

**Statistical Analysis of Computer
Models and The Use of Derivative
Information**

Literature Review

Gemma Stephenson

Department of Probability and Statistics

University of Sheffield

September 2007

Contents

1	Introduction	2
2	Gaussian Process Emulation	3
2.1	Introduction to Gaussian Process Emulation	3
2.2	Prior Knowledge	4
2.3	The Posterior Distribution	5
2.4	Design of Computer Experiments	7
3	Uncertainty in Complex Models	8
3.1	Sources of Uncertainty in Complex Models	8
3.2	Uncertainty Analysis	9
3.3	Calibration	10
3.4	Sensitivity Analysis	14
4	Use of Derivatives	16
4.1	Introduction to Using Derivatives in Complex Models	16
4.2	Methodology	16
4.3	Cost Effects	17
4.4	Applications of Derivative Information in Computer Models	20

1 Introduction

Complex models are used as representations of a real-world phenomena. They simulate real-world systems and are thus used in many different areas; engineering and oceanography are just two examples. One application of the models is to predict how the real world system may behave in the future. These models are written as computer codes and referred to as simulators. Running a computer model at a number of different input values is given the term *computer experiment*. We express the simulator by the function $y = \eta(\mathbf{x})$, where \mathbf{x} are the model inputs. Depending on the model, the output could be either a scalar or a vector; however, we will just consider the case with univariate output here. The simulators are deterministic, for each time they are run with the same inputs they will produce the same output.

The true values of the inputs may be unknown if they are properties of the real system which need to be observed. This is not always practical and when the values of the inputs can be obtained, it is likely that some form of measurement error will have occurred. We need to quantify the uncertainty in the inputs as this will relate to how uncertain we are that the simulator output matches reality. We denote the true value of the inputs by \mathbf{X} and true output is then $Y = \eta(\mathbf{X})$. The purpose of uncertainty analysis is to quantify the uncertainty in model outputs caused by uncertainty in inputs. Another source of uncertainty is in the model itself. Even when the true values of the inputs are known, running the model at these points will not produce an output which matches exactly the observation of the real-world system.

It may be of interest to learn how sensitive the model output is to its various inputs. The process of evaluating how the output of a model is modified by changes in the inputs is referred to as sensitivity analysis. Complex models tend to take an appreciable amount of computing time to run and in this sense they are expensive to execute. This is partly due to the high number of dimensions a simulation of a real-world system can require. Performing analyses such as sensitivity and uncertainty analysis can require many runs of the simulator and this quickly becomes impractical with a computationally expensive model.

The next section of this literature review describes Gaussian process emulation, including the methodology for building an emulator. Following this, the various sources of uncertainty arising through the use of computer models is discussed. The final section gives details of how derivative information can be used in complex models and the cost-issues associated with this.

2 Gaussian Process Emulation

The design and analysis of computer models has been a topic studied for around 20 years. In particular, Sacks *et al.* (1989) describe a technique for approximating an unknown deterministic computer experiment by modeling the output as a realization of a stochastic process. A Bayesian approach to predicting a computer model at untested inputs is given in Currin *et al.* (1991). An introduction to Gaussian process emulation, along with details of Bayesian methods for various analyses, is given in O'Hagan (2006), while Santner *et al.* (2003) give more detail on the Bayesian approach.

In this section we review emulation using Gaussian processes including the methodology to build a Gaussian process emulator.

2.1 Introduction to Gaussian Process Emulation

Often, it is useful for the model user to perform analyses such as sensitivity and uncertainty analysis on the model. However, with a complex model it is generally impractical to perform the number of runs required for this, due to the computational expense of the model. Hence, another model can be established which acts as a statistical approximator to the simulator. This model is termed an emulator and is built using only a small number of runs of the simulator. Any sensitivity and uncertainty analysis desired can then be performed with runs of the emulator, and greater efficiency is therefore achieved. To build an emulator we first consider the model, $\eta(\cdot)$, as an unknown function, as until the model is run the output values are unknown. We choose to represent the model as a Gaussian process. Santner *et al.* (2003) discuss the suitability of the Gaussian process

model for this. They explain how the prior model needs to be flexible so that it can adapt to the shape of $\eta(\cdot)$ and Gaussian processes exhibit this ability. In the Bayesian framework we begin by specifying our prior beliefs about the computer model and then run the code at the inputs $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ to obtain

$$\mathbf{y}^T = \{y_1 = \eta(\mathbf{x}_1), \dots, y_n = \eta(\mathbf{x}_n)\}, \quad (2.1)$$

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

2.2 Prior Knowledge

In a Bayesian approach to emulation, we first need to specify our prior beliefs. We require that the simulator, $\eta(\cdot)$, be a smooth function and we chose to model $\eta(\cdot)$ by a Gaussian process model. A Gaussian process is an infinite collection of variables, where every finite subset of the variables has a multivariate normal distribution. A Gaussian process is defined by its mean and covariance functions, so these must now be described.

We specify the prior mean as:

$$E[\eta(\mathbf{x}) | \boldsymbol{\beta}] = \mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}, \quad (2.2)$$

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

Next we define $\sigma^2 c(\mathbf{x}, \mathbf{x}')$ to be the covariance between $\eta(\mathbf{x})$ and $\eta(\mathbf{x}')$ and arrange the covariances in a matrix, A , which must be positive definite. As a result of the smoothness of $\eta(\cdot)$, when two points, \mathbf{x} and \mathbf{x}' are close there should be a high correlation between the corresponding $\eta(\mathbf{x})$ and $\eta(\mathbf{x}')$, and as the distance between \mathbf{x} and \mathbf{x}' increases, the correlation should decrease. The smoothness property of the model therefore implies that

the output will be similar for inputs close together. A common form of covariance function is

$$c(\mathbf{x}, \mathbf{x}') = \exp \left\{ -(\mathbf{x} - \mathbf{x}')^T B (\mathbf{x} - \mathbf{x}') \right\}, \quad (2.3)$$

where B is a diagonal matrix of positive roughness parameters, also known as correlation lengths,

$$B = \begin{pmatrix} b_1 & 0 & \dots & 0 \\ 0 & b_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & b_d \end{pmatrix},$$

where d is the number of input dimensions. The parameters, b_i , describe how ‘rough’ the model is in each input dimension. This is because the correlation between $\eta(\mathbf{x})$ and $\eta(\mathbf{x}')$ depends on the distance between \mathbf{x} and \mathbf{x}' and this distance is rescaled as a result of B . We can estimate B from the training data, as dealing with B using a full Bayesian approach is not practical. Methods for estimating B from the data are discussed in Section 2.3. We assume that our prior information about β and σ^2 will be weak, and so for the prior distribution use

$$p(\beta, \sigma^2) \propto \frac{1}{\sigma^2}. \quad (2.4)$$

So our prior beliefs takes the form:

$$\eta(\cdot) | \beta, \sigma^2, B \sim GP(h(\cdot)^T \beta, \sigma^2 c(\cdot, \cdot))$$

2.3 The Posterior Distribution

The next stage in building an emulator is to produce the training data by running $\eta(\cdot)$ at the selected inputs. Methods for choosing these inputs are discussed in Section 2.4.

The posterior distribution is now derived following the methods described in Haylock and O’Hagan (1996), where the following result is obtained

$$\left. \frac{\eta(\mathbf{x}) - m^{**}(\mathbf{x})}{\sqrt{\frac{n-q-2}{n-q} \hat{\sigma} \sqrt{c^{**}(\mathbf{x}, \mathbf{x})}}} \right| B \sim t_{n-q}, \quad (2.5)$$

where

$$\begin{aligned}
m^{**}(\mathbf{x}) &= \mathbf{h}(\mathbf{x})^T \hat{\boldsymbol{\beta}} + \mathbf{t}(\mathbf{x})^T A^{-1}(\mathbf{y} - H\hat{\boldsymbol{\beta}}), \\
c^{**}(\mathbf{x}, \mathbf{x}') &= c^*(\mathbf{x}, \mathbf{x}') + (\mathbf{h}(\mathbf{x})^T - \mathbf{t}(\mathbf{x})^T A^{-1}H)(H^T A^{-1}H)^{-1}(\mathbf{h}(\mathbf{x}')^T - \mathbf{t}(\mathbf{x}')^T A^{-1}H)^T, \\
H^T &= \{\mathbf{h}(\mathbf{x}_1), \dots, \mathbf{h}(\mathbf{x}_n)\}, \\
A &= \begin{pmatrix} 1 & c(\mathbf{x}_1, \mathbf{x}_2) & \dots & c(\mathbf{x}_1, \mathbf{x}_n) \\ c(\mathbf{x}_2, \mathbf{x}_1) & 1 & & \vdots \\ \vdots & & \ddots & \\ c(\mathbf{x}_n, \mathbf{x}_1) & \dots & & 1 \end{pmatrix}, \\
\hat{\boldsymbol{\beta}} &= (H^T A^{-1}H)^{-1}H^T A^{-1}\mathbf{y}, \\
\mathbf{t}(\mathbf{x})^T &= \{c(\mathbf{x}, \mathbf{x}_1), \dots, c(\mathbf{x}, \mathbf{x}_n)\}, \\
c^*(\mathbf{x}, \mathbf{x}') &= c(\mathbf{x}, \mathbf{x}') - \mathbf{t}(\mathbf{x})^T A^{-1}\mathbf{t}(\mathbf{x}'), \\
\hat{\sigma}^2 &= \frac{\mathbf{y}^T (A^{-1} - A^{-1}H(H^T A^{-1}H)^{-1}H^T A^{-1})\mathbf{y}}{n - q - 2}.
\end{aligned}$$

The posterior mean is $m^{**}(\mathbf{x})$ and can be used as a fast approximation of $\eta(\mathbf{x})$. The posterior covariance between $\eta(\mathbf{x})$ and $\eta(\mathbf{x}')$ is $\hat{\sigma}^2 c^{**}(\mathbf{x}, \mathbf{x}')$. The posterior mean is comprised of two parts. The first part corresponds to the prior mean (2.2), where $\hat{\boldsymbol{\beta}}$ is the expected value of $\boldsymbol{\beta}$, based on the training data. At the points $(\mathbf{x}_1, \dots, \mathbf{x}_n)$, the values of $\eta(\mathbf{x}_i), i \in \{1, \dots, n\}$ are known as they were evaluated directly to produce the training data. The second part of the emulator, $\mathbf{t}(\mathbf{x})^T A^{-1}(\mathbf{y} - H\hat{\boldsymbol{\beta}})$ has the effect of ensuring that $m^{**}(\mathbf{x}_i) = \eta(\mathbf{x}_i), i \in \{1, \dots, n\}$.

The matrix B which is comprised of the smoothness parameters must now be estimated. One method of estimating B is to ascertain which values of B maximise the likelihood, where the likelihood function is

$$f(B|\mathbf{y}) \propto |A|^{-\frac{1}{2}} |H^T A^{-1}H^{-1}|^{-\frac{1}{2}} (\hat{\sigma}^2)^{-\frac{(n-q)}{2}}.$$

An alternative approach to estimating B is to perform cross validation. We run the simulator at n points to obtain the training data, \mathbf{y} . We then omit one observation $y_i = \eta(\mathbf{x}_i)$ from the training data and build an emulator, for a given value of B , with the remaining $n - 1$ observations. This yields the posterior distribution of $\eta(\cdot)$ given the $n - 1$ observations and we then calculate the absolute distance between the posterior mean of

$\eta(\mathbf{x}_i)$ and the known value y_i . This procedure is then repeated for $i \in 1 \dots n$ and we establish the values of B which minimise the sum of the absolute distances between each known value y_i and posterior mean of $\eta(\mathbf{x}_i)$.

2.4 Design of Computer Experiments

We need to select the design points $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ at which the model, $\eta(\cdot)$, is to be run in order to obtain the training data. The choice of design points is referred to as the design of the experiment. The objective is to gain as much information as possible from the training data, as the more we learn about $\eta(\cdot)$ the better the emulator can approximate it. Designs can be generated based on various criteria. For example, a *maximin* design is one which selects points such that the maximum distance between any two design points is minimised.

McKay *et al.* (1979) compare three methods for choosing designs for Monte Carlo studies, where a deterministic computer code is modeling a real system. The three methods are simple random sampling, stratified sampling and Latin hypercube sampling. Suppose we have d input dimensions, with $\mathbf{x} = (x_1, \dots, x_d)$ and we need n design points at which to run the model, so we require

$$\begin{aligned} \mathbf{x}_1 &= (x_{11}, \dots, x_{d1}), \\ \mathbf{x}_2 &= (x_{21}, \dots, x_{d2}), \\ &\vdots \\ \mathbf{x}_n &= (x_{1n}, \dots, x_{dn}). \end{aligned}$$

A Latin hypercube sample ensures that the points are spread evenly over the range of each input dimension. This is achieved by first dividing the domain of $x_j, j \in \{1 \dots d\}$ into n intervals of equal marginal probability. Then one value is sampled from each interval resulting in n values in each dimension. To obtain \mathbf{x}_i , we sample from one of the n values without replacement in each dimension. The process is repeated until we have $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$. They assess the three methods by comparing estimators of the mean, variance and distribution function of the output. They conclude that Latin hypercube sampling is preferred and Santner *et al.* (2003) comment on the widespread use of this

method for generating designs in other kinds of computer experiment, observing that this form of design has been chosen extensively in the computer experiments literature.

3 Uncertainty in Complex Models

There are various uncertainties in computer experiments which need to be recognised and quantified. We begin this section by discussing some of the different sources of uncertainty and follow this with an overview of uncertainty analysis.

The topic of calibration is introduced in the next part, with particular focus on the Bayesian approach as given in Kennedy and O'Hagan (2001). We conclude this section with a discussion of sensitivity analysis.

3.1 Sources of Uncertainty in Complex Models

An overview of uncertainties in computer models is given by Kennedy and O'Hagan (2001) and a summary of this is given below.

They begin with *parameter uncertainty*. As introduced in Section 1, the true values of the inputs are unknown and we denote them by \mathbf{X} . There is then uncertainty in the values of the inputs of the code and we refer to this as parameter uncertainty.

Residual variability arises due to the real-world system not necessarily always producing the same values for the same set of inputs. The variation of the values produced by the real world system for a given, fixed set of inputs is known as *residual variability*.

As introduced in Section 1, the model is an approximation to the real-world system and so we need to consider how well the model represents reality. The discrepancy between the output of the model at the true input values and the true value of the real world system is known as *model inadequacy*. As noted above, the real-world system may not always produce the same output for the same set of inputs. In this case, discrepancy between model output and true mean value may be considered.

Observation error accounts for the uncertainty where physical observations are measured

imprecisely.

The final source of uncertainty considered is *code uncertainty*. This arises due to computer codes often being so complex, that it is not practical to run them at every desired set of inputs. As indicated in Section 2.1, we can think of the model as being unknown despite the fact that mathematically we may know exactly what the model is. In these cases, where it is not practical to run the model and observe the output at every desired set of inputs, we term the uncertainty in the outputs of the code by code uncertainty. Examples of these sources of uncertainty are given by Kennedy and O’Hagan (2001).

3.2 Uncertainty Analysis

As introduced in Section 1, the purpose of uncertainty analysis is to quantify the uncertainty in model outputs caused by uncertainty in inputs. This is done by treating the true inputs X as a random variable. Then, given a probability distribution, say G , for this random vector, uncertainty analysis is the study of the resulting probability distribution of the random output $Y = \eta(X)$. This output distribution is called the uncertainty distribution.

A traditional approach to uncertainty analysis is to use Monte Carlo methods. A brief overview of this technique is given in O’Hagan (2006). The method involves sampling values of X from G and running the simulator, $\eta(\cdot)$, at these points. To estimate the mean of Y we would take the mean of this sample of outputs. A much more efficient method of estimating the mean of Y would be to use an emulator. We would take a sample of X from G , as with Monte Carlo, but run an emulator at the points rather than the simulator. It is expected that far fewer runs of the simulator would be required to build an emulator, than would be required to obtain a sample of sufficient size to estimate the mean of Y by the Monte Carlo approach.

With the emulator approach, however, it is not necessary to take a sample to estimate the mean of Y as Haylock and O’Hagan (1996) derive the distribution of K and the mean and variance of L , where K and L are the mean and variance of the uncertainty distribution respectively. As in Section 2, the model is denoted by $\eta(\cdot)$ and they use a

Gaussian process to model prior beliefs. K is defined by the following integral

$$K = \int_{\chi} \eta(x) dG(x), \quad (3.1)$$

where χ is the input space. If $\eta(\cdot)$ were known and of a form which could be analytically integrated, then K could be evaluated directly by solving (3.1). However, we treat $\eta(\cdot)$ as an unknown function and so only have the posterior distribution for $\eta(\cdot)$. Consequently, Haylock and O'Hagan (1996) derive a posterior distribution for K which, conditional on σ^2 , is normal. L is given by

$$L = \int_{\chi} \eta^2(x) dG(x) - K^2.$$

Deriving the posterior distribution of L is difficult though, thus they continue by deriving formulae for the first two posterior moments, the mean and variance, of L .

Oakley and O'Hagan (2002) extend this to estimating the distribution function and the density function of Y . The distribution function is defined by

$$\begin{aligned} F(c) &= P(Y \leq c) \\ &= \int_{\chi} I[\eta(x) \leq c] dG(x), \end{aligned}$$

where I is the indicator function and c is some output value. The density function is defined by

$$\begin{aligned} f_Y(c) &= \lim_{h \rightarrow 0} \frac{1}{h} [F(c) - F(c - h)] \\ &= \int_{\chi} \lim_{h \rightarrow 0} \frac{1}{h} I[c - h \leq \eta(x) \leq c] dG(x). \end{aligned}$$

3.3 Calibration

Calibration is the process of estimating the values of the inputs, based on observed data. This usually involves searching for the set of input values which, after the model has been run, produces outputs which are as close as possible to the observations.

Kennedy and O'Hagan (2001) provide a review of existing methods for performing calibration as well as presenting their own Bayesian approach to the issue. They discuss in particular how uncertainty is measured in various approaches to calibration. For example,

a traditional method is to conduct an *ad hoc* search for the values of the inputs at which when the model is run, the outputs best fit the observed data. Having established these values, the inputs are set to their estimated values and then treated as if they were known. Treating the inputs as known, when they are in fact estimated indicates that parameter uncertainty is not accounted for in further runs of the model. Kennedy and O’Hagan (2001) suggest that none of the existing methods of calibration recognise fully, all sources of uncertainty. In their approach they begin by dividing the inputs into calibration inputs and variable inputs. The calibration inputs, $\boldsymbol{\theta}$, are unknown but fixed and observational data are required to estimate them. They assume the variable inputs, \mathbf{x} , to be known for the observations used for calibration, for example \mathbf{x} might be a physical location. Once the calibration inputs have been estimated, the variable inputs might be varied to predict the model output at other input points. The data will consist of two parts: \mathbf{y} and \mathbf{z} . To construct \mathbf{y} , the code, $\eta(\cdot)$, is run at n sites and both the variable inputs and the calibration inputs are known at these points. Observations of the true process are taken at N points, where the variable inputs are known, and these data form \mathbf{z} .

They relate the observations, z_i , the true process and the code through the following model:

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

Here, e_i are the observation errors, taken to be independent and normally distributed with mean 0, variance λ , ρ denotes an unknown regression parameter and $\eta(\mathbf{x}_i, \boldsymbol{\theta})$ is the computer code output. Finally, $\delta(\mathbf{x}_i)$ is a model inadequacy function and this is independent of the code output. It represents the discrepancy between the code output and the true value of the real process. Hence through this function, uncertainty about how well $\eta(\cdot)$ models the true process can be described. Prior beliefs must now be specified. They use Gaussian processes to represent their prior knowledge about $\eta(\cdot)$ and $\delta(\cdot)$, where the mean and variance functions for both are modelled hierarchically through the specification of hyperparameters.

The next step is to derive the posterior distribution of the parameters. Here, Kennedy and O’Hagan (2001) estimate the hyperparameters and so due to this, their analysis is not fully Bayesian. They propose to do this by first using only \mathbf{y} to estimate the

hyperparameters present in the covariance function for the prior Gaussian process, used to model η . Next, having fixed these values, the remaining hyperparameters are estimated using \mathbf{z} . They choose this method as it is assumed that the amount of observational data will be much less than the number of code outputs. This is because no matter how computationally expensive the code is to run, it is likely that it will be more expensive to obtain observations of the true process. In this way, they assume that N will be much smaller than n and so believe the simplification of using only \mathbf{y} in estimating the first hyperparameters to have little effect. The main aim of Kennedy and O’Hagan (2001) was to provide a method of calibration, which takes into account all the sources of uncertainty described in Section 3.2. They recognise that due to the simplification described above, not all sources are *fully* accounted for. They justify this, however, by describing the uncertainty forsaken as being only a ‘second-order effect’ of uncertainty about the hyperparameters and they believe that despite this, they have still accounted for the majority of all sources of uncertainty. They claim also, that accounting fully for all hyperparameter uncertainty would be computationally expensive, and so covering all uncertainties to a slightly lesser extent is an acceptable compromise.

Having estimated the hyperparameters, and conditional on them as well as the calibration parameters, the posterior distribution of the true process is found to be a Gaussian process. Combining this with the posterior distribution of the calibration parameters enables the user to predict the output of the true process as some specified input values for the variable inputs.

Campbell (2006) comments on the highly parameterized nature of the Kennedy and O’Hagan (2001) approach to calibration. By modeling $\delta(\cdot)$ as a Gaussian process, parameters which define its mean and covariance function need to be estimated. Along with these, $\boldsymbol{\theta}$ and also ρ and λ need to be estimated. However, the amount of data to base these estimations on may not be sufficient, partly due to the expense of obtaining observations of the true process. Campbell also suggests that λ should be estimated by other information and not by the method of calibration.

An alternative Bayesian approach to calibration is given by Craig *et al.* (1997). Rather than implementing a full Bayesian analysis, a Bayes linear approach is adopted which does

not require full specification of a prior probability distribution. Instead, only prior means, variances and covariances between the quantities of interest are necessary. Once these have been specified, they are linearly updated as data is observed. This method may be preferred for particularly high dimensional complex models, as specifying full prior beliefs for these is often difficult. Also this method is less computationally expensive, which is a key advantage over the full Bayesian approach when analysing high dimensional models. The Bayes linear approach to calibration begins with the specification of the prior mean and variance for the output of the model $y = \eta(\mathbf{x})$ and the covariance between $y = \eta(\mathbf{x})$ and $y = \eta(\mathbf{x}')$. As previously, we denote the input variables by $\mathbf{x} = (x_1, \dots, x_d)$ where d is the number of input dimensions. The physical data which have been observed, \mathbf{y}_H , will have been done so with a measurement error, \mathbf{y}_E . Hence, \mathbf{y}_H and the true values of the physical data, \mathbf{y}_T , are related through the following expression

$$\mathbf{y}_H = \mathbf{y}_T + \mathbf{y}_E.$$

Similarly, the output of the simulator does not represent the true value of the real system. Hence we have a further relation

$$\mathbf{y}_T = \eta(\mathbf{x}) + \mathbf{y}_D.$$

The difference between the true values and the simulator output, therefore, is represented by \mathbf{y}_D . This is equivalent to the model inadequacy function, $\delta(\cdot)$, in the approach by Kennedy and O'Hagan (2001). Craig *et al.* (1997) propose a method to elicit prior beliefs. They simplify the simulator to construct a simpler, faster version and run this many times. Information gained from these runs is then combined with knowledge from an expert in the physical system. If input dimensionality is high, often a form of sensitivity analysis will be performed to identify which input variables account for the majority of the variation in the output. These variables are then labeled *active variables* and the remaining input variables are fixed.

In this approach, Craig *et al.* (1997) introduce the concept of *implausability*. We want to eliminate regions of the input space which are unlikely, by some measure, to give corresponding simulator output close to \mathbf{y}_T . This implausibility measure is based on $E(\eta(\mathbf{x}) - \mathbf{y}_T)$ and is updated as we obtain data from full simulator runs. The inputs for

these runs are chosen sequentially. After each run, the output $y = \eta(\mathbf{x})$ is incorporated in the data which reduces uncertainty about the outputs for the remaining input sets. Suppose \mathbf{p} is a vector comprising of the values of $y = \eta(\mathbf{x})$ for previous inputs and y is the output corresponding to the next input choice. Then the adjusted expectation and variance of y given \mathbf{p} are given by the following two expressions,

$$\begin{aligned} E_{\mathbf{p}}(y) &= E(y) + \text{cov}(y, \mathbf{p}) \text{var}(\mathbf{p})^{-1} [\mathbf{p} - E(\mathbf{p})], \\ \text{var}_{\mathbf{p}}(y) &= \text{var}(y) - \text{cov}(y, \mathbf{p}) \text{var}(\mathbf{p})^{-1} \text{cov}(\mathbf{p}, y). \end{aligned}$$

The method for selecting the values of the inputs is as follows. The set of input values which have a high plausibility is first determined. Then from this set, the inputs which maximise the reduction in uncertainty about the output are chosen. The simulator is then run and the process of selecting inputs for the next run is repeated. In this way inputs are chosen sequentially and the region of the input space which produce plausible simulator output is reduced. When this region has sufficiently reduced the corresponding simulator output should adequately match the observed data.

3.4 Sensitivity Analysis

As introduced in Section 1, sensitivity analysis is the process of evaluating how the output of a model is modified by changes in the inputs. Sensitivity analysis can be either local or global. Local sensitivity analysis is the process of calculating partial derivatives with respect to one input variable to investigate how the output is affected by small changes in that input. This is done by varying one input whilst keeping all others constant. Saltelli *et al.* (2000) review some of the methods for calculating derivatives and obtaining local sensitivities from them. They discuss, for example, using finite-difference approximation, where one parameter is varied and the model is rerun at each stage. Another approach is the direct method which uses numerical solutions to ordinary differential equations. If the model has an adjoint then this can be run and will provide the partial derivatives. Adjoint methods are discussed further in Section 4.3.

Global sensitivity analysis quantifies the effect on the output when all inputs are more substantially varied, usually over a defined range. One form of this is probabilistic

sensitivity analysis. As with uncertainty analysis the true values of the inputs are unknown and are denoted by the random vector, X , so the true output vector of the model, $Y = \eta(X)$, is therefore also unknown. We denote the probability distribution of the uncertainty in the input vector X by G . Probabilistic sensitivity analysis is establishing how the inputs in X contribute to the uncertainty in Y . Particularly in the case of complex models, it may not be clear how sensitive the output is to each of the input variables before applying any sensitivity analysis. One approach to performing probabilistic sensitivity analysis is to use variance-based methods. A review of some of the different global sensitivity analyses using variance-based methods is given by Saltelli *et al.* (2000). These techniques assess the importance of an input, X_i , by calculating the expected reduction in the variance of Y given X_i , as stated in the following identity:

$$\text{Var}(Y) - \text{E}_{X_i}[\text{Var}(Y|X_i)] = \text{Var}_{X_i}[\text{E}(Y|X_i)].$$

Oakley and O'Hagan (2004) describe a Bayesian framework for probabilistic sensitivity analysis implementing the variance-based method described above. An example is given comparing the traditional Monte Carlo approach with that of one using Gaussian process emulation. It is found that building an emulator and performing sensitivity analysis with this emulator proves to be much more efficient.

Saltelli *et al.* (2000) provide a number of different goals for sensitivity analysis. One such goal may be to reduce dimensionality. If we find that the output of a model is sensitive to only some of the inputs, for example by establishing which inputs contribute most to the variance in the output as detailed above, the inputs having a negligible effect can be fixed and thus the dimension of the model is reduced. Similarly, sensitivity analysis may reveal that calibration of some input parameters will yield only marginal benefits. Given that obtaining observations of the real world process which the model is simulating, required for calibration, is likely to be relatively expensive then sensitivity analysis provides a means to reducing this expense.

4 Use of Derivatives

In this section the use of derivative information in complex models will be introduced and the value of observing derivatives discussed. Following this, the methodology required to implement this information will be given and then computational cost effects of producing and/or using the derivatives is discussed. The final section provides a brief summary of the previous work and conclusions in this field.

4.1 Introduction to Using Derivatives in Complex Models

It has been recognised for some time that derivative information has the potential to improve on emulation of complex models and in some cases to reduce computational expense. The benefit of observing derivatives in modelling nonlinear, dynamic systems is discussed in Leith *et al.* (2002), Solak *et al.* (2003) and Azman and Kocijan (2005). Morris *et al.* (1993) extend the work of Currin *et al.* (1991) to considering how derivatives can be used in Gaussian process emulation and the benefit of observing derivatives in compartmental models is discussed in Killeya and Goldstein (2007).

4.2 Methodology

Let $\eta(\cdot)$ be an unknown function as before in Section 2. O'Hagan (1992) shows how the theory can be extended to using Gaussian processes to model derivatives of $\eta(\cdot)$, assuming $\eta(\cdot)$ is differentiable everywhere. Differentiation is a linear operation, thus if $\eta(\cdot)$ is described by a Gaussian process then the derivatives of $\eta(\cdot)$ are also described by a Gaussian process. The prior mean of (2.2) and covariance of (2.3), assuming both are differentiable, become respectively

$$\begin{aligned} E \left[\frac{\partial^n}{\partial x_i^n} \eta(\mathbf{x}) \middle| \boldsymbol{\beta} \right] &= \frac{\partial^n}{\partial x_i^n} \mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}, \\ \text{Cov} \left[\frac{\partial^n}{\partial x_i^n} \eta(\mathbf{x}), \frac{\partial^m}{\partial x_j'^m} \eta(\mathbf{x}') \middle| \boldsymbol{\beta}, \sigma^2, B \right] &= \sigma^2 \frac{\partial^{n+m}}{\partial x_i^n \partial x_j'^m} c(\mathbf{x}, \mathbf{x}'). \end{aligned}$$

Then the posterior distribution of the derivative of $\eta(\mathbf{x})$ corresponding to (2.5) becomes

$$\frac{\frac{\partial^r}{\partial x_i^r} \eta(\mathbf{x}) - m_{r,x_i}^{**}(\mathbf{x})}{\sqrt{\frac{n-q-2}{n-q} \hat{\sigma}^2 c_{r,x_i}^{**}(\mathbf{x}, \mathbf{x})}} \sim t_{n-q},$$

where

$$\begin{aligned} m_{r,x_i}^{**}(\mathbf{x}) &= \frac{\partial^r}{\partial x_i^r} \mathbf{h}(\mathbf{x})^T \hat{\boldsymbol{\beta}} + \frac{\partial^r}{\partial x_i^r} \mathbf{t}(\mathbf{x})^T A^{-1} (\mathbf{y} - H \hat{\boldsymbol{\beta}}), \\ c_{r,x_i}^{**}(\mathbf{x}, \mathbf{x}') &= 1 - \left[\frac{\partial^r}{\partial x_i^r} \mathbf{t}(\mathbf{x})^T \right] A^{-1} \left[\frac{\partial^r}{\partial x_i^r} \mathbf{t}(\mathbf{x}') \right] \\ &\quad + \left\{ \left[\frac{\partial^r}{\partial x_i^r} \mathbf{h}(\mathbf{x})^T \right] - \left[\frac{\partial^r}{\partial x_i^r} \mathbf{t}(\mathbf{x})^T \right] A^{-1} H \right\} (H^T A^{-1} H)^{-1} \\ &\quad \times \left\{ \left[\frac{\partial^r}{\partial x_i^r} \mathbf{h}(\mathbf{x}')^T \right] - \left[\frac{\partial^r}{\partial x_i^r} \mathbf{t}(\mathbf{x}')^T \right] A^{-1} H \right\}^T. \end{aligned}$$

4.3 Cost Effects

It is necessary to consider the computational cost of using derivative information in computer experiments as it must be decided at which point the costs outweigh the benefits. For example, if generating the derivatives of the model increases the computational cost substantially, this extra computing time may be better spent evaluating the model at more points instead. Morris *et al.* (1993) suggest some general guidelines on this matter while recognising that ‘these issues are heavily application dependent’. They acknowledge that the speed of the computer, as well as the efficiency of the optimization algorithm used for estimating the smoothness parameters, has a notable effect on the overall cost. In reference to this, they suggest the ‘line search’ method presented by Welch *et al.* (1992) could yield a more cost effective process. The ‘line search’ approach employs the method of maximum likelihood estimation and involves adding correlation parameters sequentially as they are required. A brief overview of the algorithm is now given.

1. Scale the input variables such that all have the same range.
2. Set the roughness parameters b_i , $i \in \{1, \dots, d\}$ where d is the number of input dimensions, to all take the same value, b . So $b_i = b$. Maximise the log likelihood function subject to these conditions and term the value of the maximised log likelihood, l_0 .

3. For each $j \in \{1, \dots, d\}$ allow b_j to take any positive value but maintain the constraint that $b_i = b$ with $i \neq j$ and maximise the log likelihood in each case to give l_j .
4. Find the b_j which maximises $l_j - l_0$, denote it as b_{j^*} and fix this roughness parameters at its corresponding value.
5. Repeat step 3 but with b_{j^*} fixed so we have $i, j \in \{1, \dots, d\} \setminus \{j^*\}$. Continue to assign b_j values in this way until $j = d$ or until the difference, $l_j - l_0$, is not sufficiently large enough.

In the situation where the derivatives are already available, it is a matter of considering just the additional computational cost of including this information in the computer experiment. The main source of additional cost comes from the inversion of the covariance matrix, A , as this matrix is now larger in dimension. A must include the covariances between all partial derivatives and also the covariances between the partial derivatives and the function output. This means A is now a square matrix of size $n(d+1) \times n(d+1)$, where d is the number of input dimensions. Comparing A to the prior covariance matrix we when no derivative information is included, we see that A has increased in dimension by a factor of $d+1$. Estimating the smoothing parameters from the training data requires A to be inverted at each step. Therefore, approximately, the cost of including derivative information is increased by a factor of $(d+1)^3$ (Morris *et al.*, 1993).

In the case where derivatives are not already available the cost of producing them must be taken into consideration, as well as the cost of applying the information. Here we will review some of the methods for producing derivatives. One such method is the divided difference approach, which generates derivatives directly from the definition:

$$\frac{\partial}{\partial x} \eta(x) = \lim_{h \rightarrow 0} \frac{\eta(x+h) - \eta(x)}{h},$$

for some function η . It is claimed by Morris *et al.* (1993) that this method is unacceptable. Computing just one partial derivative by this method required two runs of the simulator: $\eta(x)$ and $\eta(x+h)$, where x and $(x+h)$ are very close together in the input space. However, for the cost of two runs we would expect to learn much more about $\eta(\cdot)$, if it were run at

two sites further apart in the input space. Hence, this approach to calculating derivatives can not be employed. Another disadvantage to this method, as noted by Roh *et al.* (1999), is that accuracy of the derivatives may be lost due to cancellation and truncation errors.

Another technique for generating derivatives is automatic differentiation. A brief overview of this method is given by Roh *et al.* (1999). From the code which calculates a function, a second code is generated and this code then calculates the derivatives of the function with respect to the input variables. It does this by repeatedly applying the chain rule to the combination of elementary operations in the function. The chain rule is described below. Let g be a function of x and let f be a function of g such that, $y = f(g(x))$ and $u = g(x)$. Then the chain rule states that

$$\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}, \quad (4.1)$$

assuming the functions are differentiable. Roh *et al.* (1999) show that automatic differentiation, in their example, requires half the time that the method of divided differences needs to produce the derivatives. Morris *et al.* (1993) state that the computational cost of generating both the function output and all its derivatives is a constant multiple of the time, M , required to run the model without producing derivatives. It is important that that value of M is taken into consideration when assessing how beneficial derivative information may be in a particular model. Suppose, for example $M = 10$. To run a model and evaluate both the output and the derivatives at 10 sites is approximately equivalent to running the model without producing derivatives at 100 sites. In most cases it would be expected that the emulator built with the latter data will be a better approximation to the model than that built with the initial choice of data. At the time of publishing, Morris *et al.* (1993) report that this multiple ranges between 10 and 20 but suggest it may decrease as research continues. Automatic differentiation relies on the assumption that the model is a collection of elementary operations and functions. To employ this approach, therefore, the model must be analytically differentiable.

A third possibility for generating derivatives is using the adjoint method. Adjoint models were first built mainly for the purpose of sensitivity analysis in reactor physics (Hall and Caucui, 1982) but they provide an efficient means of obtaining derivatives that can be used in other analyses. Automatic differentiation, described above, calculates the right

hand side of (4.1) in order to obtain the left hand side and this is called *forward mode*. An adjoint model employs a *reverse* mode and calculates the left hand side of (4.1) to obtain the right hand side. Detail of how an adjoint model is constructed is given by Giering and Kaminski (1998). Hall and Caucui (1982) describe the computational cost of running an adjoint model to obtain the sensitivities of a radiative-convective model, where the model itself took 8 seconds to run. There were 312 model parameters and to calculate the necessary adjoint equations a further 5 seconds were required. Then once the adjoints were available, it took an additional 7 seconds to produce all 312 derivatives. Hence the authors reported that it took an extra 12 seconds to generate all the derivatives. An adjoint model provides, therefore, an efficient means of generating derivatives. However the time taken to develop the adjoint for a model is far from trivial. Marotzke *et al.* (1999) warn that coding an adjoint to the existing code for the model is an arduous task and is prone to errors. Morris *et al.* (1993) support this by suggesting it is very time consuming. However, Errico (1997) stresses how powerful adjoint models could be and implies that an increase in the availability of adjoint models will prove very useful in many areas of science.

Killeya and Goldstein (2007) look at the use of derivatives in compartmental models. They choose not to apply any of the techniques discussed above, instead differentiation by hand is the favoured method, as it is particularly efficient. Differentiation by hand is possible due to the size of the model they use and the compartmental structure of the model. They note though, that this technique is only possible when the underlying form of the function is known. They report the computational cost of including derivative information, in their example, increases the overall cost of running the model by a factor of 1.8.

4.4 Applications of Derivative Information in Computer Models

Morris *et al.* (1993) implement the methodology of Section 4.2 to build an emulator based on the function output and all its partial derivatives. They illustrate the method with an example and assume the derivatives are already known. Selecting a design when derivative information will be generated at the points as well as the function output is discussed.

They compare 2 designs initially, a Latin hypercube sample and a maximin design and find that each have strengths and weaknesses. Thus they continue by constructing a further 2 designs which both attempt to combine the advantages of the initial designs. The first of these ‘compromise designs’ is a *maximin Latin hypercube design* and the second is a *modified maximin design*. They suggest the compromise designs are generally superior over the original design choices.

Killeya and Goldstein (2007) utilise derivative information in the analysis of compartmental models. A Bayes Linear approach is adopted rather than implementing a fully Bayesian analysis. Details of the Bayes Linear approach to emulating complex models is given by Craig *et al.* (1997), an overview of which can be found in Section 3.3. The methodology of this approach is demonstrated by a compartmental model of plankton cycles. The first-order input derivatives are calculated by hand, as noted above in Section 4.3, and are employed in a sensitivity analysis. The sensitivity analysis is performed by plotting the derivatives for the n points and considering the sample mean and variance. Input variables whose derivatives have a mean around zero and a small variance may be regarded as inactive. Whereas if an input variable has derivatives with a high variance it may be termed an *active variable*. Thus in this way a subset of the input variables are selected and this enables the input dimensionality to be reduced, lightening the computational burden of running the model. Having determined which of the inputs are active variables they continue to investigate how an emulator built with just the function output compares to one built with the additional information provided by derivatives. The comparison is performed by running both emulators at a set of new input points and examining how the mean variance across this input set differs. The model produces two outputs. The mean variance produced from 25 runs of the emulator without derivative information, is comparable to the mean variance produced from 6 runs of the emulator with derivatives for the first output and 8 runs for the second output. As discussed in Section 4.3, the expected increase in cost of generating the derivatives is by a factor of 1.8. Therefore, we can consider 25 runs of the emulator without derivatives to be similar in computational cost to approximately 14 runs of the emulator with derivatives. This shows that to achieve comparable variances, in this example, it is computationally cheaper to include derivative information in the building of an emulator. Killeya and Goldstein (2007)

also evaluate the additional reduction in mean variance caused by including derivatives, for a given set of runs. In their example for the first output variable, the maximum additional reduction in variance is 44.5% and for the second, 31.2%. Thus it is shown that uncertainty about the model is reduced substantially, if derivative information is included when building an emulator. They do not compare, however, the prediction error in the emulators.

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

References

- Azman, K. and Kocijan, J. (2005). Comprising prior knowledge in dynamic gaussian process models. *CompSysTech Proceedings*.
- Campbell, K. (2006). Statistical calibration of computer simulators. *Reliability Engineering and System Safety*, **91**, 1358–1363.

- Craig, P. S., Goldstein, M., Seheult, A. H. and Smith, J. A. (1997). Pressure matching for hydrocarbon reservoirs: A case study in the use of bayes linear strategies for large computer experiments. *Case Studies in Bayesian Statistics 3*, 36–93.
- Currin, C., Mitchell, T., Morris, M. and Ylvisaker, D. (1991). Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, **86**, 953–963.
- Errico, R. M. (1997). What is an adjoint? *Bulletin of the American Meteorological Society*, **78**, 2577–2591.
- Giering, R. and Kaminski, T. (1998). Recipes for adjoint code construction. *Transactions on Mathematical Software*, **24**, 437–474.
- Hall, M. C. H. and Caucui, D. G. (1982). Sensitivity analysis of a radiative-convective model by the adjoint method. *Journal of the Atmospheric Sciences*, **39**, 2038–2050.
- Haylock, R. G. and O’Hagan, A. (1996). On inference for outputs of computationally expensive algorithms with uncertainty on the inputs. In J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds.), *Bayesian Statistics 5*. Oxford: University Press, 629–637.
- Kennedy, M. C. and O’Hagan, A. (2001). Bayesian calibration of computer models. *Journal of Royal Statistical Society. Series B (Statistical Methodology)*, **63**, 425–464.
- Killeya, M. R. H. and Goldstein, M. (2007). Exploiting compartmental structure through derivatives in bayesian emulation of computer simulators with application to the plankton cycle. *Under revision for Journal of Statistical Planning and Inference*.
- Leith, D. J., Leithead, W. E., Solak, E. and Murray-Smith, R. (2002). Divide and conquer identification using gaussian process priors. *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, **1**, 624–629.
- Marotzke, J., Giering, R., Zhang, K. Q., Stammer, D., Hill, C. and Lee, T. (1999). Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport sensitivity. *Journal of Geophysical Research*, **104**, 29529–29547.

- McKay, M. D., Beckman, R. J. and Conover, W. J. (1979). A comparison of three methods for selecting values of input in the analysis of output from a computer code. *Technometrics*, **21**, 239–245.
- Mitchell, T. and Morris, M. (1994). Asymptotically optimum experimental designs for prediction of deterministic functions given derivative information. *Journal of Statistical Planning and Inference*, **41**, 377–389.
- Morris, M. D., Mitchell, T. J. and Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, **35**, 243–255.
- Oakley, J. and O’Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika*, **89**, 769–784.
- Oakley, J. and O’Hagan, A. (2004). Probabilistic sensitivity analysis of complex models: a bayesian approach. *Journal of the Royal Statistical Society B*, **66**, 751–769.
- O’Hagan, A. (1992). Some bayesian numerical analysis. In J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds.), *Bayesian Statistics 4*. Oxford: University Press, 345–363.
- O’Hagan, A. (2006). Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering and System Safety*, **91**, 1290–1300.
- Roh, L., Bischof, C., Chang, N., Lee, K., Kanevsky, V., Nakagawa, O. and Oh, S.-Y. (1999). Fast statistical-based interconnect modeling using automatic differentiation. *Simulation of Semiconductor Processes and Devices, 1999. SISPAS ’99. 1999 Conference on*, 159–162.
- Sacks, J., Welch, W. J., Mitchell, T. J. and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, **4**, 409–423.
- Saltelli, A., K.Chan and Scott, E. M. (2000). *Sensitivity Analysis*. John Wiley and Sons, Chichester.

- Santner, T. J., Williams, B. J. and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. New York: Springer-Verlag.
- Solak, E., Murray-Smith, R., Leithead, W. E., Leith, D. J. and Rasmussen, C. E. (2003). Derivative observations in gaussian process models of dynamic systems. *Advances in Neural Information Processing Systems 15*.
- Trucano, T. G., Swiler, L. P., Igusa, T., Oberkampf, W. L. and Pilch, M. (2006). Calibration, validation, sensitivity analysis: What's what. *Reliability Engineering and System Safety*, **91**, 1331–1357.
- Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J. and Morris, M. D. (1992). Screening, predicting, and computer experiments. *Technometrics*, **34**, 15–25.