Using Machine Learning Surrogates to Enable Wave Energy Converter Layout Optimisation

Kevin Dang - a1686730

Supervisors: Dr Markus Wagner, Dr Brad Alexander

October 2018

Thesis submitted for the degree of Honours in Bachelor of Computer Science (Honours)

SCHOOL OF COMPUTER SCIENCE

THE UNIVERSITY OF ADELAIDE AUSTRALIA

Declaration

Except where stated this thesis is, to the best of my knowledge, my own work and my supervisor has approved its submission.

Signed by student: 

Date: 02/11/2018

Signed by supervisor: M. Wagner

Date: 02/11/2018

Acknowledgements

- I would like to thank my supervisors: Dr Markus Wagner and Dr Brad Alexander and also Mrs Aneta Neumann for providing support and guidance throughout my project.
- I would also like to thank my parents for their continued support throughout my years.
- Lastly, I would like to thank the University of Adelaide for providing access to a remote server that I could perform my experiments on.

Abstract

Wave energy has great potential as a source of renewable energy. The University of Adelaide has developed a model that can accurately predict the expected power output of a layout of wave energy converters. However this process is very computationally expensive and has a long runtime. The goal of this project was to use machine learning surrogates to allow for reduced runtimes for the wave energy converter layout model.

Two candidate sections of code were identified and investigated and surrogate models were built for them. A variety of machine learning models were tested, and the auto-sklearn tool helped in this process. It was found that Random Forest Regressors performed the best with very high R^2 scores, however the wave layout model's high sensitivity to random errors prevented these surrogate models from performing well in terms of accuracy when inserted into the wave layout model.

Contents

Chapter 1

Introduction

1.1 Background

Renewable energies are an ever increasingly popular form of energy source, both as a cheaper form of energy and a renewable source that is much less harmful towards the environment. Renewable energy is also a core focus of the worldwide Paris Climate Change Agreement [9]. As such the drive to not only improve existing renewable energy technology, but to explore and utilise other forms of renewable energy is vital.

Wave energy is one form of renewable energy with very high potential. Due to the difficulties of working in an ocean environment, they have been largely unexplored until recently. However they much to offer, as they have one of the highest energy densities among the renewable energies [1].

The piece of technology that converts wave energy, either to a different form such as electricity or used to directly power other machines like desalination plants, is called a wave energy converter (WEC). There are many different types of WECs, but the most common is the buoy, which sits below the surface of the waves and is tethered to the ocean floor. An example of a 3-tether buoy is given in figure Wave movements affect the buoy which then translate into a piston at the base of the tether. This piston can then convert the kinetic energy into the desired form, such as electricity [1].

Negative interference is always present, where the location of another energy converter negatively affects a different converter. For example, in the case of wind energy if two turbines are placed directly behind one another then the second turbine outputs less power as the first turbine is blocking the wind.

However, an interesting aspect of wave energy, is the existence of positive interference. The positioning of buoys can actually amplify the wavelength of waves in certain directions after it passes by the buoy. Placing another buoy along this direction then allows it to output more energy

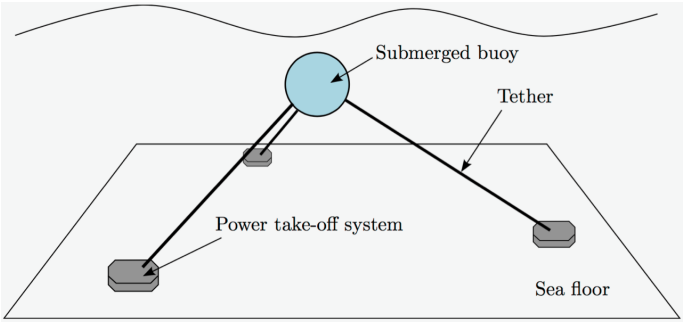


Figure 1.1: 3-tether buoy. The buoy sits below the water. Kinetic energy from waves are transferred along the buoy, through the tethers and into the power take-off system, which then converts it into other forms of energy.

than normal. This highlights another important goal in finding optimal WEC layout configurations: negative interference should be avoided while positive interference should be worked towards.

Machine learning surrogates use machine learning models to replace sections of code. They act like a black box, where input is placed into the surrogate and the predicted outputs are received. The intent of their use is to train the surrogate using data flowing in and out of the candidate section of code in the hope to eventually replace the candidate section with the surrogate. As the surrogates use machine learning models to predict their outputs, they have an inherent inaccuracy associated with the model. This accuracy can be reduced through a number of methods such as trying different models or optimising the parameters of the model. However 100% accuracy is obviously not possible. The benefit of using the surrogate model is that they are generally fast to run balanced by the potential for inaccurate predictions.

1.2 Motivation

The University of Adelaide has constructed a WEC layout model that can take the positions of a number of buoys as input and then return the expected power output of the total power and individual power outputs of the buoys. However this simulation is very computationally expensive due to the complex, hydrodynamic interactions between the waves and buoys, and the pairwise interactions between buoys. The pairwise interactions must be calculated in order to accurately model any positive and negative interferences that occur. As an example, the wave layout model takes over 9 minutes to process a 16-buoy layout takes 9 minutes to finish on the university's OptLog2 remote server.

Naturally when given a number of buoys in a specific environment, it is desirable to discover the best layout that minimises negative interference and maximises positive interference. This optimisation process, whether through evolutionary algorithms or other means, requires the evaluation of many different layouts. The slow evaluation time of a single layout severely hampers the ability to explore the search-space extensively. As such it is desirable to reduce the runtime of the wave layout model.

Machine learning surrogates present one option for reducing the runtime. However this comes at the cost of losing accuracy as well. The goal of this project is to explore this option and apply it to the existing wave layout model.

1.3 Machines Used

This section details the 3 main machines used throughout the course of this project: my personal laptop, my personal computer and the University of Adelaide's Optimisation and Logistics Group's remote server: OptLog2.

Details about my personal laptop:

- OS: Windows 10 Home 64-bit
- Processor: Intel i7-8550U @ 1.80GHz (4 cores, 2 threads each)
- RAM: 8GB

Details about my personal computer:

- OS: Windows 10 Home 64-bit
- Processor: Intel i5-4440 @ 3.10GHz (4 cores, 2 threads each)
- RAM: 12G

Details about OptLog2 server:

- OS: CentOS release 6.9
- Processor: 4 CPUs (8 cores each, 48 cores in total, no hyperthreading)
- RAM: 128GB DDR3

1.4 Related work

Moraglio and Kattan [6] look at a very similar problem. They identify the need for surrogate modelling as some tasks when cast as optimisation problems create objective functions that are computationally expensive to evaluate. They also identify that a majority of these problems, especially engineering ones, are black-box: how the problem function operates is unknown and often mathematically ill-behaved (discontinuous, non-linear, non-convex, etc).

They first detail a surrogate model based optimisation approach. This approach would sample a small set of data points to the optimisation problem at random and then evaluate them. While some limit has not been reached yet (either a time limit or a limit on the number of expensive evaluations), the known data points are used to train a surrogate model and then the surrogate model is used to search for an optimal point either through an evolutionary algorithm or other means. This is feasible as the runtime of the surrogate model is orders of magnitude faster than the expensive objective function. The data point found by the surrogate model is evaluated by the objective function and then added to the known set of data points. This process then repeats until the terminating condition is met. At the end, the best solution among the list of known data points is returned.

They then go on to show that using this approach combined with a Radial Basis Function Networks model (a type of neural network) can be successfully generalised to any solution representation using the geometric methodology. They also identify Gaussian Process Regression as another powerful model that could also have been used.

In terms of differences between their work and ours, the main one is that they look at objective functions as a whole, whereas our wave layout model is much more complex and we look at using surrogates to replace internal sections of code.

Liu et al. [5] proposed a surrogate model assisted evolutionary algorithm based around a Gaussian Process model. They aim their work towards medium scale problems with 20 to 50 decision variables. In order to deal with so many variables and the "curse of dimensionality", they employ a dimensionality reduction technique: the Sammon mapping. They choose this technique with the consideration that the neighbourhood relationship among points is maintained: pairwise distances between points should be preserved as much as possible.

They then run their proposed algorithm through a series of test problems to observe how the algorithm with dimensionality reduction performs compared to one without any reduction. They find that with lower decision variables, their algorithm without dimensionality reduction performs betters and with more decision variables. The version with dimensionality reduction only outperformed for problems with 50 decision variables.

Pyromallis [8] worked on this project previously with the goal of using surrogate models to enable optimisation of wave layout model evaluations focusing specifically on the power take-off (PTO) parameters. They trained 4 different regression and classification models for different variations of the problem, such as when PTO settings are the same across buoys or different for individual buoys. The 4 models they used were: K Nearest Neighbours, Random Forest, Multilayer Perceptron and Supper Vector Machines. They found that out of the 6 variations of the problem, Random Forest performed best for one, K Nearest Neighbours performed best for another and then Multilayer Perceptron performed the best for the remaining 4 variations.

Chapter 2

Wave Layout Model

2.1 Model description

The University of Adelaide's wave layout model was developed in MATLAB by Mrs Nataliia Sergiienko using the mathematical model created by Wu [11]. This model, when given the positions of an array of buoys, is able to accurately predict the expected power output of each individual buoy as well as the combined power output. This process also takes into account the positive and negative interference that arise from the positions of the buoys.

The model takes in additional parameters. There is the domain information detailing the waves: the angles and frequencies that can occur, the wave spectrum and the depth. This domain information is provided to the model through the siteOpts variable used in Algorithm 1. There is also information regarding the buoys itself: their mass, volume, tether angle, and power take-off settings, which is stored inside the lookUpTable variable in Algorithm 1. As it was not necessary to modify these parameters, they were left at their default settings.

An important parameter in the context of this project is the domain information about the wave angles and frequencies. The model loops over every frequency in order to calculate each buoy's power output for that frequency. The wave angles determined the direction that the positive and negative interference would occur. The wave angles were measured in radians and there were 7 in total that were equidistantly spaced around the value of 0 radians: [-0.262, -0.175, -0.087, 0, 0.087, 0.175, 0.263]. In MATLAB, 0 radians translates to the positive direction of the x-axis. As the wave angles are centred around this direction, the negative interference is predominantly seen along the x-axis of the buoys.

2.1.1 Information flow of the model

The model consists of two main functions: arrayBuoyPlacement and arraySubmergedSphere. arrayBuoyPlacement is the main function which calculates the expected power output for each individual buoy for all combinations of wave frequencies and angles supplied to it. It calls the arraySubmergedSphere function to determine the hydrodynamic coefficients and excitation forces between all the pairs of buoys, which allows it to adjust for factors such as positive and negative interference. A brief pseudo-code of the overall structure of the arrayBuoyPlacement can be seen in Algorithm 1.

There are two main ways to parallelise the wave layout model: evaluating individual layouts in parallel or evaluating individual wave frequencies in parallel. In this project the latter method is preferred, as it allowed for the evaluation of layouts in the expected order.

The arraySubmergedSphere function is called within arrayBuoyPlacement and returns 3 matrices that represent the hydrodynamic coefficients and excitation forces exerted by all the different pairings of buoys. As the number of buoys is configurable, these matrices change size as well. The size of the first two matrices is: (numBuoys * 3) x (numBuoys * 3). The size of the third matrix is: (numBuoys * 3) x numWaveAngles (which was 7).

The main body of the arraySubmergedSphere function consists of 6 nested for loops. Two loops iterate over the total number of buoys (the numSphere variable), with the other 4 loops iterating

Algorithm 1 arrayBuoyPlacement - a *very* brief overview of the general flow

```
% Output matrices:
%  ParrayW - average power output of the combined array
%  ParrayBuoyW - average power output of individual buoys
%  TetherForceBuoy - force applied to the tethers of individual buoys

procedure ARRAYBUOYPLACEMENT(array, siteOpts, lookUpTable)
    for all buoys do
        for all wave angles do
            for all wave frequencies do
                Determine power absorption of buoys without pairwise buoy interactions

    for all wave frequencies do                          % This loop can be parallelised
        Call arraySubmergedSphere      % Determines the effects of pairwise buoy interactions
        for all wave angles do
            Determine power absorption of buoys with pairwise buoy interactions

    Collect information into output matrices
```

through the numApprox variable, which is the number of approximations that are performed. In total, the exact number of iterations is:

$$numSphere^2 * \binom{numApprox + 1}{2}^2 \tag{2.1}$$

The model uses a default value of 4 for numApprox, which was not modified for the duration of this project. In the example of a 2-buoy layout, which was used most often in this project, this for loop would iterate 400 times. For each iteration, exactly 4 values are calculated and placed in the coefAB matrix. After the for loops finish, the coefAB matrix is used to create the final 3 output matrices. Pseudo-code for this general structure is displayed in Algorithm 2.

Algorithm 2 arraySubmergedSphere - a *very* brief overview of the general flow

```
% Output matrices:
%  A, B, and Xw - represent the hydrodynamic coefficients and excitation forces

procedure ARRAYSUBMERGEDSPHERE(array, wave, frequency, waveNumber, numApprox,
flag2D)
    for l = 1:numSphere do
        for n = 0:numApprox-1 do
            for m = 0:n do
                for lam = 1:numSphere do
                    for nd = 0:numApprox-1 do
                        for md = 0:nd do
                            Fill 4 values in the coefAB matrix
                            % One value for each quarter of the matrix
                            % See Sections 4.1.1 and Figure 4.2 for more info

    Use coefAB matrix to create matrices A, B and Xw
```

2.2 2D Landscape scenario

The 2D Landscape is a scenario used very often during the entirety of this project to collect data, perform tests, benchmarks and compare results. This scenario consists of a 2-buoy layout, where

the first buoy is placed at the fixed position: (100, 0). The second buoy is then placed at intervals along a 200 x 200 grid and the combined power output is recorded for this position. The result is then graphed in a surface plot, seen in Figure 2.1, where the height of the (x, y) location in the grid represents the combined power output for a 2-buoy layout with 1 buoy at (100, 0) and the second buoy at (x, y).

The interval used was a length of 5m. This results in exactly 1600 layouts being evaluated and an equivalent number of data points in the surface plot. This resolution can be increased by decreasing the interval lengths, however this exponentially increases the number of layouts that require evaluation. An interval of 2m or 1m require respectively 10,000 or 40,000 layout evaluations. As these interval lengths result in an impractical number of layout evaluations, As the distinct features of the 2D Landscape scenario are very evident with an interval length of 5m, this resolution is used for all future 2D Landscape plots.

This scenario helps to highlight the positive and negative interference that occurs. The positive interference is the ridge that appears before the power settles into a plateau. The highest power output along this ridge is 2.532e+05 W. The negative interference is the valley along the x-axis, with the lowest power output being 2.204e+05 W. As a reference, the average power output of 2 buoys in isolation is 2.446e+05 W, with a single buoy outputting 1.223e+05 W in isolation. The presence of positive interference allows the 2 buoys to output 3.5% more power than normal.

The hole centred around the point (100, 0) represents invalid layouts. Buoys must be at least 50m away from each other, so any layout that violates this safety constraint is not evaluated and returns a power output of -1.

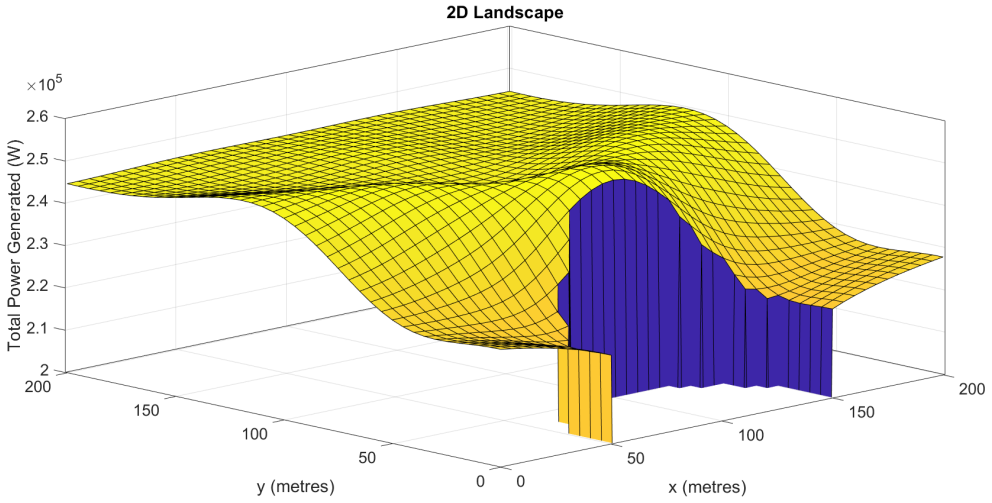


Figure 2.1: Surface plot of the 2D Landscape scenario. The height at any (x, y) position on the graph represents the combined power output of 2 buoys, where one is fixed at (100, 0) and the second, variable buoy is at position (x, y)

The MATLAB profiler was used to record the time spent on each function call and line of code in the wave layout model for the 2D Landscape scenario, which can be seen in Figure 2.2. gridSearchVariableBuoy was my personal code for calling the individual layouts of the 2D Landscape scenario, and while this function, the Main and arrayBuoyPlacement functions have very high total times, they have very small self times because the majority of time is spent in child function calls. However, arraySubmergedSphere has a very high self time.

Investigating the arraySubmergedSphere further reveals that the large majority of its time is spent making calls to the integral and quadgk functions. This can be seen in the MATLAB profiler results for this function in Figure 2.3.

Profile Summary

Generated 21-May-2018 01:01:43 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
Main	1	15221.873 s	0.049 s	
gridSearchVariableBuoy	1	15221.782 s	0.680 s	
arrayBuoyPlacement	1519	15221.009 s	92.188 s	
arraySubmergedSphere	75950	14928.801 s	6475.279 s	
integral	6531700	4295.243 s	82.732 s	
quadgk	6531700	4123.532 s	1769.061 s	
funfun\private\integralCalc	6531700	2758.931 s	349.851 s	
funfun\private\integralCalc>vadapt	6531700	2409.079 s	114.649 s	

Figure 2.2: MATLAB Profiler results for the 2D Landscape scenario, which evaluates exactly 1519 2-buoy layouts

arraySubmergedSphere (Calls: 75950, Time: 14928.801 s)

Generated 21-May-2018 01:01:46 using performance time.

function in file D:\Documents\repos\honours-wave-layout-optimisation\code\src\arraySubmergedSphere.m

Copy to new window for comparing multiple runs

Parents (calling functions)

Function Name	Function Type	Calls
arrayBuoyPlacement	function	75950

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
158	`int_mMmd2 = integral(funM, K*1...`	5620300	3711.816 s	24.9%	
157	`int_mMmd1 = quadgk(funM, 0, K*...`	5620300	3632.693 s	24.3%	
173	`int_mPmd2 = integral(funP, K*1...`	911400	608.461 s	4.1%	
172	`int_mPmd1 = quadgk(funP, 0, K*...`	911400	595.600 s	4.0%	
280	`ndmd = fctrM(nd-md);`	30152150	494.099 s	3.3%	
All other lines			5886.133 s	39.4%	
Totals			14928.801 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
integral	function	6531700	4295.243 s	28.8%	
quadgk	function	6531700	4123.532 s	27.6%	

Figure 2.3: MATLAB Profiler results for the arraySubmergedSphere function during the 2D Landscape scenario

2.3 Further experiments

Two additional experiments were performed to further understand the model. The first experiment, the line test, involved graphing the power output with a fixed buoy at (0, 0) and the variable buoy moving in only one direction. The second experiment was a simple time test to see how long different layout sizes took to evaluate on my personal laptop and the OptLog2 remote server.

2.3.1 Line test

To help understand the positive and negative interference further, tests were performed where the power output was measured along two directions: perpendicular and parallel to the wave direction, respectively called the y and x line tests.

The y line test was performed first. The first buoy is fixed at (0, 0). The second buoy is then placed at intervals of length 1 along the y-axis. This was graphed in Figure 2.4 up to a distance of 500m away from the origin point. As this direction was perpendicular to the direction of the waves, this graph clearly shows a cross section of the ridge shape that appears in the 2D Landscape scenario. Additionally it also shows a small dip in power immediately before the plateau, which is difficult to observe in the 2D Landscape image due to the lower resolution used (intervals of 5m).

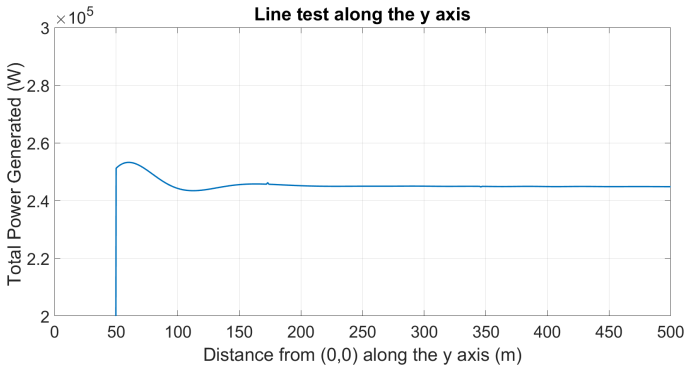


Figure 2.4: Line test along the y-axis. The first buoy is fixed at (0, 0) and the second buoy is placed at intervals of 1 along the y-axis.

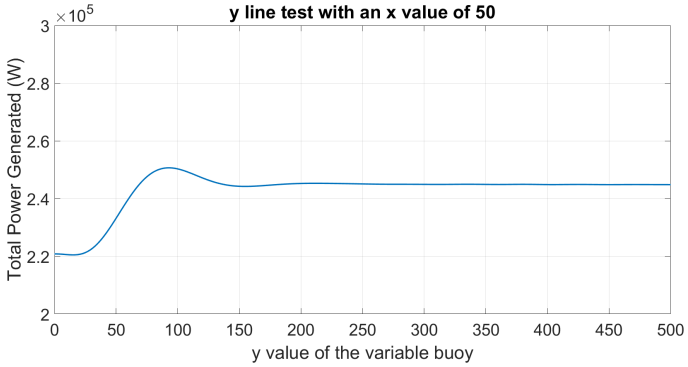


Figure 2.5: Line test with the variable buoy's x value set to 50 and moving parallel to the y-axis from (50, 0) to (50, 500) in intervals of 1. The fixed buoy is placed at (0, 0).

The same test was performed again except the except the variable buoy's x value is set to 50. Therefore the variable buoy moves from (50, 0) to (50, 500) in intervals of 1. The result, graphed in Figure 2.5, offers another cross section of the 2D Landscape scenario from a different x value but still perpendicular to the wave direction. The same properties can be seen here: the positive interference ridge occurring at the peak of the hill, the power dropping slightly below the plateau immediately afterwards, and then settling into the plateau once the buoys move far enough away from each other.

The line test was performed again except along the x-axis. Figure 2.6 shows this up to 1000m away. This direction is parallel to the direction of the waves, and as such receives the most negative interference. This test was mainly performed to see at what distance the effects of negative interference completely diminish.

Before the distance along the x-axis that the power plateaus could be discovered, an error was found. Distances of approximately 550m or more begin to throw warnings as the MATLAB function quadgk is not able to accurately approximate them. However, as this warning only appears for extreme distances and many layouts involve buoys within 200m apart this warning was ignored for the duration of this project. It simply prevented the line test from being graphed at further distances.

The random spikes in power that can be seen at the points x = 173, 346 and 519 are caused by a known error in the MATLAB code due to how the integral calculations are performed. Currently these are unavoidable and occur for very specific distances. These are avoided in the 2D Landscape scenario as none of the distances used result in these power spikes.

These line tests helped to uncover three properties. The first is that the a very small amount of negative interference occurs immediately after the positive interference ridge before the power settles into a plateau. This was not observable within the 2D Landscape scenario due to the lower resolution of 5m intervals that was used. The second property concerns one of MATLAB's

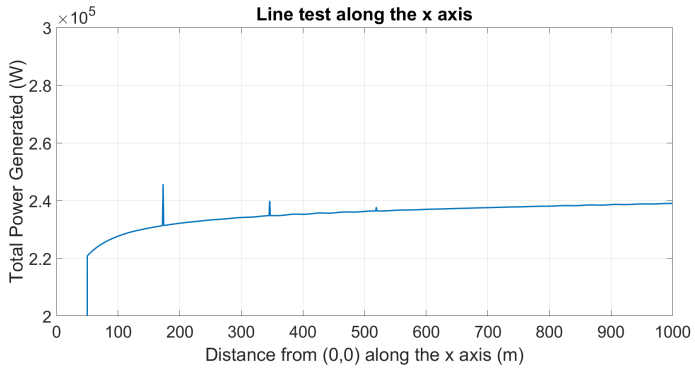


Figure 2.6: Line test along the x-axis. The first buoy is fixed at (0, 0) and the second buoy is placed at intervals of 1 along the x-axis.

functions that prevent proper approximations occurring when the distance between buoys becomes too great. However as this is only present in extreme distances, this would not affect normal usage and was thus ignored. The last property is a different error in the MATLAB code that causes incorrect, random power spikes to occur at specific distances. This error was also ignored for the duration of the project as the 2D Landscape scenario is not affected by this error.

2.3.2 Time test

A short test was performed to determine the average runtimes of layouts of different sizes on my personal laptop and the OptLog2 server. Varying numbers of buoys are placed in a fixed configuration and the wave layout model is run repeatedly 10 times on the layout in order to achieve a mean runtime. These tests were performed on the single-threaded version of the code without any parallelisation advantages. The results can be seen in Figure 2.7, which reveals that the runtime increases quadratically with the number of buoys.

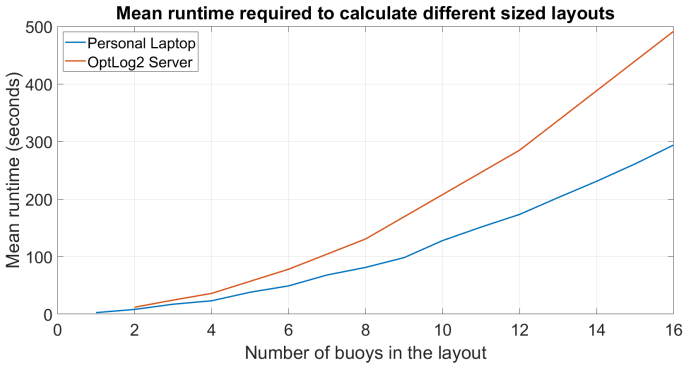


Figure 2.7: Mean runtime of the wave layout model for different layout sizes. Tested on my personal laptop and the OptLog2 server.

The quadratic increase in runtime is expected as Algorithm 2 contains two nested for loops that independently loop over each buoy in the layout. This results in the runtime increasing by at least a quadratic factor as the number of buoys increase.

2.4 Candidates for surrogate models

In deciding upon the candidates of code that we would try and model using surrogates, there were two main considerations:

1. That the candidate took up a large part of the average runtime

2. That the candidate performed independently of the size of the layout

The second consideration is an important one. Many function inputs or outputs change in size depending on the number of buoys used. Building a surrogate model for such a section limits it to only being useful for a single size of layout, which is not very useful. Ideally we would like to build a general model that can perform regardless of the layout size.

The second consideration severely limits what sections of code we can consider for candidacy. As an example, the arraySubmergedSphere function returns matrices that change size depending on how many buoys there are, violating the second consideration. However, looking inside this function reveals some sections that can be used.

The first notable section is the integral and quadgk functions which are used within the for loop of Line 7 shortly before Line 8 occurs in Algorithm 2. These two functions take up a significant portion of the total runtime, as revealed by the MATLAB profiler in Figure 2.3. In addition, the section of code that surrounds it deals with only pairwise interactions between buoys, so it is independent from the layout size. This candidate has been labelled the Integral/Quadgk candidate.

A second potential candidate is the series of for loops surrounding the Integral/Quadgk section, which is used to fill out the coefAB matrix. This is the section of code from Lines 2 to 8 of Algorithm 2. As this section deals with pairwise interactions it does not violate the second consideration and as it contains the Integral/Quadgk section this candidate also fulfils the first consideration. This candidate is labelled the CoefAB candidate.

A third candidate was also identified in the arrayBuoyPlacement function. This section of code occurs from Lines 2 to 10 of Algorithm 1 and uses the majority of calculations to construct the intermediate matrix that is used in Line 12 to create the final output matrices. This candidate is labelled the Parray candidate, after the main matrix filled in the section. However, this candidate is dependent on layout size. Additionally, as this candidate encompasses the majority of the code, it is almost directly predicting the expected power output. A very similar surrogate model was developed previously by Xia [12], Feng [2], and Li [4] had very limited success. For these reasons, this candidate is not explored further.

Chapter 3

Integral/Quadgk Candidate

3.1 Building the initial model

3.1.1 Analysis of candidate code

The Integral/Quadgk candidate code deals with pairwise buoy interactions and is comprised of two MATLAB functions: integral and quadgk. These two functions are used together twice, as shown in Figures 3.1 and 3.2, the build the int_mMmd and int_mPmd variables respectively. The MATLAB profiler revealed that the int_mMmd section is used the most as it is called over 6 times as much as the int_mPmd section. As such, we focus on the int_mMmd section first as our target for the first surrogate model.

```
funM = @(x)x.^(n+nd).*(x+K)./(x-K).*exp(x*(z_l+z_lam)).*besselj(mdmm, x*Rlaml);
int_mMmd1 = quadgk(funM, 0, K*1.5, 'Waypoints', [K*0.7, K+0.1*K*1i, K*1.25]);
int_mMmd2 = integral(funM, K*1.5, inf);
int_mMmd = int_mMmd1 + int_mMmd2;
```

Figure 3.1: Screenshot of the first half of the Integral/Quadgk candidate section in the MATLAB wave layout code. This section builds the int_mMmd variable.

```
funP = @(x)x.^(n+nd).*(x+K)./(x-K).*exp(x*(z_l+z_lam)).*besselj(mdpm, x*Rlaml);
int_mPmd1 = quadgk(funP, 0, K*1.5, 'Waypoints', [K*0.7, K+0.1*K*1i, K*1.25]);
int_mPmd2 = integral(funP, K*1.5, inf);
int_mPmd = int_mPmd1 + int_mPmd2;
```

Figure 3.2: Screenshot of the second half of the Integral/Quadgk candidate section in the MATLAB wave layout code. This section builds the int_mPmd variable.

The int_mMmd half of the Intergral/Quadgk code has 7 inputs:

- n - range of 0 to 3 inclusive (based off number of approximations)
- nd - range of 0 to n inclusive (based off number of approximations)
- K - wave frequency squared / gravity
- z_l - height of the first buoy (always -8 in our model)
- z_lam - height of the second buoy (always -8 in our model)
- mdmm - range of -3 to 3 inclusive (based off number of approximations)
- Rlaml - euclidean distance between the two buoys

The quadgk and integral function outputs are combined together to create the final output variable: int_mMmd. However we split this into their separate function outputs: int_mMmd1 and int_mMmd2 for the quadgk and integral functions respectively. Additionally, as quadgk returns a complex number, we split int_mMmd1 into a real and imaginary component as the regressor models from the Python library sklearn do not accept complex numbers. The final output labels are: int_mMmd1_real, int_mMmd1_imag, int_mMmd2.

3.1.2 Data collection

A dataset for this candidate was built by running the 2D Landscape scenario and recording the input and output values immediately after the quadgk and integral functions finished. These values are then saved inside a .csv file. The generated dataset consists of 5,620,300 rows and 10 columns.

The number of rows should be directly equivalent to the number of times that the quadgk and integral functions are called, which we can confirm is correct by using our earlier MATLAB profiler results in Figure 2.3.

An additional note is that the for loop that surrounds the Integral/Quadgk section actually executes exactly 400 times for each arraySubmergedSphere call for a 2-buoy layout. arraySubmergedSphere is called once for every wave frequency used, so a single layout sees 50 calls to arraySubmergedSphere. As exactly 1,519 layouts are fully evaluated (this is excluding invalid layouts that violate the safety distance constraint and skip evaluation), we should expect:

$$\begin{aligned} &\text{number of layouts} * \text{number of wave frequencies} * 400 \\ &= 1,519 * 50 * 400 \\ &= 30,380,000 \text{ calls to Integral/Quadgk section} \end{aligned} \tag{3.1}$$

This would result in 30 million rows in the generated dataset. This is not the case though, as the wave layout model uses memoization as a speed up mechanism, where all previously calculated int_mMmd values are stored in a map and retrieved again if the same inputs are used again. This technique therefore reduces the number of calls to the integral and quadgk functions in a single arraySubmergedSphere to unique calls only, which massively reduces the number of calls to integral/quadgk.

Although this does not make the generated dataset entirely free of duplicate rows, as the scope of the memoization cache is local to the arraySubmergedSphere function. Subsequent calls to the function therefore begin with an empty cache, resulting in duplicate rows to be captured from different arraySubmergedSphere calls.

3.1.3 Training the model

The Python library scitkit-learn [7] was used to train the various machine learning models. Initially, a few different regressors were manually tested: Linear Regressor, SVM Regressor and Random Forest Regressor (RFR). The results of these regressors can be seen in Table 3.1.

The R^2 score is used as an indication of model accuracy. This score represents the coefficient of determination and returns a number between 0 and 1 where the closer to 1, the more accurate the model is. Negative scores are also possible in situations where the model is especially terrible.

When training these models, random sampling was used to create a training set comprised of 80% of the original dataset and a testing set comprised of 20% of the dataset.

The RFR was quickly discovered to have the highest R^2 score. The parameters for this regressor was experimented with further to see how much the R^2 score could be increased. The main parameters tested was the n_estimators and max_depth parameters. n_estimators defined the number of trees used in the forest and max_depth defined the maximum depth of these trees. By default n_estimators is 10 and the max_depth is None, which means the max depth is unbounded.

The results of this testing can be found in Table 3.2, which revealed that increasing the number of estimators only marginally increased the R^2 score at the expense of exponentially increasing training time and model sizes. In addition, it was found that the max_depth parameter should generally be left unbounded.

The R^2 score was used as the initial indication of accuracy. The problem with relying on this score alone is that it compresses a lot of information into a single number. Questions such as

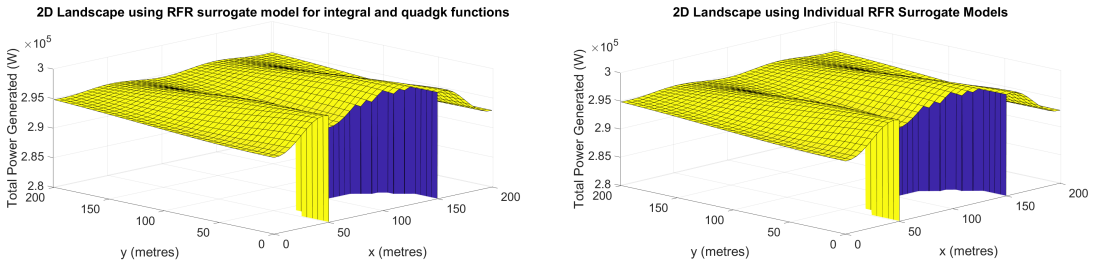
Regressor model	Parameters used	R^2 score
Linear Regressor	Default	0.056692439
RFR	Default	0.99981137
RFR for Individual Output Labels	Default	0.999884322 0.999732056 0.999729076
Support Vector Regressor	Default	-13.917976462 -16.089903219 -55.434781108

Table 3.1: R^2 scores from a few manually tested regressor models. Some models provide support for multiple output labels, these ones have only a single R^2 score. For models that do not support this, a different model is trained for each output label, so 3 R^2 scores are reported.

Parameters	R^2 score
n_estimators = 3, max_depth = None	0.999792216
n_estimators = 5, max_depth = None	0.999797773
n_estimators = 10, max_depth = None	0.999796112
n_estimators = 15, max_depth = None	0.999804942
n_estimators = 20, max_depth = None	0.999807199
n_estimators = 50, max_depth = None	0.999803787
n_estimators = 10, max_depth = 3	0.588284353
n_estimators = 10, max_depth = 5	0.932804796
n_estimators = 10, max_depth = 10	0.991479552
n_estimators = 10, max_depth = 20	0.999606931

Table 3.2: R^2 scores from testing different parameters for the multi-output label RFR.

whether or not this surrogate model is able to predict the negative and positive interferences is not something we can easily determine from normal accuracy measures such as the R^2 score or mean squared error. In order to understand the accuracy further, the surrogate model of the default RFR was used inside the wave layout model for another 2D Landscape scenario. The resulting surface plot is included in Figure 3.3a.



(a) A single RFR model

(b) 3 separate RFR models, one for each output label

Figure 3.3: 2D Landscape using either a single, or 3 separate RFR models predicting the 3 output labels for the Integral/Quadgk code section. These regressors used only the default parameters (n_estimators = 10 and max_depth = None)

Unfortunately, the resulting landscape is not at all similar to the original 2D Landscape in Figure 2.1. The slopes of the positive and negative interference areas are completely lost and and the final power output is much higher at approximately 2.95e+05 W in comparison to the highest power of 2.5e+05 W in the original 2D Landscape.

Using 3 separate RFRs individually trained for each output label did not prove much better. Included in Figure 3.3b, this plot also shares the exact same shape to Figure 3.3a with no

improvements towards modelling the original 2D Landscape shape.

3.2 Investigating the effects of the Integral/Quadgk values

It is highly suspicious that the RFR model, which boasted a very high R^2 score of 0.9998, was not able to model the 2D Landscape figure even remotely. Further investigations were performed, which led to the discovery that the wave layout model is very sensitive to random errors in the Integral/Quadgk section. This section details the investigation process that led to that conclusion.

3.2.1 Grouping by K values

The first investigation performed was to see if different wave frequencies were harder to model. This model used 50 unique wave frequencies, and as the surrogate model input K is based directly off of the wave frequency (K = wave frequency squared / gravity), the dataset was split and grouped by unique K values.

A multi-output RFR using default parameters was then trained on each of the split datasets in the same manner as described previously: with a randomly sampled 80:20 split between training and test set. The R^2 score was recorded for each model and graphed in Figure 3.4.

This experiment revealed a pattern where smaller wave frequencies are easier to predict, though only slightly as the difference in R^2 score is very small, where the minimum is 0.9995 and the maximum is 0.9999.

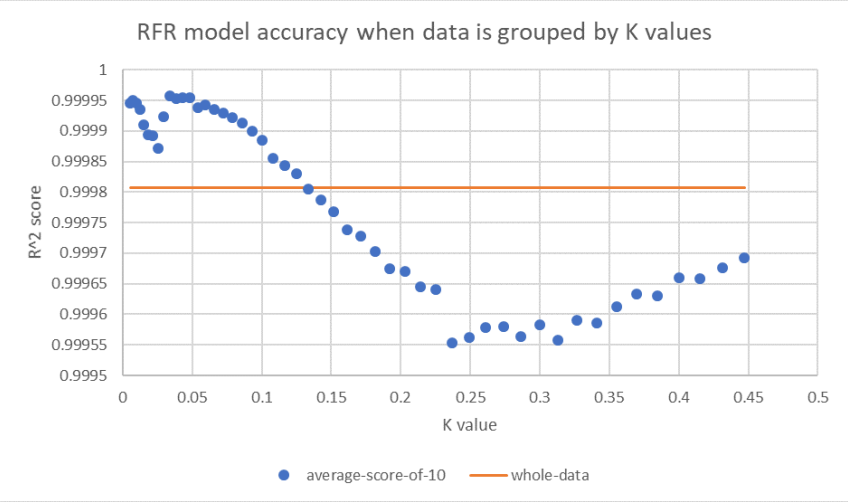


Figure 3.4: R^2 scores when a RFR is trained using only data from a specific K value. Due to the randomness of how the training and test sets are split, this process was repeated 10 times for each K value to achieve a mean R^2 score. The orange line represents the R^2 score achieved when using the full dataset.

3.2.2 Using decision trees to find problem parameters

The next experiment was the use of decision trees to see if there were any other parameters which resulted in higher errors. The dataset was sorted by the mean absolute error (MAE) and the top 10% were labelled *top10%*. The remaining were simply labelled *other*.

We then used the tool WEKA [10] to build decision trees using REPTree, a type of decision tree learner. We use this specific learner so that we can limit the depth of the decision tree in order to analyse it more easily. From there we create decision trees with depths from 1 to 6.

Depths of 1 to 4 were not interesting, as the tree simply labelled all the data as *other* because it would always be 90% accurate if it did so.

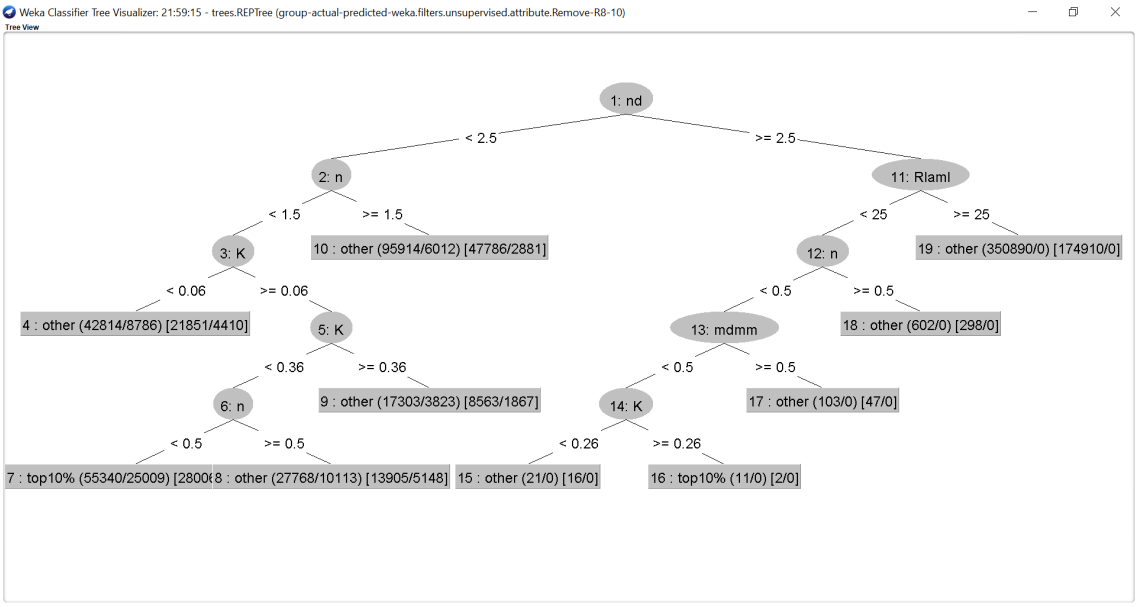


Figure 3.5: Decision tree learner of depth 5 attempting to classify the highest 10% of inaccurate predictions from the Integral/Quadgk RFR surrogate model

Depths of 5 to 6 are included in Figures 3.5 and 3.6. The decision tree learner appears to mainly use 3 of the input labels: n, nd and K in order to classify the majority of the *top10%*. In addition, this class seems to predominantly occur for K values of ¿= 0.06 and ¡ 0.36. This somewhat correlates with the previous experiment, the grouping by K values, as this region covers a large portion of the less accurate K values.

Approaching deeper depths begins to include more input labels, as Figure 3.6 shows, however these are only used to predict a small number of the *top10%* label so evidently the learner believes they are less influential in difficult predictions.

The results of this decision tree learner is not to be taken too seriously though, as the accuracy for the depths of 5 and 6 were 90.847% and 91.801% respectively, which is not much higher than the flat 90% achievable by only guessing the *other* label. This experiment mainly serves to help to identify parameters involved in difficult predictions and combined with the previous experiment of grouping by K values it appears that the K input label, the wave frequency, is a major factor.

3.2.3 Modifying the original values

The next two experiments were performed to observe the effects of modifying the original integral and quadgk values to find out how sensitive the final power output values are to changes in these values. This experiment involved modifying the output of the original integral and quadgk functions by a percentage. The 2D Landscape figure was then graphed using these modified numbers. The percentages used were: 50%, 60%, 80%, 90%, 95%, 99%, 101%, 105%, 110%, 120%, 140%, 150%, and 200%.

Three figures are used that highlight the main effects of modifying the integral and quadgk values. Figure 3.7 is an overlay of the original 2D Landscape and the 80% and 120% versions. Multiplying these values by a scalar simply scales the power accordingly, so that the 2D Landscape plot shifts along the z axis. Increasing the integral and quadgk values reduces the power output, shifting the final plot negatively along the z axis. The reverse occurs for decreasing the integral and quadgk values, with the graph shifting positively along the z axis.

For smaller scalars the shape remains very similar to the original 2D Landscape, however for much larger extremes the shape begins to change. Extremely small integral and quadgk values results in the 2D Landscape beginning to lose its shape and becoming flatter, as seen in Figure 3.8a. Extremely large integral and quadgk values have their shape exaggerated instead, as seen in Figure 3.8b.

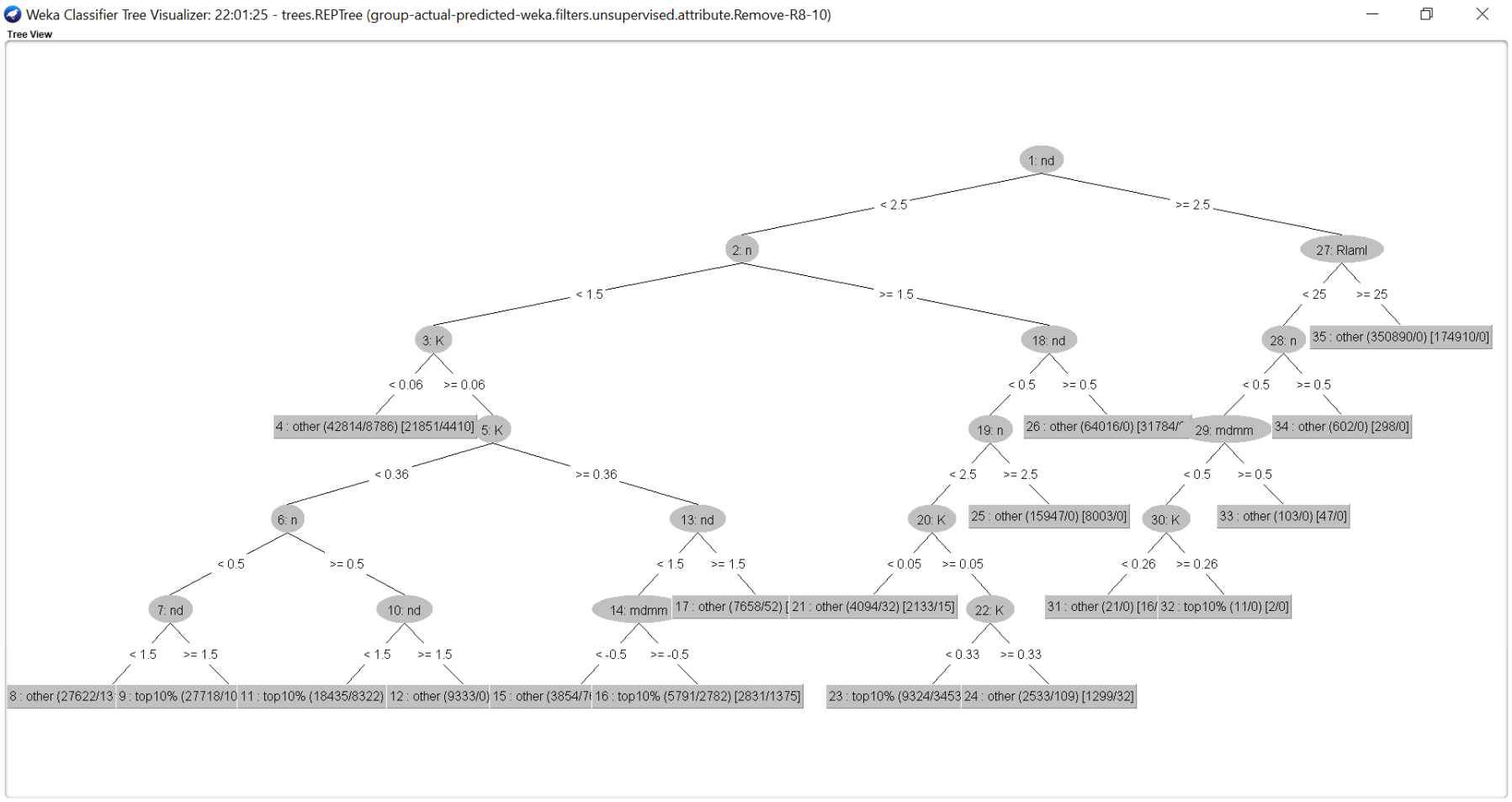


Figure 3.6: Decision tree learner of depth 6 attempting to classify the highest 10% of inaccurate predictions from the Integral/Quadgk RFR surrogate model

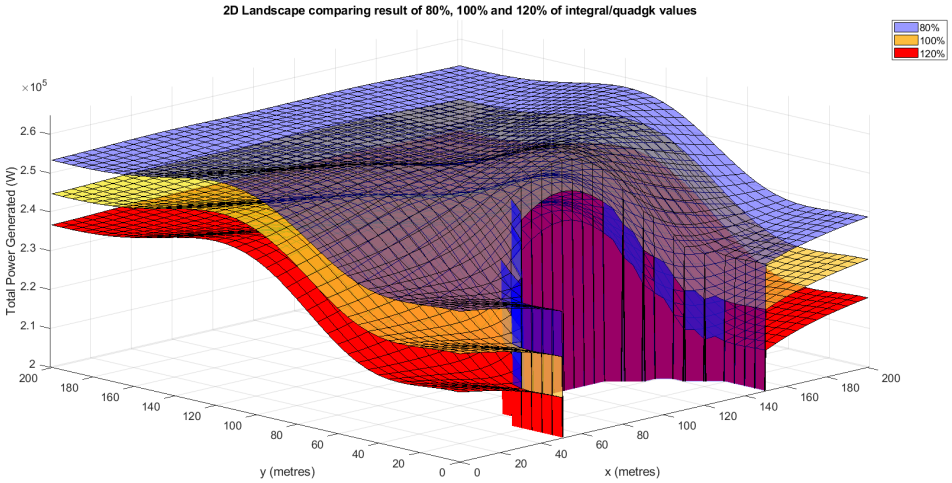
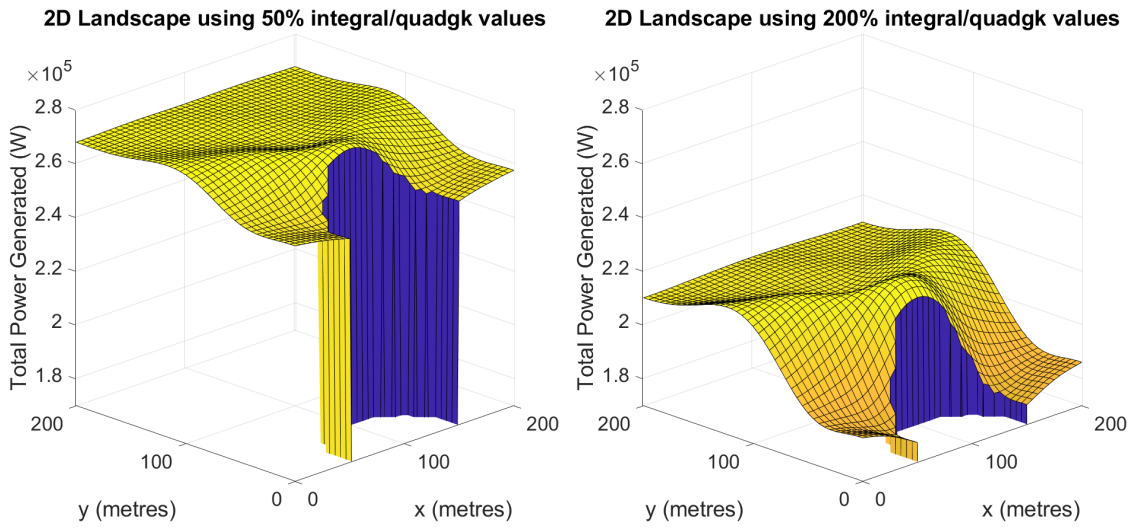


Figure 3.7: Three separate 2D Landscape figures overlaid on top of each other. The blue landscape shows a 2D Landscape where the integral and quadgk values are multiplied by 80%, the yellow one is the original 2D Landscape, and the red one has the values multiplied by 120%.



(a) 50%

(b) 200%

Figure 3.8: 2D Landscape where the integral and quadgk values are multiplied by 50% and 200%. Note that the z axis limits are different in order to fit the graph in the view, but the range is kept the same so the figures are not stretched.

This experiment showed that multiplying the integral and quadgk values by a flat scalar does not change the shape too much. In fact, reducing them to 50% of their original values only raises their power outputs to approximately 2.7e+05, which is considerably less than those given in the 2D Landscape using the RFR surrogate model.

3.2.4 Gaussian distribution of integral and quadgk values

This is the second experiment analysing the sensitivity of the integral and quadgk values. Instead of multiplying their values by a scalar, a Gaussian distribution is used instead.

Using the dataset generated in Section 3.1.2, the mean and standard deviation were calculated

for each of the three output labels. These values can be seen in Table 3.3.

Output label	Min	Mean	Max	Standard deviation
int_mMmd1_real	-0.1257	-0.0013	0.0413	0.0109
int_mMmd1_imag	-0.1443	-0.0011	0.0581	0.0103
int_mMmd2	-0.0118	3.9105e-04	0.0837	0.0047

Table 3.3: Minimum, mean, maximum and standard deviation of the three output labels of the Integral/Quadgk section using the data collected in Section 3.1.2.

The Gaussian distribution is applied by taking the original values and adjusting it by a random number normally distributed using the standard deviation. In MATLAB this code appears as follows:

$$\text{int\_mMmd1\_real} = \text{int\_mMmd1\_real} + (\text{randn}(1) * \text{standard\_deviation});$$

The randn(1) method in MATLAB returns a random floating point number normally distributed around a mean of 0 with a standard deviation of 1. Multiplying this number by the standard deviation gives a random number normally distributed around a mean of 0 with the intended standard deviation.

Gaussian test	Min	Mean	Max	Standard deviation
Original (no gaussian)	2.2048e+05	2.4209e+05	2.5323e+05	8.2818e+03
Gaussian 100%	-2.3439e+06	6.7541e+04	1.5150e+07	5.1254e+05
Gaussian 10%	-4.1160e+08	9.8496e+05	2.1739e+08	1.3478e+07
Gaussian 1%	2.0520e+05	2.4398e+05	2.9339e+05	1.1868e+04

Table 3.4: Minimum, mean, maximum and standard deviation of the power outputs of the 2D Landscape scenario using a gaussian distribution on the integral/quadgk outputs with different levels of standard deviation.

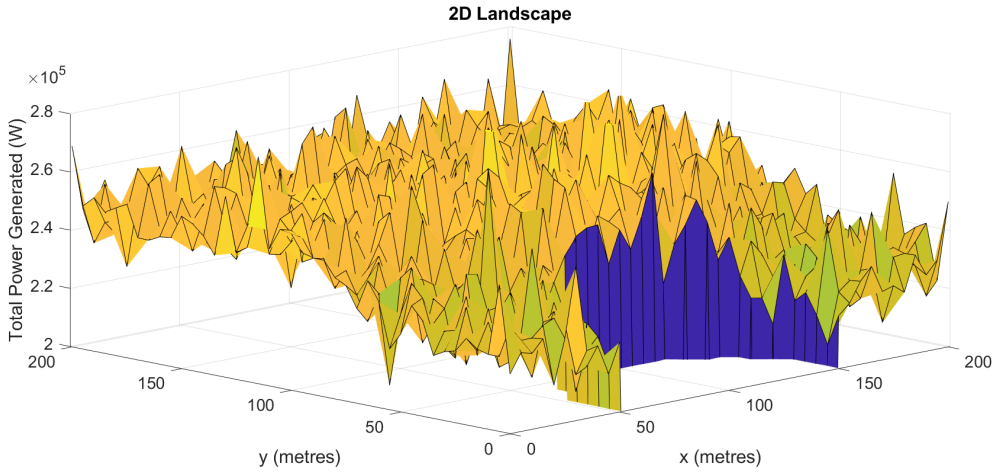


Figure 3.9: 2D Landscape figure drawn by modifying the original integral and quadgk values using a normal distribution with a standard deviation of 1% of the original standard deviation.

This was used in the 2D Landscape scenario using the original standard deviation, however the final power output values were very extreme, resembling nothing of the original shape. A similar result occurred using a standard deviation value that was 10% of its original value. Finally, trying a standard deviation 1% of its value returned an observable graph, shown in Figure 3.9. As a

comparison of the extreme power outputs of these tests, Table 3.4 displayed these values. Note that these values exclude the -1 power values that result from invalid layouts.

This experiment shows that making adjustments to the integral and quadgk function outputs based off their standard deviation results in extreme power outputs. The standard deviation was forced to be 1% of its original value in order to achieve a shape close to resembling the original 2D Landscape. Conversely, the previous experiment showed that altering the integral and quadgk function outputs by a scalar factor did not extremely affect the power output. These two experiments show two different properties: uniform adjustment, and random adjustments. It appears that the wave layout model is extremely affected by random adjustments, perhaps explaining why the very high R^2 scoring RFR ultimately performed poorly as errors are not uniform and more likely to happen in random directions.

3.3 auto-sklearn hyperparameter tuning

We only manually tested a few Regressor models. However there are many more models in existence that we did not test. As it would be impractical to manually test them all, we employed the auto-sklearn tool [3] which automatically tests different machine learning models and automatically tunes their parameters as well.

We began by using the auto-sklearn tool only for the first output label: int_mMmd1_real. Auto-sklearn takes a long time to run as it must train many different types of models repeatedly, so the intent was to discover the optimal model for the first output label and then limit auto-sklearn to only tune parameters within that model for the remaining output labels. This would severely reduce the runtimes required for auto-sklearn as the full auto-sklearn does not need to be run for all 3 output labels.

Our final auto-sklearn ran using these parameters:

Total time limit: 8 hours Per run time limit: 300 seconds Output label: int_mMmd1_real

We based our per run time limit from the time it took to train a Random Forest Regressor with default parameters, which was 29.11 seconds. Initially we used a per run time limit at least twice this amount: 60 seconds. However as this dataset was relatively quicker to train on we increased the per run time limit to 300 seconds so that auto-sklearn could explore bigger models with longer training times.

Unfortunately this only returned a Random Forest Regressor model. The parameters it chose were also mostly default except for the number of estimators, which it set to 100. We had also discovered this model earlier on as shown in Table 3.2. This result seems to indicate that there may not be a better model suited to this dataset than a Random Forest Regressor, which we know has very limited success in modelling the shape of the 2D Landscape.

Chapter 4

CoefAB Candidate

4.1 Building the initial model

4.1.1 Analysis of candidate code

The CoefAB code section revolves around filling out the coefAB matrix. This section spans the series of for loops in the arraySubmergedSphere function, from Lines 2 to 8 in Algorithm 2. For the default number of approximations (4), the size of the coefAB matrix is a square matrix with width and height: 20 $*$ the number of buoys. For the 2-buoy example used in the 2D Landscape, the coefAB matrix would have size 40 $\times$ 40. This can be seen in Figure 4.1 where the coefAB matrix is first initialised.

```
num_nm = (1+numApprox)/2*numApprox;
num_lnm = numSphere*num_nm;

% Matrices identification
coef_AB = zeros(2*num_lnm);
```

Figure 4.1: Initialisation of the coefAB matrix. numApprox is the number of approximations (default value of 4). numSphere is the number of buoys in the layout.

1	2	3	1	2	3
4	5	6	4	5	6
7	8	9	7	8	9
1	2	3	1	2	3
4	5	6	4	5	6
7	8	9	7	8	9

Figure 4.2: Example of the order that numbers are filled in for the coefAB matrix. In each loop 4 numbers are filled. The 1 numbers are filled first, then 2, etc.

The coefAB matrix is then modified 3 different times. These are labelled Parts 1, 2 and 3 for convenience. Part 1 is a special case where elements are only placed in the diagonal positions of the matrix. Parts 2 and 3 are used to fill out the rest of the matrix. In both parts exactly 4 values are placed into the matrix. If we split the matrix into 4 quadrants: top-left, top-right, bottom-left and bottom-right, then one number is modified in each quadrant. These start at the top left corner of each quadrant and move left to right and then top to bottom in that order. An example of this is given in Figure 4.2 for a smaller sized matrix.

The matrix elements being modified only change at the end of each for loop iteration, so Parts 2 and 3 can modify the same elements within a single loop. In addition Part 2 only occurs inside an if statement, so it only is executed occasionally. However part 3 is always executed immediately before the for loop iteration ends.

4.1.2 Data collection

Part 1 is a special case that only touches a subset of the matrix, so this section is ignored in favour of Parts 2 and 3 which represent the general use case. As these two sections can both modify the same elements in a single loop, the combined change is recorded instead. The contents of the matrix are recorded at the beginning of the for loop and then the difference at the end of the for loop is measured. This difference is the one recorded during data collection. As 4 elements are

```
coef1 = coef_d*(-1)^m/(2*pi*ndmd*nm);

coef_AB(ind_lnm1,ind_lamndmdA) = coef_AB(ind_lnm1,ind_lamndmdA) + coef1*I11;
coef_AB(ind_lnm1,ind_lamndmdB) = coef_AB(ind_lnm1,ind_lamndmdB) + coef1*I21;
coef_AB(ind_lnm2,ind_lamndmdA) = coef_AB(ind_lnm2,ind_lamndmdA) + coef1*I12;
coef_AB(ind_lnm2,ind_lamndmdB) = coef_AB(ind_lnm2,ind_lamndmdB) + coef1*I22;
```

Figure 4.3: The last section where elements are modified in the coefAB matrix. This section is always guaranteed to execute.

modified each for loop, this forms 4 output labels: coef1A, coef1B, coef2A and coef2B, which are named after the index variables used to access them in the matrix as seen in Figure 4.3.

This code candidate uses more complex variables than the Integral/Quadgk section. Figure 4.3 is the final section where elements are placed inside the coefAB matrix. In this figure they use the variable coef1 and the variables I11, I12, I21 and I22 in order to change the element in the coefAB matrix. However, these variables can be decomposed further. For example, the coef1 variable is made up of the variables: coef_d, m, ndmd and nm. The coef_d variable is comprised of these variables: n, eps_m and w. This process of following the variables to their components was used until the input variables could not be decomposed any further. In the end, 12 inputs variables were found through this method:

- K - wave frequency squared / gravity
- a_l - radius of the first buoy (always 5)
- a_lam - radius of the second buoy (always 5)
- z_l - height of the first buoy (always -8)
- z_lam - height of the second buoy (always -8)
- beta_laml - the angle between the two buoys
- Rlaml - euclidean distance between the two buoys
- n - range of 0 to 3 inclusive (based off number of approximations)
- nd - range of 0 to n inclusive (based off number of approximations)
- m - range of 0 to 3 inclusive (based off number of approximations)
- md - range of 0 to m inclusive (based off number of approximations)
- eps_m - either 1 (if m == 0) or 2 (if m != 0)

The variables n, nd, m and md are the for loop iterators used in Lines 3, 4, 6 and 7 of Algorithm 2.

Similar to the Integral/Quadgk candidate, data collection was performed during a 2D Landscape scenario. One very important difference this time was the sheer size of the data that was collected. Equation 3.1 earlier showed the expected number of calls to the Integral/Quadgk section, which is the same as the CoefAB section as they both reside in the same area.

This was 30,380,000 million expected calls for the 2D Landscape scenario. The Integral/Quadgk section does not actually reach this number due to the use of memoization, but the CoefAB section does. As a result the generated dataset consists of over 30 million rows with 16 columns each (12 for the input labels and 4 for the output labels). This huge dataset took up approximately 2 GB in filesize when stored as a .csv file.

This huge dataset posed a very large problem. Simply loading in the dataset requires excessive amounts of main memory. Additionally, as the training time of machine learning models is linked to the dataset's size, training times were expected to be similarly excessive.

In order to help combat the huge filesize, the decision was made to strip the dataset of duplicate rows. This reduced the filesize to 1.22 GB, with 15,085,000 rows remaining.

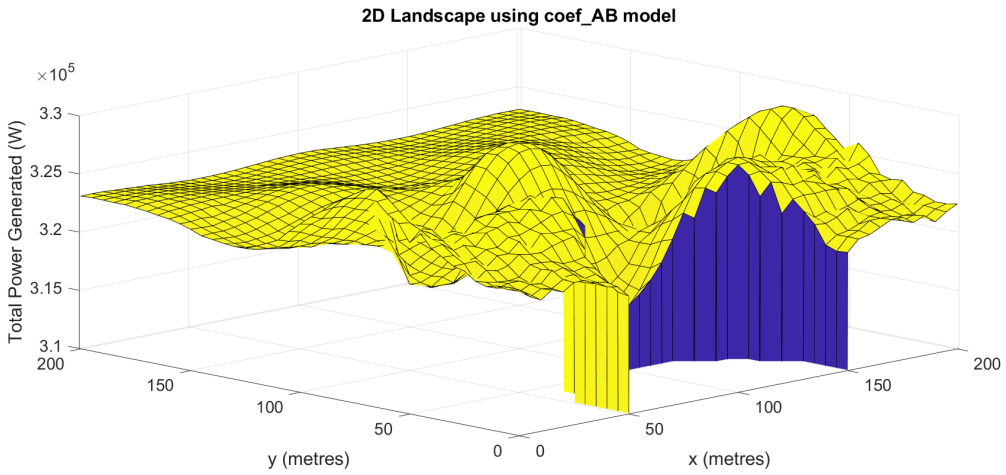


Figure 4.4: 2D Landscape surface plot generated using a RFR surrogate model with default parameters replacing the CoefAB section

4.1.3 Training the model

The first machine learning model tested was a Random Forest Regressor as they performed well previously with the Integral/Quadgk candidate. The results of using only default parameters and training a separate RFR model for each output label is contained in Table 4.1.

Output label	R^2 score	Training time (seconds)
coef1A	0.971426692	576.38
coef1B	0.922373554	641.11
coef2A	0.954129825	540.52
coef2B	0.929947523	474.65

Table 4.1: R^2 scores from training the default RFR model for each output label of the CoefAB section. The training times are also included.

These scores are much lower than the 0.9998 scores achieved by the Integral/Quadgk RFR surrogate, which was worrying as that model did not end up performing well. Additionally, the training times were very long, taking between 8-11 minutes for each model.

After these models were trained, they were used in the wave layout model and the 2D Landscape figure was generated again. The result is in Figure 4.4.

As indicated by the relatively worse R^2 scores, the 2D Landscape figure generated does not resemble the desired shape and also predicts much higher power outputs.

4.2 auto-sklearn hyperparameter tuning

The auto-sklearn tool was again used to help find any potential models that could perform better. However a large problem with this particular dataset is the huge filesize. The training time of 8-11 minutes for the default RFR model is 20 times the training time of the Integral/Quadgk dataset. To train a similar number of models in auto-sklearn would therefore require approximately 20 times the total run time, which is not feasible.

We made the decision to reduce the dataset size further through random sampling. We reduced the dataset to exactly 10% of its original size. The intent was to find the best performing model using auto-sklearn and then apply this model and its parameters to the full dataset.

Similar to our previous approach, only the first output label, diff1A, was focused on at first in order to reduce the runtime required to run auto-sklearn.

The smaller dataset was used to quickly train a Random Forest Regressor model with default parameters. This returned an R^2 score 0.530691889, much lower than the above 0.92 R^2 score achieved using the full dataset. However, the aim was to use this smaller dataset only as a guide so that the optimal model and parameters found for this dataset could be applied to the full dataset.

This smaller dataset took 72 seconds to train. Based off of this, the initial per run time limit was set to a conservative 600 seconds. The total time limit was 8 hours. This was run on both my home computer and the OptLog2 server as there were some issues at the time with using auto-sklearn on the remote OptLog2 server.

On my home computer I had to perform the random sampling again to generate another 10% sized coefAB dataset as the one stored on the remote OptLog2 was inaccessible due to limitations in download speed. As a result, the scores differ slightly due to the different random sampling. The default RFR model on my home computer achieved an R^2 score of 0.524222 and took 97 seconds to train.

On the OptLog2 server, auto-sklearn returned a Random Forest Regressor with mostly default parameters. Similar to the Integral/Quadgk section, the main parameter change was an n_estimator value of 100 instead of the default 10. During the 8 hours auto-sklearn tried 63 models in total, with 33 successfully tested and trained. From the remaining models tried, 26 failed due to exceeding the per run time limit and 4 failed from exceeding the memory limit. The R^2 score for this model was only 0.524222, which was actually a lower score than using only the default parameters. These were the parameters that auto-sklearn settled on however, and the difference in score is likely due to the randomness associated with training a Random Forest Regressor.

The auto-sklearn test on my home computer returned very similar results. After 8 hour of total runtime it had tried 74 different models. 31 were successfully trained, but 35 failed due to the per run time limit and 8 failed due to the memory limit. The final model it settled on was a RFR with 100 n_estimators again. This model had an R^2 score of 0.535564.

There were a high number of models that exceeded the per run time limit in both runs. In order to investigate the possibility that the per run time limit was preventing more complex models that could be performing better, the per run time limit was increased to 1200 seconds. The total time limit remained at 8 hours again.

Running this test on my home computer resulted in 45 different models being tried. 27 were successful, with 15 failing the per run time limit and 3 failing the memory limit. This test tried 29 less models than the previous test, which was due to the increased per run time limit allowing auto-sklearn to spend twice as much time on models and therefore having less time trying out further models. Unfortunately this test arrived at the same RFR model reached earlier.

The XGBoost regressor was also tested as well per the suggestion of a supervisor. This was tested by forcing auto-sklearn to only search for that specific type of model, allowing it to spend its entire total runtime discovering optimal parameters. Running this on the OptLog2 server with a total runtime of 8 hours and 600 seconds for the per run time limit results in an XGBoost model with an R^2 score of 0.309332 after trying 54 different models successfully.

Chapter 5

Results Discussion

It was unfortunate that the highly accurate surrogate model trained for the Integral/Quadgk section did not translate well into the wave layout model in the 2D Landscape scenario. This revealed an important observation: the accuracy of the surrogate model does not necessarily translate directly to the accuracy of the wave layout model. How sensitive the rest of the code is to the surrogate section is an important factor as well. The depth of the Integral/Quadgk section in the code also is likely a factor contributing to the high sensitivity, as small errors can propagate out and become bigger. As we discovered for the Integral/Quadgk section, the huge sensitivity is mainly towards random errors, which prevented the RFR surrogate model from performing satisfactorily.

The CoefAB section did not perform nearly as well as the Integral/Quadgk section did. The huge size of the dataset made this section especially difficult to work with, with two countermeasures taken to reduce the size: stripping duplicate rows and using only 10% of the remaining dataset sampled at random. Using auto-sklearn on this reduced dataset did not return any interesting results, as it returned a Random Forest Regressor which we already knew worked relatively well.

For both candidate code sections, auto-sklearn tended towards Random Forest Regressors. Unfortunately this did not prove too useful, as we had already discovered that these models worked well in both situations. The auto-sklearn tool only helped to reaffirm that there were no better models out of the ones auto-sklearn had support for.

It is also important to mention that auto-sklearn only has access to a limited number of machine learning models. As of this writing, it only supports 13 different regression models. Though these are popular and widely used machine learning models, this is obviously a limited selection of all of the machine learning models that are available. As such it is possible that there could be other machine learning models that could perform better.

However even if there were other models that could achieve a better R^2 score than the 0.9998 achieved by the RFR in the Integral/Quadgk section, these may still only have limited success inside the wave layout model due to its sensitivity to random errors. The main problem going forward is dealing with the model's sensitivity to these random errors.

Chapter 6

Conclusion

Two main code candidates were identified and investigated in the wave layout model: the Integral/Quadgk section and the CoefAB section. Testing several machine learning models and using auto-sklearn to help automatically discover different models lead to the discovery that the Random Forest Regressor performs very well on data gathered from these sections, especially the Integral/Quadgk section.

However this high accuracy does not translate well into the final power outputs of the wave layout model. Further investigation lead to the findings that the wave layout model was very sensitive to random errors in those particular code sections. As a result, even though the models were very accurate, small errors propagated outwards and lead to large errors in the final result of the wave layout model.

Attempting to increase the accuracy of the models further has very limited potential, as they are already very accurate, except for the CoefAB section which is relatively less accurate. The biggest challenge in the future is dealing with the extreme sensitivity of the wave layout model.

6.1 Future work

Further investigation into the CoefAB section similar to that performed for the Integral/Quadgk section was not performed due to time constraints. Future work can be done in this project by analysing this code candidate and testing how sensitive the wave layout model is to errors in this section.

There is also room for improvements in the surrogate model as the R^2 scores ranged from 0.92 to 0.97. Exploring different ways of collecting data for this section may also have some success. For example, Parts 2 and 3 of the CoefAB section as discussed in Section 4.1.2 were combined when collecting data for the output label. Separating these parts and building different models for each part may perform better as is decomposes the problem further.

Another interesting idea is to try and incorporate the results of the wave layout model into the machine learning model training. Creating some heuristic or collecting some information about the final result may help to train a model that is more likely to make predictions that lead to this final result.

Chapter 7

Bibliography

[1] B. Drew, A. R. Plummer, and M. N. Sahinkaya. A review of wave energy converter technology. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 223(8):887–902, 2009. ISSN 09576509. doi: 10.1243/09576509JPE782.

[2] Chenwei Feng. MSE 2017 Project Final Report. 2017.

[3] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf`.

[4] Mengyu Li. MSE 2017 Project Final Report. 2017.

[5] Bo Liu, Qingfu Zhang, and Georges G.E. Gielen. A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(2):180–192, 2014. ISSN 1089778X. doi: 10.1109/TEVC.2013.2248012.

[6] Alberto Moraglio and Ahmed Kattan. Geometric Generalisation of Surrogate Model Based Optimisation to Combinatorial Spaces. pages 142–154, 2011.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Constantina Pyromallis. Approximating the Power Absorption of PTO Settings of Wave Energy Converters using Surrogate Models for Optimisation Student ID : a1668648 Supervisor : Markus Wagner. 2017.

[9] Christopher J Rhodes. Current Commentary The 2015 Paris Climate Change Conference : COP21. 99:97–104, 2016. doi: 10.3184/003685016X14528569315192.

[10] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016. ISBN 0128042915, 9780128042915.

[11] X Wu. The interaction of water waves with a group of submerged spheres. *Science*, 1187(95): 165–184, 1995.

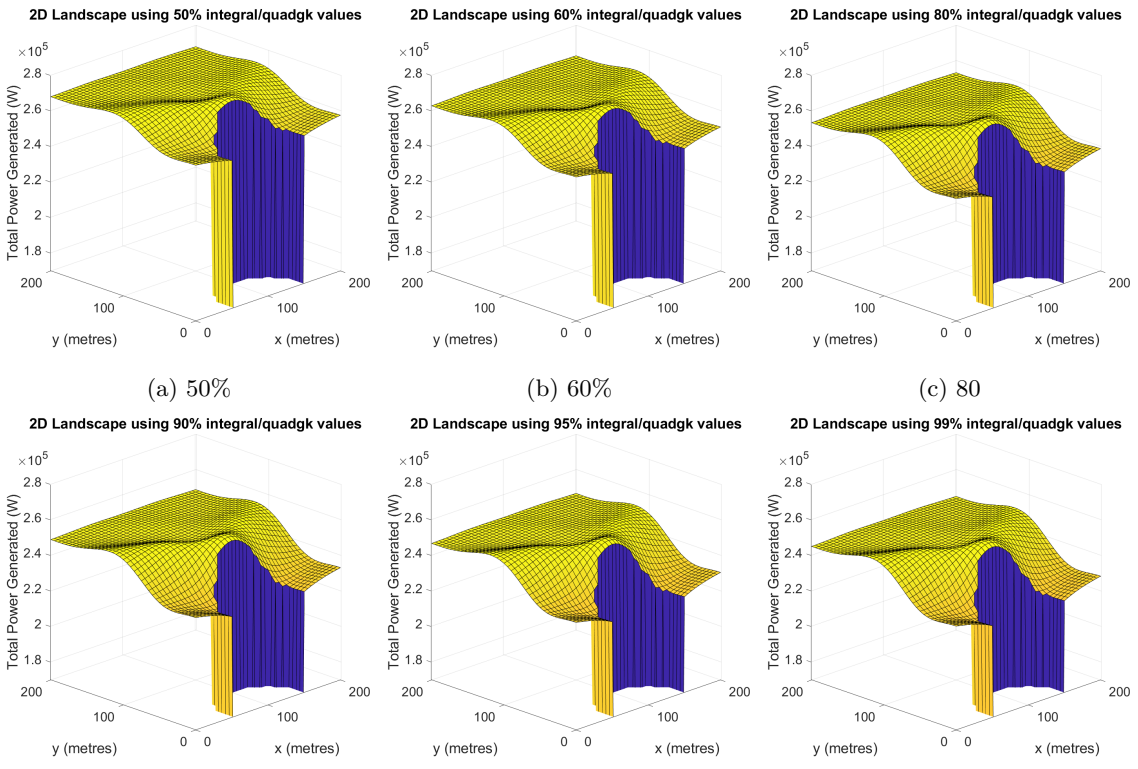
[12] Yuanzhong Xia. MSE 2017 Project Final Report. 2017.

Appendices

Appendix A

Modifying the integral and quadgk values

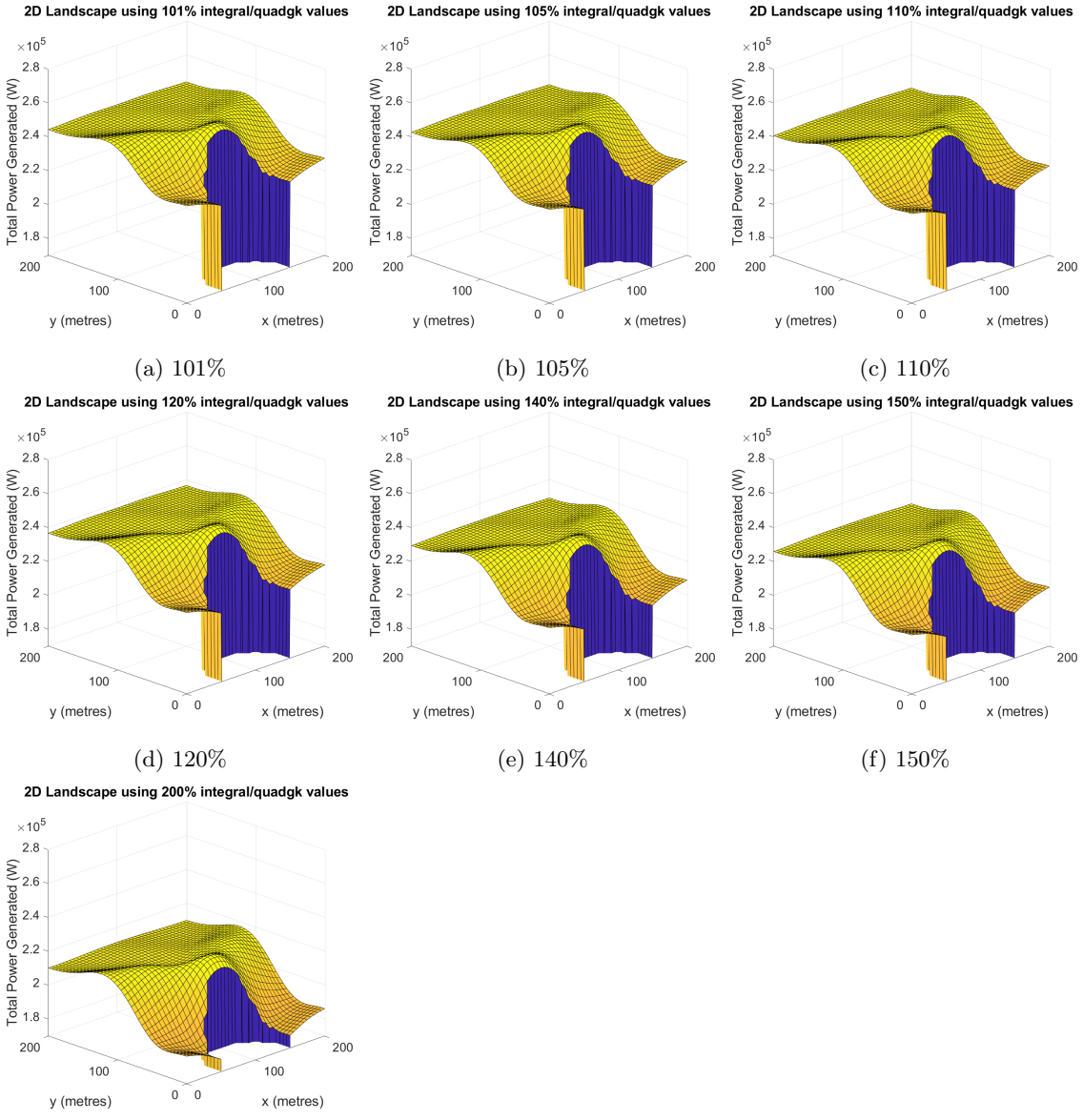
This appendix serves to store the many different figures that were a result of the experiment performed in section 3.2.3. The percentages used were 50%, 60%, 80%, 90%, 95%, 99%, 101%, 105%, 110%, 120%, 140%, 150%, 200%. For all of these figures the same z axis range was used. The standard 2D Landscape graph uses a z axis limit from 2e+05 to 2.6e+05. For those graphs who move outside these limits the z axis limits were adjusted for them without changing the range of 0.6e+05.



(a) 50% (b) 60% (c) 80

(d) 90% (e) 95% (f) 99%

Figure A.1: 2D Landscape where the integral and quadgk function outputs are multiplied by different percentages



(a) 101%

(b) 105%

(c) 110%

(d) 120%

(e) 140%

(f) 150%

(g) 200%

Figure A.2: 2D Landscape where the integral and quadgk function outputs are multiplied by different percentages