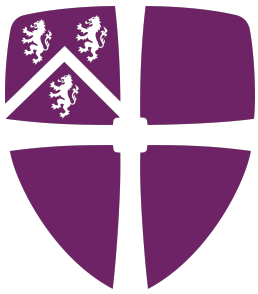


Durham University

Uncertainty Quantification Mini Project

Jack Lloyd

Department of Mathematical Sciences

Michaelmas 2023

Lecturer: Professor Ian Vernon

1 Introduction

In this report we will explore and attempt to emulate a physical model using techniques learnt in the Uncertainty Quantification course. We will be focusing on the 'Robot Arm' model which models the position of a robot arm which has four segments. The start of the arm (shoulder) is fixed at the origin, each segment of the arm has length L_i and are positioned at an angle of θ_i, with respect to the horizontal axis, assuming that the previous angle was the horizontal axis (see **Figure 1** for an example). $L_i \in [0, 1]$ and $\theta_i \in [0, 2\pi]$ for $i = 1, 2, 3, 4$. The function output is the distance from the origin to the end of the robot arm.

$$f(\boldsymbol{x}) = (u^2 + v^2)^{\frac{1}{2}}$$

with

$$u = \sum_{i=1}^{4} L_i \cos\left(\sum_{j=1}^{i} \theta_j\right) \qquad v = \sum_{i=1}^{4} L_i \sin\left(\sum_{j=1}^{i} \theta_j\right)$$

This function has been chosen due to its periodicity. It will be interesting to see how our emulators learn about the function.

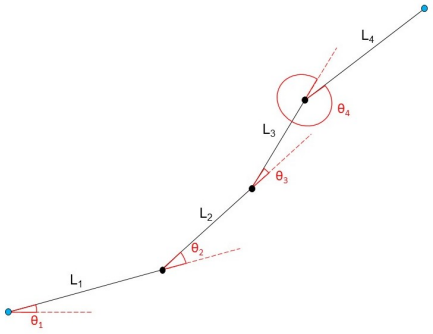
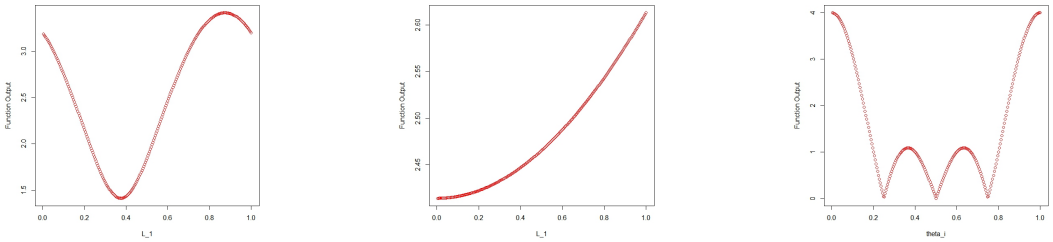


Figure 1: Example of a Robot Arm



(a) Varying θ_2 (b) Varying L_1 (c) Varying all $\theta_i's$

Figure 2: Varying Different Input Parameters to Observe Impact on Output

The function's output will be the distance between the origin and the end of the fourth segment, ie the distance between the two blue dots in **Figure 1**. We note that the first angle, θ_1, has no impact on

the output of the distance between the end of the fourth segment of the arm and the origin. Consider **Figure 1**, varying θ_1 and keeping all other inputs fixed is the same as rotating the entire image; it has no impact on the distance between the blue dots. We see that all other inputs have an impact on the output. **Figure 2** shows how the output changes when we vary the impact parameters. For all three plots, the fixed inputs have values $\theta_i = \frac{\pi}{4}$ and $L_i = 1$. In **Figure 2c**, we vary all angles at the same time, giving $\theta_i = x$, $x \in [0, 1]$. In these plots we have already altered our function so that the input angles values are in the range $[0, 1]$ before transforming them to the range $[0, 2\pi]$ as this needs to be done for the emulation stage.

2 Emulation in One Dimension

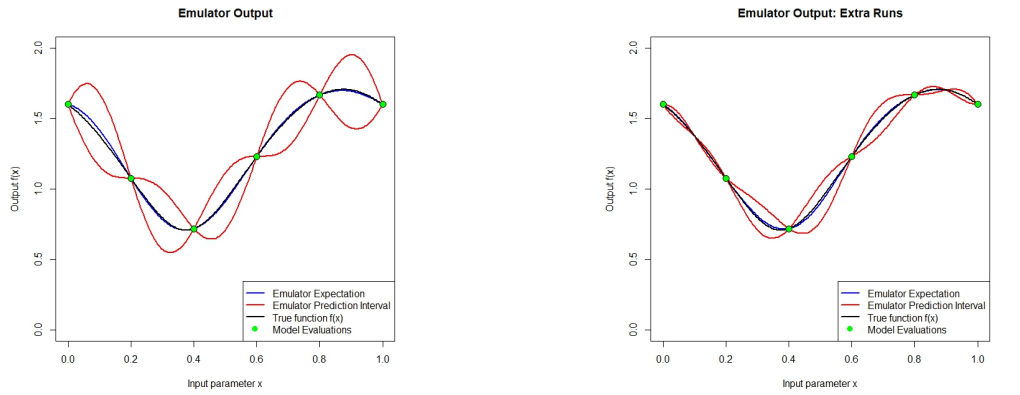
In this section we will begin our analysis by emulating in one dimension, where we choose one of the input paramters to be free and then fix all others at a chosen value. We will start by fixing all input parameters apart from θ_2, which we leave free. In our first emulation, we set $\theta_1 = 0$, $\theta_3 = 1/8, \theta_4 = 1/8$ and all lengths to equal $1/2$. An input angle of $1/8$ corresponds to an angle of $\frac{\pi}{4}$. First we selected six evenly spaced run locations

$$\theta_{2\,D} = (0, 0.2, 0.4, 0.6, 0.8, 1)$$

Since we have fixed all lengths to equal $\frac{1}{2}$, we set our $E[f(x)] = 1$. We also assumed a square exponential covariance structure as such:

$$\text{Cov}[f(x), f(x^{'})] = \sigma^2 \text{exp}\Big\{ - \frac{|x - x'|^2}{\gamma^2} \Big\}$$

Where we write γ instead of the usual θ for the correlation length parameter to avoid confusion due to the angles θ_i. We then used the Simple Bayes Linear Emulator to perform Bayes Linear Adjustment to output the adjusted expectation and variance. We set $\gamma = 0.25$ and $\sigma = 0.5$. We predicted at 201 input points and obtained the outputs in **Figure 3a** below.



(a) 1D Emulation Output with 6 Runs

(b) 1D Emulation Output with 8 Runs

Figure 3: 1D Emulation Output: Emulating θ_2

Figure 3a shows us that the emulator has performed well over the range of inputs, with our emulator expectation being extremely close to the true value of the function. We see the emulator is not perfect for lower values of θ_2, so we introduce two more run locations $\theta_2 = (0.1, 0.9)$ in attempt to make our emulator expectation closer to the true value of the function at these values. This is shown in **Figure 3b**. In turn it also greatly reduces the prediction intervals across the whole range $[0, 1]$. We now explore how our emulator works when we vary our prior standard deviation, σ, while fixing $\gamma = 0.25$, and when varying our correlation length, γ, while fixing $\sigma = 0.5$. We use our six original run locations.

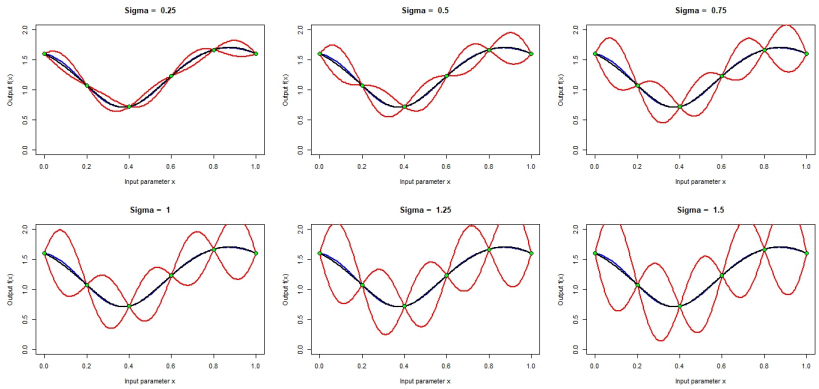


Figure 4: Emulator Performance when varying prior standard deviation σ

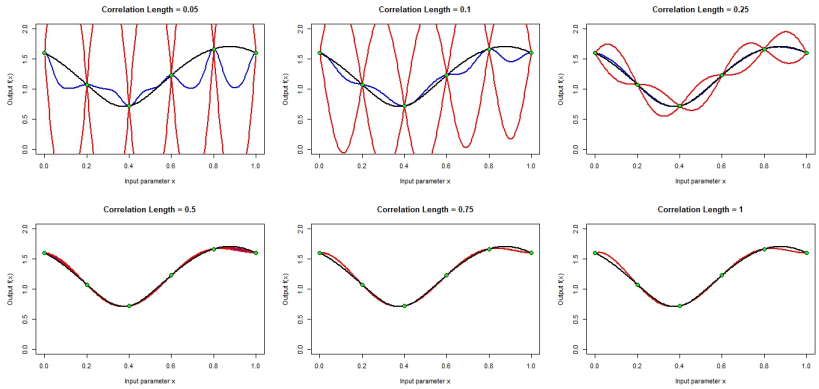


Figure 5: Emulator Performance when varying correlation length γ

We see that as we increase our prior variance, the prediction intervals get larger. Varying the prior variance has no effect on the emulator expectation. However when we vary the correlation length, we see that smaller values make our emulation expectation venture further away from the true value of the function and we see the prediction intervals become extremely large. Also note that as our correlation length goes to 1, our emulator expectation seems equal to the true function value over the entire $[0, 1]$ range!

3 Emulation in Two Dimensions

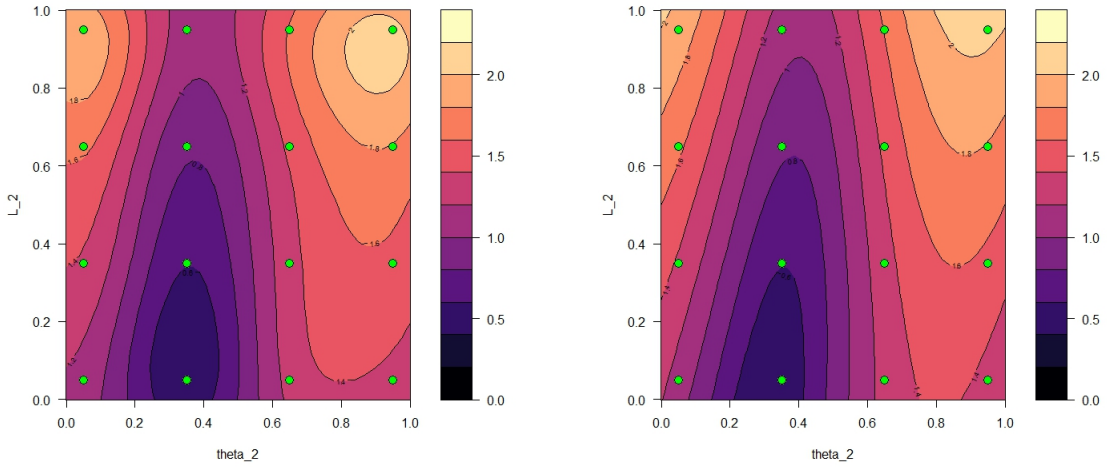
We will now perform emulation in two dimensions. To start, we will fix all inputs apart from the second angle and the second length, θ_2 and L_2. We will fix all angles to have input value $1/8$ and all other lengths to be equal to $1/2$. We slightly change the covariance structure to

$$\text{Cov}[f(x), f(x^{'})] = \sigma^2 \text{exp}\Big\{ - \frac{||x - x'||^2}{\gamma^2} \Big\}$$

We set our prior standard deviation to be $\sigma = 0.5$, correlation length to be $\gamma = 0.25$ and our prior expectation $\mathbb{E}[f(x)] = 1$. We start by using a 4×4 grid design as such

$$\{0.05, 0.35, 0.65, 0.95\} \times \{0.05, 0.35, 0.65, 0.95\}$$

which has 16 run locations, all possible combinations of values. **Figure 6a** shows the emulator adjusted expectation and **Figure 6b** shows the true model function value across the $[0, 1] \times [0, 1]$ range. The x axis represents θ_2 and the y axis represents L_2.

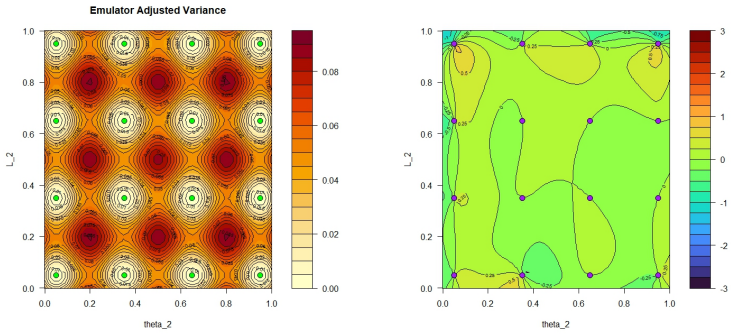


(a) Adjusted Emulator Expectation

(b) True Model Function

Figure 6: Initial Plots of Emulator and True Function Using a 4×4 Run Grid

We see that our emulator is not actually far off the true function. We see the biggest differences in the sides of the grid where we have more extreme values. We also observe the adjusted emulator variance in **Figure 7a** and the Emulator Diagnostics in **Figure 7b**.



(a) Adjusted Emulator Variance (b) Emulator Diagnostics

Figure 7: Emulator Adjusted Variance and Diagnostics for a 4×4 Run Location Grid

Emulator diagnostics give us an insight into how 'well' our emulator is performing. Here we provide the formula to explain how the diagnostics are calculated.

$$S_D(f(x)) = \frac{f(x) - E_D[f(x)]}{\sqrt{Var_D[f(x)]}}$$

Essentially it is giving the scaled difference between the adjusted emulator expectation and the true value of the function at every input location. A 'good' emulator wants the diagnostics to be as small as possible but it is widely accepted that $-3 < S_D(f(x)) < 3$ is sufficient to be classed as a competent attempt. **Figure 7b** shows that our emulator has performed fairly well with a 4×4 grid of run locations. We observe how the emulator fairs if we drop down to a 3×3 grid of run locations as such:

$$\{0.25, 0.5, 0.75\} \times \{0.25, 0.5, 0.75\}$$

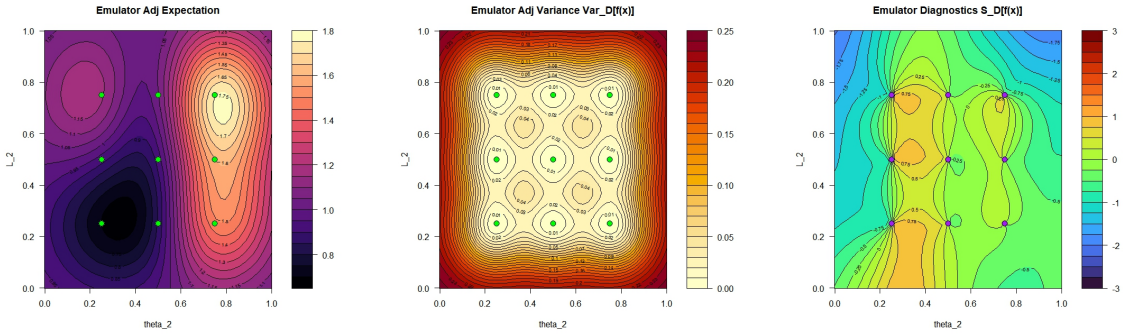
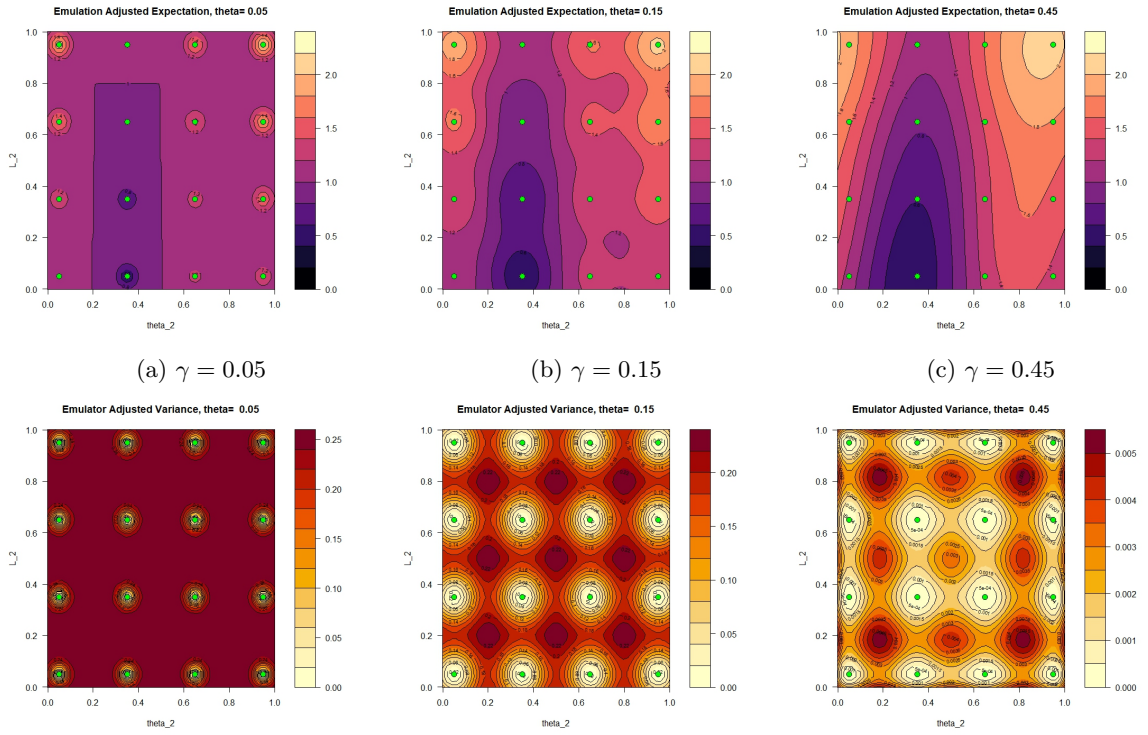


Figure 8: Plots Showing Emulator Performance on a 3×3 Run Location Grid

Figure 8 shows that the emulator struggles much more when we reduce the run locations to a 3×3 grid. The difference between the adjusted expectation and the true model function demonstrates this

struggle, and is confirmed by observing the diagnostic plot where we see areas of orange. We will also observe how the emulator changes when varying the correlation length γ. See **Figure 9** for the results.



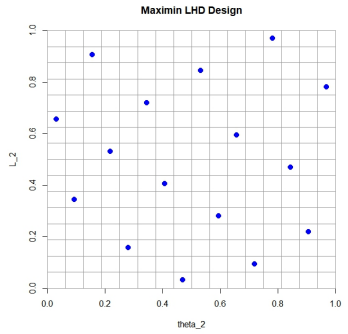
(a) $\gamma = 0.05$ (b) $\gamma = 0.15$ (c) $\gamma = 0.45$

(d) $\gamma = 0.05$ (e) $\gamma = 0.15$ (f) $\gamma = 0.45$

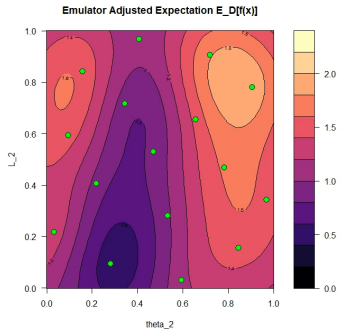
Figure 9: Adjusted Emulator Expectation and Variance for a range of values for γ

Having a symmetric $n \times n$ grid makes sense intially. Each dot is placed in such a way that it seems that each area of the range is being explored equally. However we will show in **Figure 10** how our emulator performs when we use a **Maximin Latin Hypercube** design. **Figure 10a** shows the run location grid design that our algorithm outputted, which seeks to maximise the distance between all points whilst ensuring there is only one point in each row and column.

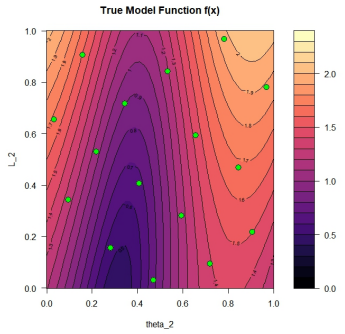
We see in **Figure 10b** and **Figure 10c** that the emulator adjusted expectation is close in appearance to the true model function. **Figure 10d** shows that the emulator adjusted variance is kept low over the majority of the range of input values, apart from in the corners. The diagnostics (**Figure 10e**) over the whole input grid are in the range $-2 < S_D(f(x)) < 2$ which suggests our emulator works well.



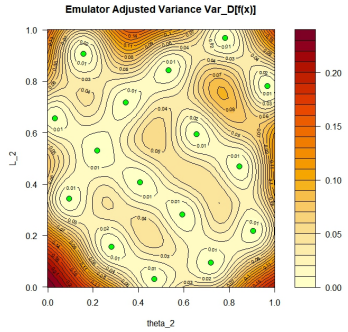
(a) Maximin LHD Design



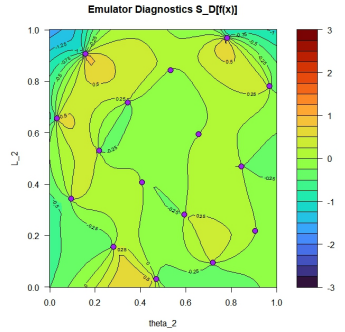
(b) Adjusted Emulator Expectation



(c) True Function



(d) Adjusted Emulator Variance



(e) Emulator Diagnostics

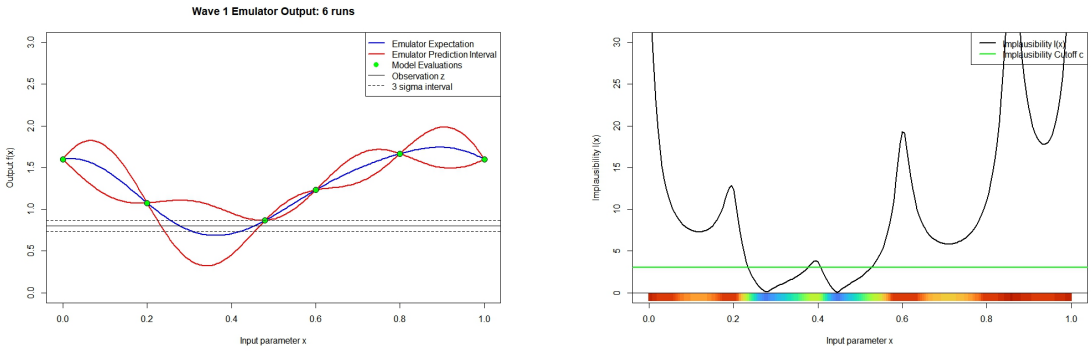
Figure 10: Multiple Plots Using the Maximin LHD Design for Run Locations

4 History Matching

In this section we will attempt to perform **history matching**, a technique used find sets of acceptable inputs using emulation. When performing history matching we use implausibility measures which essentially tell us, if we consider an observed value z, how likely we would be to obtain a match between $f(x)$ and z given an input value x. The implausibility measure $I(x)$ that is considered is of the form

$$I^2(x) = \frac{(E_D[f(x)] - z)^2}{\text{Var}_D[f(x)] + \text{Var}[\epsilon] + \text{Var}[e]}$$

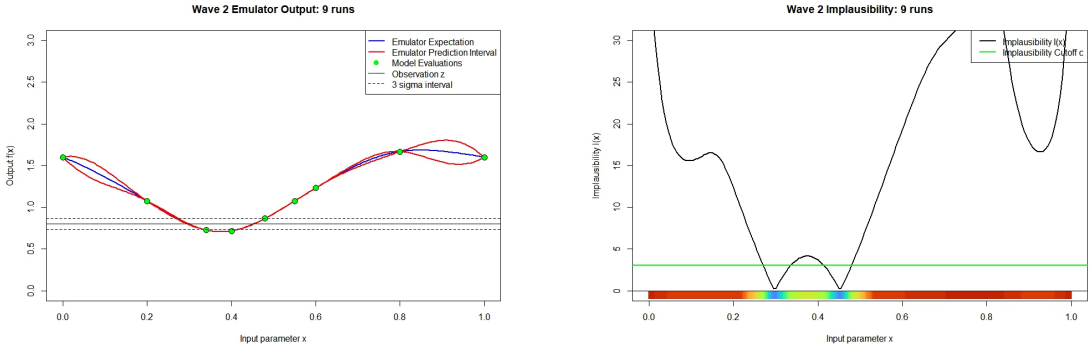
Where $\text{Var}_D(f(x))$ is the emulator variance, $\text{Var}(\epsilon)$ is the model discrepancy variance and $\text{Var}(e)$ is the observation error variance. We will start in one dimension, still varying θ_2. We consider an observed value of $z = 0.8$ with $\sigma_e = \sqrt{\text{Var}(e)} = 0.01$ and $\sigma_\epsilon = \sqrt{\text{Var}(\epsilon)} = 0.02$. We still assume $\gamma = 0.25$, $\sigma = 0.5$ and $E[f(x)] = 1$. We use a slightly different set of run locations in $\{0, 0.2, 0.48, 0.6, 0.8, 1\}$. First we observe the 1D plot for the emulator (**Figure 11a**) and see that a range of input values could give us a value of $f(x) = 0.8$. **Figure 11b** shows the implausibility measure across the range of inputs, with blue denoting the 'non-implausible' region of values. We see that this is a rather large range.



(a) 1D Emulator (b) Implausibility Measure

Figure 11: 1D Emulator Plot with Implausibility Measure for 6 runs

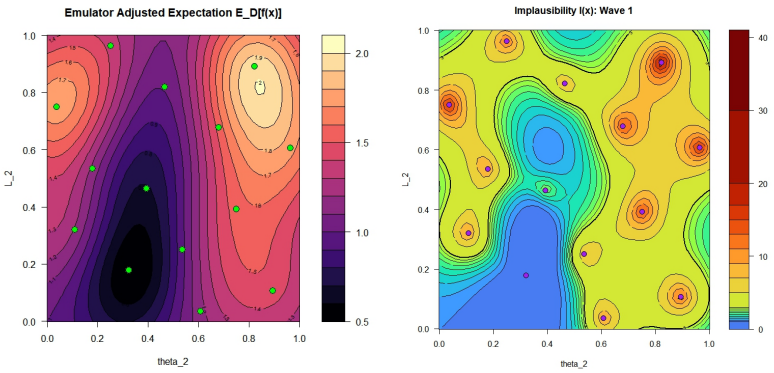
We now want to include three more run locations in an aim to make the non-implausible region smaller. We pick three points that are around the non-implausible region as $\{0.34, 0.4, 0.55\}$. We then include these points one at a time with the original six run locations when performing emulation to see how our emulator updates each time. This progress is shown in **Figure 16** in the Appendix. **Figure 12** shows the final update when we include all three extra runs.



(a) 1D Emulator (b) Implausibility Measure

Figure 12: Emulator Expectation and Implausibility Measure when including $x = \{0.34, 0.4, 0.55\}$

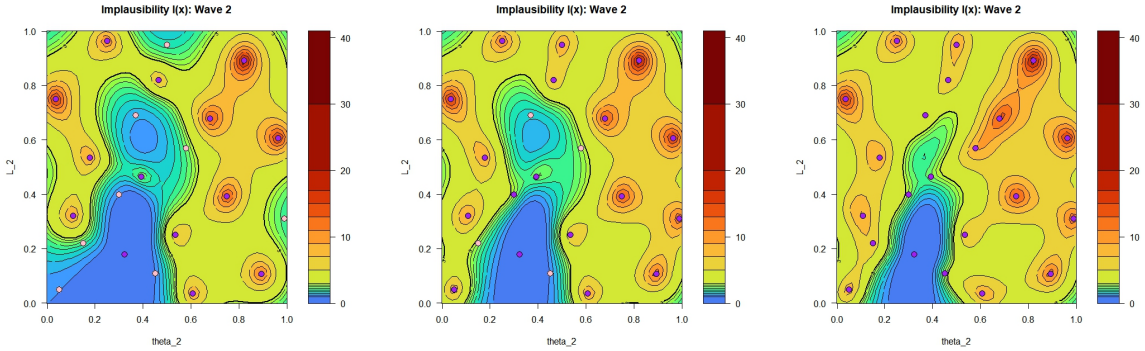
We can see that when we include all three extra run locations when performing emulation, our implausibility measure narrows down the implausible region to two very small regions. We will now see consider history matching in two dimensions. This will again be done by varying θ_2 and L_2, and setting $\gamma = 0.25$, $\sigma = 0.5$ and $E[f(x)] = 1$. We consider the observed value $z = 0.5$, with $\sigma_e = 0.06$ and $\sigma_\epsilon = 0.05$. We use a Maximin Latin Hypercube grid design as in **Section 3**. **Figure 13** shows the adjusted emulator expectation and the implausibility measure for the emulator given the initial design.



(a) Emulator Adjusted Expectation (b) $I(x)$

Figure 13: Expectation and Implausibility for 2D Emulator with Maximin LHD

We look at the implausibility measure (**Figure 13b**) and want to add in a second wave of runs where we look to shrink the non-implausibility region. All of these runs are hand picked to be inside the original non-implausible (blue) region. **Figure 14a** shows the implausibility measure with the original runs in purple and the runs we will add in one-by-one to see how the implausibility measure changes. These new runs are shown in pink. Once they are included in the emulator then they are shown in purple. We include the plots for every four runs. The run-by-run plots are included in **Figure 15** in the Appendix.



(a) 0 Runs Included (b) 4 Runs Included (c) 8 Runs Included

Figure 14: Implausibility Measure Plots Updated Run by Run *(showing every fourth run)*

5 Optimisation

The final technique we will briefly touch on is optimisation. We will aim to improve the efficiency of our emulator and measure this by returning the time taken for `R` to return our emulator adjusted expectation and variance matrix. This analysis has been done when emulating in two dimensions. If

we use our original emulation function, it takes the system 10.17 seconds to calculate the adjusted expectation and variance values. The reason for this large time is the `for` loops used to construct the covariance matrices. We can use the `dist()` function to speed this up, which automatically calculates the distances between all pairs of points in a matrix. Using this in our calculation of `em_out`, the matrix containing the adjusted expectation and variance for all input points, the system takes 1.08 seconds to return an output. We see that this has sped up calculations by a factor of 10. We can make this even quicker by using a function called `pdist()`, which returns the distances between two lists of points, which is exactly what we need to calculate the covariance matrix. Using `pdist()` instead of `dist()`, the calculation of `em_out` takes 0.02 seconds.

This is approximately 500 times faster than our original method. In this case, ten seconds has little impact on our work. However, when emulating in many dimensions at once, models can take much longer to output results. In practise, optimisation is a key area of emulation as this will greatly impact cost and efficiency.

6 Conclusion

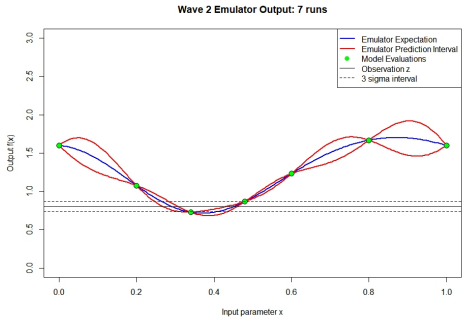
When performing 1D emulation, we saw that our emulator performed well with the adjusted expectation being very close to the true value of the function. However the prediction intervals were still rather large in between run locations. Adding two extra run locations towards the boundaries of the values of θ_2 greatly reduced these intervals. We saw that the larger the value of the standard deviation σ, the larger the prediction intervals. Using small correlation lengths made our emulator perform poorly (see **Figure 5**), with the adjusted expectation being far from the true value of the function and the prediction intervals becoming extremely large across the range of input values.

In the analysis of emulation in 2D, we saw that our initial emulator (using a 4×4 grid of run locations) performed okay. The emulator diagnostics were in the $[-3, 3]$ range and the adjusted expectation plot looked fairly similar to the true model function plot. We saw when moving to a 3×3 grid of run locations, the emulator performed much worse (see **Figure 8**). Our emulator performed best when using a Maximin Latin Hypercube design (see **Figure 10**).

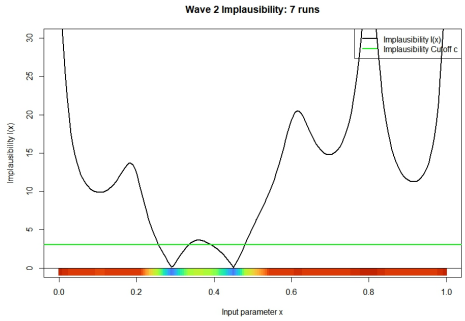
When performing history matching, we saw that when including a second wave of run locations in both 1D and 2D, our non-implausibility region shrunk considerably. It was interesting to see how our emulator updated after the inclusion of each new run location since we could see it learning more about the function each time. We finally showcased how we can make our emulators much more efficient by making very slight changes in the `R` code.

We have shown that emulation is a very efficient and accurate method of learning about an unknown function. Our emulators managed to recognise the periodicity patterns of the robot arm function and recognise how multiple input values can lead to the same output. This was a key reason why the robot arm function was chosen to perform analysis on, so we can say that we succeeded in our initial goal of emulation.

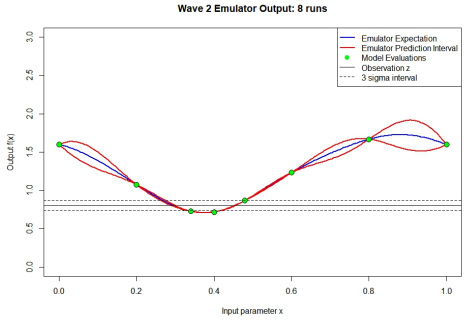
Appendix



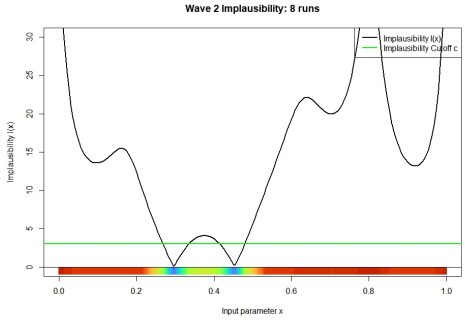
(a) $x = \{0.34\}$



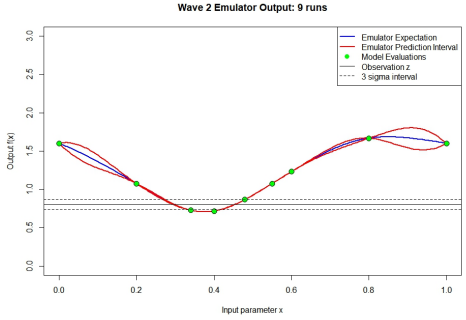
(b) I(x) including $x = \{0.34\}$



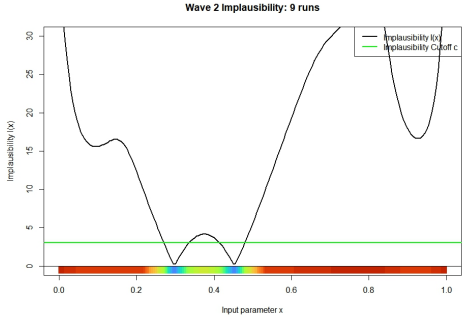
(c) $x = \{0.34, 0.4\}$



(d) I(x) including $x = \{0.34, 0.4\}$



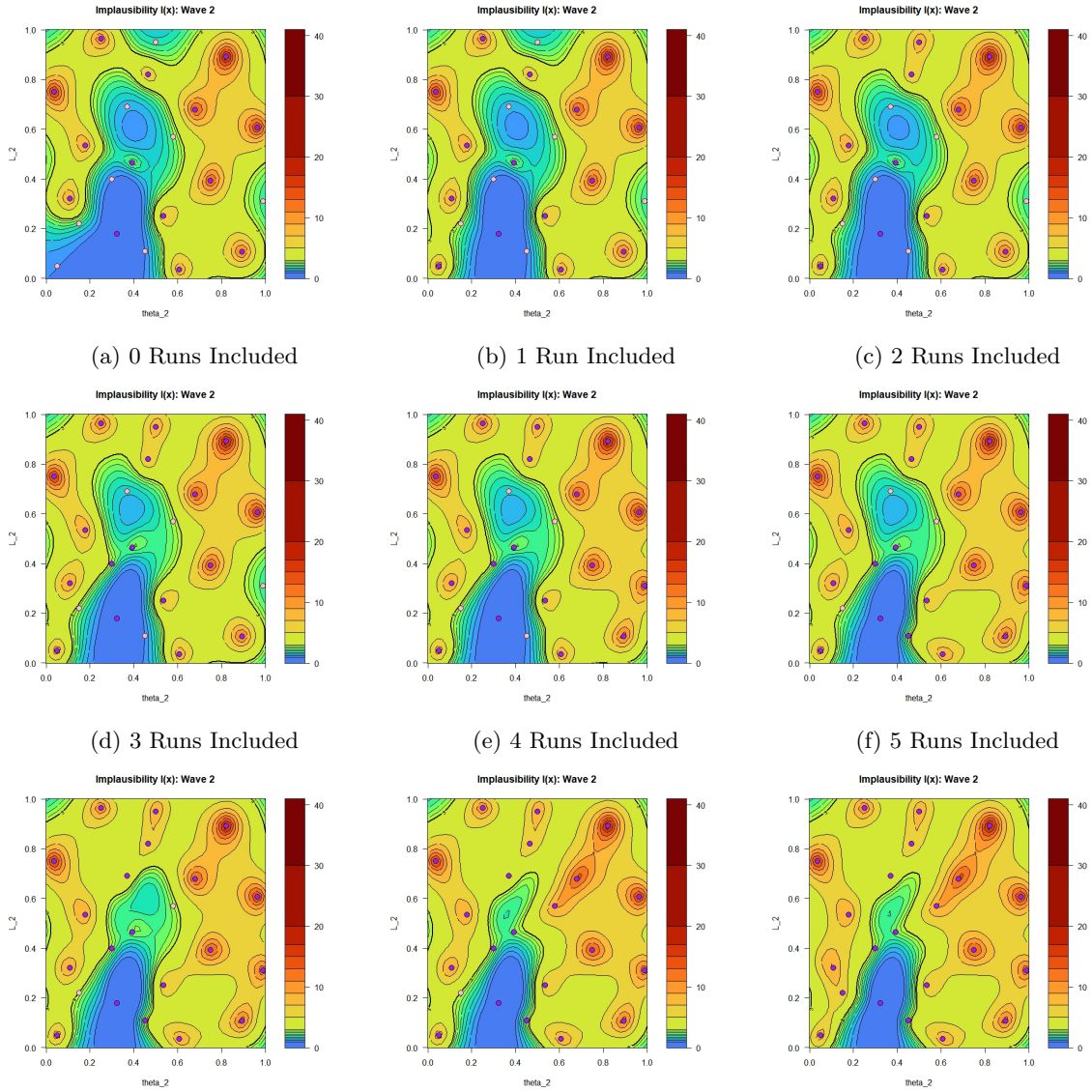
(e) $x = \{0.34, 0.4, 0.55\}$



(f) I(x) including $x = \{0.34, 0.4, 0.55\}$

Figure 15: Updating Emulator and Implausibility Measure when including more run locations

We see that our non-implausibility region gets smaller after the inclusion of every run. This is what we expect to see as we explore specific areas of the region each time we add a run, hence telling our emulator whether this point is in an area of non-implausibility.



(a) 0 Runs Included (b) 1 Run Included (c) 2 Runs Included

(d) 3 Runs Included (e) 4 Runs Included (f) 5 Runs Included

(g) 6 Runs Included (h) 7 Runs Included (i) 8 Runs Included

Figure 16: Implausibility Measure Updated Run by Run

Figure 16 shows how our non-implausibility region changes as we add each run into our emulator. **Figure 16a** shows the original non-implausibility region in blue with no extra runs included. We see that as we add each run, that the blue region it is located in turns into a region of implausibility. Once we have included all 8 extra runs it is interesting to observe how much our non-implausible region has shrunk.