Using Known Boundaries to Improve Bayesian Emulation

Rebekah Fearnhead

Durham University

A report submitted for the degree of

MSci

Easter 2024

Abstract

As more complex models that need to be evaluated are being used in a variety of situations, ways to efficiently perform these evaluations and model the function need to be developed. One way to do this is to use an emulator, for example a Bayes Linear Emulator. Whilst these perform well, many techniques have been being developed to improve the performance of these emulators and reduce the number of expensive evaluations that need to be preformed. One of these techniques is to add information about a boundary that can be easily analytically solved to the emulator.

Contents

Statement of Originality

This piece of work is a result of my own and I have complied with the Department's guidance on multiple submission and on use of AI tools. Material from the work of others not involved in the project has been acknowledged, quotations and paraphrases suitably indicated, and all uses of AI tools have been declared.

Chapter 1

Introduction

Complex Mathematical Models are being used more commonly in many scientific areas to help to describe complex physical systems. For example, these can be used in systems biology [8], disease models [9], or to simulate galaxy formation [2]. One problem with this is that it could take from a week to a month to do a single evaluation so often, it is not feasible to fully evaluate these models.

Another problem which occurs with these complex models is that they contain many sources of uncertainties. To perform analysis on these full models, all of these uncertainties would need to be identified, and their effects quantified. This is difficult to do because of the complexities of these models, and even if this is done, many more simulator evaluations would need to be performed in order to account for all the possibilities. This will also increase the time it takes for the full model to be simulated.

One way to solve these problems is to use an emulator which mimics the computer model function that we want to evaluate, but will do it much faster than the full model for the complex system. An emulator provides a prediction for the outputs of the function, $f(x)$, along with an uncertainty statement for the prediction.

Chapter 2

Emulation in One Dimension

The simplest type of emulator that can be used is a Bayes Linear Emulator. In order to mimic $f(x)$, it only uses the second order specifications: expectation and variance, to make predictions for the function. The most basic version of this emulator is $f(x) = u(x)$, with $u(x)$ being a weakly stationary process which reflects the prior beliefs about $f(x)$.

Sections 2.1 and 2.2, which describe how to specify our priors and how to update them based on data, are based on material [2].

2.1 Specifying Prior Beliefs

In order to capture our prior beliefs about $f(x)$, there are three main quantities that need to be specified. The values of these can be estimated using knowledge of the underlying equations of the real system which we are modelling.

Firstly, the location of the function needs to be given. This can be done by giving the prior expectation, $\mathrm{E}[f(x)]$.

The second quantity that needs to be given is the variability, which can be used to give the prior variance, $\mathrm{Var}[f(x)] = \sigma^2$, for the model. The value of this needs to be such that most of the values of the true function will lie within a 3σ interval of the prior expectation [4].

The final thing to be considered is the smoothness of the function. This can be specified in the emulator by defining the prior covariance structure, $\mathrm{Cov}[f(x), f(x')]$. As well as defining the covariance structure, this also requires a correlation length parameter, θ to be chosen. This defines how close points have to be to influence the values of each other. A quarter of the range of x is often a reasonable value for this.

2.2 Updating Beliefs

Once we have specified our prior beliefs, Bayes Linear Statistics can be used to update our second order beliefs for $f(x)$, given a new input x. The update equations that are used are:

$$\mathrm{E}_D[f(x)] = \mathrm{E}[f(x)] + \mathrm{Cov}[f(x), D]\mathrm{Var}[D]^{-1}(D - \mathrm{E}[D]), \tag{2.1}$$

$$\mathrm{Var}_D[f(x)] = \mathrm{Var}[f(x)] - \mathrm{Cov}[f(x), D]\mathrm{Var}[D]^{-1}\mathrm{Cov}[D, f(x)]. \tag{2.2}$$

This produces $\mathrm{E}_D[f(x)]$ and $\mathrm{Var}_D[f(x)]$, the expectation and variance, adjusted by data D, for $f(x)$. Compared to fully evaluating all the points in the model, calculating these values could be up to 10^9 or 10^{12} times faster, which shows the advantage of using an emulator for these models.

2.3 Example in 1D

An example of this process can be shown using the 1 dimensional function,

$$f(x) = 0.45\sin(7x) + 0.2\cos(6\pi x) - x^2 + 0.2, \tag{2.3}$$

with 5 evaluations of the function equally spaced between 0 and 1.

Unlike the complex models that emulators are usually used on, this function can be quickly evaluated, however, this allows us to compare the performance of the emulator to the true function.

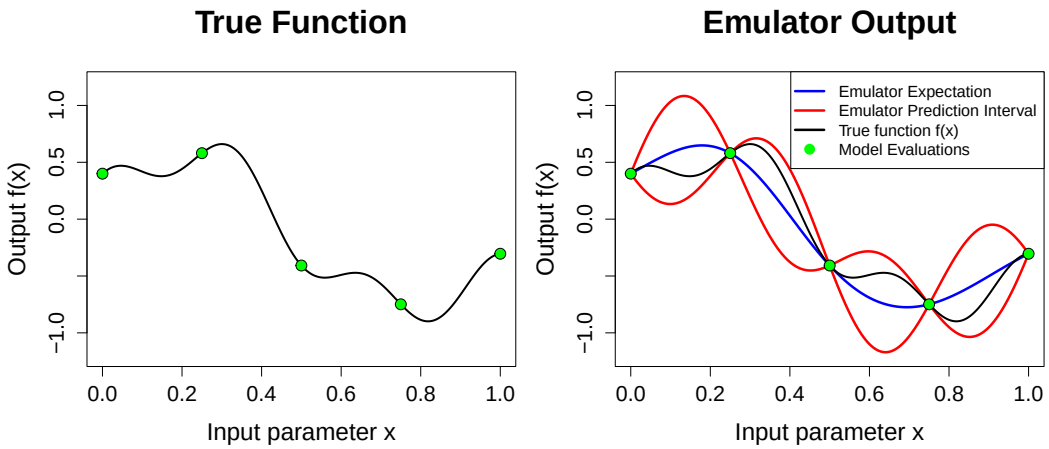
2.3.1 Specifying Priors

Before an emulator can be used, the prior beliefs from Section 2.1 need to be specified.

For this example, the prior expectation, $\mathrm{E}[f(x)]$, will be 0 as this is where the function is located. The prior variance, $\mathrm{Var}[f(x)] = \sigma^2$, gives a 3σ interval in which most of the function should lie, compared to the prior expectation. A good value, therefore is $\sigma = 0.5$, however, depending on the function, a different value may perform better. Finally, the prior covariance structure, $\mathrm{Cov}[f(x), f(x')]$, and the correlation length, θ, need to be given. As a quarter of the range of x often works well, $\theta = 0.25$. For the covariance, a squared exponential structure, $\sigma^2 \exp\left(\frac{|x-x'|^2}{\theta^2}\right)$, will be used.

2.3.2 Emulation Results

Figure 2.1 shows the results of a simple one dimensional emulator on the example function. The black line shows the true function being modelled, with the green points showing the points at which we know the true values. The blue line shows the emulator expectation for



(a) (b)

Figure 2.1: The (a) true function being modelled by the emulator, and (b) the emulator given 5 runs, with the blue line giving the emulator expectation, and the red lines giving the prediction interval.

the function $f(x)$. This goes through the known points, however, the prediction for the rest of the function is smoother than the true function. The red lines show the 3σ prediction interval, and the true function is fully inside this interval.

One way to measure the performance of the emulator is to use emulator diagnostics. This compares the predicted values of the function using the emulator, to the true values and is calculated as [6]:

$$S_D(f(x)) = \frac{f(x) - \mathrm{E}_D[f(x)]}{\sqrt{\mathrm{Var}_D[f(x)]}}. \tag{2.4}$$

A value of $S_D(f(x))$ with a magnitude greater than 3 could suggest an inconsistency in the prior model.

Figure 2.2 shows the emulator diagnostics for the emulator from Figure 2.1. All the values are between -3 and 3 which shows that the emulator is performing well, and suggests that good prior parameters were chosen for the model.

2.4 Varying Prior Parameters

Despite the emulator diagnostics showing that the emulator that was built performed well, the affects of changing each of the three parts of the prior specification can be investigated.

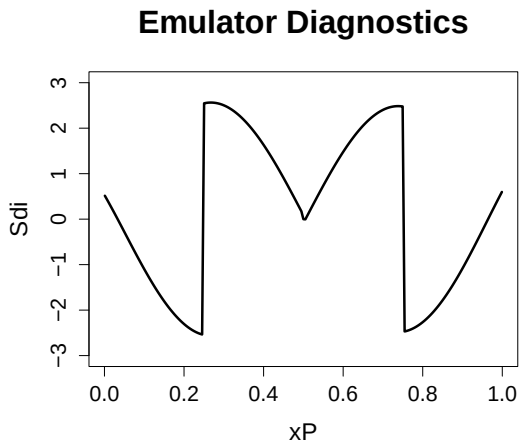


Figure 2.2: The emulator diagnostics for the function given the emulator built on 5 known runs.

2.4.1 Varying the Prior Expectation

Firstly, the prior expectation can be changed to different values between -1 and 1, whilst keeping the values of the other two parameters constant with $\sigma = 0.5$ and $\theta = 0.1$. Decreasing the value of theta compared to the original emulator better shows how changing the prior expectation varies the output from the emulator.

From Figure 2.3, it can be seen that increasing the prior expectation causes the values that the predicted function takes between the known points are higher. As the affect of the evaluated runs on predicting the value of the function decreases, the predicted values will tend towards the prior expectation. This also causes the maximum value that the prediction interval takes to increase due to the prior expectation increasing.

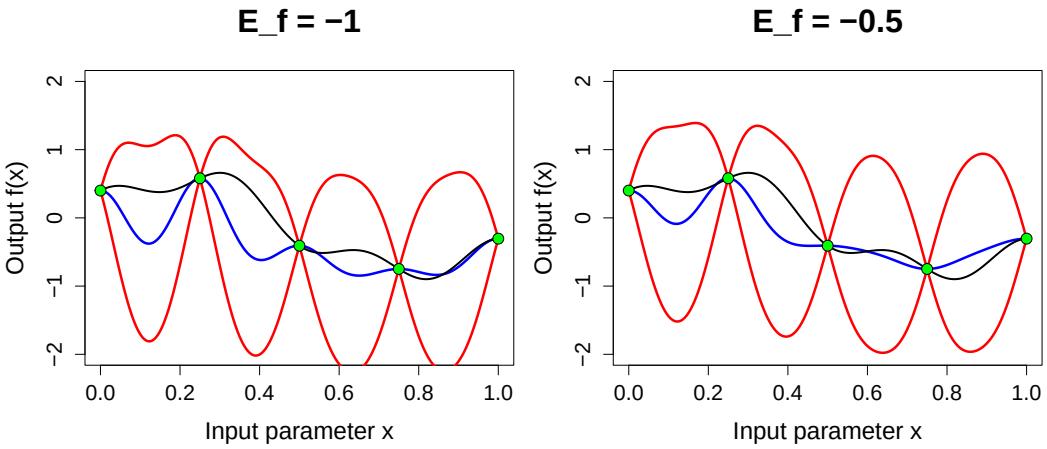
2.4.2 Varying the Prior Variance

The second prior value that can be varied is the prior variance, $\text{Var}[f(x)] = \sigma^2$. The prior expectation will be kept at 0, and the correlation length, $\theta = 0.25$, whilst sigma will take values between 0.25 and 1.

As expected, Figure 2.4 shows that increasing the value of sigma, causes the size of the prediction interval to increase. However, this does not cause any changes to the value of the adjusted expectation, shown in blue.

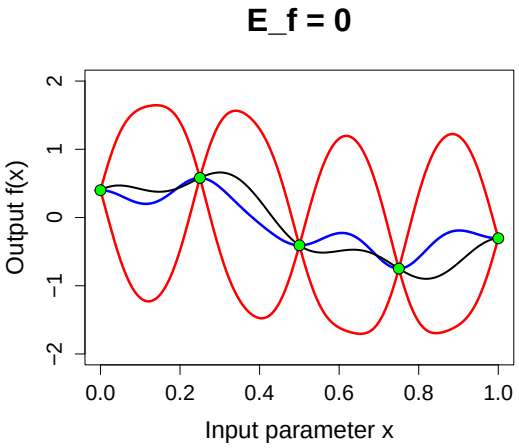
2.4.3 Varying the Correlation Length

The final prior parameter that can be varied is the correlation length, θ. This is used to describe how much of the area around each of the known runs is affected by them. Whilst

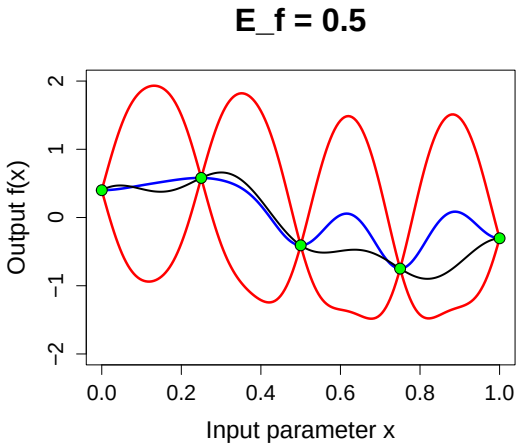


(a)

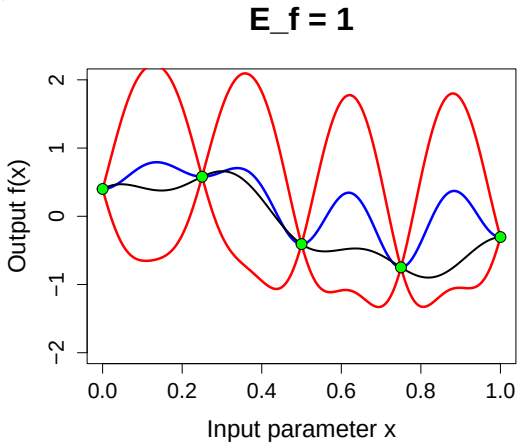
(b)



(c)



(d)



(e)

Figure 2.3: The emulator expectation and prediction interval, with $\theta = 0.1$, $\sigma = 0.5$ and prior expectation, $\mathrm{E}[f(x)]$ = (a) -1, (b), -0.5, (c) 0, (d) 0.5 and, (e) 1.

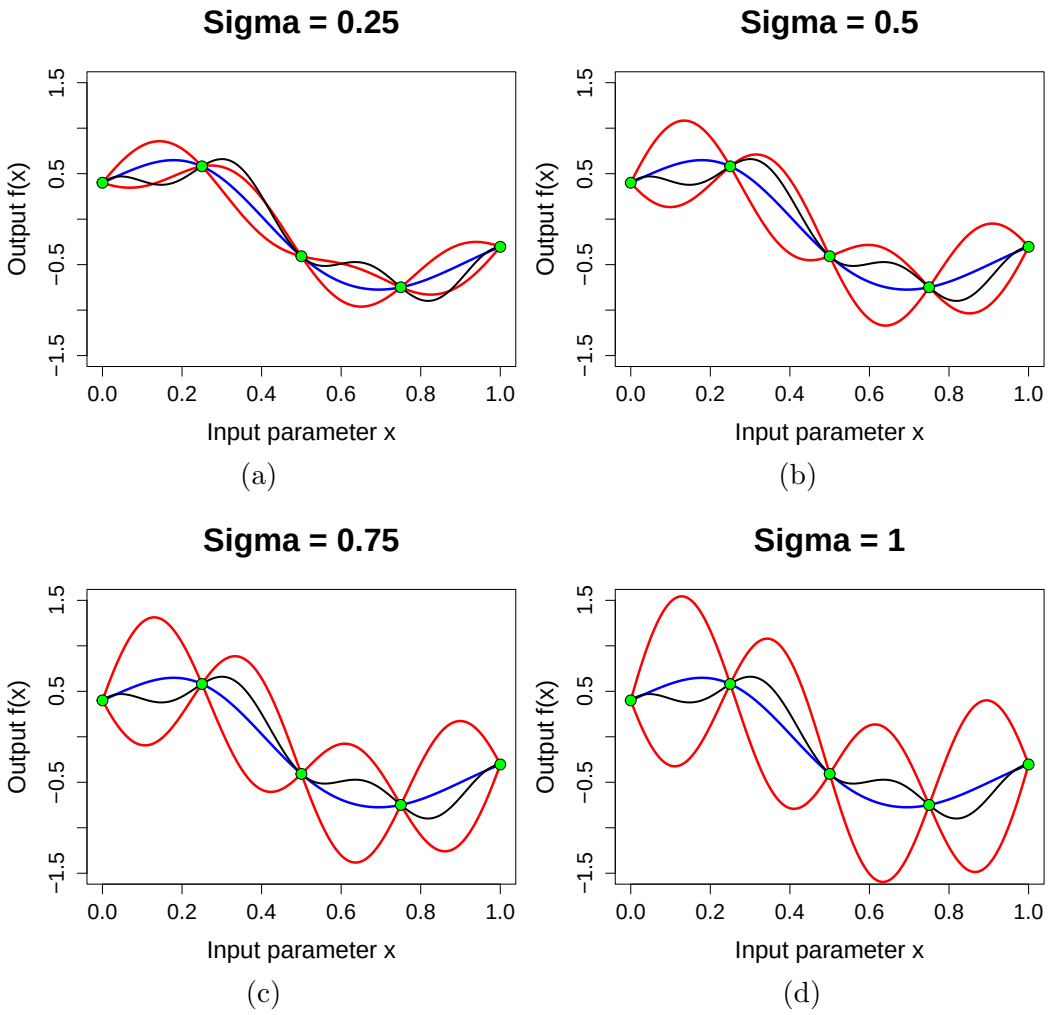


Figure 2.4: The emulator expectation and prediction interval, with $\theta = 0.25$, $\mathrm{E}[f(x)] = 0$, and $\sigma =$ (a) 0.25, (b), 0.5, (c) 0.75 and, (d) 1.

keeping the prior expectation at 0, and prior variance at 0.5^2, the correlation coefficient can be varied between 0.1 and 0.5.

Figure 2.5 shows that by changing the correlation length, both the adjusted expectation and adjusted variance are affected. With a smaller correlation length, a smaller area around each known run is influenced, so areas far from the runs tend to the prior expectation, and variance. However, with larger values for θ, the known runs have a bigger influence on more of the function, which leads to the emulator being more sure of the shape of the function, decreasing the prediction interval. For $\theta = 0.5$, the emulator is sure of the shape of the function, however, its adjusted expectation does not model the true function well, and most of the true function, in black, is outside of the prediction intervals in red. This means, that whilst using a high value for theta leads to emulators producing results with a smaller

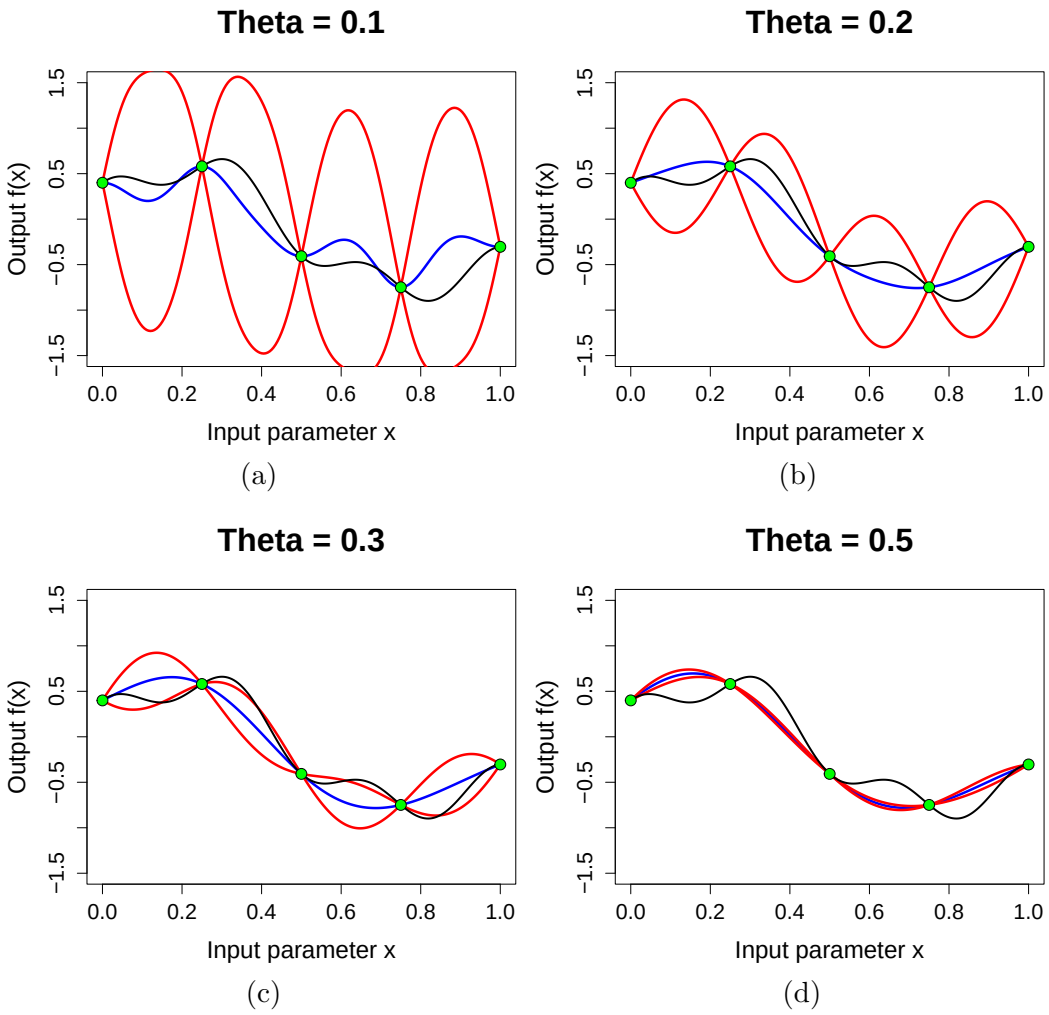


Figure 2.5: The emulator expectation and prediction interval, with $\sigma = 0.5$, $\mathrm{E}[f(x)] = 0$, and correlation length, $\theta =$ (a) 0.1, (b), 0.2, (c) 0.3 and, (d) 0.5.

adjusted variance, this is not always better.

Chapter 3

Emulation in Two Dimensions

3.1 Generalising from One Dimension to Two Dimensions

Emulation can be expanded to work in more than one dimension. This requires some of the priors to be changed so that they can take into account the multiple dimensions being used. These generalisations are based on ideas from [8].

3.1.1 Specifying Prior Beliefs

Whilst both the prior expectation, $\mathrm{E}[f(x)]$, and prior variance, $\mathrm{Var}[f(x)]$, can be kept the same when generalising to multiple dimensions, some adjustments need to be made to the covariance structure.

The squared exponential structure, which was used for emulation in one dimension, can be generalised to use the full Euclidean distance, between pairs of vector input points. This leads to the new covariance structure,

$$\mathrm{Cov}[f(x), f(x')] = \sigma^2 \exp -\frac{||x - x'||^2}{\theta^2}, \tag{3.1}$$

being used.

3.2 Example in 2D

This generalisation to two dimensions can be shown using the following example function:

$$f(x) = \sin(3\pi x_1) + \frac{1}{20}\cos(2\pi x_2). \tag{3.2}$$

The function will be evaluated at 9 points in a grid design, with x_1 and x_2 both taking the values $\frac{1}{6}, \frac{3}{6}$ and $\frac{5}{6}$. The prior expectation will be set to 0, the prior variance, $\sigma^2 = 1$, and the correlation coefficient, $\theta = 0.35$.

Figure 3.1 shows the prediction of the function, made by the emulator, compared to the true function, $f(x)$. It can be seen that, whilst the emulator models the function well for

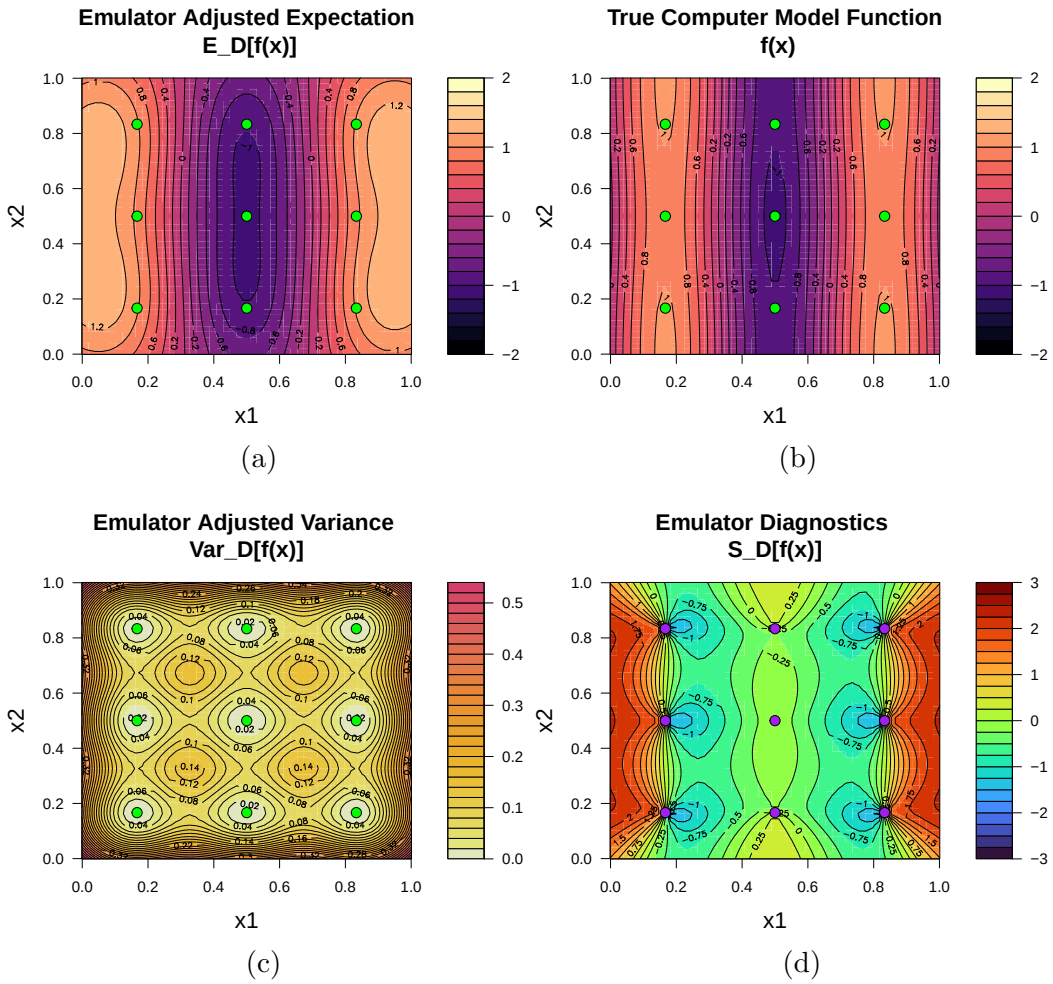


Figure 3.1: (a) The emulator adjusted expectation, (b) the true known function, $f(x)$, (c) the emulator adjusted variance, and (d) the emulator diagnostics.

x_1 values between 0.3 and 0.7, when x_1 gets closer to 0 or 1, the behavior of the function is not captured as well, especially the decreasing behaviour near the boundaries. This can be explained partially from the adjusted variance graph which shows higher variances at the boundaries of the emulator.

The emulator diagnostics for a 2D emulator can be calculated in the same way as for a 1D emulator, using (2.4). The emulator is performing well as all values are between -3 and 3, meaning that $f(x)$ is within the 3σ prediction interval, however at the boundary values for x_1, the emulator diagnostics give values above 2, showing how the model built by the emulator is not performing as well in these areas, compared to the rest of the model.

3.3 Picking Design Points

As seen in Figure 3.1, using a grid design for picking the points at which to evaluate the true function at does not always produce the best performing emulator. There are two characteristics of the example function which mean that a grid design for picking evaluation points does not perform particularly well.

The first challenge for emulators is modelling the behaviour of the function close to the boundaries. In the example, the emulator is not able to capture the decreasing behaviour of the function when x_1 gets close to the boundaries, as shown by the high emulator diagnostic values. This problem could be solved by moving the points of the gird closer to the boundaries of the function, however, this would mean that the points would not be able to influence the prediction made by the emulator as much, as more of the area each point influences will be outside the area of the function we are interested in.

The second challenge occurs when the true function, $f(x)$, being modelled by the emulator displays periodic behaviour, especially when the gaps between the points in the grid are a similar distance apart to the period of the function. In this example, this means that whilst the periodic behaviour of x_1 is easily captured by the emulator, the variation in the behaviour of x_2 is not modelled well.

3.3.1 Latin Hypercube Designs

One way to solve these problems is to use a Latin Hypercube Design when choosing the points at which to evaluate the function at[8]. If we want to pick n points, this is done by splitting each of the m input dimensions into n equally sized intervals, and then placing the n runs so that for each input dimension, x_j, each interval has one run in it.

Once the intervals that contain a run are decided, either the run can be located in the centre of the interval, or the position inside the interval for each dimension can be decided by using a Uniform distribution.

3.3.1.1 Improving the Example

A Latin Hypercube Design can now be used on the example function, (3.2), with $n = 9$. Once the intervals for the 9 points have been decided, we want to make sure that the picked points are spread out to make the emulator more efficient, as, for points close together the areas that they provide information for will overlap more. This can be done by iteratively picking two of the decided points and seeing if, by swapping the interval they are located in for one of the dimensions, this will increase the minimum distance between any two points

in the design. If this occurs, then this change is chosen. This is a fast algorithm which can be repeated hundreds of times, in a very short amount of time.

Once the intervals have been picked, the position of each point, with respect to the centre of its interval is decided by adding random independent and identically distributed uniform draws, $u \sim U[-1/(2l), 1/(2l)]$, where l is the width of the interval [6].

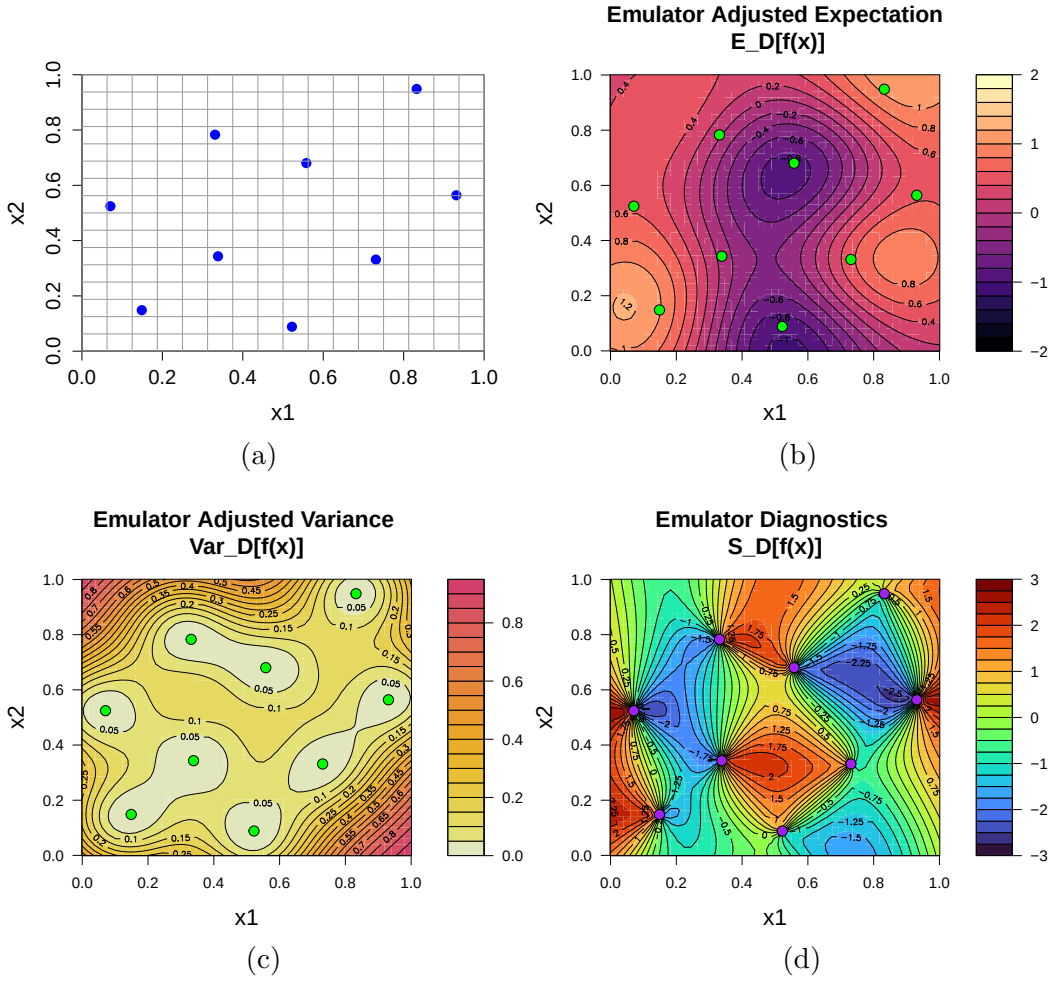


Figure 3.2: (a) The Latin Hypercube Design points chosen, (b) the emulator adjusted expectation, (c) the emulator adjusted variance, and (d) the emulator diagnostics.

Figure 3.2 shows the location of the 9 points chosen for the Latin Hypercube Design, and the performance of the emulator using these points. The emulator adjusted expectation, $\mathrm{E}_D[f(x)]$, does not show as much cyclic behaviour as in Figure 3.1, and is also able to show the lower values obtained at the boundaries of the x_1. However, the emulator prediction for the function is not as similar to the true function compared to the prediction using the grid design, especially in the top left and bottom right areas of the function, and this can be seen from the high values in the emulator adjusted variance. However, looking at the

emulator diagnostics, the large values near the boundaries of x_1 are no longer present, even in these more uncertain areas. This shows the improved performance that choosing the points at which to evaluate using this method provides.

3.4 Changing Covariance Structure

There are other ways that the emulator can also be improved, and one of these is by changing the prior covariance structure. Our choice describes the prior beliefs we have about the differentiability of the function we are trying to model [10].

3.4.1 Squared Exponential Covariance Function

The covariance function we have used in the previous examples is the squared exponential covariance function which takes the form:

$$\text{Cov}[f(x), f(x')] = \sigma^2 \exp -\frac{||x - x'||^2}{\theta^2}. \tag{3.3}$$

This performs well if we think that $f(x)$ is smooth, and therefore infinitely differentiable. However if it is used on a function that is not infinitely differentiable, it can be too smooth and lead to poor performing emulator diagnostics.

3.4.2 Matérn Covariance Function

This covariance function works well if we believe that k derivatives of $f(x)$ exist, and takes the form:

$$\text{Cov}[f(x), f(x')] = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(2\sqrt{\nu}\frac{||x - x'||}{\theta}\right)^{\nu} K_\nu \left(2\sqrt{\nu}\frac{||x - x'||}{\theta}\right), \tag{3.4}$$

where K_ν is a modified Bessel function of the third kind, and ν rounded down to the next integer gives the k, the number of derivatives that exist.

When $\nu \to \infty$, the Matérn covariance function will tend to the Squared Exponential covariance (3.3). When ν is a half integer, it will simplify to an exponential multiplied by a polynomial, with the most common versions being $\nu = 3/2$ and $\nu = 5/2$, for when we think $f(x)$ is differentiable 1 or 2 times respectively.

3.4.3 Exponential Covariance Function

When we do not believe that $f(x)$ is differentiable, but is still continuous, the Exponential covariance function can be used:

$$\text{Cov}[f(x), f(x')] = \sigma^2 \exp -\frac{||x - x'||}{\theta}. \tag{3.5}$$

This is another special case of the Matérn covariance function, where $\nu = 1/2$ and θ is rescaled to $\sqrt{2}\theta$.

3.5 Changing Correlation Lengths

Another way to improve the emulator is to use different correlation lengths, θ_i, for each input dimension, x_i. In the example function, (3.2), x_1 causes more dramatic changes in the behaviour of the function, compared to x_2. This can be modelled by setting $\theta_1 = 0.25$ and $\theta_2 = 1$, while still keeping $\sigma = 1$. Using the grid design for picking known runs allows us to more easily see the affect of the different values of θ_i.

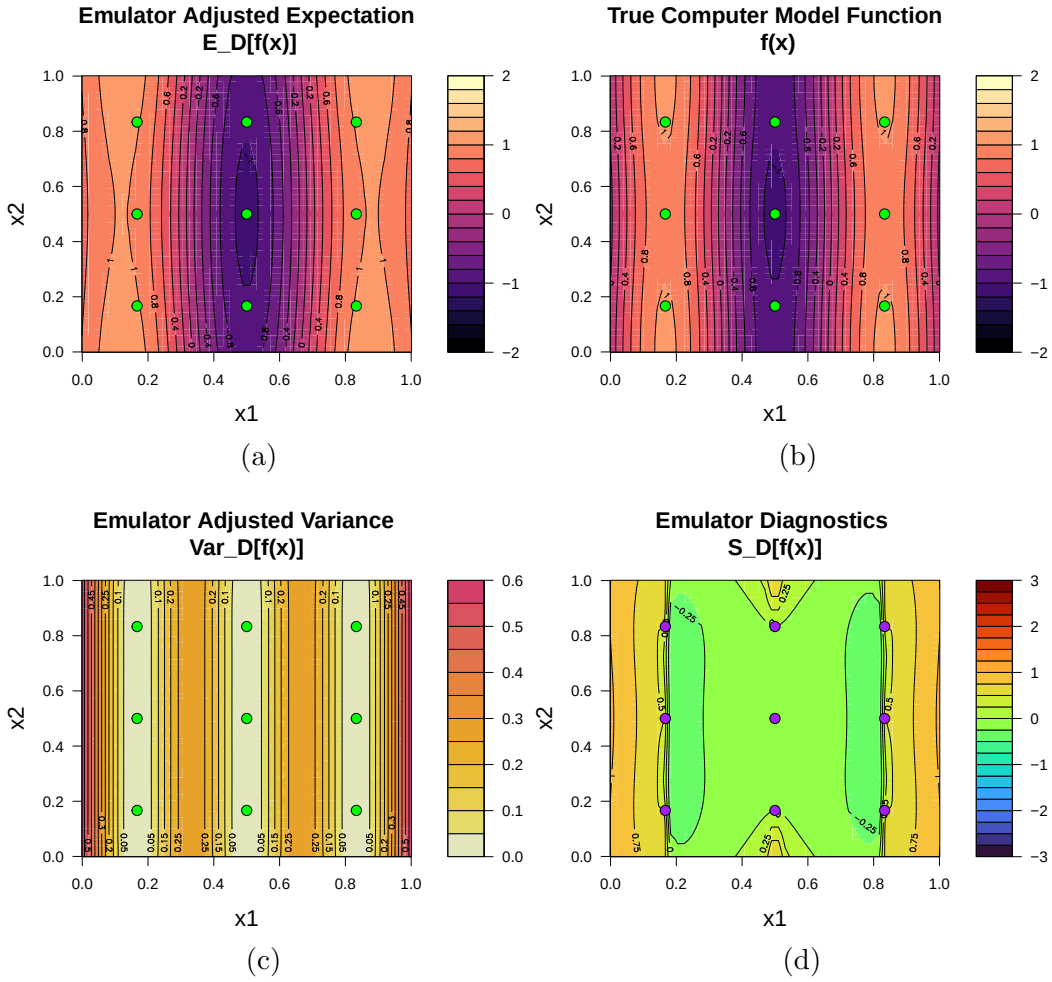


Figure 3.3: (a) The emulator adjusted expectation, (b) the true known function, $f(x)$, (c) the emulator adjusted variance, and (d) the emulator diagnostics, when using $\theta_1 = 0.25$ and $\theta_2 = 1$.

Figure 3.3 shows how using different values of θ for each dimension is able to improve the performance of the emulator, even when a grid design is used. Looking at the adjusted variance shows how the area that is affected by each known point is no longer circular. This is due to the higher value of θ_2 meaning that the area affected by the known points

in the x_2 dimension is bigger than in the x_1 dimension, leading to a striped pattern for the adjusted variance. As the function varies a lot more in the x_1 dimension, this allows the emulator to perform better and this is shown by the decreased emulator diagnostics throughout the whole model. The improvement in the emulator in Figure 3.3, compared to the original emulator with a grid design, shown in Figure 3.1 can also be seen in the adjusted expectation graphs, as the emulator with different correlation lengths for the dimensions is able to show the decreasing behaviour of the function at the boundaries of x_1, which was not modelled using the original emulator.

Chapter 4

History Matching

One use of emulators created for complex physical systems is history matching. Using the computer model, $f(x)$ and observed data z of the true system, y, we can try to learn about the inputs, x, which can produce these results. There are two main questions that can be answered. Firstly, are there input parameters, x, that lead to acceptable matches between the emulator output, $f(x)$, and observed data, z? Secondly, what is the set, $\mathcal{X}$, that contains all of the possible input parameter settings?

4.1 Sources of Uncertainty

There are two main sources of uncertainty that need to be taken into account before history matching can be performed [9]. This is due to the complex model, $f(x)$, only mimicking the real system, y.

4.1.1 Model Discrepancy

The first source of uncertainty is the model discrepancy, represented by ϵ. Even if we picked the best input, $x = x^*$, for our model, $f(x)$, it would not be precisely in agreement with the true system, y, due to the simplifications and approximations of the model. This discrepancy between the model and the real system can be represented by:

$$y = f(x^*) + \epsilon, \tag{4.1}$$

where ϵ is assumed to be independent of $f(x^*)$. We can use the second order statements, $\mathrm{E}[\epsilon]$, and $\mathrm{Var}[\epsilon]$, to express our prior beliefs about ϵ, in Bayes Linear analysis.

4.1.2 Observations of the Real System

The second source of uncertainty comes from the imperfect observations of the real system, y, represented by z. These observations are often performed with some observational error,

e, and the relationship between these observations, and the real system, can be represented by:

$$z = y + e. \tag{4.2}$$

The prior beliefs about e can be represented by $\mathrm{E}[e]$, and $\mathrm{Var}[e]$, based on knowledge about the observation process.

4.2 Implausibility Measures

Implausibility measures can be used during history matching to identify parts of the input space, $\mathcal{X}_0$, that need further investigation. For a univariate output, $f(x)$, we want to know, for an unexplored input parameter, x, how far would the emulator's expected value for the function output have to be from the observed value z, before we could deem it unlikely for $f(x)$ to give an acceptable match, if we were to evaluate the function at this value of x.

The square of the implausibility measure, $I(x)$, is given by [5]:

$$\begin{aligned} I^2(x) &= \frac{(\mathrm{E}_D[f(x)] - z)^2}{\mathrm{Var}(\mathrm{E}_D[f(x)] - z)}, \\ &= \frac{(\mathrm{E}_D[f(x)] - z)^2}{\mathrm{Var}_D[f(x)] + \mathrm{Var}[\epsilon] + \mathrm{Var}[e]}. \end{aligned} \tag{4.3}$$

The numerator of (4.3) gives the difference between the emulator expectation, $\mathrm{E}_D[f(x)]$, and the observation, z. The denominator standardises this by the uncertainties, obtained by viewing z as uncertain, but D as known.

A large value of $I(x)$ for a given x occurs when the emulator expectation is far from the observed data z, even when considering the uncertainties produced in the model. This would mean that we would be unlikely to obtain a acceptable match between $f(x)$ and z, if we ran the model there. We can discard x, the input, from the parameter search if $I(x) > c$, for some cutoff c, for example setting $c = 3$. We then refer to this input, x being implausible.

However, a small value for $I(x)$, can occur for two different reasons. Either the emulator expectation, $f(x)$, is close to the observed data, z, or the uncertainties in the denominator of the implausibility measure are large. We do not refer to these inputs as being plausible, as after performing more runs to reduce the uncertainty of the emulator, we may the rule this value of x out. Instead we refer to these low implausibility outputs as being non-implausible.

These values are therefore good candidates for further runs as they are likely to either produce acceptable matches, or may reduce the emulator variance and therefore be highly informative.

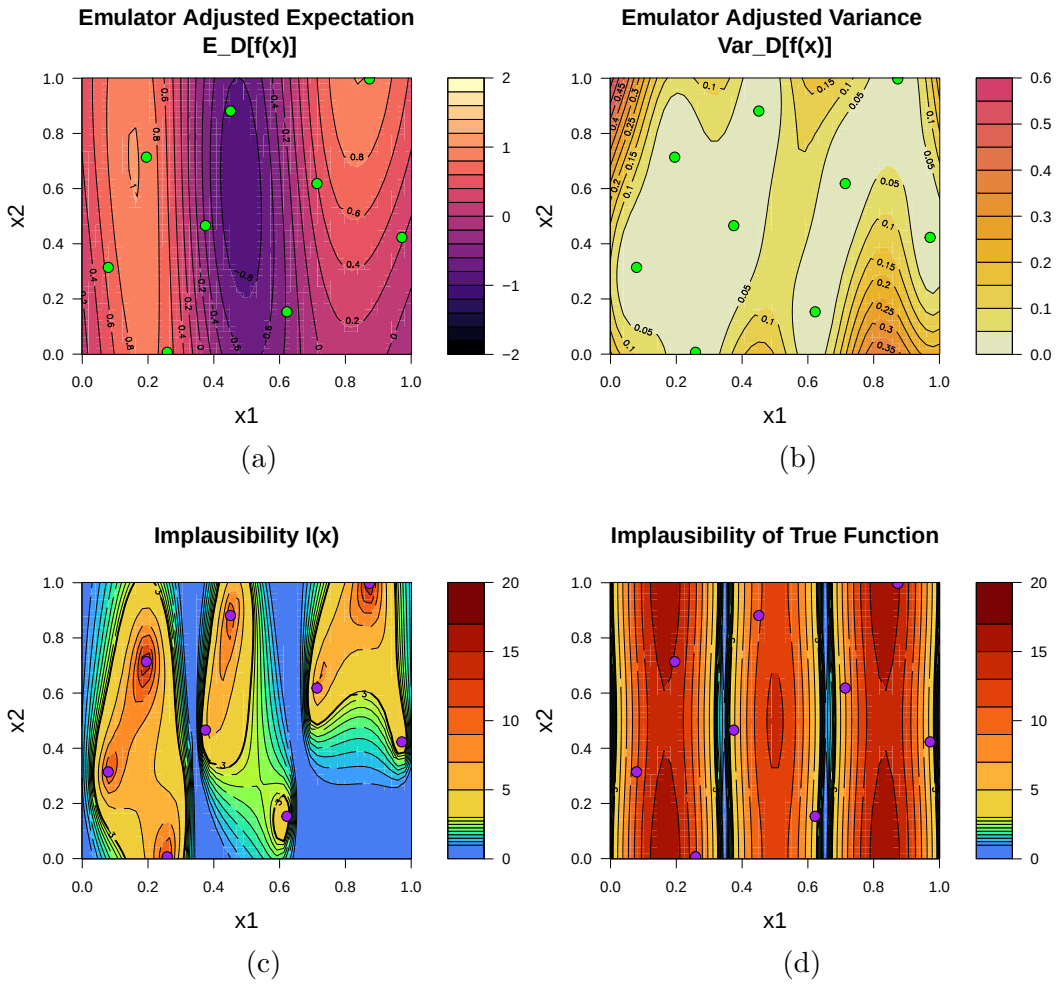


Figure 4.1: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the implausibility, $I(x)$ of the emulator for $z = -0.1$, and (d) the implausibility for the true model, for $z = -0.1$, with cutoff, $c = 3$.

4.3 Example

The process of history matching can be demonstrated on (3.2), the function that was modelled using the emulator in Section 3.2. Both the model uncertainty variances, $\text{Var}[e]$ and $\text{Var}[\epsilon]$ will be set to 0.05^2, and the observed data, z will be -0.1. Nine evaluated runs will be used, arranged in a Latin Hypercube design.

Figure 4.1 shows the implausibility, $I(x)$, of the emulator for $z = -0.1$. As we know the true function, we can compare this to the implausibility of the true function,

$$I^2_{true}(x) = \frac{(f(x) - z)^2}{\text{Var}[\epsilon] + \text{Var}[e]}. \tag{4.4}$$

This cannot usually be done as we normally do not know the true form of the model, however it allows us to see how the difference between the implausibility of the emulator, compared

to the most accurate implausibility that can be achieved. Looking at the implausibility from the emulator, it is possible to see the 3 large strips of implausible values, similar to the true implausibility, however, there are still many values that are currently non-implausible, which, after placing more runs would be ruled to be implausible.

As evaluating runs is often expensive, and there is normally a limit to the number that can be performed. As we are not normally interested in the behaviour of the whole function, but instead what inputs, x, allow for the observed output, z, then the evaluation of two waves of runs could improve the emulator and the results it produces. By calculating the implausibility of the function after the first wave of runs, we can then be better informed about where to place the runs in the second wave to obtain the information we want to learn.

4.4 Iterative History Matching

Iterative history matching allows us to perform our allocated number of run evaluations within two, or more, waves [5]. Using measures such as implausibility between these waves would then allow us to gain more information about where to place future runs. This would allow us to learn as much information about the function we are modelling as possible, in the areas that we are most interested in.

4.4.1 Picking Points for Wave k

Whilst using a Latin Hypercube design for the first wave of runs is seen to perform well, as this means that the points are well spread out while avoiding problems due to periodicity, there may be different factors that need to be considered when placing further waves of points. As we will have knowledge of some of the implausible regions of the function, we can use this information when placing further points [1].

4.4.1.1 Latin Hypercube for Wave 2

The simplest way to pick runs for wave 2 is to again generate a Latin Hypercube design for the points, however, after this is done, any of the suggested runs that are located in the implausible region are discarded, and not evaluated at. As we do not want wave 2 runs to be very close to runs already performed in wave 1, due to them not providing us with much new information, wave two runs within a certain distance of a wave 1 run can also be discarded.

Using Figure 4.1 as the first wave of runs, the second wave can be picked, by creating a Latin Hypercube design with 6 points, and then discarding the three points that are either located in the implausible region, or too close to already known runs.

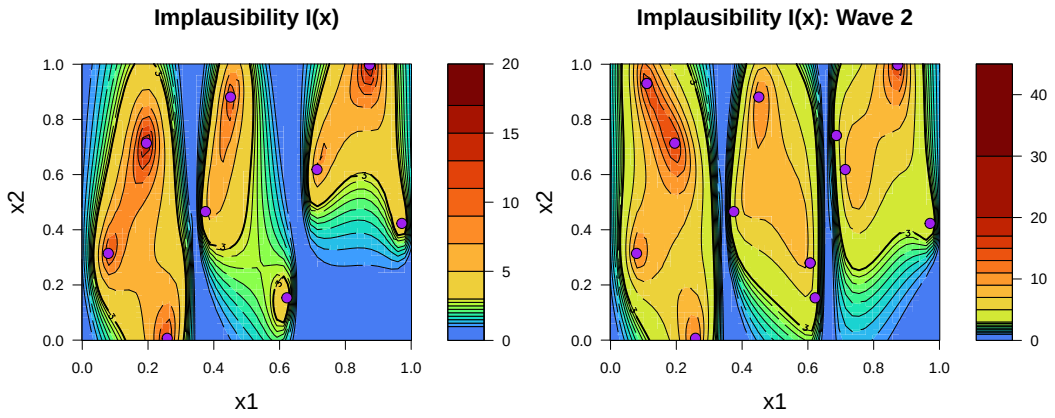


Figure 4.2: The implausibility of the emulator for $z = -0.1$ after (a) nine points forming wave one, (b) adding three more points in wave 2, using a Latin Hypercube Design.

Figure 4.2 shows how, after adding the second wave of points, the non-implausible region has become smaller, which now means that there are fewer possible output values to consider which could match the observed value, z. One problem with this method for choosing the runs for wave 2 is that there is still a large amount of randomness in both how many additional runs are chosen, and where these runs are placed. This leads to some areas being far away from any known runs, leading to a high adjusted variance, and therefore small implausibility measure. To avoid this, different techniques could be used to decide the position of the runs in the second wave.

4.4.1.2 Maximising Minimum Distance to a Known Run

To make sure that the second wave of runs explore the space that does not have any first wave runs nearby, and therefore has the potential to provide the most information with a run, runs could be picked to be far from the known runs. This can be done by looking at each possible input value that is currently non-implausible, and finding its distance to its nearest known run. The point that has the highest distance to its nearest known run can be picked to be evaluated at in wave 2. Points located directly on the edge of the function space are not considered as these would provide a lot less new information than a run even slightly away from the boundary. Once we have evaluated this new run, the new implausibility measures can be calculated, and this can then be repeated to pick more new values for us to evaluate runs at.

Using the same wave 1 runs as in Figure 4.1, three more runs can be iteratively added using this technique.

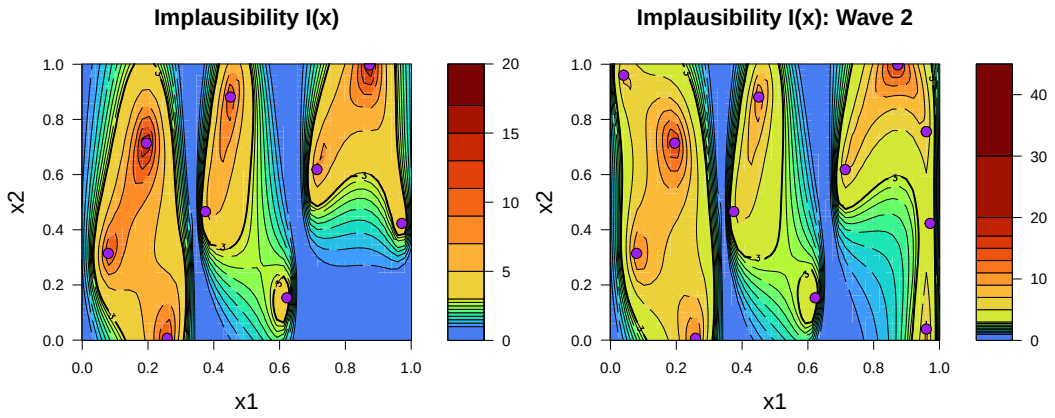


Figure 4.3: The implausibility of the emulator for $z = -0.1$ after (a) nine points forming wave one, (b) adding three more points in wave 2, by maximising the minimum distance to a known run.

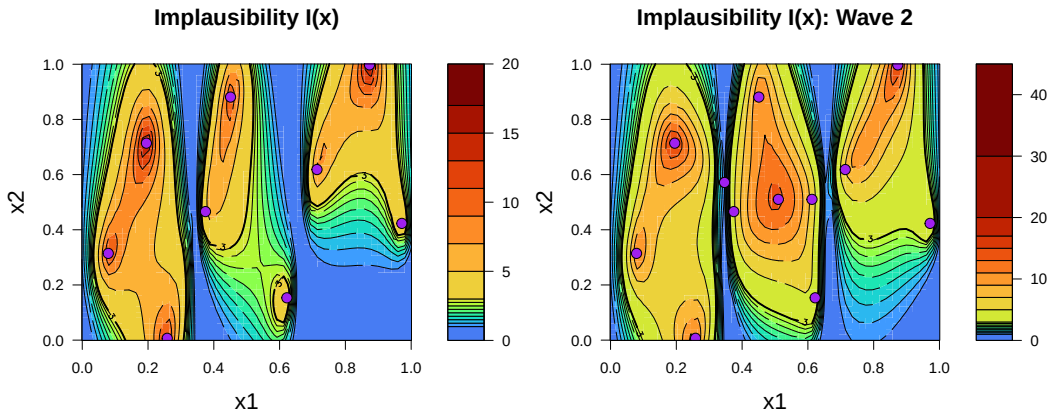
Figure 4.3 shows the advantage of using this method compared to using a Latin Hypercube Design shown in Figure 4.2. This algorithm for adding new runs to evaluate at does better at ensuring that these runs are more spread out, especially as the three new runs are added iteratively, instead of all at once. This can be seen by the size of the rightmost implausible region being increased more than it was using the Latin Hypercube technique which does not cause the region to reach the $x_2 = 0$ boundary. There is however, still a large non-implausible area which does not have any runs in it. Adding another one or two more runs should place a run in this area which would solve this problem. Another solution would be to expand the area around the boundaries of the space that the function is being evaluated on, in which the wave 2 runs are not allowed to be placed.

4.4.1.3 Minimising Distance to Furthest Part of the Non-Implausible Region

Another method which could work well is by placing the wave 2 runs at the non-implausible points which have the shortest distance to the non-implausible point the furthest away from them. This would work best if there is a main non-implausible area in the middle of the function, but the other methods are choosing to prioritise placing the extra runs in areas near the boundaries or corners of the function. This could possibly be due to the original runs in the Latin Hypercube Design not being close to some of the corners.

The affects of this can also be seen by adding 3 more runs iteratively to Figure 4.1.

Whilst in some scenarios this may work well, Figure 4.4 shows that this does not perform as well as either of the other techniques on this function. This is because, as there are non-implausible regions at the boundaries for both $x_1 = 0$ and $x_1 = 1$, the points that are



(a) (b)

Figure 4.4: The implausibility of the emulator for $z = -0.1$ after (a) nine points forming wave one, (b) adding three more points in wave 2, by picking runs to minimise the distance to the furthest non-implausible region of the function to the chosen run.

picked will always be as close to $x_1 = 0.5$ as long as there is still a non-implausible region there. This means that points chosen for the extra runs will not favour the bigger regions of non-implausibility, which are closer to the boundaries.

Chapter 5

The Lotka-Volterra Model

5.1 The Model

One real life example where emulation and history matching could be used, is in a Predator-Prey model. An example of one of these complex models is the Lotka-Volterra Predator-Prey model. The simplest version of this is used to describe the dynamics of two species, a prey (represented by g_1), and a predator (represented by g_2), across a time series, labelled by time t.

The system follows the deterministic differential equations [3]:

$$\frac{dg_1}{dt} = x_1 g_1 - x_2 g_1 g_2, \tag{5.1}$$

$$\frac{dg_2}{dt} = x_2 g_1 g_2 - x_3 g_2. \tag{5.2}$$

The system involves three inputs, $\mathbf{x} = (x_1, x_2, x_3)$, which determine the speed of reproduction of the prey, the predator-prey interaction rate, and the predator death rate, respectively. Recommended ranges for these input variables are $0.7 < x_1 < 1.5$, $4.4 \times 10^{-5} < x_2 < 7.5 \times 10^{-4}$, and $1.4 < x_3 < 2.5$.

5.1.1 Example of Behaviour

Setting the initial conditions of the system to be $(g_1(t = 0), g_2(t = 0)) = (2000, 800)$, and the input variables to $x1 = 1$, $x2 = 0.00044$, and $x3 = 1.8$, this model can then be run until $t = 10$.

The output of this run is shown in Figure 5.1. The behaviour of this run shows the lag between the peaks of the population of the prey and the predators. This is due to a large population of prey providing more food to the predators which then causes the population of predators to increase and prey to decrease. This then leads to there being less food for the predators to eat, meaning that their population decreases, allowing the population of the prey to increase again.

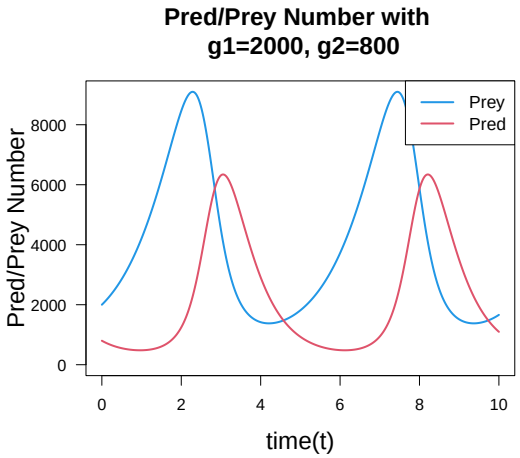


Figure 5.1: The population of Prey (blue), and Predators (red), from $t = 0$ until $t = 10$, with starting conditions $(g_1, g_2) = (2000, 800)$.

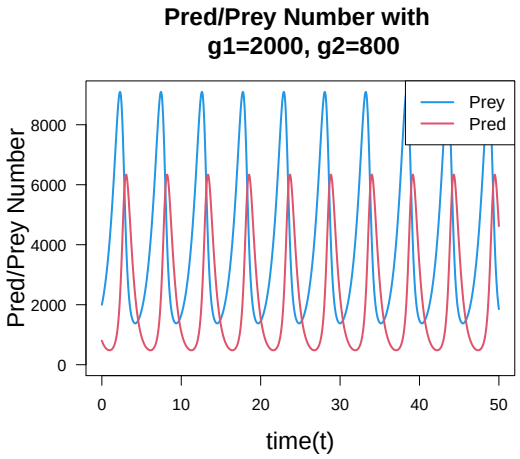


Figure 5.2: The population of Prey (blue), and Predators (red), from $t = 0$ until $t = 50$, with starting conditions $(g_1, g_2) = (2000, 800)$.

By increasing the amount of time the model is run for, by setting $t = 50$, it can be seen in Figure 5.2 that the behaviour of the populations of predators and prey is cyclic, and that the time between the peaks, and the height of them stays constant. This may not fully represent how the populations would behave in real life, as there may be other factors that affect the populations. However, using this simple version of a predator-prey model allows for it to be analytically solved meaning that results from emulation and history matching can be compared to the true model.

5.1.2 Challenges of Modelling

Compared to the two dimensional example from Section 3, there are additional things that need to be considered when performing emulation and history matching, on this model.

The first thing that needs to be considered is the role of time. There are two main ways that this time series can be modelled. The first option is to treat t as an input, and therefore the emulation will take place in four dimensional space. This will increase the size of the input space that the emulator has to perform in, and will therefore increase the amount of computation that has to be performed to build each emulator. This will also lead to more runs needing to be evaluated to achieve the same performance from the emulator. The other option is that t can be considered as a constant, and the time series will only be considered at a chosen point in time, instead of across the whole time series. This can then be repeated at multiple time points, and as the function is smooth, then the behaviour between the chosen time points can then be estimated.

Even if the function is evaluated at a constant time point, t, the model still has three input variables, therefore emulation will take place in three dimensions. Whilst the emulator structure for two dimensions can be generalised to more dimensions, one problem that occurs is the visualisation of the results. Whilst in two dimensions, the emulator diagnostics and implausibility measures can be shown in a 2D grid, in three dimensions this is harder to achieve.

There are two main techniques to model the implausibility in more than two dimensions, using pair plots [2]. The first is to show the minimised implausibility for each pair of input variables, over all the possible values of the other input variables. This is good at showing which pairs of input values still have the potential to be non-implausible. The other option is to use optical depth which involves, for each pair of input values, showing what percentage of the values for the other variables will produce non-implausible output values. This has the advantage of showing which of the non-implausible areas are more likely to produce the wanted output values, and therefore where the second wave runs should be placed.

Another option is that if some of the input variables have a larger affect on the behaviour of the model compared to the others, only these could be considered during the history matching, with the other variables set to their mid-value, or other chose value.

Finally, as the predator-prey model has multivariate output, because it models the populations of both the predators and prey, the emulator will have to produce models predicting the behaviour of each output variable separately, and this will also impact the history matching techniques used.

5.2 2D emulation

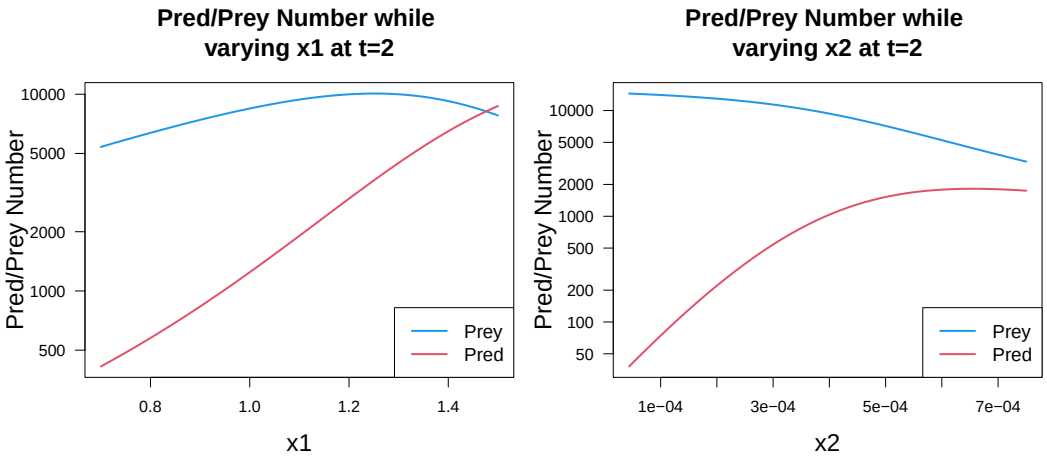
5.2.1 Variable Selection

In order to perform 2D emulation on this model, the input variables which we want to vary need to be chosen. This can be done by looking at the affect of individually varying each of these variables, while keeping the other two constant.

By plotting Figure 5.3 on a log scale, it can be seen that varying x_1 and x_2 has a bigger affect on the populations of the predators and prey compared to varying x_3 which causes the population of prey to exponentially increase and the population of predators to exponentially decrease. The affects of varying x_1 and x_2 produces more complicated behaviour of the populations. For example, once x_1 increases above 1.25, the population of the prey starts to decrease, whilst for lower values of x_1, this population is increasing. This suggests that these are the two variables we are most interested in when performing emulation and history matching.

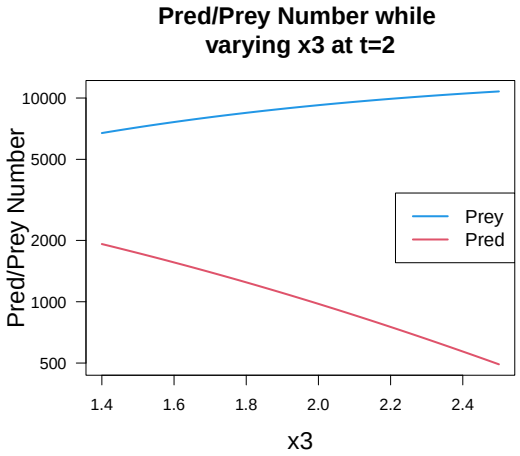
5.2.2 Emulation for x_1 and x_2

Emulation can be performed for x_1 and x_2, while keeping $x_3 = 1.8$, and for $t = 2$. Before doing this, the priors need to be defined. Firstly, the prior expectations can be set by looking at the behaviour of the populations in Figure 5.3, for varying x_1 and x_2. For the prior expectation for predators, 2000 is a suitable value as this is the average population when varying x_1, and around the population produced for higher values of x_2. For the prior expectation for the prey, 7000 is a suitable value as this is located halfway between 5000 and 10000 on a log scale, which is the range that the population is located in while varying both x_1 and x_2. A good value for θ is often a quarter of the range of values that the input parameter can take. For x_1, $\theta = 0.2$, and for x_2, $\theta = 0.00018$. For the variance, we want most of the function to lie within 3σ of the mean. By looking at the ranges of both populations in Figure 5.3, σ for the population of prey will be set to 2500, and σ for the predator population will be 1000.



(a)

(b)



(c)

Figure 5.3: The population of predators (red), and prey (blue), at $t = 2$, whilst varying (a) x_1, (b) x_2, and (c) x_3. Keeping the other two input variables constant at $x_1 = 1$, $x_2 = 0.00044$, and $x_3 = 1.8$.

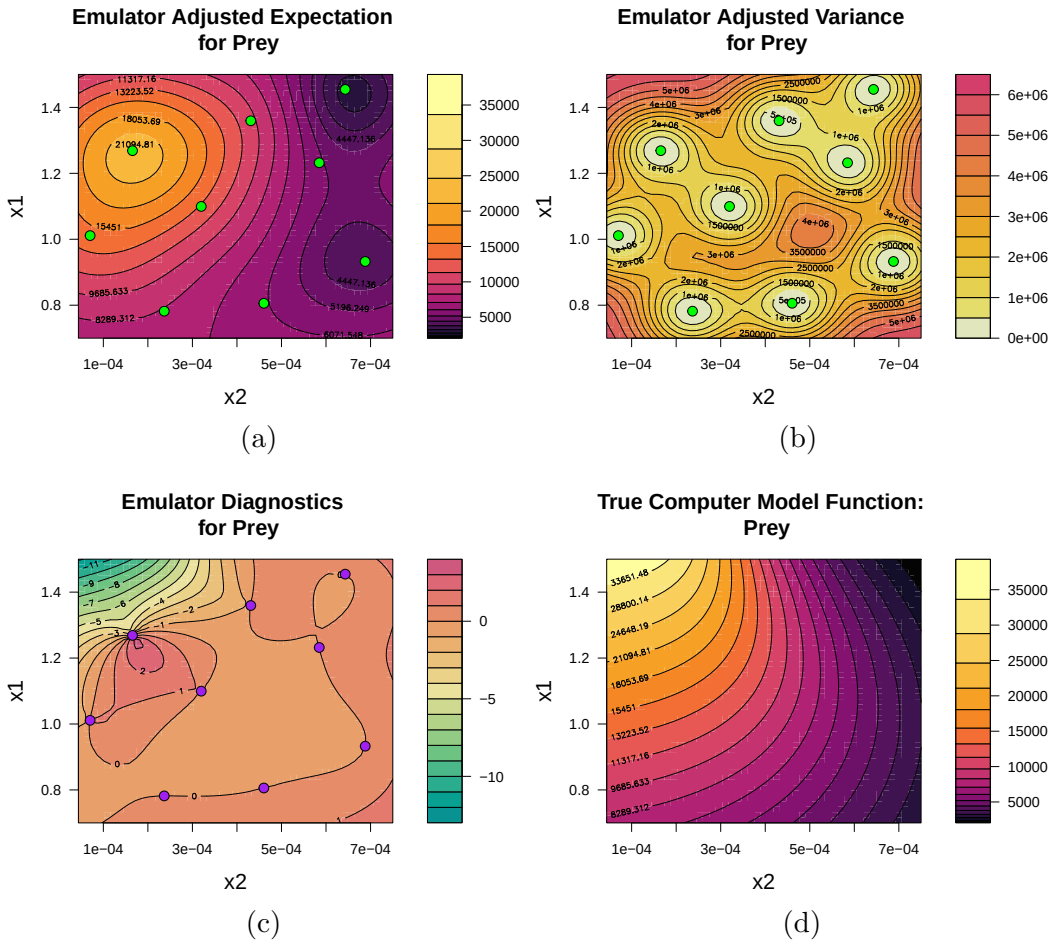


Figure 5.4: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the prey at $t = 2$.

A Latin Hypercube Design will be used to pick the 9 points at which to sample runs at, and the population will be displayed on a log scale due to the exponential behaviour of the model.

5.2.2.1 Emulation for Prey

Figure 5.4 shows the results of this emulation for the population of the prey. Whilst the emulation for the middle values of x_1 and x_2 performs well, with emulator diagnostics close to 0, the emulator struggles to capture the increasing behaviour of the population, especially for high values of x_1 and low values of x_2. This can be seen by the high emulator diagnostic values in this area. This may partially be caused by the steep increase in population due to the exponential form of the model. Due to the high prior values for σ caused by the high range of values that the population can take, the adjusted variance of the emulator quickly becomes large away from the observed runs.

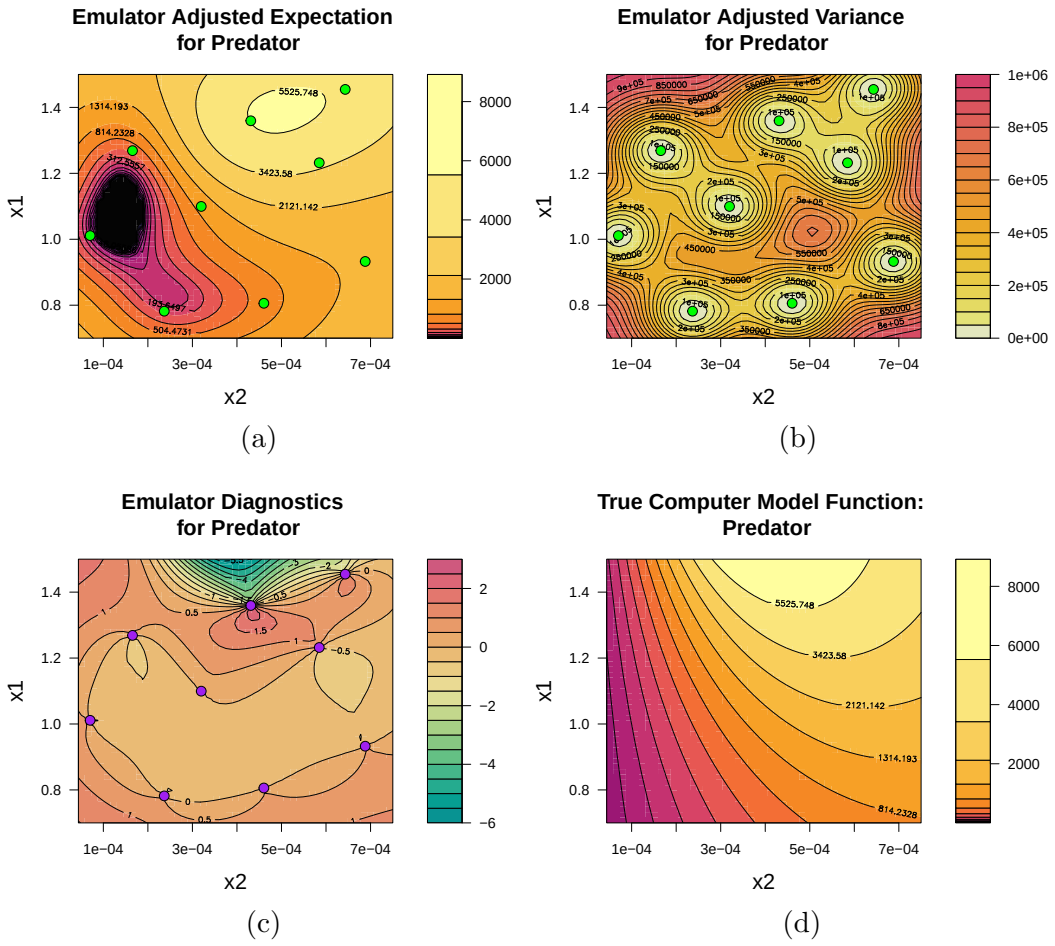


Figure 5.5: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the predators at $t = 2$.

5.2.2.2 Emulation for Predators

Figure 5.5 shows the results of emulation on the population of the predators. For the emulator adjusted expectation, $\mathrm{E}_D[f(x)]$, some of the values produced were negative, however, as this is measuring population, this is not possible, so these values have been set to 1. This can be seen by the black oval in adjusted expectation. Similar to the prey, the emulator adjusted variance increases rapidly away from points due to the high σ. There are high emulator diagnostics for higher values of x_1, possibly due to the rapid increasing of the predator population. The emulator also struggles to model the function for the lower populations, when x_2 is small, as the emulator predicts the function to decrease quicker than the true function does.

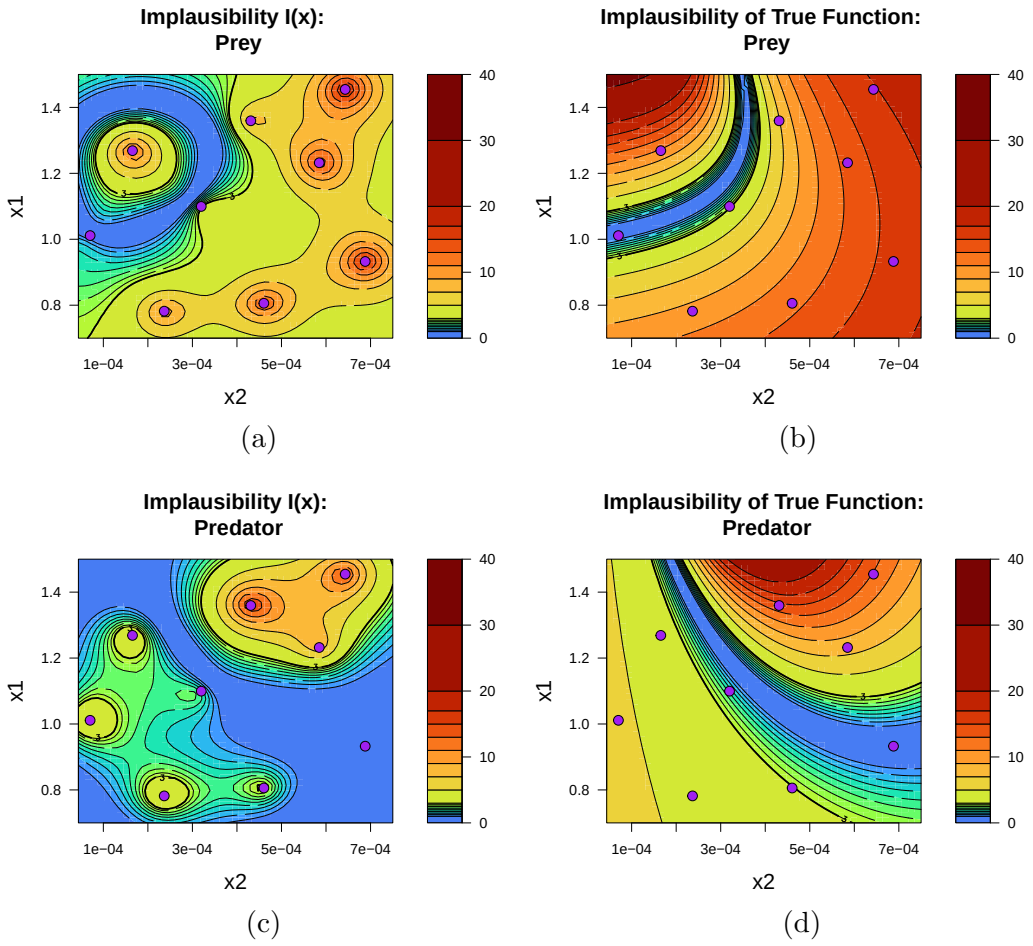


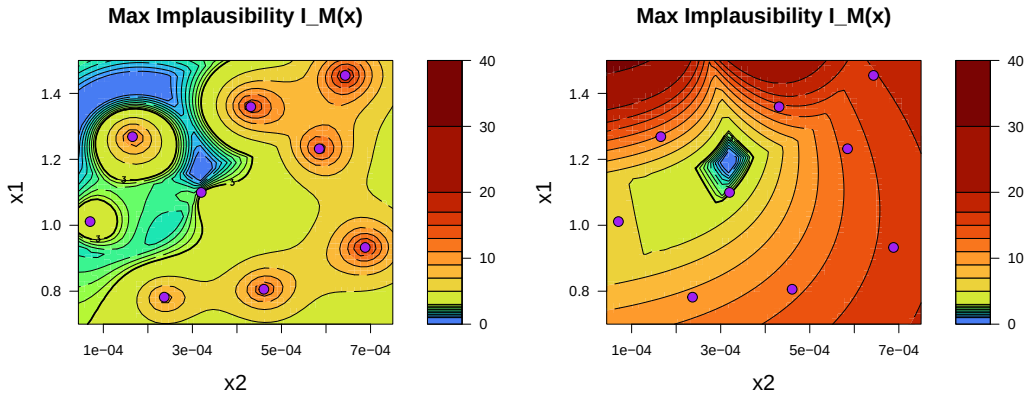
Figure 5.6: (a) The implausibility for the population of the prey from the emulator, and (b) the implausibility for the population of the prey from the true model, for $z = 15000$. (c) The implausibility for the population of the predators from the emulator, and (d) the implausibility for the population of the predators from the true model, for $z = 1500$.

5.3 History Matching

History matching can also be performed on both populations, using the techniques discussed in Section 4. For the population of the prey, the observed value, $z = 15000$. The two model uncertainty variances, $\text{Var}[e]$ and $\text{Var}[\epsilon]$, will be set to 500^2.

For the population of the predators, the observed value, $z = 1500$, and the two uncertainty variances will be set to 200^2, as the prior variance for the predators is also lower.

For both the prey and the predators, Figure 5.6 shows that the non-implausible region for both outputs is much bigger than the non-implausible regions for the true functions used to model the populations. This could be caused by the high prior variance defined for each of the models, or could also suggest that more runs need to be evaluated to be able to have



(a) (b)

Figure 5.7: The maximised implausibility, for (a) the emulators, and (b) the true functions.

a good idea of the behaviour of the functions and the shape of the non-implausible regions. A second wave of runs could be performed, however, different techniques may have to be used to best incorporate the information we have gained from both the outputs.

5.3.1 Multivariate Output Techniques

As we have two different outputs, when history matching, we want to find points that match the observed values for both of the outputs. In order to identify these areas, we need to be able to combine the information we have from both of the implausibility plots.

We can combine these individual implausibilities, $I_i(x)$, together by calculating the maximised implausibility [2]:

$$I_M(x) = \max_i I_i(x). \tag{5.3}$$

By then imposing the constraint $I_M(x) < c_M$, where $c_M = 3$ is the implausibility cutoff, means that all the implausibilities, $I_i(x)$, must be lower than this value to be in the non-implausible region.

Figure 5.7 shows the performance of maximised implausibility on both the emulator outputs, and the true function. This decreases the size of the non-implausible region, meaning that second wave runs can be better picked so that they will provide information for narrowing the shape of the non-implausible region for both outputs.

Whilst the non-implausible region for the maximised implausibility of the emulator outputs does contain the true non-implausible region, it also contains a lot of the area for low values of x_2. This could be due to there not being many runs from the original wave of runs in this area, or due to the high emulator adjusted variance caused by the high

prior variance, however, it suggests that the performance of the emulator could be further improved in this region.

Chapter 6

Known Boundary Emulation

6.1 Benefits of Technique

Even in complicated models, sometimes setting a variable to a given value will simplify its equations, possibly to a constant. Therefore, one way to improve the performance of an emulator, without having to evaluate more runs, is to add the values on a known boundary which can be easily calculated. The methods used in this chapter to perform this technique are based on material [7].

6.2 Updating Second Order Beliefs

If the computer model is able to be solved analytically on a lower dimensional boundary, $\mathcal{K}$, we can evaluate $\{f(x) : x \in \mathcal{K}\}$, for multiple points on $\mathcal{K}$ in a short amount of time. These can be used to supplement the standard emulator evaluations to produce a emulator which respects the behaviour of $f(x)$ along the boundary, $\mathcal{K}$. By evaluating a large, but finite number, m, of points on $\mathcal{K}$, we can then use the standard Bayes linear update to analyse the expected behaviour of the model.

Plugging the m runs into the Bayes linear update equation would be unfeasible because of the $m \times m$ matrix inversion that would need to be performed on the prior variance matrix.

Let K be the length m vector of model evaluations, which can easily be analytically calculated. In order to capture the model behaviour on $\mathcal{K}$, we evaluate $f(x)$ at a large, but finite, number of points m, on the boundary. We can denote these as $y^{(1)}, \ldots, y^{(m)}$. We can also evaluate the perpendicular projection of x, the point of interest, onto $\mathcal{K}$, denoted as x^K. The collection of boundary evaluations, K, can then be a $m+1$ column vector, $K = (f(x^K), f(y^{(1)}), \ldots, f(y^{(m)})$. The calculation for the emulator adjusted expectation, $\mathrm{E}_K[f(x)]$, is infeasible to calculate due to the $\mathrm{Var}[K]^{-1}$ term.

However, if we evaluate this at the point x^K which lies on $\mathcal{K}$, because we can easily evaluate $f(x^K)$, we can then find that $[E]_K[f(x^K)] = f(x^K)$, and $\mathrm{Var}_K[f(x^K)] = 0$. Because

$f(x^K)$ is the first element of K, then [7]:

$$\begin{aligned} I_{(m+1)} &= \text{Var}[K]\text{Var}[K]^{-1}, \\ &= \begin{pmatrix} \text{Cov}[f(x^K), K] \\ \text{Cov}[f(y^{(1)}), K] \\ \vdots \\ \text{Cov}[f(y^{(m)}), K] \end{pmatrix} \text{Var}[K]^{-1}, \end{aligned} \tag{6.1}$$

with $I_{(m+1)}$ being an identity matrix of dimension $(m+1)$.

The first row of (6.1) gives $\text{Cov}[f(x^K), K]\text{Var}[K]^{-1} = (1, 0, \ldots, 0)$. We can then write $x = x^K + (a, 0, \ldots, 0)$, where a is a constant representing the perpendicular distance from x to the known boundary, $\mathcal{K}$. By defining $r_i(a)$ as the correlation structure of the model in dimension i between the point x and x^K, the covariance between a point x and the boundary conditions can be written as:

$$\text{Cov}[f(x), K] = r_1(a)\text{Cov}[f(x^K), K]. \tag{6.2}$$

This removes the need to explicitly calculate the inversion of the matrix $\text{Var}[K]$. This therefore allows an analytically solvable boundary to be added to the emulator without greatly increasing the amount of computation or time it takes, while increasing the amount of information about the behaviour of the function that is available.

We can then use this to form the emulator adjusted expectation and variance given K, a length m vector of model evaluations on the analytically solvable boundary, $\mathcal{K}$.

6.2.1 Update Equations for Adding a Known Boundary

Given the orthogonal projection, x^K, of a point x, onto the boundary $\mathcal{K}$ at distance a. With the correlation structure between x and x^K in dimension i being represented by $r_i(a)$, the Bayes Linear update equations for the known boundary, $\mathcal{K}$, are [7]:

$$\text{E}_K[f(x)] = \text{E}[f(x)] + r_1(a)(f(x^K) - \text{E}[f(x^K)]), \tag{6.3}$$

$$\text{Cov}_K[f(x), f(x')] = \sigma^2(r_1(a - a') - r_1(a)r_1(a'))r_{-1}(x^K - x'^K), \tag{6.4}$$

$$\text{Var}_K[f(x)] = \sigma^2(1 - r_1(a)^2). \tag{6.5}$$

6.2.1.1 Limiting Behaviour

One way to analyse the behaviour of this update is to look at the limiting behavior of the emulator for the expectation, variance and covariance.

As x moves towards $\mathcal{K}$, the emulator expectation should tend towards the known boundary function. On the other hand, as it becomes further away from the boundary, it should

tend towards the prior expectation:

$$\lim_{a\to 0} \mathrm{E}_K[f(x)] = f(x^K), \qquad \lim_{a\to\infty} \mathrm{E}_K[f(x)] = \mathrm{E}[f(x)]. \tag{6.6}$$

For the adjusted variance, when x moves towards the boundary, the emulator will tend towards the known boundary function with a variance of 0 as the true value on the boundary is known. Similarly to the expectation, as x gets further away from the boundary, and $a \to \infty$, the the variance will tend to the prior variance.

$$\lim_{a\to 0} \mathrm{Var}_K[f(x)] = 0, \qquad \lim_{a\to\infty} \mathrm{Var}_K[f(x)] = \mathrm{Var}[f(x)]. \tag{6.7}$$

Finally, for the covariance, when x and x' are far from the known boundary, and $a - a'$ is finite, the covariance should tend to its prior form, whereas, if either a or a' tends to zero, the covariance will also tend to zero:

$$\lim_{a\to 0} \mathrm{Cov}_K[f(x), f(x')] = 0, \qquad \lim_{a,a'\to\infty} \mathrm{Cov}_K[f(x), f(x')] = \mathrm{Cov}[f(x), f(x')]. \tag{6.8}$$

6.2.2 Updating by Model Evaluations

Once the known boundary, $\mathcal{K}$, has been emulated at, and we have calculated analytic expressions for $\mathrm{E}_K[f(x)]$,$\mathrm{Var}_K[f(x)]$ and $\mathrm{Cov}_K[f(x), f(x')]$, we may then want to add our expensive simulator evaluations into the model. To do this, we preform n expensive runs, represented by D, of the full model.

We then want to update the emulator by the union of the two sets of evaluations, D and K. This will give us $\mathrm{E}_{D\cup K}[f(x)]$,$\mathrm{Var}_{D\cup K}[f(x)]$ and $\mathrm{Cov}_{D\cup K}[f(x), f(x')]$, which can be calculated using a sequential Bayes linear update [7].

$$\begin{aligned}
\mathrm{E}_{D\cup K}[f(x)] &= \mathrm{E}_K[f(x)] + \mathrm{Cov}_K[f(x), D]\mathrm{Var}_K[D]^{-1}(D - \mathrm{E}_K[D]), && (6.9)\\
\mathrm{Var}_{D\cup K}[f(x)] &= \mathrm{Var}_K[f(x)] - \mathrm{Cov}_K[f(x), D]\mathrm{Var}_K[D]^{-1}\mathrm{Cov}_K[D, f(x)], && (6.10)\\
\mathrm{Cov}_{D\cup K}[f(x), f(x')] &= \mathrm{Cov}_K[f(x), f(x')] - \mathrm{Cov}_K[f(x), D]\mathrm{Var}_K[D]^{-1}\mathrm{Cov}_K[D, f(x)]. && (6.11)
\end{aligned}$$

As the number of evaluations, n is usually small due to the expense of evaluating runs in the full simulator, we do not need to worry about $\mathrm{Var}_K[D]^{-1}$ being intractable due to its size.

6.3 Adding a Known Boundary to the Lotka-Volterra Model

This technique of adding a known boundary could be used on the Lotka-Volterra model from Section 5.1. It would improve the emulation and the performance of history matching on the model, and would also help to mitigate against some of the problems that were seen when just using runs to build the emulator. For example, one problem that occurred was a decrease in performance of the emulator towards the boundaries of the function space. By adding a known boundary, this would improve the emulation at one of the boundaries, and would therefore mean that the runs to be evaluated could be placed closer to the other boundaries.

6.3.1 Adding a Boundary

The Lotka-Volterra model is well suited to the known boundary technique, as, if certain input variables are set to 0, the equations in the model are simplified and are able to be solved analytically and a lot quicker than most evaluations are able to be made. Despite it not being inside the recommended values for x_2, it may help the emulation to set $x_2 = 0$. This represents there being no interaction between the two populations of animals. This means that the prey will not be eaten by the predators, so their population will be able to exponentially increase, and as the predators will not have any food, their population will eventually die out.

Setting $x_2 = 0$ will result in the following simplified version of (5.2):

$$\frac{dg_1}{dt} = x_1 g_1, \tag{6.12}$$

$$\frac{dg_2}{dt} = -x_3 g_2. \tag{6.13}$$

Using the same prior parameters as in Section 5.2.2, along with $t = 2$, $x_3 = 1.8$ and $0.7 < x_1 < 1.5$ emulation can be performed. Compared to last time, however, $0 < x_2 < 7.5 \times 10^{-4}$, which allows us to evaluate a known boundary at $x_2 = 0$.

6.3.1.1 Emulation for Prey

Firstly, the performance of adding a known boundary at $x_2 = 0$ to the emulator for the population of the prey can be observed.

Comparing Figure 6.1 to the results of the emulation using a Latin Hypercube Design in Figure 5.5 shows the benefits of adding the information gained from this analytically solvable boundary into the emulator. By including the knowledge about the behaviour of the model at this boundary, it removes the high emulator diagnostics that the previous emulation produced for high values of x_1, while x_2 was close to 0. Whilst it is not able to

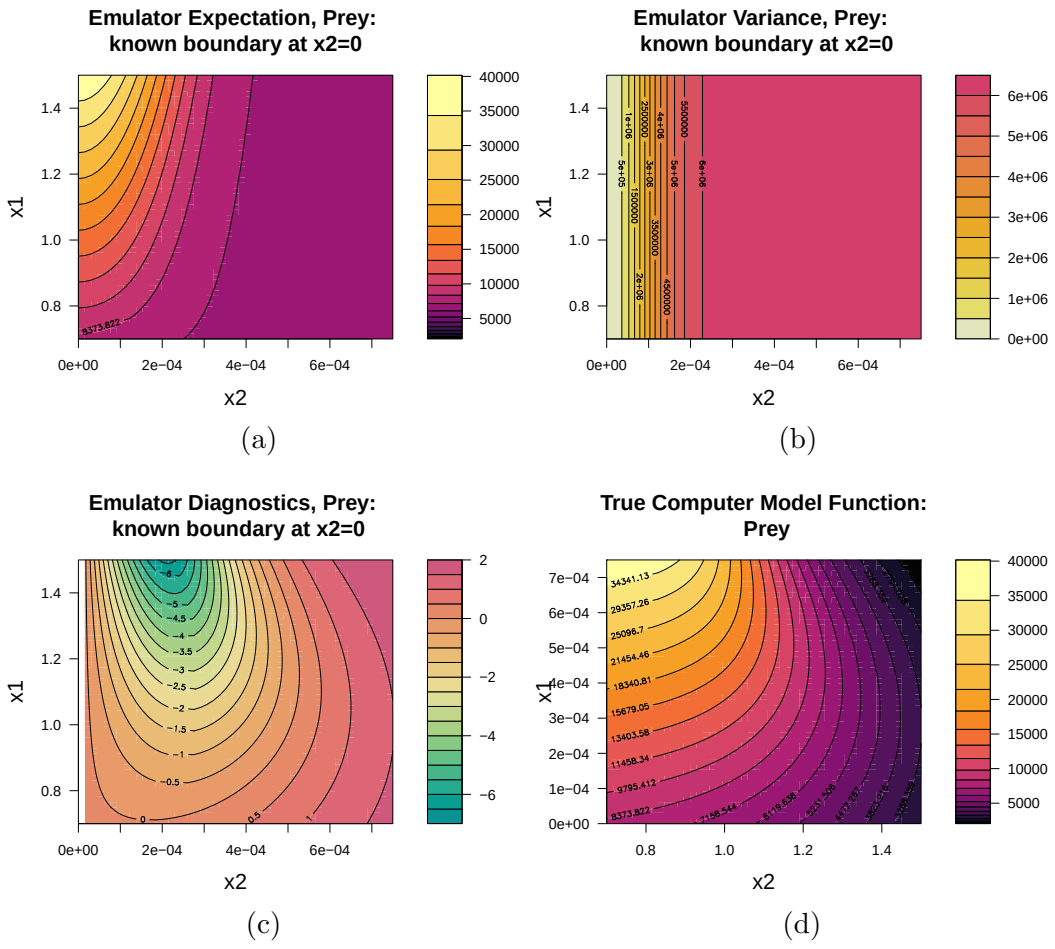


Figure 6.1: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the prey at $t = 2$, with a known boundary at $x_2 = 0$.

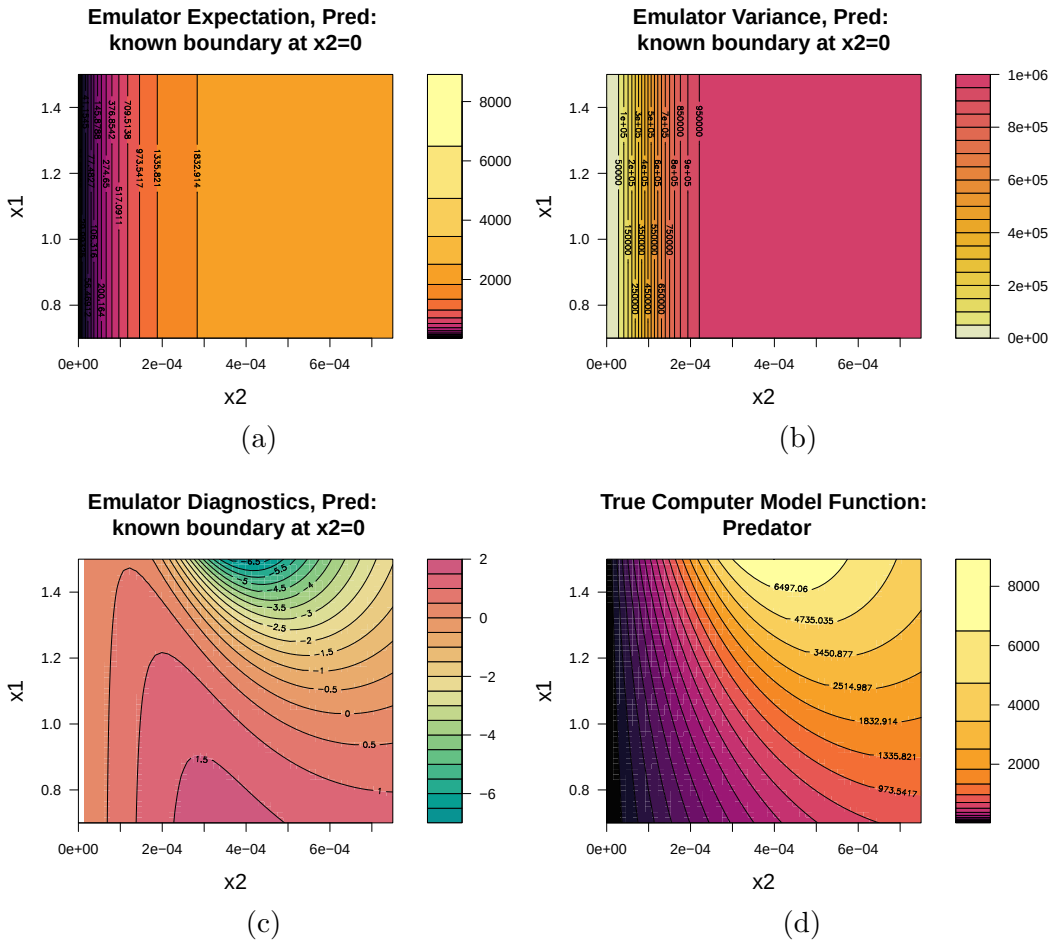


Figure 6.2: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the predators at $t = 2$, with a known boundary at $x_2 = 0$.

capture the behaviour of the model for higher values of x_2, as the emulator resorts back to the prior expectation and variance, this can be changed by adding expensive model evaluations, by using a sequential Bayes Linear update.

6.3.1.2 Emulation for Predators

Adding this known boundary when emulating the population of predators does not have as much of an affect on the emulator, compared to when emulating the population of prey.

Whilst the emulator is no longer predicting negative values for the population of the predators, Figure 6.2 shows that not as much information about the behaviour of the model can be gained compared to with the prey. As x_1 does not affect the population of the predators while $x_2 = 0$ due to there being no interaction between the two populations, the known boundary is constant for all values of x_1. The emulator diagnostics, once away

from the known boundary, shows trends in behaviour similar to the true population from the model. This is due to the emulator adjusted variance and expectation being the same across this area, so the diagnostics is only based on the distance the true population is away from the prior expectation.

If the emulation was instead for input variables x_2 against x_3, with x_1 being kept constant, adding this boundary would have shown more about the behaviour of the model. Adding expensive evaluations will improve the emulator performance for larger values of x_2, and this boundary should help the emulator to only predict positive populations of the predators.

6.3.2 Adding Known Runs

Whilst the positions of the model evaluations can be picked using a Latin Hypercube design as before, as we have the known boundary in the emulator, we can adjust this technique to increase the amount of information that these evaluated runs will provide. If points picked to be evaluated are too close to x_2 then they will not provide as much new information about the behaviour of the model as some of the information will have been gained from the boundary already. We therefore want the points picked to be evaluated at to be located further away from this boundary. This can be done by picking points in a Latin Hypercube Design, but by setting the range of x_2 that the points can be placed in to be $7.5 \times 10^{-5} < x_2 < 7.5 \times 10^{-4}$. This means that no runs will be placed in the 10% of the function space closest to the known boundary, but 9 runs will still be evaluated.

6.3.2.1 Full Emulation Results: Prey

Figure 6.3 shows how, combining the known boundary technique, with the emulation of the known runs, helps to improve the overall performance of the emulator. This can be seen by the emulator diagnostics being closer to 0 than they are in the original emulation without the boundary. This is because the boundary is able to show the increasing population values at $x_2 = 0$, when x_1 is increasing. This is something that the original emulator was not able to achieve because the Latin Hypercube Design for picking the runs meant that the runs were spread over the whole input variable ranges, so were not able to accurately reflect the behaviour at this boundary.

6.3.2.2 Full Emulation Results: Predator

However, Figure 6.4 shows that this known boundary technique does not work as well for the population of the predators. This could be because we have chosen the input variable x_3, which controls the death rate of the predators, to be kept constant. Even though the

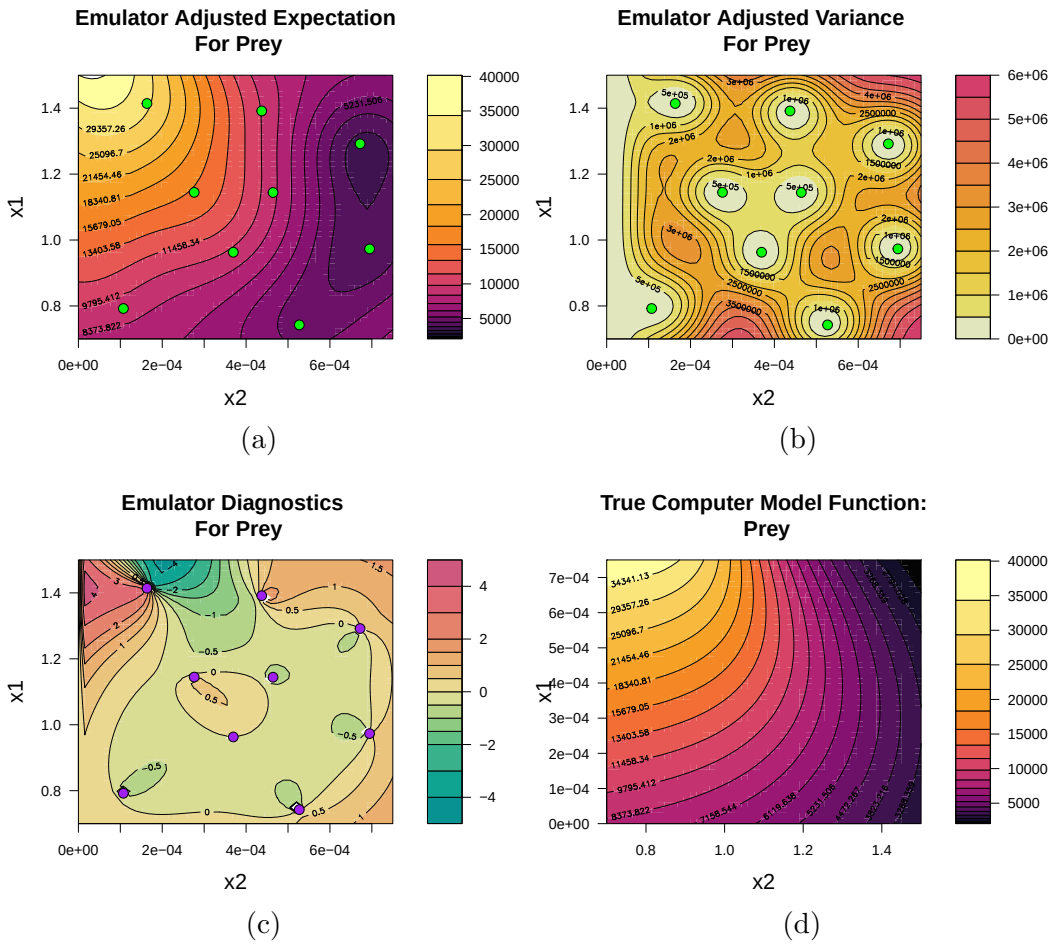


Figure 6.3: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the prey at $t = 2$, with a known boundary at $x_2 = 0$ and 9 evaluated runs.

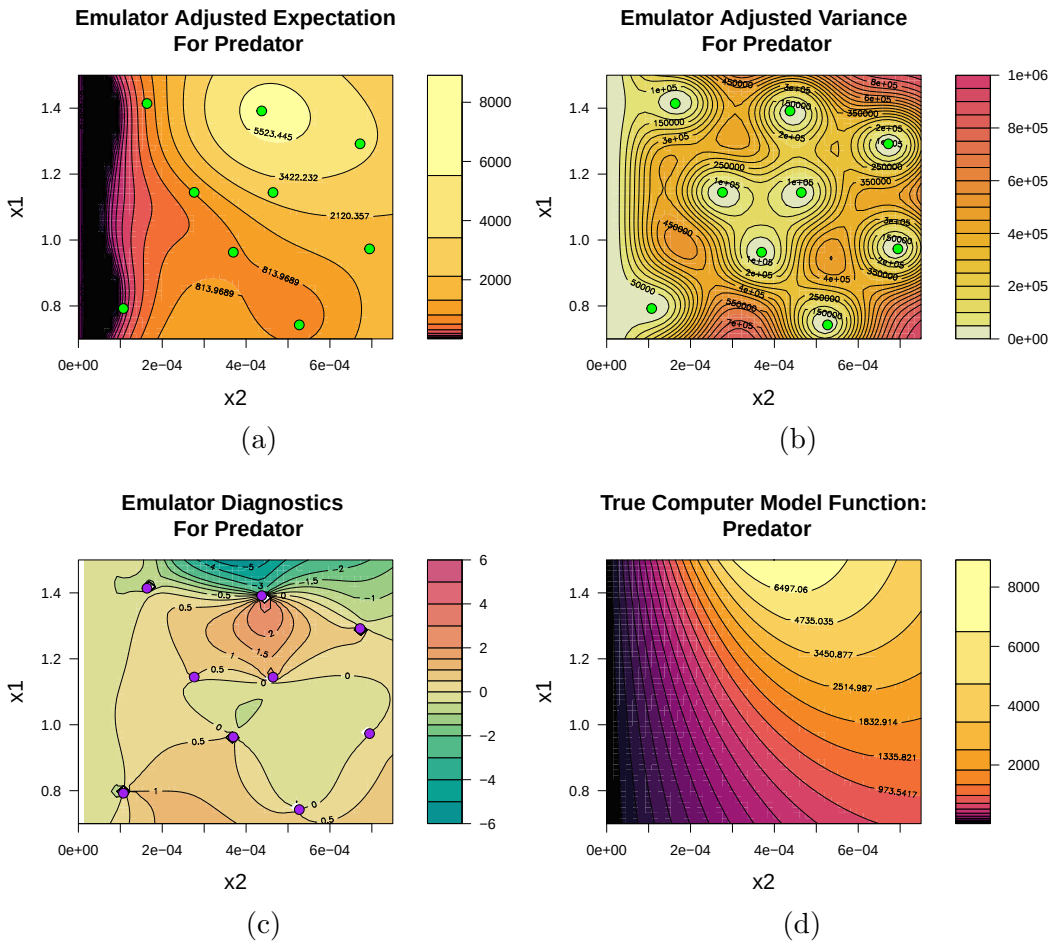


Figure 6.4: (a) The emulator adjusted expectation, (b) the emulator adjusted variance, (c) the emulator diagnostics, and (d) the true population of the predators at $t = 2$, with a known boundary at $x_2 = 0$ and 9 evaluated runs.

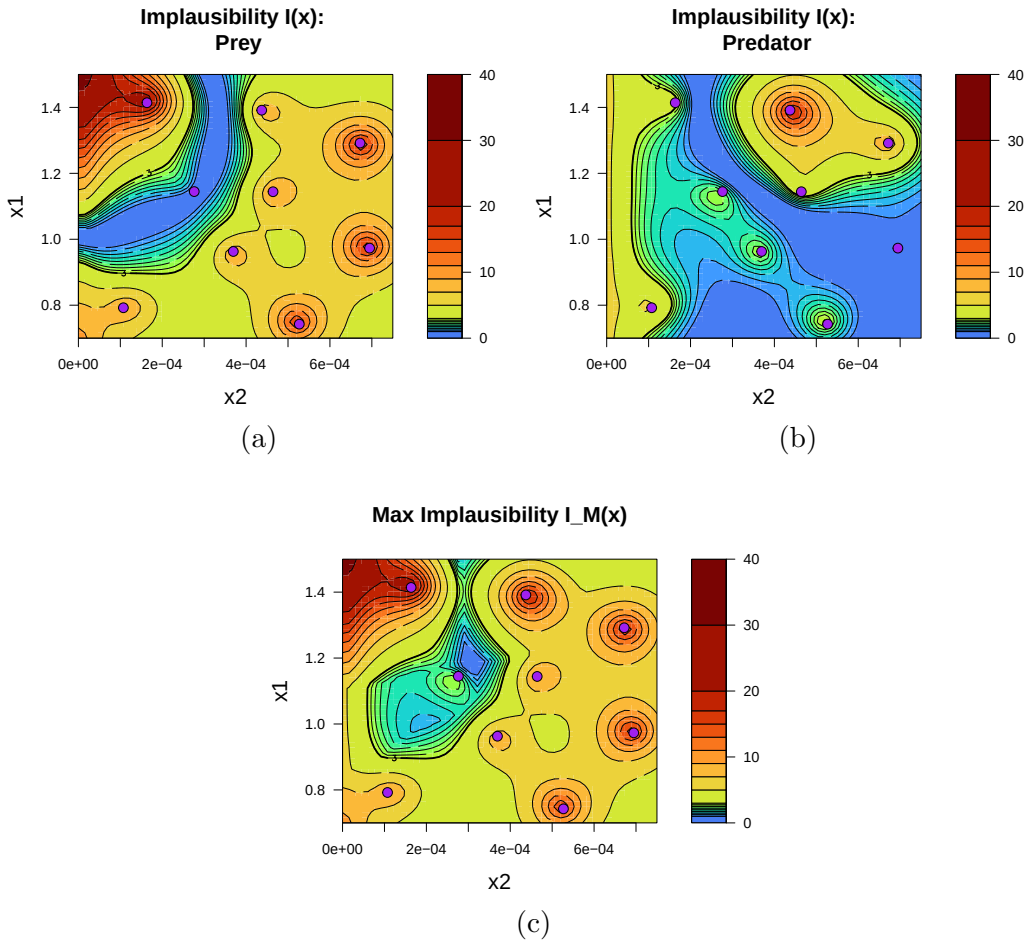


Figure 6.5: (a) The implausibility for the population of the prey from the emulator, (b) the implausibility for the population of the predators from the emulator, and (c) the maximised Implausibility for the two emulators, with a known boundary at $x_2 = 0$ and 9 evaluated runs.

boundary x_2 is known, the emulator still predicts populations of predators close to this boundary to be negative, which can not happen in reality. Even with 9 evaluated runs, the emulator is not able to accurately capture the true behaviour. As the model has exponential behaviour, the large increases in population that occur for large values of x_2 means that the emulator also tries to show this behaviour for lower values of x_2, and this leads to the negative values of population being produced.

6.3.2.3 History Matching

Another way to see the improvement of adding a known boundary to the emulator is to perform history matching and to plot the implausibility functions for these new emulators, using the same z and prior values as in Section 5.3.

Despite not seeming to improve the results of emulation by much, Figure 6.5 shows that the non-implausible region for the predator population is smaller than it is in Figure 5.6. This shows that some information has been gained from adding the boundary, which will also help when deciding where to place a second wave of runs. As the size of the non-implausible region for the population of prey has also decreased, this means that the area that is non-implausible using the maximised implausibility measure I_M has also shrunk. This is now a lot more similar in shape compared to the maximised implausibility of the true function, shown in Figure 5.7. It can more easily be seen, therefore, the affect that adding the known boundary has had to the performance of the emulator, especially with respect to identifying the non-implausible region, and performing history matching. With a smaller non-implausible region, this would allow us to concentrate our second wave of runs in a smaller area and allow us to gain more information about the behaviour of the function in the area we are particularly interested in.

Chapter 7

Further Work

Whilst Chapter 6 shows that using a known boundary technique is beneficial and can improve the performance of an emulator without increasing the number of expensive evaluations that need to be performed, there are still more improvements that can be made. One of these improvements is that more known boundaries could be added to the model, if these are easy to analytically evaluate. These boundaries could either be parallel to the first known boundary, $\mathcal{K}$, or perpendicular [7]. The difficulty, however, of adding another known boundary is working out how the boundaries interact with both each other, and the later evaluated runs.

The performance of this method when varying all three of the input variables at once could also be investigated, as this would also allow us to have a better idea about the affect of all the three variables on the behaviour of the model. When just considering x_1 and x_2, adding the known boundary did not have much affect on the population of the predators, however adding in x_3 would show the improvements that this technique could have. This would mean, however, that there would be a bigger input space that would need to be investigated, so more runs may be needed to achieve the same amount of information. Also, the visualisation techniques mentioned in Section 5.1.2 would need to be used.

The performance of this technique on other complex models could be investigated, either in a completely different scientific area, or by improving the Lotka-Volterra model. This model could be improved by adding more variables to help better model the real system it is representing. For example, other animals, either predators or prey, could be added to better represent the animal population and interactions in an area. Other input variables could also be added, for example, a natural death rate for the prey, or time, t, could be interpreted as a input variable instead of keeping it constant. This would increase the size of the input space and therefore having computationally cheap ways to add information for the emulator would become more important.

Bibliography

[1] Ioannis Andrianakis, Nicky McCreesh, Ian Vernon, Trevelyan J McKinley, Jeremy E Oakley, Rebecca N Nsubuga, Michael Goldstein, and Richard G White. Efficient history matching of a high dimensional individual-based hiv transmission model. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):694–719, 2017.

[2] Richard G. Bower, Michael Goldstein, and Ian Vernon. Galaxy Formation: a Bayesian Uncertainty Analysis. *Bayesian Analysis*, 5(4):619 – 669, 2010.

[3] Jonathan Cumming, Ian Vernon, and Michael Goldstein. History matching. Chapter in the Handbook of Uncertainty Quantification (in submission).

[4] Friedrich Pukelsheim. The Three Sigma Rule. *The American Statistician*, 48(2):88–91, 1994.

[5] Danny Scarponi, Andrew Iskauskas, Rebecca A. Clark, Ian Vernon, Trevelyan J. McKinley, Michael Goldstein, Christinah Mukandavire, Arminder Deol, Chathika Weerasuriya, Roel Bakker, Richard G. White, and Nicky McCreesh. Demonstrating multi-country calibration of a tuberculosis model using new history matching and emulation package - hmer. *Epidemics*, 43:100678, 2023.

[6] Ian Vernon. Uncertainty Quantification 4 Lecture Notes, 2023. Durham University, MATH4337.

[7] Ian Vernon, Samuel E Jackson, and Jonathan A Cumming. Known Boundary Emulation of Complex Computer Models. *SIAM/ASA Journal on Uncertainty Quantification*, 7(3):838–876, 2019.

[8] Ian Vernon, Junli Liu, Michael Goldstein, James Rowe, Jen Topping, and Keith Lindsey. Bayesian uncertainty analysis for complex systems biology models: emulation, global parameter searches and evaluation of gene functions. *BMC systems biology*, 12:1–29, 2018.

[9] Ian Vernon, Jonathan Owen, Joseph Aylett-Bullock, Carolina Cuesta-Lazaro, Jonathan Frawley, Arnau Quera-Bofarull, Aidan Sedgewick, Difu Shi, Henry Truong, Mark Turner, et al. Bayesian emulation and history matching of june. *Philosophical Transactions of the Royal Society A*, 380(2233):20220039, 2022.

[10] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.