Application of Derivative Information in Toy Models

Gemma Stephenson

September 2008

Abstract

It is possible to obtain derivatives of model outputs, for example through the adjoint of the model. The value of learning derivatives when building Gaussian process emulators is investigated to determine whether additional efficiency can be achieved. Furthermore, emulators can be employed to predict derivatives and the performance of these predictions is also examined.

Contents

1 Introduction

1.1 Purpose and scope of this document

Emulators are typically built on the information gained through a small number of runs of the simulator. One aim of this study is to investigate what value derivative information could be to that process. The performance of emulators built with derivative information in addition to standard simulator output will be compared to that of emulators which rely solely on the function output. Clearly, the computational cost of attaining the derivatives must be considered in this investigation. The cost varies depending on the method employed to calculate the derivatives. One of the most efficient procedures for generating derivatives is running the model's adjoint; this is assuming the adjoint of the model exists. We can consider the investigation in the following way: we have a limited amount of computing time to build an emulator over a specified input region. We can either spend that time on $n + m$ runs of the simulator, in order to obtain function output, or on n runs of the adjoint of the simulator to obtain both function output and partial derivatives. The question is, which approach results in an emulator which is more representative of the simulator over the specified input space. We compare the emulators through their predictive performance and measure of uncertainty.

Running a model's adjoint to obtain derivatives, while more efficient and accurate than other methods, is a computationally expensive task. In addition, the effort taken initially to write the adjoint is considerable and the task, time consuming. Therefore, if we can emulate the derivatives of a simulator this would decrease the demand for writing and running adjoints. The second aim of this study is to perform an initial investigation into the suitability of an emulator to predict derivatives.

We perform the above investigations on *toy* models. These models are therefore, not computationally expensive. They are appropriate though to demonstrate the techniques on and provide preliminary results, on the basis of which a real complex model can then be chosen to provide a more appropriate study.

1.2 Structure of this document

The following section describes the methodology for the inclusion of derivatives when emulating function output. This is demonstrated in an example and then more thoroughly in Section 3 where the investigation into toy models is detailed. The subject of Section 4 is emulation of derivatives. The methodology is introduced and illustrated with examples. An investigation similar to that of Section 3 is not included.

2 Use of derivatives in emulating function output

In this section the use of derivative information in complex models will be introduced and the value of observing derivatives discussed. The methodology required to implement this information in a Gaussian process emulator is given and we adopt the notation employed by Oakley and O'Hagan (2002).

2.1 Introduction

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

2.2 Methodology

Here we build an emulator with derivative information in addition to function output. We compile the training data by running the simulator at the inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ but also we evaluate the partial derivatives at each of these input sites. The training data, $\mathbf{y}_D$, in

the case where we have two input dimensions, $d = 2$, is complied as follows:

$$\mathbf{y}_D = \begin{pmatrix} \eta(x_{1_1}, x_{2_1}) \\ \vdots \\ \eta(x_{1_n}, x_{2_n}) \\ \frac{\partial}{\partial x_1}\eta(x_{1_1}, x_{2_1}) \\ \vdots \\ \frac{\partial}{\partial x_1}\eta(x_{1_n}, x_{2_n}) \\ \frac{\partial}{\partial x_2}\eta(x_{1_1}, x_{2_1}) \\ \vdots \\ \frac{\partial}{\partial x_2}\eta(x_{1_n}, x_{2_n}) \end{pmatrix}.$$

O'Hagan (1992) shows how the theory can be extended to using Gaussian processes to model derivatives of $\eta(.)$, assuming $\eta(.)$ is differentiable everywhere. The prior mean and covariance function, assuming both are differentiable, become respectively

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

Continuing an example of building an emulator for a two-input dimensional simulator and with a linear form for $\mathbf{h}(\mathbf{x})$ this gives:

$$E\left[\frac{\partial}{\partial x_1}\eta(x_1, x_2)|\,\boldsymbol{\beta}\right] = \frac{\partial}{\partial x_1}\mathbf{h}(x_1, x_2)^T\boldsymbol{\beta} = (0\ 1\ 0)\,\boldsymbol{\beta},$$

and

$$E\left[\frac{\partial}{\partial x_2}\eta(x_1, x_2)|\,\boldsymbol{\beta}\right] = \frac{\partial}{\partial x_2}\mathbf{h}(x_1, x_2)^T\boldsymbol{\beta} = (0\ 0\ 1)\,\boldsymbol{\beta}.$$

Thus,

$$H_D = \begin{pmatrix} 1 & x_{1_1} & x_{2_1} \\ \vdots & \vdots & \vdots \\ 1 & x_{1_n} & x_{2_n} \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}.$$

The prior covariance matrix is now

$$A_D = \begin{pmatrix} \text{var}[\eta(.), \eta(.)] & \text{cov}[\eta(.), \eta^1(.)] & \text{cov}[\eta(.), \eta^2(.)] \\ \text{cov}[\eta^1(.), \eta(.)] & \text{var}[\eta^1(.), \eta^1(.)] & \text{cov}[\eta^1(.), \eta^2(.)] \\ \text{cov}[\eta^2(.), \eta(.)] & \text{cov}[\eta^2(.), \eta^1(.)] & \text{var}[\eta^2(.), \eta^2(.)] \end{pmatrix}, \tag{2.1}$$

where $\text{cov}[\eta(.), \eta^1(.)]$ is the matrix of covariances between $\eta(\mathbf{x})$ and $\frac{\partial}{\partial \mathbf{x_1}}\eta(\mathbf{x})$ and $\text{cov}[\eta(.), \eta^2(.)]$ is the matrix of covariances between $\eta(\mathbf{x})$ and $\frac{\partial}{\partial \mathbf{x_2}}\eta(\mathbf{x})$. The covariances between derivatives are given by

$$Cov\left[\frac{\partial}{\partial x_i}\eta(\mathbf{x}), \frac{\partial}{\partial x_j}\eta(\mathbf{x}')\middle| \boldsymbol{\beta}, \sigma^2, B\right] = \sigma^2 \frac{\partial^2}{\partial x_i \partial x_j} c(\mathbf{x}, \mathbf{x}'),$$

assuming the function is differentiable. For example,

$$\text{cov}[\eta^1(.), \eta(.)] = 2b_1(\mathbf{x}_1 - \mathbf{x}_1') \exp\{-b_1(\mathbf{x}_1 - \mathbf{x}_1')^2 - b_2(\mathbf{x}_2 - \mathbf{x}_2')^2\}.$$

If we are evaluating all the derivatives at every design point, each sub-matrix in A_D will be of size $n \times n$ resulting in the dimensions of $n(d+1) \times n(d+1)$ for the full A_D matrix.

The calculation of $\hat{\boldsymbol{\beta}}$ becomes

$$\hat{\boldsymbol{\beta}} = (H_D^T A_D^{-1} H_D)^{-1} H_D^T A_D^{-1} \mathbf{y}_D,$$

The covariances between the training data and a point where the emulator is predicting at, $\mathbf{t}(\mathbf{x})_D^T$, are arranged in the following form.

$$\mathbf{t}(\mathbf{x})_D^T = \{c(\mathbf{x}, \mathbf{x}_1), \ldots, c(\mathbf{x}, \mathbf{x}_n), c(\mathbf{x}, \mathbf{x}_1^1), \ldots, c(\mathbf{x}, \mathbf{x}_n^1), c(\mathbf{x}, \mathbf{x}_1^2), \ldots, c(\mathbf{x}, \mathbf{x}_n^2)\},$$

where $c(\mathbf{x}, \mathbf{x}_1^1)$ is the covariance between the point, $\mathbf{x}$, which we are predicting at and the derivative of the first input variable at the first design point. Finally, $\hat{\sigma}^2$ becomes

$$\hat{\sigma}_D^2 = \frac{\mathbf{y}_D^T(A_D^{-1} - A_D^{-1}H_D(H_D^T A_D^{-1} H_D)^{-1} H_D^T A_D^{-1})\mathbf{y}_D}{n(d+1) - q - 2}.$$

Now $m^{**}(\mathbf{x})$ and $c^{**}(\mathbf{x}, \mathbf{x}')$ can be calculated as in standard emulation.

2.3 Examples

Here we demonstrate the methodology of Section 2.2 though a 1-dimensional example. The true function is

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

and we choose a linear form for the prior mean, so $\mathbf{h}(\mathbf{x})^T = (1 \quad \mathbf{x})$ and $q = 2$. The covariance function is $\mathrm{c}(x, x') = \exp\{-b\,(x - x')^2\}$ and we decide to run the simulator at the following, $n = 5$, design points:

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

We also calculate the derivatives of the output with respect to x at each of 5 points and together, this yields the training data,

$$\mathbf{y}^T = (-3.87,\ -3.00,\ 2.21,\ 2.52,\ 2.84,\ -0.39,\ 1.75,\ 0.98,\ -0.27,\ 1.38).$$

We then follow the method described in Section 2.2 and derive the distribution of $\eta(x)$ conditional only on y. Maximum likelihood estimation is employed to estimate the roughness parameter, b. To test the emulator we run it at a number of new input points and as here the simulator is computationally cheap, we can also run the simulator at these points to evaluate the predictive performance of the emulator.

The results are shown in Figure 1.

The posterior mean is the blue, dashed line and the blue, dotted lines show the value of two standard deviations above and below the mean. In order to assess the emulator, the true simulator output at these points is shown by the solid black line. We can see from Figure 1 that in this example, the posterior mean is very close to the true value of the simulator and the uncertainty is very small.

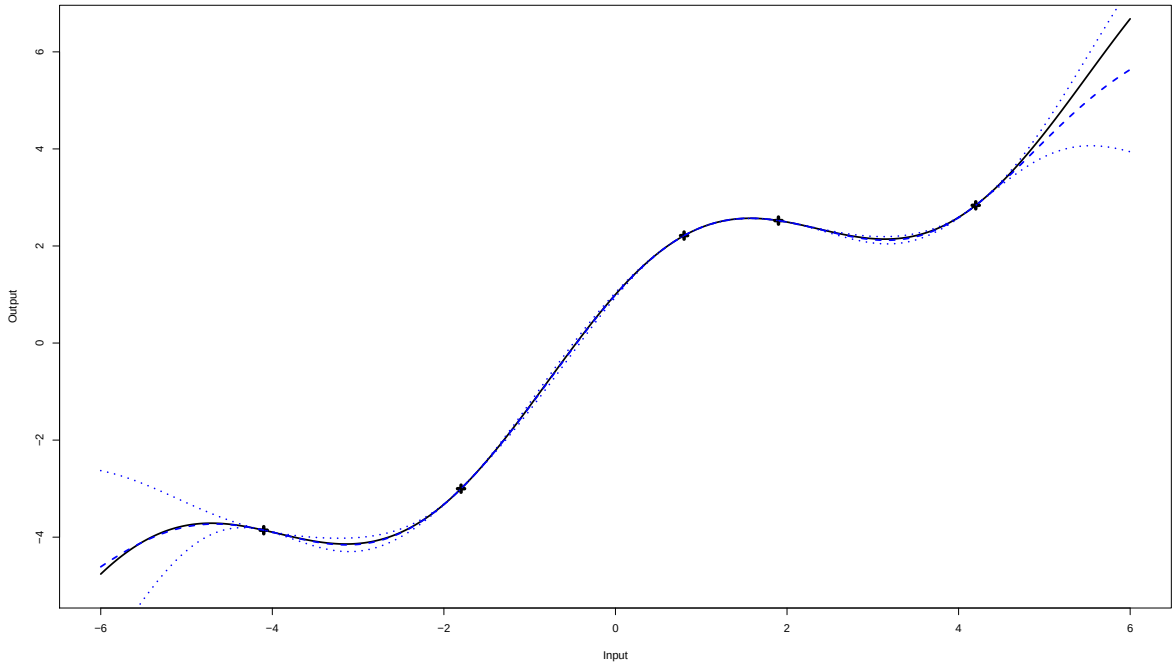


Figure 1: Emulator with 5 design points and derivative information

To compare, the performance of an emulator built with only function output at the 5 points is shown in Figure 2.

The posterior mean is the red, dashed line and the red, dotted lines show the value of two standard deviations above and below the mean. For comparison the solid black line shows the true simulator output at these points.

The performance of the emulator in Figure 1, while very good, is such that it is difficult to identify precisely how and where the derivative information is having an effect. Due to this we now repeat the example but remove two of the simulator runs from the training data. Figure 3 illustrates the performance of this emulator. The blue dashed line, representing the posterior mean, is close to the true simulator output which is given by the black, solid line. The uncertainty, as in the previous figures, is shown by the blue, dotted lines. The uncertainty reduces to zero at design points, as expected, but whereas in Figure 2 the uncertainty becomes appreciable once we start predicting away from a design point, here the uncertainty remains very small for predictions closes to the design points. It is the derivative information in the model which allows for this reduced uncertainty.

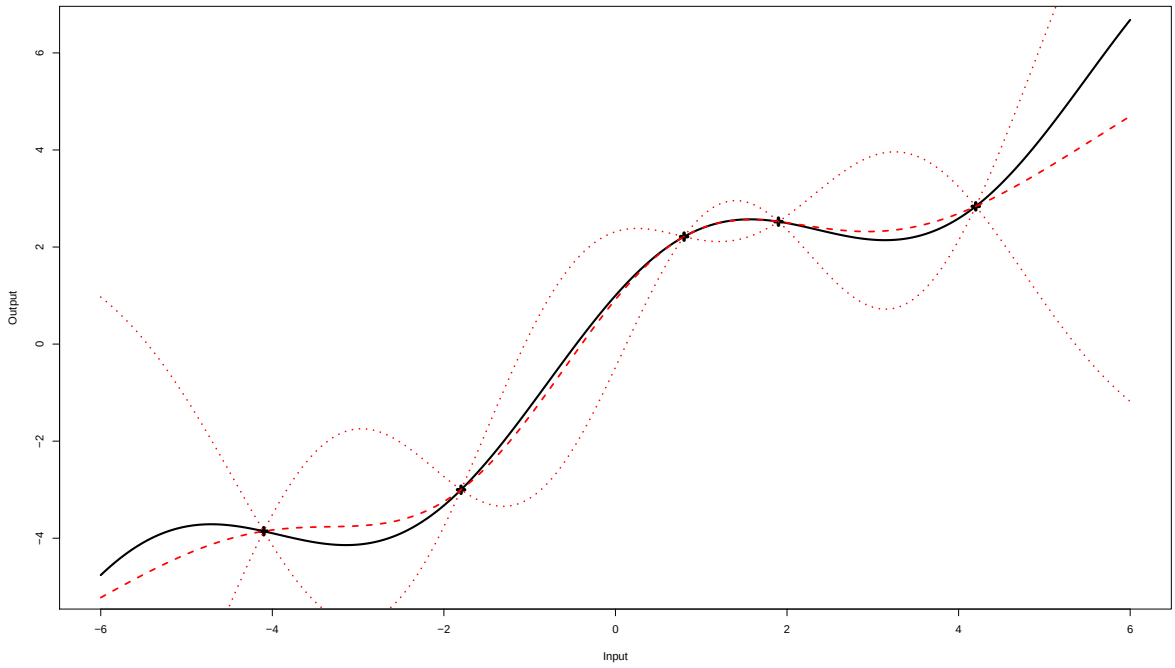


Figure 2: Emulator built with function output only

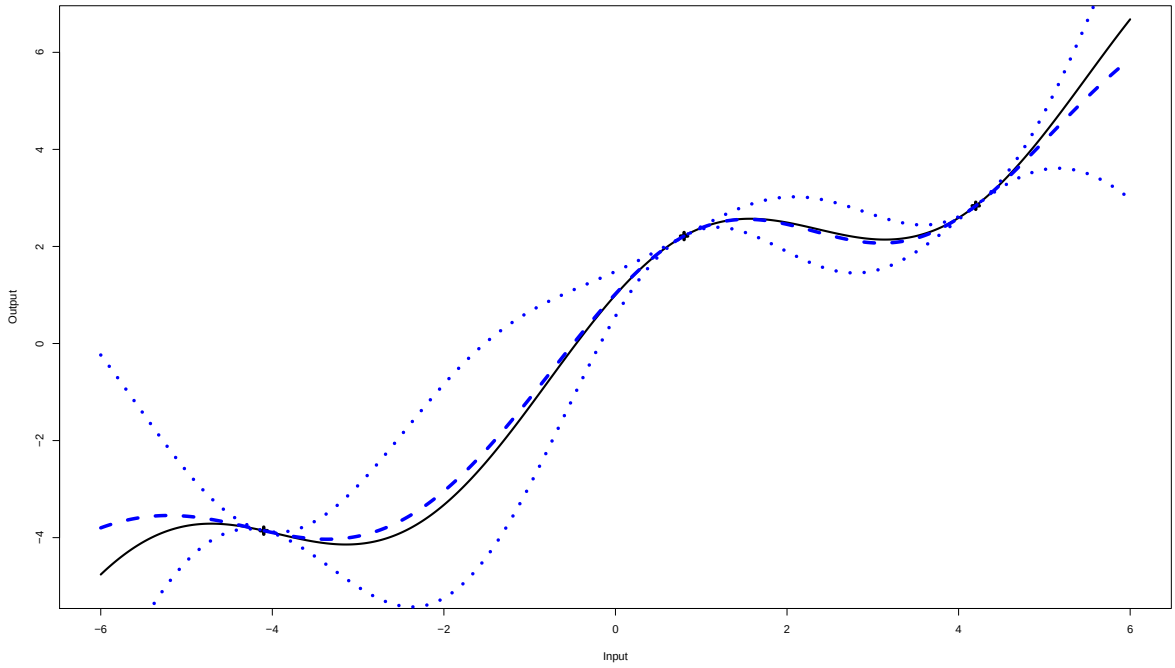


Figure 3: Emulator with 3 design points and derivative information

2.4 Computational cost of obtaining and applying derivative information

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

For a review of some of the methods for producing derivatives see Section 4.3 of Literature Review, MUCM internal report 3.2b.1.

3 Emulation of toy models

3.1 The models

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

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

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

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

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

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

$$\text{flow rate} = \frac{2\pi T_u(H_u - H_l)}{\ln(\frac{r}{r_w})\left[1 + \frac{2LT_u}{\ln(\frac{r}{r_w})r_w^2 K_w} + \frac{T_u}{T_l}\right]}. \tag{3.1}$$

   This is the model chosen by Morris *et al.* (1993) to demonstrate their methodology for including derivative information in the analysis of computer models. There are eight inputs and these are shown in Table 1 along with a range of values that each input takes. Morris *et al.* (1993) choose to vary only two of the inputs, namely r_w and K_w, though they do not a provide an explanation as to why these particular inputs were chosen. In line with their work we continue to vary r_w and K_w though also choose a further input, L, to vary. The remaining five inputs are fixed and set at the lower bound of their range.

All of the 1-dimensional toy models over their respective input regions are shown in Appendix A and are emulated employing the methodology of Section **??**. Where

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

Table 1: Inputs for borehole model

derivatives are included, the additional information is implemented as in the methodology of Section 2 and the models have been differentiated by hand. We choose as a linear form for the prior mean in all emulation here.

The design points at which to run the simulator are chosen by generating a maximin Latin hypercube sample. In the analysis of models 1 to 4, a maximin Latin hypercube sample is generated for the minimum number of runs required for building an emulator with derivatives. To compare this emulator with one built without derivative information subsequent runs of the simulator are required. The design points at which to perform these additional runs are selected by augmenting the current Latin hypercube sample while attempting to conserve the characteristics which make it latin. The resulting collection of points, therefore, is also a Latin hypercube sample. This is performed in R using the function *optAugmentLHS* {*lhs*}, which is part of the *LHS* package. This function attempts to augment a current Latin hypercube sample such that S optimality is maximised. S optimality ensures the mean distance from each point to all other points is maxmised.

It may not be necessary though to ensure existing points are maintained when performing further simulator runs. We want to investigate whether it is better to spend computing time on $n + m$ runs of a simulator or on n runs of the adjoint of a simulator. Different designs will be appropriate in the different cases and as such we may not want to run

the simulator and the adjoint at any of the same points. In this study we do not build adjoints as the simulators can easily be differentiated by hand, but we continue in this way in the analysis of model 5: we create a new maximin Latin hypercube sample for each emulator. We are only emulating toy models in this study and as such, running the simulators at a new set of design points each time we increase the design is possible.

3.2 Method of comparison

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

$$\text{Prediction Error} = \frac{1}{N}\sum \frac{(|\text{True Value} - \text{Predicted Value}|)}{|\text{True Value}|}, \tag{3.2}$$

where N is the number of points the we are predicting the function output at.

It is difficult to quantify exactly the computational cost of obtaining derivatives. In Section 2.4 various techniques for computing derivatives are reviewed, but any cost factor given is model specific. Though it is generally agreed an adjoint provides the most efficient method of obtaining derivatives, there is not a general rule for the additional computational cost required to run the adjoint model. One reason for this is that the computational cost depends greatly on how efficiently the adjoint is written. Therefore, in order to investigate the value of derivative information in toy models, we will analyse how many extra runs are required for an emulator without derivatives, to give a comparable prediction error to that of an emulator built with derivative information, from a specified number of runs.

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

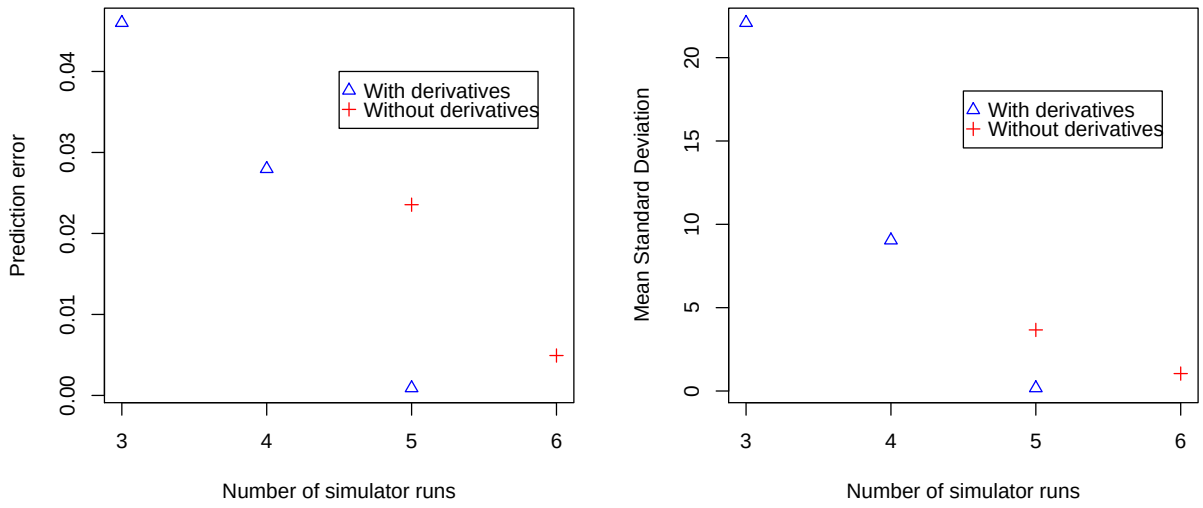
$$\text{Mean Standard Deviation} = \frac{1}{N}\sum \hat{\sigma}\sqrt{\mathrm{c}^{**}(x_i, x_i)}, \tag{3.3}$$

where $i \in (1, \ldots, N)$.

Bastos and O'Hagan (2008) give various diagnostics for the validation of Gaussian process emulators. The focus is on comparing predictions made by an emulator to the simulator outputs, under the same inputs. The diagnostics presented by Bastos and O'Hagan (2008) may therefore be appropriate to the comparison of two emulators.

3.3 Results

To compare the differences in the predictive performance and in the uncertainty between emulators, Figures 4 - 9 have been produced. Part (a) of these figures show the prediction error, as calculated by (3.2) for emulators built with an increasing number of simulator runs. The measure of uncertainty across emulators built with a growing number of design points, as evaluated by (3.3), are shown in part (b). Throughout Figures 4 - 9, blue triangles represent the performance of emulators built with derivative information while red crosses display the equivalent for emulators built only with function output.

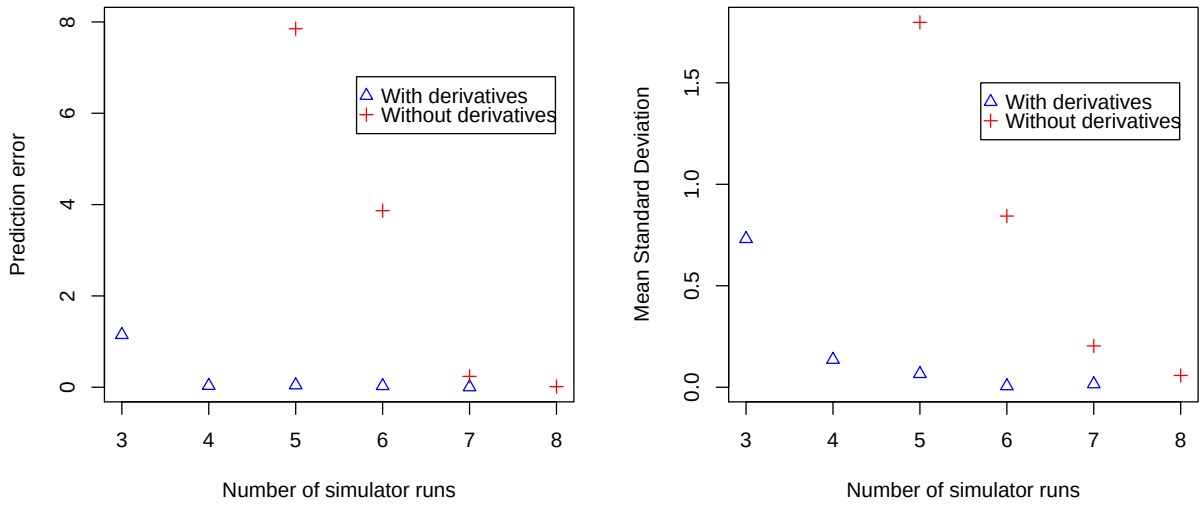


(a) Comparison of prediction error

(b) Comparison of uncertainty

Figure 4: Performance of emulators built with varying numbers of runs of model 1

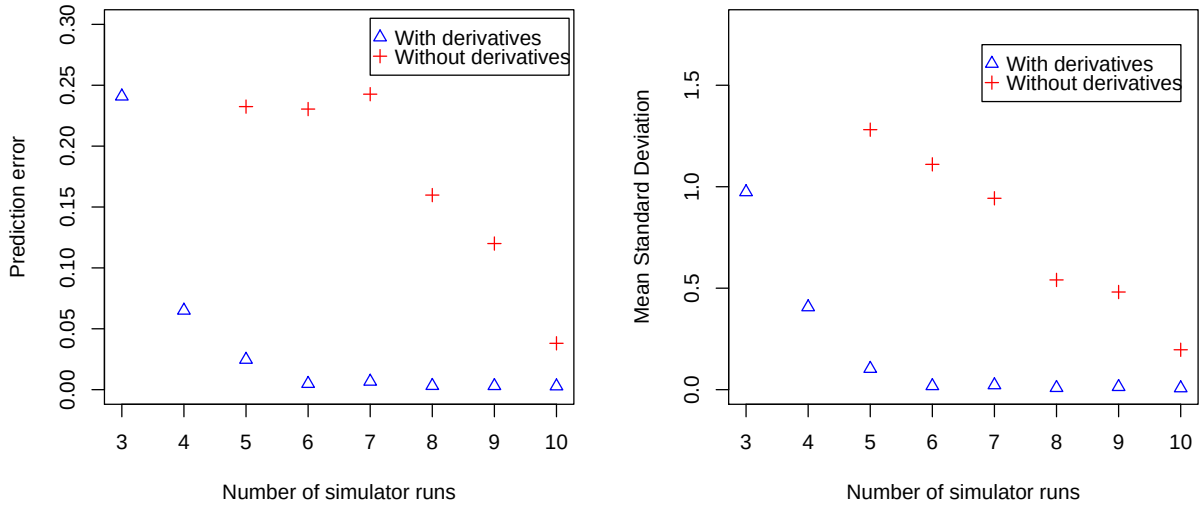
Figures 4 - 9 clearly show that for all the toy models tested here, the mean of emulators built with the additional information of derivatives provide a closer approximation to the relevant simulator. The mean standard deviation is also reduced for that of emulators with simulator derivatives. The prediction error and the uncertainty for the rough simulator, as shown in Figure 7, follow a less strict trend than the other toy models. This may be due to the difficulty in estimating the smoothness parameter, b, for this model. We continue by fixing this parameter at an appropriate value and rebuilding the emulators, with and without derivatives. Due to the roughness of model 4 it is necessary to select a very



(a) Comparison of prediction error

(b) Comparison of uncertainty

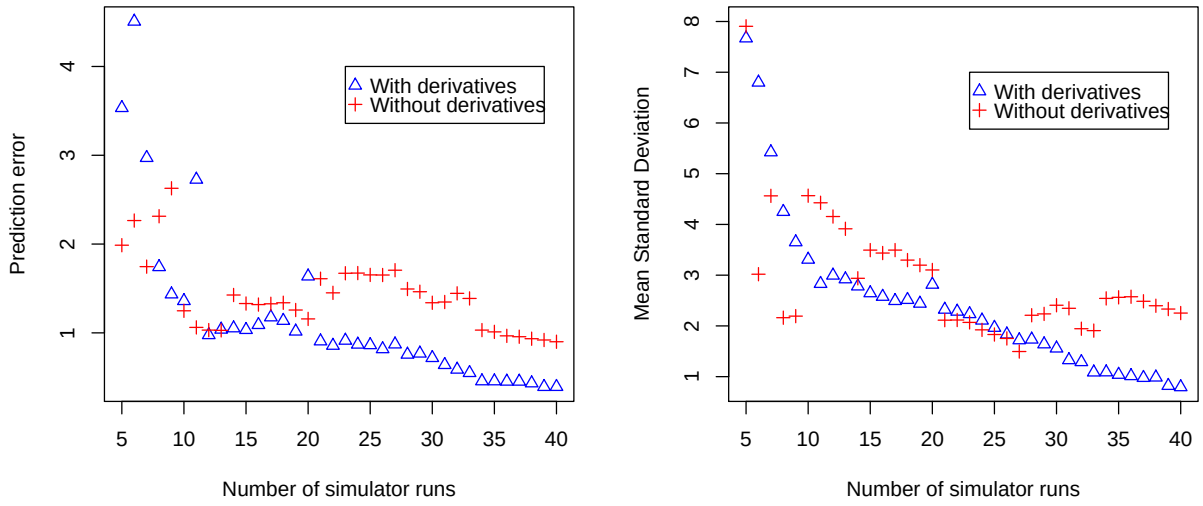
Figure 5: Performance of emulators built with varying numbers of runs of model 2



(a) Comparison of prediction error

(b) Comparison of uncertainty

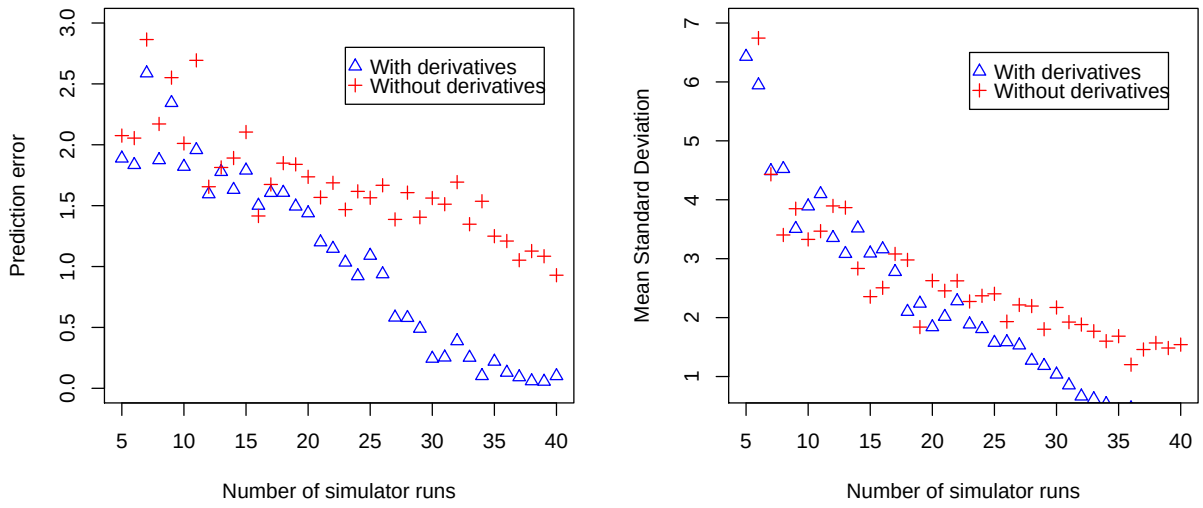
Figure 6: Performance of emulators built with varying numbers of runs of model 3



(a) Comparison of prediction error

(b) Comparison of uncertainty

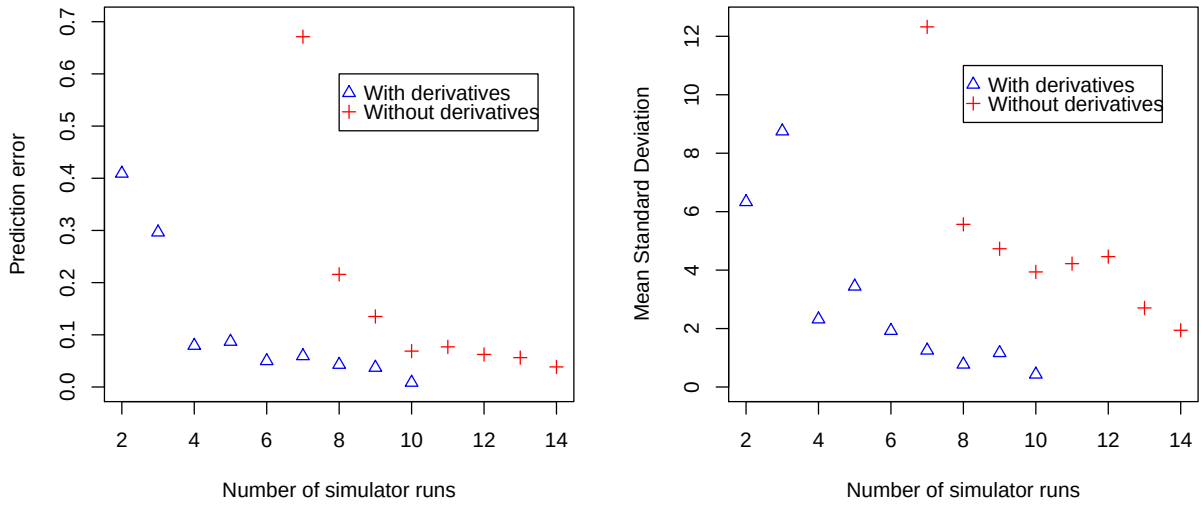
Figure 7: Performance of emulators built with varying numbers of runs of model 4



(a) Comparison of prediction error

(b) Comparison of uncertainty

Figure 8: Performance of emulators built with varying numbers of runs of model 4 with $b = 2000$



(a) Comparison of prediction error

(b) Comparison of uncertainty

Figure 9: Performance of emulators built with varying numbers of runs of model 5

high value for b and the results from the previous emulation show 2000 is approximately suitable. Figure 8 shows the resulting comparisons of the prediction error and uncertainty with $b = 2000$. The derivative information appears to have no effect in the emulators built with a small number of simulator runs. This is due to the high value for b. As the number of design points are increased beyond 19 the prediction error for the emulators built with derivative information decreases much more quickly than for the corresponding emulators built only on function output. A similar pattern is observed in Figure 8b which shows how the uncertainty is effected.

However, as discussed in Section 2.4 and Section 3.2, the computational cost of obtaining derivative information must be taken into consideration. Table 2 shows how many extra runs are required for the emulators without derivatives, to achieve similar prediction errors to the emulators with derivative information; while Table 3 displays the equivalent information for mean standard deviation. We can see from Tables 2 and 3, for models 2, 3 and 4, an emulator without derivative information requires approximately twice as many simulator runs as an emulator with derivative information to achieve similar accuracy.

Model	Derivatives	Prediction Error	Design Points
1	Yes	0.02798	4
1	No	0.02355	5
1	Yes	0.0009	5
1	No	0.0049	6
2	Yes	0.0336	4
2	No	0.0135	8
3	Yes	0.02467	5
3	No	0.03807	10
4	Yes	0.90597	21
4	No	0.90004	40
4 (fixed b)	Yes	0.92182	24
4 (fixed b)	No	0.92875	40
5	Yes	0.0372	9
5	No	0.0385	14

Table 2: Comparison of predictive performance of different emulators

Model	Derivatives	Mean Standard Deviation	Design Points
1	Yes	0.1766	5
1	No	1.046	6
2	Yes	0.0659	5
2	No	0.05816	8
3	Yes	0.1033	5
3	No	0.1966	10
4	Yes	2.235	23
4	No	2.253	40
4 (fixed b)	Yes	1.53095	27
4 (fixed b)	No	1.54153	40
5	Yes	1.924	6
5	No	1.941	14

Table 3: Comparison of the uncertainty of different emulators

4 Predicting derivatives

In this section we discuss the methodology of emulating derivatives and illustrate it on a 1-dimensional example.

4.1 Methodology

As detailed in Section 2.2, if $\eta(.)$ is described by a Gaussian process then the derivatives of $\eta(.)$ are also described by a Gaussian process. The posterior distribution of the derivative of $\eta(\mathbf{x})$ becomes

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

where

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

4.2 Example

We illustrate the method described in Section 4.1 with a 1-dimensional example. We choose to continue the example of Section 2.3 where the true model is

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

We have a linear form for the prior mean and evaluate the function output and the derivatives of the simulator at $n = 5$ design points. We follow the methodology of Section 4.1 and derive the posterior distribution of the derivative of $\eta(\mathbf{x})$. We then attempt to predict the derivatives of the true function at a number of new input points. We can see the performance of this emulator in Figure 10. The posterior mean is the green, dashed line and two standard deviations above and below this mean are shown by the green,

dotted lines. The black, solid line reveals the true derivatives of the simulator at these points. Figure 10 shows that in this example, we can emulate derivatives very accurately.

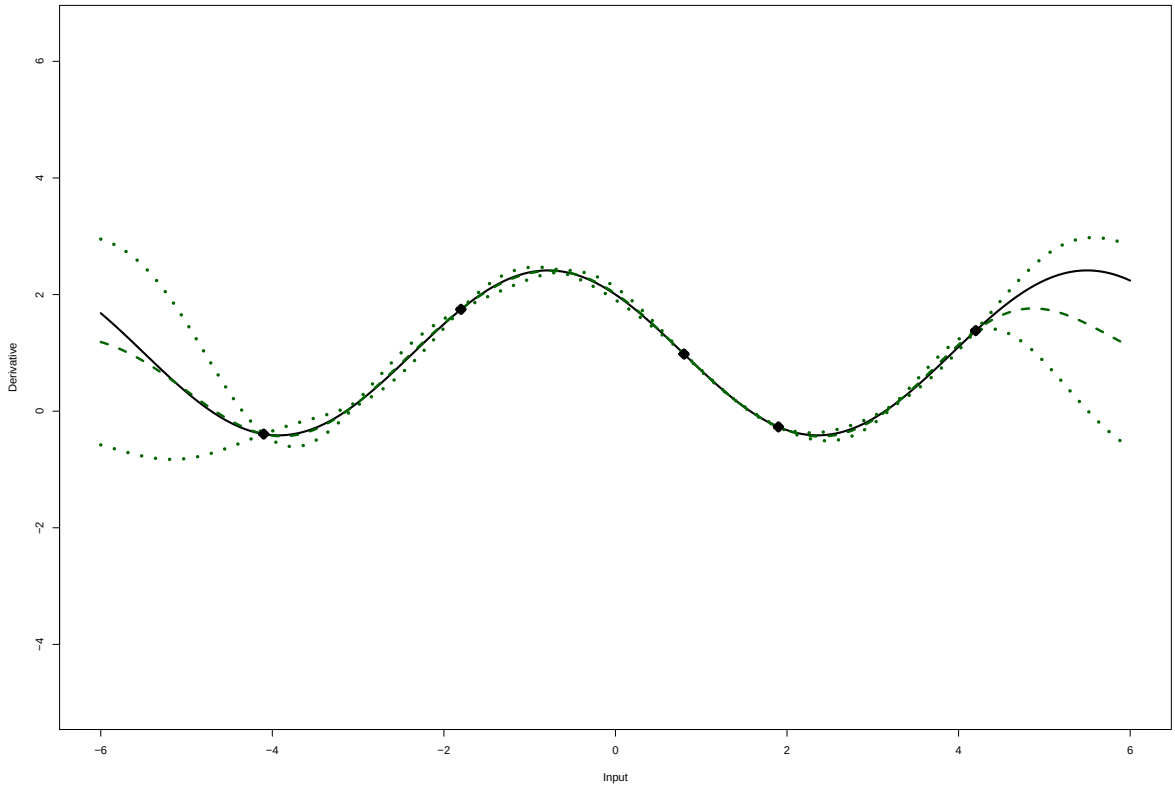


Figure 10: Emulating derivatives based on function output and known derivatives at 5 points

We may not have a model's adjoint and as such, the derivatives of the simulator at the design points are unknown. We can still, however, emulate the derivatives. Figure 11 shows the performance of an emulator predicting the derivatives of the the true model, $\eta(x) = x + \cos(x) + \sin(x)$, based on training data which consists only of the simulator output at the $n = 5$ design points. As in Figure 10, the posterior mean is the green, dashed line and two standard deviations above and below this mean are shown by the green, dotted lines. The black, solid line reveals the true derivatives of the simulator at these points. We can see from Figure 11 that though we can still predict the derivatives of the true function, the uncertainty is much greater and the posterior mean is further from the true value than in the emulator which had derivative information included in

the training data. It should be noted, though, that the computational expense required to build the two emulators of Figures 10 and 11 are not equal.

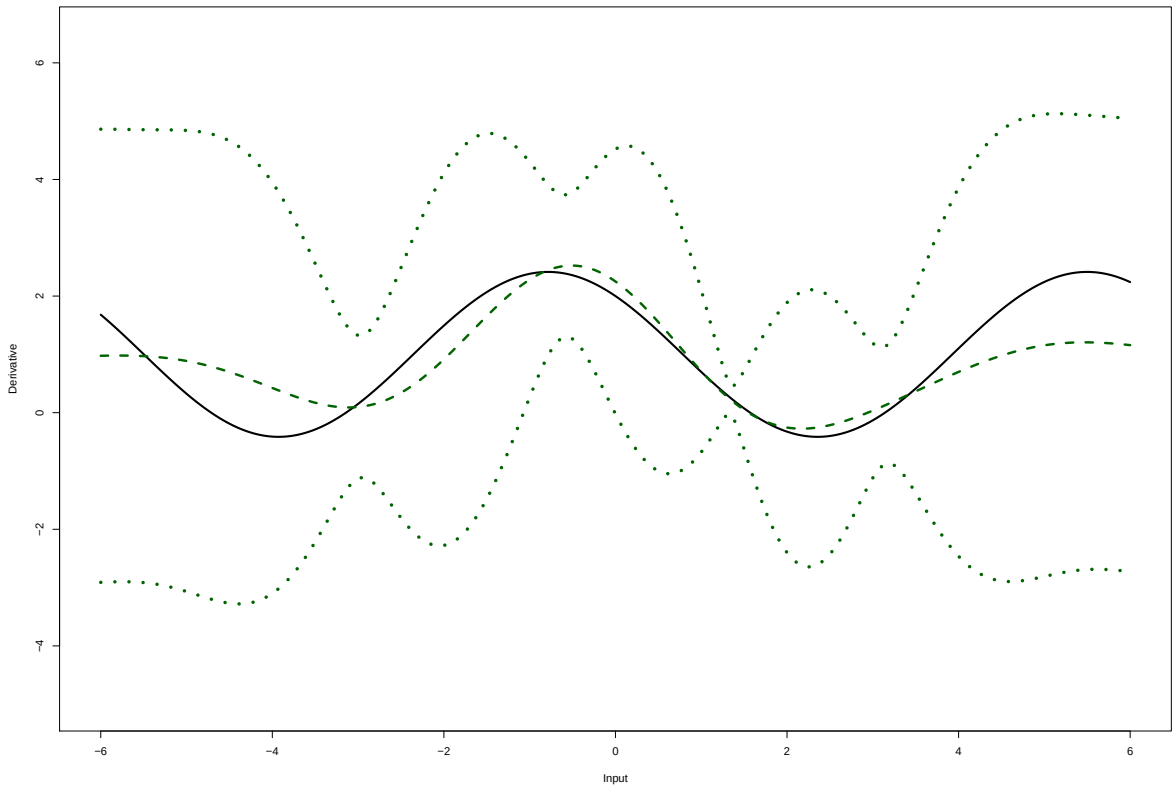


Figure 11: Emulating derivatives based on function output at 5 points

We are particularly interested in emulating derivatives when there isn't an adjoint, or similar technique for obtaining simulator derivatives, available. It would appear from Figure 11 that without derivative information in the training data, 5 runs of this simulator aren't enough for the emulator mean to produce an adequate approximation. Continuing with the example, we evaluate 3 more runs of the simulator, $\eta(x) = x + \cos(x) + \sin(x)$, at the points $x_6 = -0.50, x_7 = 3.05, x_8 = -4.90$. This yields an emulator, of which the mean and standard deviation is shown in Figure 12.

Clearly with 8 runs of the simulator, the emulator mean is approximating the derivatives of the simulator very well and with little uncertainty. For completeness, the performances of an emulator built with 6 simulator runs and 7 simulator runs respectively, are shown in Figures 17 and 18 in Appendix B.

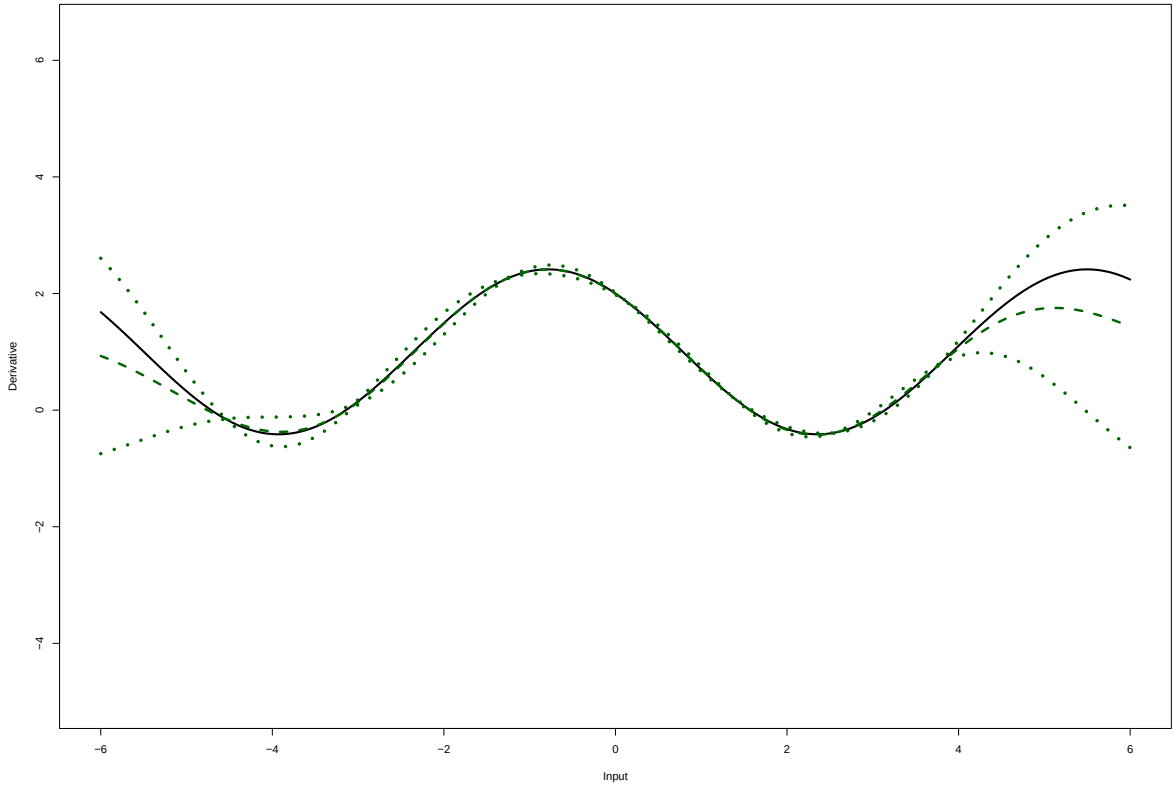


Figure 12: Emulating derivatives based on function output at 8 points

These examples serve to illustrate the methodology of Section 4.1 only and further investigation into the performance of these emulators on different models and with varying amounts of training data is required.

5 Conclusions

In this document we have described how derivative information can be used in the statistical analysis of computer models and investigated the value of this information in toy model examples. The emulation of all toy models is good with the inclusion of derivatives though whether this results in a more efficient emulator depends on the computational cost of obtaining the derivative information. We have shown that in most of the 1-dimensional toy models, to obtain similar levels of prediction error and uncertainty, an emulator without derivatives requires approximately twice as many runs of the simulator as the emulator with derivative information. This suggests that if the computational expense of running the simulator and generating derivatives is less than twice that of running the model alone, then derivatives provide a more efficient use of computing time.

In the 1-dimensional examples, with the exclusion of the very smooth model (toy model 1), the level of smoothness of the simulator didn't appear to have any effect on the value of the derivative information. The very smooth model, however, showed that an emulator without derivatives could achieve a similar level of prediction error with only one more simulator run.

The 3-dimensional example (toy model 5), which involves modelling the flow of water through a borehole provided fairly similar results to those of the 1-dimensional models. The emulator with derivatives required approximately two thirds as many runs as the emulator without derivatives to produce a similar prediction error. Whereas, to obtain similar levels of uncertainty, the emulator with derivatives required just under half as many simulator runs as the emulator without derivatives. As the number of input dimensions of a model goes up, we would expect the computational cost of obtaining the derivatives, in comparison with running the model alone, to also go up. Thus it would appear the derivative information is less valuable in the emulation of the borehole model than it is

the in the 1-dimensional models. The borehole model is particularly smooth though, and so this supports the result of the very smooth toy model (toy model 1). It may have been preferable to have performed some sensitivity analysis on this model, in order to more formally determine which three inputs to vary and this may be considered in further work.

The ability to emulate derivatives as well as function output has also been considered. Section 4 showed that it's possible to build an emulator, with or without derivative information, which can predict the derivatives of the simulator. While the performance of an emulator is improved if derivative information is included in the training data, further investigation into the value of derivatives here with respect to the computational expense is required. We have shown however that given enough runs of the simulator, we can emulate derivatives to a satisfactory degree without any derivative information in the training data. This is shown by Figure 12.

Further work on the application of derivatives information in toy models, currently in progress, includes investigating the impact of derivative information when estimating the smoothness parameters. Data from a Gaussian process is generated such that the value of b is known and we examine what effect the derivatives have when estimating b. Also, currently in progress is a more detailed investigation into emulating derivatives, similar to that of emulating function output in Section 3.

Appendix

A Simulators for toy model experiment

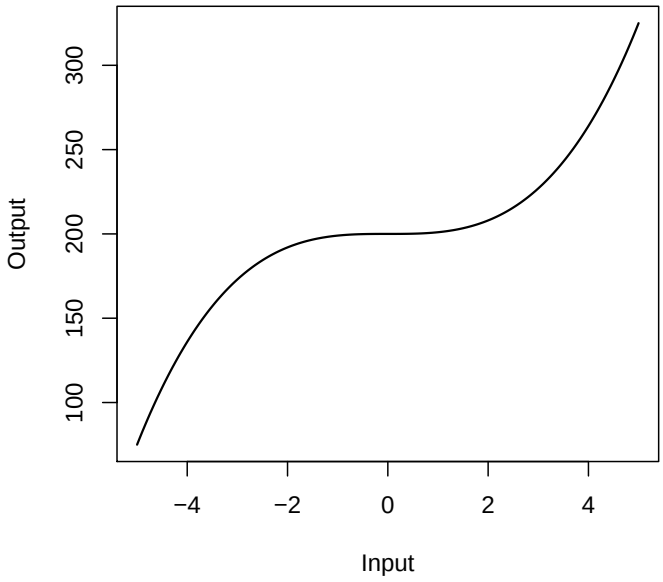


Figure 13: Simulator 1

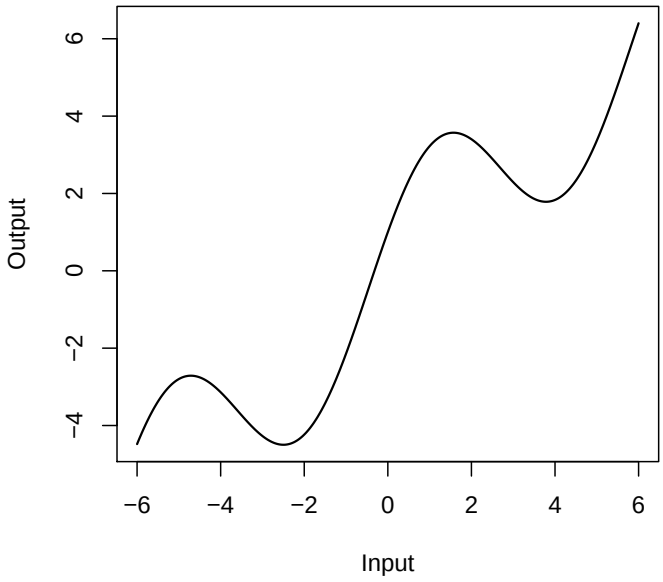


Figure 14: Simulator 2

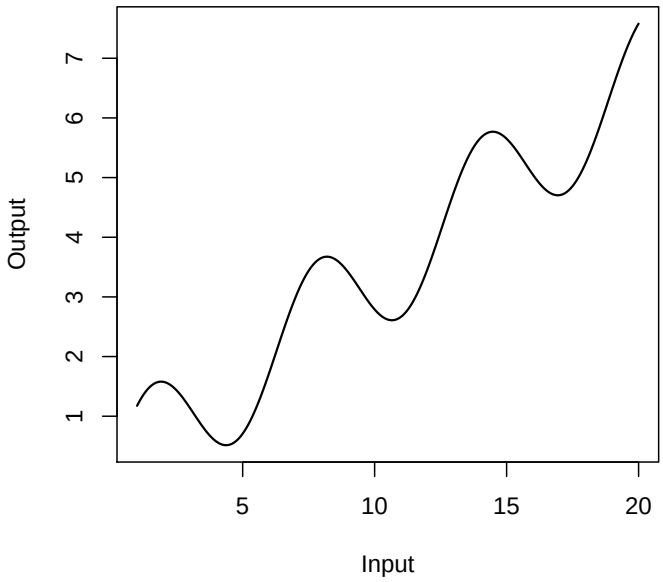


Figure 15: Simulator 3

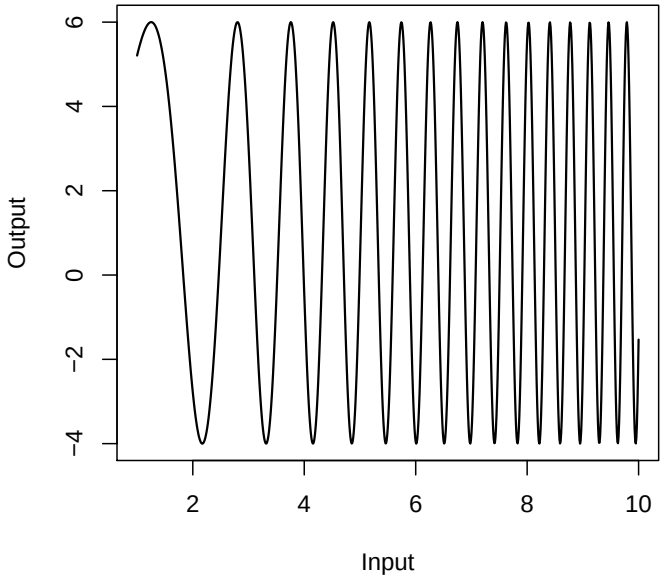


Figure 16: Simulator 4

B Emulating derivatives

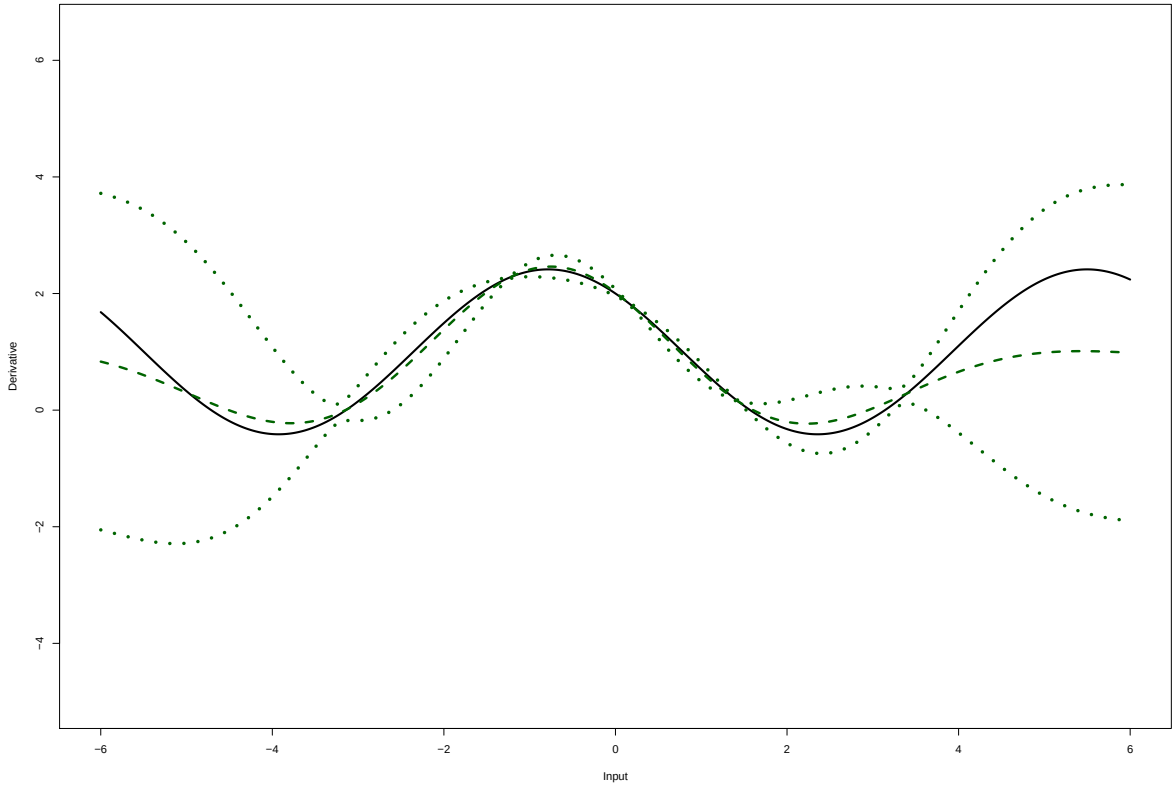


Figure 17: Emulating derivatives based on function output at 6 points

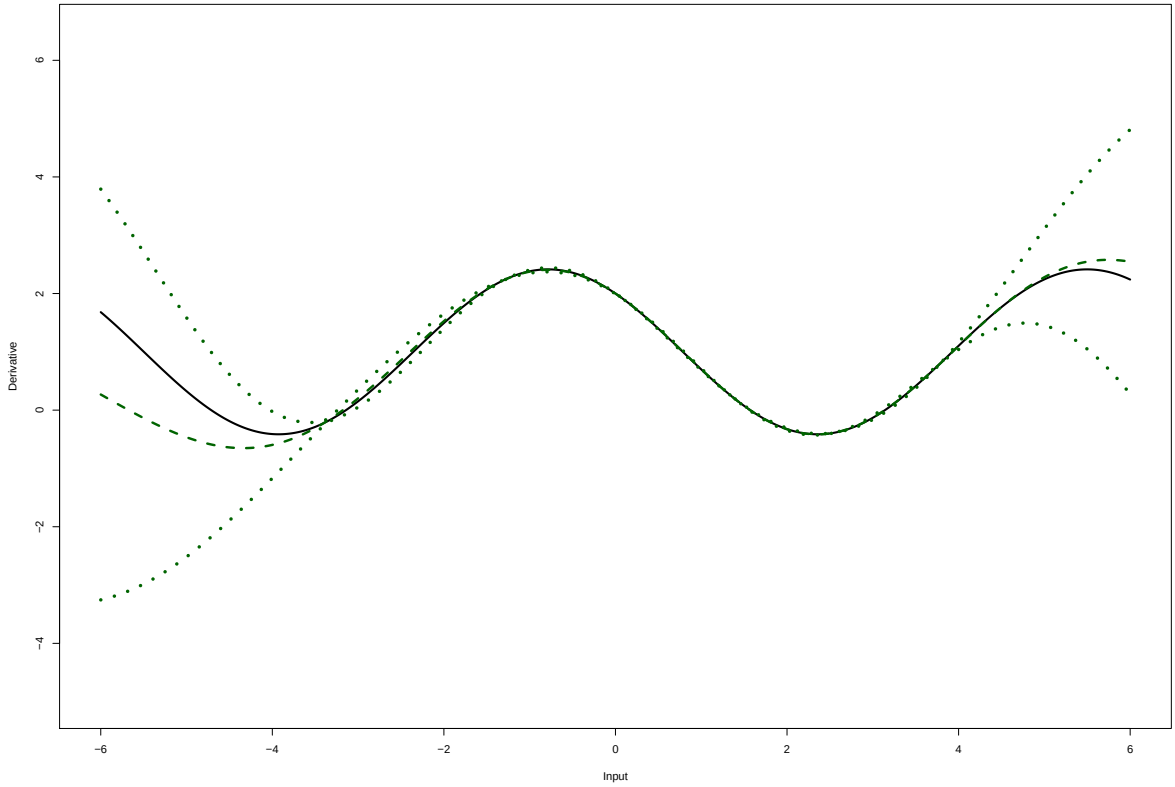


Figure 18: Emulating derivatives based on function output at 7 points

References

Azman, K. and Kocijan, J. (2005). Comprising prior knowledge in dynamic gaussian process models. *CompSysTech Proceedings*.

Bastos, L. S. and O'Hagan, A. (2008). Diagnostics for gaussian process emulators. *Submitted to Technometrics*.

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.

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.

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.

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.

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*.