Surrogate Modeling Approaches for Use with Prognostics Models Package (INTERN NOTES)

Elizabeth Hale

30 July 2021

1 Introduction

The Prognostics Models package offers several different physics-based models for systems such as batteries and valves [21]. While these models offer a high degree of accuracy and minimize computation time, they can still present computational challenges, especially when needed for repeated simulations such as in the inner loop of an optimization or when immediate, real-time updates are needed.

Surrogate models, also know as response surface models or metamodels, are commonly used to reduce the complexity of a model, though possibly at the expense of some accuracy. Taking a black-box approach to the problem, these models map inputs to outputs by fitting a response surface for the model, which is particularly useful when relationships between variables may be unknown or are computationally expensive to compute [14, 24]. Such models can be used to emulate the behavior of a physical system and in optimization.

Developing a surrogate model feature for generating quick, low-fidelity models for the Prognostics Models package could offer significant speedups in many applications. Specifically, the Data and Reasoning Fabric, which is working toward developing services to enable enhanced air mobility such as trajectory and system health services, may find this feature useful in performing trajectory simulations with the battery model to quickly provide updates on the capabilities of an aircraft battery.

This document contains a summary of some initial work toward developing this feature for the Prognostics Model package. In particular, we discuss features of different samplers and model types, as well as some initial thoughts on how the input and output of a such a model may be structured. We also include an overview of an initial implementation of a simple surrogate for the electrochemistry-based battery model through linear regression, and conclude with some thoughts on next steps toward the design of surrogates for the models in the package.

2 Background

There are three parts to generating a surrogate model: sampling the input variable space, training the model, and evaluating accuracy. In sampling, input values for training are sampled from specified ranges, and then used in simulation to create input/output pairs for training. After choosing a model type, associated parameters must be optimized based on the training data. Accuracy of the model is then quantified using a separate set of test data to determine the utility of the model [22, 24]. Here, we focus on common sampling methods for training data followed by the model cores.

2.1 Samplers

Sampling is an important part of the model development because sample vectors should accurately represent the input variable space. To achieve this, values should cover the space uniformly [24]. Many methods have been designed to provide such sampling, varying in complexity and effectiveness.

2.1.1 Random/Monte Carlo Sampling

Implementing Monte Carlo sampling is very straightforward. Values from desired ranges are selected randomly for each dimension of the sample vectors [24, 25]. However, the simplicity and speed of this method comes with the trade off of low efficiency and potentially poor coverage of the parameter space with increased dimensionality [13, 25].

2.1.2 Latin Hypercube Sampling

Latin hypercube sampling (LHS) employs a simple randomized structure as that of the Monte Carlo sampler; however, in order to minimize repeatedly sampling the same region, the LHS sampler splits the domain of each input variable into disjoint subsets and takes a sample from each subset [22]. While providing a more uniform sampling than Monte Carlo with little sacrifice to speed and ease of implementation, there is still no guarantee of a representative sample in higher dimensions[13].

2.1.3 Hammersley Sampling

Hammersley sampling is a quasi-Monte Carlo method based on low-discrepancy sequences, meaning that as the sample size grows the samples become increasingly equidis-

tant [10, 13]. In general, it is efficient and shows better uniformity in n-dimensional problems; however, in high dimensions, samples can have spurious correlations [13, 23].

2.1.4 Sobol Sampling

Sobol sampling is another popular quasi-Monte Carlo technique that produces a low discrepancy sequence [15]. Similar to Hammersley sampling, it attempts to create more uniform samples, but in higher dimensions it is prone to spurious correlations [13], and based on experiments in [22], it may not provide much of an advantage over LHS in terms of the final results of a surrogate model.

Many other sampling techniques exist with the goal of producing uniform samples, including variations on those listed above [13, 18, 24] and gradient-based methods [11].

2.2 Surrogate Models

There are numerous methods for creating the core of the model. Covered here are basis functions, machine learning, and neural network based approaches.

2.2.1 Basis functions

Basis function models are a form of regression that model the response surface as the linear combination of basis functions corresponding to each input variable [24]. Basis functions can come in a variety of forms including polynomials [14, 18, 24], Kriging basis functions [14, 18], and radial basis functions [12, 14, 18]

In the polynomial basis function models, the degree of the polynomial depends on the problem. Frequently, a second degree polynomial is used and has the form

$$\hat{f}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \sum_{j \leq i}^{n} \beta_{ij} x_i x_i,$$

where $\hat{f}$ is the output corresponding to the input vector $\mathbf{x}$, n is the dimension of the design space, and β are parameters to be estimated [18, 24]. Note that for an n-dimensional design space, there are $(n+1)(n+2)/2$ coefficients that need to be estimated [14, 24]. Combined with the need for higher polynomials to fit non-linear response surfaces, the number of parameters to be estimated can become unwieldy [24].

Kriging basis functions, named for the South African mining engineer D. G. Krige [18], are

a popular approach to surrogate modeling due to their ability to fit non-linear response surfaces [24]. Simply put, the Kriging method estimates the value of output at a new sample as the sum of a linear model, such as a polynomial, and a component reflecting variations around the trend. These variations are assumed to have correlation depending only on the distance between them [18]. In particular, this model estimates a function

$$f(x) = \mu + \varepsilon(x), \quad E(\varepsilon) = 0$$

$$\text{cov}(\varepsilon(x_i), \varepsilon(x_j)) \neq 0, \quad \forall i, j,$$

where μ represents the mean output of the sample vectors and ε is error. The error term should have covariance that is a function of the distance between sample vectors. One possible correlation between two sample vectors x_i and x_j is

$$\text{cov}(\varepsilon(x_i)) = \exp\left(-\sum_{\ell=1}^{N} \theta^{(\ell)} \left|x_i^{(\ell)} - x_j^{(\ell)}\right|^{p_\ell}\right)$$

where $\theta^{(\ell)}$ and p_ℓ are parameters that would need to be estimated. In the case where $p_\ell = 2$ and the data is normally distributed, this is equivalent to a Gaussian process [18, 24].

Estimates for unsampled points are given by

$$\hat{f}(\mathbf{x}) = \overline{\mu} + \mathbf{r}^\top \mathbf{R}^{-1}(\mathbf{f} - \mathbf{1}\overline{\mu})$$

where variables with bars denote estimates, $\mathbf{r}$ is the correlation vector between the set of prediction points and the points used to construct the model, $\mathbf{R}$ is the correlation matrix from the N samples, $\mathbf{f}$ is the vector of outputs corresponding to the sample vectors, and $\mathbf{1}$ is an N-dimensional vector of ones [14, 18]. While popular, a major drawback of the Kriging approach is the computation of the covariance matrix, which is $N \times N$ [24]. In addition, Díaz et al. report that, compared to other approaches such as support vector regression and radial basis functions, while Kriging performs favorably at low dimensions, in higher dimensions its accuracy is diminished [12].

Radial basis functions are a third common approach to surrogate models. The response surface is approximated with a linear combination of M radially symmetric functions, h_i, where

$$h_i(\mathbf{x}) = h(\|x_i - \mathbf{x}\|)$$

for some radial, monotonic function h [18]. Specifically, we write

$$f(\mathbf{x}) = \sum_{i=1}^{M} w_i h_i(\mathbf{x}) + \epsilon_i$$

where ϵ_i is independent error with variance σ^2 [7, 14, 18]. Parameters to be selected for

radial basis functions are their centers, distance scaling, and shape. An example of a common radial basis function with scalar input is the Gaussian

$$h_i(\mathbf{x}) = \exp\left(-\frac{(\mathbf{x} - x_i)^2}{\delta^2}\right)$$

which would have center x_i and radius δ [18]. One way to express this problem in order to solve for the parameters w_i is in matrix form as

$$\mathbf{f} = \mathbf{H}\mathbf{w}$$

where $\mathbf{f}$ is the collection of sampled outputs, $\mathbf{H}$ is a matrix with entries $\mathbf{H_{ji}} = \mathbf{h_i}(\mathbf{x_j})$, and $\mathbf{w}$ is a vector containing the weights. Then,

$$\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{H}^\top\mathbf{f}$$

where $\mathbf{A} = (\mathbf{H}^\top\mathbf{H})^{-1}$. The final prediction for an unsampled $\mathbf{x}$ is given by

$$\hat{f}(\mathbf{x}) = \mathbf{h}^\top\hat{\mathbf{w}}$$

where $\mathbf{h}$ is a vector with entries $\mathbf{h_i} = h_i(\mathbf{x})$ [18].

Results from Díaz et al. [12] and Cook et al. [7] suggest that radial basis functions can be more accurate than other methods, even in higher dimensions.

2.2.2 Machine Learning

Other common approaches to surrogate modeling have their roots in machine learning. Two in particular are random forests and support vector machines.

Random forests combine the results of several decision trees and are flexible enough to be used for both classification and regression problems. First introduced by Breiman [4], random forests have become popular for their flexibility and relatively limited number of parameters to be tuned [2].

Biau and Scornet offer an introduction to the theoretical foundations of random forests [2]. Let $\mathcal{X}$ denote the parameter space and consider a collection of training points, $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and suppose the forest to be generated will have M regression trees. For the j^{th} tree, we select m observations from the sampled data set D; call this collection A_j. These are the values that will be used to create the tree. The space is split into two smaller "cells" along one of the dimensions in the design space using a splitting criterion based on the sample A_j. This process is repeated within each cell until each cell contains fewer than a specified number of points from the set A_j. For a given unsampled input

vector $\mathbf{x}$, the j^{th} decision tree predicts an output given by

$$\hat{f}_j(\mathbf{x}) = \sum_{(x_i,y_i)\in A_j} \frac{\chi_{x_i\in C(\mathbf{x})} y_i}{|C(\mathbf{x}) \cap A_j|}$$

where $C(\mathbf{x})$ is the final cell containing the input $\mathbf{x}$ and $|C(\mathbf{x}) \cap A_j|$ is the number of pre-selected points from A_j that share the same cell as $\mathbf{x}$. That is, the predicted value of $\mathbf{x}$ from a given tree is the average of the outputs from the pre-selected sample points it shares a cell with.

This process is used across the M trees, each based on a different initial sample A_j of data points. The final prediction produced by the forest is the average of all these predictions and is given by

$$f(\mathbf{x}) = \frac{1}{M} \sum_{j=1}^{M} \hat{f}_j(\mathbf{x}).$$

Many studies indicate that random forests can outperform other methods in various contexts [5, 8, 22]. However, a comprehensive study by Williams et al. covering a wide variety of surrogate modeling techniques suggests that random forests can be slow and inaccurate, though they were highly effective at determining optimums in surrogate-based optimization [22].

Support vector machines (SVM) are extremely popular approaches to surrogate modeling and can also be used for either classification or regression. They are part of a class of methods called kernel methods, in which a data set is embedded in a higher dimensional space called a feature space in which to build a linear algorithm (which would not necessarily be linear with respect to the original data space) [24, 19].

When using SVM for regression, given training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, we look to find a function $f(x_i)$ that that deviates from true output y_i by at most ϵ. In the linear case, the function can be written as

$$f(x_i) = \langle \mathbf{w}, x_i \rangle + b$$

such that $\frac{1}{2}\|\mathbf{w}\|^2$ is minimized [19, 20]. We further introduce "slack variables" to ensure such a function may exist. The problem becomes

$$\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*)$$

$$\text{subject to } \begin{cases} y_i - \langle \mathbf{w}, x_i \rangle - b \leq \xi_i + \epsilon \\ -y_i + \langle \mathbf{w}, x_i \rangle + b \leq \xi_i^* + \epsilon \end{cases}$$

where $\xi_i, \xi_i^* > 0$ deal with training samples that result in error larger than ϵ. The constant

C is set by the user and measures the trade off between minimizing error and the regularization term [19, 20]. Solving the SVR problem incorporates several different strategies, including introducing Lagrange multipliers α_i, α_i^*, and the prediction is given by

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} (\alpha_i - \alpha_j^*) K(\mathbf{x}, x_i) + b$$

where K is a kernel function.

Ciccazzo et al. found SVMs to be "promising" for use with surrogate modeling, particularly in the context of working with circuits [6]. However, studies performed by Williams et al. [22] and Caruana et al [5] suggest that compared to other methods based on basis functions and neural networks, SVMs are not as accurate or robust, and Criminisi notes that for classification purposes, SVMs are not as flexible as random forests for adapting to multi-class problems [8].

2.2.3 Neural Networks

Finally, neural networks are currently one of the most popular approaches to surrogate modeling. Neural networks try to mimic the behavior of the brain through a system of activation functions with corresponding weights. The first layer in the neural network is given by the sample vectors. Next are hidden layers comprised of the weighted activation functions followed by an output layer that gives the prediction. To define the neural network, we must choose the number of layers, the number of neurons within each layer, the activation functions for each neuron, and a method to check accuracy while training the model [22, 24].

Activation functions are commonly smooth sigmoid functions, such as logistic or hyperbolic tangent functions, as well as radial basis functions [7, 17]. The weights for the activation functions are optimized during training based on the sampled data. A popular method for training the weights is Levenberg-Marquadt back-propagation due to its computational efficiency [17, 22, 24].

Several studies suggest that neural networks form robust, accurate surrogate models, even in high dimensions [5, 7, 22, 24]. However, lack of interpretability is considered to be a significant downside to working with neural networks [8].

3 Methods

In this section, we identify tools that can be used for generating different types of surrogate models in Python, the approach to the overall structure of the model, and the initial implementation of a simple surrogate model for the electrochemistry battery model [9, 21].

There are three main tools that can be used to implement a surrogate model in Python: Surrogate Modeling Toolbox (SMT) [3], SciKit-Learn [16], and TensorFlow [1]. SMT is designed to provide tools for use with developing surrogate models, including samplers, model cores, and several test problems. Though it provides many general approaches, it is notable for its emphasis on derivative based methods. Models provided in SMT are primarily regression-based, including polynomial and Kriging basis functions. Sci-Kit Learn offers several different classification, regression, and clustering models, as well as methods for reducing dimensionality, preparing data sets, and benchmarking models. Sci-Kit Learn offers the widest variety of models, including regression, SVM, and random forests. TensorFlow is the most specialized of the three, designed to develop and train neural networks. Generating a surrogate model might be best approached using a combination of the tools available, as the SMT provides samplers and Sci-Kit Learn provides several benchmarking methods.

In regards to the model structure, the desired input and output need to be determined. In general, it is assumed that the model configuration is fixed. That is, intrinsic properties about the model are known and constant. For example, in the case of the electrochemistry battery model, we assume that model configuration parameters such as nominal voltage, mobile ions, resistance, etc. are known and fixed. Variables to be considered for input are then state parameters, load(s) to be applied, event states, or time cutoffs. Output can take many forms, such as future states, events, or model outputs such as temperature or voltage, with each of these having a different structure. Output could be a scalar in the case of events or model outputs, have vector value in the case of future states, or even be binary if one simply wanted to determine if an event threshold was reached.

To begin exploring possible input/output structures, a very simple surrogate model was built for the electrochemistry-based battery model in Python. For this implementation, the battery was assumed to have full charge and the default configuration was used, with the exception of setting the process noise to 0 [21]. Input was a constant load applied (current, A) and output was the state of charge (SOC) at a fixed point in time.

Using a random number generator, 1,000 input values were sampled uniformly from the range (0A, 5A). Simulations were ran with each input value as a constant applied load for t seconds. The resulting data set had entries of the form (i_{app}, SOC after t seconds), where t is fixed throughout. This was done for $t = 250s, 500s, 750s, 1,000s$, and $1,250s$

Of these data points, 200 of them were randomly selected to be reserved as test data; the

remaining 800 were used to train the model. Figure 1 shows the graph of the training data when the cutoff time is 1000s. Since the data is linear, the model will be fit using linear regression with Sci-Kit Learn.

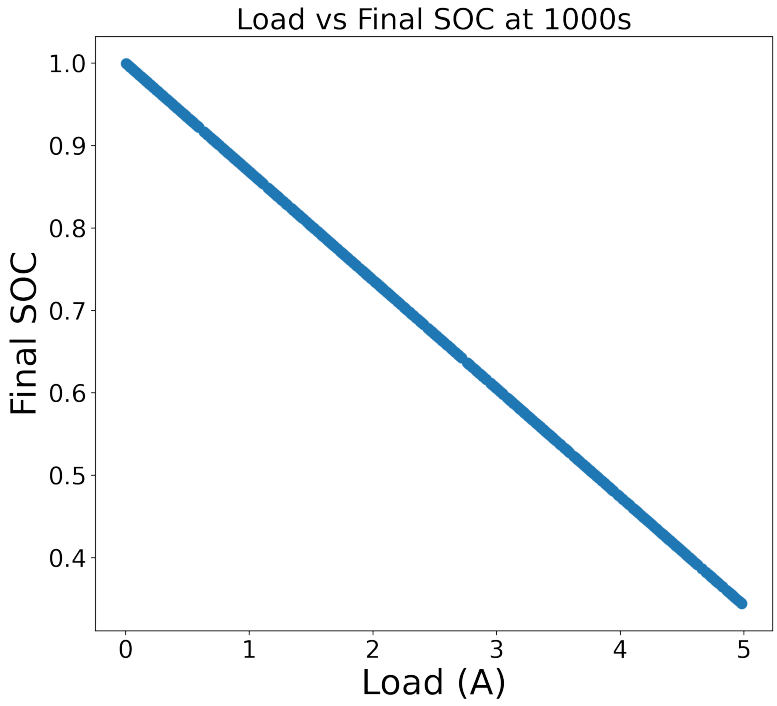

Figure 1: Training data for load vs final SOC after 1000s

4 Initial Results

Here, we describe the results from the initial implementation of the surrogate model for the electrochemistry-based battery model. Fixing the cutoff time for the simulations to be 1,000s, the linear model describing the data is given by

$$y = -0.1316x + 1.0000$$

with $r^2 = 1.0000$, as seen in Figure 2. Clearly, the model fits the data very well. The mean squared error for this model was nearly zero, specifically 3.569×10^{-28}.

These nearly exact linear models can be observed for the data from all time cutoffs, as can be seen in Figure 3. The corresponding models and errors for each are listed in Table 1.

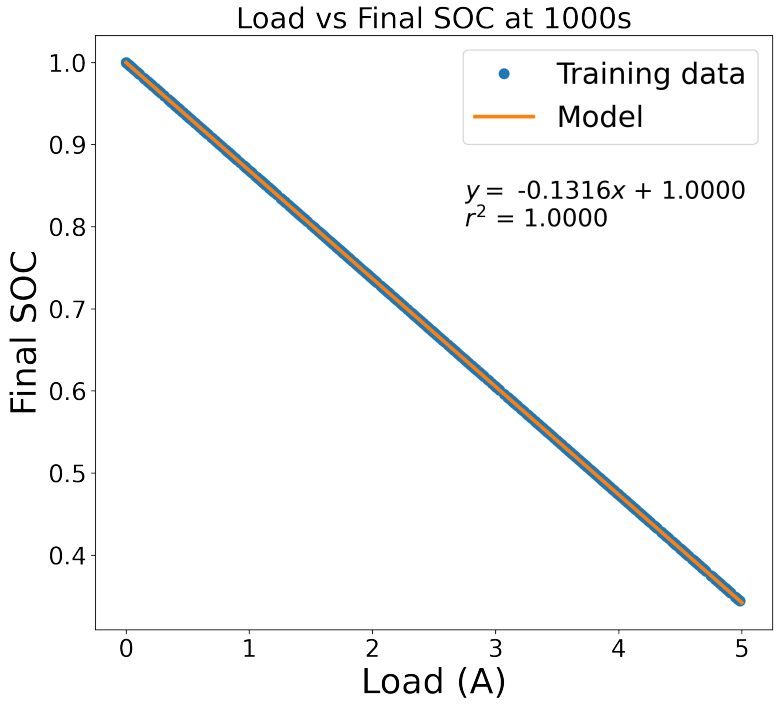

Figure 2: Model for load vs final SOC after 1000s

Note that all of them have an intercept of 1.0000, except the case where $t = 1,250s$. This makes sense as it corresponds to applying no load, which would result in no change in state of charge. The exception to the data being fit nearly perfectly with the simple linear

Time (s)	**Model**	r^2	**MSE**
250	$y = -0.0329x + 1.0000$	1.0000	3.7459×10^{-31}
500	$y = -0.0658x + 1.0000$	1.0000	2.1107×10^{-29}
750	$y = -0.0987x + 1.0000$	1.0000	1.67218×10^{-23}
1,000	$y = -0.1316x + 1.0000$	1.0000	3.569×10^{-28}
1,250	$y = -0.1661x + 1.0028$	0.9970	8.319×10^{-5}

Table 1: Linear model, r^2, and mean squared error (MSE) for load versus final SOC after different times t

model is the case where the cutoff time is 1,250s. This is likely because this time frame is long enough that when the higher loads are applied, the battery is sufficiently depleted for the voltage to be used in the computation of the state of charge, causing the sharp drop.

Looking at the linear portion of the data, however, these models can also be derived using

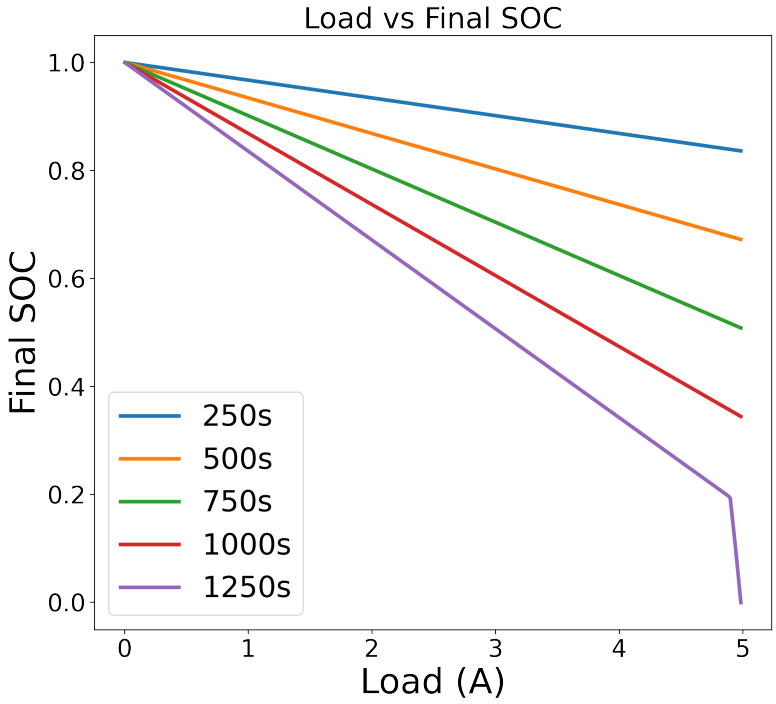

Figure 3: Training data for load vs final SOC after 1000s

equations that the state of charge is based on [9, 21]. In particular, we have

$$SOC = q_n/(0.6q^{max})$$

where q represents Lithium ions. Specifically, q_n is the number of Lithium ions in the negative electrode and q^{max} is the total number of Lithium ions. The movement of Lithium ions is described with the equations:

$$\begin{aligned} q_p &= q_{s,p} + q_{b,p} \\ \dot{q}_{s,n} &= \dot{q}_{bs,n} - i_{app} \\ \dot{q}_{b,n} &= -\dot{q}_{bs,n} + i_{app} - i_{app} \end{aligned}$$

where $\dot{q}_{bs,i} = \frac{1}{D}(c_{b,i} - c_{s,i})$ gives the diffusion rate between the bulk and surface volumes

with D as the diffusion constant. Therefore,

$$\begin{aligned}\dot{q}_n &= \dot{q}_{s,p} + \dot{q}_{b,p} \\ &= (-\dot{q}_{bs,n} + i_{app} - i_{app}) + (\dot{q}_{bs,n} - i_{app}) \\ &= -i_{app}\end{aligned}$$

Integrating with respect to time, we obtain

$$q_n = -i_{app}t + q_{n0},$$

where $q_{n0} = 0.6q^{max}$ is the initial number of ions in the negative electrode. Substituting this expression into the SOC equation gives

$$SOC = (-i_{app}t + 0.6q^{max})/(0.6q^{max}) = -t/(0.6q^{max})i_{app} + 1$$

which is a linear equation in i_{app}. With the exception of the model corresponding to the data with the cutoff time of 1,250s, this equation yields the linear models given in Table 1.

5 Future Work

Moving forward, key things to look at are input and output structure, comparing accuracy and efficiency of model cores, and eventually incorporating the surrogate model as a feature in the Prognostics Models package [21].

In regards to the input structure, as previously mentioned, possible inputs could include model states. One way to allow these to vary would be to sample input values from the distribution generated by the state estimator. That is, given a model and state estimates with uncertainties, a surrogate model could be generated specific to the estimated state by sampling from the associated distribution.

Another thing to consider is how to deal with inputs that may be a function of time, in particular the applied load. In practical application, the load is not constant. Methods for dealing with a varied load could include treating it as constant by working with an average load or approximating the load with a step function and then using the surrogate model on each individual "step". However, to do this, the surrogate model would have to produce intermediate state vectors as output, which may hinder the goal of reduced complexity.

Once an input and output structure is determined for the surrogate model, different approaches can be compared for accuracy and computational efficiency. Based on the seemingly disparate results from different studies [5, 6, 7, 22], the best surrogate model core may vary by model, and there may not be one optimal surrogate model.

Eventually, the surrogate model could be incorporated as a feature in the PrognosticsModel class in the Prognostics Models package [21]. This could either be in the form of a single model core that was determined to be best, or perhaps a variety of choices for samplers and cores could be made available to the user.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Gérard Biau and month= pages=197-227 Erwan Scornet, year=2016. A random forest guided tour. *TEST*, 25.

[3] Mohamed Amine Bouhlel, John T. Hwang, Nathalie Bartoli, Rémi Lafage, Joseph Morlier, and Joaquim R. R. A. Martins. A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102662, 2019.

[4] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[5] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. *International Conference on Machine Learning*, pages 96–103, 2008.

[6] Angelo Ciccazzo, Gianni Di Pillo, and Vittori Latorre. Support vector machines for surrogate modeling of electronic circuits. *Neural Computing and Applications*, 2014.

[7] Jared Cook, Ralph Smith, Jason Hite, Razvan Stefanescu, and John Mattingly. Application and evaluation of surrogate models for radiation source search. *Algorithms*, 12, 2019.

[8] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning, and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2-3):81–227, 2012.

[9] Matthew Daigle and Chetan S. Kulkarni. Electrochemistry-based battery modeling for prognostics. In *Annual Conference of the Prognostics and Health Management Society 2013*, pages 249–261, 2013.

[10] Ishaan L. Dalal, Deian Stefan, and Jared Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *2008 International Conference on Application-Specific Systems, Architectures and Processors*, pages 108–113, 2008.

[11] Dirk Deschrijver, Karel Crombecq, Huu Nguyen, and Tom Dhaene. Adaptive sampling algorithm for macromodeling of parameterized s-parameter responses. *Microwave Theory and Techniques, IEEE Transactions on*, 59:39 – 45, 2011.

[12] Alan Díaz-Manríquez, Gregorio Toscano-Pulido, and Wilfrido Gómez-Flores. On the selection of surrogate models in evolutionary optimization algorithms. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2155–2162, 2011.

[13] Nishant Dige and Urmila Diwekar. Efficient sampling algorithm for large-scale optimization under certainty problems. *Computers & Chemical Engineering*, 115(12):431–454, 2018.

[14] Zhong-Hua Han and Ke-Shi Zhang. Surrogate-based optimization. In *Real-World Applications of Genetic Algorithms*, chapter 17, pages 343–362. InTech, 2011.

[15] Stephen Joe and Frances Y. Kuo. Constructing sobol' sequences with better two-dimensional projections. *SIAM Journal on Scientific Computing*, 30:2635–2654, 2008.

[16] 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.

[17] Kevin Priddy and Paul Keller. *Artificial Neural Networks*. Technology & Engineering. SPIE Press, 2005.

[18] Nestor V. Quiepo, Raphael T. Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P. Kevin Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28, 2005.

[19] S. Salcedo-Sanz, J. L. Rojo-Álvarez, M. Martínez-Ramón, and G. Camps-Valls. Support vector machines in engineering: an overview. *WIREs Data Mining and Knowledge Discovery*, 5:234–267, 2014.

[20] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2003.

[21] Christopher Teubert, Matteo Corbetta, Chetan Kulkarni, and Matthew Daigle. Prognostics models python package, April 2021.

[22] B. A. Williams and S. Cremaschi. Surrogate model selection for design space approximation and surrogate-based optimization. *Computer Aided Chemical Engineering*, 47:353–358, 2019.

[23] Tien-Tsin Wong, Wai-Shing Luk, and Pheng-Ann Heng. Sampling with hammersley and halton points. *Journal of Graphics Tools*, 2, 1997.

[24] Mustafa Berke Yelten, Ting Zhu, Slawomir Koziel, and Paul D Frazon. Demystifying surrogate modeling for circuits and systems. *IEEE Circuits and Systems Magazine*, 12(1), 2012.

[25] Jinhang Yin, Wenxi Lu, Xin Xin, and Lei Zhang. Application of monte carlo sampling and latin hypercube sampling methods in pumping schedule design during establishing surrogate model. In *2011 International Symposium on Water Resource and Environmental Protection*, volume 1, pages 212–215, 2011.