47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition
5 - 8 January 2009, Orlando, Florida

AIAA 2009-802

A User Friendly Interface for Gaussian Process Metamodeling

Collin R. Baukol*, Dr. Rob A. McDonald† and Nicholas Delmas‡
California Polytechnic State University, San Luis Obispo, CA, 93407

As research into metamodeling and using advanced metamodeling techniques continues, it is important to remember design engineers who need to use these advancements. Even experienced engineers may not be well versed in the material and mathematical background that is currently required to generate and fully comprehend advanced complex metamodels. A user friendly metamodeling environment is being developed to help bridge the gap that is currently growing between the research community and the design engineers. This tool allows users to easily create, modify, and assess the quality of metamodels.

I. Introduction

MODERN day engineers are continually given new and complex grand challenges, including simultaneously reducing emissions, noise and cost, all while developing multi-mission aircraft. Those engineers striving to develop further complex systems continually heighten the demands placed on modern computing resources. Thankfully, we expect computing resources to become faster and more powerful; however, the designer cannot put progress on hold while we wait for the ultimate computer. The increasing complexity of systems and the tightening environmental, economic, and performance demands placed on them drive the need for a continual increase in the fidelity of analysis throughout the design process. Naturally, running numerous complex models in the design process quickly becomes prohibitive.

Another problem plaguing engineers, specifically those working in a multidisciplinary design optimization field, is the problem of size[1]. Koch points out that traditional parametric design approaches which work well for small, simple problems, become inefficient and inappropriate when applied to large scale, complex systems. Three specific issues that are related to size are: the number of variables and responses, computational expense, and multiple objectives with uncertainty. As the number of input variables and responses increase, traditional methods of holding variables constant while changing others become extremely inefficient and do not provide deep insight to the problem space. As mentioned earlier, as computational power increases, the need for complex codes will continue to increase. In a preliminary design there might be a level of uncertainty associated with the requirements, which drives the need for regions of good designs instead of an optimal design point, which becomes increasingly complex as data and variables are added[1].

These continuing challenges have led designers to adopt approximate surrogates for high fidelity models known as metamodels. Unfortunately, developing good engineering metamodels comes with its own set of challenges. Metamodeling (the process of building metamodels) has produced new and improved developments as research has continued in this field. However, as pointed out by Wang[2], a gap appears to be growing between the research community and design engineers. Wang suggests that this widening gap is due to the mathematical involvement of metamodeling and that metamodeling evolves with information from multiple disciplines. Because of this, the research community needs to move its focus on metamodeling towards the needs of the design engineers. The goal with this research is to make not only building and using, but understanding metamodels easier for an entry level design engineer.

II. Metamodel Background

Metamodels use a set of sample data to build an approximate model of the function used to evaluate the sample data. The metamodel may then be used as a surrogate for the original function; this enables entirely new approaches

---

* Graduate Research Assistant, Department of Aerospace Engineering, cbaukol@calpoly.edu, Student Member.
† Lockheed Martin Professor, Department of Aerospace Engineering, ramcdona@calpoly.edu, Member.
‡ Undergraduate Research Assistant, Department of Computer Science, ndelmas@calpoly.edu.

to design and allows design studies to be carried out more easily, faster, and cheaper. There are a wide variety of metamodeling techniques available for use, each with its own characteristics, strengths, and weaknesses. Regardless of the technique being used, metamodels can play a role in the following: model approximation, design space exploration, problem formulation, and optimization support[2]. Model approximation is one of the main goals of metamodeling, and is discussed in the next section. Design space exploration not only allows the engineers to generate a better understanding of the design problem cheaply, but this area also has the potential for improved visualization of the design space. As the level of knowledge about a design space increases, the ability to efficiently formulate a specific design problem can be enhanced, and a problem that is easier to solve can be created. Lastly, various types of optimization problems that require computationally expensive operations can be run on inexpensive metamodels.

There are three main goals for any metamodel: 1) build a good approximation, 2) generate performance measures to assess the quality of the approximation, and 3) provide a confidence indicator for the estimated performance measures[3]. Without any one of these three parts, the ability to confidently use the metamodel drops dramatically.

The first goal, to build a good approximation, is essential because that is the entire purpose of creating a metamodel. It is designed to model the functionality of computationally intensive and expensive models in a cheap and efficient manner. However, if this approximation is not adequate, then there is no point in having a metamodel.

The second goal, to generate performance measures to assess the quality of the approximation, is crucial because it provides a metric which allows the user to get an understanding of the quality of the metamodel. This knowledge allows even a less experienced user to make changes to a metamodel and get an understanding of whether the changes they made have made things better or worse for the model. This also allows for the ability to run optimization techniques on the metamodel to get the best possible model for the given data.

The third goal, to provide a confidence indicator for the estimated performance measures, is important because it allows the user to obtain results relating how their estimated performance measurements compare to the true performance measurements. This is a challenge in practice because it can require additional analysis to obtain true performance, which can be very expensive.

III. Metamodeling Techniques

There are a variety of types of metamodeling techniques ranging from the very basic to highly advanced. Many people have actually used a basic metamodel without even knowing it when they have used linear interpolation. Linear interpolation can be used in N-dimensions by choosing $N + 1$ non-degenerate points in the neighborhood of the point of interest to specify a linear N-D function[4].

Another method that is commonly used is the response surface methodology[5]. This method fits first- and second-order polynomials to the system response. Meckesheimer states that a second-order response surface is not intended to fit well over the entire region of operability, but only in a relatively narrow region of interest that has been located by prior experimentation. The simplicity of using response surfaces makes them enticing, however their narrow scope can be a disadvantage if a large design space is being investigated, or if the behavior is known and is highly non-linear.

Radial basis functions use linear combinations of radially symmetric functions based on a Euclidean distance as a more heuristic approach to building a metamodel[5]. A simple, and commonly used form of a radial basis function is,

$$\varphi(\boldsymbol{x}) = \sum_{i} \beta_i \| \mathbf{1} + r^3 \| \tag{1}$$

where $\varphi(\boldsymbol{x})$ is replaced with $f(r)$ and the resulting system can be easily solved for β_i. This is a method that the authors are seeking to implement because of its relative ease, and promising results[6].

A more advanced technique of creating a metamodel is to use a Gaussian process, also known as Krieging models. A Gaussian process is a statistical generalization of the Gaussian probability distribution. It works by using Gaussian random functions to obtain function behavior at a finite number of points. The Gaussian process framework is sophisticated, consistent, and is computationally tractable[7]. The mathematics behind this process are by no means trivial, and the authors recommend turning to reference 7 for a detailed and encompassing discussion.

Regardless of what technique is being used to develop a metamodel, the model needs to learn how to behave, and a set of training data is used to do this. For radial basis functions, the training data is used to set the β_i coefficients, and for Gaussian processes, the training data is used to set the hyperparameters. In a Gaussian process, the model is specified by its mean function, and a covariance function (a function which looks at the covariance between responses at a pair of data points). More often than not a zero mean function is used, so the covariance function becomes the main component of the Gaussian behavior[7]. The parameters that control the covariance function are called hyperparameters. The training data points are the only points where the true model function value is known, and because of this, it is generally thought that the more training data one has, the better one's metamodel will be[4]. However, in a Gaussian process, the covariance matrix needs to be inverted during optimization, which can become computationally expensive, depending upon the number of training points. Because of this, this research will look at the number of training points being used in a metamodel and the associated error of the metamodel.

For advanced metamodeling techniques to become more widespread it is important to have them in an environment that is easily understood without too much of a learning curve. Of course, it is possible to develop a metamodeling tool which will insulate the user from all the mathematical complexity; unfortunately, such a tool would probably just create another black box, still depriving the user from any confidence in the resulting metamodel. The best way to learn a program is to use it and gain experience, but if a tool is too complicated, it can have a negative impact by driving people away from the tool due to frustration. It is up to the designers of the system tool itself to make a product that is manageable for its target user. The aim of this research is to create a user friendly metamodeling tool for design engineers to create, assess, and use advanced metamodels. This tool should not only be easy to use, but it should provide insight as to how the metamodel, and the math generating the metamodel, work and change.

IV. Test Functions

To perform the error analysis to develop the code, two different test functions were used. Both were classic optimization and metamodeling test cases. Two different functions were used because the two functions had different behavior and could provide insight into different aspects of the research problem. The first function was initially used by Osio and Amon as an example problem[8]. The test function is highly non-linear in one dimension, and linear in the other direction[8]. The function is given as:

$$y(X) = \cos\Big(6(x_1 - \qquad . \quad |x_1 - \ . \ |) + 2(x_1 - \ . \quad \frac{}{|x_1 - \ . \ | + \ .}\Big) + \ . \qquad . \tag{2}$$

A surface plot of this function showing its characteristics can be seen in Fig. 1. This test function was expected to be difficult to model because of the cusp behavior obtained from the absolute value terms $|x_1 - 0.7|$ and $|x_1 - 0.5|$. The modeling technique being used is not designed for discontinuous type behavior, and will most likely have error associated with this region. The second test function that was used was a two dimensional problem with quadratic behavior in each direction[9]. The equation for this problem can be found below and its surface plot can be seen in Fig. 2.

$$y(X) = \ . \qquad - x_1 x_2 - 7x_1 - 7x_2. \tag{3}$$

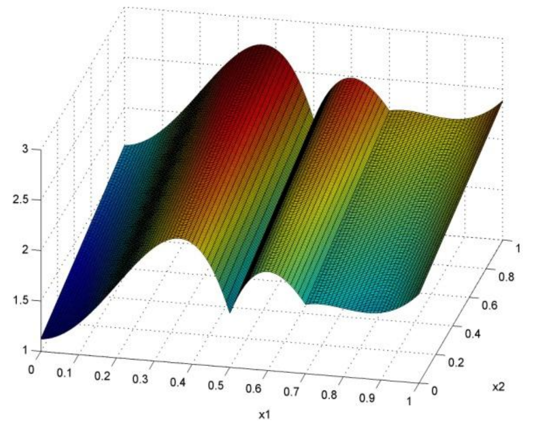


Figure 1. Test Function Surface Plot

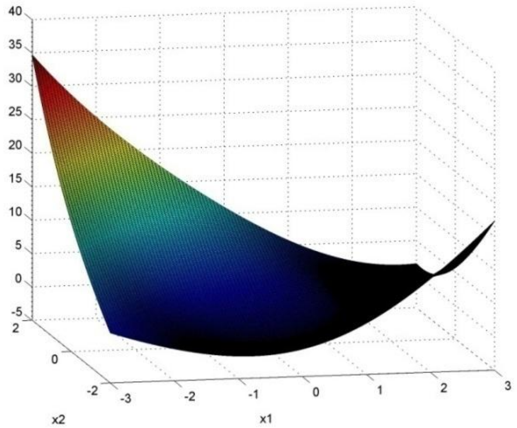


Figure 2. Second Test Function

For this problem, a set of training data was needed in order to train the metamodel. Looking at previous literature, the most common technique of choosing the locations for test points was to use a Latin Hypercube sampling design[10]. This has the statistical advantages of a Monte Carlo, but with better spatial separation when working with a set number of training points. However, this statistical significance disappears when an additional training point is added or removed because of the spatial separation. Because this research is looking at the differences in the number of training points used and what affect that has on the error of the metamodel, the number of training points will need to be varied. Because of this, the locations of the training points will be determined using a Monte Carlo sampling technique choosing random points. With this method, each point is just as random as the next, and the overall statistical significance of the model does not change[10]. The layout of all of the training points that were used can be seen in Fig. 3.

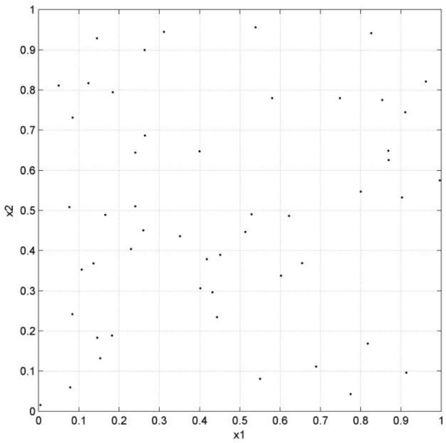


Figure 3. Monte Carlo Distribution of Points Used

V. Metamodel Assessment

While assessing the metamodel, there are many different aspects that can be looked at. The train of thought for this research was to first fix the number of training data points and perform error analysis purely on a large set of verification data using a root mean squared error method (RMSE)[3]. The verification, or test, data is fundamentally no different than the training data. The only difference is that none of the test data is used to create the metamodel. Because the true value is known at the test points, the test points are a useful tool for error analysis. Having large sets of test data might not seem useful because, as mentioned earlier, more training data should increase the fidelity of the model. However, this can become expensive computationally depending on the metamodeling technique being used, and in this case keeping those points for testing purposes becomes more useful. Using so many test points is a good method to try and estimate the true error of the metamodels, and is considered the gold standard by the authors. The next step is to look at the variation of the RMSE as the number of training points is decreased while using both fixed hyperparameters, and while using adaptive hyperparameters. The goal for this is to come to a conclusion about the number of training points needed before hyperparameters stop behaving differently and need not change anymore.

Calculating the RMSE is a computationally inexpensive process, but getting large amounts of test data can be problematic and is not practical for real-world applications. Because of this, alternative methods of error estimation are being investigated along with how they compare to the large test data standard. To perform this work, some aspects of the current metamodel interface are used, while some aspects are not yet included. In the latter cases, the authors are working on implementing the work into the metamodel interface and are using additional resources to perform a proof-of-concept.

To look at the true error of the metamodel, the predicted output values are plotted against the actual output values for all of the verification points. A straight line along $y = x$ is evidence that the actual and predicted values are similar. This can be seen for both test functions in Fig. 4 and Fig. 6. Another way to visually look at the error is to plot the residuals between the actual and predicted values. A well developed model will be a flat line with approximately an equal amount of variation in the points above and below the zero error line. The residuals for both test functions can be seen in Fig. 5 and Fig. 6. Even if the actual vs. predicted plots and the residuals do not look ideal, their behavior can help show how the metamodel is behaving, and in what areas it needs work.

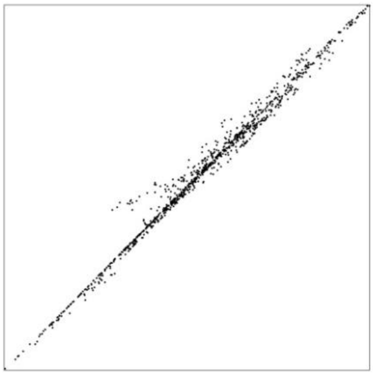


Figure 4. First Function Actual vs. Predicted

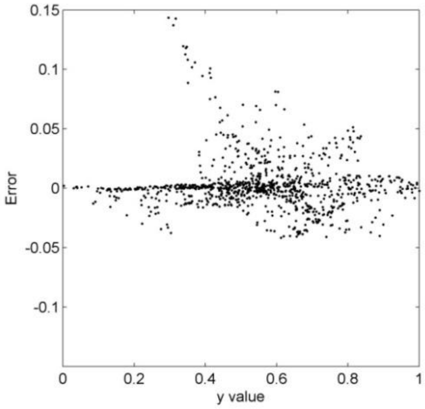


Figure 5. First Function Error Residuals

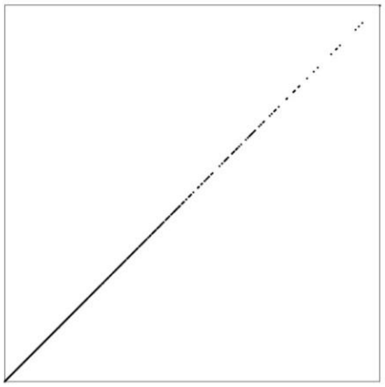


Figure 6. Second Function Actual vs. Predicted

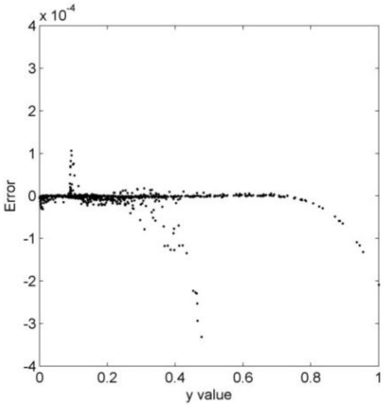


Figure 7. Second Function Error Residuals

As part of developing a program that can help designers create and validate metamodels, the error behavior associated with the number of training points needs to be investigated. The starting point for this research was to take the test functions and vary the number of training data used to generate the metamodels, and use a set of 1000 known test points to estimate the RMSE of the model. Even with this simple idea, there exist multiple ways of handling the process, specifically involving the hyperparameters. The authors decided to run three separate cases, each starting with all 51 training points and ending with only 2 training points: to fix the hyperparameters to their optimized values with the full set of points, to continuously re-optimize the training data using the previous values as the starting point, and finally to re-optimize the hyperparameters by reinitializing the initial starting point for each iteration. The RMSE was calculated for each case at each iteration and the trends can be seen in Fig. 8 and Fig. 9.

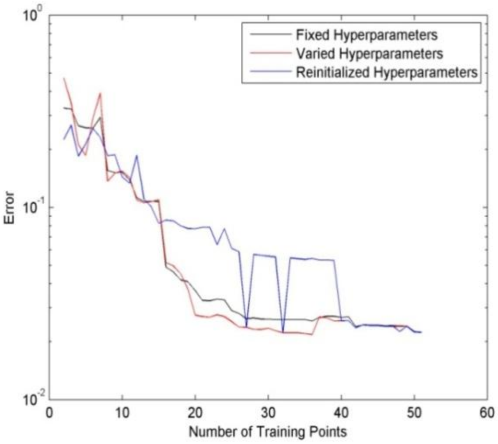


Figure 8. First Test Function RMSE

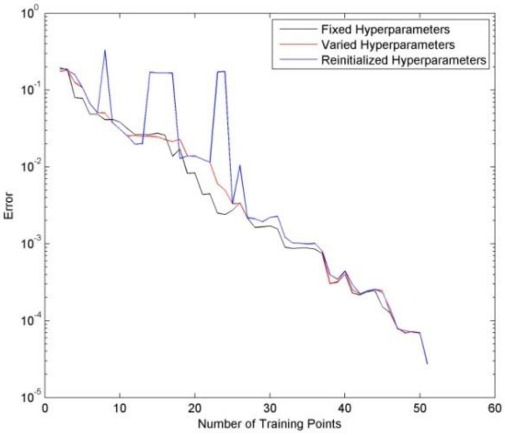


Figure 9. Second Test Function RMSE

These plots begin to give insight as to the behavior of the model error as the number of training points change. Both figures show that all three methods do follow the same trends, with the variation of the hyperparameters causing the only differences. The figures also show that after enough training points are obtained, the three different methods will converge to the same amount of error, showing that eventually, depending on the test function in question, equivalent hyperparameters are obtained, and the remaining error is purely due to the number of training points.

It was noted while looking at Fig. 8 that the error was not actually converging to zero, but was converging to a specific value greater than zero. Because this test function is only in two dimensions, this curious aspect could be visually investigated. Surface plots of the model were generated for three different sets of training points. A model with 15 training points and with 16 training points were looked at because adding the 16th training point caused a large drop in the error. A surface using all 51 training points was also looked at to help see why the model error did not converge to zero. The original surface from Fig. 1 and the three metamodel surfaces can be seen in Fig. 10 through Fig. 13.

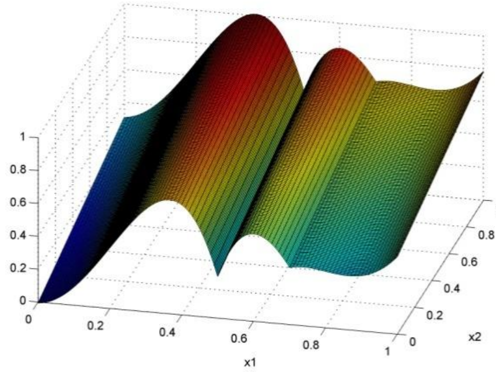


Figure 10. Original Test Function

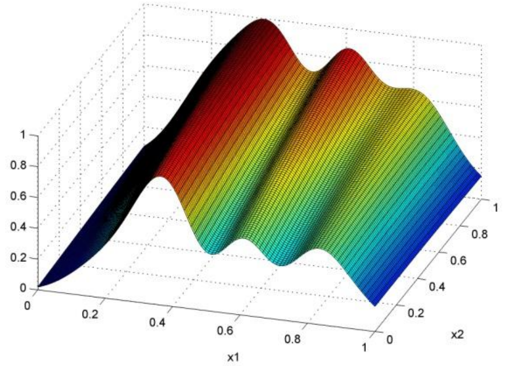


Figure 11. Metamodel Using 15 Training Points

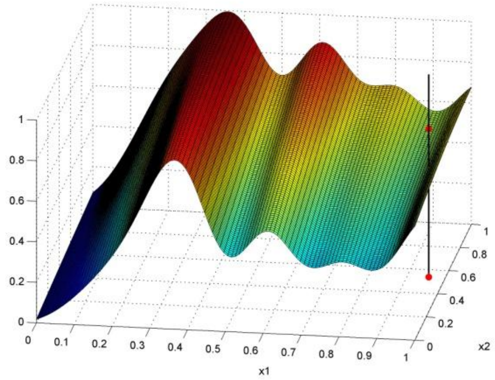


Figure 12. Metamodel Using 16 Training Points

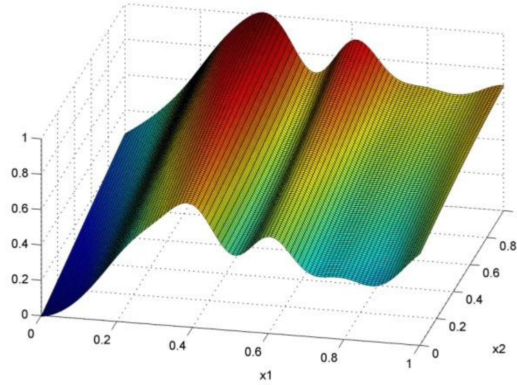


Figure 13. Metamodel Using 51 Training Points

The difference between the 15 training point metamodel and the 16 training point metamodel can be clearly seen in looking at Fig. 11 and Fig. 12. The large drop in error is associated with the right hand side of the surfaces, and is due to adding a training point in that region. The difference between the 16 training point model and the 51 training point model is harder to see, but does still exist. Looking at the 51 training point model and the original test function it is clear that the model has difficulty modeling the areas of discontinuity. Because of this, there will always be some type of associated error at those areas. A surface plot of the error between the original test function and the 51 training point model showing the sharp peak in error at the cusp can be seen in Fig. 14. Methods of correcting this type of error will be discussed in a later section. This error surface plot also shows a large portion of error in one of the corners of the data field. This large portion of error is not due to function behavior, but is actually because of a lack of training points in that region, which can be seen in the training point distribution from Fig. 3.

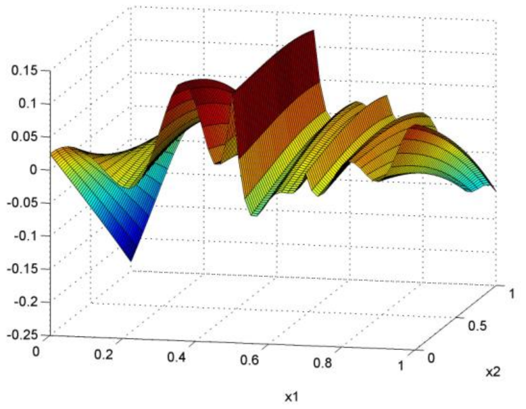


Figure 14. Surface Plot of Metamodel Error

For many instances it is not practical to perform a full RMSE test with hundreds or thousands of test points because in many cases the test points do not exist, and if they do, they can be used to help build the model. Because of this, an effective and computationally inexpensive error estimation method needs to be explored and implemented. Two methods that are growing in use and popularity are the take-1-out method and the take-*k*-out method[3]. Once again, there are multiple methods of performing the take-1-out method, including using one random point, or one random point m number of times, the most recently added point, or all points.

The method of taking one random point *m* number of times was not looked at because that method only makes sense for extremely large training set sample sizes[11]. This is because of the equation for standard error, which for this method would be $1/\sqrt{m}$. This means if the user desires to estimate the error within 10%, a sample size of 100 is needed. This is impractical with the current set of 51 training points being used, but could become a useful tool if the number of training points becomes extremely large. This is not currently the case, and will need to be looked at as this research continues. The other methods of handling the take-1-out routine were looked at and can be seen in Fig. 15 and Fig. 16.

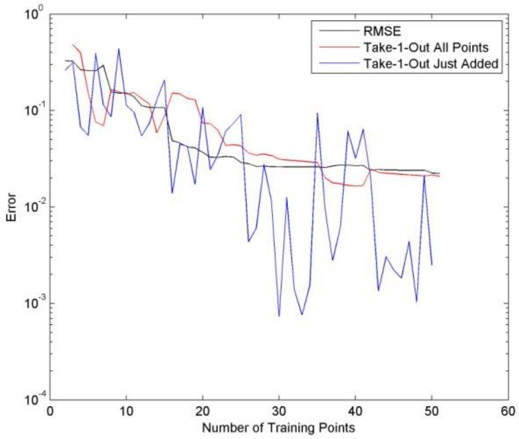


Figure 15. First Test Function Take-1-Out Error

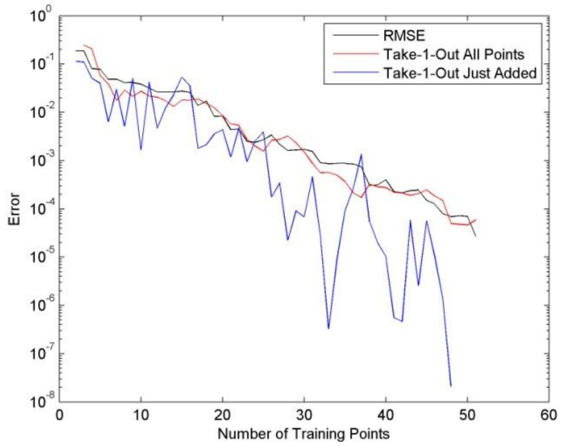


Figure 16. Second Test Function Take-1-Out Error

Two things can be learned by looking at Fig. 15 and Fig. 16. The most notable aspect is that the take-1-out error estimation method produces remarkably similar results to the RMSE method using 1000 test data points not only in the general trend, but in the error value as well. This is considered notable because of the type of error being measured with the take-1-out routine. In the take-1-out method, one point is removed, a model is built without the point and the error at that point is calculated, the point is then placed back in and the next point is analyzed until all of the points have been looked at. Once each point is placed back in, the error that was calculated at that point drops to zero. Because of this the error at each point is meaningless, but the RMSE type sum of the error at all of the points is a strong indication of the overall error for the model.

The next aspect of Fig. 15 and Fig. 16 that is notable is the behavior of the 'just added' method. This method started with only two training data points, and would then use the third as a test point, then add it into the model and use the fourth as a test point, and so on. The error in this method looks noisier because this method is purely looking at only one test point and not an average. Expectedly, sometimes the model gets lucky and there is not much error in the one point, and sometimes the model gets unlucky. The usefulness of this method is that computationally it is cheap to run through a large amount of training data. It is the author's suggestion that this method be used to gauge when the metamodel is coming within an order of magnitude or two of the desired error. Even with the noise like behavior, the general trend of the error can still be seen. It is then the authors' suggestion that a take-1-out routine using all training data points be used to estimate the error of the model overall.

At this point in the research, the take-k-out method has not been explored in depth. The same challenges that exist with the take-1-out method exist in the take-*k*-out method, but some additional problems are added. Whereas using all of the training points can be desirable in the take-1-out method, translating this method means taking every combination of k out. As k increases, every combination of k increases by $m!$. This can quickly become a very computationally expensive method if left unchecked and alternative methods of error estimation should be looked at. Some literature even suggests that the take-1-out method is superior to the take-*k*-out method, and only take-1-out should be used[3].

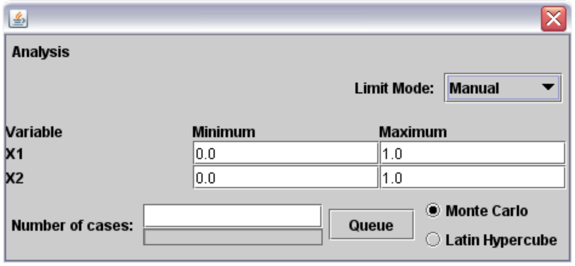
VI. User Friendly Metamodel Interface

A. General Metamodel Interface

The Gaussian process metamodel described above has been implemented in a program with a graphical user interface. This interface is first used for developing and tuning a metamodel, then the interface may be used for exploring the metamodel. This section highlights the primary features of the interface. The metamodel interface provides for controlling all aspects of the metamodel behavior as well as a mechanism appropriate for facilitating decision making about how the metamodel should be changed based on its behavior.

When the metamodel is first initialized, it not only allows existing metamodels to be opened, but it allows the user to create a metamodel from scratch. A metamodel can be created by two different methods, one that automatically gets created using an analysis server such as Phoenix Integration's ModelCenter[12], and one that the user manually creates. When manually creating a metamodel, the metamodel interface requires basic information such as the number of inputs and the number of outputs from the data. From there, the input and output parameters can be edited to set the minimum, the maximum, and other aspects. The training data can also be easily added manually to a new model simply by copying and pasting data from a spreadsheet, such as Excel, into the training data spreadsheet in the metamodel interface.

If a designer is creating a brand new metamodel from scratch, and there are no current data points to use for training, the metamodel interface can easily queue a batch of points. The designer just needs to set the ranges for the different input values, the number of points to be queued, and then the designer can choose whether to use a Monte Carlo or a Latin Hypercube sampling technique. An image of this interface can be seen in Fig. 17. This batch queuing is directly integrated to work with an analysis server, and can solve for the desired data points directly from the user interface.



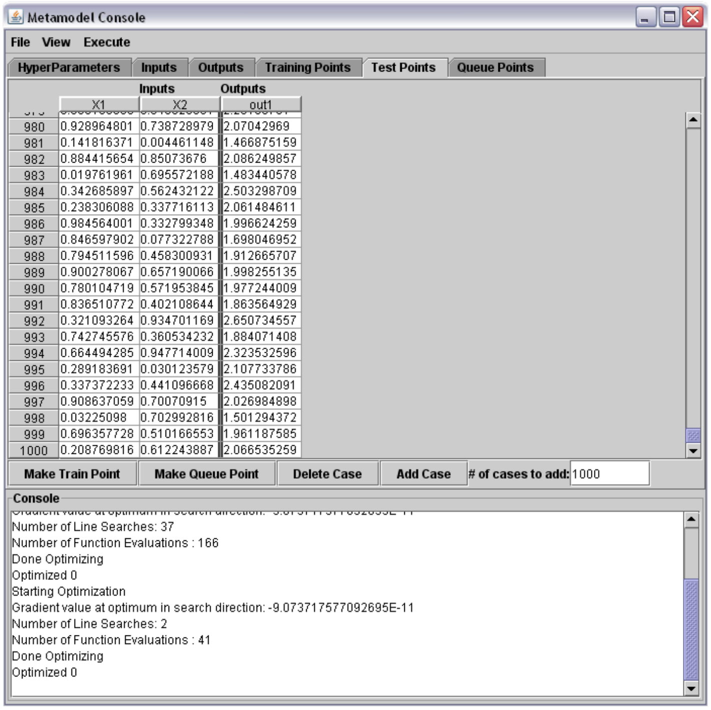
Limit Mode: Manual

Variable	Minimum	Maximum
X1	0.0	1.0
X2	0.0	1.0

● Monte Carlo
○ Latin Hypercube

Figure 17. Batch Queuing

Once a metamodel has been created and saved to an XML file, it can be easily accessed when the metamodel interface is initialized. The overall user interface contains a toolbar at the top which acts in a familiar way to any computer user, and is how features such as saving models, running test cases, and opening the metamodel explorer are accessed. Below this toolbar are a series of tabs that can switch the metamodel interface from displaying information about the inputs, the outputs, the training points, test points, and queued points. All of the information within these tabs is displayed in spreadsheets below the tabs. Underneath where the spreadsheets are displayed, there are a series of buttons that control the data points, such as adding and removing cases, within the current spreadsheet. At the very bottom of the metamodel interface is an output only console which displays information about the different actions being taken.



	Inputs		Outputs
	X1	X2	out1
980	0.928964801	0.738728979	2.07042969
981	0.141816371	0.004461148	1.466875159
982	0.884415654	0.85073676	2.086249857
983	0.019761961	0.695572188	1.483440578
984	0.342685897	0.562432122	2.503298709
985	0.238306088	0.337716113	2.061484611
986	0.984564001	0.332799348	1.996624259
987	0.846597902	0.077322788	1.698046952
988	0.794511596	0.458300931	1.912665707
989	0.900278067	0.657190066	1.998255135
990	0.780104719	0.571953845	1.977244009
991	0.836510772	0.402108644	1.863564929
992	0.321093264	0.934701169	2.650734557
993	0.742745576	0.360534232	1.884071408
994	0.664494285	0.947714009	2.323532596
995	0.289183691	0.030123579	2.107733786
996	0.337372233	0.441096668	2.435082091
997	0.908637059	0.70070915	2.026984898
998	0.03225098	0.702992816	1.501294372
999	0.696357728	0.510166553	1.961187585
1000	0.208769816	0.612243887	2.066535259

# of cases to add: 1000

```
Number of Line Searches: 37
Number of Function Evaluations : 166
Done Optimizing
Optimized 0
Starting Optimization
Gradient value at optimum in search direction: -9.073717577092695E-11
Number of Line Searches: 2
Number of Function Evaluations : 41
Done Optimizing
Optimized 0
```

Figure 18. Metamodel Interface with Test Points

A depiction of the metamodel interface immediately after a typical use case is included in Fig. 18. The function being modeled is a function of two variables out1 = out1(X1,X2). In this use case, 1000 test case points have been added and it can be seen in the console that optimization has been performed.

The hyperparameters tab contains a spreadsheet with all of the information regarding the hyperparameters for the given model. The spreadsheet has one column for each response and one row for each hyperparameter. There are two base hyperparameters and one for each input direction. If the spreadsheet grows beyond the size of the panel, scroll bars appear to allow the user to navigate the entire spreadsheet. The data scrolls, but the row and column headers remain visible. The data in the spreadsheet can be altered manually, or the hyperparameters can be optimized by clicking on the response column that is desired. All of the responses can be optimized without clicking each column simply by executing the “Optimize All” command in the toolbar.

The gradient based optimizer used in this research is rather verbose and outputs numerous diagnostic messages to the console. It uses a Polak-Ribiere search technique along with Wolfe-Powell conditions to reach its optimum value[13]. The optimizer can be re-run after an optimum has been found to confirm that it has found the optimum point. The minimum number of line searches that are executed during the optimization routine is 2, while the minimum number of function evaluations that are performed is 41. The appearance of these numbers in the console along with a steady value for the gradient value in the search direction confirms when an optimum point has been obtained. In general, it should only take one optimization run to find a near optimum point.

The Input and Output tabs are easily accessed from the main display of the metamodel interface and, as discussed earlier, it is within these tabs that the specific information about each input and output quantity, such as a maximum and a minimum value, can be entered. In the Execute tab, there is an option that can be used to reset the normalization constants. When executed, a confirmation dialog appears, reminding the user of the implications of resetting the normalization parameters. If the user chooses to continue, the program automatically sets the normalization parameters based on the range of the variables observed in the data set. Typically, the exact settings of the normalization parameters are not important and they only need to be set when the data set is initialized. If, however, in the process of performing a study, sufficient data points are added to change a representative value or the range of values of the data set by an order of magnitude, the normalization should be reset. This will cause the hyperparameters to lose their validity and they must be re-optimized.

The next three tabs contain spreadsheets with information about the training points, the test points, and queued points. The different types of points are designed to be rather self explanatory; however, there are some underlying differences. Cases can easily be added to any type of point simply by creating more rows in the desired spreadsheet, and either manually entering data, or copying and pasting data. Cases can also be added to the queued points spreadsheet from within the metamodel explorer, which will be addressed later in the paper. The interface does allow for movement between the different types of points. Training points and test points can be easily interchanged, whereas queued points can become either training or test points after the queued point has been

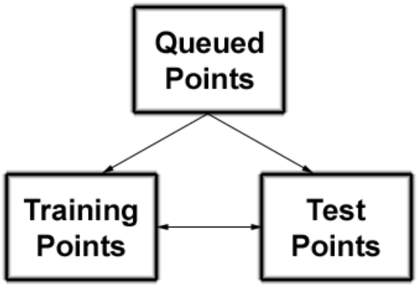
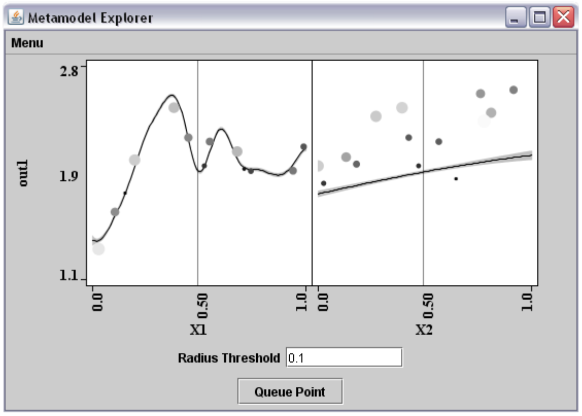


Figure 19. Point Movement

evaluated. This movement is characterized in Fig. 19.

B. Metamodel Explorer

In the View tab of the metamodel interface toolbar, there is an option to launch a metamodel explorer interface. An example of the metamodel explorer interface corresponding to an optimized state can be seen in Fig. 20. The metamodel explorer represents a single component as a function of its local variables. The gray band behind the response curve in the metamodel explorer represents the variance of the Gaussian random function used in the metamodel. Furthermore, the response value throughout the input space is indicated by a line crossing each box in the grid. The line may be thought of as a slice through the response surface. It represents the function value obtained by varying that particular input (per the column) while holding all other inputs constant. It may be thought of as a variation in the same sense as a partial derivative. In Fig. 20, the X1 component clearly follows the data set whereas X2 appears to be lower than the displayed points. This is actually because the X1 hairline is near the bottom of the test function's cusp. If the hairline is moved to the right in the flat portion of the X1 variable, the X2 portion will appear to go directly through almost all of the displayed points. If the X1 hairline is moved to the left in the linear region, the displayed training points will appear to be evenly distributed above and below the model.



Radius Threshold 0.1

Figure 20. Metamodel Explorer Interface

Figure 21 depicts the slices through the domain presented by the metamodel explorer. The cutting plane of each slice is shown and the curve where the plane cuts the function surface is highlighted. This three-dimensional surface plot is very effective for a function of two variables, but is not possible for higher-dimensional problems. The metamodel explorer extends effectively to problems of any dimensionality.

The gray band surrounding this line indicates the variance of the Gaussian process. The variance does not correspond directly to the error of the underlying metamodel. Loosely speaking, it represents how well the current Gaussian process (hyperparameter settings) agrees with the training data. A good metamodel will have a very thin (nearly nonexistent) band. However, if there is insufficient training data, a metamodel may nevertheless have a vanishingly thin band but still not do a good job of predicting the response throughout the domain. A thin variance band is necessary but not sufficient for a model to generalize well.

In a one-dimensional problem with only two training points, the Gaussian process will have a high degree of confidence in a model with linear behavior, even though other more complex models could explain the observances equally well. If the underlying function is actually more complex, say cubic, this linear model will be a poor predictor. If one increases the training set to include four data points, the Gaussian process will have a low degree of confidence in a model with linear behavior, as it will not agree with the observations[4]. Furthermore, the Gaussian process will have a high degree of confidence in a model with cubic behavior, as it is the simplest model that agrees with the observations. The addition of more training points (which agree with the cubic model) further cements this confidence. In essence, choose the simplest model which agrees with the observations.

Finally, the metamodel explorer contains a depiction of the training points used to define the metamodel. In a one-dimensional problem, the concept of a slice through the function space becomes degenerate and plotting the training points is straightforward. However, in a multi-dimensional problem any arbitrary slice through the function space is unlikely to pass through any training points. This can be seen in Fig. 21 which also depicts a set of training data throughout a two-dimensional domain. Of course, simply plotting all of the training points on each of the charts would present so much information that the trends indicated by the data would not be visible. Consequently, for multi-dimensional problems, the points displayed are selected based on how close they are to the slice through the input space. The displayed points may be interpreted as being those points particularly relevant to defining the metamodel in the neighborhood of the slice. Points far from the slice do not play a strong role in defining the metamodel near the slice, and therefore do not need to be represented.

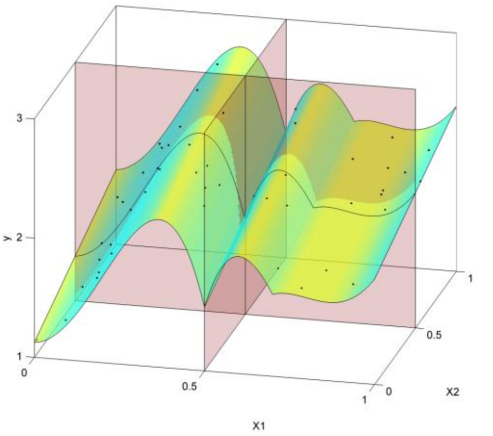


Figure 21. Explorer Slices and Training Points

This method of displaying a multi-dimensional metamodel in conjunction with the contributing training data represents an advance in visualization for understanding any kind of metamodel. This intuitive display is simple to implement and should be incorporated into existing metamodeling methods and tools.

The radius threshold for selecting which points are displayed can be adjusted by the user through the entry field below the metamodel explorer. The radius threshold is a non-dimensional constraint that controls how close a point must be to a slice in order to be displayed. Because a point may be close to a slice in one dimension, but not in others, the points displayed in each column may not appear in the other columns. The data points are represented by gray circles, the size and intensity of which varies with the distance from the slice. Points that are very close to the slice appear as small black dots while points that are further from the slice appear as larger gray circles. Points far from the slice, near the limit of the constraint, appear as large, nearly white circles.

This point scaling is consistent with selecting points important to the definition of the metamodel in the neighborhood of a slice. Points nearest the slice (black point) play a strong role in determining the exact behavior of the metamodel in that region. Points further from the slice (gray circle) play a lesser role in determining the approximate behavior of the metamodel in that region. Points near the constraint limit (off-white circle) and beyond (white or invisible) play a very weak role in determining the rough behavior of the metamodel in that region.

The user may adjust the radius parameter to get a satisfactory display; this setting will depend on the number of data points in the training set and on the dimensionality and linearity of the problem. The parameter should be adjusted such that the data points depict the expected behavior of the underlying function. Too small a radius will not display sufficient points; too large a radius will include points from far away in the design space, confusing the source of the variation. Experience has shown that this technique for representing multi-dimensional training data produces good results for problems of moderate dimensionality (roughly six dimensions) and nonlinearity[4]. While more work is needed to produce intuitive displays for high-dimensional problems, the visualization advance developed in this research is a dramatic improvement over the traditional solution limited to one-dimensional problems.

Returning to Fig. 20, below the radius threshold control there is a button that allows the user to queue points to be added to the queued points set. When the button is pressed a dialogue box appears with the current hairline settings in a series of entry fields. The user may adjust the point and then queue the specified point when satisfied. This allows the user to identify a region of the input space that has been inadequately covered, due to insufficient points, or due to the local behavior of the function needing more points. This is another example of providing the user with a simple tuning parameter and the interface tools needed to make good decisions quickly. This man-in-the-loop adaptation capability is a key benefit of using a local metamodel.

VII. Future Work

This project is a work in process, not a finished piece. With this being said, the authors have a clear vision of where this project is heading, and what needs to be added. The future work can be split into two main ideas: interface additions, and interface modifications. These two ideas are similar, but include slight differences.

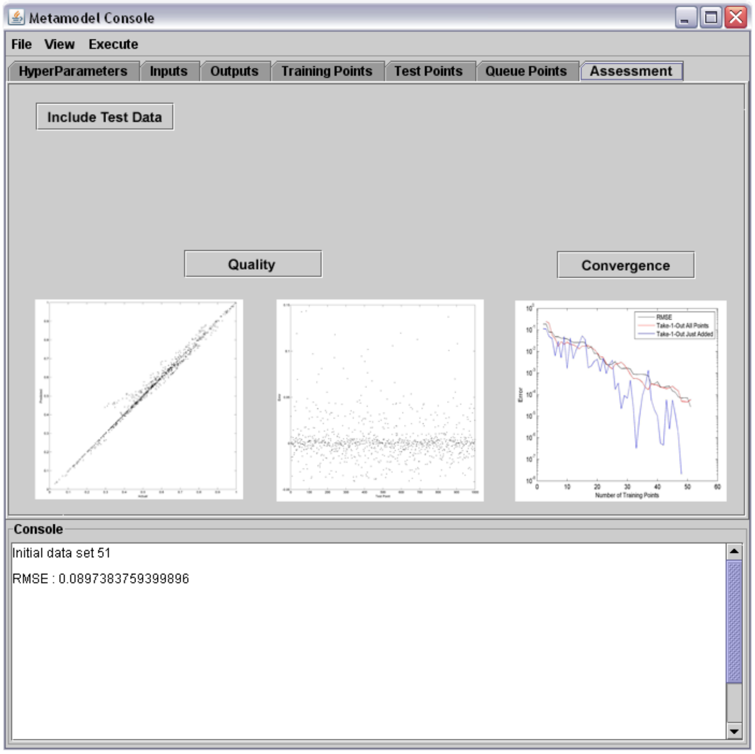
A. Metamodel Additions

Currently the only type of metamodel that is supported is a Gaussian Process model. The Gaussian Process is a very powerful tool, but in some cases is overkill, and can have the unwanted effect of slowing down the modeling and design process. In these cases, it would be nice to take special cases of the Gaussian process, such as radial basis functions, and use them instead. This simplified approach might make the design process faster and more efficient.

Another similar addition that will be implemented will be to allow for different covariance functions. The current Gaussian Process uses an exponential covariance function, which works well as an initial guess, but if knowledge about the behavior of the data is known, it is more beneficial to use a covariance function that follows the behavior being modeled. Different covariance functions could lead to being better able to model cusps such as the cusp in the first test function. The goal is to make each hyperparameter adjustable to account for specific variations in one dimension. For example, if one dimension of the problem being looked at is oscillatory, it would be beneficial to use some type of sinusoidal covariance function in that dimension.

Having these different components in the metamodel interface has the potential to intimidate users who might not be completely familiar with their data and what works best with it. In order to create a user friendly environment, a metamodel "wizard" is being developed. This wizard will walk the user through a series of simple steps to determine which kind of model would be best to start with. If the selected model is not adequate, or if the user wants to try a different model, this can be easily done through the rest of the metamodel interface.

Another addition that is currently being developed is to move all of the error assessment metrics discussed in this paper to the metamodel interface. This will be implemented as another tab and will include options for both checking the convergence and the quality of the metamodel. Much thought is being put into the user interface of this specific area to not only make things implicit, but to also make them all-encompassing for the many different methods of handling error estimation. An early idea of how this assessment tab might look can be seen in Fig. 22.



```
Initial data set 51
RMSE : 0.0897383759399896
```

Figure 22. Metamodel Assessment Interface

B. Metamodel Modifications

The first enhancement that needs to be made to the metamodel interface is the ability to handle large data sets. Currently, the optimizer can have difficulty because of the behind-the-scenes math that is being performed and the size of the matrices that are used. Different solutions are being looked at, including looking at moving away from Gaussian processes in these cases, and breaking down the training points into different subsets, but more research is needed.

The general layout of the hyperparameters being used is also in the process of changing. Currently, the hyperparameters are displayed in a spreadsheet form as a series of numbers attached to each response. Unfortunately, the numbers don't really describe anything in particular to most users. Instead, it is much more intuitive to use a graphical interpretation of the hyperparameters, such as a length scale with a slide bar, in order to show which parameters have a stronger affect on which responses.

The final modification that is being made is to have better integration with outside programs. These programs include ModelCenter, Matlab, Excel, and can possibly be tailored to work with other future or specialty programs.

VIII. Conclusion

The metamodeling community has put forth a lot of work and much effort to make metamodeling a practical engineering device. This research is an attempt to help bridge the gap between researchers and designers by creating a useful and intuitive tool which not only helps develop and understand a metamodel, but also provides cheap and reliable quality measures. As with any developing field, advancements will continuously need to be made to help further the field’s adaptation and implementation. The research being performed is a step in the right direction, and will hopefully be used as a catalyst for others to investigate and frequently use metamodeling. It is the opinion of the authors that current aspects of this research, such as the point location in the prediction profiler, should be put into any future metamodeling tools. As far as error assessment is concerned, using a combination of the just added and all points methods of the take-1-out routine should be used for the most computationally efficient convergence and quality assessment.

Acknowledgments

The authors gratefully acknowledge Craig Nickol, the primary technical monitor for the National Aeronautics and Space Administration (NASA) NASA Research Announcement Grant NNX07AO14A, which supported this work.

---

[1] Koch, P. N., Simpson, T. W., Allen, J. K., and Mistree, F., 1999, “Statistical Approximations for Multidisciplinary Design Optimization: The Problem of Size,” J. Aircr., **36**(1), pp. 275-286.

[2] Wang, G. G., Shan, S., “Review of Metamodeling Techniques in Support of Engineering Design Optimization,” *Journal of Mechanical Design*, Vol 129., pp.370-380.

[3] Meckesheimer, M., Booker, A. J., Barton, R. R., and Simpson, T. W. “Computationally Inexpensive Metamodel Assessment Strategies.” AIAA *Journal*, 40(10):2053–2060, 2002.

[4] McDonald, R. A., *Error Propogation and Metamodeling for a Fidelity Tradeoff Capability in Complex Systems Design*. PhD. Dissertation, Georgia Institute of Technology. August 2006.

[5] Meckesheimer, M., Barton, R. R., Simpson, T., Limayem, F., Yannou, B., “Metamodeling of Combined Discrete/Continuous Responses,” AIAA *Journal*., 39(10):1950-1959, 2001.

[6] Bors, A., *Introduction of the Radial Basis Function (RBF) Networks*. University of New York. New York.

[7] Rasmussen, C. E., Williams, C. K. I., *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] Osio, I. G., Amon, C. H., “An Engineering Design Methodology with Multistage Bayesian Surrogate and Optimal Sampling,” *Research in Engineering Design*, **8**(4), pp. 189-206, 1996.

[9] Jin, R., Chen, W., Simpson, T. W., “Comparative Studies of Metamodeling Techniques Under Multiple Modeling Criteria,” *Struct Multidisc Optim*., pp.1-13., 2001.

[10] Keane, A. J., Nair, P. B., *Computational Approaches for Aerospace Design: The Pursuit of Excellence*., John Wiley & Sons Ltd., West Sussex, 2005.

[11] Devore, J., Farnum, N., *Applied Statistics for Engineers and Scientists*. Thomson Brooks/Cole. Belmont. 2005.

[12] Phoenix Integration, Inc., ModelCenter Ver. 7.1.2, August, 2007.

[13] Fletcher, R., *Practical Methods of Optimization*. John Wiley & Sons Ltd. West Sussex. 1987