator accurately, but how can one be absolutely sure this is true? Diagnostic checks not only answer this question, but also quantify the performance of the emulator.

Before the diagnostic checks are carried out, it will be necessary to obtain an additional set of data, called ‘validation data’. This is collected by carrying out 20 more runs of the simulator. The inputs are also entered into each of the ten emulators, to obtain ten sets of outputs. The diagnostic checks will use information from the validation data. In total, there will be three diagnostic checks, and these are given as follows:

- (i) The first diagnostic will consists of two sets of plots. The first set is a scatter-plot of the *validation outputs* (simulator) against the *prediction outputs* (emula-

tor), repeated for each of the tens set of outputs. The second set will plot the standardised prediction errors, again for each sets of outputs.

(ii) The second diagnostic computes the Mahalanobis distance between the *validation outputs* and the *prediction outputs*. Extreme values of this distance suggest that the emulator is not representing the simulator to a sufficiently accurate degree.

(iii) As with the first, the third diagnostic check will consist of two plots for each of the ten emulators. The first plot will be a plot of the pivoted cholesky errors against the pivoting order. The second will be a quantile-quantile plot.

## 5.2 Validation Data

All of the diagnostic checks use a new dataset call the *Validation data*. Essentially, 20 more runs of the simulator are carried out, and the emulator is asked to predict what these set of outputs would be based on the same set of inputs. Therefore, the validation data consists of three groups of data:

- *validation inputs*: the input design is the same as was used for the training data inputs.
- *validation outputs*: these are the outputs obtained from running the simulator at the validation inputs.
- *prediction outputs*: these are the prediction outputs (given as the expected values) obtained from the emulator at the validation inputs.

Note that in the context of this dissertation, ten emulators were built as the

simulator output is multi-dimensional. Therefore whenever the word ‘emulator’ is used, this actually refers to each of the ten emulators.

## **5.3 Diagnostic Check 1**

The first diagnostic check is more of a descriptive statistics diagnostic. While the second and third diagnostic checks will go more in depth, the aim here is to get an initial idea of how the emulators are performing. This first diagnostic consists of two sets of plots. The code to reproduce these two plots in R is stored in the appendix, section 8.2.

### **5.3.1 Plot of the emulator’s output against the simulator’s output**

The first plot involves a straight forward comparison of the validation outputs and the prediction outputs. If the emulator is performing well, the prediction outputs should be approximately equal to the validation outputs. This is represented graphically by constructing a scatter plot with one output on the x-axis and the other output on the y-axis, and checking to see if the points lie on or very close to the line  $y = x$ .

### **5.3.2 Plot of the standardised prediction errors**

The second set are each a plot of the standardised prediction errors. The shape of this plot should correspond to the shape of the first plot of this diagnostic check. This is because it uses the same information as the first plot. In other words,

the prediction error is defined as the difference between the validation outputs and the prediction outputs. It is conventional to standardise the prediction errors since the posterior variance will be different at each prediction output. Therefore, the  $i$ th individual standardised error can be abbreviated by

$$D_i(\mathbf{y}^*) = \frac{y_i^* - E[f(\mathbf{x}_i^*)|\mathbf{y}]}{\sqrt{V[f(\mathbf{x}_i^*)|\mathbf{y}]}} \quad (5.1)$$

These standardised prediction errors are then plotted against the index  $i$ . A value of  $D_i(\mathbf{y}^*)$  outside a certain range, say  $(-2,2)$  suggest that the emulator is not predicting that corresponding validation output well enough. One might be willing to accept one or two standardised prediction errors to fall outside  $(-2,2)$ , but any more could raise serious doubt about the predictive performance of the emulator. If this plot does reveal that the emulator is likely to be performing badly, then the particular layout of the points can reveal some information of where the exact problem is, for example the emulator could be under- or over-estimating parameters (see Bastos & O’Hagan, 2008).

## 5.4 Diagnostic Check 2

Knowing the individual standardised prediction errors is useful in one sense, but it may also be useful to quantify the errors in a collective way. One option is to simply add up the squares of  $D_i(\mathbf{y}^*)$ . It can be shown that as the amount of training data gets larger and larger (ie as  $n \rightarrow \infty$ ), this sum converges to a chi-squared distribution with  $m$  degrees of freedom. However, this ignores any correlations between the outputs. To take these into account, the correlations are treated as *Mahalanobis distances*. More formally, a Mahalanobis distance between



the validation outputs and prediction outputs is defined as:

$$D_{MD}(\mathbf{y}^*) = (\mathbf{y}^* - E[f(\mathbf{X}_i^*)|\mathbf{y}])^T (V[f(\mathbf{X}_i^*)|\mathbf{y}])^{-1} (\mathbf{y}^* - E[f(\mathbf{X}_i^*)|\mathbf{y}]) \quad (5.2)$$

where  $E[f(\mathbf{X}_i^*)|\mathbf{y}]$  and  $V[f(\mathbf{X}_i^*)|\mathbf{y}]$ , the respective predictive mean vector and covariance matrix, is given by  $m_1(\cdot)$  and  $v_1(\cdot, \cdot)$  from (2.5). Extreme values of this distance suggest that the emulator is not estimating the simulator to an sufficiently accurate degree.

The distribution of  $D_{MD}(\mathbf{y}^*)$  conditional on  $\mathbf{y}$  and  $b$  can also be specified as follows:

$$\frac{(n-q)}{m(n-q-2)} D_{MD}(f(\mathbf{X}^*))|\mathbf{y}, b \sim F_{m,n-q} \quad (5.3)$$

Note that  $m_1(\cdot)$  and  $v_1(\cdot, \cdot)$ , as mentioned above as being from (2.5), had to be derived. This is because when GEM-SA gave the prediction outputs, only a vector of the variances (as well as the vector of the mean values  $m_1(\cdot)$ ) was specified. In other words, only the diagonal elements of the matrix  $v_1(\cdot, \cdot)$  were given; the off-diagonal elements which represent the correlations between the outputs are not given. This meant that it was necessary to write a function in R, which would build the same emulator as done in GEM-SA. The result would include all the elements of the matrix  $v_1(\cdot, \cdot)$  for the given training data. To ensure the code is correct, the mean vector  $m_1(\cdot)$  is also be computed and this should give exactly the same answer as the prediction outputs as given by GEM-SA. In a similar way, the diagonal elements of  $v_1(\cdot, \cdot)$  as computed from the R code should also be the variances of each predicted output which is also given by GEM-SA. The entire code to this section can be found in the appendix, section 8.3

## 5.5 Diagnostic Check 3

In diagnostic check 2, an extreme value of  $D_{MD}(\mathbf{y}^*)$  (ie very large or very small) suggests that the emulator is not performing well in terms of modelling the input-output relationship of the simulator. If this is the case, individual errors need to be extracted and examined. These will hopefully reveal what might be causing the poor performance. Recall that in diagnostic check 1, individual S.P. (standardised prediction) errors were analysed but correlations between the errors was not accounted for. Allowing for correlations between the S.P. errors is important, as it is not always the case that a good plot of these S.P. errors (ie all or most within the range  $(-2,2)$ ) will mean that the errors are in fact small. For example, if two S.P. errors are small but have opposite signs and are strongly positively correlated, then this is an indication that the emulator may not be performing well.

In order to allow for correlations between S.P. errors, it is necessary to decompose the covariance matrix  $V[f(\mathbf{X}_i^*)|\mathbf{y}]$ . The following definition is used (Bastos & O'Hagan, 2008): *Let  $G$  be a standard deviation matrix such that  $V[f(\mathbf{X}_i^*)|\mathbf{y}] = GG^T$ . Then the vector of the transformed errors,*

$$D_{\mathbf{G}}(\mathbf{y}^*) = G^{-1}(\mathbf{y}^* - E[f(\mathbf{X}_i^*)|\mathbf{y}]), \quad (5.4)$$

*are uncorrelated and have unit variances. If the normality assumption made for the outputs is reasonable, the distribution of each of these errors is a standard Student-t with  $(n-q)$  degrees of freedom.* Note that  $\mathbf{y}^*$  refers to the vector of prediction errors, unlike  $y_i^*$  which refers to the prediction errors individually.

In essence, this diagnostic follows three steps:

- (i) Decompose the covariance matrix.

- (ii) Construct a plot of the transformed standardised prediction errors, after decomposition, against a re-ordered index (known as the pivoting order<sup>1</sup>).
- (iii) Construct a quantile-quantile plot, also using the transformed standardised prediction errors, to check the assumption of normality.

Two additional plots will be constructed in the results section, to check if the mean function is reasonable and to check for stationarity. The transformed standardised prediction errors will not be able to be used in this check. Therefore the (untransformed) standardised prediction errors, as used in diagnostic check 1, can only be used here. The first plot is of the standardised prediction errors against the emulator's predictions. The points in this plot should be randomly scattered about the  $y = 0$  line. If particular kinds of patterns occur, this indicates that there could be a problem with the mean function. One way is to see if the points have errors which are all positive or all negative over a particular range of the x-axis. The assumption of the stationarity will also come into doubt if the errors appear heteroscedastic (most commonly recognised if the points appear to fan out). Finally, if a large portion of the absolute errors are big (or small), then this suggests that the predictive variance has been under-estimated (or over-estimated). The second plot actually consists of five plots made up of plotting the standardised prediction errors against each input. If there are problems with the emulator, this can give clues as to whether certain parts of the input space are the cause.

An addition point to note is that in the results section, only the best emulator and

---

<sup>1</sup>Pivoting order will be defined later.

the worst emulator (corresponding to the emulator with the lowest and highest Mahalanobis distances respectively) will be analysed. This is to avoid this chapter becoming excessively long. Finally as before, the code to reproduce the results in R is stored in the appendix, subsection 8.4.1.

### 5.5.1 (i) Pivoted Cholesky Decomposition

#### Cholesky Decomposition

There are many methods to decompose  $V[f(\mathbf{X}_i^*)|\mathbf{y}]$ , a positive definite matrix.<sup>2</sup> A common approach is the cholesky decomposition, where  $\mathbf{G}$  is a lower triangular matrix. So, denoting  $V[f(\mathbf{X}_i^*)|\mathbf{y}]$  as  $V$ ,  $V = \mathbf{G}\mathbf{G}^T$  looks like:

$$\begin{bmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \dots & v_{nn} \end{bmatrix} = \begin{bmatrix} g_{11} & 0 & \dots & 0 \\ g_{21} & g_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \dots & g_{nn} \end{bmatrix} \begin{bmatrix} g_{11} & g_{21} & \dots & g_{n1} \\ 0 & g_{22} & \dots & g_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & g_{nn} \end{bmatrix}$$

To compute the elements of  $G$ , the matrices on the RHS are multiplied together and each element compared with the corresponding element of the matrix on the LHS. From this, the elements of  $G$  can be determined. The following two formulae generalise this relationship. (Wang & Lui, 2006):

$$g_{ii} = \sqrt{\left(v_{ii} - \sum_{k=1}^{i-1} g_{ik}^2\right)} \quad i = 1, \dots, n \quad (5.5)$$

---

<sup>2</sup>A symmetric matrix  $\mathbf{A}$  is said to be a positive definite matrix if  $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$  for all non-zero  $\mathbf{x}$  (Cook & Upton, 2004).

and

$$g_{ji} = (v_{ji} - \sum_{k=1}^{i-1} g_{jk}g_{ik})/g_{ii} \quad j = i + 1, \dots, n \quad (5.6)$$

Note that if doing this multiplication by hand, the above two formulae would be easy to verify.

By denoting the elements of the vector  $D_{\mathbf{G}}(y^*)$  by  $D_i^C(y^*)$ , Bastos and O'hagan (2008) state that, “ $\mathbf{G}^{-1}$  is also a triangular matrix, and  $D_i^C(y^*)$  is the unique linear combination of the first  $i$  validation errors that is uncorrelated with the first  $i-1$ . Its predictive variance is the conditional variance of the  $i$ th validation error given the preceding  $i - 1$  errors.” Although the result is as desired, producing a set of uncorrelated transformed errors, there is still one problem, as the authors continues: “the decomposition is not invariant to how we order the validation points, and patterns of high or low values have no obvious interpretation.” To overcome this problem, the authors propose what they call “Pivoted Cholesky decomposition”.

### Pivoted Cholesky Decomposition

The Pivoted Cholesky decomposition is an extension of Cholesky decomposition, where the validation data is ordered according to the conditional predictive variance, in descending order. So, the first element is the one with largest variance, the second element is the one with the largest predictive variance conditioned on the first element, and so on. This order is called the pivoting order. The LHS of (5.4) then gets replaced with  $D_i^{PC}(\mathbf{y}^*)$ , and this is known as the vector of the pivoted Cholesky errors<sup>3</sup>.

---

<sup>3</sup>See appendix 8.4.2 for the code used to compute the pivoted Cholesky errors.

Figure 5.1: An example to help understand what is meant by the pivoting order

*Example.* To understand (in terms of the second element) what ‘*the one with largest predictive variance conditioned on the first element*’ means, suppose we have built an emulator based on training data given below. Imagine also that  $v_1$ ,  $v_2$  and  $v_3$  are the outputs to three validation data (given as their expected value). Let the layout of the points be as in figure 5.1 below, with  $v_2$  having the largest variance followed by  $v_1$ , then  $v_3$ . Now as stated before, the first element is the one with the largest variance, namely  $v_2$ . The second is the largest variance, conditional on the first element. In other words, it is the element with the largest variance, given that we know the first element. In effect we are treating  $v_2$ , the first element, as an additional piece of training data. Now, although  $v_1$  had the second largest variance at the start,  $v_3$  is now furthest away validation point from the training data. Hence,  $v_3$  has the largest variance conditional on the first element.

### 5.5.2 (ii) Plot of the Pivoted Cholesky errors against the Pivoting Order

In order to not suggest that the emulator is performing badly, the points should be randomly distributed about the  $y = 0$  line. Too many large errors suggest that the variance has been under-estimated; however, too many small errors suggest the opposite. On top of this, if either situation occurs, this indicates that the

emulator is a non-stationary Gaussian process. Final comments can be made about correlation structure. If the errors are noticeably small (or noticeably large):

- on the far left of the plot, then there is a suggestion that the predictive variance was not estimated well;
- on the far right of the plot, this indicates that the correlation length parameters were under- (or over-) estimated. It could also suggest that the chosen correlation structure is not appropriate.

### 5.5.3 (iii) Quantile-quantile plot

The QQ-plot (quantile-quantile plot) checks that the assumption of normality is valid, or at least that there is not a strong suggestion that it is not valid. If the normality assumption of the simulator outputs holds true, then  $D_G(\mathbf{y}^*) \sim t_{n-q}$  where  $n$  is the number of validation points and  $q = p + 1$  ( $p$  is the dimension number of the input space). This can be verified by checking that the points in the QQ-plot lie close to the line  $y = x$ . If the gradient of the line of best of the points is less than (or greater than) 1, this indicates that the predictive variance was over-estimated (or under-estimated). If the shape of the points exhibit a degree of curvature, this would indicate that normality assumption is likely to be invalid.

## 5.6 Results

Throughout the results section, *emulator* will be used in the context of a particular output. For instance, *emulator a* refers to the emulator build using the training outputs from output a.

### 5.6.1 Roughness parameters

Before analysing the results of the three diagnostic checks, it is worth looking at the roughness parameters, which are estimated when each emulator is built (see section 2.5.1 for a definition). Table 2 below shows the roughness parameters for each emulator. For all ten emulators, the values are small for the first four input variables. However for S/N ratio, the corresponding roughness parameters are much greater, not far off the maximum of 99 for some of the emulators. This suggests that the simulator is much less smooth with respect to S/N ratio compared to any of the other variables.

Table 2: The roughness parameters for each emulator.

Output	Wind speed	Wind dir.	Water depth	Radar freq.	S/N ratio
a	0.00757	0.0304	0.00526	0.223	74.0
b	0.324	0.181	0.0652	0.0705	21.0
c	1.85	0.0620	0.0001	0.767	17.9
d	0.761	0.100	1.46	2.66	3.15
e	0.170	0.0458	0.0001	0.979	41.2
f	0.0151	0.102	0.0661	0.726	24.6
g	1.33	0.558	1.47	1.02	4.50
h	1.89	0.225	0.00251	0.0456	23.3
i	0.0115	0.0509	0.0001	2.23	53.2
j	0.0108	0.158	1.73	0.111	17.3



Figure 5.2: Expected Emulator Output ( $D_i(\mathbf{y}^*)$ ) vs. Observed simulator Output ( $y_i^*$ ) for: (a) emulator i, (b) emulator e; (c) emulator d.

### 5.6.2 Diagnostic check 1

#### Plot of Validation outputs verses Prediction Outputs

The validation output verses the prediction outputs were plotted for each of the ten emulators. The plots can be grouped into three different types, as shown in figure 5.2. The first consists of emulators a, b, c and i; the second consists of e, f, h and j; the third consists of d and g. Figure 5.2 show a plot corresponding to one of the emulators in each respective group. For plot (a) (first group), we can see that the points follow the 45 degree line very closely, suggesting that emulators a, b, c and i appear to have a good predictive performance. For plot (b), the points are not as close to the line, suggesting that emulators e, f, h and j have reasonable predictive performance. Finally for plot (c), the points are far more scattered giving an indication that emulators d and g are performing the least well.

#### Individual Prediction Errors

As with the previous subsection, the emulators were grouped into three types, according to the look of the corresponding plots of the individual prediction er-

Figure 5.3: Index no. vs. standardised prediction error for: (a) emulator i, (b) emulator e; (c) emulator a.

ror against index. Figure 5.3 shows the plot from an emulator of each group. The first group (plot (a)) consists of emulators b, h and i. For this group, each corresponding plot shows only one or two outside the desired range  $(-2,2)$  and of the ones inside the two dashed lines the vast majority are near the  $y = 0$  line. The second group consists of emulators d, e, f, g (plot (b)). As with with the first group, all but a small number of the points are within  $(-2,2)$ . Unlike the first group, the ones that are between  $y = -2$  and  $y = 2$  are more spread out. The final group (a, c, i) is again similar to the first group in that the majority of the points lie between  $-2$  and  $2$ , most of which are close to the  $y = 0$  line. However, the distinction is that at least one of the two or three outliers have a large absolute value, ie in excessive of  $5$ . In subsection 5.3, it was stated that the plot of individual standardised prediction errors should roughly correspond to the plot of the validation outputs verses the prediction errors. This appears to be not completely true. For instance, we can see that while emulator a appeared to perform very well in the first plot (similar to figure 5.2a), we can see that for the second plot (figure 5.3c), it has two enormous outliers. Similar comments can be

made for some of the other emulators. Obviously there is a logical explanation for this. For each emulator prediction (given as its expected value), the variance for each prediction varies. Hence, the order of the errors (each given as an absolute value) will undoubtedly change after standardisation.

### 5.6.3 Diagnostic check 2

Table 3 below shows the observed value of  $D_{MD}(\cdot)$  for all ten emulators. Table 4 shows the expected value, standard deviation, lower quartile, median, upper quartile, and a central 95% credible interval<sup>4</sup> of  $D_{MD}(\cdot)$  (see appendix 8.3 for details of computation).

Table 3: The observed Mahalanobis distance for each emulator.

<b>Emulator</b>	a	b	c	d	e	f	g	h	i	j
<b><math>D_{MD}(\cdot)</math></b>	153.3	56.9	65.0	74.5	86.8	115.0	103.8	38.4	47.3	102.7

Table 4: Summaries of the predictive distribution of the Mahalanobis distance.

	Expected	Std. dev.	1stQ	Median	3rdQ	95% Credible Interval
<b><math>D_{MD}(\cdot)</math></b>	20.0	7.87	14.38	18.74	24.21	(9.64, 38.83)

All of the observed values are far away from the expected value of 20.0, with the exception of emulator h which remains within the bounds of central 95% credible interval of  $D_{MD}(\cdot)$ . Excluding emulator h, it is therefore very clear that none of the emulators are performing well. To explore the cause of the problem in each case, we analyse the individual errors.

---

<sup>4</sup>A *central* 95% credible interval is defined a 95% credible interval where the lower and upper bounds are the 2.5% and 97.5% quantiles.

### 5.6.4 Diagnostic check 3

#### The mean function and Stationarity

The best performing emulator is emulator h while the worst performing is a, since these two have the lowest and highest respective observed Mahalanobis distances. Given this new knowledge, we first return briefly to some other graphical diagnostics involving the standardised prediction errors, since checking for problems with mean function and checking for stationarity using the Pivoted Cholesky errors is harder to interpret.

Figures 5.4a and 5.5a are plots of the individual standardised prediction errors against the expected value of the emulator's predictions, for emulators h and a respectively. First notice that in the plot for emulator a, the standardised errors are systematically negative for a large portion of the emulator's output range. This suggests a problem with mean function. There is no evidence here that this problem is true with emulator h.

For each plot, notice also that there is a fanning out of the points as one goes from left to right (more obvious in 5.4a). This suggests that these standardised errors are heteroscedastic, implying a possible a stationarity problem. To investigate this further, the individual standardised prediction errors ( $D_i(\mathbf{y}^*)$ ) are plotted against each input (figures 5.4(b)-(f) and 5.5(b)-(f)). For both emulators, the points show no obvious pattern for the first four inputs. However, for the fifth input, S/N Ratio, all of the extreme values occur on the right hand side. In essence, there is a breakdown in both emulators when they try to predict for values of S/N ratio

Figure 5.4: Graphical Diagnostics using emulator h: (a) Scatterplot of  $D_i(y^*)$  vs emulator's predictions; (b)-(f) Scatterplot of  $D_i(y^*)$  vs each input.

Figure 5.5: Graphical Diagnostics using emulator a: (a) Scatterplot of  $D_i(y^*)$  vs expected output from emulator; (b)-(f) Scatterplot of  $D_i(y^*)$  vs each input.

between -20 and -10. A solution could be to carry out more training runs for this sub-region of the input space. However from table 2 (subsection 5.6.1), recall that the roughness parameter for S/N ratio for each of the ten emulators is a lot greater than the other four input variables. Thus, the problem is likely to be a conflict between the emulator assuming a linear relationship between S/N ratio and the expected output, while the simulator is assuming a non-linear relationship.

Given this, we reduce the domain of S/N ratio to be between -70 and -20 (instead of -70 to -10) for the validation data only. Three of the validation inputs had an S/N ratio between -20 and -10, so we remove all three. The observed Mahalanobis distances for the ten emulators are now recomputed for the 17 remaining validation data (table 5)<sup>5</sup>. The expected mean value for distribution of  $D_{MD}(\cdot)$  (where  $m$  now equals 17) is also given in table 6. While all the observed Mahalanobis distances are extreme values, all are a lot closer to the revised expected value of 17.0. In fact, 8 of the distances are only just outside the central 95% credible interval bounds. This indicates that the emulators are performing much better compared to before, supporting the claim that a large part of the problems with the emulators occur when S/N ratio is between -20 and -10.

As a final point, it is obvious that the problems stated above to do with the mean function and heteroscedasticity are far less of an issue when we remove those pieces of validation data where the S/N ratio is between -20 and -10 in the validation inputs. This is because the excessively extreme values (in absolute form) of the

---

<sup>5</sup>The values in tables 5 and 6 were computed in exactly the same way as those in tables 3 and 4.

standardised prediction errors in figures 5.4(a) and 5.5(a) refer to two or all three of the removed validation points.

Table 5: The observed Mahalanobis distance for each emulator with only 17 validation points (excluding the outliers).

<b>Emulator</b>	a	b	c	d	e	f	g	h	i	j
<b><math>D_{MD}(\cdot)</math></b>	4.63	3.21	4.34	38.45	69.96	53.35	34.55	3.44	4.57	3.23

Table 6: Summaries of the predictive distribution of the Mahalanobis distance, with only 17 validation points (excluding the outliers).

	Expected	Std. dev.	1stQ	Median	3rdQ	95% Credible Interval
<b><math>D_{MD}(\cdot)</math></b>	17.0	7.08	11.94	15.84	30.19	(6.69, 34.01)

For the remaining part of this subsection on diagnostic check 3, we continue to use emulators h and a in our analysis as the best and worst emulators, respectively. In other words, we revert back to using the complete validation data (all 20 pieces) and continue to use the results from the first table of observed Mahalanobis distances. We could argue in some sense that we should use the best and worst emulators based on the reduced validation data (table 5). However, in removing the three validation inputs, the input design is now not a Maximin Latin Hypercube Design (MmLHD). If we obtained a MmLHD for the validation inputs reducing the S/N ratio upper limit to -20, and computed the observed Mahalanobis distances (based on revised validation outputs and prediction outputs), we may find that a different pair of emulators are the best and worst based on this diagnostic. Therefore, it is safest to revert back the results of table 3, and base the best and worst emulators on the original Mahalanobis distances.

Figure 5.6:  $D_i^{PC}((y)^*)$  against Pivoting Order (using pivoted Cholesky errors) for  
(a) emulator h ; (b) emulator a.

### Correlation structure

Here, we make comments about the correlation structure using the pivoted Cholesky errors ( $D_i^{PC}(\mathbf{y}^*)$ ) which are uncorrelated. Figure 5.6 displays the plot of the pivoted Cholesky errors against the pivoting order for emulators h (left) and a (right). For each plot, there is large error on the far right hand side. This most likely suggests that there is a problem with the correlation structure. For emulator h only, there is another large error at the beginning of the plot. This adds value to the non-stationarity claim or could imply that the predictive variance has been poorly estimated.

In both cases, there are a large number (around two-thirds) of small errors, suggesting that the variance may have been over-estimated. This gives further evidence for the non-stationarity claim of the simulator.



When we remove the 3 pieces of validation data, corresponding to when S/N ratio is between -20 and -10 in the validation input, we find a different picture emerges. Both plots can be found in the appendix (figure 8.1 in subsection 8.4). For each plot there is only one large value on the far left or far right of the plot, but both are not large enough to imply any problems. What is most noticeable however is the fact that points lie very close to the  $y = 0$  line, instead of being randomly scattered about this line, as we would hope. In other words, most of the individual errors are small in absolute terms. This backs up the claim of the previous paragraph that the predictive variance may have been over-estimated.

### **Quantile-quantile plot**

Figure 5.7 shows the quantile-quantile plots for emulators h (left) and a (right). For emulator h plot, there is curvature in the points, suggesting that simulator outputs do not follow a normal distribution. The emulator a plot is worse. Although most of the points do roughly follow the 45 degree line, there exists two very extreme points at the bottom of the plot. It is almost certain that these two points are the same two extreme points as seen in figure 5.6(b). This either indicates that there is a problem with non-stationarity or problems with local fitting. Given the preceding discussion, both are likely.

As with the plot of the individual pivoted Cholesky errors, we construct a Quantile-quantile plot based on the 17 pieces of validation data where S/N ratio is not between -20 and -10 in the validation inputs. The revised plots are not shown here, but can be found in the appendix, in subsection 8.4 (figure 8.2). For both

Figure 5.7: Quantile-quantile plot (using Cholesky errors) for: (a) emulator h; (b) emulator a.

emulators, this revised plot is much better. We see much less curvature than corresponding plot for emulator h (figure 5.7(a)), and the two extreme points corresponding to the plot for emulator a (figure 5.7(b)) are no longer present. Excluding the three outlierish pieces of validation data, we have no reason to doubt the normality assumption of the simulator outputs. It should be noted also that for both revised plots (figures 8.2(a)&(b)) the gradient of the line the points cluster around appears to be less than one. For both emulators therefore, this gives further weight to the claim that the predictive variance has been over-estimated.

## 5.7 Conclusion

Three groups of diagnostic checks have been presented in this chapter, to determine and quantify to what extent the emulator is an accurate substitute for the simulator.

**Diagnostic Check 1.** In the first diagnostic, we plotted the validation outputs (simulator) against the prediction outputs (emulator). Here, we found that most emulators appeared to be performing adequately. The second set of plots consisted of plotting the standardised individual prediction errors against the index. Again, from these plots, most emulators seemed to be performing reasonably well. However, it was evident in most of them that there were 2 or 3 extremely large errors (ie with absolute value much greater than 2).

**Diagnostic Check 2.** For the second diagnostic check we compute the Mahalanobis Distance, which for each emulator should be close to the expected value, 20.0 of the corresponding distribution, denoted as  $D_{MD}(\cdot)$ . With the exception of emulator h, the observed Mahalanobis distances for all the emulators were considered extreme values in the sense that they were outside the bounds of the central 95% credible interval of  $D_{MD}(\cdot)$ , (9.64, 38.83). In addition, most were in fact very far away from the limits of the bounds, for instance 7 of the observed distances were between 60 and 160.

Using the results from the second diagnostic, emulator h had an observed Mahalanobis distance closes to the expected distribution value of 20.0 (which was 38.4), and emulator a recorded a value furthest away from 20.0 (which was 153.3). Hence, we refer to emulators h and a as the best and worst emulators, respectively. Diagnostic 3 based most of its analysis on these two emulators.

**Diagnostic Check 3.** For the third diagnostic, we returned briefly to the stan-

standardised prediction errors. For both emulators h and a, we plotted the individual standardised prediction errors, first against the prediction output (ie from the emulator), and second against each input variable. We concluded the following:

- (i) there is a problem with the mean function;
- (ii) the standardised prediction errors are heteroscedastic, implying a possible stationarity problem;
- (iii) the standardised prediction errors are large when S/N ratio is between -20 and -10.

To investigate point (iii) further, the observed Mahalanobis distances were recalculated (from the second diagnostic check), for the 17 pieces of validation data where the corresponding validation inputs had a value of S/N ratio not between -20 and -10. We discovered that the observed Mahalanobis distances were all significantly less than before, indicating that the emulator is indeed having a modelling problem when S/N ratio is between -20 and -10. The problems stated in points (i) and (ii) above result from this problem too. In terms of the actual values of the revised Mahalanobis distances, although all are considered extreme, 8 of them lie only just outside the bounds of the corresponding central 95% credible interval, given as (6.69,34.01). It is therefore justifiable to suggest from this that the emulators are in general performing reasonable.

The other part of the third diagnostic plotted the pivoted Cholesky errors against the pivoting order, for emulators h and a, based on the complete validation set. This revealed that there may be possible problems with the correlation structure,

that the stationarity assumption is cast in doubt, and the predictive variance seems to have been poorly estimated. A quantile-quantile plot was also constructed for both emulators. With curvature being evident in one of the plots, and two very extreme values present in the other, the normality assumption of the simulator outputs seemed doubtful. However, as before we repeated these analyses using the validation data without the three outliers occurring when S/N ratio is between -20 and -10. We found that all of the possible problems associated with both the plot of the pivoted Cholesky errors and the quantile-quantile plot were much less apparent. The only issue which did remain was to do with the predictive variance appearing to have been over-estimated.



# Chapter 6

## Sensitivity Analysis

### 6.1 Introduction

Once each of the ten emulators are built and diagnostic checks are sufficient, it will then be possible to answer the question of whether HF radar backscatter (the simulator output) is sensitive to changes in wind-speed (one of the five simulator input variables). At this point, it is important to recap the conclusions of the previous chapter. The diagnostic checks showed that there is evidence that the emulators are performing reasonable, given that we do not include points corresponding to -20 to -10 in the S/N ratio. However, since the training data does include data where S/N ratio is between -20 and -10, and since sensitivity analysis is carried out on the emulators built from the training data, the results should be interpreted with caution.

An overview of the chapter now follows. Section 6.2 outlines the theory and practice of SA. Section 6.3 shows the results when sensitivity analysis is performed

using emulators h and a, the best and worst performing emulators (from table 3).

## 6.2 Theory and Background

### 6.2.1 What is sensitivity analysis?

Sensitivity analysis is concerned with knowing if the output of a model<sup>1</sup> is sensitive to changes in each input variable. For example, imagine a model has two input variables,  $x_1$  and  $x_2$  with the output represented by  $y$ . Averaging over  $x_1$ , suppose that the model was run several times for a variety of values of  $x_2$ . If the  $y$  was observed to vary for these values of  $x_2$ , then one would say that  $y$  is sensitive to changes in  $x_2$ . Additionally, suppose we averaged over  $x_2$  and the model was run at different values of  $x_1$ . If  $y$  was observed to vary very little, then one would say that  $y$  is not sensitive to changes in  $x_1$ . Note how we *average over*  $x_1$  to determine how  $y$  is sensitive to change for different  $x_2$  (similarly if we interchange  $x_1$  and  $x_2$ ). It might seem better to instead *fix*  $x_1$ , say at its central value. However if we do this, we would underestimate the influence of this input.

Imagine now that SA is being performed on a function  $y = \eta(x_1, x_2, x_3)$ , so we are interested to know how  $y$  is sensitive to changes in each of  $x_1$ ,  $x_2$  and  $x_3$ . To explore say how  $y$  changes with respect to  $x_1$ , one option is to fix  $x_2$  and  $x_3$  at say the respective central values. As explained before, a better option is to average over them. So, instead of using  $\eta(x_1, x_2 = a, x_3 = b)$  and seeing how it varies for

---

<sup>1</sup>A model in this circumstance does not refer to any type, ie. it could be statistical or deterministic.



different  $x_1$ , we use:

$$E[y|x_1] = \int \int \eta(x_1, x_2, x_3) f(x_2, x_3) . dx_2 dx_3,$$

where  $f(x_2, x_3)$  is the probability distribution of  $x_2$  and  $x_3$ . In general, we refer to  $E[y|x_i]$  as the expected output conditional on  $x_i$ , averaging over the other inputs.

In the context of this dissertation, even though sensitivity analysis is only required for the input variable ‘wind speed’, comments will be also be made with regard to the other four variables.

### 6.2.2 The Sensitivity Index

In this section, we review the methodology of Santner (2003). Sensitivity can be quantified by specifying the percentage of variance which is apportioned to each input variable. If  $V_i (= Var_{x_i}(E[y|x_i]))$  denotes the variance of the main effect  $x_i$ , the *first order sensitivity index* ( $S_i$ ) for this input is:

$$S_i = \frac{V_i}{V}.$$

It measures the main effect of  $x_i$  on the output; in other words it measures the proportion of the variance  $V$  which is due to  $x_i$ . For  $i < j$ ,  $S_{ij}$  is the *second-order sensitivity index*, and measures the interaction effect due to inputs  $x_i$  and  $x_j$ ; in other words, it measures the proportion of the total variance above that of their main effects. We write it as:

$$S_{ij} = \frac{V_{ij}}{V}.$$

This can be extended to higher orders. In general, for any  $s = 1, \dots, p$  and  $1 \leq i_1 < \dots < i_s < \dots \leq p$ , the  $s$ -th order sensitivity index is defined by:

$$S_{i_1, \dots, i_s} = \frac{V_{i_1, \dots, i_s}}{V}.$$

Santner (2003) states that by construction, the sensitivity indices satisfy:

$$\sum_{i=1}^p S_i + \sum_{1 \leq i < j \leq p} S_{ij} + \dots S_{1,2,\dots,p} = 1$$

In GEM-SA, sensitivity indices for all orders are taken into account, but only the ones for first-order and second-order are actually displayed.

### 6.2.3 Total Effects

The total sensitivity of an input  $x_i$  is defined as the sum of all orders of sensitivity indices involving that input. Therefore, if  $T_i$  denotes the total sensitivity due to input  $x_i$  ( $1 \leq \dots \leq i \leq \dots p$ ), then:

$$\begin{aligned} T_i = & S_i + [S_{1,i} + S_{2,i} + \dots + S_{i-1,i}] + [S_{i,i+1} + \dots S_{i,q}] \\ & + [\text{all higher order sensitivity indices which include } i \text{ in the index}] \end{aligned}$$

Or in more mathematical terms:

$$T_i = S_i + \sum_{j>i} S_{ij} + \sum_{j<i} S_{ji} + \dots + S_{1,2,\dots,p}. \quad (6.1)$$

It is easy to see from (6.1) that the difference between  $T_i$  and  $S_i$  measures the influence of  $x_i$  due to the total of all the other second or higher order interactions involving  $x_i$ .

### 6.2.4 How sensitivity analysis is summarised in GEM-SA

When an emulator is built in GEM-SA, it produces two windows of output for SA. The first is a tabulated form and the other is graphical.

### Tabulated Form

The output window showing the tabulated form gives two columns for each  $x_i$ . The first shows the first order (and second order if specified in the setup) sensitivity indices. The second column gives the  $p$  values of  $T_i$ .

### Graphical Form

The output window showing the graphical form gives  $p$  plots, one for each of the inputs. For the  $i$ th input, it is a plot of  $E[y|x_i]$  against  $x_i$ , where as stated earlier,  $E[y|x_i]$  is the expected output conditional on  $x_i$  averaging over all the other inputs. In this plot GEM-SA produces a band of lines, rather than a single line, since  $\eta(\cdot)$  is uncertainty. The thickness of the band is an indicator of the emulator uncertainty. If the band is approximately horizontal, this indicates that the output is not sensitive to changes in that input. On the other hand, if the band has noticeable vertical movement, this indicates the converse.

### 6.2.5 Code Uncertainty

When carrying out SA on an emulator, an extra source of uncertainty is produced. This results from the fact that if SA had been carried out on the simulator directly, the values of  $S_i$  and  $T_i$  would be different than those given by the emulator, precisely because the emulator is only an approximation of the simulator. This extra source of uncertainty is called *code uncertainty*. The code uncertainty will be negligible as long as the emulator accurately represents the input-output relationship of the simulator.

## 6.3 Results

Throughout this section, sensitivity analysis will only be conducted on emulators h and a, which as stated earlier are the best and worst performing emulators respectively. This is because emulators h and a recorded the highest and lowest respective observed Mahalanobis distances (table 3).

### 6.3.1 Exploratory Scatter Plots

Figures 6.1 and 6.2 show scatter-plots of each input against output h and output a respectively. In figure 6.1, it can be seen that for the first four inputs (Frequency, Water Depth, Wind Speed and Wind Direction), all but seven of the training points roughly lie in a horizontal straight line. For the plot of input 5 versus output h, there is a clear non-linear relationship between the two axes, with none or very few points straying off the general trend of the points. For figure 6.2, exactly the same comments can be made. These comments are consistent with the interpretations of the roughness parameters (table 2) for both simulators h and a, which are much higher for the fifth ones (23.33 and 74.01 respectively) than the remaining four roughness parameters in each case (all less than 2).

### 6.3.2 Main Effect Plots

The main effect plots are shown in figures 6.3 and 6.4 for simulators h and a respectively. As with the previous figures, we can make similar comments about linearity of the plots of the first four inputs (for both 6.3 and 6.4), and non-linearity of the plot of the fifth input. For the plots of the first four inputs, we can also say that since the bands are approximately horizontal, this suggests that

the HF radar backscatter (simulator output) is not sensitive to changes in each of the four simulator inputs. For the fifth input, the fact that the band does not remain horizontal throughout the entire input range gives a suggestion that HF radar backscatter is sensitive to changes in S/N Ratio.

A final interesting observation is that the actual width of each band is wider for each of the five plots in figure 6.3 than the corresponding plots in figure 6.4. This implies that there is greater uncertainty in emulator h than emulator a. This is unusual, since emulator h is regarded as being the best performing emulator, while emulator a is regarded as being the worst performing.

### **6.3.3 Table of variances and Total Effect**

Tables 5 and 6 show the variances and total effects for the all main effects and some interactions for simulators h and a respectively. For table 5, the main effects sum to 89.00% of the total variance. The greatest main effect by far is S/N Ratio (input 5), which is consistent with the corresponding plots in figures 6.3 and 6.4. The total effects for Frequency and Water Depth are significantly greater than their individual contributions to the variance. This implies that interactions of each of these with other terms are present, and not surprisingly the strongest interaction terms are with S/N ratio in each case. Therefore, there is evidence that HF radar backscatter is partially sensitive to changes in Frequency and Water depth.

In table 6, the dominance of S/N ratio is even stronger, with it contributing 97.74%

Figure 6.1: Scatterplot of each input verses output  $h$ .

Figure 6.2: Scatterplot of each input verses output  $a$ .

Figure 6.3: Main Effect Plots for simulator h.

Figure 6.4: Main Effect Plots for simulator a.

to the variance. The only possible interaction term is between Wind direction and S/N ratio, however its contribution to the variance is very small.

What is clear from both tables is that HF radar backscatter is not sensitive to changes in wind speed.

Table 5. Table of variances and total effect for simulator h.

<i>Input Variable</i>	<i>Variance (%)</i>	<i>Total Effect</i>
Frequency	0.66	9.15
Water Depth	0.61	3.91
Wind Speed	0.04	0.12
Wind Direction	0.10	1.34
S/N Ratio	87.59	97.89
Frequency.S/N Ratio	6.45	
Water Depth.S/N Ratio	1.99	

Table 6. Table of variances and total effect for simulator a.

<i>Input Variable</i>	<i>Variance (%)</i>	<i>Total Effect</i>
Frequency	0.06	0.18
Water Depth	0.03	0.21
Wind Speed	0.02	0.08
Wind Direction	0.12	1.82
S/N Ratio	97.74	99.77
Wind direction.S/N Ratio	1.67	



## 6.4 Conclusion

Sensitivity analysis (SA) is concerned with explored how the output of a model is sensitive to changes in each input variable. We have found no evidence that HF radar backscatter (the simulator output) is sensitive to changes in the wind speed input.



# Chapter 7

## Overall Conclusion and Discussion

### 7.1 Overall Conclusion

The purpose of this dissertation was to carry out sensitivity analysis to explore the wind speed (one of the simulator inputs) dependencies in HF radar backscatter (simulator output). We have found no evidence that HF radar backscatter is sensitive to changes in wind speed.

In reaching this answer, we first substituted the simulator for an Gaussian process emulator. In order to build the emulator, we found that we needed 50 training data, where the best design for the training inputs was a Maximin Latin Hypercube Design. For each of the training outputs, we represented the 512 output values by 10 values, and built ten emulators based on these output values. Diagnostic checks were carried out on the ten emulators. They concluded that apart

from problems the emulators were having when S/N ratio was between -20 and -10, they were in general performing reasonable. Given this, sensitivity analysis was able to be performed on the ten emulators, which enabled the answer (as given in the first paragraph) to be reached.

## 7.2 Limitations, further discussion

There are a number of limitations to this dissertation. First of all, there are a number of potential problems in using the Gaussian process in building the emulator. Bastos & O'hagan (2008) state that the assumption of a stationary Gaussian process with particular mean function and covariance structures may be inappropriate. In addition, if the training data are poor representations of the input-output spaces of the simulator, this can lead to poor estimates of the model parameters. These two reasons can result in the emulator poorly representing the input-output relationship of the simulator.

A second area of concern was the fact that it was difficult to know how to represent the simulator output which consisted of 512 values. In this dissertation, we grouped the values of each output into sets of 51 values in each (ignoring the first and last values). The values in each set were then summed, resulting in ten different numbers. This is clearly insufficient to describe how the power is changing for different doppler frequencies. However even if we did build 512 emulators, one for each output value, we may be ignoring information about how different output values are correlated with each other. This is a limitation which we could do nothing about, since GEM-SA (the emulator building program) can only deal

with outputs which are scalar. Even still, supposing such multi-dimensional output emulators could be built, there would still remain the difficulty of how to carry out and interpret sensitivity analysis when the output consists of many values. Current research is being carried out into building emulators which can deal with multi-dimensional outputs, and how to carry out subsequent sensitivity analysis.

Finally, the diagnostic checks revealed that there was evidence that the ten emulators used were performing badly when S/N ratio was between -20 and -10. As stated in subsection 5.6.4, this was realised by removing three pieces of the validation data, which had S/N ratio between -20 and -10. To be completely sure however, it is recommended that the 50 training data outputs are re-computed using the same same input design and same domains of each of the inputs, except for S/N ratio which is suggested to be limited to be between -70 and -20. In fact, the main effects plots for simulators h and a showed that the relationship between the output and S/N ratio is non-linear from -30. So, it would in fact be better to reduce this sub-domain even further to be between -70 and -30. It is then expected that diagnostic checks would be significantly improved, and we would be more confident with the results of the subsequent sensitivity analysis.



# Chapter 8

## Appendix

### 8.1 How to obtain the training outputs

#### 8.1.1 Instructions for how to use the simulator

How to use FORTRAM program to carry out the simulations

(1) Go into the putty program (note that putty can be downloaded from [www.putty.org](http://www.putty.org)).

(2) Type `acms1.shef.ac.uk` into the Host name box, then press 'open'.

(3) Login username and password.

(4) Type the following instructions:

> `pwd` (says where you are - ie what directory you're in)

> `ls`

```

> cd dir-for-lrw

> pwd

> ls -l

> ./run_model

> ENTER      (use default for model pointer - which is one)

> test1      (enter a name, it can be anything, e.g. 'simulation')

> Y

```

(5) Input values: recall that there are five input values:

wavenumber (k); wave direction (theta); short-wave directional spreading (s); wind speed (U); wind direction (d). For first two inputs, we do not input these into the model as they are done automatically for different values simultaneously. For the remaining three (s, U and d), enter different values of these:

```

> ENTER  (Radar frequency - use default)

> N

> ENTER  (water depth - use default)

> [wind speed input value]

> [wind direction input value]

> [direction spread input value]

> ENTER  [S/N ratio - use default]

```

(6) Suppose we've called the file input values test1 as above.

Then type the following command:



```
> more model/BAtest1.SIML
```

(7) Hold down the enter key on the keyboard to reveal the entire set of frequencies (x-values) and powers (y-values) to construct the power spectrum plot for that set of input values.

(8) To transfer the data to excel, it's not possible to use copy and paste, as that will only paste both columns of data into one excel column. Instead you need to go to the psftp.exe program on the desktop (like putty, it's very easy to download for free from the internet).

(9) In the first command line, type 'open stp07er@acms1.shef.ac.uk'. On the next line, type the same password as used for putty, ie aditl0m.

(10) Type the following commands:

```
> cd dir-for-lrw  
  
> cd model  
  
> get BAtest1.SIML
```

(11) The file should now be copied over into the home computer. To find where it is stored, do a search from the start menu to locate which folder it's in. Though, in this test run, it went to

U:/ManXP/Desktop.

(12) Open excel, select 'data' from the menu, and then 'Import External Data'. Go to U:/ManXP/Desktop, select BAtest1.SIML (or whatever it's called), and then 'open'.

(13) In the next screen, make sure 'fixed width' is selected, then 'next' and finally 'finish'.

### 8.1.2 How to obtain the ten outputs

Recall, that the 25 outputs are stored in Excel, with each output having its own sheet (each with its own tab). Ten extra sheets are now created, where the sum of the values from each of the ten groups will be stored. In order to transfer this information to the ten newly created sheets, it is necessary to use a macro. Otherwise, it would be necessary to type out the required formula for each output separately, which would be very time consuming.

Instead, the macro enables this transfer of information to be done instantaneously and simultaneously for all 25 outputs. The code of the actual macro (with explanation) used is stored in the next subsection (appendix 8.1.3).

Having obtained the ten sets of outputs, each set of outputs is now copied over to a text file, from which the outputs can be read into GEM-SA. Note that the

inputs are also copied over to a text file for the same reason.

### 8.1.3 Excel macro

The code below displays the code for the macro, as explained in the previous subsection. . ‘*nnum* = 35’ refers to the 35 sheets in the excel document: the first ten sheets store the scalar outputs (the sums of the values of each output group), while sheet 11 to 35 correspond to the location of each of the 25 outputs.

In line 5, ‘*c2*’ refers to the cell number where the sum of the values of the first output group has been stored. Essentially, the values for the first output group are stored in cells *b2* to *b52*. The formula for sum, for the set of values for each output, is entered into the cell *c2* for each sheet before the macro is run on sheet number 1.

To obtain the output values of the second group, ‘*c2*’ is changed to ‘*c53*’, then the macro is run again on the second sheet. This is repeated for the remaining output groups, ie ‘*c53*’ is change to ‘*c104*’, then ‘*c155*’, etc...

```
Sub CreatFormula()  
  
    Dim SheetNames() As String  
  
    Dim i As Integer  
  
    nnum = 35  
  
    Range("c2").Select
```

```

SheetCount = ActiveWorkbook.Sheets.Count

ReDim SheetNames(1 To SheetCount)


For i = 1 To SheetCount

    SheetNames(i) = ActiveWorkbook.Sheets(i).Name

Next i


For i = 11 To nnum

    j = IIf(11 - i = 0, Space(0), Str(11 - i))

    ActiveCell.FormulaR1C1 = "=" + SheetNames(i) + "!R[" + j + "]C"

    ActiveCell.Offset(1, 0).Select

Next

End Sub

```

## 8.2 R Code for Diagnostic Check 1

For this diagnostic, emulators a, d, e and i are used. The code below is written four times, where for each the ? is replaced by each of the four letters.

```

inputs=read.table("U:/Dissertation/Emulator/Output_?/
    validation_inputs.txt",header=F)

val?_simulator=read.table("U:/Dissertation/
    Emulator/Output_?/4y?.txt",header=F)

val?_emulatormean=read.table("U:/Dissertation/Emulator/
    Output_?/2mean_?_GEMSA.txt",header=F)

```

```

val?_emulatorvariance=read.table("U:/Dissertation/

    Emulator/Output_?/3var_?_GEMSA.txt",header=F)

spe_?=(val?_simulator-val?_emulatoremean)/

    sqrt(val?_emulatorvariance)

```

For 5.2, the code below is repeated three times with i, d and e replacing ?:

```

matplot(val?_simulator,val?_emulatoremean,xlim=c(0,500),

ylim=c(0,500), xlab="Observed Simulator Output", ylab="Expected

Emulator Output",pch=16) abline(0,1,lty=3,col=2)

```

For 5.3, the code below is repeated three times with a, e and i replacing ?:

```

index=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)

matplot(index,spe_i, ylim=c(-8,8), main="(a)", xlab="Index

number", ylab="Standardised Prediction Error",pch=16) abline(h=0)

abline(h=-2,lty=3,col=2) abline(h=2,lty=3,col=2)

```

## 8.3 R Code for Diagnostic Check 2

All of the code given here was used for emulator a. Except for the letter, exactly the same code was used for the other 9 emulators.

### 8.3.1 The mean function, $E[f(\mathbf{X}_i^*|\mathbf{y})]$

```

m1a=function(x)

{

train.input=read.table("U:/Dissertation/Emulator/inputs50.txt")

training.input=as.matrix(train.input)

```

```

h=matrix(c(1,x),nrow=6,byrow=FALSE)

H=matrix(c(rep(1,50),training.input),nrow=50,byrow=FALSE)

B=diag(c(0.007568/(max(training.input[,1])-min(training.input[,1]))^2,
0.0304088/(max(training.input[,2])-min(training.input[,2]))^2,
0.005258/(max(training.input[,3])-min(training.input[,3]))^2,
0.223446/(max(training.input[,4])-min(training.input[,4]))^2,
74.0081/(max(training.input[,5])-min(training.input[,5]))^2))

A=matrix(0,nrow=50,ncol=50)

for(i in 1:50)
{
for(j in 1:50)
{
A[i,j]=exp(-(t(training.input[i,]-training.input[j,]))
            %*%B%*(training.input[i,]-training.input[j,]))
}
}

training.outputa=as.matrix(read.table("U:/Dissertation/
            Emulator/output50a.txt"))

beta.hat=solve(t(H)%*%solve(A)%*%H)%*%t(H)%*%solve(A)%*%training.outputa

tx=matrix(0,nrow=50,ncol=1)

for(i in 1:50)
{
tx[i,]=exp(-(t(x-training.input[i,]))%*%B%*(x-training.input[i,]))
}

```

```

first=t(h)\%*\%beta.hat

second=t(tx)\%*\%solve(A)\%*\%(training.outputa - (H%\%beta.hat))

first + second }

validation.input=as.matrix(read.table("U:/Dissertation

                                /Emulator/validationa_simulator.txt"))

mean.a.R=matrix(0,nrow=20,ncol=1)

for(i in 1:20)

{

mean.a.R[i]=m1a(validation.input[1,]

}

```

*# To check that this code for the mean vector is correct, it should be equal to the vector of predictions (given as the expected value for each) produced by GEM-SA. All emulators showed this to be true.*

```

mean.a.GEMSA=as.matrix(read.table("U:/Dissertation/Emulator/

                                Output_a/2mean_a_GEMSA.txt"))

mean.a.compare=cbind(mean.a.R,mean.a.GEMSA)

mean.a.compare

mean.a.R - mean.a.GEMSA

```

### 8.3.2 The covariance matrix $V[f(\mathbf{X}_i^*|\mathbf{y})]$

```

v1a=function(x,y)

{

train.input=read.table("U:/Dissertation/Emulator/inputs50.txt")

training.input=as.matrix(train.input)

```

```

h=matrix(c(1,x),nrow=6,byrow=FALSE)

H=matrix(c(rep(1,50),training.input),nrow=50,byrow=FALSE)

B=diag(c(0.007568/(max(training.input[,1])-min(training.input[,1]))^2,
0.0304088/(max(training.input[,2])-min(training.input[,2]))^2,
0.005258/(max(training.input[,3])-min(training.input[,3]))^2,
0.223446/(max(training.input[,4])-min(training.input[,4]))^2,
74.0081/(max(training.input[,5])-min(training.input[,5]))^2))

A=matrix(0,nrow=50,ncol=50) for(i in 1:50)
{
for(j in 1:50)
{
A[i,j]=exp(-(t(training.input[i,]-training.input[j,]))
%%B%(training.input[i,]-training.input[j,]))
}
}

training.outputa=as.matrix(read.table("U:/Dissertation/
Emulator/output50a.txt"))

beta.hat=solve(t(H)%solve(A)%H)%t(H)%solve(A)%training.outputa

tx=matrix(0,nrow=50,ncol=1)

for(i in 1:50)
{
tx[i,]=exp(-(t(x-training.input[i,]))%B%(x-training.input[i,]))
}

ty=matrix(0,nrow=50,ncol=1) for(i in 1:50)

```



```

{
ty[i,]=exp(-(t(y-training.input[i,]))**B**(y-training.input[i,]))
}

middle=solve(A)-((solve(A)**H)**solve(t(H)**solve(A)**H)**
(t(H)**solve(A)))

sigma2hat=(t(training.outputa)**middle**training.outputa)/42

cxy=exp(-(t(x-y)**B**(x-y)))

hx=matrix(c(1,x), nrow=1) hy=matrix(c(1,y), nrow=1)

first=cxy - (t(tx)**solve(A)**ty)

second=hx - (t(tx)**solve(A)**H)

third=solve(t(H)**solve(A)**H)

fourth=t(hy - (t(ty)**solve(A)**H))

sigma2hat**(first+(second**third**fourth))
}

validation.input=as.matrix(read.table("U:/Dissertation
/Emulator/validationa_simulator.txt")

covaR=matrix(0,nrow=20,ncol=20)

for(i in 1:20)

{
for(j in 1:20)

{
covaR[i,j]=v1a(validation.input[i,],validation.input[j,])
}
}
}

```

# Once again, to check that the code for this covariance matrix is correct, the diagonal elements of the matrix (the variances) should be equal to the vector of variances (corresponding to each prediction given as an expected value) produced by GEM-SA. All emulators showed this to be true.

```
var.a.GEMSA=as.matrix(read.table("U:/Dissertation/Emulator/Output_a/
3var_a_GEMSA.txt"))

var.a.compare=cbind(diag(covaR),var.a.GEMSA) var.a.compare
diag(covaR)-var.a.GEMSA
```

### 8.3.3 Deriving the values of $D_{MD}(\mathbf{y}^*)$

For emulator a again, we have:

```
ya=read.table("U:/Dissertation/Emulator/Output_a/3var_a_GEMSA.txt"))

D=t(ya-mean.a.R)%*%solve(covaR)%*%(ya-mean.a.R)
```

### 8.3.4 Deriving the distributional values for $D_{MD}(\cdot)$ (table 4)

Rearranging the expression in (5.3), we get:

$$D_{MD}(f(\mathbf{X}))|\mathbf{y}, b \sim \frac{m(n-q-2)}{n-q} \mathbf{F}_{m,n-q}$$

Substituting  $p = 5$ ,  $q = 6$ ,  $n = 50$ ,  $m = 20$ , this becomes:

$$D_{MD}(f(\mathbf{X}))|\mathbf{y}, b \sim \frac{840}{44} \mathbf{F}_{20,44}$$

An F-distribution with  $d_1$  and  $d_2$  degrees of freedom has mean and standard deviation given by:

$$\frac{d_2}{d_2 - 2} \quad \text{and} \quad \frac{2d_2^2(d_1 + d_2 - 2)}{d_1(d_2 - 2)^2(d_2 - 4)}.$$

Hence the expected value of  $D_{MD}(\cdot)$  is  $\frac{840}{44} \times \frac{44}{44-2} = 20$ .

The standard deviation is  $\sqrt{\left(\frac{840}{44}\right)^2 \times \left(\frac{2 \times 44^2 (20 + 44 - 2)}{20(44-2)^2(44-4)}\right)^2} = 3.248$ .

To obtain the 25th, 50th and 75th quantiles, the following code was used:

```
qf(c(0.25,0.5,0.75),20,44)*(840/44)
```

## 8.4 R Code for Diagnostic Check 3

### 8.4.1 Figures 5.4 and 5.5

The code below was used for figure 5.4. For figure 5.5, it is exactly the same except for  $h$  being changed to  $a$ .

```
inputs=read.table("U:/Dissertation/Emulator/Output_h  
/validation_inputs.txt",header=F)  
valh_simulator=read.table("U:/Dissertation/Emulator/Output_h  
/4yh.txt",header=F)  
valh_emulatormean=read.table("U:/Dissertation/Emulator/Output_h  
/2mean_h_GEMSA.txt",header=F)  
valh_emulatorvariance=read.table("U:/Dissertation/Emulator/Output_h  
/3var_h_GEMSA.txt", header=F)  
spe_h=(valh_simulators-valh_emulatormean)/sqrt(valh_emulatorvariance)  
  
matplot(valh_emulatormean,spe_h, type="n")  
matpoints(valh_emulatormean,spe_h, type="p", pch=16)  
abline(-2,0,lty=3)
```

```
abline(2,0, lty=3)
```

```
abline(0,0)
```

```
matplot(input[,1],spe_h, type="n")
```

```
matpoints(valh_emulatormean,spe_h, type="p", pch=16)
```

```
abline(-2,0,lty=3) abline(2,0, lty=3) abline(0,0)
```

```
# The last three lines are repeated four times, eaching time replacing 1 with 2, 3,  
4 and 5
```

## 8.4.2 Figures 5.6 and 5.7

As before, the following code relates to emulator h. By changing  $h$  to  $a$ , the corresponding plots for emulator a are produced. Note that  $covhR$  refers to the covariance matrix (see 8.2). Also note that on the numbers in third and fourth lines will be different for the two emulators. They refer to the pivoting order, which is obtained after the first and second lines have been executed.

```
R=chol(covhR,pivot=T) P=matrix(0,nrow=20,ncol=20)
```

```
R
```

```
P[2,1]=P[11,2]=P[10,3]=P[17,4]=P[8,5]=P[3,6]=P[15,7]=P[20,8]=
```

```
P[13,9]=P[5,10]=P[7,11]=P[14,12]=P[4,13]=P[16,14]=P[18,15]=
```

```
P[6,16]=P[9,17]=P[12,18]=P[19,19]=P[1,20]=1
```

```
G=P%*%t(R)
```

```
covhR-(G%*%t(G))
```

```
yh=as.matrix(read.table("U:/Dissertation/Emulator/Output_h
```

```

/4yh.txt",header=F))

Diag3errors.h=solve(G)%*%(yh-mean.h.R)

Diag3errors.h

plot(c(1:20),Diag3errors.h, xlab="Pivoting Order",
main="(a)",ylab="Pivoted Cholesky Errors", ylim=c(-9,9), pch=16)

abline(0,0,lty=1,col=1) abline(-2,0,lty=3,col=1)

abline(2,0,lty=3,col=1)

```

### 8.4.3 Plot of Pivoted Cholesky errors and Quantile-quantile plot for validation data without outliers

Figure 8.1:  $D_i^{PC}((y)^*)$  against Pivoting Order (for the 17 validation points which did not have S/N ratio between -20 and -10) for (a) emulator h ; (b) emulator a.

Figure 8.2: Quantile-quantile plot (for the 17 validation points which did not have S/N ratio between -20 and -10) for: (a) emulator h; (b) emulator a.

## 8.5 Poster Presentation

The 12 slides which made up the poster presentation for this dissertation are given on the pages that follow

# Chapter 9

## Bibliography

Bastos and O'hagan (2008). Diagnostics for Gaussian Emulators. In submission, currently available at [www.tonyohagan.co.uk/academic/pdf/diagtech.pdf](http://www.tonyohagan.co.uk/academic/pdf/diagtech.pdf).

Bastos, L. (2007). Diagnostics and Validation of Computer Models: First year PhD report.

Butler, N. (2001). Optimal and orthogonal Latin hypercube designs for computer experiments. *Biometrika*. textbf88, 847-857.

Cook & Upton (2004). *Oxford Dictionary of Statistics*. 2nd Edition. New York: Oxford University Press.

Conti et al. (2007). Gaussian process emulation of dynamic computer codes. In submission, currently available at [j-p-gosling.staff.shef.ac.uk/Pub/DyEm.pdf](http://j-p-gosling.staff.shef.ac.uk/Pub/DyEm.pdf).

Conti & O'Hagan (2007). Bayesian Emulation of Complex Multi-Output and Dynamic Computer Models. In submission, currently available at <http://www.tonyohagan.co.uk/academic/abs/multioutput.html>.

Gosling (2006). Differences between estimates of expected computer code output with GEM-SA. *MUCM Technical report 07/03. Department of Probability and Statistics, University of Sheffield.*

Husslage et al. (2006). Space-filling Latin hypercube designs for computer experiments. CentER Discussion Paper No. 2006-18. Available at SSRN: <http://ssrn.com/abstract=895464>.

Jin et al. (2005). An efficient algorithm for constructing optimal design of computer experiments. *J. Statist. Plann. Inference* **134**, 268-287.

Joseph, R. V., Hung, Y. (2008). Orthogonal-Maximin Latin Hypercube Designs. *Statistica Sinica*, **18**, 171-186.

Kennedy and O'hagan (2001). Bayesian calibration of computer models (with discussion). *Journal of the Royal Statistical Society, Series B.* **63**, 425-464.

Loeppky, J. (2008). Choosing the sample size of a computer experiment: a practical guide. *NISS*, **170**.



Morris and Mitchell (1995). Exploratory designs for Computational experiments. *Journal of Statistical Planning and Inference*, **43**, 381-402.

McKay, M.D.; Conover, W.J.; and Beckman, R.J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, Vol. 42, No.1, 55-61.

Oakley, J. (1999). Bayesian uncertainty analysis for complex computer codes. PhD Thesis, Department of Probability, University of Sheffield.

Oakley (2005). Decision Theoretic Sensitivity Analysis for Complex Computer Computer Models. *SAMO (2004)*, pp. 90-95.

Oakley & O'Hagan (2004). Probabilistic sensitivity analysis of complex models: a bayesian approach. *Journal of the Royal Statistical Society B*, **66**, 751-769.

O'Hagan (1978). Curve fitting and optimal design for predictions. *Journal of the Royal Statistical Society B*, **40**, 1-42.

O'Hagan (2004). Bayesian Analysis of Computer Code Outputs: A Tutorial. *Reliability Engineering and System Safety*, **91**, 1290-1300.

O'Hagan (2007). Uncertainty in Computer models - part 1. Available from

<[www.lsp.ups-tlse.fr/Fp/Gamboa/GDR/Day\\_1.ppt](http://www.lsp.ups-tlse.fr/Fp/Gamboa/GDR/Day_1.ppt)>

[Accessed 10th August 2008].

O'Hagan (2007). Uncertainty in Computer models - part 2. Available from

<[www.lsp.ups-tlse.fr/Fp/Gamboa/GDR/Day\\_2.ppt](http://www.lsp.ups-tlse.fr/Fp/Gamboa/GDR/Day_2.ppt)>

[Accessed 10th August 2008].

O'Hagan (2008). Uncertainty Analysis using GEM-SA. Available from

<<http://www.tonyohagan.co.uk/academic/GEM/UncertaintyAnalysis.ppt>>

[Accessed 10th August 2008].

Rasmussen (2006). Advances in Gaussian processes. Available from

<[learning.eng.cam.ac.uk/car1/talks/gpnt06.pdf](http://learning.eng.cam.ac.uk/car1/talks/gpnt06.pdf)>

[Accessed 10th August 2008].

Santner, T.J., Williams, B.J., Notz, W.I. (2003) . *The design and analysis of computer experiments*. New York: Springer.

Saltelli et al. (2008). *Global Sensitivity Analysis. The Primer*. Chichester: Wiley.

Van Dam et al. (2006). Maximin Latin hypercube designs in two dimensions. *Operations Research*. **55**, 158-169.

Wang, G. (2003). Adaptive Response Surface Method Using Inherited Latin Hypercube Design Points. *Journal of Mechanical Design*, **125**, 210-220.

Winters, R. (2008). MSDN Forums [website]. Available from:

`<http://forums.microsoft.com/msdn/ShowPost.aspx?PostID=3254326\&SiteID=1>`

[Accessed 27th July, 2008].

Ye, K., et al. (2000). Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of Statistical Planning and Inference*. **90**, 145-159.