EMULATION OF EXPENSIVE SIMULATION MODEL FOR OPERATION AND MAINTENANCE OF OFFSHORE WIND FARMS

Jayanta Majumder, Iraklis Lazakis, Yalcin Dalgic, Iain Dinwoodie, Matthew Revie, David McMillan

University of Strathclyde, Glasgow, United Kingdom, (author email : jaymaj21@gmail.com)

Abstract

As wind farms move deeper offshore to tap into stronger and relentless winds, intense wind and wave conditions pose a great challenge in terms of their operation and maintenance (O&M). There are several factors that determine the profitability of offshore wind farms, and the most critical factors among them are the parameters on allocation of maintenance resources. These parameters interact with environmental factors and make it impossible to estimate profitability using simple formulas. On the other hand, existing simulation models, which describe the behaviour of wind farms by using mathematical models of wind, wave, and their effects on O&M, can be extremely detailed resulting in simulations being computationally very expensive. Depending on the number of scenarios to evaluate, it can take up-to several days to complete the computation. In order to address this difficulty, a statistical model fitting approach has been adopted to emulate the behaviour of the computationally expensive simulator. Neural networks, splines, and decision trees are combined to capture numerical and discrete variables and their influence on availability and profitability. This approach is useful because it affords quick exploration of the space of operating choices, which would be difficult to achieve by repeated simulations due to their computational expense. The performance results show that the statistical model can evaluate hundreds of scenarios per second, and the approximation error is low.

Keywords: Statistical Models, Neural Networks, Decision Trees, Splines, Offshore Wind Farms, Operation and Maintenance

1 INTRODUCTION

In offshore wind power production, moving deeper offshore, which offers both stronger winds and larger windfarms, entails better power yield. However, this comes with the additional challenges arising from the operation and maintenance aspects. Being further away from shore means that a substantial amount of time and logistics is spent in only transporting technicians and equipment to farms. While turbulent wind conditions cause turbines to fail more often, harsher wave conditions make turbines harder to reach and access for repairs. Maintainers need to keep repairing them regularly in order to avoid catastrophic failures and to keep the yield high.

The main parameters of the maintenance activity pertain to allocation and scheduling of resources. These allocation parameters interact with environmental factors and make it impossible to estimate the consequent profitability by a simple calculation. A detailed simulation model has been developed (described in [1] and [2]) that models the behaviour of wind farms hour-by-hour according to discrete and numerical parameters describing the process. Examples of such allocation parameters are (a) number of maintenance vessels (b) helicopter charter hours (c) jack-up vessel charter period (d) number of crew (e) shift start time (f) failure process parameters, etc. Based on these parameters and an environmental yield model, the simulator estimates several outputs that determine production and profitability of the simulated wind farm. The level of detail makes the simulation computationally very expensive, especially because the statistical estimation method (known as *Monte Carlo simulation*) requires several runs on each scenario. It takes several hours to simulate a single scenario and estimate the expected figures. If maintainers need to make quick decisions on the operating parameters, the running time can become a serious bottleneck. In order to address this problem, an approximation tool has been developed based on pre-computed runs of the simulator. We call this tool the *emulator*. The emulator allows quick evaluation of scenarios,

typically within a few milliseconds on a consumer grade computer.

The following sections describe in detail the methodology used in the emulator and the results obtained thereby.

2 SURVEY OF APPROXIMATION AND INTERPOLATION METHODS

Interpolation is the art of filling gaps between given data points using a reasonable continuous representation. The history of interpolation dates back to Issac Newton. His method is based on finite-differences and is equivalent to the Lagrangian interpolation. The degree of the polynomial involved in such interpolation goes up equally rapidly with the number of points. However, at high degrees polynomials have the undesirable property of making big swings between and beyond the data points. It is possible to get around that problem using multiple low-degree polynomial segments to interpolate/approximate different portions of datasets. Hermite and B-Spline interpolation methods [3] belong to this category. When dealing with multiple inputs (i.e. where the interpolated function is that of several variables), there is an additional challenge arising from the lack of structure in (i.e. scatter nature of) the given data points. It is only if the data points are organised in a structured n-dimensional lattice, that tensor product techniques like B-Spline and Hermite interpolation are applicable [3]. The problem with lattice structure in higher dimensions is that the local boundary conditions increase exponentially with the number of input variables, whereas the number of coefficients of a multi-variate polynomial of a given degree grow at a polynomial rate.

The methods mentioned so far were interpolation oriented, in that the continuous representation can be made to pass exactly through the data points. If this restriction is relaxed and the continuous form is allowed to approximate the data points, that opens up a new class of methods. In strict interpolation each data point gives rise to an equation in its own right and each such equation is solved for. Whereas in approximation, the data points collectively join hands to form an *approximation error* function which is then minimised in order to find a good approximation. In approximation the candidate continuous representation has a number of free variables. The idea is to have a function in two sets of variables - one set of variables are the inputs to the model, and another set of parameters to allow sufficient alteration of the function to make it take the shape dictated by the data. The process of fitting the approximation is that of tuning the second set of parameters so that the input-output relationship of the final function is as close as possible to a reasonable continuous representation passing through the data points. This can be formalised in the following mathematical statements. If $F(\mathbf{a},\mathbf{x})$ is an approximation architecture with shaping parameters $\mathbf{a}$ and input variables $\mathbf{x}$, in order to approximate a dataset:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), \dots \quad \quad)\}$,

our objective would be to tune $\mathbf{a}$ such that F predicts the y values as closely as possible. This objective can be achieved by minimising the following error function by varying $\mathbf{a}$

$$E(\mathbf{a}) = \sum_{i=1}^{n} \left(F(\mathbf{a}, \mathbf{x}_i) - y_i \right)^2$$

Here the function $F(\mathbf{a},\mathbf{x})$ is called the architecture of the approximation model. Once the parameters $\mathbf{a}$ gets tuned according to the dataset, the approximation function is said to be fitted and, with $\mathbf{a}$ being frozen F becomes a function of $\mathbf{x}$ that approximates the relationship between $\mathbf{x}$ and y.

It can be a substantially creative process to hypothesise the architecture that best approximates a given relationship. The terms in the approximation architecture can come from insights into the phenomenon, or it can come from studying shapes in the slices of the dataset. Fortunately, there are some universal approximation architectures that help with relieving the creative and investigative work. Two universal approximation architectures are *sum of radial basis functions* (RBF) [4] and *neural networks* [5]. The neural network architecture is a non-linear function that arose from the study of brain neurons, but can be understood entirely as a mathematical tool. The approximation function of a neural network has the following form:

$$y = \sigma_{output}\left(\alpha_k^{output} + \sum_{i=1}^{m} W_{ik}^{output} \sigma_{input}\left(\alpha_i^{input} + \sum_{j=1}^{n} W_{ij}^{input} x_j \right) \right)$$

Here $\alpha^{output}, W_{ik}^{output}, \alpha_i^{input}, W_{ij}^{input}$ are free shaping parameters that are tuned in the fitting process,

and $\sigma_{output}, \sigma_{input}$ are sigmoid shaped functions that bring non-linear shape into the architecture. There are mathematical theorems, which prove that the neural network architecture has universal approximating power, i.e. it can take pretty much any shape given sufficiently large size of the W^{input} and W^{output} matrices and correspondingly the α vectors.

A fitted neural network serves as a hyper-surface that approximately interpolates through the dataset. For higher dimensions, we can not visualise such hyper-surfaces entirely, but as a special case, when there are two input variables, the surface may be visualised as a 3d plot as in figure 1. The two input variables in this plot are *failure rate* and *number of repair vessels* and the output is *the natural logarithm of total revenue*.

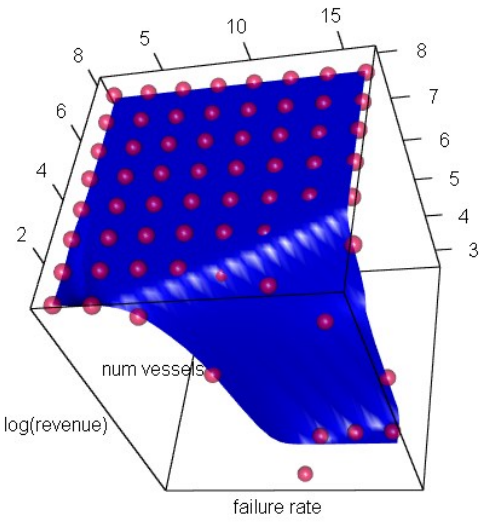

Figure 1. Neural network surface

3 COMBINING NUMERICAL AND DISCRETE INPUT VARIABLES

Many input variables of the O&M simulation are numerical, and so continuous function models can be used for them, but there are certain variables that are *categorical* (also called *ordinal*) representing a finite number of discrete possibilities on which no meaningful arithmetic can be performed. For such variables, we need to create a discrete lookup structure. A *decision tree* is commonly used for this purpose. Let us consider a concrete example. Table 1 shows four ordinal variables (given in the header row) and the values that they can take. The number of values that each variable can take are $3, 2, 2,$ and 4, and as such the total number of combinations we can get from them is $3 \times 2 \times 2 \times 4 = 48$. A decision tree for this would be a tree-like structure that bifurcates at 4 stages - one stage for each ordinal variable, and the number of bifurcations at each branching point is the number of values that the variable of that stage can take. The leaf-branches (i.e. the final branches) are the places, where we could place the output we want to look up for the combination of ordinal values that leads from the root of the tree to that leaf. Figure 2 shows a part of such a decision tree.

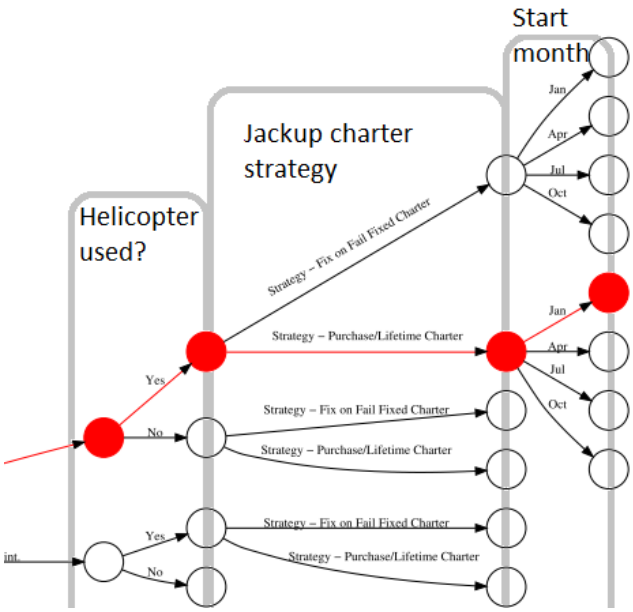

Figure 2. Decision tree (partial)

Table 1. Example of Categorical Variables

Ordinal Variable Name →	CTV Tech. Maintenance Schedule	Heli-copter Usage	Jack-up Charter Strategy	Start Month
Allowed Values →	Either corrective maintenance or scheduled maintenance	Yes	Fix on Fail Fixed Charter	Jan
	Scheduled maintenance after corrective maintenance	No	Purchase/Life time Charter	Apr
	Scheduled maintenance ONLY after corrective maintenance			Jul
				Oct

When a query is posed with a given combination of ordinal variables, one of the possible paths is activated (an example of such a path is shown in red in figure 2) and the output sitting at the leaf point on the path would be pulled out. Such outputs could be numbers if ordinal variables were all we had. Instead, in our application we have a mix of ordinal and numerical variables. So, at the leaf end of each branch we keep a neural network trained with the corresponding numeric combinations. As a special case, when there is exactly one numeric input in the model, we use a spline curve instead of neural networks. Thus, the statistical model takes the form of a combination of decision trees, spline curves, and neural networks. This model is created and stored in a data file, which is loaded and processed when answering interpolation queries. Consequently there are two tools in the emulator - one for creating the model data files, and the other for querying the model.

4 CASE STUDY

In an initial case study, there were 8 input variables and several dozens of output quantities in the dataset in question. Under this condition, evaluation of a single point query costs 3 milliseconds on a Pentium™ 2.4 GHz CPU. This is a satisfactory performance figure as it allows exploration of about 300 query points per second, which would have taken days to explore using the simulator. More important is the question of accuracy, since it is pointless to produce wrong results, no matter how fast. We demonstrate the accuracy of the trained neural networks using output comparison plots as in figures 3 and 4. In these plots, a perfect fit would lie on the $y = x$ line as shown in red. The scatter around the $y = x$ line represents the fit error. We also present the accuracy using Bland-Altman style plots in which the relative fit error is plotted against the corresponding values, as in figure 5. This dataset has about 4k points, so these plots might look too crowded to understand the distribution of errors. An error histogram (as in figure 6) can give better insight into the distribution of error.

Due to the high dimensionality of the input, it is impossible to visualise the complete fitted hyper-surfaces. However, in order to examine artefacts of fitting, we can look at one or two dimensional slices of the hyper-surface. Figure 7 shows one such plot, in which a single input variable is varied keeping the others constant. The red circles in this figure represent data points intercepted by that slice.

It was found that the *single-numeric-variable* case is very special because of a common workflow, in which the simulation runs are progressively refined in favour of the region of interest. In this workflow, an initial coarse set of simulation runs are used to span a broad range of the numeric variable. The emulator is then used to produce a continuous curve *through* the coarse point-set, which then is used for identifying regions of interest (e.g. turning points, steep rises and falls etc.) and additional simulations are scheduled to refine such regions. It was also found to be of interest to plot standard-deviations as bounds around mean-values using the single-numeric-variable interpolation feature. Figure 8 shows such a plot.

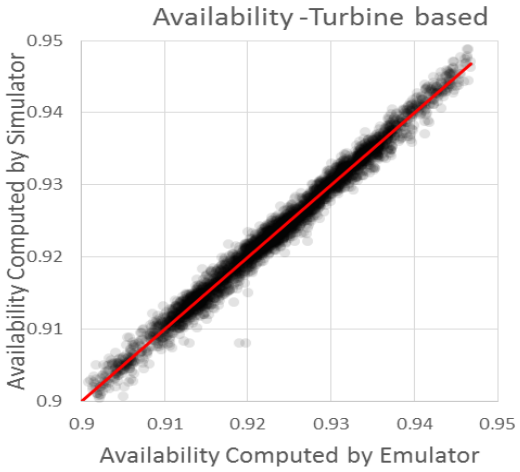

Figure 3. Emulated vs Simulated Availability

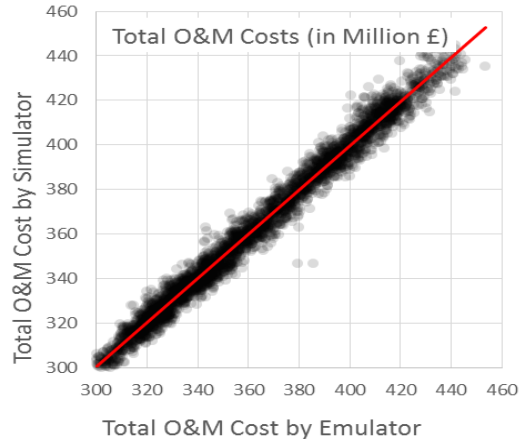

Figure 4. Emulated vs Simulated Total O&M Cost

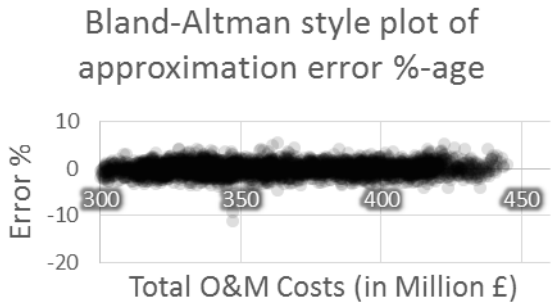

Figure 5. Total O&M Cost vs Neural Network Approximation Error

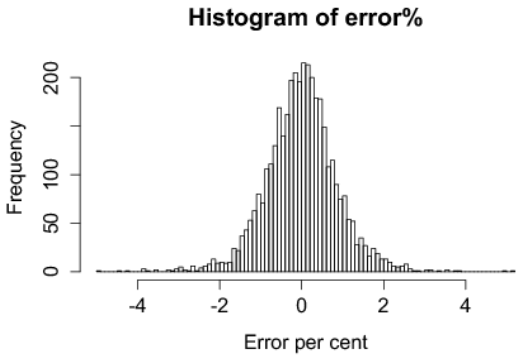

Figure 6. Histogram of Neural Network Approximation Error

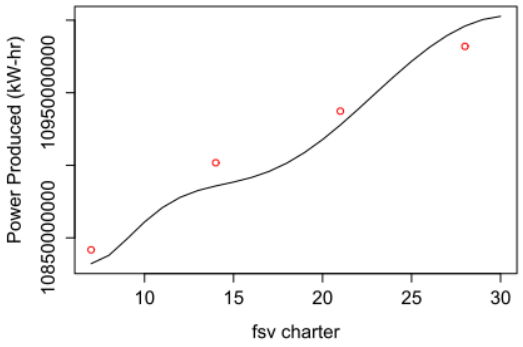

Figure 7. A Slice of the Neural Network Approximated Hypersurface

5 SOFTWARE DESCRIPTION

The core functionality of the emulator is written using R [6], a statistical computing language and environment. The user interface is written using a scripting language called Tcl/Tk [7]. As indicated earlier, following are the two functions of the emulator:

(a) Creation of a model from a given dataset.
(b) Querying the model.

This choice is provided by two buttons displayed when the tool is launched. Figure 9 shows a screenshot of these two buttons. A model can be created by choosing the *Create Model* button. Once that button is pressed, the tool lets the user choose the input data. The input data file is a specifically formatted excel spreadsheet, whose format is described in the user manual for the emulator. The tool then generates the model data in a folder as stated in the input file.

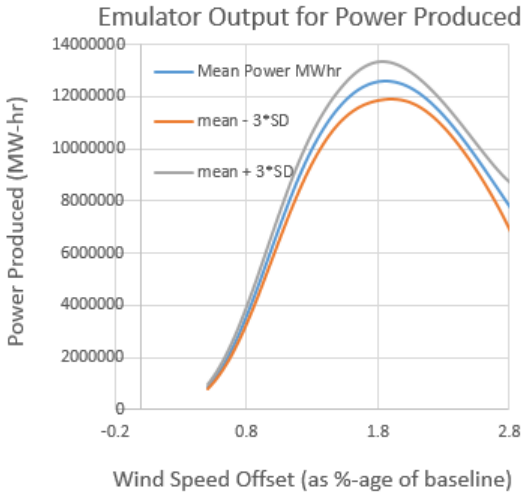

Figure 8. Plot of Wind Speed against bounds of Power Produced

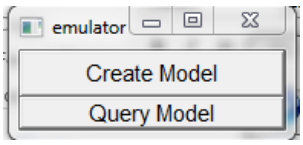

Figure 9. The Initial Screen of Emulator.

On choosing the *Query Model* button, the query screen is displayed. A screenshot of this screen is shown in figure 10. On entering the dataset name, username, and password, the user has to press the *Open Model* button to load the model. This will load the output variable names in the *Output Variable* list. The query points are entered in the *Query Points* box in a tab separated format. This is the format of data we get on clipboard when the data is copied from Microsoft Excel. On entering the query points and choosing an output variable, the user can press the *Submit Query* button to get the query evaluated. The evaluation results are copied into the clipboard in tab-separated format, which can then be pasted onto Excel for plotting or further analysis.

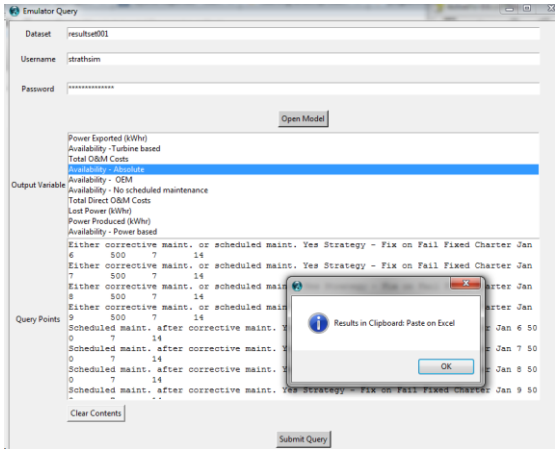

Figure 10. The emulator query screen

6 CONCLUSION AND SCOPE FOR FUTURE WORK

There are infinitely many ways of connecting a finite number of dots using continuous functions. Figure 11 illustrates this with a plot of several continuous curves going through 6 data points, each made with a different type of spline method. This tells us that we should treat the results from the emulator with caution and not take it as absolute truth. This is why one must use a separate dataset to validate a model after the model has been fitted using the *training* data-set.

Statistical models have been in use for a long time, and currently model-creation is usually carried out by specialists using versatile tools like SPSS, SAS, and R. The emulator is an attempt at packaging the functionality of model generation in such a way that the model is automatically generated from a simple input-output specification given by a spreadsheet. While this makes the model creation more user-friendly, it has its drawback that advanced validation tools and data pre-processing tools are not available on it. Future work may be done in that direction. It might also be worthwhile to revisit the wind farm simulation methodology with the aim of reducing computational cost (for example, by de-coupling and caching the weather/yield part, while simulating only the failure/repair events for O&M studies).

Further work might be directed towards modularising the emulator in a way that allows the model architectures and fitting algorithms to be specified by the user.

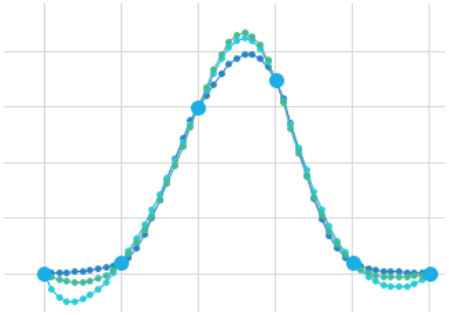

Figure 11. Multiple smooth functions interpolating 6 data points

ACKNOWLEDGEMENTS

The authors would like to express their appreciation to Scottish Power Renewables (SPR), Scottish and Southern Energy (SSE), and Technip for their financial sponsorship of this work. This work is funded through the Technology Innovation Centre (TIC), University of Strathclyde.

REFERENCES

[1] Yalcin Dalgic et al, "*Cost benefit analysis of mothership concept and investigation of optimum chartering strategy for offshore wind farms*.", 12th Deep Sea Offshore Wind R&D Conference, EERA DeepWind'2015, Trondheim

[2] Yalcin Dalgic et al, "*Advanced logistics planning for offshore wind farm operation and maintenance activities*", Ocean Engineering, 101(0):211-226, 2015.

[3] Gerald Farin et. al. Editors, "*Handbook of Computer Aided Geometric Design"*, ISBN: 978-0-444-51104-1, North Holland Pub., 2002

[4] C.M. Bishop, *Pattern Recognition and Machine Learning,* ISBN 978-0387310732, Springer, 2006.

[5] Simon Haykin, "*Neural Networks and Learning Machines: A Comprehensive Foundation*", 3rd Edn, ISBN: 978-0131471399, Prentice Hall, 2008

[6] R Core Team., "*R: A Language and Environment for Statistical Computing*", R Foundation for Statistical Computing, Vienna, Austria, 2014. http://www.R-project.org/

[7] John K. Ousterhout et al, *"Tcl and the Tk Toolkit"*, ISBN: 978-0321336330, Addison Wesley; 2nd edition (2 Sept. 2009)