

# New Approaches to Conceptual and Preliminary Aircraft Design: A Comparative Assessment of a Neural Network Formulation and a Response Surface Methodology

Debora D. Daberkow and Dimitri N. Mavris  
Georgia Institute of Technology

1998 World Aviation Conference  
September 28-30, 1998  
Anaheim, CA

---

**SAE** *The Engineering Society*  
*For Advancing Mobility*  
*Land Sea Air and Space*<sup>®</sup>  
**INTERNATIONAL**

SAE International  
400 Commonwealth Drive  
Warrendale, PA 15096-0001 U.S.A.



American Institute of Aeronautics  
and Astronautics  
370 L'Enfant Promenade, S.W.  
Washington, D.C. 20024

Published by the American Institute of Aeronautics and Astronautics (AIAA) at 1801 Alexander Bell Drive, Suite 500, Reston, VA 22091 U.S.A., and the Society of Automotive Engineers (SAE) at 400 Commonwealth Drive, Warrendale, PA 15096 U.S.A.

Produced in the U.S.A. Non- U.S. purchasers are responsible for payment of any taxes required by their governments.

Reproduction of copies beyond that permitted by Sections 107 and 108 of the U.S. Copyright Law without the permission of the copyright owner is unlawful. The appearance of the ISSN code at the bottom of this page indicates SAE's and AIAA's consent that copies of the paper may be made for personal or internal use of specific clients, on condition that the copier pay the per-copy fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. This consent does not extend to other kinds of copying such as copying for general distribution, advertising or promotional purposes, creating new collective works, or for resale. Permission requests for these kinds of copying should be addressed to AIAA Aeroplus Access, 4th Floor, 85 John Street, New York, NY 10038 or to the SAE Publications Group, 400 Commonwealth Drive, Warrendale, PA 15096. Users should reference the title of this conference when reporting copying to the Copyright Clearance Center.

ISSN #0148-7191

Copyright© 1998 by SAE International and the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

All AIAA papers are abstracted and indexed in International Aerospace Abstracts and Aerospace Database.

All SAE papers, standards and selected books are abstracted and indexed in the Global Mobility Database.

Copies of this paper may be purchased from:

AIAA's document delivery service  
Aeroplus Dispatch  
1722 Gilbreth Road  
Burlingame, California 94010-1305  
Phone: (800)662-2376 or (415)259-6011  
Fax: (415)259-6047

or from:

SAExpress Global Document Service  
c/o SAE Customer Sales and Satisfaction  
400 Commonwealth Drive  
Warrendale, PA 15096  
Phone:(724)776-4970  
Fax: (724)776-0790

SAE routinely stocks printed papers for a period of three years following date of publication. Quantity reprint rates are available.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without prior written permission of the publishers.

Positions and opinions advanced in this paper are those of the authors(s) and not necessarily those of SAE or AIAA. The author is solely responsible for the content of the paper. A process is available by which discussions will be printed with the paper if it is published in SAE Transactions.

# New Approaches to Conceptual and Preliminary Aircraft Design: A Comparative Assessment of a Neural Network Formulation and a Response Surface Methodology

Debora D. Daberkow and Dimitri N. Mavris

Georgia Institute of Technology

Copyright © 1998 Society of Automotive Engineers, Inc.

## ABSTRACT

This paper critically evaluates the use of Neural Networks (NNs) as metamodels for design applications. The specifics of implementing a NN approach are researched and discussed, including the type and architecture appropriate for design-related tasks, the processes of collecting training and validation data, and training the network, resulting in a sound process, which is described. This approach is then contrasted to the Response Surface Methodology (RSM). As illustrative problems, two equations to be approximated and a real-world problem from a Stability and Controls scenario, where it is desirable to predict the static longitudinal stability for a High Speed Civil Transport (HSCT) at takeoff, are presented. This research examines Response Surface Equations (RSEs) as Taylor series approximations, and explains their high performance as a proven approach to approximate functions that are known to be quadratic or near quadratic in nature. It also reveals why this approach fails for large numbers of variables and extended ranges, and how it can be deceptive. Neural Networks prove to be a more suitable alternative with improved performance over RSEs in such cases, provided they are trained and assessed appropriately.

## INTRODUCTION

In aerospace engineering, sound design methodologies that can handle a high degree of complexity become crucial as aerospace systems become more complex and aircraft design grows more challenging. A paradigm shift in this area that has been widely discussed is to achieve more knowledge about the design early in the design stages. This helps to keep the design "open" as long as possible and decisions can be made with as much knowledge as possible.<sup>1</sup>

One way to achieve the goal of keeping the design 'open' is through a parametric rather than a fixed product representation. Such an approach calls for parametric studies. These can quickly become extremely time

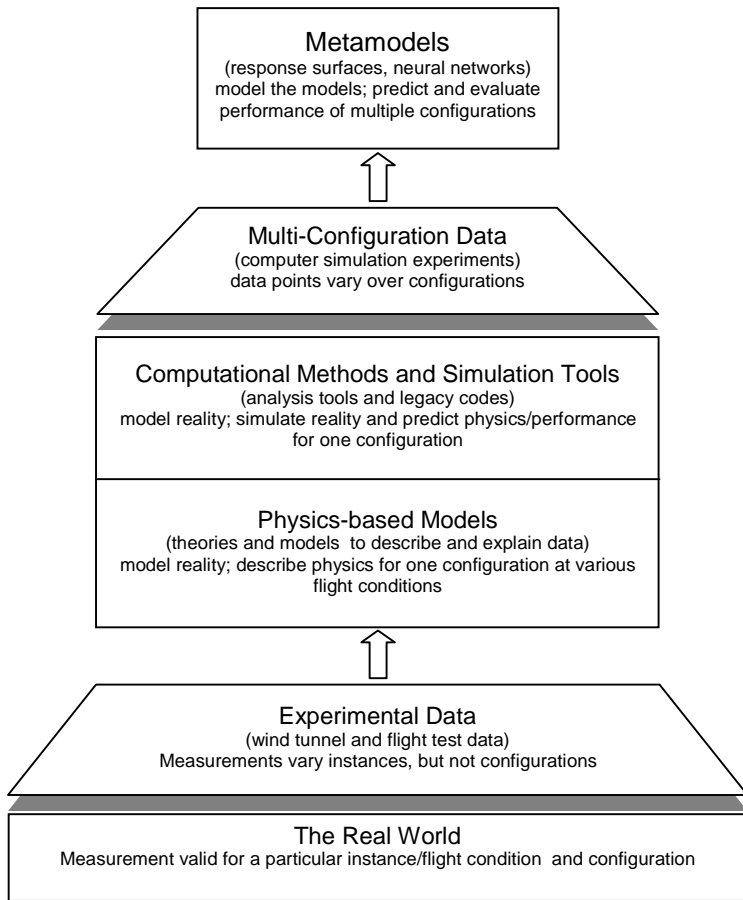
consuming and computationally extensive and thus thought must be given to find ways of achieving the desired results while keeping the computational effort at a reasonable level.

A solution to this problem is the use of metamodels, approximations of certain chosen responses in terms of the design parameters. Typically, for a specific study or task a metamodel would be created using certain data points and then configured to predict the desired response for any combination of parameters. Therefore, an important step in evaluating a design space is the formulation of metamodels. The creation of a metamodel is not a trivial task, and many aspects and assumptions go into formulating one. Thus, it is necessary and informative to assess different possible types of metamodels, in terms of their accuracy, generalization properties, efficiency and specifics to setting up a particular model in the most efficient manner.

## METAMODELS IN AIRCRAFT DESIGN

Metamodels, in general, are similar to the mathematical models, which are developed and used in science to describe physical phenomena. They are concerned with the information that can be expressed during the modeling process, and thus referred to as information models.<sup>2</sup> While a physics-based model is intended to describe and/or explain actual data that has been collected, metamodels are different in that as information models, they attempt to model the *model*, rather than the actual data itself. This is illustrated in Figure 1.

Metamodels can be employed in several ways to be beneficial in various fields and for many different purposes. In aircraft design, and in the design methodologies for complex systems in general, the place for a metamodel implementation arises from a paradigm shift that is underway. Here, the focus is to make significant decisions when the most knowledge about the design is available, thus committing the cost later in a design process and avoiding costly re-designs.<sup>1</sup>



**Figure 1: Metamodels in Aircraft Design**

In keeping with this paradigm shift, the desire arises to study configurations in a parametric fashion. To span the entire design space that is set up through such an approach, usually a large number of different configurations need to be analyzed and evaluated. This poses a problem, since a typical analysis for just one configuration can be quite involved and require a significant amount of time. In many cases, this may render impractical, or even impossible.

*A solution is to directly investigate the relation between the design parameters and selected representative responses to build an approximation that models this behavior both in a qualitative and quantitative way.* These approximations could provide much faster results. Thus, for studies involving the evaluation of many design configurations, this may be the only practical solution.

Such an approximation model can be referred to as a metamodel, since it refers to the information revealed in a modeling process rather than a physical phenomenon directly. Having established the necessity and role of metamodels in the design process, the question arises of how a metamodel should be evaluated.

*The requirement that any designer building a metamodel would strive to fulfill is that of maximum efficacy, where efficacy is the combination of effectiveness and*

*efficiency.*<sup>3</sup> For an approximation metamodel, to be effective means, primarily, to yield a good, accurate representation of the approximated response. Typically, such a representation is based on certain data points, and then these data points are approximated to yield the representation.

Maximum *efficiency* is reached, when a minimum of data points is required to achieve an acceptable representation, since these data points are each actual design configurations, often referred to as cases. In order for the metamodel to be *effective*, it must exhibit a good representation of the underlying, unknown function. This representation should satisfy several criteria; it must be accurate, it must have good generalization properties, and it is desirable to have reasonable extrapolation properties.

So the first requirement for a metamodel to be effective is related to the accuracy, meaning it needs to fit the given data points well. How well, depends on how much accuracy is desired, and this is often captured in the form of an error goal or other convergence criteria, or a fit that is the result of a regression. Clearly, the approximation needs to be sufficiently accurate; that is, a certain level of accuracy will be necessary for subsequent studies to be successful and meaningful. However, it is not desirable to obtain perfect accuracy. Rather, the optimal degree of accuracy will be a function of the level of noise in the modeling process, which the metamodel is approximating.

Second, to be effective the metamodel representation must also model potential data points sufficiently, not just those that happen to have been chosen as a basis for the approximation. This is captured by the term *generalization properties*. Partially, it is a matter of collecting a representative data set. This aspect is closely related to the efficiency of the metamodel – the objective will be to choose those cases which minimize the number of cases to be analyzed but still yield a representative data set. This is where Designs of Experiments (DOEs)<sup>4</sup> have been found useful.

The third criterion, which would make a metamodel more effective, would be for it to remain a good approximation even if it were to be applied outside the bounds it was originally designed for. In practice, the need to extrapolate computational codes arises surprisingly often. Therefore, although it seems inherently unnecessary to explore extrapolation capabilities, it remains to be a thought to be aware of and kept in mind, in case the possibility presents itself. Naturally, the extrapolation properties in general will largely depend on the problem itself, and a general treatment may well be practically impossible.

## RESPONSE SURFACES AS METAMODELS

An method to create metamodels that has been applied and investigated is the Response Surface Methodology

(RSM).<sup>5</sup> It approximates the behavior of a response (i.e. a disciplinary or system level metric) with respect to certain design parameters through a specific function, a Response Surface Equation, which is usually a polynomial, often of second order. Certain cases for different variable settings according to a carefully chosen Design of Experiments table are created, and the estimated coefficients for the RSE are determined via an Analysis of Variance (ANOVA) approach. This method has been studied previously and shown limitations in some aspects, such as the number of variables. In addition, RSEs can yield an insufficient fit since the nature of the approximation (quadratic, cubic, etc.) has to be predefined.<sup>6</sup> A typical Response Surface equation would include linear, quadratic and interaction terms as shown below,

$$R = b_o + \sum_{i=1}^n b_i x_i + \sum_{i=1}^n b_{11} x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n b_{ij} x_i x_j \quad (1)$$

where  $R$  is the response of interest,  $x_i$  the inputs of inputs and,  $b_i$  and  $b_{ij}$  the related coefficients and  $n$  the number of variables.

## NEURAL NETWORKS AS METAMODELS

The use of neural networks is an alternative approach to the metamodeling process with RSEs and offers a number of benefits. A neural network is a computational system, implemented in terms of either software or hardware.<sup>7</sup> It is motivated by the way the brain functions. Some basic elements of neural networks in general and for feed-forward backpropagation NNNs in particular are presented in the following sections.

### Structure of Neural Networks

Each neural network generally consists of a large number of simple processing units, called *artificial neurons*, which are interconnected in a specific way. The strength of each of these connections is addressed as its *weight*, and these connection weights are adjusted during *learning*.<sup>8</sup>

The power of neural networks lies in their ability to combine logical parallel computations with serial operations.<sup>9</sup> The main characteristics needed to accurately describe a neural network are its *structure*, *dynamics* and *learning*, where the weights are obtained during learning.

Since many neural networks try to mimic the way a brain functions, the artificial neurons are usually simplified versions of a brain cell, known as a biological neuron. The basic structure of a simple artificial neuron is shown in Figure 2, where  $x_i$  are the network inputs,  $w_i$  the weights,  $y$  the weighted sum of inputs and input to the transfer function  $f$ , and  $a$  the network answer.

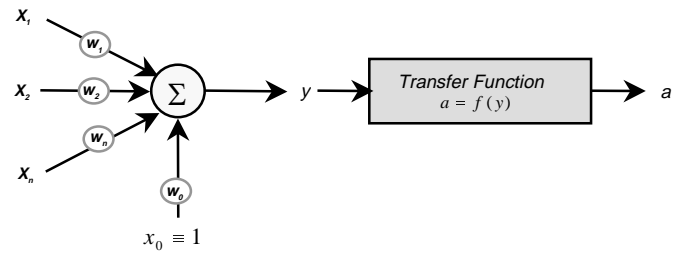


Figure 2: Depiction of an Artificial Neuron

Each neuron calculates the weighted sum of its inputs, and uses this sum as the input to its scalar transfer function:

$$\text{Vector of Inputs} \quad \bar{x} = \{1, x_1, x_2, \dots, x_n\} \quad (2)$$

$$\text{Vector of Weights} \quad \bar{w} = [w_0, w_1, w_2, \dots, w_n] \quad (3)$$

$$y = \sum_{i=0}^n x_i w_i = \bar{x} \cdot \bar{w} \quad (4)$$

In principle, any transfer function can be used. Currently, among the most popular ones for backpropagation neural networks are hard limit functions, linear and sigmoidal functions.<sup>10</sup>

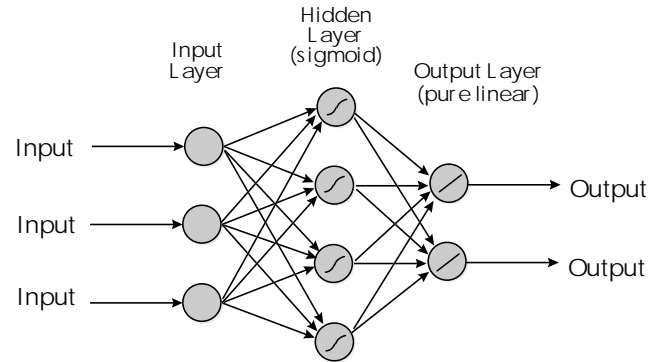


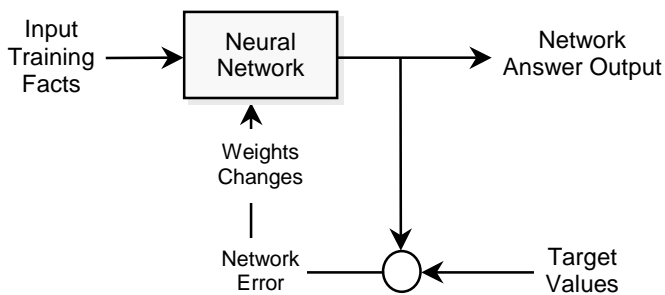
Figure 3: Feed-Forward Backpropagating NN

Several artificial neurons can be grouped together in layers or other structures, as in the feed-forward neural network shown in Figure 3. This resulting network can approximate any function arbitrarily well, that is, perfectly, as long as there are enough neurons in the hidden layer.<sup>10</sup> This type of neural net is usually trained via backpropagation. The way in which these neurons are organized and connected is called the *architecture* or *structure* of a neural network. It is defined through the transfer function used by its neurons, the number of scalar inputs, the weights to go with them, and the value of the biases. The bias can be thought of as the weight associated with an additional input that always takes the value 1. It essentially has the effect of an intercept.<sup>11</sup> The weights and bias values are adjusted as the network learns.

## Unsupervised and Supervised Learning

Two types of learning scenarios occur when neural networks are used. They can be distinguished by the type of learning that takes place. In both cases, the required changes in the weights are obtained through *learning rules*. These learning rules are fundamentally different for each scenario, and in many cases, there are several different learning rules available to choose from for a particular implementation.

The **supervised learning** scenario utilizes training input values for which the target output values are known. These target values are compared to the answer the network arrived at from the training input vectors, yielding the network error. This process is indicated in Figure 4.



**Figure 4: Supervised Learning**

This scenario reflects the case where a network is trained to perform a specific task, and once it is trained, it is actually used 'in the field'.

In this case, learning often occurs through **backpropagation**, which means propagating the network error back through the different layers of the network. The associated learning rule is often based on a gradient approach to minimize the network error, provided the transfer functions used are smooth and differentiable.

An example is the Widrow-Hoff learning rule<sup>12</sup>, where the weight changes are given by

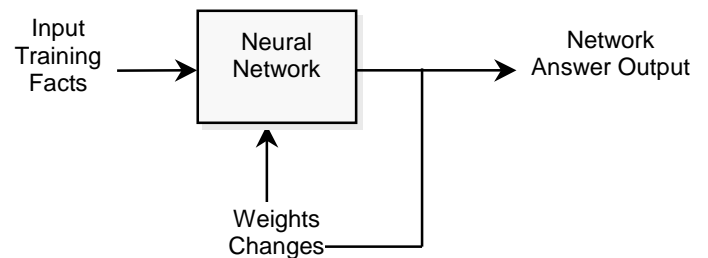
$$\Delta w_{ij} = \eta(t_i - w_i X) \cdot x_j \quad (5)$$

where  $\eta$  denotes the learning rate,  $t$  the target vector  $w$  the weights, and  $X$  the vector of inputs.

Essentially, using a gradient-based method to train the network is equivalent to finding the minimum on an error-surface. This is a hypersurface, with the weights and biases of the NN at the axes and the network error as a response.<sup>12</sup> A problem that comes with this approach, is that a gradient based method works locally, not globally, and can get trapped in a local minimum. Therefore, steps have to be taken to prevent this; either in form of

employing a globally working optimization scheme, like a genetic algorithm, or by working with multiple initializations, which statistically maximize the chance of finding the global maximum.<sup>10</sup> Some modifications to static learning rules that utilize pure backpropagation, are to use an adjustable learning rate, train with momentum, or the Levenberg-Marquardt learning rule,<sup>12</sup> which works with the Jacobean matrix.

In the case of an **unsupervised learning** scenario, there are no target values to compare against the network answers. The weights are adjusted depending solely on the value of the network answer. This is indicated in Figure 5.



**Figure 5: Unsupervised Learning**

In this scenario, the network never stops learning. Rather than having a dedicated training phase, the network learns as it performs its task. This reflects the situation of 'learn as you go'. The Hebbian learning rule<sup>12</sup> is an example of what a learning rule could look like for this scenario. Here, the network learns to produce similar output vectors to certain input vectors through association.

The weight changes are given in (6).

$$\Delta w_{ij} = \eta f(w_i \cdot X) x_j \quad (6)$$

Again,  $\eta$  denotes the learning rate,  $t$  the target vector,  $w$  the weights, and  $X$  the vector of inputs.

While for control tasks neural networks using unsupervised learning have been employed with great success<sup>9</sup>, for a function approximation application, a backpropagation neural network, which learns with supervision, is more appropriate and likely to yield better results.

In many cases, the best way to train a NN may be a hybrid approach, utilizing gradient approaches and other traditional methods as well as new and rather unconventional methods, like genetic algorithms. In general, neural networks can be combined to hybrid systems with almost any artificial intelligence method.<sup>9</sup>

The appropriateness of such an approach in this case will be subject to further research in the future.

### The Number of Neurons in the Hidden Layer

A very tricky question to answer is how many neurons need to be in the hidden layer. The number of neurons in the input and output layers are determined through the number of inputs and outputs, respectively, but the number of neurons in the hidden layer can vary.

It is important and desirable to find the number of neurons in the hidden layer that yields the best results in terms of both accuracy and generalization properties. In general, the number of hidden units is determined by the complexity of the problem. The more complex the problem is, the more neurons are needed for a good approximation.<sup>9</sup>

Unlike other approximation procedures, such as the Response Surface Methodology, with neural networks the desire to achieve both good accuracy and generalization properties becomes a trade-off. With more neurons in the hidden layer, the accuracy will increase, but the generalization properties may suffer enormously. With few neurons in the hidden layer, generalization properties are unlikely to cause problems, but the accuracy may not be reached.

**Underfitting** occurs when there are too few neurons in the hidden layer. Then the network is not able to capture the complexity of the approximated function in full. Therefore, the error goal will never be reached, no matter how much training takes place, and how many training values are provided. This is illustrated in Figure 6, where the '+' signs indicate training values.

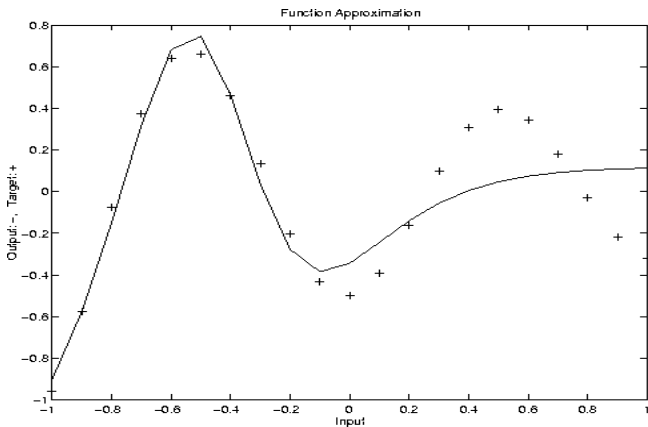


Figure 6: Underfitting<sup>11</sup>

**Overfitting** occurs when there are too many neurons in the hidden layer. In this case, the training phase runs smoothly, fast and without apparent problems; the desired fit on the training values (the error goal) is reached fast and easily.

However, the generalization properties can be extremely poor. Essentially, the large number of neurons implies a large number of adjustable weights. If the function to be approximated is limited in its complexity, then only a certain number of neurons in the hidden layer and their respective weights are needed to accurately describe the training values. However, if there are too many neurons in the hidden layer, then this may lead to redundancy within the network. Some of the weights might also be adjusted to cancel each other out for the particular set of training values that have been presented. The situation is comparable to a mechanical system with too many degrees of freedom.

In such a case, arbitrariness remains in the weights, undetected throughout the training phase. Consequently, the training values can be represented immaculately, but the generalization shows the arbitrariness and yields poor results. This is illustrated in and Figure 7.

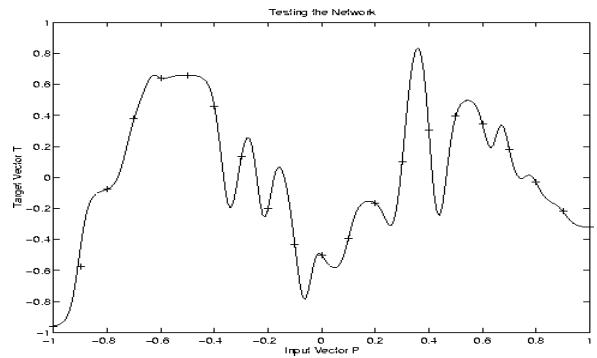


Figure 7: Overfitting; Generalization<sup>11</sup>

### SELECTING THE TYPE OF NEURAL NETWORK

The type of neural network best suited to approximate the relation between the desired response and the selected design parameters, and thus for use as a metamodel in a function approximation implementation, is a feed-forward neural network. Specifically, a two layer feed-forward NN, as shown in Figure 3, is selected for this purpose. It has a tangent-sigmoidal transfer function in the first and hidden layer and a pure linear transfer function in the output layer. Both input and output values are scaled appropriately.

The pure linear function in the output layer has the purpose of scaling the values to the desired outputs, where a sigmoidal function in the hidden layer introduces the non-linearity required to approximate a non-linear function. Specifically, a tangent-sigmoid function is selected to be the transfer function in the hidden layer, since this function maps the real line to values between -1 and 1, rather than between 0 and 1, thus including negative values. The pure linear function in the next layer then scales the values appropriately. This way, any numerical value on the real line can be achieved.<sup>11</sup>

## DETAILS OF IMPLEMENTING A NEURAL NETWORK AS A METAMODEL IN DESIGN

Several considerations are necessary to successfully implement a neural network approach in the design process. Through researching these questions in search of the optimal methods, a sound and successful approach was developed, and indicated in this section.

### METHODS TO INITIALIZE AND TRAIN THE NEURAL NETWORK

A good environment in which to perform neural network related computations is MATLAB with the neural network toolbox<sup>12</sup>, which was used extensively for the research in this paper. The training and target vectors are derived from the collected design data, and the network is initialized according to the number of parameters and their ranges.

In general, the training of feed-forward neural networks is established first by initializing the weights and then training via a backpropagation method. Specifically, the weights of a neural network were initialized using the initialization function available in the MATLAB toolbox, and then trained using the Levenberg-Marquardt routine. This method works with the Jacobean matrix and typically is significantly faster than a pure backpropagation method based solely on the gradient.<sup>12</sup>

The drawback of this approach is that there is no way of knowing whether the inherent optimization located a local minimum on the error-surface or the global minimum was found.

The solution is to start from several different sets of weights, that is, initialize the network several times, resulting in multiple networks. Then each NN can be trained for several iterations (referred to as epochs) and those that yield the lowest error and exhibit a similar pattern in the weights are selected for further training if it is needed.

Depending on the task, the number of NNs that should be created, and how long each of them ought to be trained before one set of weights can be selected, varies greatly. For few neurons in the hidden layer, only few initializations have to take place, but as the number of hidden units increases, this should be extended. Since there is not too much computational time associated with initialization and training, in general the approach is: the more, the better.

The necessary number of initializations for any specific application task should be determined primary to the actual NN selection process in a brief preliminary study. With the right objective function and using the full Levenberg-Marquardt algorithm, only few iterations will typically be needed.

## METHODS TO FIND THE OPTIMAL NUMBER OF NEURONS IN THE HIDDEN LAYER – AVOIDING OVERFITTING AND UNDERFITTING

The objective behind finding the optimal number of neurons in the hidden layer is to find the network that yields the most accurate representation of the underlying training data while still producing a reasonable generalization.

The generalization properties of a feed-forward NN can be insufficient for two reasons. On one hand, the training data may not be representative. This can also be a particular problem with the RSM. Any approximation based on a non-representative set of training values will automatically yield poor generalization properties. On the other hand, the data may be representative, but the neural network may still produce poor generalization due to too many neurons in the hidden layer, as has been discussed before.

Therefore, the first aspect to ensure is that the training data cases were selected such that they form a representative set. This implies that the designing engineer is familiar with the general behavior and degree of complexity of the problem under investigation, and that he or she is able to determine what would be a representative set. An approach that has proven effective in selecting cases to form a representative data set while minimizing the actual number of cases, is the Design of Experiments approach. In order to concentrate on evaluating the specifics of the NN in question, it will be assumed that the designer has obtained a representative set of cases, for example via the DOE approach. Further, the assumption is made that this representative set of cases is available as training data to train the neural network.

For too few neurons in the hidden layer, the generalization is not a problem; it remains consistent with the approximation function character. Actually meeting the required error goal poses the problem.

Assuming for the moment that there are too few neurons in the hidden layer, for every number of neurons there is a certain best error goal that can be achieved with sufficient training. As the number of neurons increases, the final error of the NN decreases. A good approach is to start with a NN that has a low number of neurons in the hidden layer and to gradually increase the number of hidden units. At some point, a NN will be achieved that has the lowest possible number of neurons in the hidden layer while reaching the desired error goal. Each NN has to be trained as far as possible, that is, until it actually reaches a minimum in the error surface.

Whenever a NN is initialized and trained, precautions have to be made to minimize the chance of it getting caught in any local minima. The question that remains is how to actually add the neurons in the hidden layer. Should the weights of the previously trained NN be kept

and only extended to reflect an additional neuron? Alternatively, should the new NN be regarded as a completely new solution approach, and initialized and trained new from the beginning?

This leads to two fundamentally different approaches. One means keeping the numerical values for the weights obtained during previous training, and just adding appropriate weights to indicate an additional neuron in the hidden layer. Since the learning rules yield weight changes of zero or near zero for weights that are zero or near zero, respectively, there would be no or very slow learning where the weights are very small. Therefore, the numerical values for the weights of the new neuron would be small, so as to not increase the error too much and keep the intrinsic knowledge already acquired, but should not be zero or a very small number. The other approach is to treat each NN with an additional neuron like a completely new network, and start by initializing it again, several times as has been discussed previously to ensure the global minimum is found.

Both approaches have their advantages and disadvantages. The first will be more efficient, provided it reaches the optimum. The latter however, seems safer in that it is not dependent on previous results. Both approaches have been applied to the example problem, and from a comparison of their performance, it has become clear that initializing the network totally new yields the better results.

#### METHODS OF IMPROVING THE GENERALIZATION PROPERTIES

There are several methods and philosophies on how to improve the generalization properties of a neural network where overfitting might be involved. Two such approaches which are implemented in the MATLAB neural networks toolbox<sup>12</sup> include 'early stopping' and 'regularization'.

The approach of early stopping implies that in addition to the training data that has carefully been collected, two additional data sets are available, for validation and test purposes. It builds on the fact that as a NN is trained, initially both the errors on the training data set as well as on the validation data set decrease. When overfitting starts to occur, the error on the training data set will continue to drop, while the error on the validation increases. With early stopping<sup>12</sup>, the error on the validation data set is monitored throughout the training phase, and when it starts to increase, the training is terminated. This is repeated several times, and then the NN with the lowest error on the remaining test data is selected as the best representation.

'Regularization' aims for a smooth and regular function rather than one with many edges and peaks. It employs the idea, that the NNs that yields the most regular and smoothest approximations are those where the weights are numerically small. Thus, for this approach, the

objective function, which is used by the optimization and learning rules, is modified to aim for a low mean squared error as well as small values for the weights.<sup>12</sup>

#### THE SELECTION PROCESS

Ultimately, the process to achieve the most appropriate neural network with the best representation capabilities involves all aspects of initializing and training neural networks and finally selecting those, which yield the lowest error.

It is recommended to create validation and test data with random parameter settings in addition to the experimental design data. This increases confidence in the representativeness of the DOE data and thus in the metamodel itself. Further, if validation and test data exists, it should be utilized during the selection process.

When considering error, or the fit to data, the maximum relative error should be taken into account in addition to the mean square error (MSE). Thus, an overall evaluation criterion (OEC) to use in evaluating metamodels could be a weighted sum of the maximum relative error and the MSE of all available data sets combined. The weighting needs to be adjusted from problem to problem as the absolute magnitude of the MSE changes.

Alternatively, in cases where there is no validation and test data available, the regularized MSE, which is the performance and objective function in the above mentioned regularization approach, serves well as a selection criterion.

#### RESPONSE SURFACES AND NEURAL NETWORKS IN FUNCTION APPROXIMATION

Some helpful conclusions can be drawn from the way Response Surfaces and Neural Networks approximate certain functions.

Response surfaces are typically represented with polynomial equations and are comparable to Taylor series approximations. As such, they can approximate any sufficiently differentiable function well within a reasonably small neighborhood of a selected point. However, the response surface equations exhibit rather poor approximation qualities in cases with many variables and large ranges. This can be explained by taking a mathematical look at Taylor series.

#### MATHEMATICAL ASPECTS

The well known representation of a Taylor Series of a sufficiently differentiable function  $f(x)$  in one variable about the point  $x_0$  is given as

$$f(x) = P_n(x) + R_{n+1}(x) \quad (7)$$

with the polynomial portion of degree  $n$  to be

$$\begin{aligned}
 p_n(x) &= f(x_0) + \frac{(x-x_0)}{1!} f'(x_0) \\
 &\quad + \dots + \frac{(x-x_0)^n}{n!} f^{(n)}(x_0) \\
 &= f(x_0) + \sum_{j=1}^n \frac{(x-x_0)^j}{j!} f^{(j)}(x_0)
 \end{aligned} \tag{8}$$

and the remainder  $R_{n+1}$

$$\begin{aligned}
 R_{n+1}(x) &= \frac{1}{n!} \int_{x_0}^x (x-t)^n f^{(n+1)}(t) dt \\
 &= \frac{(x-x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi)
 \end{aligned} \tag{9}$$

for some  $\xi$  between  $x_0$  and  $x$ .<sup>6</sup>

In this case, a polynomial  $p_2(x)$  describes a response surface equation of degree 2 and the remainder  $R_3(x)$  the error associated with the response surface approximation.

For a Taylor series approximation of a function  $f(x,y)$  in two variables, about a point  $(x_0, y_0)$  the expressions become more involved:

$$\begin{aligned}
 f(x_0 + \xi, y_0 + \eta) &= p_n(x_0 + \xi, y_0 + \eta) \\
 &\quad + R_{n+1}(x_0 + \xi, y_0 + \eta)
 \end{aligned} \tag{10}$$

with

$$\begin{aligned}
 p_n(x_0 + \xi, y_0 + \eta) &= f(x_0, y_0) \\
 &\quad + \sum_{j=1}^n \frac{1}{j!} \left[ \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^j f^{(j)}(x, y) \Big|_{\substack{x=x_0 \\ y=y_0}}
 \end{aligned} \tag{11}$$

and the remainder term

$$\begin{aligned}
 R_{n+1}(x_0 + \xi, y_0 + \eta) & \\
 &= \frac{1}{(n+1)!} \left[ \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} \right]^{n+1} f^{(n+1)}(x, y) \Big|_{\substack{x=x_0+\theta\xi \\ y=y_0+\theta\eta}}
 \end{aligned} \tag{12}$$

for some  $\theta$  between 0 and 1.<sup>6</sup>

Generalizing this term to  $r$  variables leads to the following expression for the remainder

$$\begin{aligned}
 R_{n+1}(x_{1,0} + \xi_1, \dots, x_{r,0} + \xi_r) & \\
 &= R_{n+1}(\bar{x}_0 + \bar{\xi}) \\
 &= \frac{1}{(n+1)!} \left[ \sum_{j=1}^r \xi_j \frac{\partial}{\partial x_j} \right]^{n+1} f^{(n+1)}(\bar{x}) \Big|_{\bar{x}=\bar{x}_0+\theta\bar{\xi}}
 \end{aligned} \tag{13}$$

Since the term in (13) contains a sum expression, which is raised to a power, it becomes more complex and is likely to increase numerically as the number of variables  $r$  increases. This remainder term is associated with the error, thus the smaller it is, the more accurate the approximation will be.

In most cases where few variables are involved, a quadratic function sufficiently approximates the unknown underlying function. However, as the number of variables increases, so will the error term (13), and eventually the desired accuracy cannot be achieved any more. Therefore, when dealing with a large number of variables, it might be necessary to go beyond the second derivative in the Taylor series approximation to ensure this same accuracy. This is equivalent to the implementation of an approximating function of degree larger than two, which is why a quadratic RSE may perform poorly for large numbers of variables.

Similarly, the expression in (13) is dependent on the value of the  $\xi_i$ , which are related to the variable ranges. Thus, the larger the ranges are, the more derivatives have to be taken into account to achieve a high accuracy and the greater the degree of an approximating polynomial has to be.

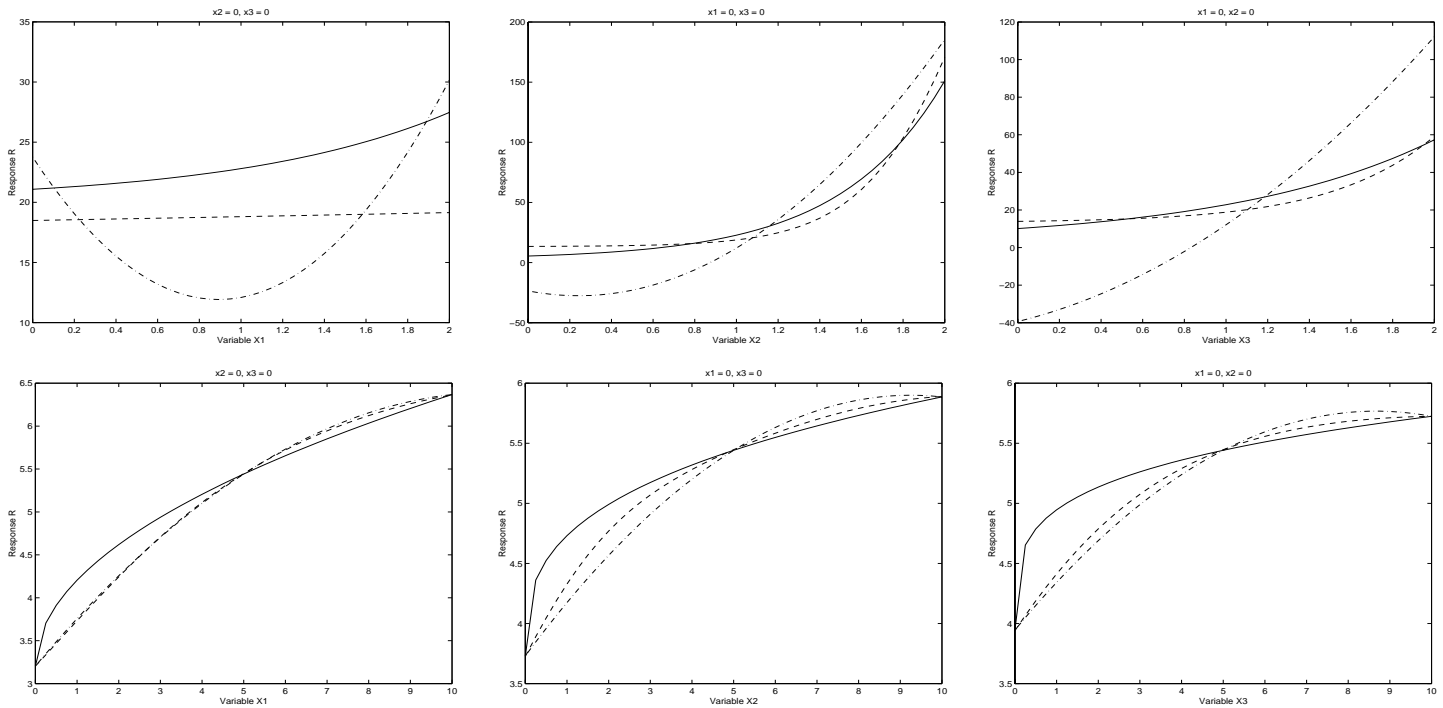
## ILLUSTRATIVE EXAMPLES

Two function approximations, both dependent on three variables were selected to illustrate these features. They are intended to examine relationships which could realistically occur in a real world design situation and are just complicated enough to exhibit the limitation of either the RSE or NNs. Thus, several sets of ranges for the three variables in both functions were evaluated and then selected such that the function exhibits an interesting, mildly nonlinear behavior, without getting unrealistically complicated.

The first function,

$$f_1(\bar{x}) = e^{x_1} + e^{(2x_2+x_3)} \tag{14}$$

is a relation that could be found in aircraft design. The values for each variable range from zero to two.



**Figure 8: Plots of Function 1 and Function 2 at Midpoints**

This results in a noticeably nonlinear relation.

The second function includes root terms, which can also be found in engineering problems:

$$f_2(\bar{x}) = x_1^{\frac{1}{2}} + x_2^{\frac{1}{3}} + x_3^{\frac{1}{4}} \quad (15)$$

For this second equation, the values range from zero to ten. This function shows a steep, significantly nonlinear portion for low values of  $x_i$ , and then a certain 'smoothing out' towards greater values.

For both equations and the above ranges, a DOE<sup>4</sup> approach is employed and a Central Composite Design<sup>13</sup> (CCD) created, with a single center point, yielding 15 cases. In addition, 30 random cases are created for later validation and testing purposes.

With these data points, for each equation, an RSE is fit using the DOE cases and several feed-forward Neural Networks are initialized. The training of the NNs occurs via the principal Levenberg-Marquardt backpropagation with regularization described earlier to desensitize the problem of overfitting. In addition, the number of neurons in the hidden layer is varied from 1 to 10 hidden units for the first equation and from 1 to 15 for the second. For each number of neurons, several NNs are initialized, and the two with the lowest OEC on the entire data, and with the lowest regularized MSE on the DOE data are selected. This represents the cases where testing and validation data is or is not available, respectively.

For both functions, the RSM and NNs yield different approximations, and the NNs selected with the OEC are

slightly different from those chosen through the regularized MSE.

The NNs selected with regard to the OEC show a constant improvement in performance as the number of neurons in the hidden layer increases. In contrast, the performance of the NN selected by the lowest regularized MSE first improves, reaches an optimum, and then declines with further increase in the number of hidden, indication an overfitting on the DOE data.

In general, the NNs outperformed the RSE approximations. In Figure 8, the graphs of both functions are shown as solid lines, along with the RSE approximation as a dash-dot, and a NN as a dashed line.

The NN plotted for the first function was selected with respect to the OEC but has only one neuron in the hidden layer. It approximates the function better than the RSE, especially in the  $x_2$  and  $x_3$  dimensions. Most likely, the reason for this surprisingly good approximation with just one hidden unit is the similarity of the function to be approximated with the tangent-sigmoidal transfer function the NN uses. In contrast, the RSE is unable to predict the desired function sufficiently.

For the second function the shown NN was selected with respect to the regularized MSE and has seven neurons in the hidden layer. Thus, both the NN and the RSE have been created using only the DOE data, which in both cases is fit nearly perfectly, with a MSE and maximum relative error of less the  $10^{-4}$ . However, the two approximations are very different from each other, and from the original function. This illustrates, how a perfect

fit can be deceptive, if the data set is not representative of the underlying function and thus statistically not valid.

### AN EXAMPLE IMPLEMENTATION: ADDRESSING STABILITY & CONTROL ISSUES IN DESIGN

A field that is often left out in the conceptual design stages of an airplane design is the stability and controls discipline. In order to avoid costly time delay there is a desire to address these issues as early as possible, especially when working with unconventional configurations. The aircraft chosen for this study is a High Speed Civil Transport (HSCT), carrying 300 passengers and with a range of 5000 nautical miles and a cruise speed of Mach 2.4. Its unconventional nature makes it a prime example of an aircraft design where the above mentioned paradigm shift is extremely desirable.

The task of evaluating the static longitudinal stability as part of the stability and control assessment is selected to examine the implementation and performance of various metamodel-building approaches. A suitable response for the static longitudinal stability is the stability derivative  $C_{m\alpha}$ , which indicates the pitching moment of an aircraft due to the angle of attack.

The most critical phase for the static stability is often the take-off phase, where the angle of attack is extremely high for a supersonic transport. The flight condition at which the aircraft is analyzed is selected accordingly to be at Mach 0.3 and at 10 degrees angle of attack.

The static margin, the distance between the aerodynamic center and the center of gravity of an aircraft is closely related to the value of  $C_{m\alpha}$ .

Therefore, the design aspects that are most likely to affect this derivative are any parameters associated with the location of these points. Specifically, the wing planform and horizontal tail parameters are those that indicate the positions of wing and horizontal tail in longitudinal direction (commonly denoted as x-direction). To visualize this influence, common parameters, such as sweep angle and reference areas were preferred over grid points to describe the planforms. The selected design variables are depicted in Figure 10 and listed, along with their baseline values in Table 1. The results were produced using the conceptual/preliminary design level subsonic aerodynamic prediction code HASC 95 (High Angle of attack Stability and Control).<sup>14</sup>

### VARIATION OF THE RANGES

An important aspect to keep in mind when metamodels are used is that the ranges in which the design parameters vary may have a significant impact on the approximation. They might influence the type of metamodel, which is selected, the analysis approach, and the design of experiments based on which the data are collected. The underlying cause is the argument that if the ranges are chosen larger, the response may follow a more complicated relation than it would if the ranges were narrow. Therefore, a previously chosen approximation method may not be suitable anymore, once the ranges are increased.

To investigate the influences of varying the ranges, a design of experiments was set up and conducted for different ranges. The ranges for the thirteen parameters were varied from +/- 2% to +/- 10% of their baseline value, with the exception of the wing and horizontal tail x-positions. The ranges for these two variables were varied from +/- 2 ft to +/- 10 ft.

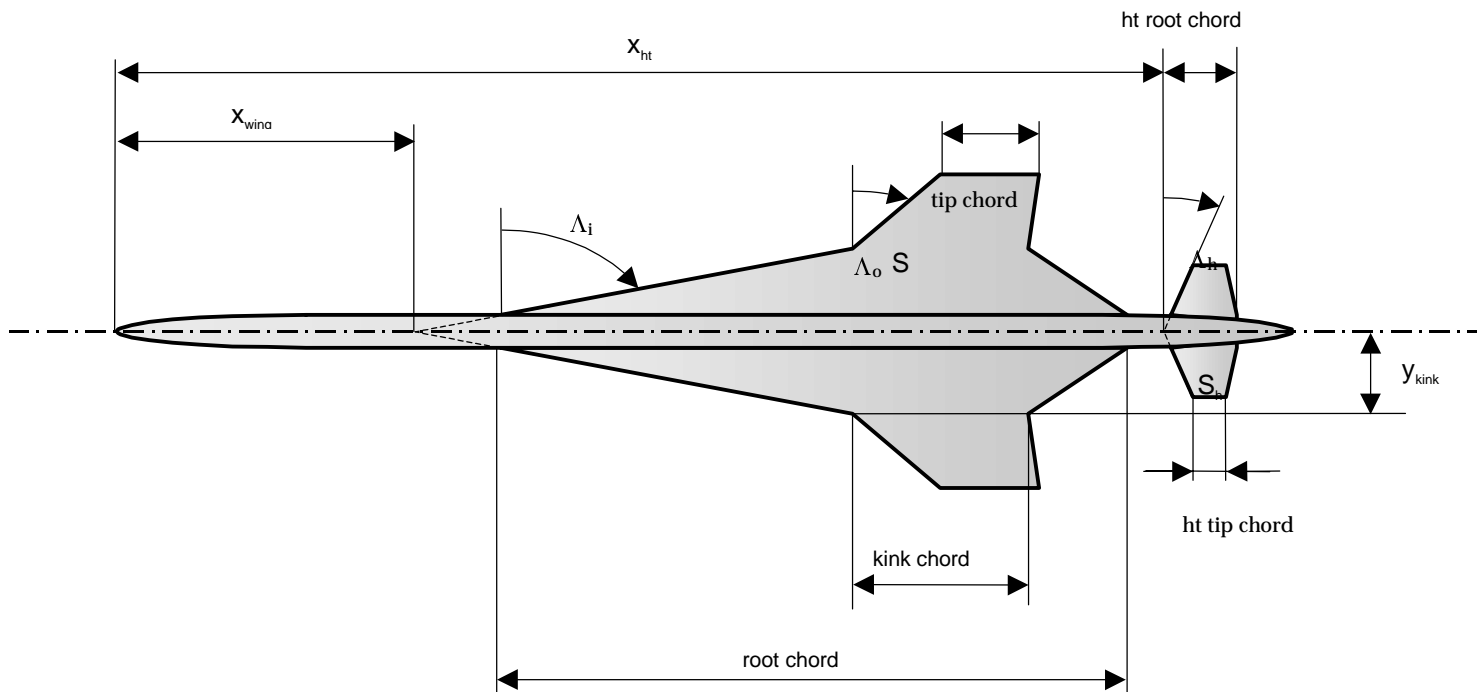


Figure 9: HSCT Planform

**Table 1: Baseline Values and Ranges**

Variables	Baseline Value	Ranges
S (wing reference area)	9000 sq. ft	+/- 2% to +/-10%
$X_{wing}$ (wing x-position)	66.5 ft	+/- 2 ft to +/- 10 ft
$\Lambda_i$ (inner sweep)	71.5 degrees	+/- 2% to +/-10%
$\Lambda_o$ (outer sweep)	51.5 degeered	+/- 2% to +/-10%
Tip chord (normalized by semi-span)	0.24	+/- 2% to +/-10%
Kink chord (normalized by semi-span)	0.60	+/- 2% to +/-10%
Root chord (normalized by semi-span)	2.43	+/- 2% to +/-10%
$Y_{kink}$ (normalized by semi-span)	0.55	+/- 2% to +/-10%
$S_h$ (HT reference area)	220 sq. ft	+/- 2% to +/-10%
$X_{HT}$ (HT x-position)	267 ft	+/- 2 ft to +/- 10 ft
$\Lambda_{HT}$ (HT sweep angle)	28	+/- 2% to +/-10%
HT tip chord (normalized by semi-span)	0.16	+/- 2% to +/-10%
HT root chord (normalized by semi-span)	0.68	+/- 2% to +/-10%

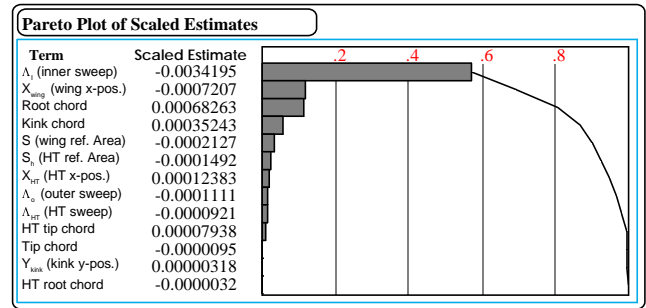
In each of the cases, the inner sweep angle was by far the most predominant contributor, and in comparison, other factors are seemingly insignificant. To minimize the chance that the selection of ranges was skewing these results, the range for the inner sweep angle was reduced by half to range from +/- 1% to +/- 5%. This did not alter the results significantly. The Pareto plot in Figure 10 shows the importance ratings of the predictors for a representative set of relatively high ranges. Here all variables ranged from -10% to +10, or within +/- 8 ft of their baseline value, the inner sweep angle ranges from -5% to + 5% of the baseline value. This set of ranges was used to conduct the following studies.

**BUILDING THE RSE AND NN METAMODELS**

Both the RSM and the NN approach require data for the regression and training, respectively. There is no fundamental difference between these data sets. They both need to be representative and thus statistically valid sets. In order to achieve maximum efficiency, a DOE is employed, and the results are used as data for both methods. This has the added benefit that the performances of the NN and RSM strategies are directly comparable.

A Central Composite (CC) Fractional Factorial Design was implemented for all 13 variables. It requires 283 cases, which were used to fit the regression equation in the RSM approach and as training values in the NN

approach. In addition, 500 test cases were run with arbitrary setting for the 13 variables within the given ranges. These are used to validate the approaches and test the generalization properties of both methods.



**Figure 10: Pareto Plot**

**RESULTS**

The regression to fit a quadratic RSE with interactions, as given in Equation 1, yields a high R<sup>2</sup> value and thus seems to expose a very good fit.

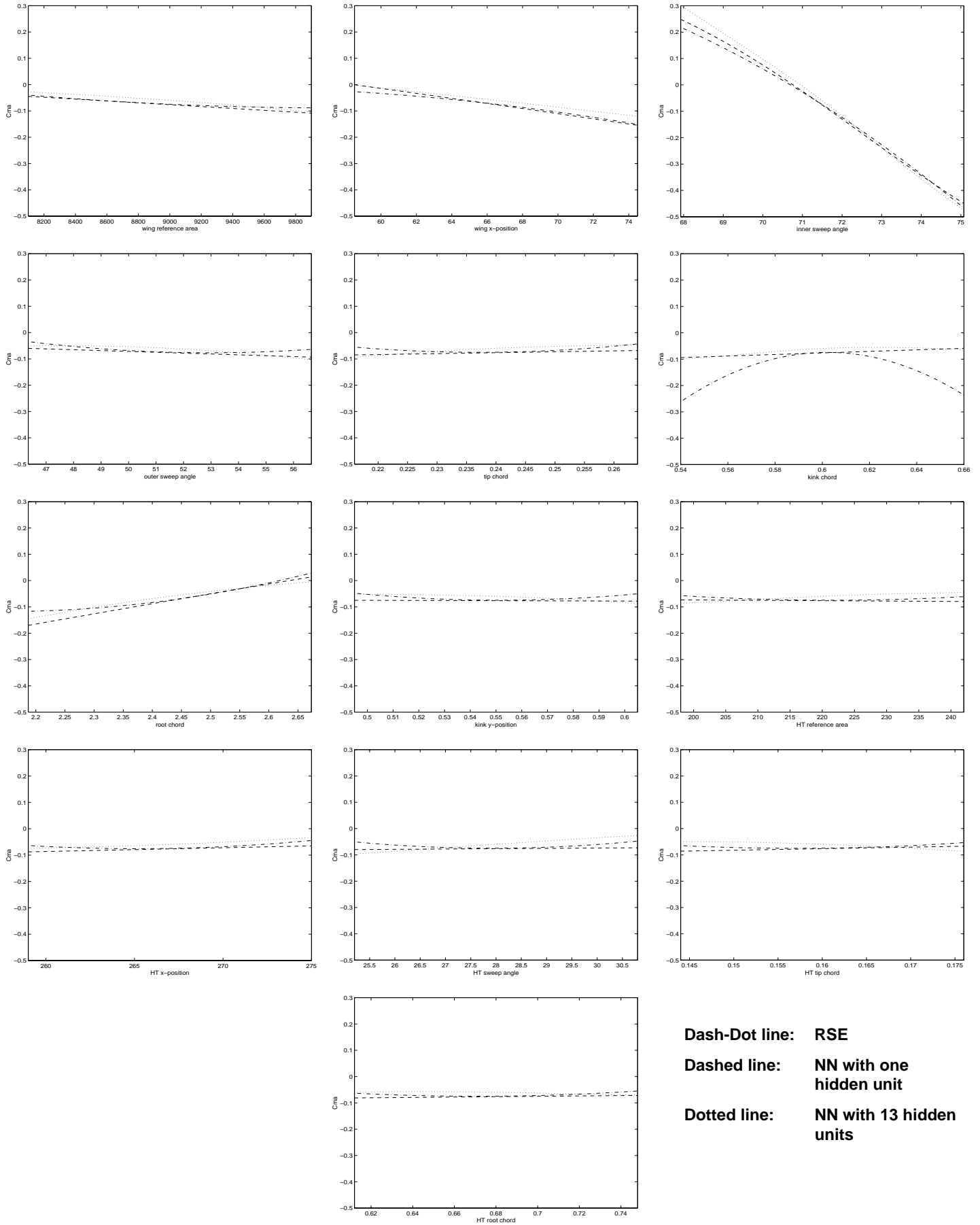
Although the average fit to the DOE data, thus the MSE shows good results, the maximum relative error for the RSE is 20.14% for the DOE data and 34.13% for the total data set. This shows how even a good fit with average generalization properties can yield insufficient accuracy, and thus another example why a designer should not solely rely on an R<sup>2</sup> or MSE alone.

An OEC with approximately equal weighting of the MSE and maximum relative error of the total data is about 0.47 for the RSE. All NNs outperform the RSE in terms of MSE for design and validation data, and the OEC.

The NNs that are selected with regard to the OEC perform better than those selected based on either the MSE or the maximum relative error for the entire data. Similarly, a network selected with respect to the regularized MSE on the design data performs better than one selected either on the MSE or the maximum relative error on the design data.

A NN with one neuron yields an approximation with OEC 0.44, which already is a reduction by 94% compared to the RSE performance. The more neurons are added, the better the approximation gets. A NN with 13 neurons in the hidden layer yields an OEC of 0.34, which is a reduction of 72% compared to the RSE. The maximum relative error on the entire data drops under 30 % for a NN with at least 15 neurons.

The prediction profiles for  $C_{m\alpha}$  are depicted in Figure 11.



**Dash-Dot line:** RSE  
**Dashed line:** NN with one hidden unit  
**Dotted line:** NN with 13 hidden units

**Figure 11: Prediction Profiles for  $C_{m\alpha}$**

## SUMMARY AND CONCLUSION

The use of metamodels in design brings useful benefits. They are mostly employed as approximations to unknown and potentially complicated functions. It has been shown that NN can be used in place of RSEs to fulfill this purpose.

The NN probably best suited for such a task and under investigation here, is a two layer feed-forward neural network, with a tangent-sigmoid transfer function in the hidden layer and a pure linear one in the output layer.

To train a neural network as a metamodel, it should be initialized repeatedly and trained with a method that enables improved generalization properties. Then the set of weights that has the lowest error of those trained with enhanced generalization should be selected for further training. These methods to enhance the generalization properties desensitize the problem of finding the exact optimal number of neurons in the hidden layer. The recommended procedure is regularization or a similar method.

If an additional neuron is needed in the hidden layer, it is advisable to create a completely new network and start the process from the beginning, rather than trying to keep any implicit knowledge. The hypersurfaces that represent these NNs have been found to change drastically with any added dimension, such that knowledge cannot be transferred trivially, as has been assumed previously.

A comparative assessment of the performances of both an RSE and NN with several numbers of neurons in the hidden layer was performed. It established that for a problem with few variables and small ranges, such that, it is linear, near linear, or quadratic in nature, the RSM performs well based on a central composite design. However, it is also evident, that this assumption is rather restrictive, and validation and testing should always take place, to verify it.

In addition, a good fit on the design data can be deceptive, and may still result in insufficient generalization properties, if the data is not a representative set, or if the maximum relative error exceeds the desired value. To create a representative validation data set, a large number of random cases is recommended. Going beyond a CCD design to a full factorial or similar may likely be of little value considering the increase in computational effort. Random testing data sets tend to avoid systematical errors.

Thus, if the nature of a problem is unknown, or it involves a larger number of variables or extended ranges, the RSM is likely to fail and not yield a sufficient fit and/or generalization properties. In such cases, neural networks are likely to yield the better approximations, since there are fewer assumptions involved, provided they are initialized, trained and selected appropriately. These specifics were investigated and outlined.

## ACKNOWLEDGEMENTS

The authors would like to thank Ms. Katherine Drew and Mr. Sam Dollyhigh for making this research possible. It was sponsored under the following grants: Office of Naval Research grant N00014-97-1-0783, National Science Foundation grant DMI-9734234, and NASA Langley grant NAG-1-1793.

## REFERENCES

1. Mavris, D.N., DeLaurentis, D.A., Bandte, O., Hale, M.A., "A Stochastic Approach to Multi-disciplinary Aircraft Analysis and Design", AIAA 98-0912
2. Friedman, L.W., *The Simulation Metamodel*, Kulwer Academic Publishers, 1996
3. Chen, W., "A Robust Concept Exploration Method for Configuring Complex Systems", Doctoral Dissertation, Georgia Institute of Technology, School of Mechanical Engineering, 1995
4. Box, G.E.P., Draper, N.R., *Empirical Models Building and Response Surfaces*, John Wiley & Sons, Inc., 1987
5. DeLaurentis, D.A., Mavris, D.N., Schrage, D.P., "Systems Synthesis in Preliminary Aircraft Design Using Statistical Methods", 20<sup>th</sup> ICAS conference, Sorrento, Italy, 1996
6. Atkinson, K.E., *An Introduction to Numerical Analysis*, John Wiley & Sons, Inc., 1989
7. Bishop, C.M., *Neural Networks for Pattern Recognition*, Oxford University Press, 1995
8. Ripley, B.D., *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996
9. Maren, A.J., Harston, C.T., Pap, R.M., *Handbook of Neural Computing Applications*, Academic Press, Inc., 1990
10. Chauvin, Y., Rumelhart, D.E., (Eds.), *Backpropagation: Theory, Architectures, and Applications*, Lawrence Erlbaum Associates, Inc., 1995
11. *Neural Networks Toolbox User's Guide*, 2<sup>nd</sup> Ed., The MathWorks, Inc., 1994
12. *Neural Networks Toolbox User's Guide*, 3<sup>rd</sup> Ed., The MathWorks, Inc., 1998
13. Montgomery, D.C., *Design and Analysis of Experiments*, 3<sup>rd</sup> Ed., John Wiley & Sons, Inc., 1991
14. Albright, A.E., Dixon, C.J., Hegedus, M.C., "Modification and Validation of Conceptual Design Aerodynamic Prediction Method HASC 95 With VTXCHN," NASA Contractor Report 4712, NASA Langley Research Center, 1996