Computationally and Statistically Efficient Model Fitting Techniques

Christine Harvey*, Scott Rosen*, James Ramsey*, Christopher Saunders*†, and Samar K. Guharay*
*The MITRE Corporation, McLean, Virginia 22102
†South Dakota State University, Brookings, South Dakota 57007

***Abstract*—In large-scale stochastic simulations, analysis with sufficient accuracy is often extremely time consuming. The complexity of the analysis is exacerbated with increasing dimensionality of the parameter space and sudden abruptness in the topology of the input-output response surface. This paper addresses computational issues in fitting and generating error measures of simulation metamodels, demonstrating the merit of high-performance computing in Python. We systematically compare the speed of programming languages including MATLAB, R and Python as well as using different computing architectures including high-performing laptops and high-power parallel processing clusters. The experimentation is discussed in this paper using a simple scenario, and activities are being pursued to study other scenarios with varying complexities that will be reported at the conference.**

I. Introduction

In large-scale stochastic simulations, completing a single simulation with sufficient accuracy is often time consuming. The complexity of the problem is exacerbated with increasing dimensionality of the parameter space and sudden abruptness in the topology of the input-output response surface. These effects have been reported in multiple disciplines of basic and applied science and technology, including high-energy astrophysics [1]–[3], atmospheric science [4], systems biology [5], power system networks [6], and many others. The importance of developing new computing architectures and novel modeling and simulation are well recognized [7]–[11]. Significant activities are being pursued to advance modeling and simulation (M&S) through high-performance computing.

Simulation modeling frameworks often lead to increased run times, which hinder analysis. Simulation metamodeling, which involves building a model of a simulation model, alleviates these long run times. A large body of literature exists on metamodeling [12], [13]. One of our recent works reported a complex case study with more than one hundred input parameters and demonstrated effective means of identification of the most relevant parameters incorporating both traditional factor screening methods as well as controlled sequential bifurcation [9].

The efficiency and effectiveness of metamodeling depends on the underlying statistical and mathematical methods that build the analytical foundation of the techniques approximating metamodels to large-scale models. Additionally, the overall computing architecture that develops the metamodel from the large-scale M&S plays a very critical role in increasing the speed of the experimentation with the simulation and the generation of the statistical measures used to evaluate the metamodels goodness-of-fit.

This paper addresses some challenging questions for validating metamodels and builds an effective and efficient computational framework for quantifying standard error and bias. In this pursuit, algorithms are developed for computing Bootstrap error and the PRESS statistic, which requires massive calculations. A systematic speed comparison is made involving the implementation of different programming languages including MATLAB, R, and Python through different computing architectures such as high-performing laptops and high-power parallel processing clusters. One of the major objectives is to build a path to efficiently and effectively identify a metamodeling method that most suits data structures of varying complexity, especially with increasing dimensionality and sensitivity of the response surface. The basic foundation of the concept is discussed in this paper using a simple scenario, and activities are being pursued to study other scenarios with varying complexities that will be reported at the conference.

Section II describes the background of the problem. The technical approach and implementation are discussed in Section III and IV, respectively. Section V reviews the methodology. Results are given in Section VI. Finally, Section VII summarizes the work.

II. Background

Kleijnen and Sargent first introduced the concept of simulation metamodel validation [14]. They define validation with regards to a simulation and/or a metamodel as the substantiation that a model within its domain of applicability possesses satisfactory accuracy consistent with the intended applications of the model. Relating accuracy to the intended application addresses the question of metamodel sufficiency, which is a subjective topic.

The accuracy of a metamodel is assessed based on a combination of minimizing the required computational expense and maximizing the reliability or accuracy of the assessment. Based on how an analyst would prioritize these criteria, a chosen validation procedure may either sample additional runs from the large-scale simulation or utilize the simulation data that is already in hand. When generating additional data from the large-scale simulation, predicted responses of the metamodel are compared to the true responses using additional runs of the original simulation within the design space applied to calibrate the metamodel. Basic measures to be considered here, as Kleijnen [14] suggests, are the coefficient of determination R^2 as well as an adjusted-R^2 value, which accounts for the

problem of R^2 always increasing as additional regression coefficients are added. The root mean square error $RMSE$, where

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{m}}$$

and the maximum absolute error (MAX), where

$$MAX = max\left|y_i - \hat{y}_i\right|, i = 1, ..., m$$

where $\hat{y}_i$ is the predicted value at point i, y_i is the true value of the parameter being estimated at point i and m is the number of sample points. These are descriptive measures for understanding error across the design space designated for the metamodel. These measures are reliable when the simulation can be resampled to generate a large testing set, but as noted above, this is not practical in many situations, especially when considering very flexible families of metamodels [14].

Quite often, it is more efficient and practical for the analyst to use existing simulation runs to avoid resampling on the simulation model. In this practice, the simulation runs used to calibrate the metamodel are reapplied for validation purposes, as suggested by Laslett [15]. This includes the concept of p-fold-cross-validation adaptation where the simulation data set is split into p subsets with the metamodel being fit p times, each time leaving out one of the subsets from training and using the excluded set for computation of the error measure. The leave-k out approach was pursued by the simulation metamodeling community and studied with different metamodel techniques [16]. The leave-k out approach is a variation of the p-fold cross-validation. Mitchell and Morris [17] proposed leave-one-out cross-validation, where the left out subsets are of size one.

The Bootstrap procedure [18] is a family of methods for estimating the statistical accuracy of an error measure and is applicable to simulation metamodeling. Kleijnen and Deflandre [19] provide an adapted procedure for regression simulation metamodels that does not assume normality to derive a R^2 statistic and a lack-of-fit F statistic.

The Bootstrap method consists of resampling data points with replacement to expand the simulation data to a very large set of training and testing data. Various error estimates can be computed on each of the n Bootstrap samples. The error measure of interest is then averaged across the Bootstrap samples.

As we are moving from a single personal computer to a clustered simulation experimentation environment, we are interested in a Bootstrapped standard error measure that involves recalibration of the metamodel at each Bootstrap sample. This gives insight into the sensitivity of the goodness-of-fit of the metamodel family in addition to its standard error without resampling of the large-scale simulation. In addition, this can be coupled with a PRESS statistic computed through a leave-one-out resampling procedure that gives insight into bias. Both of these two error measures used in conjunction can provide deep insight into the accuracy of the metamodel, which we measure as a function of standard error and bias.

The drawback to this Bootstrapping approach for standard error and PRESS computation involving repeated metamodel recalibrations is extensive computational time. In most situations the precision of the Bootstrap standard error is dependent on the number of Bootstrap samples drawn. The PRESS statistic will be at least of the order of the number of design points times the different number of metamodel classes that are currently being considered (usually with a much greater computational complexity). This computational time is further increased when more complicated metamodel structures such as Neural Networks or Stochastic Kriging are applied. We propose a method to parallelize the computations in this Bootstrap method to reduce computational time especially for complex systems analysis and avoid cumbersome simulation resampling that arises due to scenario configuration or licensing issues hindering parallelization of the running of the large-scale simulation.

III. Approach

The standard error statistic is the measure of the stability for an estimated metamodel. Unfortunately, in complex metamodels with a large number of parameters, the standard error takes the form of a large covariance matrix or even covariance functional. To facilitate the interpretation of the stability of the estimated metamodel, we propose a univaraite summary statistic that captures the distance between two fitted metamodels.

With the summary of the distance between two fitted metamodels, we will proceed to use a Bootstrap algorithm to estimate the expected distance between two fitted metamodels using a specified number of simulation runs. Grossly speaking, a Bootstrap algorithm works by drawing samples with replacement from the original set of samples. The statistic is then fitted into a new Boostrap sample. This process is repeated a large number of times.

We assume that we have results from a large-scale simulation model generating data from a set of design points $\{x_i\}_{i=1}^{n}$. For the i^{th} design point, we observe the results for k simulations to fit a metamodel $f_0(x_i)$, which approximates the simulation at point x_i. The simulation output at the i^{th} design point for the j^{th} repetition is

$$y_{ij} = f_0(x_i) + \epsilon_{ij}$$

where ϵ_{ij} are independent identically distributed random variables with a mean of zero and a common variance σ_e^2. For the current results from the simulations, denoted as $D_{nk} = \{(y_{ij}, x_i)\}$, we can estimate f_0 with an estimation strategy, $e = D_{nk} \mapsto m \in M, i.e. \hat{m}_{nk} = e(D_{nk})$

We use two metrics to compute the accuracy of candidate metamodels:

1) For two classes of metamodels, M_1 and M_2, for an estimation strategy e, we would like to compare the bias for the two candidate classes of metamodels.
2) To characterize the variance of $\hat{m}_{nk}$, we tend to refer to $var(\hat{m}_{nk})$ as the stability of the estimator from the class, M.

Regarding the first metric, we implement a $PRESS$ statistic, which is standardized by an $ANOVA$ type estimator of σ^2. We will use a slightly modified version of the PRESS statistic as defined in Allen [20]. This PRESS statistic takes the form of

$$PRESS = \frac{1}{nm}\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij} - \hat{f}_{-i}(x_i))^2,$$

where $\hat{f}_{-i}$ is the estimate of the true model within the metamodel family that does not use the points associated with the design point x_i.

To solve the second problem, we implement a Bootstrap estimate of the $var(\hat{m}_{nk})$.

IV. Implementation[1]

The PRESS and Bootstrap statistical calculations were first implemented in MATLAB to perform evaluations of metamodels. This application of these methods did not efficiently test and evaluate metamodels for large datasets containing ten thousand or more data points. The amount of time required to perform computations of the statistics overshadowed the usefulness of the scripts.

We then implemented PRESS and Bootstrap statistics in Python and R. Additionally, the original MATLAB scripts were updated to reflect best MATLAB practices in regards to vectorization. All scripts were developed to read input data files in standard CSV format. To further explore the limits of the implementations, a parallel version of the Python script were also developed. Each of these implementations used the language's specific nonlinear model fitting function. Options were adjusted for all of the models to ensure the same tolerance was being used in each implementation. Unnecessary looping was avoided in all instances and each script was designed to make use of the strengths and best practices of each language.

A. MATLAB Implementation

MATLAB's function to fit nonlinear regression models is the `fitnlm` function. The goal of `fitnlm` is to find values for the parameters β that minimize the mean squared differences between the observed responses y and the predictions of the model [21]. This function takes in a model function, predictor and response values, and initial guesses for the values of β. The algorithm used in this nonlinear model fitting tool is the Levenberg-Marquardt least squares algorithm.

This application also used the `feval` function in MATLAB to evaluate the model using a certain set of predictor values. This function evaluates a given model using the input arguments and produces the expected output values of the function. Both the PRESS and Bootstrap implementations only use a single loop to perform the iterations and computations. All other computations are completed using vectorization. These applications of the statistical methods were developed to avoid unnecessary overhead.

[1]Scripts for model implementations can be provided on a case by case basis by contacting the authors.

B. R Implementation

The standard version of R includes the Nonlinear Least Squares (`nls`) function that computes the nonlinear least-squares estimates of the parameters of a nonlinear model [22]. This standard version of the model uses the Gauss-Newton algorithm by default which has been proven to be much quicker than the Levenberg-Marquardt algorithm [23]. In order to maintain consistency between each implementation, an alternate version of the `nls` function was used from the minpack.lm package. This package contains a function, `nlsLM`, which uses the Levenberg-Marquardt algorithm. This function uses a nonlinear model with variables and β parameters, data that includes predictor and response values, and an optional list of starting guesses for the β values. The `nlsLM` function produces a fitted model object.

These executions of the PRESS and Bootstrap computations also use the predict function in R to evaluate the models at specific predictor values. The predict function works with fitted model objects in R to determine the predicted model outcomes at defined values. The R scripts for the computations also only use a single loop to perform the model fitting and take advantage of vectorization in R to perform other calculations.

C. Python Implementation

The model fitting portion of the computations depends on the Scipy library. Scipy's optimization package provides multiple commonly used optimization algorithms including `curve_fit` which is the function used in this research. The curve fitting function uses non-linear least squares to fit a function, f to provided data. The algorithm used in this function is the Levenberg-Marquardt algorithm [24] which produces a set of β values that represent the model's best fit.

The model fitting function also requires both the input of a model function as well as predictor and response values. The model function is implemented as a traditional function in Python in a separate module. The initial guess for the parameters is optional in this function, but the feature is utilized to match the MATLAB implementation. The fitted models are then evaluated using the original model function with the predictor values and the computed β values.

D. Python Implementation with MPI4Py

Due to the embarrassingly parallel nature of this problem, utilizing parallel techniques in Python is an obvious solution to enhance the timing of the process. Both the Bootstrap and PRESS computation scripts were altered to use the Message Passing Interface (MPI) tool for Python, mpi4py, to parallelize the scripts [25].

This version of the program distributes the model fitting and evaluation required in a single cycle of the loop evenly amongst multiple processors. This technique distributes the workload and computational power needed across multiple computing nodes, which has a significant impact on the time needed to compute the overall statistics for large or complex computations.

V. Methodology

All simulations were run on the same 64-bit Windows 7 laptop with an Intel i7 Processor which has four cores and 8GB of RAM. The dataset used was output data from a model simulating border crossings. This model uses three independent predictor variables and generates a single response value, referred to as the utility value.

Two hundred different desgin points were used for the experiment with ten replications completed at each design point. Results were gathered taking the average of three runs of the analysis scripts.

Timing analysis was done using special profiling functions specific to each language. MATLAB has a profiling function that provides information on the running time of the script as well as individual functions executed within the script. This allows us to see which areas of the script are responsible for any potential bottlenecks. Python has a similar profiling function, `cProfile` which provides information on the total running time as well as individual running times of child functions. Finally, the R scripts were profiled using the `Rprof` function, which times the execution of code between specified start and ending points. Each of these profiling functions produces comparable outputs and results which can be used to compare each of the applications.

All of the profilers produce extremely detailed information on all functions and child functions called in the scripts. Since each profiler generates comprehensive data on each function call, only pertinent information is displayed in the following tables and charts. Non-crucial function calls have been omitted from the profiling data represented in this paper.

Different nonlinear metamodels of increasing order were used to fit the dataset. All of these models include an error term ϵ. The first model, shown in Equation 1, is a second order nonlinear model with interactions among the parameters.

$$\begin{aligned} f_0(x) = {} & \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 \\ & + \beta_6 x_2 x_3 + \beta_7 x_1^2 + \beta_8 x_2^2 + \beta_9 x_3^2 + \epsilon \end{aligned} \tag{1}$$

The next equations used to perform nonlinear fitting were third order equations. Equation 2 is a third order equation without interactions between the parameters.

$$\begin{aligned} f_0(x) = {} & \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1^2 + \beta_5 x_2^2 \\ & + \beta_6 x_3^2 + \beta_7 x_1^3 + \beta_8 x_2^3 + \beta_9 x_3^3 + \epsilon \end{aligned} \tag{2}$$

Equation 3 is a more complex third order equation which contains additional terms that represent interactions between the different predictor values.

$$\begin{aligned} f_0(x) = {} & \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 \\ & + \beta_6 x_2 x_3 + \beta_7 x_1^2 + \beta_8 x_2^2 + \beta_9 x_3^2 + \beta_{10} x_1^2 x_2 \\ & + \beta_{11} x_1^2 x_3 + \beta_{12} x_2^2 x_1 + \beta_{13} x_2^2 x_3 + \beta_{14} x_3^2 x_1 \\ & + \beta_{15} x_3^2 x_2 + \beta_{16} x_1 x_2 x_3 + \beta_{17} x_1^3 + \beta_{18} x_2^3 + \beta_{19} x_3^3 + \epsilon \end{aligned} \tag{3}$$

The final two equations used to perform the nonlinear model fitting were of the fourth order. Equation 4 is another basic equation without any terms interactions between the parameters.

$$\begin{aligned} f_0(x) = {} & \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1^2 + \beta_5 x_2^2 + \beta_{10} x_1^4 \\ & + \beta_6 x_3^2 + \beta_7 x_1^3 + \beta_8 x_2^3 + \beta_9 x_3^3 + \beta_{11} x_2^4 + \beta_{12} x_3^4 + \epsilon \end{aligned} \tag{4}$$

Equation 5 is the most complex equation used in this methodology and is a fourth order equation that contains additional terms which represent interactions between the different predictor values.

$$\begin{aligned} f_0(x) = {} & \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 \\ & + \beta_6 x_2 x_3 + \beta_7 x_1^2 + \beta_8 x_2^2 + \beta_9 x_3^2 + \beta_{10} x_1^2 x_2 \\ & + \beta_{11} x_1^2 x_3 + \beta_{12} x_2^2 x_1 + \beta_{13} x_2^2 x_3 + \beta_{14} x_3^2 x_1 \\ & + \beta_{15} x_3^2 x_2 + \beta_{16} x_1 x_2 x_3 + \beta_{17} x_1^3 + \beta_{18} x_2^3 \\ & + \beta_{19} x_3^3 + \beta_{20} x_1^2 x_2 x_3 + \beta_{21} x_1 x_1^2 x_3 + \beta_{22} x_1 x_2 x_3^2 \\ & + \beta_{23} x_1^2 x_2^2 + \beta_{24} x_1^2 x_3^2 + \beta_{25} x_2^2 x_3^2 + \beta_{26} x_1^3 x_2 \\ & + \beta_{27} x_1^3 x_3 + \beta_{28} x_2^3 x_1 + \beta_{29} x_2^3 x_3 + \beta_{30} x_3^3 x_1 \\ & + \beta_{31} x_3^3 x_2 + \beta_{32} x_1^4 + \beta_{33} x_2^4 + \beta_{34} x_3^4 + \epsilon \end{aligned} \tag{5}$$

These metamodels functional forms were chosen because they vary in complexity for solving the equations. The second order model is fairly simple in nature and is computationally simple compared to the fourth order model with interactions. This sampling of models allows us to also examine which of the tools has the ability to scale up not only in dataset size but also in metamodeling complexity.

VI. Results and Discussion

Results gathered from testing both the PRESS and Bootstrap computations can be seen in Table I and Table II. These results show the average running time for each computation using each of the models reviewed in Section V. Results are also displayed in Figure 1 and Figure 2.

These results show that MATLAB has the worst performance across all of the tested implementations. The MATLAB scripts consistently underperformed compared to the R and Python applications. This can be further explored by looking into the profiling information on the PRESS and Bootstrap statistics. The example shown in Table III is for the MATLAB computation of the PRESS statistic using the third order model with interactions, Equation 3. This profiling information only shows the top level functions called in the model.

In the MATLAB terminology for profiling, Total Time is the total amount of time spent inside of a function and all of its child functions. These results show that the entire MATLAB function, `fitnlm` is inherently slow and detrimental to the performance of the MATLAB script. A possible explanation for this impediment could be the way this function handles the model input formula. This equation formula is read in as a string and must be parsed through and deconstructed each time the function is called.

The performance of the R scripts were much better than the MATLAB performance for every testing scenario and similar in timing results to the Python and parallel Python results. A better understanding of the performance of the R script is

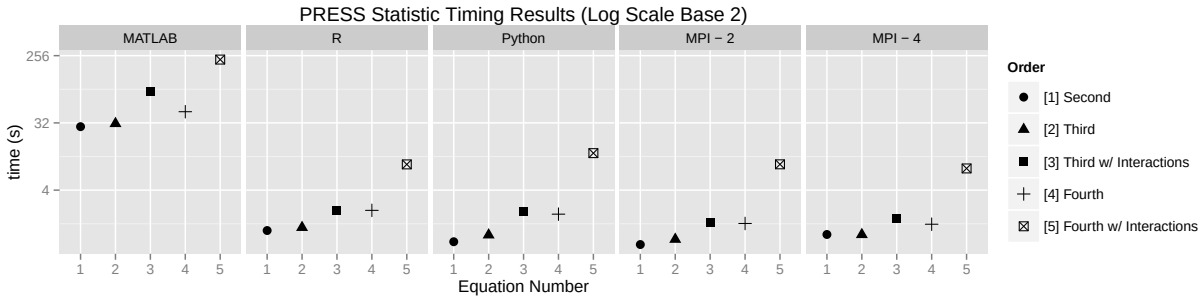

Fig. 1. Timing analysis for different levels of complexity and implementation of the PRESS statistic. Combined results from the parallel processing applications in Python show the best results in all cases. The computation in R is competitive while MATLAB has an inferior performance. The horizontal axis shows the equation numbers referenced in Section V.

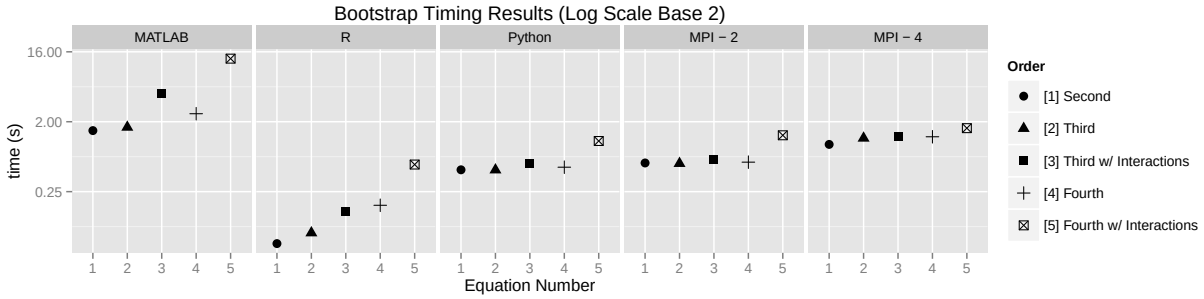

Fig. 2. Timing analysis for different levels of complexity and implementation of the Bootstrap statistic. All Python methods show promising results, but the computation in R produces the best overall outcomes. MATLAB has inferior performance for every tested equation. The horizontal axis shows the equation numbers referenced in Section V.

Equation	Nonlinear Model	MATLAB (s)	R (s)	Python (s)	MPI Python (2 Cores) (s)	MPI Python (4 Cores) (s)
1	Second Order	28.481	1.147	0.811	0.744	1.015
2	Third Order	31.315	1.260	0.999	0.873	1.011
3	Third Order with Interactions	84.907	2.120	2.062	1.488	1.686
4	Fourth Order	45.222	2.140	1.902	1.426	1.394
5	Fourth Order with Interactions	226.722	8.867	12.554	8.887	7.825

TABLE I. TIMING COMPARISON FOR PRESS STATISTIC COMPUTATION IN MATLAB, R, PYTHON AND PARALLELIZED PYTHON.

Equation	Nonlinear Model	MATLAB (s)	R (s)	Python (s)	MPI Python (2 Cores) (s)	MPI Python (4 Cores) (s)
1	Second Order	1.540	0.053	0.479	0.587	1.019
2	Third Order	1.698	0.073	0.477	0.578	1.227
3	Third Order w Interactions	4.574	0.140	0.577	0.660	1.279
4	Fourth Order	2.538	0.167	0.517	0.601	1.279
5	Fourth Order w Interactions	13.063	0.560	1.127	1.336	1.652

TABLE II. TIMING COMPARISON FOR BOOTSTRAP STATISTIC COMPUTATION IN MATLAB, R, PYTHON AND PARALLELIZED PYTHON.

Function	Calls	Total Time (s)	% Time
fitnlm	200	94.929	98.8
feval	200	1.001	1.0
csvread	2	0.044	0.0
mean	1	0.000	0.0

TABLE III. PROFILING INFORMATION ON MATLAB PRESS STATISTIC.

Function	self.total (s)	% Time
nlsLM	1.76	77.19
predict	0.10	4.39

TABLE IV. PROFILING INFORMATION ON R PRESS STATISTIC.

achieved by looking into the profiling data in Table IV gathered by `Rprof` using Equation 3.

In the `Rprof` tool, self.total is the amount of time spent in the function and its callees. The profiling information from the R script shows that the nonlinear model fitting function, `nlsLM`, takes up nearly 78% of the running time of the script. This is a much smaller and less significant portion of time compared to the MATLAB script.

All versions of the Python scripts performed well in the testing of the PRESS and Bootstrap statistics. All Python scripts outperformed the MATLAB scripts and these scripts were executed at a comparable speed to the R scripts. More information can be found on the intricacies of the script executions in the profiling data in Table V.

The `cProfile` tool reports on the number of calls made (nCalls) and the cumulative time (cumtime), which is the

	Python			MPI Python (2 Cores)			MPI Python (4 Cores)		
Function	nCalls	cumtime (s)	% Time	nCalls	cumtime (s)	% Time	nCalls	cumtime (s)	% Time
curve_fit	200	1.557	76.436	100	0.923	59.978	50	0.641	44.391
genfromtxt	2	0.038	1.865	2	0.054	3.450	2	0.066	4.571
evalModel	200	0.032	1.865	100	0.020	1.278	50	0.012	0.931

TABLE V. PROFILING INFORMATION FOR PRESS STATISTIC COMPUTATION IN PYTHON AND PARALLELIZED PYTHON.

amount of time spent in each function and all sub-functions. These profiling results show a close resemblance to the R timing output. Profiling information is also included in Table V on the Parallelized Python code for simulations run on 2 and 4 cores. The results here do show that the computational time is well distributed.

Additional work to compile more design points for the sample model is ongoing. Future work should be able to provide results of these techniques with much larger datasets. These larger datasets will be able to more completely test the capabilities and limitations of the PRESS and Bootstrap statistic implementations.

VII. SUMMARY

Overall, the timing results show that the MATLAB model fitting implementations of both the PRESS and Bootstrap statistics are inherently less efficient than their R and Python counterparts. These results show that the MATLAB approach is an entirely inefficient application of these two statistical computations.

Upon looking deeper into the MATLAB profiling information, we realized that the MATLAB nonlinear fitting function reads in a model in string format. Therefore, a major bottleneck in this process is converting this string into a proper, numerical function. This becomes especially computationally intensive when the model or function being used in the nonlinear model fitting implementation is particularly long. According to the MATLAB documentation, there is an alternate way to use `fitnlm` function [26], but this method has not yet been evaluated for this research.

For the PRESS statistic computation, the Python and R scripts produce similar timing results. It can be seen in Table I that the Python implementation outperforms the R version in all instances except for the model fitting with Equation 5. While the serial Python implementation did not outperform the R, the parallel versions had better overall performance than the serial version. The MPI Python version running on two cores performed better than the Python application for all models and was only slightly slower than the R implementation for the fourth order model with interactions. The MPI Python application with four cores had worse performance than the serial Python for the two simplest models, but as the models became more complex, the value of the multi-core application is apparent.

The Bootstrapping implementations are less computationally intensive and have a faster running time, which makes it more difficult to distinguish differences in performance across the different implementations. The MATLAB application is still consistently the slowest version of the code. All of the R implementations of the model fitting code have the best timing performance followed closely by the serial Python code. Both the two and four core MPI Python implementations have a slower running time than the serial Python version. This lack of improvement in speed is likely due to the already quick running time of the script. With parallel programming, overhead is required to launch the program on multiple processors and perform communication. Since the script already has such a quick run time, no improvement can be seen from parallelizing the code.

Another possible reason for the parallel Python scripts to perform less than expected in both the PRESS and Bootstrap statistics is the additional time required to perform the reading in of data files. This slowdown can be seen in Table V for the `genfromtxt` function call, which is responsible for reading in the data. This increase in time to read in files is a common side effect with parallel programming, only one of the processors can access and read the data file at a time, which leads to a bottleneck in the file input portion of the script. A possible alternate approach to this issue would be to only have the root node read in the data files and then broadcast this information to the rest of the nodes. This might be a good potential solution for smaller datasets but would not be a viable option for large amounts of data. Another way to avoid this bottleneck would be to run the parallel scripts on separate machines with distributed memory. This would also solve the problem of all processors trying to read the same file at once.

Overall, this study on the different implementations of the PRESS and Bootstrap statistics found Python to be the optimal choice for performing these computations for large, computationally complex datasets. The serial Python implementation was able to achieve results comparable to the statistical software, R. Like R, Python is easily accessible and has a permissible free software license. Python is easily portable and easily parallelized using MPI techniques improved performance.

ACKNOWLEDGMENT

This work is supported by the Systems Engineering area in the MITRE Innovation Program. Thanks are due to Marie Francesca for her continued support and many insightful suggestions.

REFERENCES

[1] "High performance computing," http://space.mit.edu/research/high-performance-computing, accessed: 2014-08-13.

[2] J. C. Cuevas-Tello, "High performance computing on astrophyics with artificial intelligence algorithms," http://ciep.ing.uaslp.mx/publicaciones/592012Paper, accessed: 2014-08-13.

[3] I. G. Francis and C. Drugan, "Groundbreaking astrophysics accelerated," *HPC Source*, pp. 9–12, February 2013.

[4] L. Hwang, T. Jordan, L. Kellog, J. Tromp, and R. Willemann, "Advancing solid earth system science through high performance computing," February 2013.

[5] R. Stevens, "Biology and high-performance computing," *UK HPC Users Meeting*, September 2002.

[6] S. Khaitan and A. Gupta, *High Performance Computing in Power and Energy Systems*. Springer, September 2013.

[7] A. Biberman and K. Bergman, "Optical interconnection networks for high-performance computing systems," *Reports on Progress in Physics*, vol. 75, 2012.

[8] Intel, "Big data meets high performance computing," 2013. [Online]. Available: http://www.intel.com/content/www/us/en/software/intel-lustre-big-data-meets-high-performance-computing.html

[9] S. L. Rosen and S. K. Guharay, "A case study examining the impact of factor screening for neural network metamodels," *Proceedings of the 2013 Winter Simulation Conference, ed. R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill and M.E. Kuhl*, pp. 486–496, 2013.

[10] E. Bonaneau, "Big data and the bright future of simulation," in *Proceedings of the 2013 Winter Simulation Conference*, 2013.

[11] B. Nelson, "The simulation curmudgeon," in *Proceedings of the 2013 Winter Simulation Conference*, 2013.

[12] R. Barton and M. Meckesheimer, "Metamodel-based simulation optimization," *Handbook in OR & MS*, vol. 13, 2006.

[13] G. Klir, "Review of model-based systems engineering," *International Journal of General Systems*, vol. 25, pp. 179–180, 1996.

[14] J. Kleijnen and R. Sargent, "A methodology for the fitting and validation of metamodels in simulation," *European Journal of Operational Research*, pp. 120–131, 2000.

[15] G. Laslett, "Kriging and splines: An empirical comparison of their predictive performance in some applications," *Journal of the American Statistical Association*, pp. 391–400, 1994.

[16] M. Meckesheimer, A. Booker, R. Barton, and T. Simpson, "Computationally inexpensive metamodel assessment strategies," *AIAA Journal*, pp. 2053–2060, 2002.

[17] T. Mitchell and M. Morris, "Bayesian design and analysis of computer experiments: Two examples," *Statistica Sinica*, pp. 359–379, 1992.

[18] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. Chapman and Hall, New York, NY, 1993.

[19] J. Kleijnen and D. Deflandre, "Validation of regression metamodels in simulation: Bootstrap approach," *European Journal of Operational Research*, pp. 120–131, 2006.

[20] D. M. Allen, "The relationship between variable selection and data augmentation and a method for prediction," *Technometrics*, vol. 16, pp. 125–127, 1974.

[21] MATLAB, *version 2013b*. Natick, Massachusetts: The MathWorks Inc., 2013.

[22] D. M. Bates and D. Watts, "Nonlinear least squares," http://stat.ethz.ch/R-manual/R-patched/library/stats/html/nls.html , accessed: 2014-08-13.

[23] Y. Jianchao and C. Chern, "Comparison of newton-gauss with levenberg-marquardt algorithm for spapce resection," *22nd Asian Conference on Remote Sensing*, November 2001.

[24] E. Jones, E. Oliphant, and P. Peterson, "Scipy: Open source scientific tools for python," http://www.scipy.org/, accessed: 2014-08-13.

[25] L. Dalcin, "Mpi for python," http://pythonhosted.org//mpi4py/, accessed: 2014-08-13.

[26] MATLAB, "fitnlm: Fit nonlinear regression model," http://www.mathworks.com/help/stats/fitnlm.html, accessed: 2014-10-24.