5-1-2020

Airfoil analysis and design using surrogate models

Nicholas Alexander Michael

Airfoil analysis and design using surrogate models

By

Nicholas A. Michael

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Aerospace Engineering
in the Department of Aerospace Engineering

Mississippi State, Mississippi

May 2020

Airfoil analysis and design using surrogate models

By

Nicholas A. Michael

Approved:

Richard A. Weed
(Major Professor)

J. Mark Janus
(Committee Member)

David S. Thompson
(Committee Member/Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Nicholas A. Michael

Date of Degree: May 1, 2020

Institution: Mississippi State University

Major Field: Aerospace Engineering

Major Professor: Richard A. Weed

Title of Study: Airfoil analysis and design using surrogate models

Pages of Study: 156

Candidate for Degree of Master of Science

A study was performed to compare two different methods for generating surrogate models for the analysis and design of airfoils. Initial research was performed to compare the accuracy of surrogate models for predicting the lift and drag of an airfoil with data collected from high-fidelity simulations using a modern CFD code along with lower-order models using a panel code. This was followed by an evaluation of the Class Shape Transformation (CST) method for parameterizing airfoil geometries as a prelude to the use of surrogate models for airfoil design optimization and the implementation of software to use CST to modify airfoil shapes as part of the airfoil design process. Optimization routines were coupled with surrogate modeling techniques to study the accuracy and efficiency of the surrogate models to produce optimal airfoil shapes. Finally, the results of the current research are summarized, and suggestions are made for future research.

DEDICATION

I would like to dedicate this to my family, Kristen, and all those who have helped me reach this point along the way.

ACKNOWLEDGEMENTS

This material is based upon work supported by the ERDC under Contract No. W912HZ-17-C-0020, Surrogate Modeling and Computational Steering. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Army Engineer Research and Development Center (ERDC).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The design of modern aircraft is an extremely expensive process whose total cost can reach into the billions of dollars. Traditionally, aircraft design was based on a combination of wind-tunnel testing supported by the most accurate analysis methods available. A typical design process might evaluate hundreds or even thousands of potential designs before settling on a few candidate designs. The candidate designs are then refined to produce an eventual ”best” design. Recently, modern optimization methods have been coupled with a variety of analysis methods to reduce the number of potential designs that must be evaluated. Although recent advances in Computation Fluid Dynamics (CFD) methods that can predict fully viscous turbulent flows at steady-state has led to a reduction in wind-tunnel testing, these methods are still prohibitively expensive to use for the entire design process. This has led researchers to pursue less costly approaches based on the development of fast running surrogate models to reduce the number of expensive tests and high-fidelity CFD analyses required to generate the ultimate design. Therefore, over the last few years, a large amount of research has been devoted to the use of surrogate models in aircraft analysis and design optimization.

At their core, surrogate models are basically advanced methods for doing multi-dimensional interpolation. All surrogate models require a training set of data based on experimental data, analytical results, or a combination of the two. The basic idea behind surrogate models is that with a sufficient set of training data and advanced methods for generating approximations to the data sets, a sufficiently accurate model can be generated that can replace both costly and time consuming experiments and analysis. The goal is to generate a surrogate model with minimum cost but with sufficient accuracy for preliminary analysis and design. This will allow the use of more expensive analysis and design methods to be delayed until much later in the design process.

There are multiple approaches to using surrogate models for the purpose of engineering design. A common approach is to generate an initial set of sampling points, perform evaluations for the sampling points, and train some sort of predictive model using these points. Once an initial surrogate model has been generated, infill techniques can be used to add points to locations where there is a higher likelihood for variations in the model. Evaluations are then made at these points, the model is retrained, and the process repeats.

The two methods that will be discussed in this paper will be Sparse Grids and Kriging. Kriging is a method that uses co-variances and means to build a predictive model. The point locations are arbitrary, as are locations for infill. Sparse Grid sampling is more rigid in point selection but more flexible with the evaluation method. Sparse Grids can use a variety of different basis functions to make predictions including but not limited to B-splines, Chebyshev polynomials, and Gaussian quadratures. However, due to the hierarchical way

Sparse Grids perform training, the locations of where infill can occur on each iteration of refinement are limited.

Work presented in the following chapters looks to analyze the effectiveness of two different surrogate modeling techniques for the tasks of airfoil analysis and design. The primary objective is to be able to improve upon existing airfoil designs using a variety of optimization procedures. Other areas of focus look to compare surrogate modeling techniques and reduce the number of high fidelity simulations needed to obtain accurate lift and drag predictions. Another objective is to determine whether or not higher-order CFD methods are actually needed to obtain lift and drag data for sampling points in the surrogate models, or if panel methods with viscous corrections are accurate enough. To quantify the process of airfoil parameterization, multiple techniques, including Bézier functions, B-splines, and CST are evaluated for their ability to produce reasonably shaped airfoils given a set of parameters. Furthermore, modifications are made to the CST method so that only small perturbations of the shape function are necessary to make reasonable changes from a base airfoil shape. Finally, analysis of procedures and data is performed to determine a best course of action for building models in the future.

A review of literature and past work by other authors is given in Chapter 2. The process of generating a surrogate model is described in Chapter 3. Chapter 4 discusses various airfoil parameterization techniques. An initial study to determine the viability of Sparse Grid surrogate modeling techniques for airfoil analysis and the effectiveness of lower order panel methods for point evaluation in comparison with CFD are described in Chapter 5. Chapter 6 describes the two optimization procedures used in this research and compares

their efficiency. Chapter 7 describes a set of procedures used to develop a real design for airfoil optimization using surrogate models. Results from tests comparing efficiency in different implementations of Sparse Grids and Kriging is presented in Chapter 8. Finally, a summary of conclusions and recommendations for future work is presented.

CHAPTER 2

BACKGROUND

Surrogate modeling is an effective way to study complex problems while reducing the need for high fidelity simulations. One example where this might be useful is optimization. Suppose there is a need to design a wing of an aircraft for a certain altitude and cruise condition. The goal might be to minimize drag while keeping lift and weight constrained. If there are multiple potential sizes of aircraft wings, cruise conditions, and other variables, it would be unwise to perform optimization using CFD every time it is needed. This is where surrogates are useful. Surrogate models allow an initial set of high fidelity simulations to be performed, and statistical methods can be used to perform optimization. This chapter will discuss surrogate modeling and optimization strategies used by authors of other works for the task of airfoil design.

2.1 Comparison of Surrogate Modeling Approaches

Han and Zhang provide an overview of surrogate model-based optimization and the benefit of using surrogate models in optimization [46]. They first give an overview of sampling methods showing comparisons between a Latin hypercube sampling plan and a uniform design distribution. Various methods are then discussed such as the quadratic response surface method, Kriging, and radial basis functions. Error checking and frame-

works for both the surrogate models and optimization process are discussed. Han and Zhang describe how expected improvement predictions can be used to calculate infill of the surrogate models. For a real world example, Han and Zhang try to minimize drag for a transonic airfoil. They first compare quadratic response with Kriging. It is shown in their research that Kriging is able to produce a flatter pressure distribution than a quadratic response surface. For their second design, they attempt to maximize the lift to drag ratio and minimize the weight for a subsonic transport-aircraft wing. They compare accuracy of both the quadratic response surface and Kriging noting a similar accuracy for both models. The conclusions that they are able to draw are that, while regression models are accurate for smooth distributions, they are not well suited for interpolation of highly non-linear systems that may contain discontinuities.

Butnaru, Pflüger, and Bungartz discuss the application of computational steering using Sparse Grids [5]. Computational steering is a way to manually modify an automatic procedure of some kind in order to get a desired result [42]. For the case with Sparse Grids, one ultimate goal is to use steering to model a real world time dependent physics simulation. The authors point out that for complex design problems, full exploration of the design space is impractical. They go on to discuss the theory behind Sparse Grids and the use of computational steering to modify the sparse grid space over time. For their real world application, they explore using Sparse Grids to model incompressible viscous fluid flow for a lid driven cavity. They use a level 3 sparse grid for 15 time steps. Predictably, the highest errors are produced at a time step of 0. This occurs because the lid suddenly goes from a state of motionlessness to a state of motion. In reality this is not physically possible. The

authors conclude with a statement about their work and state interest in using these same techniques utilizing GPGPUs.

Baar and Harding explore the application of using gradient-enhanced Sparse Grids for the purpose of uncertainty quantification [7]. The paper compares gradient-enhancement with Sparse Grids with the more traditional Kriging surrogate model. Both techniques are discussed briefly, and an overview of Monte Carlo sampling for the uncertainty quantification is discussed [38]. A more complete overview of Sparse Grids is given as well as the gradient-enhancement procedure. For testing, a two-dimensional Rosenbrock function is used. Gradients are determined through a complex step process. By using gradient-enhancement, the authors show that errors are significantly reduced, and fewer points are needed for the same level of accuracy as with non-gradient-enhanced sparse grids. They then account for gradient noise and determine that errors do not increase until the magnitude of the noise becomes greater than 0.01. Comparing this method with Kriging, it is shown that while sparse grids are less accurate than both Kriging and gradient-enhanced Kriging for this test, gradient-enhanced sparse grids perform best overall. However, one issue with the gradient-enhancement is that computational costs increase by a multiple of 2d+1, where ”d” is the number of dimensions of the domain, when the method is used. Applying uncertainty quantification to the Rosenbrock function, a similar decrease in relative error is observed. For a real world design problem, the authors model drag of a transonic airfoil. In this case, increases in accuracy are not as large using gradient-enhancement as with the Rosenbrock case. Furthermore, the number of samples needed for target accuracy increases exponentially with dimension, giving way to the curse of dimensionality.

However, advantages of the gradient-enhancement are still observed when using it for the purpose of uncertainty quantification.

2.2 A Study of Airfoil Parameterization Techniques

Li, Bouhlel, and Martins explored the use of singular value decomposition (SVD) and mode shapes of a database of different airfoils for the task of airfoil parameterization [28]. With the SVD method the authors demonstrate how coordinates for a series of airfoils can be assembled into a matrix, and use an equation of the form **y** = A***x**, where ”A” is a set of corrdinates for multiple airfoil samples, and ”x” is a vector of weights used on ”A” to perturb airfoil ”y.” Decomposing this matrix using SVD, columns in the ”U” matrix represent modes for the database of airfoils. The authors go on to split these into camber and thickness based modes. They then demonstrate that decomposing mode shapes in this way simplifies the imposition of constraints on airfoil thickness. They then show how to put limitations on the number of modes used by observing how many modes need be used to keep errors associated with shape predictions and airfoils within a certain tolerance. For the most accurate results, a similar number of camber and thickness modes should be used. Before performing further analysis, they perform preprocessing on all airfoils in their database to produce smooth shapes, rotate airfoils to 0 degrees angle of attack, and sharpen the trailing edge of airfoils with blunt trailing edges. They then split this database between subsonic and transonic airfoils. To enforce boundaries, mode shapes past certain maximum and minimum thickness and camber modes were discarded. To ensure smooth shapes are produced, Laplacian smoothing is also enforced. Kriging was used to drive the

optimization process. Because the training model for Kriging scales with complexity, a clustering approach was used that divided the domain of the lift and drag surrogate models into smaller Kriging surrogate models and performed evaluations using specific models based on location in the domain. One of the goals of the authors was to determine the accuracy of optimization with surrogates as opposed to just using CFD. To accomplish this they decided to use a gradient based approach for the task of optimization. Testing for both subsonic and transonic flow, they were able optimize drag to 0.4 drag counts for the subsonic case and 2.5 counts for the transonic case.

Masters et al. compare a variety of airfoil parameterization techniques [31]. They explored seven different parameterization methods (B-splines, CST, SVD, PARSEC, Hicks-Henne bump functions, Bézier surface Free-form Deformation (FFD), and Radial Basis Functions (RBF)). To test airfoil geometries, they used a database with NACA four-series airfoils as well as a database with the UIUC airfoils. As in the work of Li et al. Masters et al. [28] implemented smoothing to remove any potential oscillations during parameterization. Instead of using a sharp trailing edge, for all airfoils in their database, the authors used a blunt trailing edge with consistently the same thickness for all airfoil shapes. A NACA 0012 was used for the initial geometry. To fit other airfoil shapes, a weighted least squares solution was applied using a Moore-Penrose pseudo-inverse [1]. Kulfan's geometric tolerance [25] was used to determine whether a parameterization technique managed to fit a particular airfoil with sufficient accuracy.

First, Masters et al tested the accuracy of B-splines using both uniform and cosine distributions. It was found that for a uniform distribution, very high order polynomials were

needed (12th to 14th-order), and the number of parameterization attempts that converged based on Kulfan's geometric tolerance decreased once 25 design variables or more were used. By using a cosine distribution, all distributions of third-order or higher converged after using 20 design variables. For the CST case, the effects of modification to the leading edge were explored. Comparisons were made for four cases. Tests were run with and without the leading edge modification using both the NACA and UIUC libraries [17]. Without using the leading edge modification, tests on several UIUC library airfoils performed the worst, needing over 30 design variables to reach 90 percent convergence. All other cases reached convergence using little under 20 design variables. For the SVD parameterization case, three sets of mode shapes were created. One set was created using the NACA airfoil database, another was created using the UIUC database, and the third was created using a combined set of both databases. All three SVD methods converged only after using between 10 and 20 design variables for both the UIUC and combined libraries. All NACA geometries converged with less than 10 design variables using the NACA modes. For the combined and UIUC modes, only half of the NACA geometries ever converged. For the remaining test cases, with the exception of PARSEC [39], convergence occurred only after using between 10 and 20 design variables. PARSEC was limited due to each variable having a very specific job in the parameterization of airfoil geometries. With these limitation, only 10 percent of all geometries were able to converge to Kulfan's tolerance. Further analysis of parameterization methods will be discussed in later chapters.

CHAPTER 3

COMPUTATIONAL APPROACHES FOR SURROGATE MODELING

This chapter will discuss the various steps required to create surrogate models. First the development of sampling plans is discussed. This includes a screening step for limiting the number of variables that are used by the surrogate model as well as various sampling plans and their advantages and disadvantages. Next surrogate modeling strategies are described. Then refinement strategies are evaluated for increasing the accuracy of the surrogate model. These topics are discussed in greater detail in the book by Forrester, Sobester, and Keane [12].

3.1 Sampling Plans

3.1.1 Screening

Screening is an important part of the engineering design process. For any type of simulation, there could be dozens or even hundreds of variables that have some influence on a particular design condition. Most of these, however, will usually either have only a minute effect on the design condition, or will have minimal effects in conjunction with other variables. The function that uses these variables to minimize or maximize a certain condition is known as the objective function. Forrester et al. [12] describes four different

criteria to look for in the relationship between these variables and a potential objective function.

- If $\partial f/\partial x_i = 0$, $\forall \mathbf{x} \in D$, x_i can be neglected
- If $\partial f/\partial x_i = k \wedge k$ is a non-zero constant, $\forall \mathbf{x} \in D$, x_i is linear and additive
- If $\partial f/\partial x_i = g(x_i)$ where $g(x_i)$ is not constant, f is nonlinear in x_i
- If $\partial f/\partial x_i = g(x_i)$ where $g(x_i,x_j,\dots)$ are not constant, f is nonlinear in x_i and involved with interactions with other variables

3.1.2 Sampling Plan Design

In order for a surrogate model to be created, an initial sampling plan is needed. Ideally, this sampling plan should not only be able to fill the sampling space as efficiently as possible, but it should also allow for effective refinement. Forrester states that a good sampling plan should be both stratified and space filling. What this means is that the sampling plan will not only create an even distribution of points throughout the sampling space, but it will also produce an even distribution of points for each dimension of the sampling space. For example, if a 1000 point, three-dimensional, full factorial grid were used, it would be space-filling; however, it would not be stratified. If the grid was projected onto one dimension, there would only be 10 points represented along that dimension with 100 samples at each location. A better solution that would be both space filling and stratified would be the use of either Latin Squares or Monte Carlo sampling. Monte Carlo sampling uses a random uniform distribution of points. Latin Hypercubes take this one step further by creating a grid of subspaces of potential sampling points in the sampling space.

Latin Hypercubes are based on the concept of Latin Squares [6]. Latin Squares are similar to a Sudoku puzzle in that a series of N numbers or letters are used to fill a two-

C	B	A	D
D	A	C	B
B	C	D	A
A	D	B	C

Figure 3.1

Latin Squares

dimensional NxN grid of squares. Each number or letter may only be represented once in each row and column. This is shown in Figure 3.1 [32]. Latin Hypercubes extend this concept into d-dimensional space. A point is chosen at random in the sampling space, and the sub-space that it is located in is recorded. Recursively, other points are chosen at random with restrictions based on the d-dimensional ranks of subspaces with already allocated points. Latin Hypercubes are therefore made to be as stratified as possible with close to an even distribution of points along each rank of the sampling space.

3.2 Creating a Surrogate

A surrogate model is a statistical way to predict the behavior of an objective function with respect to changes in specific variables [33]. Most surrogate models are non-intrusive in that they are not based on physical aspects of the design problem. There are many techniques for developing and using surrogate models for engineering design. Two mod-

els that immediately come to mind are polynomial regression and radial basis functions [34]. Polynomial regression functions are functions of the form **y** = A***b**, where A is a matrix of sample points of increasing polynomial order. The vector **b** can be found using a least squares solver such as Cholesky, QR, or SVD factorization. Cholesky is used in Equation (3.1).

$$\mathbf{b} = (A^T A)^{-1} A^T \mathbf{y} \tag{3.1}$$

Predicted values $\hat{\mathbf{y}}$ can then be determined using **b** as weights with a set of points **x** as shown in Equation (3.2).

$$\hat{\mathbf{y}} = b_0 + \sum_{j=1}^{n} b_j \mathbf{x}^j \tag{3.2}$$

Polynomial basis functions, however, are not immune to the curse of dimensionality. As seen in Equation (3.3), as the order of the basis function increases, so does the order of complexity of the solver, increasing by $O(n^d)$.

$$\hat{\mathbf{y}} = b_0 + \sum_{j=1}^{n} b_j \mathbf{x}^j + \sum_{i=1}^{n} \sum_{j \leq i}^{n} b_{ij} \mathbf{x}^i \mathbf{x}^j \tag{3.3}$$

Radial basis functions (RBF) offer a solution to this problem by incorporating radially symmetric basis functions instead of polynomials [24]. This gives RBF the advantage of being able to select a number of nodes from which evaluations are made based on the distance or norm from those points. Like polynomial basis functions, an initial set of data with known values is selected. Points in the radial basis functions can be chosen randomly or based on given information as would be the case in something like a deep learning neural network [27]. Instead of polynomial **x** values being used in a matrix A, the norm of values **x**-**c** are used to populate a matrix Φ. The equation for **y** takes the form **y** =

Φ*λ. Equations (3.4) and (3.5) show the Cholesky factorization of β and the function for calculating $\hat{\mathbf{y}}$.

$$\beta = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \tag{3.4}$$

$$\hat{\mathbf{y}} = \sum_{j=0}^{K} \lambda_j \phi(||\mathbf{x} - \mathbf{c}^{(j)}||) \tag{3.5}$$

3.2.1 Kriging

A special type of surrogate modeling method was developed by Danie G. Krige for the purpose of statistically modeling subterranean geography for mining operations [13]. This model was named Kriging in his honor. Kriging, which is a form of RBF, is similar to Gaussian based basis functions in that it uses radial data from specific centralized points to make statistical predictions. The difference comes from how Kriging uses the statistics of the underlying data to modify the shapes of each individual basis function. Since each basis function has a slightly different shape depending on its location in the domaing of the surrogate model, Kriging basis functions tend to be more accurate than Gaussian basis functions. Correlations in response values between data points and basis function central locations are determined using Equation (3.6), and a correlation matrix Ψ can be found. In this equation, $\mathbf{x}_*$ is the vector of basis function node locations.

$$\psi = exp\left(-\sum_{j=1}^{k} \theta_j |x_j^{(i)} - \mathbf{x}_j|^{p_j}\right) \tag{3.6}$$

In this function, *p* controls how sharp the peak of each basis function is, and θ controls the extent of the influence of each basis function. To determine values for *p* and θ, a maximum

likelihood evaluation can be made by evaluating where the first derivative of the likelihood function, shown in Equation (3.7), is zero.

$$L = \frac{1}{(2\pi\sigma^2)^{n/2}|\Psi|^{1/2}} exp\left[-\frac{(\mathbf{y}-\mu)^T\Psi^{-1}(\mathbf{y}-\mu)}{2\sigma^2}\right] \tag{3.7}$$

In this equation μ and σ^2 represent the mean and variance of the distances between points in the Kriging surrogate model and are calculated using Equations (3.8) and (3.9).

$$\hat{\mu} = \frac{\mathbf{1}^T\Psi^{-1}\mathbf{y}}{\mathbf{1}^T\Psi^{-1}\mathbf{1}} \tag{3.8}$$

$$\hat{\sigma}^2 = \frac{(\mathbf{y}-\mu)^T\Psi^{-1}(\mathbf{y}-\mu)}{n} \tag{3.9}$$

Using this information, a better likelihood function can be determined by using Equation (3.10). The right hand side of this equation is dependent on the values of p and θ, which can be modified to minimize the likelihood function using optimization techniques. This process is known as training the data.

$$ln(L) \approx -\frac{n}{2}ln(\hat{\sigma}^2) - \frac{1}{2}ln|\Psi| \tag{3.10}$$

In order to train and add new values to the model, correlations can be added to the global matrix $\tilde{\Psi}$. This augmentation can then be used to retrain the maximum likelihood of $\mathbf{y}$ as seen in Equation (3.11).

$$lnL \approx -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\hat{\sigma}^2) - \frac{1}{2}ln|\tilde{\Psi}| - \frac{(\tilde{\mathbf{y}}-\hat{\mu})^T\tilde{\Psi}^{-1}(\tilde{\mathbf{y}}-\hat{\mu})}{2\hat{\sigma}^2} \tag{3.11}$$

Using a correlation matrix, predictions can be made at random points $\hat{\mathbf{x}}$ in the Kriging space using the trained sampling points. This is shown in Equation (3.12), where Ψ is the trained set of correlation data, and ψ is the set of correlation data to be determined.

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \psi^T\Psi^{-1}(\mathbf{y}-\hat{\mu}) \tag{3.12}$$

3.2.2 Sparse Grids

Sparse Grids provides a hierarchical method of evaluating a function or problem [4]. There is an advantage in the Sparse Grids approach in that old information is not dependent on new information. Unlike neural networks and Kriging, which both have to be retrained when new data is added, only the new data has to be used for training Sparse Grids. This is accomplished by using multilevel subspace decomposition methods. This also makes the method better suited for parallel processing since refinement can take place in separate areas of the Sparse Grid space without interfering with one another. Sparse Grids work, like polynomial and radial basis functions, by using a series of multilevel basis functions to interpolate between Sparse Grid points. Methods may differ based on whether local or global solvers are used. Some solvers force the use of local basis functions such as local polynomials or B-splines. For a more complex but faster assembly of the Sparse Grid space, global methods that use spectral functions such as Chebyshev, Hermite, and Legendre polynomials can be used.

As an example of how a linear local polynomial grid would be generated, in the one-dimensional subspace, a series of elements can be described as $\mathbf{k} = (k_1, k_2, ..., k_d) \in \mathbb{R}$. Each component in **k** represents some level size of the grid []. The mesh size for each level $\mathbf{h_k}$ would therefore be $(2^{-k_1}, 2^{-k_2}, ..., 2^{-k_d})$. At each level, the vector space of discrete elements in the sparse grid can be described using Equation (3.13).

$$\mathbf{V}_l = span\{\phi_{i,j} | j_t = 0, ..., 2^{l_t}, t = 1, ..., d\} \tag{3.13}$$

Equation (3.14) would then represent the one-dimensional basis functions.

$$\phi_{i,j}(x) = \begin{cases} 1 - \left|\frac{x}{h_k} - j\right|, if x \in [(j-1)h_k, (j+1)h_k] \cap [0,1]; \\ \\ 0, otherwise \end{cases} \tag{3.14}$$

For a multi-dimensional problem, since an evenly distributed hierarchical basis function would produce points for each new level that are halfway between points and boundaries from previous levels, the node value for each new level would then need to be calculated by subtracting the previous level subspaces from the overall space $\mathbf{V}_k$. Using multilevel subspace decomposition, the vector space of the objective function can be calculated, and the hierarchical objective function can be determined. This is shown in Equation (3.15) - (3.19).

$$\mathbf{W}_k = \mathbf{V}_k / \prod_{t=1}^{d} \mathbf{V}_{k-e_t} \tag{3.15}$$

$$\mathbf{B}_k = j \in \mathbb{N}^d \begin{cases} j_t = 1, ..., 2^{k_t} - 1, j_t odd, t = 1, ..., d, if k_t > 0 \\ \\ j_t = 0, 1, t = 1, ..., d, if k_t = 0 \end{cases} \tag{3.16}$$

$$\mathbf{W}_k = span\{\phi_{k,j} | j \in \mathbf{B}_k\} \tag{3.17}$$

$$V_n = \prod_{k_1=0}^{n} ... \prod_{k_d=0}^{n} \mathbf{W}_k = \prod_{|k|_\infty \leq n} \mathbf{W}_k \tag{3.18}$$

$$f(x) = \sum_{|k|_\infty \leq n} \sum_{j \in B_k} \alpha_{k,j} * \phi_{k,j}(x) \tag{3.19}$$

An easier way to state this is that each Sparse Grids subspace can be represented by a tensor product of of the one-dimensional components of the subspace as shown in Equation (3.20). Then, for full interpolation, the direct sum of all subspaces is taken as seen in Equation (3.21).

$$V_n = \otimes_{j=1}^{d} \mathbf{W}_k, j \tag{3.20}$$

$$V = \oplus_{k \in \mathbb{N}^p} \mathbf{W}_k \tag{3.21}$$

This hierarchization can be better visualized in the color image in Figure 3.2.

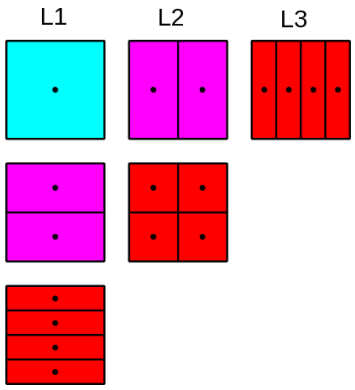

Figure 3.2

Sparse Grid Hierarchization

For higher-dimensional grids, a better way to visualize them would be to represent each dimension of a Sparse Grids space as a one-dimensional set of containers of (n-1)-dimensional Sparse Grids. For a level-zero grid, a single point could be assigned at the center of the n-dimensional grid and given a value of zero. This value would represent the level size in the n-1 dimension. When the level size increases in the n-dimension, a new one-dimensional set of Sparse Grids is generated; however, these new points are assigned a value of zero. Furthermore, the value of the grid points from the previous level would increment by one. This would indicate that the n-1 grid associated with that point has a level size of one. This process can be carried through the entire set Sparse Grids recursively, and grid points can continue to be generated by assigning values of zero to

each new grid point generated and incrementing the values of all other grid points at the previous level by one.

3.3 Surrogate Model Refinement

After an initial surrogate model has been generated, the process of design and optimization can proceed in either one of two directions. If the surrogate model provides a good fit for a given set of data, the optimization process can be started; however, this is rarely the case. In all likelihood, certain sections of the surrogate design space will converge poorly, whether that is due to non-linearity in the data or excessive noise. If the problem is the latter, either data used to produce the surrogate model in those sections can be removed all together, or a smoothing procedure such as SVR can be used. If the problem is the former, additional refinement will be necessary in regions where data has not been appropriately captured. Forrester et al discusses two different strategies for surrogate model refinement. These strategies are prediction-based exploitation and error-based exploration. Figure 3.3 presents a diagram for how the refinement-optimization process might work using artificial neural networks for the purpose of airfoil design [23].

3.3.1 Exploitation and Exploration

Forrester et al. discuss using function evaluations after a minimum value has been located using the surrogate model. Once a point has been placed at this location, the model is retrained. Ideally, if the true optimal value is different from this original point, its location or one nearby is revealed. However, in many cases, this refinement approach will fail to capture global optimal points. This can be seen in Figure 3.4. Every time a minimum

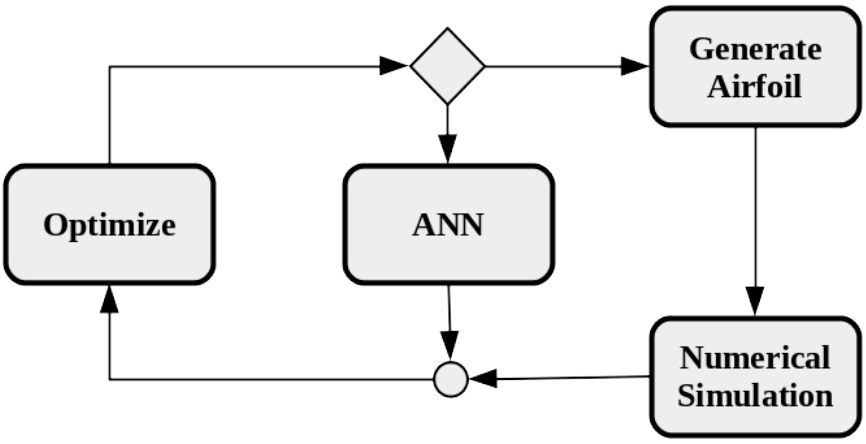

Figure 3.3

Optimization Using Artificial Neural Networks

is reached, and a RBF node is added, it marches away from the true global minimum. Therefore, an exploitation approach such as this will never find the true minimum.

One approach to refinement that is more likely to find global errors is through error base exploration [3]. With this approach, the objective is to search for the maximum mean squared error (MSE). This can be calculated using Equation (3.22).

$$\hat{s}^2(\mathbf{x}) = \sigma^2 \left[1 - \psi^T \Psi^{-1} \psi + \frac{1 - \mathbf{1}^T \Psi^{-1} \psi}{\mathbf{1}^T \Psi^{-1} \mathbf{1}} \right] \tag{3.22}$$

Though more likely to find a global minimum than exploitation models, there are two problems associated with the exploration approach. One problem is the speed of convergence. Since exploration seeks to minimize errors, absolute convergence may take significantly longer. The second problem occurs when the model assumes that it has reached a converged state before it actually has. If points are chosen in such a way that the function does not change over time, the model will assume that there are no errors to be explored. This

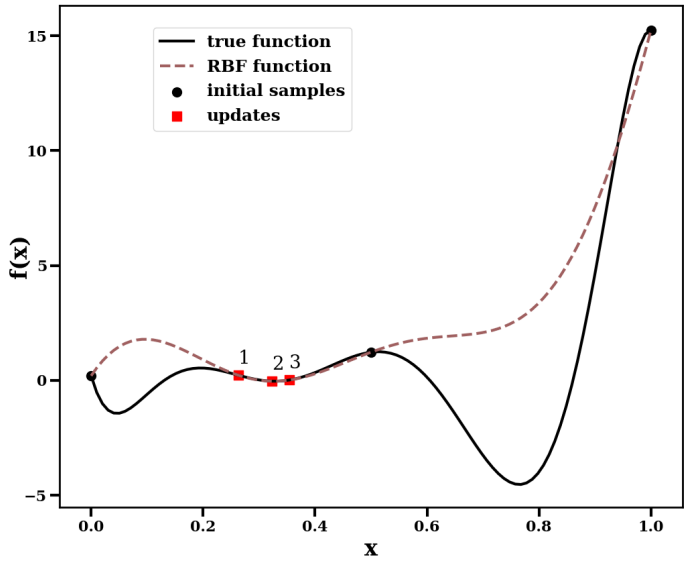

Figure 3.4

Prediction Based Exploitation

type of exploration issue is shown in Figure 3.5, where three initial samples produce a flat line, and all errors in the model are assumed to be zero.

3.3.2 Conditional Likelihood Approach

Conditional likelihood approaches take away the question of, ”does potential improvement exist?” and replaces it with, ”is it possible potential improvement may exist?” This works by setting a goal for f(**x**) and maximizing the conditional in-likelihood at a point $\mathbf{x}_p$. This is done using a slightly modified version of Equation (3.11) as seen in Equa-

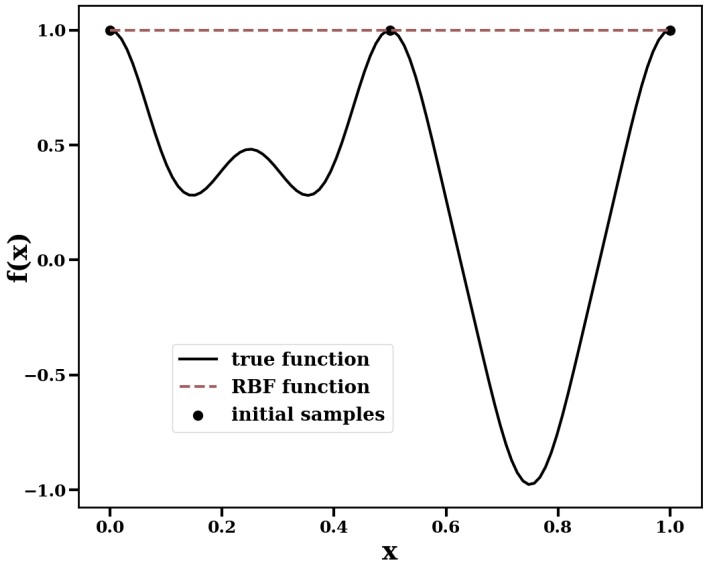

Figure 3.5

Exploration Method Failing to Converge

tion (3.23). The modification comes into the calculation of **C** and **m**, which are calculated using Equations (3.24) and (3.25) respectively [19].

$$-\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\hat{\sigma}^2) - \frac{1}{2}ln|\mathbf{C}| - \frac{(\mathbf{y}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{y}-\mathbf{m})^T}{2\hat{\sigma}^2} \tag{3.23}$$

$$\mathbf{m} = \mu + \psi(\hat{y}^p - \mu) \tag{3.24}$$

$$\mathbf{C} = \Psi - \psi\psi^T \tag{3.25}$$

CHAPTER 4

AIRFOIL PARAMETERIZATION METHODS

In order to build a surrogate model that can predict lift and drag based on airfoil geometry parameterization, there needs to be a way to quantify the generation and modification of airfiol shapes. This chapter will explore various ways of parameterizing airfoils. Ultimately, what will be created will be a method that can take a series of arguments between zero and one and produce a wide range of different airfoil shapes from subsonic to supercritical. This will be crucial for the development of statistical models.

4.1 NACA Airfoil Series

The NACA four digit airfoil series are a set of airfoils labeled with the format NACA CPXX where the **C** term specifies camber, **P** specifies the location of the camber from the leading edge, and **XX** specifies the maximum thickness of the airfoil. Substituting **XX** with **T** the profile of a NACA 4 series can be calculated using Equations (4.1), (4.2), and (4.3) [16].

$$y_c = \frac{C}{P^2}(2Px - x^2)\ if\ 0 \leq x < P \tag{4.1}$$

$$y_c = \frac{C}{(1-P)^2}[(1-2P) + 2Px - x^2]\ if\ P \leq x \leq c \tag{4.2}$$

$$y_t = \frac{t}{0.2}(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4) \tag{4.3}$$

This was one of the earliest attempts to standardize the parameterization of airfoil shapes. While NACA 4 digit airfoils tend to use integer values for each of the four digits, any floating point number could be used by the algorithm. Even so, the general shape of the airfoil is fixed, and changes can only be made to the camber line and the overall maximum thickness. This heavily restricts the range of shapes this airfoil can take on. Figure 4.1 shows two airfoils parameterized using the NACA four digit series.

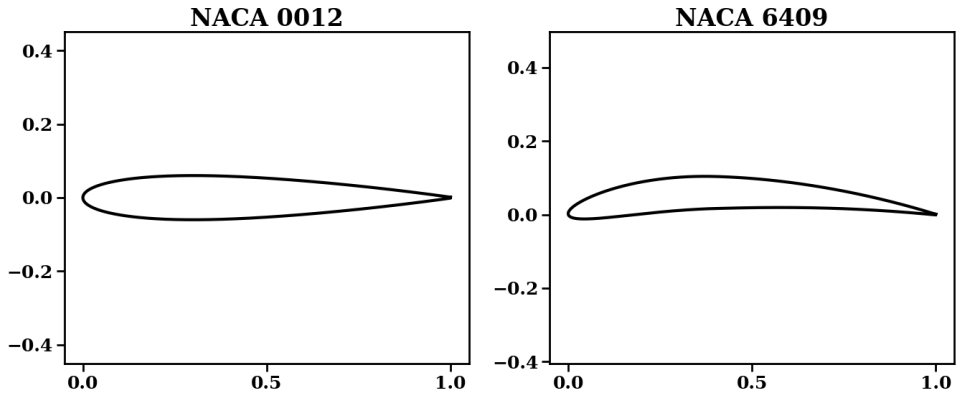

Figure 4.1

NACA 4 Digit Airfoil Series

The NACA five digit airfoil series uses the same thickness profile as the four digit series. However, it does make changes to how camber is calculated. Instead of specifying the maximum camber directly, it is determined based on a desired lift coefficient. This lift value is determined by multiplying the first digit of the airfoil by 1.5. The location of the maximum camber is more precise. It is calculated by dividing the second third numbers of the airfoil by two. Finally, the digits four and five determine the maximum thickness of the

airfoil. Overall, the additional complexity allows for a wider range of airfoil shapes when compared with the four digit series.

4.2 Parameterization using Splines

Splines provide an advantage over schemes used for NACA airfoils in that there are infinitely many shapes that can be generated. Splines are functions that are built using a series of piece-wise polynomials [37]. To build a spline that fits a desired shape, the polynomials are augmented with the addition of weights.

Bézier curves are types of parametric curves that can be used for a variety of tasks including shape parameterization. Bézier curves are defined by functions comprised of Bernstein basis polynomials [43]. The smoothness that Bernstein polynomials provide allow Bézier curves to accurately map to a large variety of different shapes. Not only is this method used in airfoil and wing parameterization, but it is also used for tasks such as smoothing objects in video games and designing fonts. There are distinct disadvantages to using Bézier curves alone for parameterization. When trying to solve for a best fit for a shape such as an airfoil, Bézier curves fit using a least-squares solution may have trouble fitting the trailing edge correctly. One solution is to require the first and last terms of the Bézier function to be zero. This solution works since only the first and last terms control the trailing edge of the airfoil.

Bézier functions are limited in the fact that they are continuous functions over an entire domain. This means, that to increase the accuracy of the function, higher-order Bernstein polynomials must be used. This becomes problematic with more complex shapes,

as higher-order polynomial approximations are less likely to produce smooth shapes. One solution to this problem is to use B-splines. B-splines (which is short for Basis splines) [8] are spline functions defined by piecewise collections of Bézier functions over a larger domain in such a way as to create a smooth curve that uses piecewise-functions to construct a larger function. These functions are separated by structures called knots. With traditional B-splines, these knots are spaced out equally. Knots can be either internal or external, with external knots lying on the endpoints and internal knots being spread out evenly within the parametric space. The number of internal knots is dependent on the number of total knots and the dimension of each Bézier curve. For a set of d-dimensional curves, at least d+1 knots must be used as external knots to correctly generate a high-enough order function at the end-points.

Beyond uniform B-Splines are Non-Uniform Rational B-splines (NURBS). NURBS generalize the Bézier curve to an even greater level. These work by providing each B-spline element with a weight. This weight increases the effect of that particular element on other elements in the B-spline domain.

4.3 CST Parameterization

Using a more naïve implmentation of Bézier and B-spline curve parameterization, a single curve must be used to maintain a smooth curve around the leading edge of an airfoil. However, the abrupt change at the leading edge makes parameterization more difficult since the shape is harder to capture with lower-order polynomials, and higher-order polynomials tend to create shapes that are not quite as smooth. The Class Shape Transfor-

mation, or CST method, provides a way to circumvent this problem. The CST method is an airfoil parameterization technique developed by Kulfan [26] that takes a function for a class of shapes and combines it with a flexible shape function like Bézier curves, B-splines, or Chebyshev polynomials [15]. The class function is a smooth function that provides an initial mold for the shape that is being parameterized. This allows the parameterization scheme to be broken into many sections while still providing a smooth curve. For example, an airfoil could be subdivided into an upper and lower surface or into camber and thickness.

Equations (4.4), (4.5), and (4.6) show the equations for the class function, shape function, and full CST function used by the traditional implementation of the CST method.

$$C_{N2}^{N1}(x) = x^{N1}(1-x)^{N2} \tag{4.4}$$

$$S(x) = \sum_{k=0}^{n} \binom{n}{k} p_k x^k (1-x)^{n-k} \tag{4.5}$$

$$f(x) = C_{N2}^{N1}(x) S(x) + x\Delta z_{te} \tag{4.6}$$

Of notice in these equations is the addition of the Δz_{te} term in the CST function. Since the CST method is a transformation of a base shape, and since the shape being transformed typically does not have an open trailing edge, this extra term is needed if an open trailing edge is needed. This might be necessary since the class function forces zero values at the leading and trailing edge. Figure 4.2 shows airfoils parameterized using CST for the upper and lower surfaces. In the figure, the red x points are data collected for the actual geometries of the airfoils. The black line is a shape fitted using CST parameterization. Looking at this figure, CST does a good job of capturing the upper and lower surfaces for

both airfoils; however, the very far end of the trailing edge of the SC(2)-0612 airfoil does have difficulty fully converging. This could be fixed by providing a value to Δz_{te}.

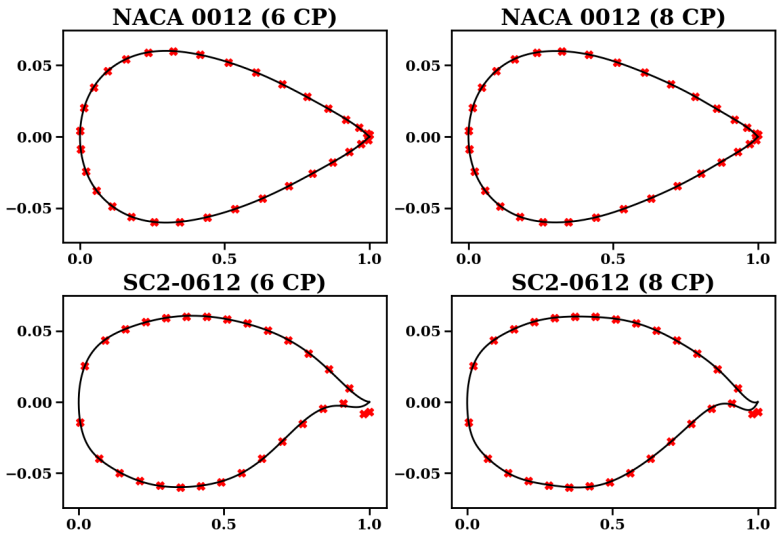

Figure 4.2

CST Parameterization

A modification was made to the CST class function to generate a base shape resembling a more realistic airfoil. This modification is shown in Equation (4.7).

$$C_{mod}(x) = (1-x)\sqrt{x}\sum_{k=0}^{n} w_k x^k \tag{4.7}$$

As seen in this equation, a polynomial function is appended onto the basic class function of Kulfan. This was done in large part because it would make mapping normalized Latin Hypercube and Sparse Grids values easier. The basic airfoil class shape, shown on the left image in Figure 4.3, is far too thick to be used for any type of wing design. With the

modified class function, the weights in the polynomial function can be determined such that the class function will fit a particular airfoil shape. The right image in Figure 4.3 shows the new base shape using the modified class function. Noticeably, it is very similar in shape to a NACA 6412 airfoil, since that was the shape used to perform the modification. In order to get additional shapes, the p term of the shape function can be modified. With the modified class function, a value of 1.0 can be used as the midpoint of the p vector space. Another useful approach to parameterizing an airfoil would be to use modifications to the chord and thickness instead of the upper and lower surface. This approach would make constraining the thickness easier and would present more flexibility in modifying the leading edge radius and trailing edge thickness.

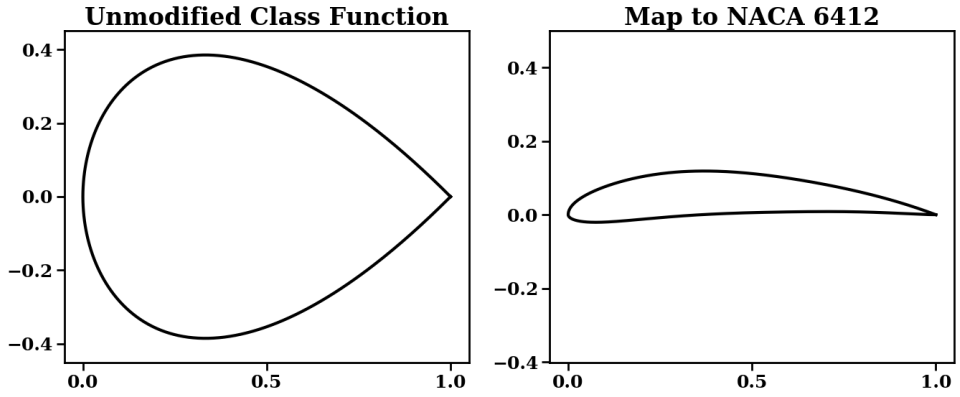

Figure 4.3

Modification to CST Class Function

CHAPTER 5

EVALUATION OF SPARSE GRID SURROGATE MODELS FOR AIRFOIL ANALYSIS

This chapter focuses on the application of Sparse Grids surrogate models for airfoil analysis at low subsonic Mach numbers. Examples of the use of Kriging methods for airfoil analysis can be found in the current literature [21]. The problem of low-speed analysis was selected because of the existence of very fast, lower-order analysis methods that could be used to rapidly generate the required training sets needed to generate a surrogate model. The XFoil panel code [9] was selected for this purpose. For low Mach numbers, XFoil can generate reliable extimates for aerodynamic parameters such as lift and drag and only requires a few seconds of CPU time to run on a modern workstation. For real-world design and analysis, higher-fidelity models such as full Navier-Stokes solvers must be used; however, for the initial phase of this research, XFoil was deemed to be sufficiently accurate. XFoil's speed allowed it to be executed directly from Fortran or Python.

The approach taken in this evaluation was to first determine the accuracy of a surrogate model for predicting airfoil lift and drag for a known family of airfoils (the NACA four-series airfoils). This is a first step toward the ultimate goal of coupling the surrogate model with optimization methods to design airfoils with specified parameters. A series of tests were run with a symmetric airfoil (NACA 0012) and a cambered airfoil (NACA

6412) where Mach number and Angle of Attack (AOA) were used to define the dependentvariables for the surrogate model. For viscous simulations, the Reynolds number was computed to be consistent with Mach number at an altitude of 10,000 ft.

5.1 Overview of Sparse Grid Software

Both Tasmanian [40] and SG++ [36] provide a variety of different interpolation rules that define the location of the interpolation points. These can vary from uniformly spaced local rules, with the local support restricted by the order of the polynomial, to Gauss quadrature type rules where the interpolation nodes are defined by Chebyshev or Gauss-Legendre-type point distributions. The underlying polynomial approximations are hierarchical in nature meaning that higher-order approximations are composed of combinations of lower-order approximations. Figure 5.1 illustrates the difference between a local and Clenshaw-Curtis point distribution. Each Sparse Grid point represents a sampling point at which a simulation must be performed to generate a database of the m-dimensional variables required to construct the surrogate.

After reviewing the capabilities of Tasmanian and SG++, pros and cons of each package were determined. Tasmanian is the more limited of the two and is focused more toward stochastic interpolation and integration. Tasmanian supports a wide variety of interpolation and quadrature types. Furthermore, the package has some basic support for adaptive interpolation and quadrature but is still in the development stage. SG++ is a much more comprehensive package with support for a broader range of problems than Tasmanian. In particular, in addition to basic interpolation and integration using Sparse Grids, SG++

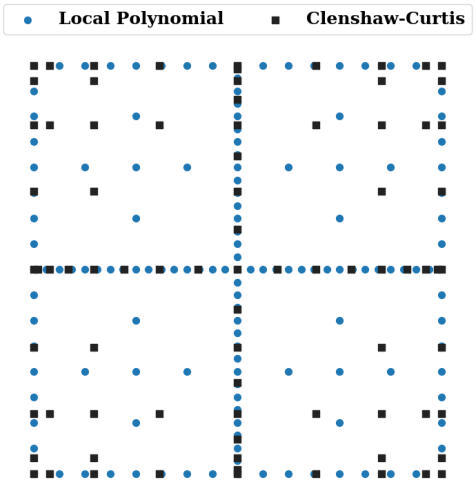

Figure 5.1

Sparse Grid Point Distributions

can be used to solve differential equations and (of most interest in the area of surrogate modeling) provides support for a variety of Sparse Grids-based techniques for developing surrogates for Uncertainty Quantification (UQ).

One potential drawback to the Sparse Grid approach is that the position of the interpolation points are fixed by the choice of interpolation. For surrogate modeling, this can potentially lead to an excessive number of sampling points that must be evaluated to construct the surrogate. Using an adaptive approach can alleviate part of this problem but a better solution might be to construct a sampling plan using a space-filling approach such as Latin Hypercubes to generate sites to evaluate the basis functions of a particular interpolation scheme and then solve a least-squares problem to construct the control points for the

final surrogate. This approach would work best with either radial basis functions (RBF) or Kriging.

5.2 Application of Sparse Grids to Airfoil Analysis Surrogates

As outlined in the preceding synopsis, the pilot application chosen for evaluating Sparse Grids for surrogate modeling is the problem of aerodynamic analysis and design of an airfoil. One of the driving factors in the construction of any surrogate model is the expense required to produce the training samples needed to define the high-fidelity data base that is used to construct the surrogate. For the aerodynamic analysis of airfoils, there are several potential computational methods of varying fidelity and computational cost. At the low end of the spectrum, classical inviscid potential flow panel methods such as XFoil can produce accurate results but only over a relatively narrow range of angles of attack and Mach numbers. The major issue with panel methods such as XFoil is that they are restricted to sub-critical Mach numbers and cannot accurately predict flows in the transonic regime where shock waves are present. This is due to the computation methods inability to model the effects of compressibility. However, as long as evaluations are bounded for flow regimes where XFoil is valid, it is a useful tool for initial surrogate model construction and the development of screening and sampling plans because of its very low computational costs. Several hundred samples can be generated in a matter of seconds, which allows various surrogate model procedures to be tested and validated without resorting to more expensive aerodynamic analysis methods. It is also a useful tool for determining regions of the potential analysis and design space where higher fidelity tools must be used.

In order to better understand the limits of XFoil's analysis range, a series of simulations were performed using the Flowpsi Navier-Stokes CFD code developed at Mississippi State University by Luke et al. [29]. Flowpsi is an academic version of the Loci-Chem CFD code [30]. Flowpsi can predict fully turbulent viscous flow around both 2D and full-3D geometries over a broad range of Mach numbers up to and including hypersonic flow. The code is fully parallel, runs on most HPC systems, and can scale to thousands of processors. Both structured and unstructured grids can be used along with overset grids to reduce the total number of grid points required for complex geometries. Results from Flowpsi were compared with results from XFoil to illustrate the strengths and weaknesses of both codes.

5.2.1 XFoil and Flowpsi Comparisons

The Flowpsi simulations were performed using a 300x100 structured computational mesh for a NACA 0012 airfoil. All computational meshes were generated using the open-source Construct2D mesh generator [18]. Typical runs used 40 processors on MSU HPC systems and ran for 10 to 20 minutes every time. Viscous runs used the Spalart-Allmaras one-equation turbulence model. Initial inviscid and viscous runs were made with Flowpsi where the angle of attack was varied from 2 to 6 degrees and the Mach number was varied from 0.4 to 0.8. These results were used to identify angles of attach and Mach numbers where local transonic flow would produce shock waves. Comparisons were made with XFoil results to demonstrate the limits of XFoil's analysis range. Figure 5.2, Figure 5.3, and Figure 5.4 show computed pressure distributions for Flowpsi viscous simulations for Mach 0.4, 0.6, and 0.8 for three angles of attack (2, 4, and 6 degrees). The results demon-

strate that even at relatively low Mach numbers, transonic flow can occur locally and result in shock waves.

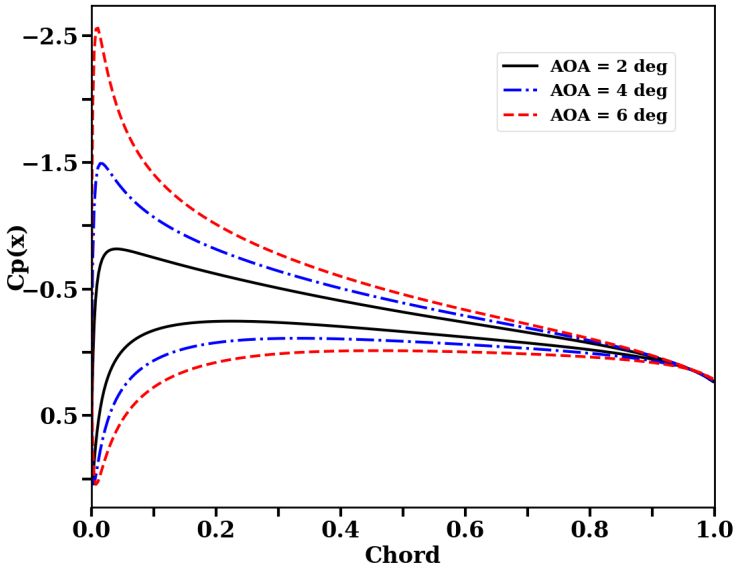

Figure 5.2

Viscous Cp Distribution at Mach 0.4

Figure 5.5 and Figure 5.6 illustrate the differences in results from inviscid and viscous simulations at an angle of attack of four degrees and Mach numbers of 0.4 and 0.8, respectively. As can be seen, for a symmetric airfoil like NACA 0012, viscous effects are small at the lower Mach numbers but increase dramatically with increasing Mach numbers.

Figure 5.7, Figure 5.8, and Figure 5.9 compare XFoil results with results from Flowpsi simulations. At the lower Mach numbers, the results from Flowpsi and XFoil are quite close. As expected, the comparisons degrade with increasing Mach number due to the

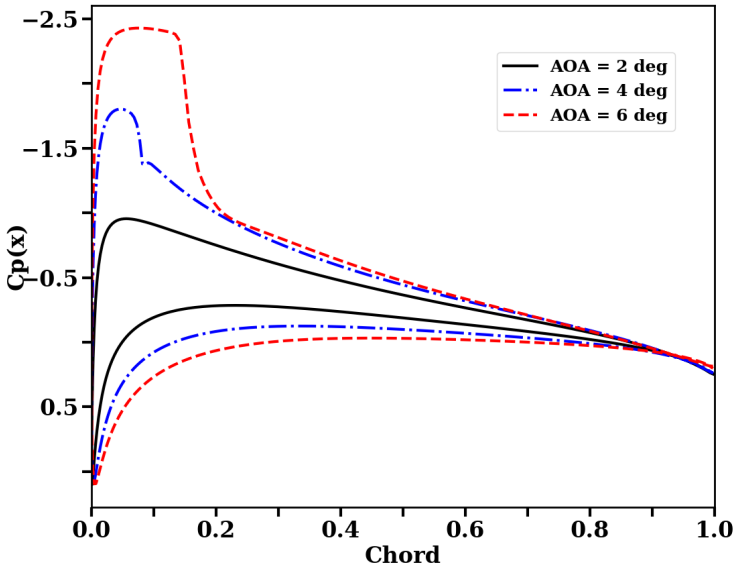

Figure 5.3

Viscous Cp Distribution at Mach 0.6

increased regions of transonic flow that lead to shock waves. Even with boundary layer corrections, XFoil lacks the underlying physics needed to capture the effects of shocks and extended regions of separation. However, the comparison of the results for the lower Mach numbers indicates that XFoil will yield acceptable results in the 0.0 to 0.4 Mach number range as well as angles of attack up to 10 degrees for this airfoil.

5.2.2 Development of Sparse Grid Surrogate Models with XFoil

The Tasmanian Sparse Grid software was used as the basis for constructing Sparse Grid surrogates for a NACA 0010 and a NACA 6412 airfoil. As with the NACA 0012 airfoil used in the Flowpsi study, the NACA 0010 is a symmetric airfoil with a maximum thickness of 10 percent of chord. The NACA 0010 was chosen since it is slightly thinner

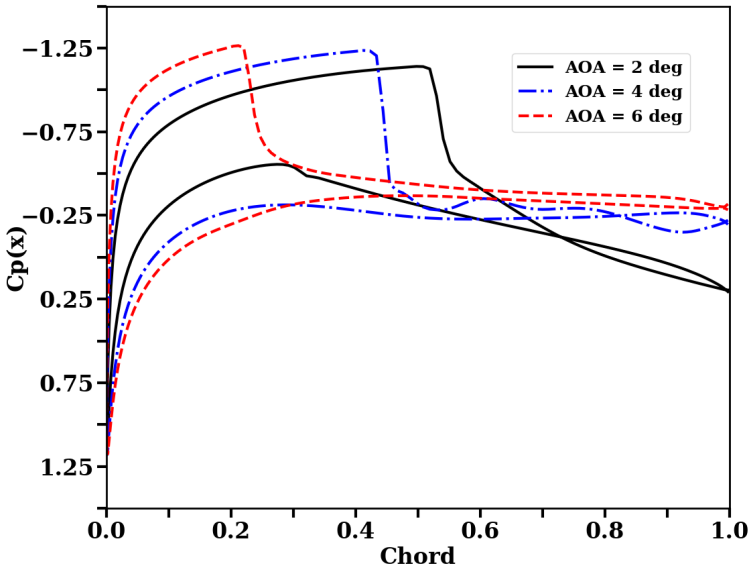

Figure 5.4

Viscous Cp Distribution at Mach 0.8

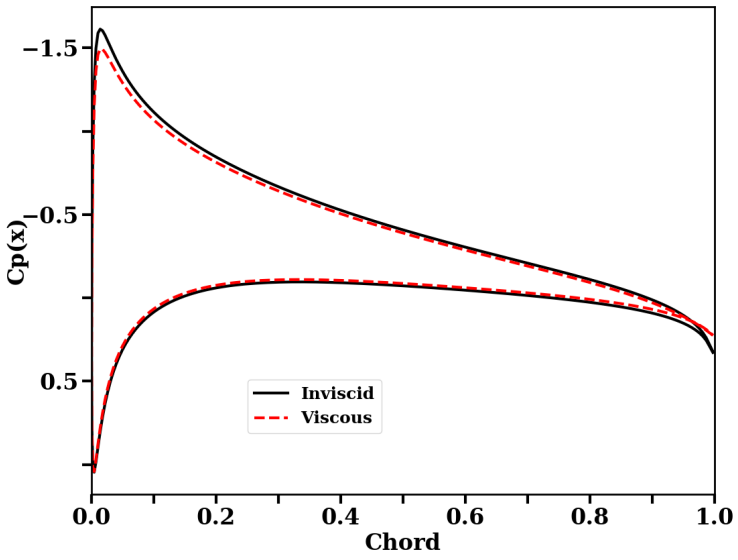

Figure 5.5

Viscous V. Inviscid Comparison at M = 0.4 and AOA = 4.0

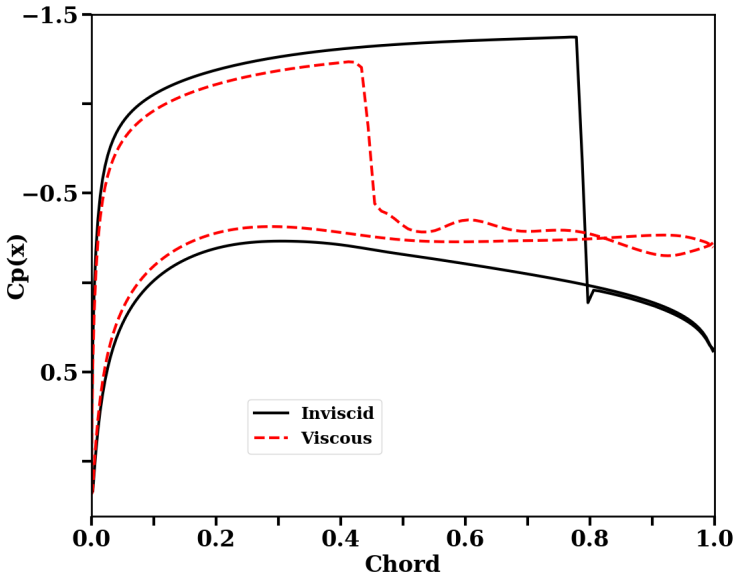

Figure 5.6

Viscous V. Inviscid Comparison at M = 0.8 and AOA = 4.0

than the 0012 and is therefore a little less prone to the development of regions of transonic flow. NACA 6412 is a 12 percent thick cambered airfoil with a maximum camber of 6 percent of chord at a chord location of 40 percent X/C. Unlike symmetric airfoils that produce no lift at zero degrees angle of attack, cambered airfoils produce lift at zero angle of attack and zero lift at a negative angle of attack. This means that the lift curve, which is the lift versus angle of attack, is shifted slightly to the left of the origin. The NACA 6412 was selected because of its high camber and is more realistic of the types of airfoils seen in low-speed applications. The profiles for the NACA 0010 and NACA 6412 airfoils shown in Figure 5.10 illustrate the difference in the shapes of the two airfoils.

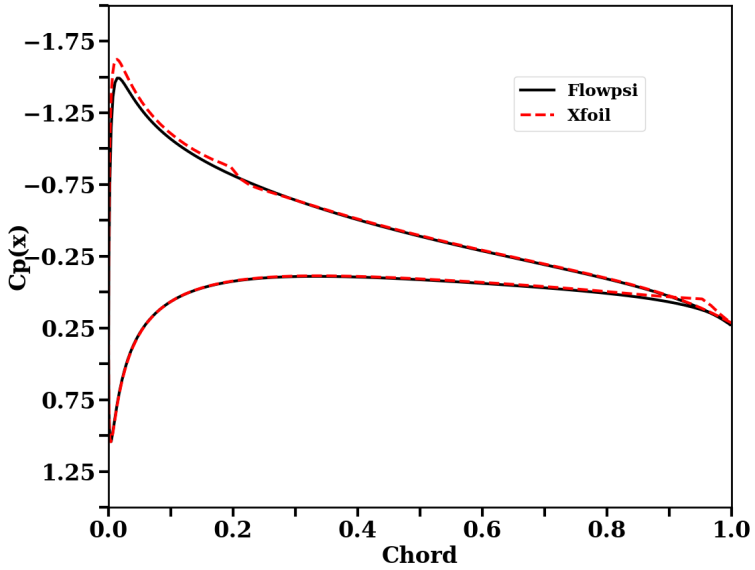

Figure 5.7

Viscous Flow Comparison at Mach = 0.4 and AOA = 4.0 deg

The procedure for constructing surrogates for the two airfoils was as follows. Programs were written in both Python and Fortran to call the Tasmanian Sparse Grid routines required to define a set of sampling (interpolation) points for both local polynomial and global Sparse Grids with Leja and Clenshaw-Curtis point distributions. The order of the interpolation polynomials along with the number of hierarchical levels were varied. The number of hierarchical levels determines the number of sampling or evaluation points required. For example, for a local polynomial grid only 13 points are required for a Level 2 set of Sparse Grids, but 145 points are required for a Level 5 grid. In comparison, a more Gauss-like interpolation rule such as Clenshaw-Curtis, which uses a Chebyshev (cosine) distribution to define the interpolation points, requires only 81 points for a third order ap-

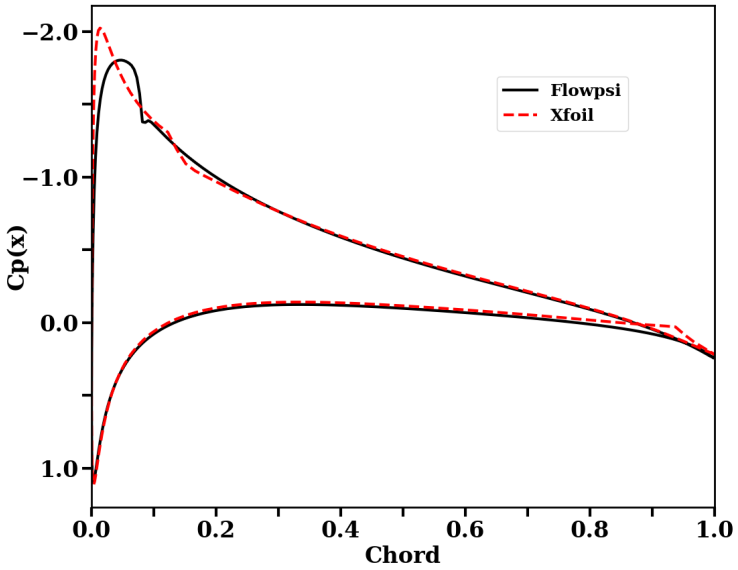

Figure 5.8

Viscous Flow Comparison at Mach = 0.6 and AOA = 4.0 deg

proximation with nine levels. While the Local Polynomial grid tends to distribute points more uniformly throughout the analysis space, the Clenshaw-Curtis grid tends to cluster points closer to the boundaries of the domain. The angle of attack and Mach number pairs, along with a Reynolds number computed to be consistent with the Mach number at a design condition (10,000 ft), were passed to XFoil and the resulting lift and drag coefficients were collected and stored in a file. A separate code was written that would read the lift and drag data and use it to reproduce lift and drag curves at arbitrarily chosen Mach numbers (in this case 0.15, 0.25, and 0.35) and a range of angles of attack (0 to 10 degrees). The resulting polar curves were then compared with results from XFoil at the chosen Mach numbers.

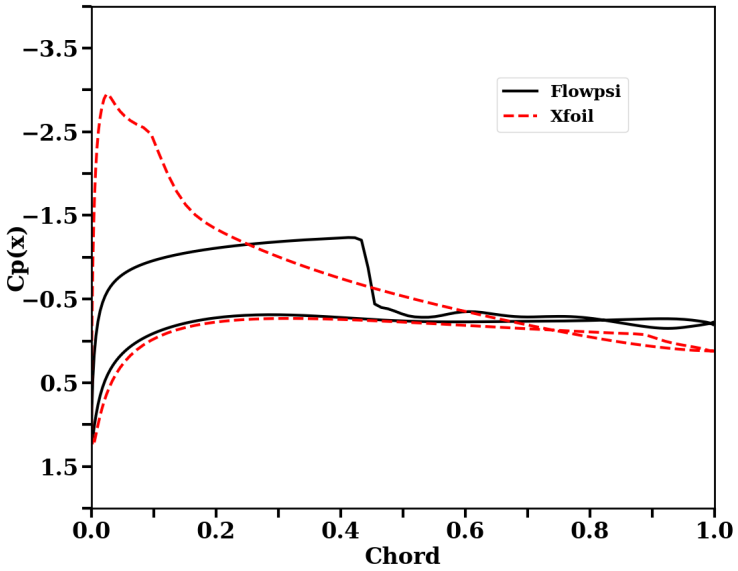

Figure 5.9

Viscous Flow Comparison at Mach = 0.8 and AOA = 4.0 deg

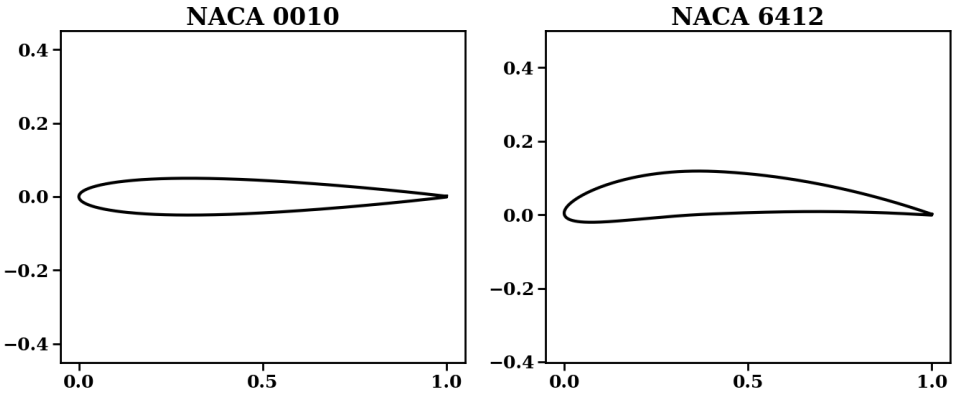

Figure 5.10

NACA 0010 Vs. NACA 6412

Results were generated for different levels of Local Polynomial and Global Sparse Grids along with various orders of approximation and point types.

Surrogate models were first constructed for the NACA 0010 airfoil. Tests were performed to determine the effect of polynomial level, polynomial order, and polynomial type on predicted lift and drag. Results for the surrogate models at Mach numbers of 0.15, 0.25, and 0.35 were obtained for an angle of attack range of 0 to 10 degrees and compared with results from XFoil at the same Mach numbers and angles of attack. XFoil was run in viscous mode to ensure realistic values for drag coefficient. A Reynolds number consistent with the chosen Mach numbers at a design altitude of 10,000 ft was used in the viscous simulations. The two-level grids contained 13 sample points and the five-level grids contained 145 sample points. Due to the linear nature of the lift curve slope in this angle of attack range, both orders of approximation and levels give almost identical results. This is somewhat expected for the 0.25 Mach number since, as shown in Figure 5.1, this Mach number lies on one of the lines of sample points.

Unfortunately, the relative insensitivity to the polynomial level seen in the lift curve prediction did not carry over to the drag prediction. This is seen in Figure 5.11, Figure 5.12, and Figure 5.13, which compare drag polar results for a third-order Local Polynomial Grid for four levels with varying numbers of sample points for each level. As expected, the best correlations occur for the 0.25 Mach number case. However, for the 0.15 and 0.35 Mach number cases, substantial variation in accuracy with polynomial level can be seen. These results indicate that for Local Polynomial Grid surrogates, at least four approximation levels are required for accurate results.

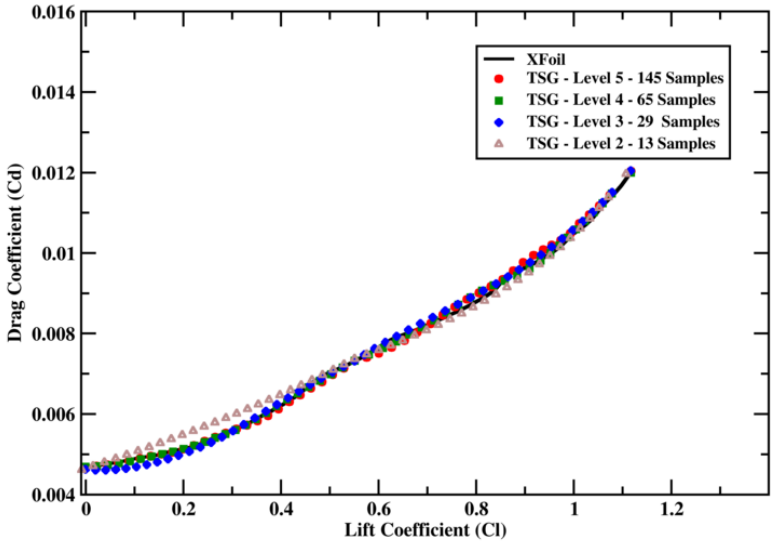

Figure 5.11

XFoil Sparse Grids NACA 0010 M=0.15

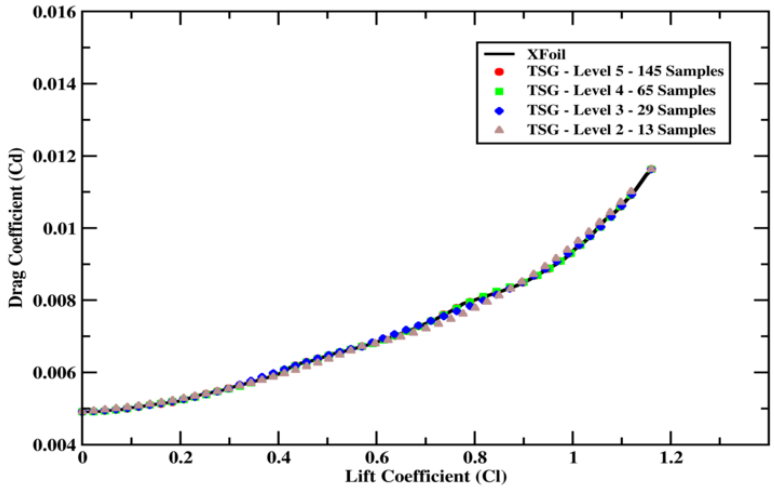

Figure 5.12

XFoil Sparse Grids NACA 0010 M=0.25

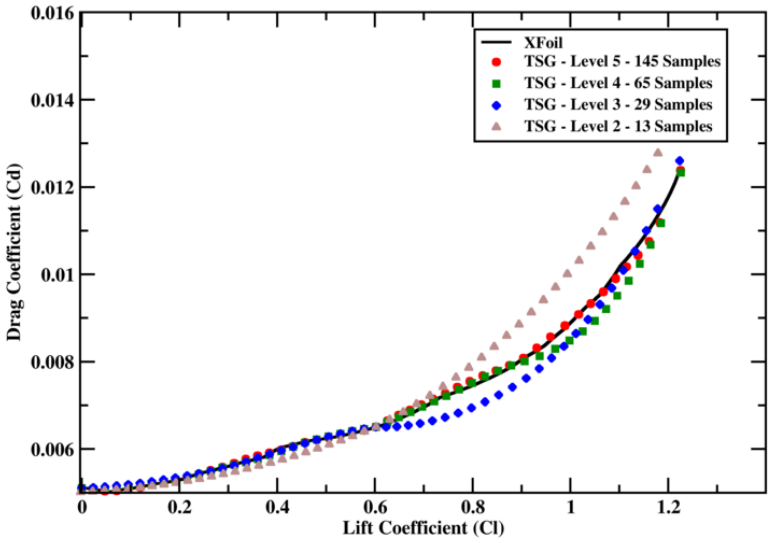

Figure 5.13

XFoil Sparse Grids NACA 0010 M=0.35

A second set of surrogate models were constructed using the NACA 6412 airfoil. This airfoil was chosen because of its high camber which is more representative of airfoils designed for high lift at low Mach numbers such as those found in many general aviation and UAV applications. Since one of the ultimate goals of this research is to develop a framework for both analysis and design using surrogate models, the determination was made that this airfoil was representative of a wider range of potential design applications and a good candidate to be used in the development of surrogate design methods. As with the NACA 0010 airfoil, surrogate models were constructed using XFoil and Local Polynomial Sparse Grids with varying levels of support. In addition, alternate Global Polynomial models were evaluated.

The NACA 6412 surrogates were evaluated at the three Mach numbers used for the NACA 0010 surrogates. Figure 5.14 shows the lift curves for the three Mach numbers using a third-order, five level Local Polynomial Grid. The Local Polynomial grid required 145 sample points but the Leja and Clenshaw-Curtis grids only required 55 and 81 sample points, respectively. As with the NACA 0010 results, the relatively linear nature of the lift curve makes the surrogates for lift somewhat insensitive to the number of sample points. The non-linearity in the curve around an angle of attack of 7.5 degrees is due to issues with the viscous interaction model in XFoil failing to fully converge. These results also illustrate the importance of accurate samples because the resulting surrogate will only be as good as the initial data it is developed from.

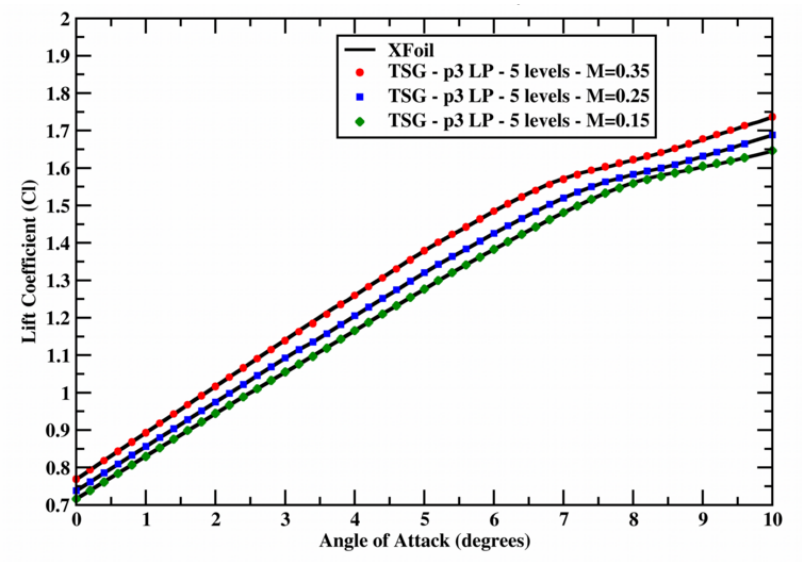

Figure 5.14

Local Polynomial Sparse Grids for NACA 6412 Airfoil

Figure 5.15, Figure 5.16, and Figure 5.17 show the drag polars for the three Mach numbers and compare the Local Polynomial Sparse Grids with the Global Sparse Grids. As with the NACA 0010 results, the best correlations were obtained using the Local Polynomial Grids. Most of the differences in the surrogates in the XFoil predictions occur in the *Cl* value ranges of 1.0 to 1.5, where the data shows an unevenness that provides a further indication that the XFoil viscous interaction procedure was having trouble converging to a realistic solution. In general, the overall performance of the surrogate models are very good and show that Sparse Grids can give accurate results provided that a sufficient number of samples are used.

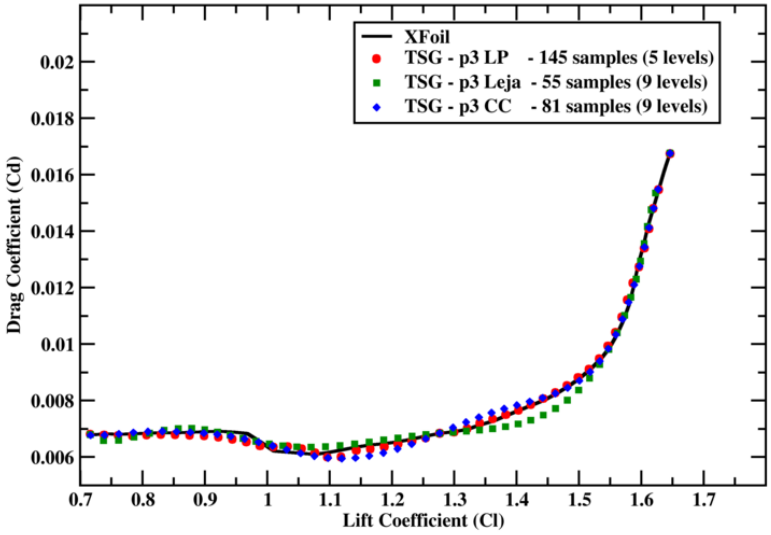

Figure 5.15

XFoil Sparse Grids NACA 6412 M=0.15

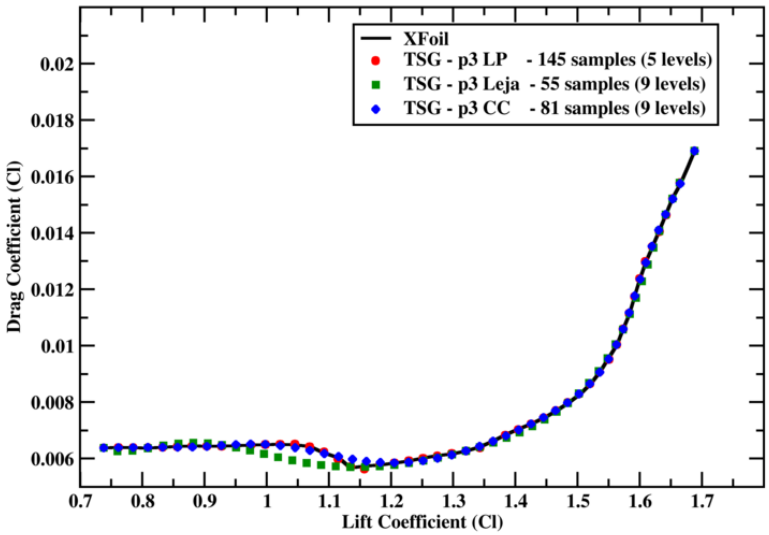

Figure 5.16

XFoil Sparse Grids NACA 6412 M=0.25

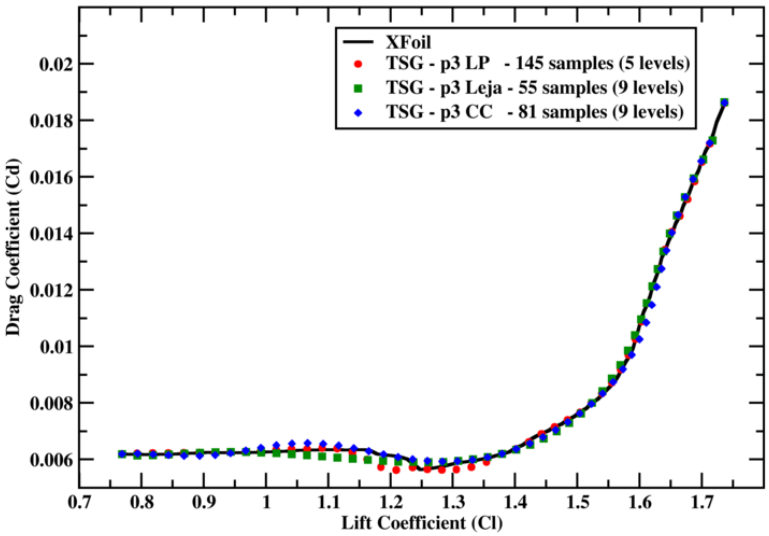

Figure 5.17

XFoil Sparse Grids NACA 6412 M=0.35

5.3 Software Choices for CFD and Surrogate Models

Though Flowpsi and Tasmanian were used in initial research for CFD and Sparse Grids, respectively, neither one was used for the airfoil parameterization surrogate modeling tests. For the CFD solver, the open source code SU2 [10] was used. The SU2 package had a simpler interface that made automating the process of generating configuration files, mesh files, and run scripts easier. Because of the large number of simulations that were needed to build Sparse Grids and Kriging surrogate models, this ability to automate was a necessity.

SG++ was used for Sparse Grids surrogate modeling instead of Tasmanian. This change was made for three main reasons. The first reason SG++ was chosen was because it has more flexible and modular source code than Tasmanian. For example, the Sparse Grid, the Sparse Grid generator, and the Sparse Grid evaluator are treated as separate objects in SG++. This proved to be useful when building multiple Sparse Grids. Because the Sparse Grid data structure, the generator, and the evaluator are separate objects, a single generator can be used to build all Sparse Grids, and grids can be deallocated once evaluators have been built. This has the potential to significantly save memory. The second reason SG++ was chosen over Tasmanian was due to its simpler grid hierarchy. Since grids and grid generators have to be built separately, there is more consistency and similarity in the point distribution between implementations of local and global Sparse Grids. This made changing between local and global methods using the same Sparse Grids point distributions possible. The final reason SG++ was chosen over Tasmanian was because of its much larger selection of routines for making predictions. For example, the B-spline

Sparse Grid allows for extrapolation of predictions to boundaries if the Sparse Grid does not have boundaries. B-spline grids are not available in Tasmanian.

For Kriging, a Python library called PyKriging [35] was used. While there might be faster and more memory efficient algorithms for Kriging, this is not as large of a problem with Kriging as it is with Sparse Grids since Kriging does not restrict how sampling is performed. The codes used for airfoil parameterization and optimization were designed from scratch. One reason this was done was because the algorithms for these tasks were easy enough to implement in a reasonable period of time. The other reason these were written from scratch was as a learning exercise. Designing code tends to make concepts easier to understand than reading about them in a book alone.

CHAPTER 6

OVERVIEW OF OPTIMIZATION METHODS

As a prelude to airfoil design optimization, four commonly used optimization methods (gradient descent, simulated annealing, Cuckoo Search, and Particle Swarm) were evaluated. Based on an initial analysis using the 2D Michaelwicz function, the two genetic based algorithms (Cuckoo Search and Particle Swarm) were selected for further analysis. These two methods are described and compared with each other and with gradient descent and simulated annealing results. A detailed description of each algorithm is given by Yang et al [44].

6.1 Cuckoo Search

Cuckoo Search is a nature-inspired algorithm. It is based on the nesting behaviors of wild cuckoo birds. Cuckoo birds do not build their own nests. Instead, they rely on camouflaging their eggs to look like those of a host bird species [45]. If the eggs do not mimic the eggs of the host bird well enough, then they are removed by the host. If the eggs do match and eventually hatch, the baby cuckoo bird pushes out the remaining eggs of the host. The nesting behaviors of the cuckoo bird can be modeled using random walks based on Lévy flights [22]. Lévy flights are a type of highly-skewed normal distribution that tends to produce random walks close to the previous point in the domain.

The Cuckoo Search algorithm works by taking an initial population of nests and determining their fitness. For a specified number of iterations, a test flight is made from a random nest using a Lévy flight, and a cuckoo egg is chosen and given a fitness. Another random nest is chosen. If the fitness of the test nest is better than that of the randomly chosen nest, the egg (value) of that nest is replaced with the one of the test nest. Also, for each iteration, a specified percentage of the worst nests are abandoned, and Lévy flights are used to generate new nests. Ideally, as the population evolves, hosts get better at finding bad eggs, and cuckoo birds get better at concealing eggs. In this case, the measurement of whether or not an egg is good or bad is based on the fitness function. Table 6.1 shows how a naïve approach to this might work.

Given that new nests are chosen using random numbers that can land anywhere within the sampling space, the Cuckoo Search is guaranteed to converge to an optimal solution eventually. However, eventual convergence does not necessarily mean fast convergence. Yang discusses a method for Cuckoo Search that increases the speed at which the Cuckoo Search method will converge to a solution. This method starts out the same way a regular Cuckoo Search does by generating an initial set of nests and fitness values. The first change occurs when choosing a step size for Lévy flights. Rather than being constant, the step size decreases over time as the Cuckoo Search algorithm starts to converge to a global optimum. This removes some of the randomness associated with searching the design space later in the process. The next change to the search algorithm is more significant. Instead of choosing a single nest each iteration from which to potentially remove eggs, a specified percentage of the best nests are chosen for evaluation. For each nest in this population,

Table 6.1

Naïve Cuckoo Search

```
nests[population_size, dimension] := random(population_size, dimensions)
fitness[population_size] := fitnessFunction(population)
sorted_arg := argSort(fitness)
nests := nests[sorted_arg]
fitness := fitness[sorted_arg]
num_abandoned := p_a*population_size
for i in range(number_iterations):
    test_arg := intRandom(0, population_size)
    test := nests[test_arg]
    step := α*levyFlight(λ, dimensions)
    test += step
    checkBounds(test)
    test_fitness := fitnessFunction(test)
    random_arg := intRandom(0, population_size)
    if test_fitness < fitness[random_arg]:
        nests[random_arg] := test
        fitness[random_arg] := test_fitness
    for worst in bottom(nests, num_abandoned):
        step := α*levyFlight(λ, dimensions)
        worst += step
        checkBounds(worst)
        fitness[arg_worst] := fitnessFunction(worst)
    sorted_arg := argSort(fitness)
    nests := nests[sorted_arg]
    fitness := fitness[sorted_arg]
```

another potential nest is chosen from the best members of the population, potentially including itself. If the nest is to be compared with itself, then the process follows that of the normal Cuckoo Search algorithm. Otherwise, the distance from the two nests is calculated, and a test nest is generated by moving this distance from the worst nest to the best nest. Then, a random nest is chosen from the entire population. If the fitness of the test egg is better than this one, then the egg of the random nest is replaced by the test egg. This ensures a bias towards the best nests as the system evolves. This modified algorithm is shown in Table 6.2.

Table 6.2

Modified Cuckoo Search

```
nests[population_size, dimension] := random(population_size, dimensions)
fitness[population_size] := fitnessFunction(population)
sorted_arg := argSort(fitness)
nests := nests[sorted_arg]
fitness := fitness[sorted_arg]
num_abandoned := p_a*population_size
num_not_abandoned := population_size - num_abandoned
generation := 1 for i in range(number_iterations):
    for best in top(nests, num_not_abandoned):
        test_arg := intRandom(0, num_not_abandoned)
        test := nests[test_arg]
        if test is nests[arg_best]:
            α := levy/generation^2
            step := α*levyFlight(λ, dimensions)
            test += step
            checkBounds(test)
        else:
            scalar dx := ||nests[j]-test||_2 /φ
            scalar dbw := ||nests[0]-nests[-1]||_2
            vector vbw := (nests[0]-nests[-1])/dbw
            test := nests[-1]+dx+vbw
        test_fitness := fitnessFunction(test)
        random_arg := intRandom(0, population_size)
        if test_fitness < fitness[random_arg]:
            nests[random_arg] := test
            fitness[random_arg] := test_fitness
    for worst in bottom(nests, num_abandoned):
        α := levy/√generation
        step := α*levyFlight(λ, dimensions)
        worst += step
        checkBounds(worst)
        fitness[arg_worst] := fitnessFunction(worst)
    sorted_arg := argSort(fitness)
    nests := nests[sorted_arg]
    fitness := fitness[sorted_arg]
    ++generation
```

6.2 Particle Swarm

Particle Swarm is based on swarm behaviors seen in birds and schools of fish [20]. In this algorithm, a set of particles is initialized with fitness. Each particle starts out with a velocity of zero, and a global best fitness is determined by finding the best fitness value for the entire swarm. Given a specified number of iterations, the velocity of each particle is updated as shown in Equation (6.1).

$$\mathbf{v}_j^{n+1} = \gamma \mathbf{v}_j^n + \alpha \eta_g (\mathbf{g}^* - \mathbf{x}_j^n) + \beta \eta_l (\mathbf{x}_j^* - \mathbf{x}_j^n) \tag{6.1}$$

$$\mathbf{x}_j^{n+1} = \mathbf{x}_j^n + \mathbf{v}_j^{n+1} \tag{6.2}$$

In this equation, $\mathbf{x}_j^*$ is the best position for a particle, and $\mathbf{g}^*$ is the best overall position of the swarm. Therefore, α is the acceleration term for the global best solution, β is the acceleration term for the local best solution, and γ is the damping constant for the solution as a whole. Furthermore, the new position of the particle for the next iteration is calculated by adding the new velocity to the old position as shown in Equation (6.2). The full algorithm for Particle Swarm is shown in Table 6.3.

Table 6.3

Particle Swarm

```
swarm[population_size, dimension] := random(population_size, dimensions)
fitness[population_size] := fitnessFunction(swarm)
best[population_size, dimension] := swarm
best_fitness[population_size] := fitness
global_best_arg := argMin(best_fitness)
global_best[dimension] := best[global_best_arg]
global_best_fitness := best_fitness[globatl_best_arg]
velocity[population_size, dimension] := 0
for i in range(number_iterations):
    velocity := γ*velocity + α*random( population_size, dimensions) * (best - swarm)
        + β*random(population_size, dimensions) *(global_best - swarm)
    swarm += velocity
    for particle in(swarm):
        checkBounds(particle)
        fitness[arg_particle] := fitnessFunction(particle)
        if fitness[arg_particle] < best_fitness[arg_particle]:
            best[arg_particle] := particle
            best_fitness[arg_particle] := fitness[arg_particle]
            if fitness[arg_particle] < global_best_fitness:
                global_best := particle
                global_best_fitness := fitness[arg_particle]
```

6.3 Initial Tests in Optimization

For initial testing of optimization functions, the Michalewicz function [11] was used. This function was chosen due to it's similarity with the function that was used to optimize drag for the CST test case that is discussed later in this paper. To restrict lift and airfoil thickness and ensure only locations in the domain with lower drag than the base airfoil were evaluated, the Kriging function for drag was flattened to a specific maximum value wherever constraints were not met. This gives the drag function a similar shape to that of the Michalewicz function. The Michalewicz function is generated using Equation (6.3).

$$f(\mathbf{x}) = -\sum_{k=1}^{d} sin(x_i) sin^{2m}\left(\frac{kx_k^2}{\pi}\right) \tag{6.3}$$

For the two-dimensional form of the equation with all $x_i \in [0, \pi]$, the minimum value and location should be $f(\mathbf{x}^*) = -1.8013$ at $\mathbf{x}^* = [2.20, 1.57]$. Each optimization function was run 10 times with identical conditions. This was done to check the average error and variance of each optimization method. Each optimization scheme used a population size of 100 specimens.

The first optimization method that was investigated was gradient descent (GD). Figure 6.1 shows the finished run-state of the GD algorithm. The coloration of the figure depicts solutions of the Michalewicz function for x and y values between zero and five. The points in the figure show the final location of each particle used by the GD algorithm. Although GD only takes 5-20 iterations to reach a converged state, a true optimal solution is rarely found for this function. This is likely due to the overall flat shape of the Michalewicz function. Since movement by the GD algorithm is dependent on the existence of gradients,

this algorithm assumes an optimal value has been found whenever a particle enters a flat region of the domain. Only points that start at a hill or valley will actually search for a minimum value. Table 6.4 shows the spread in optimal values for the GD algorithm.

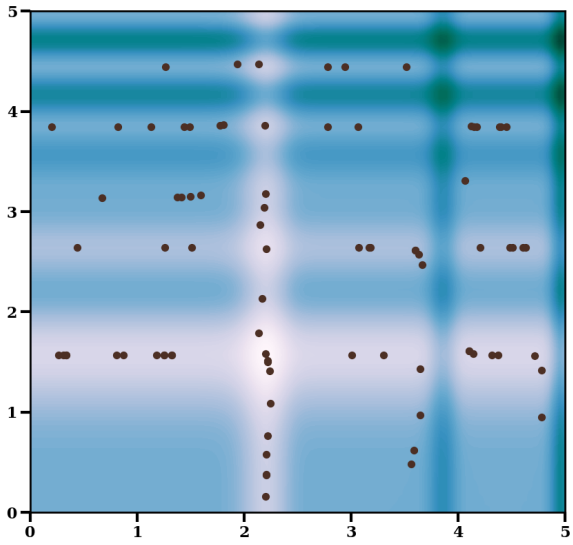

Figure 6.1

Finished Run State of Gradient Descent Optimization

Though not a true gradient-based algorithm, simulated annealing was analyzed to see if it might be able to reduce some of the problems associated with a true GD. While simulated annealing encourages movement of particles in the direction of the local gradient, movement is still random. This is most true at the begining of optimization when the particles have the most energy. This allows particles to escape flat regions; however, there is a trade-off in the number of iterations needed for a converged solution. The samples shown

Table 6.4

Gradient Descent List of Runs

Iteration	Specimen		Fitness
0	2.2019244413328590	1.5642766495772036	-1.8010972136649794
1	2.2249698284574344	1.4865655183898694	-1.7632643886483912
2	2.1959620606909440	1.5823914289075358	-1.7999137192577153
3	2.1866081471113086	1.6096794129433267	-1.7900916544109153
4	2.1912703547381716	1.5962361849900740	-1.7961659124732472
5	2.2135974546733660	1.5267385189669138	-1.7909466521564563
6	2.2441105369631110	1.4040339000038693	-1.6646783474805553
7	2.2039980707119655	1.5578240861759165	-1.8005324982734807
8	2.2285604565325543	1.4728612944327957	-1.7503552713689965
9	2.2021321271360605	1.5636341359271604	-1.8010638093769833

in Figure 6.2 took 100 iterations to reach the state shown. Because simulated annealing behaves like a gradient descent algorithm, once the energy in the particles gets low enough, it is easy to modify the initial temperature and rate of cooling to affect the number of iterations needed for convergence. Because of this, as seen in the figure, more particles are able find globally optimal results than they did for GD. In tests, the initial temperature was set to 2.8, and the rate of cooling was set to 0.48. Table 6.5 shows the fitness values for 10 runs of this algorithm.

Cuckoo Search is unique in that a percentage of all particles will perform random searches in the domain of the test function indefinitely. This means that if allowed to run long enough, a global optimal point will always be found. As has already been discussed in the previous chapter, the modified form of the Cuckoo Search forces a certain percentage of nests to be abandoned each iteration of the search. For the trial, 40 percent of nests were abandoned and forced to take a Lévy flight each iteration. The α term, which controls the

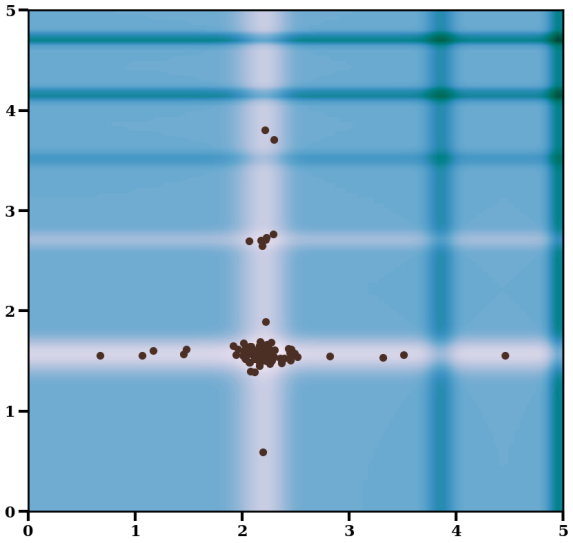

Figure 6.2

Finished Run State of Simulated Annealing

Table 6.5

Simulated Annealing List of Runs

Iteration	Specimen		Fitness
0	1.9821061137143214	1.4876876061473050	-1.7963391705533800
1	2.2062796909083913	1.5732073106202957	-1.8008823698695111
2	2.2206237476108344	1.5760338819046424	-1.7950635292213886
3	2.1883916309466550	1.5652925240832696	-1.7966864286751938
4	2.1766241071087666	1.5727358993711196	-1.7901269963508093
5	2.1194370726620420	1.5979340070296360	-1.7969631783008033
6	2.2013012240152143	1.5821845475030240	-1.7959849364968394
7	2.1920575837419847	1.5855298989208650	-1.7905667004448655
8	2.2024504545042520	1.5780610260391459	-1.7991550769597144
9	2.1834595029898436	1.5446408096434379	-1.7683352086635802

rate at which the distance of each Lévy flight for each iteration decreases over time, was set to 0.1. The λ term, which controls the average distance of a Lévy flight before the α term is applied, was set to 1.5. Figure 6.3 shows how this search algorithm will partially converge in one area and allow for continuous flights among the weakest members of the group. The best fitness from each trial is shown in Table 6.6.

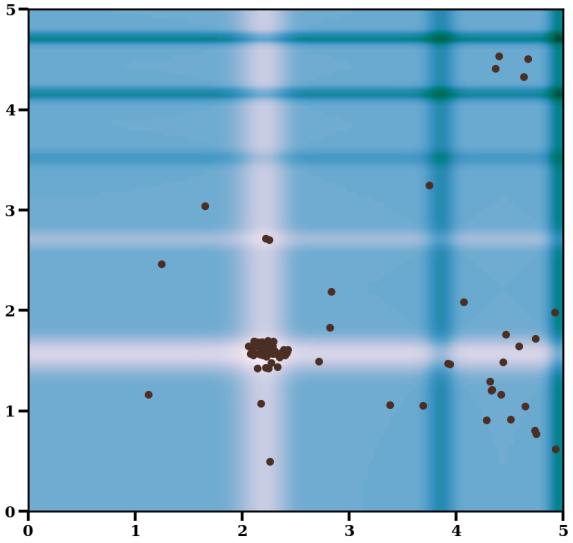

Figure 6.3

Finished Run State of Cuckoo Search

Particle Swarm, as a genetic algorithm, has behavior very similar to that of GD. The difference between the two is that GD looks for improvement based on knowlege of its environemnt, while Particle Swarm looks for improvement based on knowledge of the entire domain. This is seen in Figure 6.4, which shows the evolution of the swarm. Much

Table 6.6

Cuckoo Search List of Runs

Iteration	Specimen		Fitness
0	2.1983492906831190	1.5784904794919150	-1.7985603440705410
1	2.2036324623490975	1.5719078356151620	-1.8012447491414738
2	2.1966334312889386	1.5807427907116620	-1.7966420320479144
3	2.2238331151324178	1.5726771939364340	-1.7940079190942577
4	2.2168060292902454	1.5556010249493355	-1.7889291336779447
5	2.2126516931401470	1.5685095234004764	-1.7995430165069473
6	2.2018928383526974	1.5876624332608265	-1.7897087268516625
7	2.1912206523072930	1.5534124549093460	-1.7870634914163750
8	2.2173624424649345	1.5760447272187386	-1.7967734890469012
9	2.2265736208252225	1.5646002904905072	-1.7906082034215680

like flocks of birds and schools of fish, the particles in the Particle Swarm algorithm move together, displaying behavior not unlike that of a single organism. The particles lock onto the location of the particle with the best fitness. As the particles move towards that location, if another particle finds a better location, the other particles will lock onto the new location. If a particle starts moving towards locations with worse fitness values, it will slow down and search for locations in the domain with potentially better fitness values. With damping, in this, the swarm will cycle between the boundaries of the domain and a global optimal location, until it eventually reaches a converged state. Table 6.3 shows the best specimen for each trial run of the Particle Swarm search. In all trial runs, the global and local constants used to accelerate particles toward the best spot of the swarm and best spot of the particle were both set to 0.1.

Looking at Table 6.8, it is clear that the genetic algorithms perform much better than gradient descent and simulated annealing. Oddly, Particle Swarm seemed to perform much

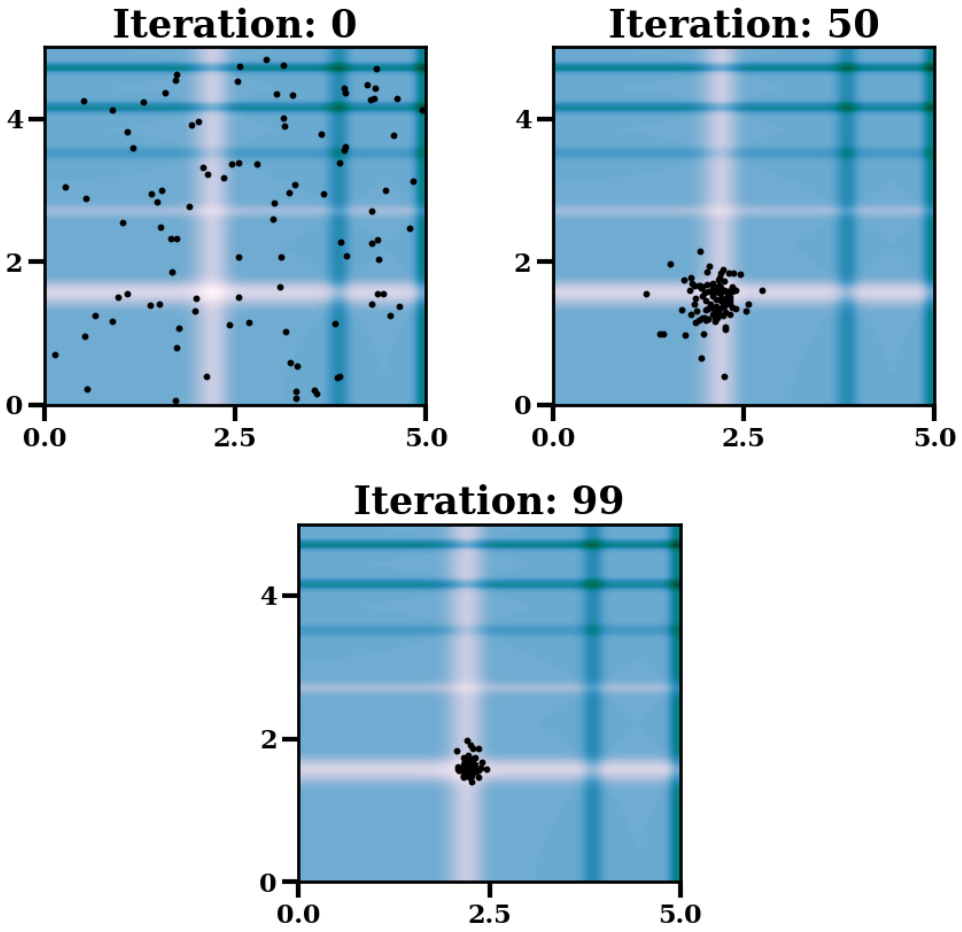

Figure 6.4

Evolution of Particle Swarm Search

Table 6.7

Particle Swarm List of Runs

Iteration	Specimen		Fitness
0	2.2028195679392537	1.5709485159678550	-1.8013023518987912
1	2.2029384064330824	1.5708241856525385	-1.8013033610875282
2	2.2034287710228093	1.5703415861298160	-1.8012905866841010
3	2.2027149442094727	1.5708742401058750	-1.8013025739916548
4	2.2030106251616792	1.5704831491535882	-1.8012992590847410
5	2.2026958060356523	1.5703928151557673	-1.8012961027821075
6	2.2030901703095314	1.5705028989319487	-1.8012993695061000
7	2.2030783252112810	1.5709519308437680	-1.8013019440206328
8	2.2033148768500443	1.5708764168445437	-1.8013004262884766
9	2.2031556734242757	1.5701873324469555	-1.8012873783995658

better than Cuckoo Search. This can likely be attributed to the fact that, while Cuckoo Search is very good at optimizing away weakness, it is not necessarily the best at building on strengths. This can be seen with how the algorithm abandons bad nests, but does not actively search for better solutions among better specimens.

Table 6.8

Optimization Algorithm Statistics

	Gradient Descent	**Simulated Annealing**	**Cuckoo Search**	**Particle Swarm**
μ	-1.77581095	-1.79301036	-1.79430811	-1.80129834
σ^2	28.38154067	28.93397535	28.97587436	29.20208124
MSE	2.29693e-03	1.46334e-03	7.10345e-05	2.91465e-11
RMSE	4.79263e-02	1.20969e-02	8.42810e-03	5.39875e-06

CHAPTER 7

TEST CASES FOR SURROGATE MODELING AND OPTIMIZATION

The final step of this research was to build a surrogate model for an airfoil design problem and test the two chosen optimization algorithms using the surrogate models to drive the optimization process. To build a working surrogate model, code had to be written to perform a series of tasks. First, the code had to be able to generate a set of data points appropriate for the surrogate model being used. For the Kriging model, these points could be chosen arbitrarily. For the Sparse Grids model, numerical evaluations were restricted to specific locations in the domain. The code also had to be able to parameterize airfoils in such a way that a small but unique set of data could be used to build, store, and test these airfoils. For this task, the CST method was used. Furthermore, the code had to be able to automate the process of generating computational meshes for analysis of different airfoil shapes. Finally, the surrogate model had to be able to extract lift and drag data from the output files to generate the surrogate models.

For the process of optimization, the drag function was minimized for a specific range of lifts. This was done primarily to prevent the algorithms from producing unnaturally thin or cambered airfoils. With both Kriging and Sparse Grid surrogates, a numerical-based gradient descent algorithm, simulated annealing, cuckoo search, and particle swarm search

were used. For the sake of comparison, all algorithms were limited to 250 samples and 100 iterations. This testing process is elaborated upon further in the chapter.

7.1 Applying CST for a Thin Subsonic Airfoil

A database of airfoils was generated using the CST parameterization method with the NACA 6406 airfoil as its base. This airfoil was chosen for its highly cambered shape. For the model that was generated, the goal was to minimize drag for a high-lift airfoil at low Mach numbers. Though surrogate models may be possible for transonic and supersonic flows, that was not the goal of this research. The NACA 6406 was chosen due to its high lift and low drag at subsonic speeds. To constrain the radius of the leading edge and thickness of the trailing edge, the first and last elements of the **p** vector of the shape function, shown in Equations (4.5), were set to 1.0. The upper surface used five CST control points with values that could range between 0.0 and 2.0. The middle three Bézier points were used as the first three variables in the Sparse Grids and Kriging models. The lower surface used a total of six CST control points. As with the upper surface, the first and last points were set to 1.0; furthermore, the third point on the lower surface was also set to 1.0. This was done to help prevent crossing between the upper and lower surfaces. The last three variables in the surrogate models used the second, fourth, and fifth CST parameters. For the Sparse Grid and Kriging models, these CST values were divided by two since both models make predictions using values between zero and one. An example random perturbation using this method is shown in Figure 7.1 corresponding with the CST values and normalized Sparse Grid values shown in Table 7.1. In the table, the middle three values in the Upper CST

Coordinates column map to the first three values in the Sparse Grids and Kriging Model Coordinates column. The second, fourth, and fifth values in the Lower CST Coordinates column map to the last three values of the Sparse Grids and Kriging Model Coordinates column.

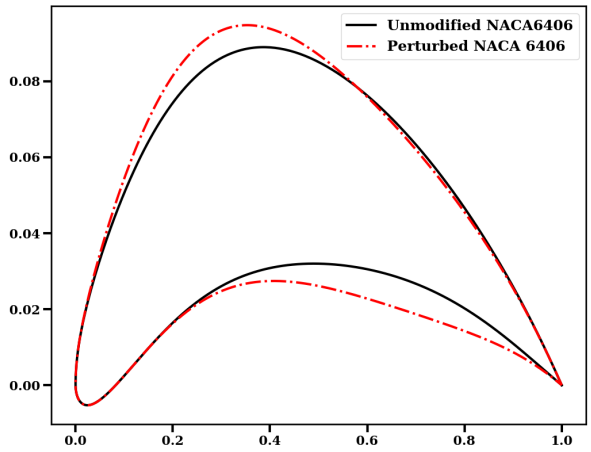

Figure 7.1

NACA 6406 with a Random Perturbation

For the first stage of tests, a level-three Sparse Grid was used with 545 different perturbations of the NACA 6406 airfoil. In addition to the parameterization Sparse Grids, a local polynomial boundary grid was used in order to make lift and drag predictions at different angles of attack. This Sparse Grid was one dimensional and used solutions from five different CST parameterization Sparse Grids for evaluation of its nodes. This brought the total number of CFD simulations to 2725. In order to test the accuracy of the surrogate models, for five angles of attack, a set of 250 random airfoils was tested with surrogate

Table 7.1

Coordinates for Modified NACA 6406

CST Coordinates		Sparse Grids / Kriging Model Coordinates
Upper	**Lower**	
1.000000	1.000000	0.636377
1.272754	1.021646	0.452402
0.904804	1.000000	0.480183
0.960366	0.681922	0.510823
1.000000	0.438000	0.340961
	1.000000	0.219000

model analysis and CFD evaluations. For 15 randomly selected airfoils from this list, simulations for 41 angles of attack were made to compare the accuracy of the models over a range of angles of attack and drag. The results from these tests are elaborated upon further in the next chapter.

In order to reduce the need for repetition, the Kriging surrogate model used many of the same routines that the Sparse Grid model used for tasks such as airfoil parameterization and the automation of CFD mesh generation. In order to get an accurate comparison of the convergence of both methods without taking into consideration differences in accuracy due to sampling, an initial test of the Kriging solver was performed on the same points that the Sparse Grid model used. To further show the differences in accuracy of the models due to sampling, a Kriging model that used a random LH sample was also used. The initial sample size for the LH model was 100 points for each of five angles of attack. Furthermore, four stages of refinement were used to take the total number of simulations to 200 per angle

of attack or 1000 simulations total. Figure 7.2 shows how the process of building the surrogate model, refining it, and performing simulations works.

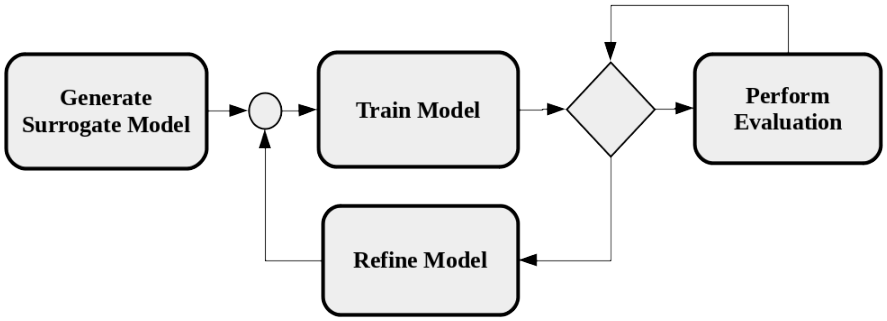

Figure 7.2

Surrogate Model Generation, Training, Refinement, and Evaluation

7.2 SU2 Configuration

To avoid having to create every configuration file, CFD mesh, and run script by hand, a series of templates was used to build these files for each CFD test. For the .cfg files, information about Mach number, angle of attack, and the name of the SU2 mesh file were left blank in the template. Using the boost file-system library and the ”getline” method in the C++ ”fstream” class, for each new test case that was built, the .cfg file was copied to a new directory for the test case, and each line of the file was evaluated for specific strings of text. These strings contained information about Mach number, angle of attack, and mesh file-name. Since these values were unique for each test case, the lines of text with this information were modified accordingly. Manipulation of the shell script used to submit

the job to the back end of the supercomputers at Mississippi State University and ERDC followed a similar procedure. Just as with the configuration file template, modifications were made to produce a unique run script for each test case. Unlike with the configuration file template, of which only one template was used, multiple templates were needed for the bash scripts, since project numbers, back end behavior, and MPI behavior were all different depending on which cluster was used.

To generate meshes for each airfoil, the geometry was first determined using CST. Construct2D was again used to generate a mesh for each airfoil and output a .xyz file. In order for SU2 to be able to read the mesh information, the .xyz file had to be converted to .su2 format. This procedure was done using a converter function, built from scratch in C++. The most unique aspect of this process was the fact that Construct2D generates structured meshes. SU2 on the other hand runs using unstructured grids. For a c-grid, the .xyz file generated by Construct2D assigns the value of each point along the wake of the airfoil to portions of the first vector of points in a structured computational grid twice. Therefore, when iterating through the .xyz file, special care had to be taken to ensure that these points were moved to the unstructured SU2 file once. Once the .su2 file was created, excess files used to generate the mesh were moved to /dev/null. Each mesh contained 100 points, extending perpendicular to the the airfoil and wake. Along the surface of each airfoil, 250 points were used. Finally, 75 points extended along the wake of the airfoil. Figure 7.3 shows a mesh generated using this procedure.

Each test was run as viscous and fully turbulent. The laminar viscosity model that was used was the Sutherland model. For turbulence, the Spalart-Allmaras model was used.

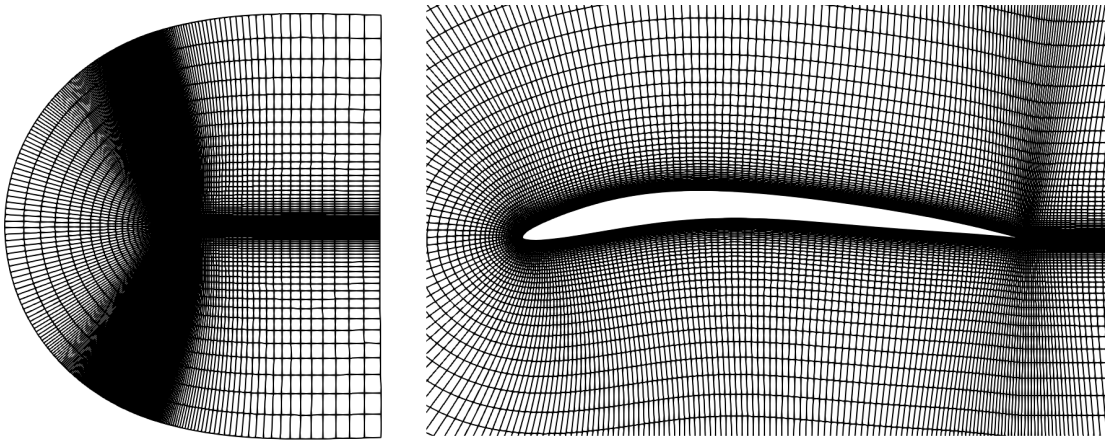

Figure 7.3

NACA 6406 Mesh Full Grid and Close Up

Though multi-grid was available, it was not used. The freestream pressure, freestream temperature, and Reynolds number for all airfoils were set to 101.325 kPa, 273.15 K, and 1.0E6, respectively, and the fluid used was air. Weighted least-squares was used for the gradient. The total number of maximum iterations was set to 25000. The SU2 CFD solver can finish early depending on if a certain set of user defined conditions are met. For the tests that were run, the SU2 solver was allowed to finish early if the drag coefficient stayed within one drag count for 250 iterations. The Flexible Generalized Minimal Residual method (FGMRES) was used as the linear solver. Lower-upper symmetric Gauus-Seidel (LU-SGS) was used for linear preconditioning. To increase speed of convergence, cfl number adaption was enabled. Roe's scheme was used to solve for the convective term of the Navier Stokes problem, and the Venkatakrishnan slop limiter was used.

7.3 Optimization Procedure

Optimization took place in two parts. First a set of five airfoils was chosen, and interpolations were made by linearly lofting between two airfoils at a time. At this stage, no extra constraints were added other than the maximization of the lift to drag ratio for a Mach number of 0.4 at all five angles of attack. Once a best case base airfoil was chosen using this method, a CST surrogate model was generated using Kriging. The goal at this point to make small perturbations to the chosen airfoil shape to minimize drag. To maintain similar thickness and camber, constraints were placed on the cross-sectional area of the airfoil and lift coefficient such that the area was constrained to 90 percent of the base airfoil area, and lift was constrained to 99 percent of the base airfoil lift. Because training takes so long, particularly for Kriging, optimization procedures ran all cases in a single run. The reason so many trials were made for each combination of lift and angle of attack was to get a good sense of the behavior of each optimization algorithm.

The optimization methods that were used were the Cuckoo Search and Particle Swarm. For Cuckoo Search, the probability of abandonment was set to 0.4. The initial α term was set to 0.1, and λ was set to 1.4. With particle swarm, both local and global acceleration rates were set to 0.1, and damping was set to 0.95. To verify results, CFD simulations were run at the locations for the average optimal drag for both optimization algorithms, as well as for the overall average optimal drag. To ensure that the optimization algorithm found minimum drag values, 25 CFD simulations with CST parameters within a p2-norm distance of 0.125, and 50 completely random parameterizations were compared with the

optimal airfoil to ensure no other airfoil with better lift and drag characteristics was found. The results for these tests are discussed in the next chapter.

CHAPTER 8

RESULTS

This chapter includes results from tests run for a Sparse Grid and Kriging surrogate model for perturbations of a thin, highly-cambered NACA 6406 airfoil. Using this thin airfoil shape ensured that all tests could be run as steady-state problems making comparisons between surrogate modeling approaches more consistent. For optimization tests, a technique called linear lofting [2] was used to produce airfoils interpolated between five different airfoils, and the best airfoil was chosen based on optimal lift to drag ratio. The following sections include analysis from these tests.

8.1 Surrogate Model Analysis Results

8.1.1 Sparse Grids Analysis

To test for accuracy in the Sparse Grids surrogate model, a random distribution of airfoils were generated, and CFD simulations were run for these airfoils. A naming convention was used that assigned a name to each airfoil based on it's location in an array of randomly chosen airfoils. For example, for the airfoil named "Airfoil_147" would correspond to the 147th airfoil generated in the database of airfoils. As shown in the initial testing discussed in Chapter five, Sparse Grids predictions tend to be most accurate along a line of points. Based on the distribution of points of a set of Sparse Grids, conventional

wisdon would indicate that the accuracy of Sparse Grids predictions would loose accuracy further from the center of the domain. To test this hypothesis, airfoils were ranked based on their location in the domain, with lower ranks being closer to the center and closer to an axis. CFD and Sparse Grids evaluations were performed for all airfoils. The airfoils were placed in groups of 50 based on their ranking, and mean squared error (MSE) and root mean squared error (RMSE) were calculated for lift and drag based on errors between the CFD and Sparse Grids evaluations. These results are shown in Figure 8.1. In addition to these evaluations, the overall accuracy was determined by finding the RMSE and MSE of the lifts and drags for all 250 airfoils. This is seen in Table 8.1.

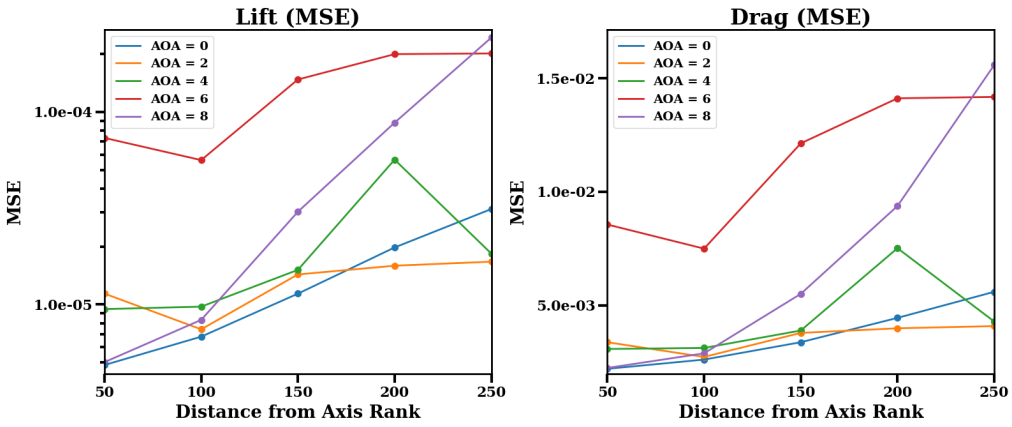

Figure 8.1

MSE using Sparse Grids

Looking at the MSE lift and drag charts in Figure 8.1, it would appear that errors increase based on distance from a three-dimensional axis plane. The only odd cases are at two and four degrees angle of attack, where errors appear to remain constant at two

Table 8.1

MSE and RMSE for Sparse Grids

Angle of Attack	Lift		Drag	
	MSE	RMSE	MSE	RMSE
0.0	1.482495e-05	3.850318e-03	1.190302e-07	3.450076e-04
2.0	1.314768e-05	3.625972e-03	2.331787e-07	4.828858e-04
4.0	2.183991e-05	4.673319e-03	1.698844e-07	4.121704e-04
6.0	1.353503e-04	1.163402e-02	1.809475e-06	1.345167e-03
8.0	7.484930e-05	8.651549e-03	7.810630e-06	2.794750e-03

degrees angle of attack, and appear to increase but suddenly spike at a distance rank of 200. In all likelihood, there is a handful of samples at these locations that produce unrealistic airfoils. If errors were constant, this would mean that predictions could just as accurately be made through interpolation using multiple three-dimensional surrogate models. This would be significant, since the number of sample points needed for accurate results typically increases exponentially with increasing dimension. However, using multiple low-dimensional surrogate models and combining the results using interpolation would significantly reduce the number of samples needed for accurate predictions.

In order for this surrogate modeling approach to be viable, the model should be accurate for a range of angles of attack. To test this, Sparse Grids evaluations were made for five angles of attack evenly distributed between 0.0 and 8.0. These evaluations were then used to generate a one-dimensional Sparse Grid for predicting lift and drag for a range of angles of attack given a specific airfoil shape. Figure 8.2 shows lift and drag curves for three different airfoils. The full series of lift and drag curves for these tests are shown in appendix A.3. The corresponding airfoils are shown in appendix A.2.

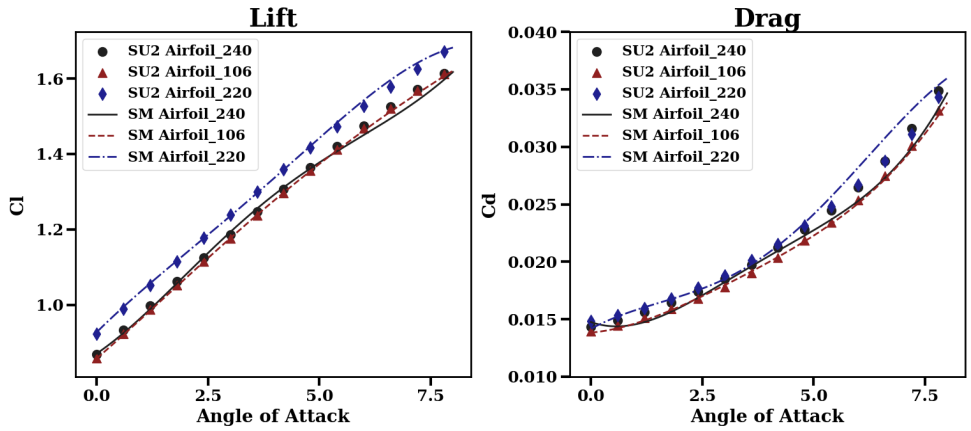

Figure 8.2

Lift and Drag Curves for B-spline Sparse Grids

As seen in Figure 8.2, the three airfoils chosen have very similar but slightly different drag curves. Though lift and drag are similar, an accurate surrogate model should still be able to reproduce each curve independently. This will be especially important once optimization methods are implemented. Looking at the figures, hierarchical B-splines are able to model the lift curve fairly accurately, but they still fail to pick out exact drag curves. Therefore, Sparse Grids hierarchical B-splines are not accurate enough to be used for sthe task of optimization with this distribution of points.

8.1.2 Kriging Using Sparse Grid Points

By using the same points for the initial Kriging model as for the Sparse Grid model, the full benefit of using the Kriging training model could be more accurately assessed. Evaluations were performed using the same random set of test cases as those used in the Sparse Grid analysis. As with the Sparse Grid evaluations, RMSE and MSE results were calcu-

lated based on distances to a three-dimensional axis as shown in Figure 8.3. Furthermore, global accuracy was also determined using all 250 airfoils. This is shown in Table 8.2.

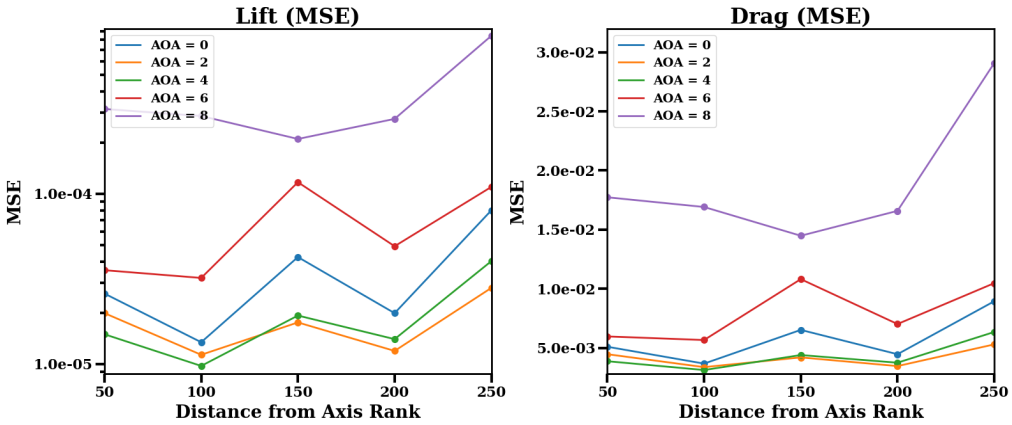

Figure 8.3

MSE using Kriging with Sparse Grid Points

By using the Kriging approach with a Sparse Grid distribution, the rate of change of errors with respect to distance from a three-dimensional axis appear to be more level. This is probably due to how Kriging uses training methods to reduce the likelihood of improvement on a global scale. While this does decrease global errors, it also increases local errors. Comparing Table 8.1 and Table 8.2, errors in lift appear to be higher using Kriging than using hierarchical B-splines. Still, the Kriging approach does not appear to increase errors until a much higher angle of attack is reached. For further analysis, comparisons can be drawn using a more even distribution of points. This approach will be discussed in further detail in the next section.

Table 8.2

MSE and RMSE for Kriging using Sparse Grid Points

Angle of Attack	Lift		Drag	
	MSE	RMSE	MSE	RMSE
0.0	3.630738e-05	6.025561e-03	9.531985e-08	3.087391e-04
2.0	1.773872e-05	4.211736e-03	4.916237e-08	2.217259e-04
4.0	1.962593e-05	4.430117e-03	1.050573e-07	3.241254e-04
6.0	6.863396e-05	8.284561e-03	5.913179e-07	7.689720e-04
8.0	3.861302e-04	1.965020e-02	1.313143e-06	1.145925e-03

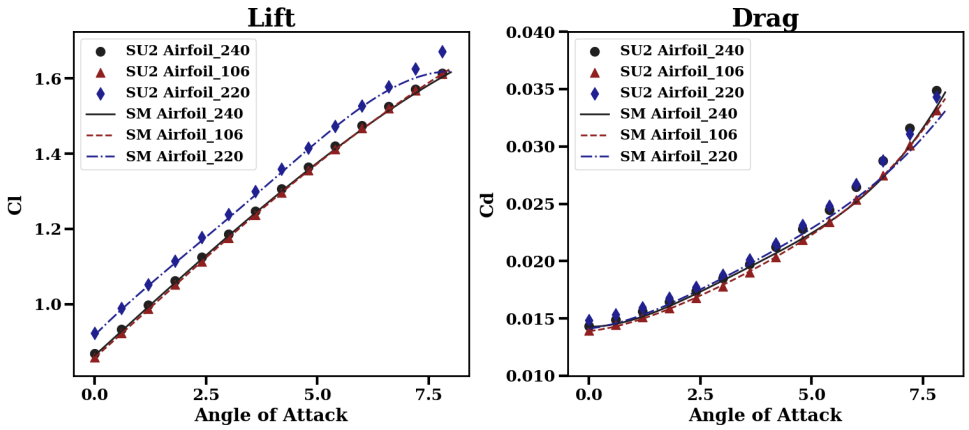

Figure 8.4

Lift and Drag Curves for Kriging Sparse Grids

Once again, overall MSE and RMSE increase with angle of attack. Lift and drag curves were also generated using Kriging. Three of these sets of curves are shown in Figure 8.4. The full series is shown in appendix A.4. At first glance, it would appear as if the Kriging implementation of Sparse Grids produces a better model than hierarchical B-splines; however, for the task of optimization, accuracy of model in relation to CFD tests is not as important. For the task of optimization, the fitness of airfoils is determined based on the relationship of lift and drag with other airfoils. Therefore, the emphasis should be placed on how surrogate models predict lift and drag of airfoils in relation to other airfoils. Looking at the drag curves for B-spline Sparse Grids, while these results are less accurate than the results of the Kriging Sparse Grids, B-splines do a better job of capturing the differences in lift and drag between different airfoil parameterizations.

8.1.3 Kriging Using Latin Hypercubes

Traditionally, Kriging models use a Latin Hypercube point distribution. This type of point distribution gives the surrogate model the benefit of being space filling, which should translate to more accurate surrogate model predictions. This approach was used to compare the accuracy of both the Latin hypercube and Sparse Grids point distributions. Since Latin Hypercubes use more evenly distributed points, errors should be less susceptible to location in the domain of the surrogate model. To show this difference, the same procedure for ranking airfoils and calculating MSE and RMSE used for the Sparse Grids evaluations was used for Kriging with Latin Hypercubes. These results are shown in Figure 8.5. Further-

more, accuracy throughout the full sampling space was determined by taking the RMSE and MSE for all 250 samples, as shown in Table 8.3.

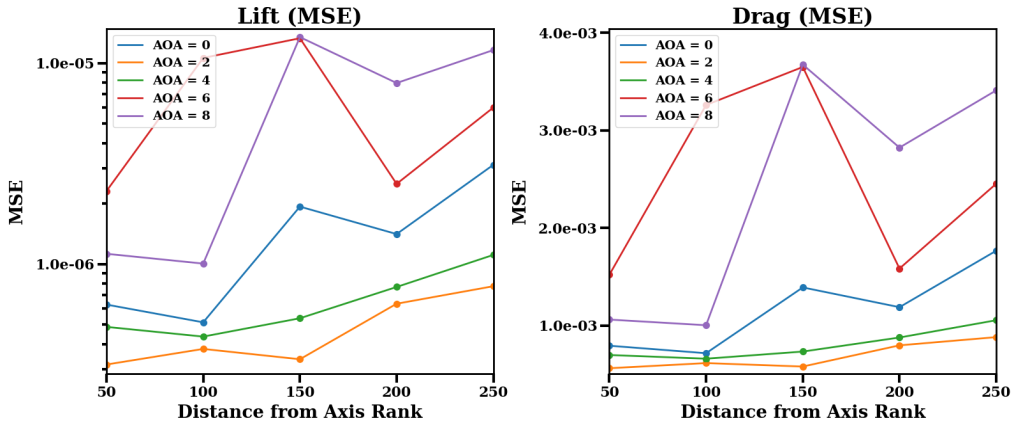

Figure 8.5

MSE using Kriging with Latin Hypercube Distribution

Looking at the MSE for lift and drag using Latin Hypercubes, the variation in errors appears to be less likely linked to location in the domain. Furthermore, the overall accuracy of this surrogate modeling approach appears to be much better than either approach using Sparse Grids.

Lift and drag curves were generated using the same procedures explained in the previous sections. These are shown in Figure 8.6 and appendix A.5. Looking at the Figure 8.6, Kriging using a Latin Hypercube distribution performs much better than either approach using Sparse Grids. Not only does the Kriging model the results more accurately, but it also captures the behavior much better as well. The only place where accuracy is lost is at eight degrees angle of attack. In all likelihood, these errors can be explained by the un-

Table 8.3

MSE and RMSE for Kriging Model with Latin Hypercubes

Angle of Attack	Lift		Drag	
	MSE	RMSE	MSE	RMSE
0.0	1.521151e-06	1.233349e-03	4.841478e-09	6.958073e-05
2.0	4.884366e-07	6.988824e-04	1.688204e-08	1.299309e-04
4.0	6.682356e-07	8.174568e-04	1.431547e-08	1.196473e-04
6.0	6.953940e-06	2.637032e-03	2.332175e-07	4.829260e-04
8.0	7.043135e-06	2.653891e-03	2.006328e-06	1.416449e-03

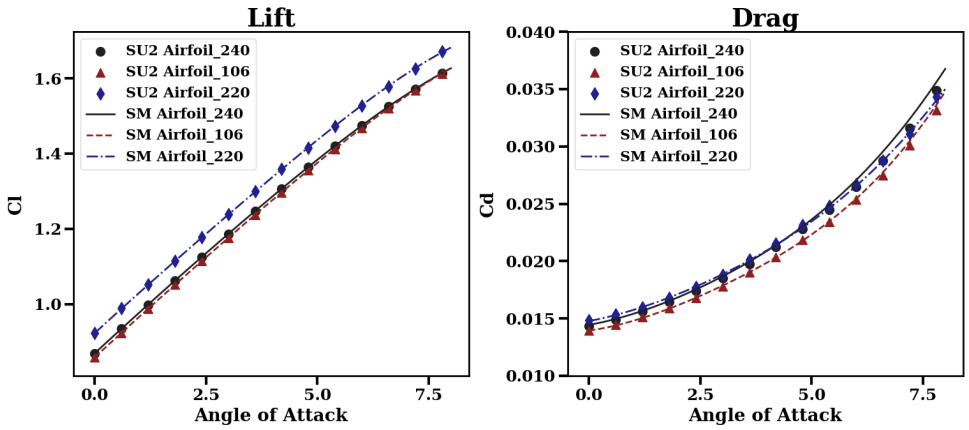

Figure 8.6

Lift and Drag Curves for Latin Hypercube Kriging

steadiness in the airflow associated with high angles of attack. At this point it was decided to use Kriging on a Latin Hypercube distribution for optimization.

Even without airflow separation, there are likely sources of slight unsteadiness in the CFD results. To determine if sources of errors in the Kriging model were due to actual errors or just due to variations in CFD simulations due to unsteadiness, four stages of infill were produced for the Kriging model. If errors were to remain constant, even with infill, this would likely mean that the errors were due to slight unsteadiness in the CFD results. A series of graphs were produced to show the progression of RMSE and MSE with infill for five angles of attack. These results are shown in Figure 8.7. Looking at these results, it does appear that applying infill techniques does not add much accuracy to the model. This indicates that most, if not all, sources of error in the surrogate model are not due to inaccuracy of the surrogate model itself.

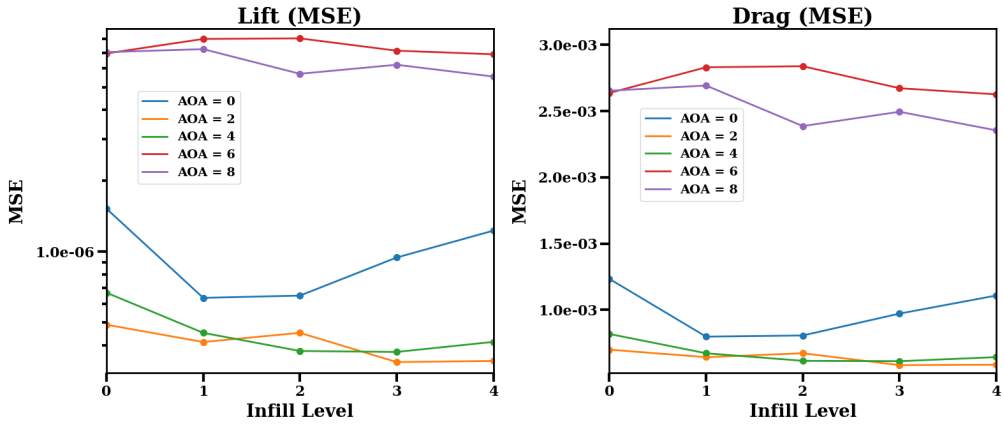

Figure 8.7

MSE after Infill

8.2 Optimization Results

For optimization testing, a 125-point Kriging model with no infill was used. It was determined that this would be accurate enough based on the previous surrogate model testing. The surrogate modeling implementation that has been used thus far was primarily designed to produce small perturbations to the airfoil shape. In order to limit the range of movement of the leading edge, trailing edge, and minimum thickness, the first and last CST parameters for the upper and lower surfaces as well as the third CST parameter for the lower surface are always set to a constant value of 1.0. For small perturbations in the base airfoil shape, realistic shapes are still produced. For very large changes, however, airfoil shapes become non-realistic. Therefore, to perform optimization effectively, a good initial airfoil was needed. To accomplish this task, a technique called linear lofting was used to interpolate lift and drag between two different airfoils. Five different airfoils were chosen for this task, as seen in Figure 8.9, with lofting being performed between pairs of each airfoil. This brought the number of linear interpolations to 10. Results for interpolations between the NACA 0009 and NACA 6412 and the NACA 6412 and SC(2)-0612 are shown in Figure 8.8.

Of the five airfoils, the one with the best lift to drag ratio at Mach 0.4 is the NACA 6412. While the high camber of the NACA 6409 does provide high lift, it also significantly increases the drag. The relatively thin and flat shapes of the Applying lofting technqiques to the NACA 6412 using the NACA 0009 and SC(2)-0612 airfoils produces a flatter shape on the lower surface of the airfoil. This helps decrease the drag without decreasing the lift

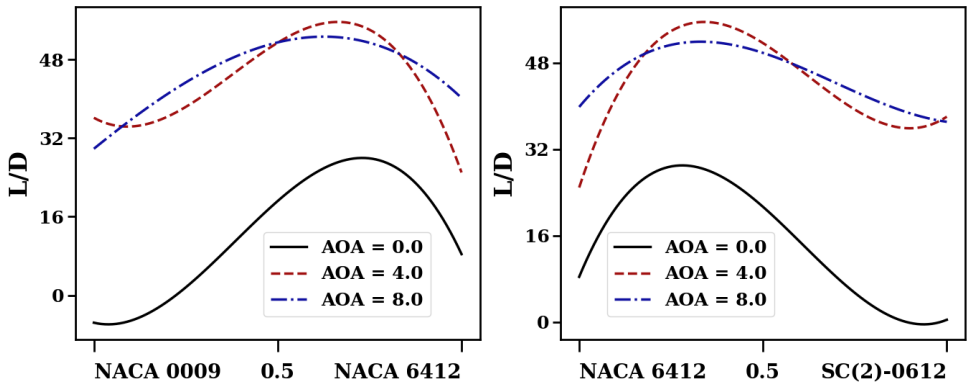

Figure 8.8

Interpolations for Lift and Drag using Linear Lofting

too much. This occurs because, at lower Mach numbers, curvature on the upper surface has more of an impact on lift.

After finding a good base airfoil shape, CST was used to model the shape of the base airfoil, and a Kriging model was generated using this airfoil. Before optimization was performed, limitations were placed on the shape of the optimal airfoil such that the lift coefficient would only vary by ± 1 percent, and the cross-sectional area of the airfoil would be no less than 90 percent of the area of the base airfoil. If it was determined that a test value did not meet these requirements, the drag coefficient was automatically set to 1.0. If the drag coefficient was greater than that of the base airfoil, the drag coefficient was automatically set to 1.0. By placing these restrictions, it was easier to gauge whether or not the surrogate model was able to optimize the airfoil shape. The following sections will discuss the optimization behavior of both Cuckoo search and particle swarm and will draw comparisons of results between both.

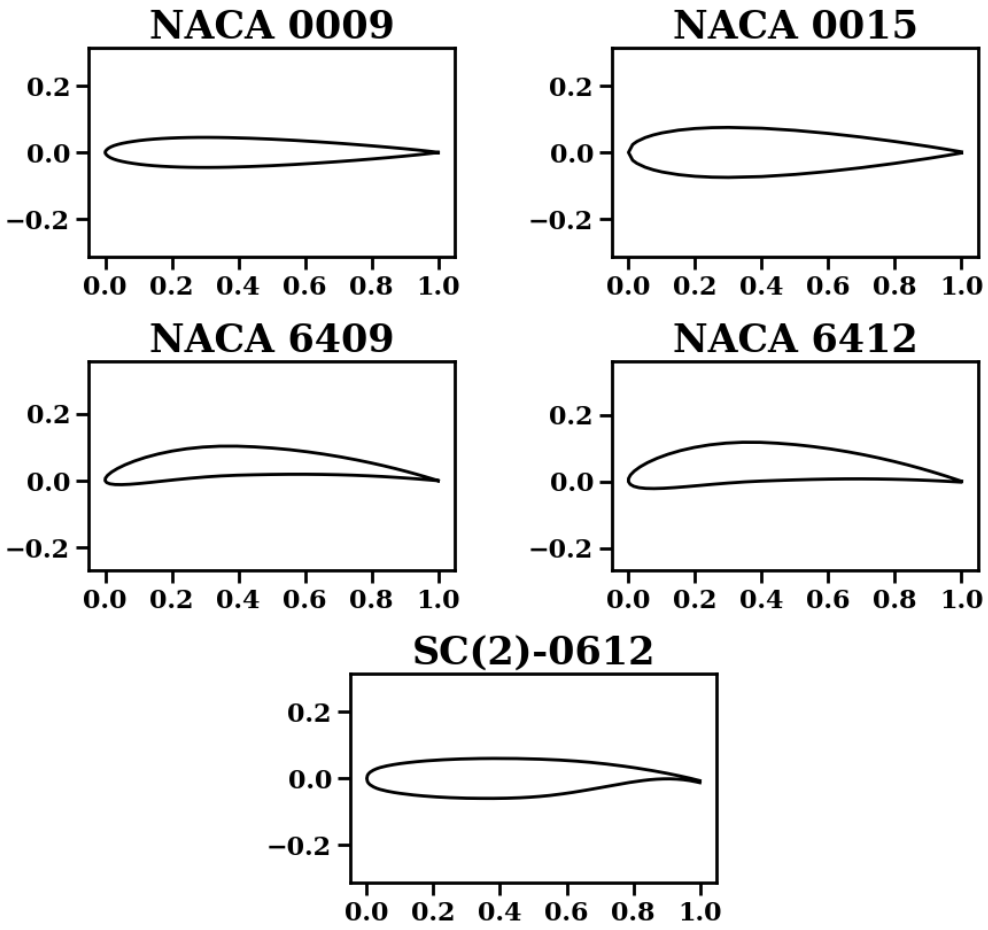

Figure 8.9

Set of Airfoils to Perform Lofting Technique

Out of 100 runs using optimization, the best optimal drag, the average optimal drag, and the percentage of successful runs were gathered. Looking at Table 8.4, it is apparent that for lower angles of attack, only two to four counts of drag are saved by performing optimization. However, with higher angles of attack, optimization of the airfoil shape can have a much bigger impact on drag. Figure 8.10 shows variations in the optimal airfoil shape with angle of attack using Cuckoo Search. As seen in the figure, for most angles of attack, a very similar shape is produced. The exception in this case is the optimal shape for eight degrees angle of attack. At this angle of attack, separation is more likely, and a shape like the one shown would probably be necessary to provide the greatest delay in separation. Furthermore, since there is greater potential for optimization at higher angles of attack, the success rate of discovering an optimal solution increases as well.

Table 8.4

Optimal Drag Data using Cuckoo Search

AOA	Success	Base Cd	Best Cd	Average Cd
0.0	84%	0.0121	0.0119	0.0120
2.0	80%	0.0132	0.0129	0.0130
4.0	96%	0.0153	0.0150	0.0151
6.0	99%	0.0182	0.0177	0.0180
8.0	100%	0.0251	0.0236	0.0244

Optimization using Particle Swarm generates similar results for the most part. As seen in Table 8.5, Particle Swarm was less successful at discovering optimal solutions at lower angles of attack. However, for the optimization attempts that were successful, Particle

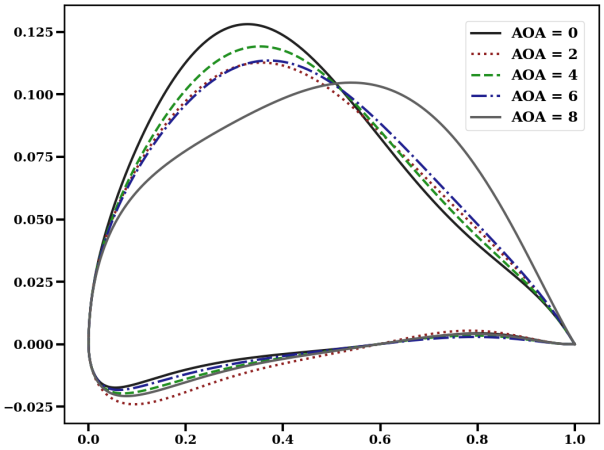

Figure 8.10

Optimal Airfoil Shapes Produced using Cuckoo Search

Swarm produced the same average optimal drags and same overall optimal drags. Furthermore, at higher angles of attack, Particle Swarm had more success than Cuckoo Search, shaving off two extra drag counts for the eight degree angle of attack case. What this shows is that, while Cuckoo Search is good at finding potential for optimization, it is not necessarily as good as Particle Swarm at zeroing in on an absolute optimal solution. This problem could likely be alleviated by using gradient descent at the end of the genetic optimization process. Just as with Cuckoo Search, similar optimal shapes are generated for angles of attack up to 6 degrees. At eight degrees, a completely different optimal shape is produced. With the similarities between optimal airfoil shapes for different angles of attack, a robust design that decreases drag across the full range of angles of attack can be generated. This approach will be discussed in the next section.

Table 8.5

Optimal Drag Data using Particle Swarm

AOA	Success	Base Cd	Best Cd	Average Cd
0.0	89%	0.0121	0.0119	0.0120
2.0	72%	0.0132	0.0129	0.0130
4.0	100%	0.0153	0.0150	0.0151
6.0	99%	0.0182	0.0177	0.0180
8.0	100%	0.0251	0.0234	0.0244

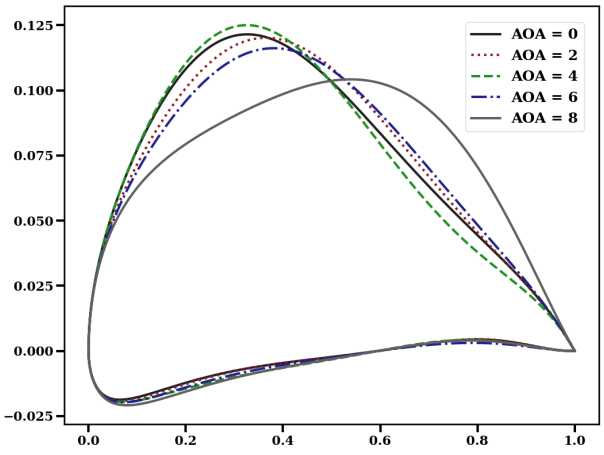

Figure 8.11

Optimal Airfoil Shapes Produced using Particle Swarm

Looking at Figure 8.12 and Figure 8.13, the optimal airfoil shapes for a range of angles of attack appear to be the airfoils optimized for an angle of attack range of zero to four degrees. Even at six degrees angle of attack, these airfoils appear to be on par with the airfoil optimized for that angle of attack. The only airfoil that performs better at higher angles of attack is the airfoil optimized for eight degrees. While there is a sizeable gap in drag performance at this range, this airfoil performs much worse at lower angles of attack, even performing worse than the base airfoil. This shows that minimal improvement for a specific design condition can translate to a much larger improvement for other design conditions. These results also show that optimizing for one specific angle of attack may produce a design that is not very useful for other angles of attack.

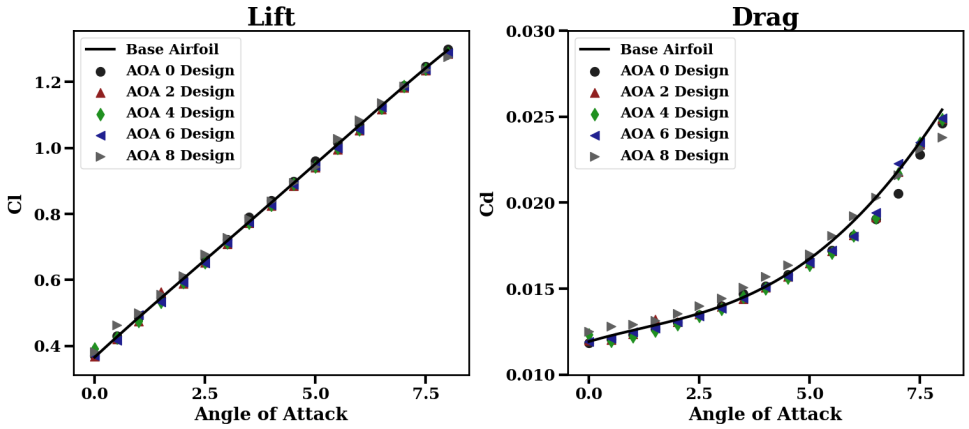

Figure 8.12

Airfoil Performance from Cuckoo Search Optimization

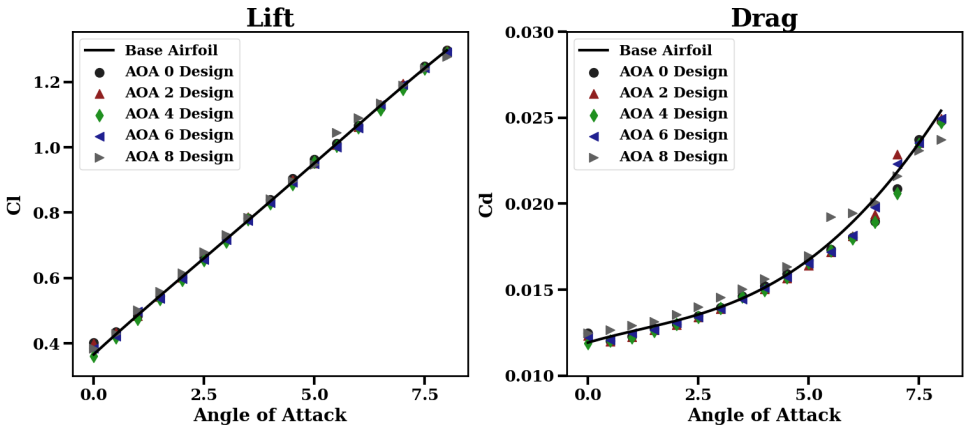

Figure 8.13

Airfoil Performance from Particle Swarm Optimization

CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

The research reported in this paper evaluated the accuracy and effectiveness of genetic optimization coupled with surrogate modeling. It was shown that sampling distributions can have a significant effect on the accuracy of the model. For example, Kriging using a Latin Hypercube point distribution was shown to produce results with accuracy an order of magnitude better than that for sparse grids. That is not to say that there is nothing to be taken from the results of Sparse Grid models. The main advantage of Sparse Grids is the way they use hierarchization to train surrogate models. One observation that was made was how changes in multiple variables in models can often be separated without having to account for their relationships with one another. For example, instead of building a single six-dimensional Kriging model for the task of predicting lifts and drag for variations in airfoil parameters, multiple lower dimensional Kriging models, with fewer points, could be used, and a separate set of Sparse Grids could be used to interpolate using combinations of the Kriging surrogate models.

Four initial optimization schemes were evaluated. Ultimately the decision was made to use non-gradient based methods for the task of airfoil optimization. Two unique nature-inspired algorithms called Cuckoo Search and Particle Swarm were used. It was shown that

while Cuckoo Search will continue to search for an optimal solution indefinitely, Particle Swarm can pinpoint a more precise optimal solution much faster. Due to the random nature of the algorithm, Cuckoo Search has difficulty zeroing in on a solution once a certain accuracy has been reached. For problems such as airfoil optimization, in which there is a certain degree of uncertainty anyway, this lack of pinpoint accuracy does not pose too great of an issue. In fact, some tests showed Cuckoo Search to optimize drag as effectively if not more so than Particle Swarm. For cases where uncertainty is low, and solutions are smooth, gradient methods could be used in conjunction with Cuckoo Search after a specified level of optimization certainty is reached. Tests run also showed that simpler methods such as linear lofting can be just as effective at optimizing drag. In the tests that were run using Kriging and genetic optimization, improvements greater than two or three drag counts were only achieved at higher angles of attack. With that in mind, optimizing to this level still has advantages in that airfoils can be generated that are robust and stable over a larger range of angles of attack.

Another approach that is gaining popularity in the field of computer science is machine learning using artificial neural networks. Artificial neural networks (ANN) [41] are inspired by the neural behavior of the human brain. Artificial neural networks are mathematical structures that choose an optimal condition via an output layer. This layer is dependent on information that is passed to a series of hidden layers that send signals towards the output layer using functions of the form $\mathbf{y} = W^{*}\mathbf{x} + \mathbf{b}$. Essentially, ANNs behave as a multilayered RBF. A diagram for this type of structure is shown in Figure 9.1.

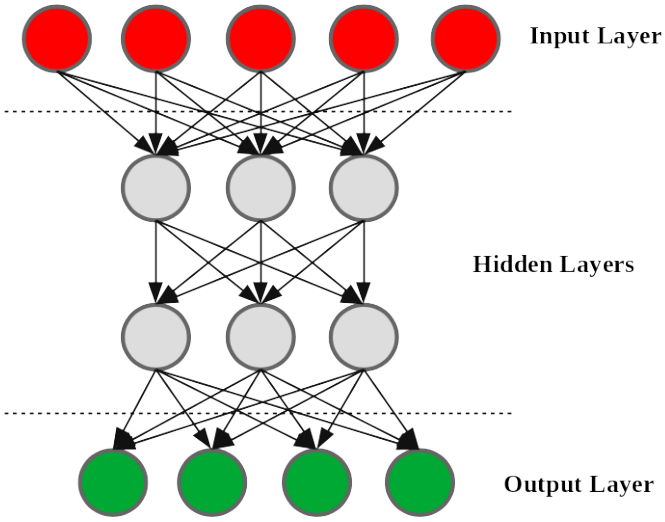

Figure 9.1

Artificial Neural Network

Similar to Kriging, ANNs improve through learning and adaptation to new information through a process called back-propagation [14]. Back-propagation is similar to learning in Kriging in that either error analysis or improvement likelihood is used to generate a model that is most likely to fit a set of data. Typically this process is carried out using some sort of optimization algorithm. Often times, this process is used with larger numbers of layers that branch out in complex ways not seen in a typical ANN. Neural networks like these are often called deep neural networks and encompass an area of machine learning called deep learning.

For future work in airfoil optimization, if an approach like the one used here were used, a better implementation might be to find an optimal set of chord and thickness modes using SVD initially. Then a more precise infill strategy such as PARSEC [39] could be used to perform final optimizations. One issue using CST was that a large percentage of

the surrogate modeling domain did not meet the necessary constraints that were imposed. Because of this, both Cuckoo Search and Particle Swarm had a difficult time finding an optimal airfoil shape. There is likely an airfoil shape that would perform much better than the one that found using CST. Smaller perturbations would likely make this airfoil easier to discover.

Summarizing, this thesis analyzed the use of surrogate models and genetic algorithms for the task of airfoil optimization. Kriging and hierarchical basis function methods were compared using using Sparse Grid and Latin Hypercube point distributions. The CST method was used to specify a mathematical way to describe and perturb an airfoil. Cuckoo search and particle swarm were used with these methods to perform optimization. Ultimately, it was determined that the combination of these methods are capable of handling the task airfoil optimization.

REFERENCES

[1] J. C. A. Barata and M. S. Hussein, "The Moore-Penrose Pseudoinverse: A Tutorial Review of the Theory," *Brazilian Journal of Physics*, Dec. 2011.

[2] S. A. Berger, W. C. Webster, R. A. Tapia, and D. A. Atkins, *Mathematical Ship Lofting*, vol. 10, 4 edition, The Society of Naval Architects and Marine Engineers, Dec. 1966.

[3] O. Berger-Tal, J. Nathan, E. Meron, and D. Saltz, "The Exploration-Exploitation Dilemma: A Multidisciplinary Framework," *PLoS ONE*, Apr. 2014.

[4] H.-J. Bungartz and M. Griebel, "Sparse Grids," *Acta Numerica 2004*.

[5] D. Butnaru, D. Pflüger, and H.-J. Bungartz, "Towards High-Dimensional Computational Steering of Precomputed Simulation Data using Sparse Grids," *Procedia Computer Science*.

[6] L. Childs, "Latin Squares," *Springer Reference*.

[7] J. H. S. de Baar and B. Harding, "A Gradient-Enhanced Sparse Grid Algorithm for Uncertainty Quantification," *International Journal for Uncertainty Quantification*, 2015.

[8] C. de Boor, "On Calculating with B-splines," *Journal of Approximation Theory*, July 1972.

[9] M. Drela, "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils," *Lecture Notes in Engineering Low Reynolds Number Aerodynamics*, 1989.

[10] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, "SU2: An Open-Source Suite for Multiphysics Simulation and Design," *AIAA Journal*, Mar. 2016.

[11] A. E. Eiben and C. H. M. van Kemenade, *Performance of Multi-Parent Crossover Operators on Numerical Function Optimization Problems*, Rijksuniversiteit Leiden Vakgroep Informatica, 1995.

[12] A. Forrester, A. Sóbester, and A. Keane, *Engineering Design via Surrogate Modelling: a Practical Guide*, 1 edition, J. Wiley, 2008.

[13] Z.-H. Han and S. Görtz, "Hierarchical Kriging Model for Variable-Fidelity Surrogate Modelling," *AIAA Journal*, Sept. 2012.

[14] M. E. Haque and K. V. Sudhakar, "ANN Back-Propagation Prediction Model for Fracture Toughness in Microalloy Steel," *International Journal of Fatigue*, Sept. 2002.

[15] P. M. Hartwich and S. Agrawal, "Orthonormal Functions for Airfoil and Wing Parametrization," *14th Applied Aerodynamics Conference*, June 1996.

[16] https://airfoiltools.com/search/, "Airfoil Tools,".

[17] https://m selig.ae.illinois.edu/ads/coord_database.html, "UIUC Airfoil Coordinates Database,".

[18] https://sourceforge.net/projects/construct2d/, "Construct2D,".

[19] R. M. Huggins, "Some Practical Aspects of a Conditional Likelihood Approach to Capture Experiments," *Biometrics*, June 1991.

[20] M. Imran, R. Hashim, and N. E. A. Khalid, "An Overview of Particle Swarm Optimization Variants," *Procedia Engineering*, 2013.

[21] S. Jeong, M. Murayama, and K. Yamamoto, "Efficient Optimization Design Method using Kriging Model," *42nd AIAA Aerospace Sciences Meeting and Exhibit*, Jan. 2004.

[22] A. Kamaruzaman, A. Zain, S. Yusuf, and A. Udin, "Levy Flight Algorithm for Optimization Problems - A Literature Review," *Applied Mechanics and Materials*, vol. 421, Sept. 2013.

[23] M. S. Khurana, H. Winarto, and A. K. Sinha, "Application of Swarm Approach and Artificial Neural Networks for Airfoil Shape Optimization," *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Oct. 2008.

[24] A. Krzyżak and T. Linder, "Radial Basis Function Networks and Complexity Regularization in Function Learning," *IEEE Transactions on Neural Networks*, Mar. 1998.

[25] B. M. Kulfan, "Universal Parametric Geometry Representation Method," *Journal of Aircraft*, Jan. 2008.

[26] B. M. Kulfan and J. E. Bussoletti, ""Fundamental" Parameteric Geometry Representations for Aircraft Component Shapes," *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Sept. 2006.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, May 2015.

[28] J. Li, M. A. Bouhler, and J. R. R. A. Martins, "A Data-Based Approach for Fast Airfoil Analysis and Optimization," *2018 AIAA/ASCE/AHS/ASC Structure, Structural Dynamics, and Materials Conference*, Jan. 2018.

[29] E. A. Luke, X. Tong, and R. Chamberlain, "FlowPsi: An Ideal Gas Flow Solver - The User Guide," Oct. 2017.

[30] E. A. Luke, X.-L. Tong, J. Wu, L. Tang, and P. Cinnella, "CHEM: A Chemically Reacting Flow Solver for Generalized Grids," *Tetra Research*, 2003.

[31] D. A. Masters, N. J. Taylor, T. C. S. Rendall, C. B. Allen, and D. J. Poole, "A Geometric Comparison of Aerofoil Shape Parameterisation Methods," *54th AIAA Aerospace Sciences Meeting*, Jan. 2016.

[32] N. Michael and R. Weed, "Aplication of Sparse Grid Surrogate Models to Airfoil Analysis and Design," *AIAA Aviation 2019 Forum*, June 2019.

[33] R. Mukesh, K. Lingadurai, and U. Selvakumar, "Airfoil Shape Optimization Based on Surrogate Model," *Journal of the Institute of Engineeris (India): Series C*, Mar. 2017.

[34] E. Ostertagová, "Modelling using Polynomial Regression," *Procedia Engineering*, 2012.

[35] C. Paulson and G. Ragkousis, "PyKriging,", https://www.pykriging.com.

[36] D. Pflüger, "Spatially Adaptive Sparse Grids for High-Dimensional Data-Driven Problems," *Journal of Complexity*, Oct. 2010.

[37] D. Rajnarayan and A. Ning, "Universal Airfoil Parametrization Using B-Splines," *2018 Applied Aerodynamics Conference*, June 2018.

[38] P. Saracco and M. G. Pia, "An Exact Framework for Uncertainty Quantification in Monte Carlo," *Journal of Physics: Conference Series*, vol. 513, no. 2, June 2014, p. 022033.

[39] H. Sobieczky, "Parametric Airfoils and Wings," *Notes on Numerical Fluid Mechanics (NNFM) Recent Development of Aerodynamic Design Methodologies*, 1999.

[40] Stoyanov, K. Miroslav, Webster, and G. Clayton, "A Dynamically Adaptive Sparse Grids Method for Quasi-Optimal Interpolation," *Computers and Mathematics with Applications*, vol. 71, no. 11, 2016, pp. 2499 – 2465.

[41] G. Sun and S. Wang, "A Review of the Artificial Neural Network Surrogate Modeling in Aerodynamic Design," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, July 2019.

[42] R. van Liere, J. D. Mulder, and J. J. van Wijk, "Computational Steering," *Future Generation Computer Systems*, Apr. 1997.

[43] L. Yang and X.-M. Zeng, "Bézier Curves and Surfaces with Shape Parameters," *International Journal of Computer Mathematics*, July 2009.

[44] X.-S. Yang, *Nature-Inspired Optimization Algorithms*, 1 edition, Elsevier, 2014.

[45] X.-S. Yang and S. Deb, "Cuckoo Search via Lévy Flights," *2009 World Congress on Nature and Biologically Inspired Computing*, 2009.

[46] Zhong-Hua and K.-S. Zhang, "Surrogate-Based Optimization," *Real-World Applications of Genetic Algorithms*, Mar. 2012.

APPENDIX A

LIFT AND DRAG CURVES FOR CST SURROGATE MODELS

A.1 Airfoil CST Parameter Table

Table A.1

Indexed List of Random Airfoil Perturbations Using CST

Index	**CST Airfoil Paramters**					
0	0.698504	0.064033	0.968737	0.994107	0.806781	0.139014
10	0.802405	0.538481	0.417790	0.062000	0.950865	0.618717
13	0.521531	0.801413	0.654962	0.752989	0.148380	0.963948
18	0.494930	0.432103	0.626462	0.708959	0.043654	0.466078
50	0.751773	0.281032	0.410021	0.307737	0.502188	0.469927
60	0.173704	0.586616	0.699647	0.777812	0.452885	0.541216
64	0.617808	0.492265	0.778280	0.536736	0.240976	0.070053
70	0.602689	0.556261	0.198847	0.512412	0.775420	0.067290
73	0.857681	0.540913	0.161161	0.762899	0.620671	0.274720
76	0.726186	0.841175	0.710249	0.774939	0.479588	0.463408
87	0.847375	0.687653	0.537442	0.478205	0.580151	0.696835
88	0.517358	0.729535	0.268916	0.245243	0.206737	0.592972
106	0.556985	0.697213	0.852290	0.233876	0.924353	0.493251
124	0.350034	0.296354	0.657818	0.339646	0.090064	0.556556
135	0.437438	0.947204	0.913947	0.077919	0.542011	0.286779
154	0.636337	0.452402	0.480183	0.510823	0.340961	0.219000
156	0.736349	0.726302	0.308948	0.253565	0.739114	0.435494

Table A.1

(continued)

Index	CST Airfoil Paramters					
192	0.465260	0.692492	0.359895	0.517394	0.515670	0.391620
204	0.694253	0.943416	0.430389	0.240281	0.392179	0.816442
220	0.917288	0.429501	0.999404	0.293769	0.509991	0.730852
224	0.933869	0.180962	0.992064	0.491095	0.633669	0.163408
229	0.499721	0.905238	0.124911	0.040295	0.104221	0.681522
238	0.367698	0.385788	0.824617	0.406281	0.286854	0.155481
240	0.414597	0.911798	0.822011	0.529721	0.013354	0.998202
246	0.948564	0.766059	0.574563	0.713193	0.426732	0.733265

A.2 Airfoil Shapes

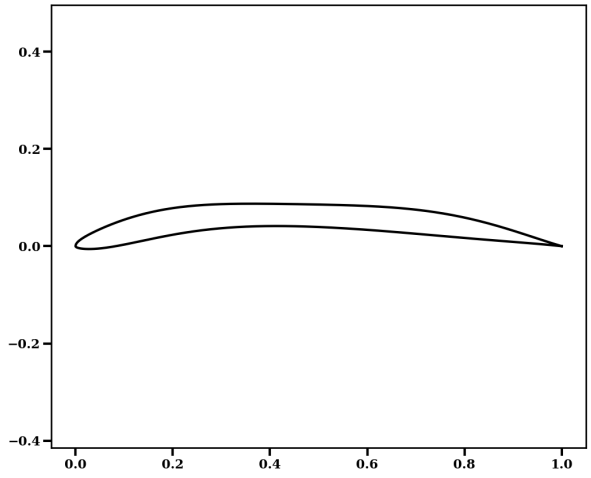

Figure A.1

CST Airfoil, Index: 0

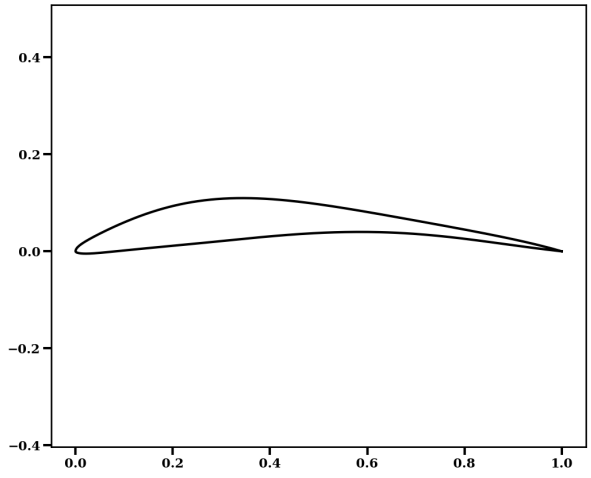

Figure A.2

CST Airfoil, Index: 10

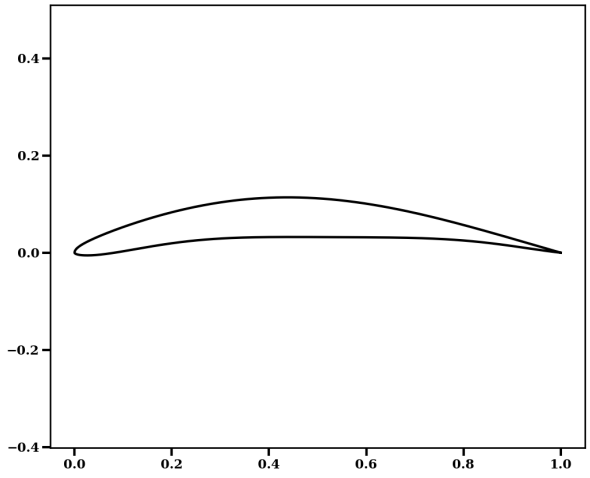

Figure A.3

CST Airfoil, Index: 13

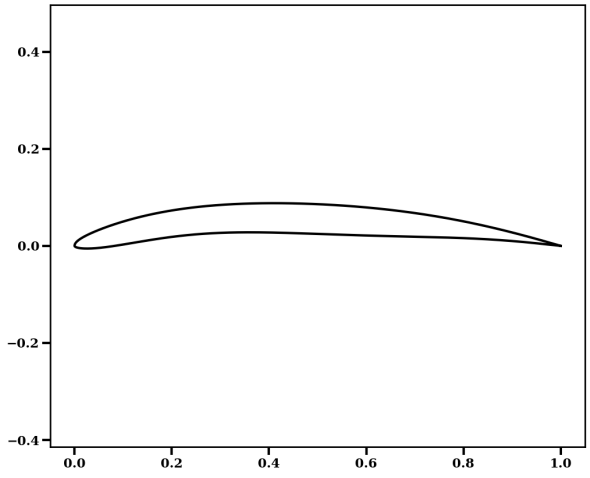

Figure A.4

CST Airfoil, Index: 18

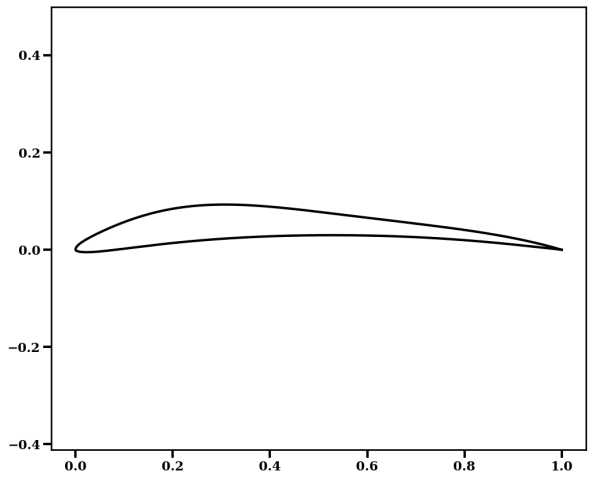

Figure A.5

CST Airfoil, Index: 50

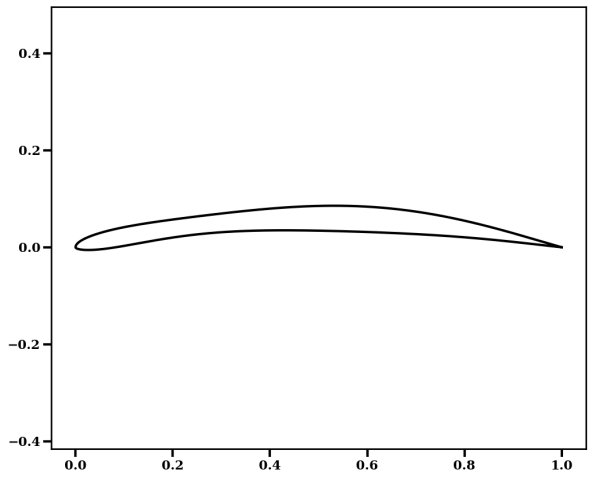

Figure A.6

CST Airfoil, Index: 60

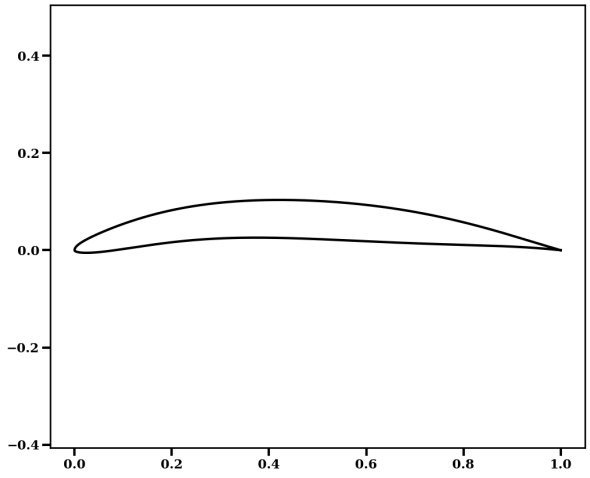

Figure A.7

CST Airfoil, Index: 64

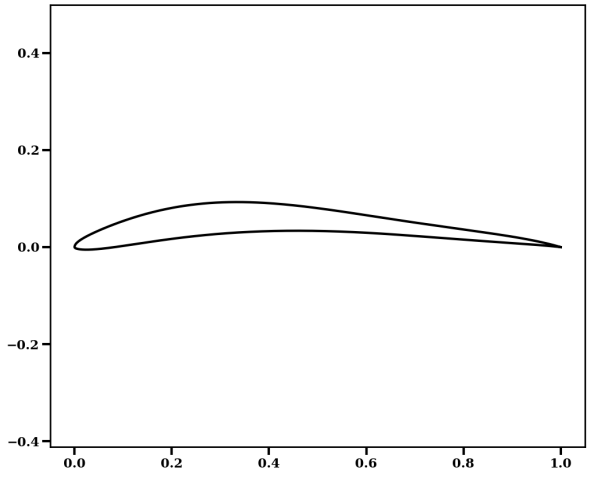

Figure A.8

CST Airfoil, Index: 70

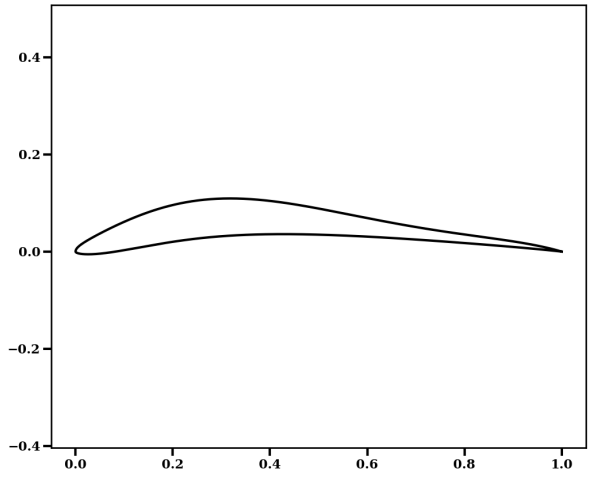

Figure A.9

CST Airfoil, Index: 73

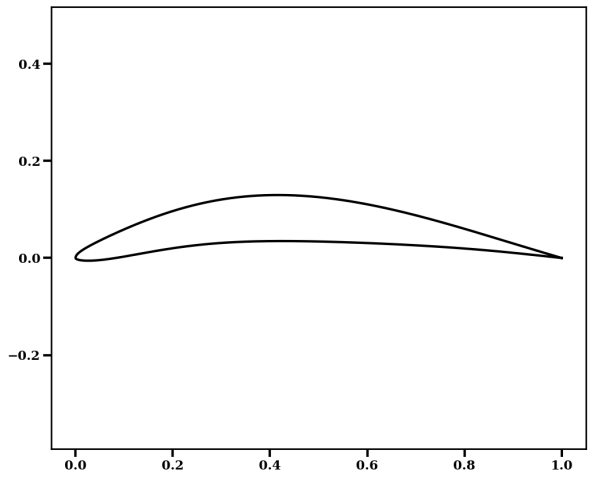

Figure A.10

CST Airfoil, Index: 76

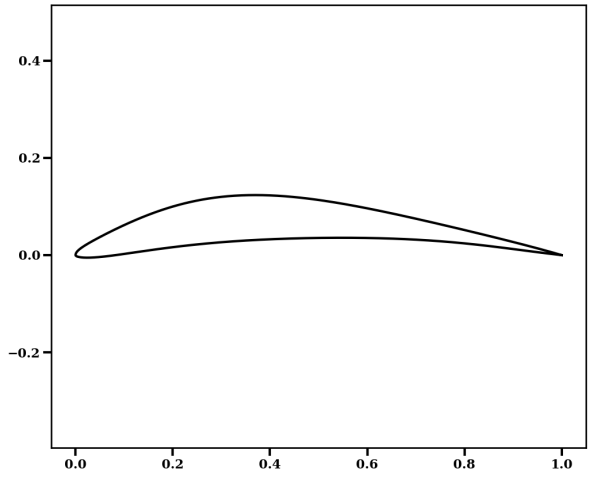

Figure A.11

CST Airfoil, Index: 87

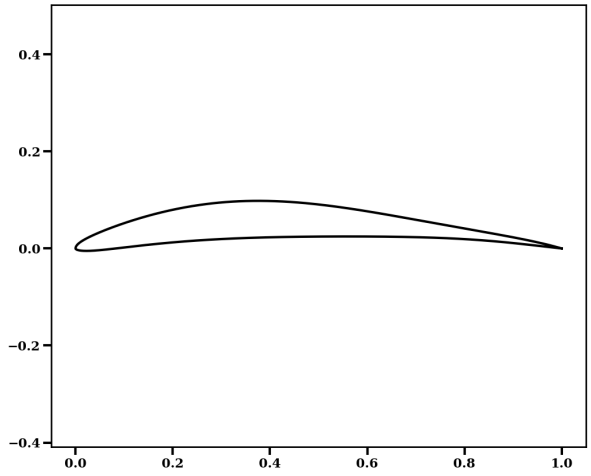

Figure A.12

CST Airfoil, Index: 88

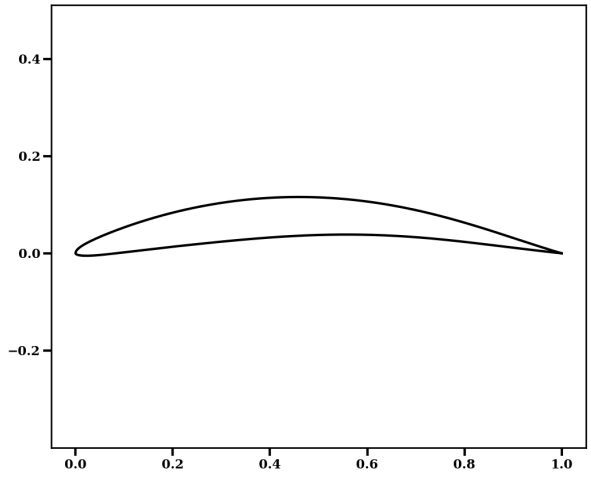

Figure A.13

CST Airfoil, Index: 106

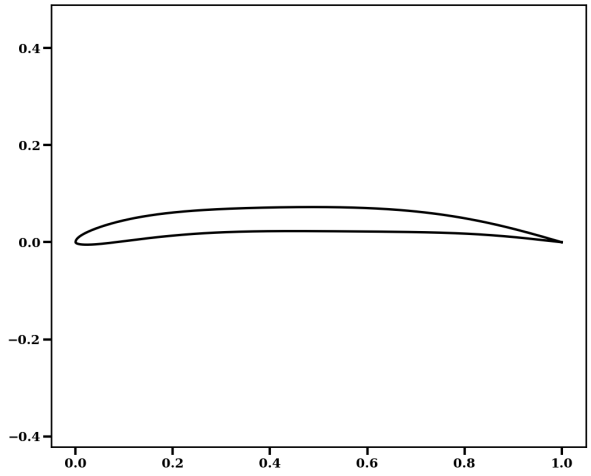

Figure A.14

CST Airfoil, Index: 124

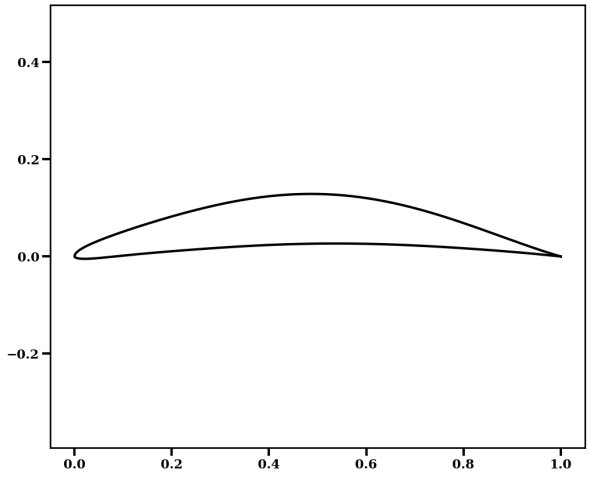

Figure A.15

CST Airfoil, Index: 135

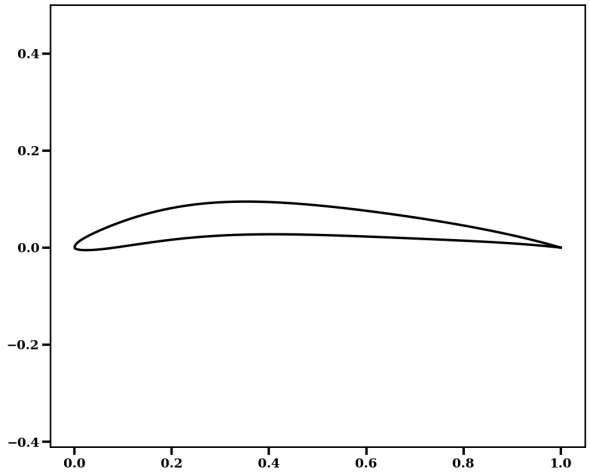

Figure A.16

CST Airfoil, Index: 154

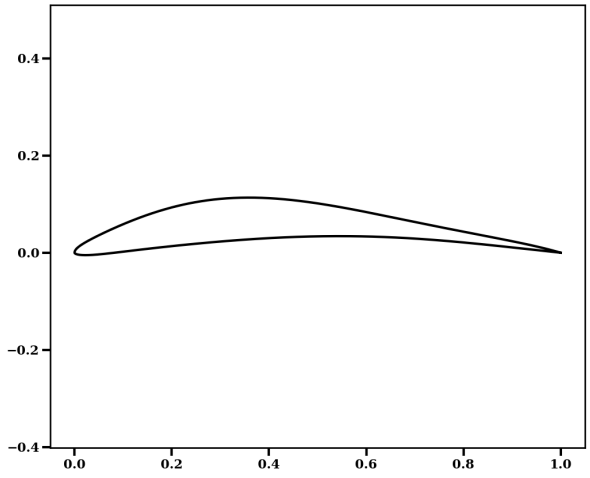

Figure A.17

CST Airfoil, Index: 156

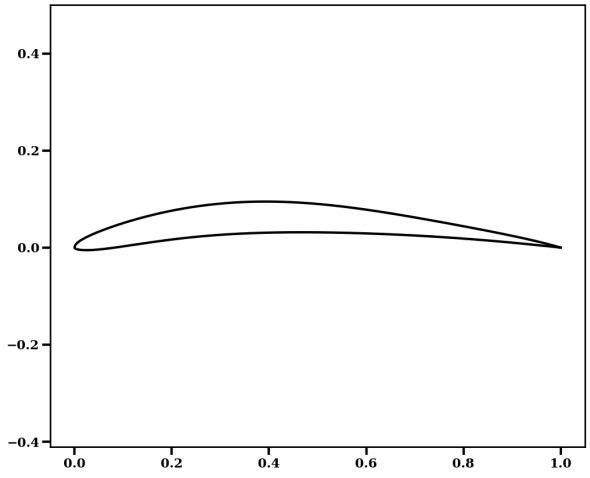

Figure A.18

CST Airfoil, Index: 192

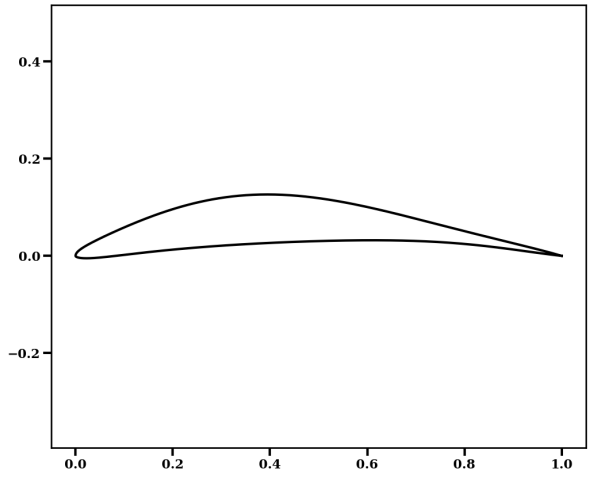

Figure A.19

CST Airfoil, Index: 204

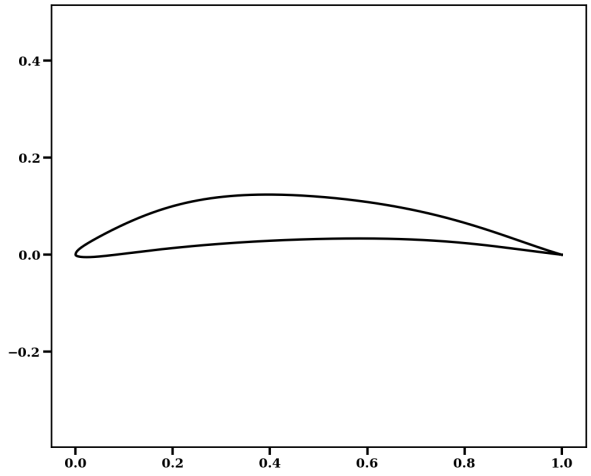

Figure A.20

CST Airfoil, Index: 220

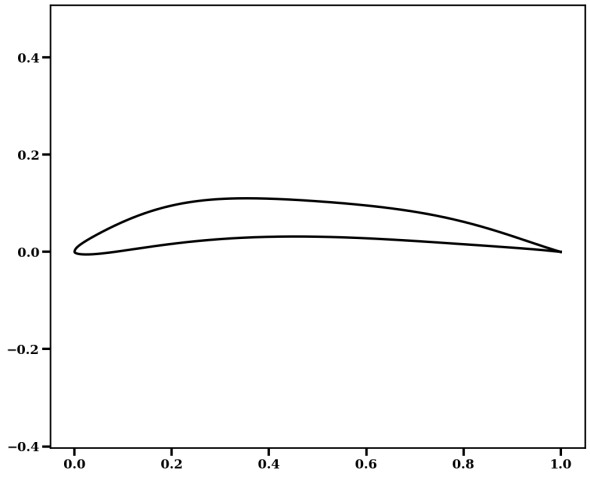

Figure A.21

CST Airfoil, Index: 224

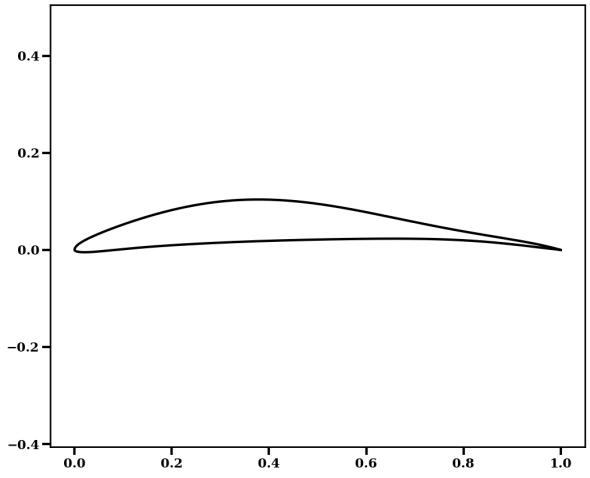

Figure A.22

CST Airfoil, Index: 229

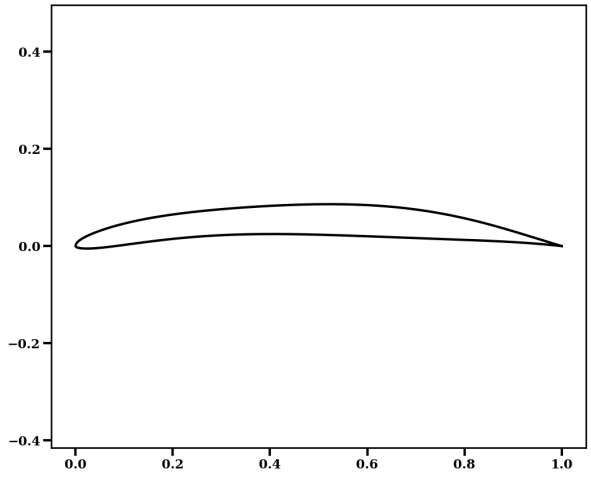

Figure A.23

CST Airfoil, Index: 238

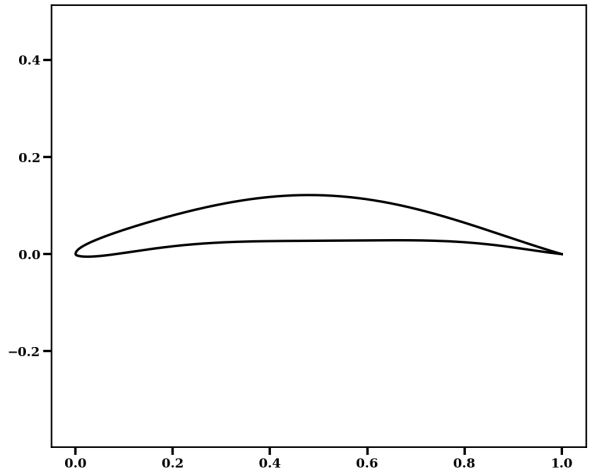

Figure A.24

CST Airfoil, Index: 240

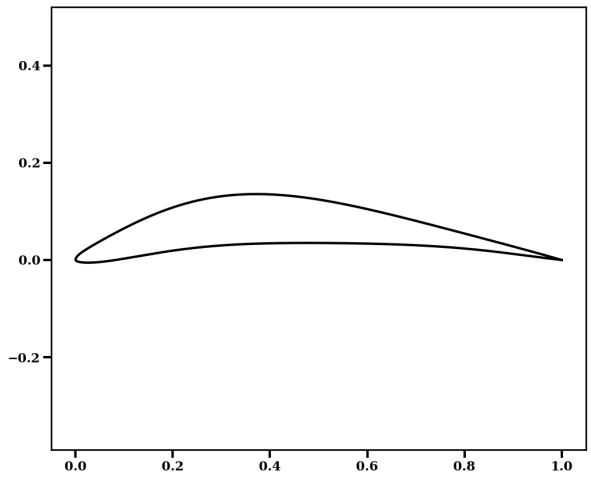

Figure A.25

CST Airfoil, Index: 246

A.3 Hierarchical B-Spline Sparse Grids

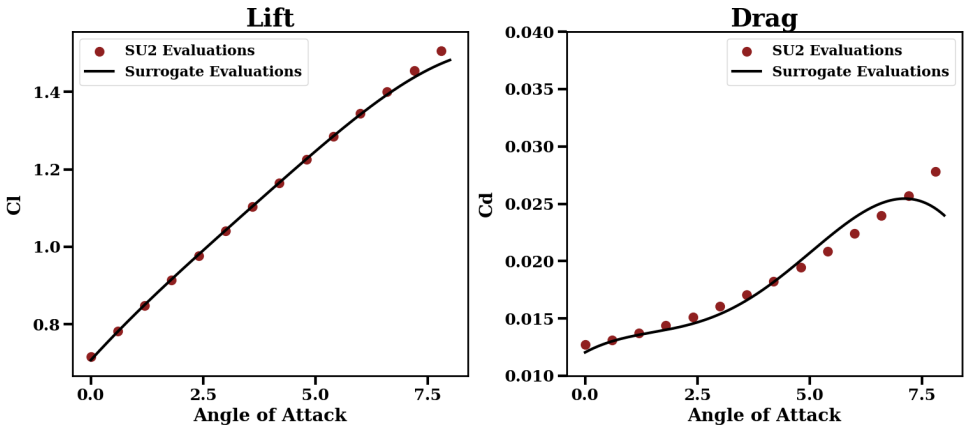

Figure A.26

Sparse Grids B-Spline, Index: 0

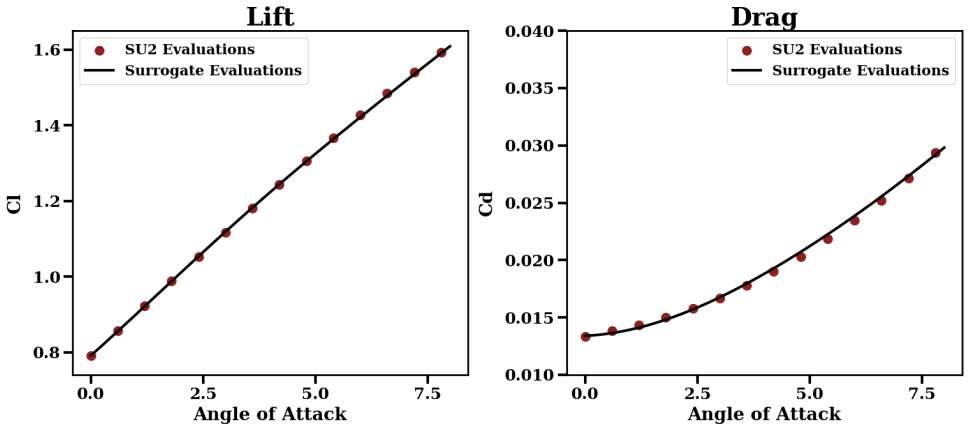

Figure A.27

Sparse Grids B-Spline, Index: 10

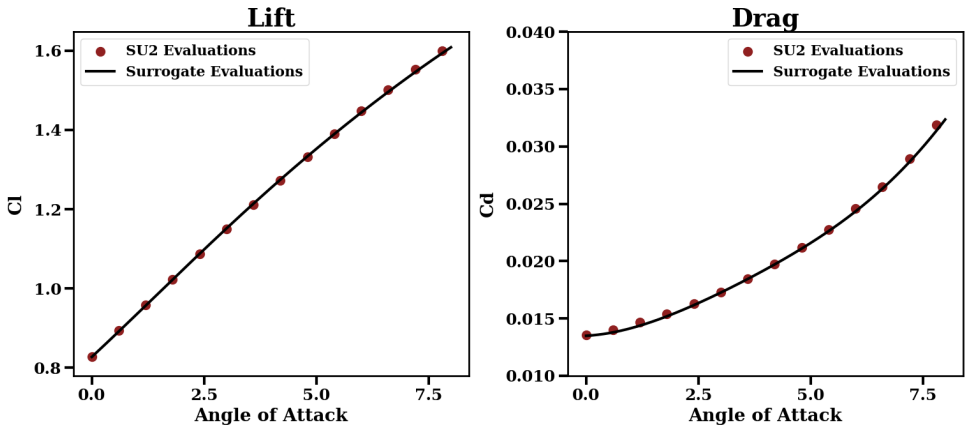

Figure A.28

Sparse Grids B-Spline, Index: 13

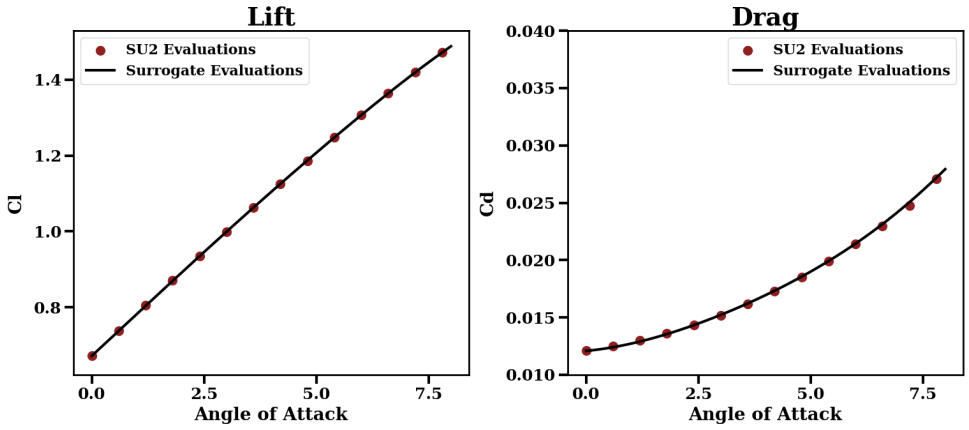

Figure A.29

Sparse Grids B-Spline, Index: 18

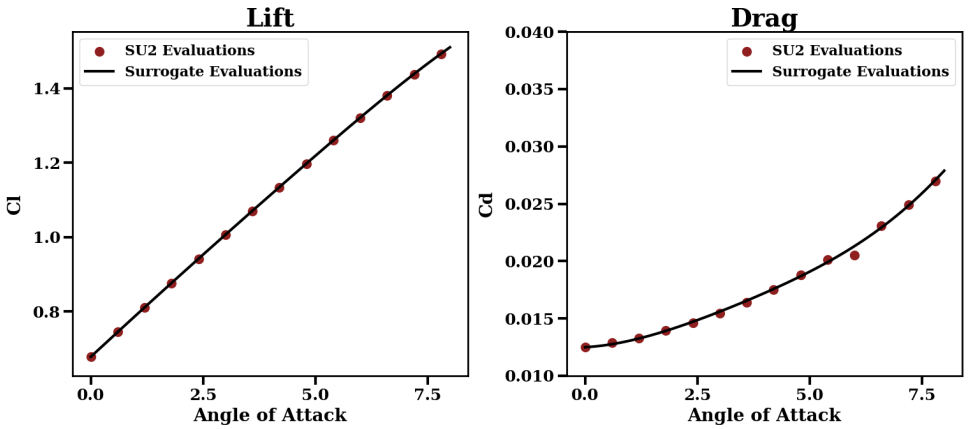

Figure A.30

Sparse Grids B-Spline, Index: 50

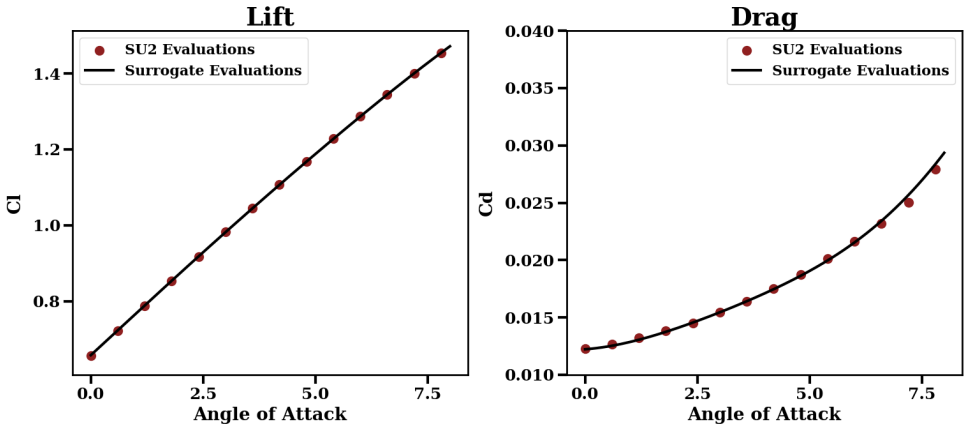

Figure A.31

Sparse Grids B-Spline, Index: 60

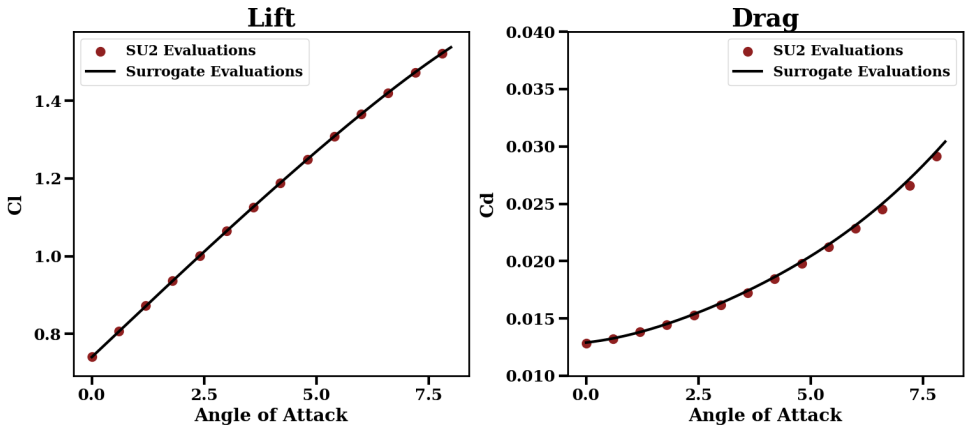

Figure A.32

Sparse Grids B-Spline, Index: 64

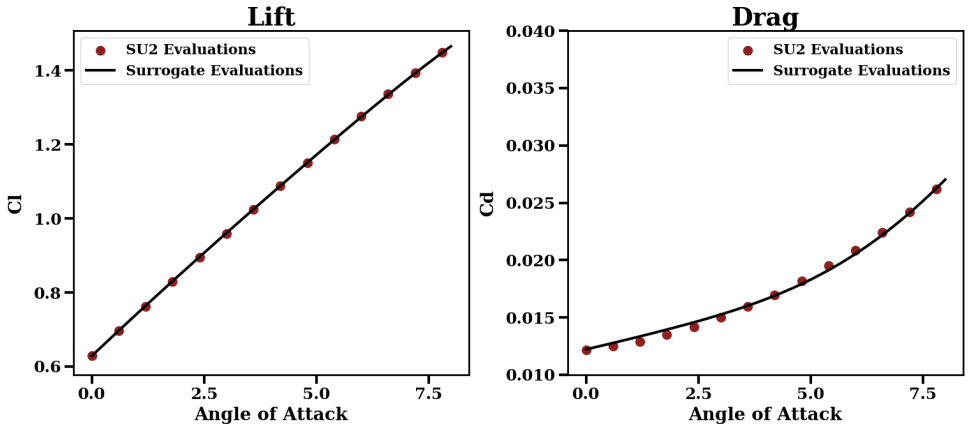

Figure A.33

Sparse Grids B-Spline, Index: 70

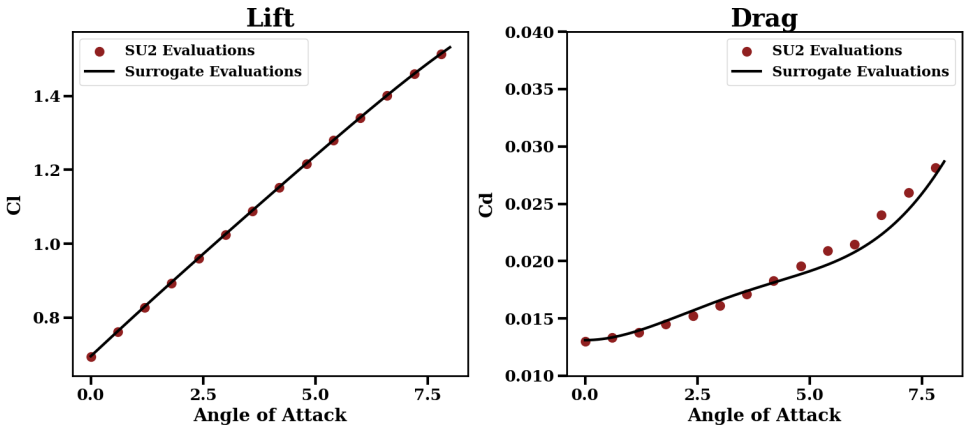

Figure A.34

Sparse Grids B-Spline, Index: 73

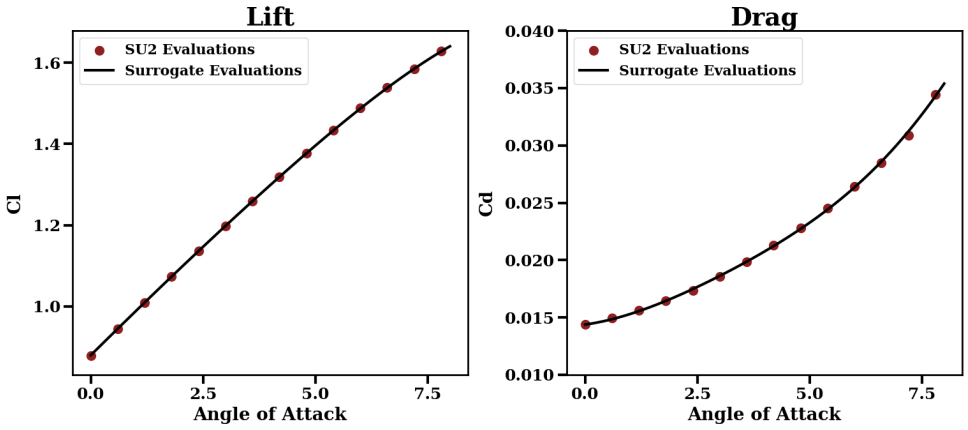

Figure A.35

Sparse Grids B-Spline, Index: 76

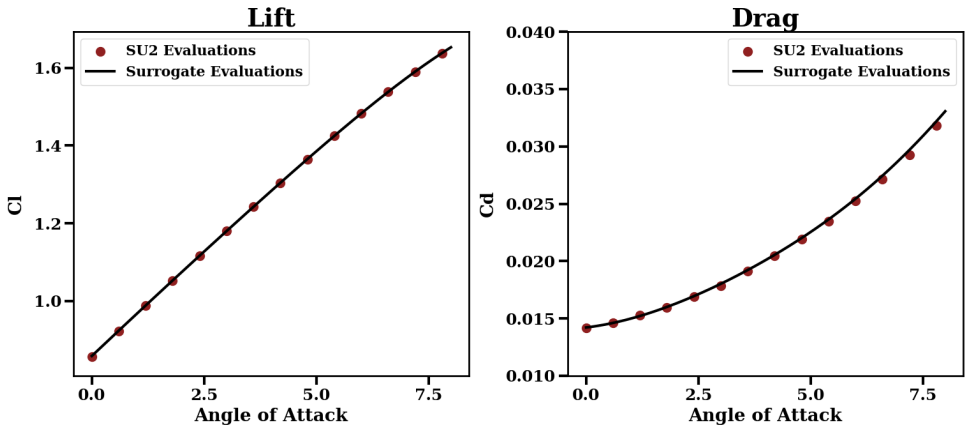

Figure A.36

Sparse Grids B-Spline, Index: 87

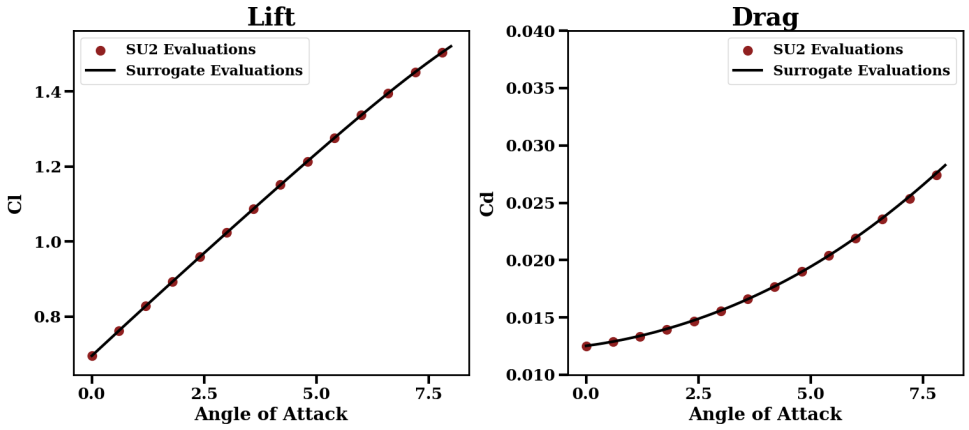

Figure A.37

Sparse Grids B-Spline, Index: 88

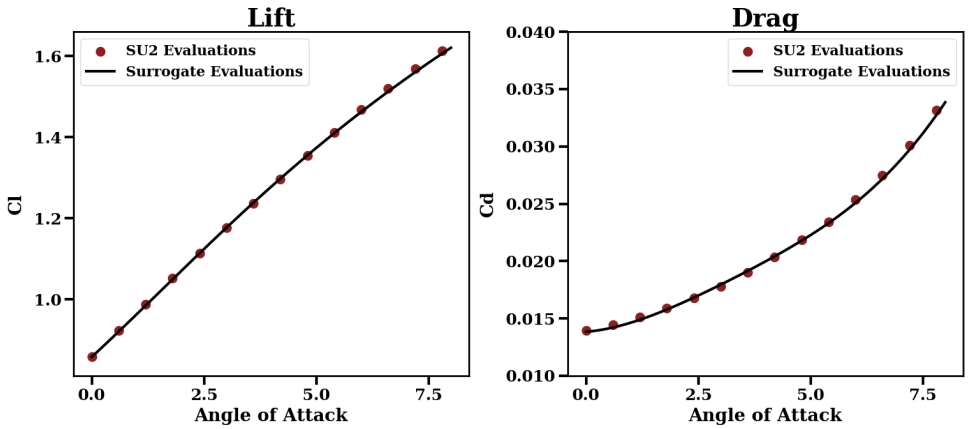

Figure A.38

Sparse Grids B-Spline, Index: 106

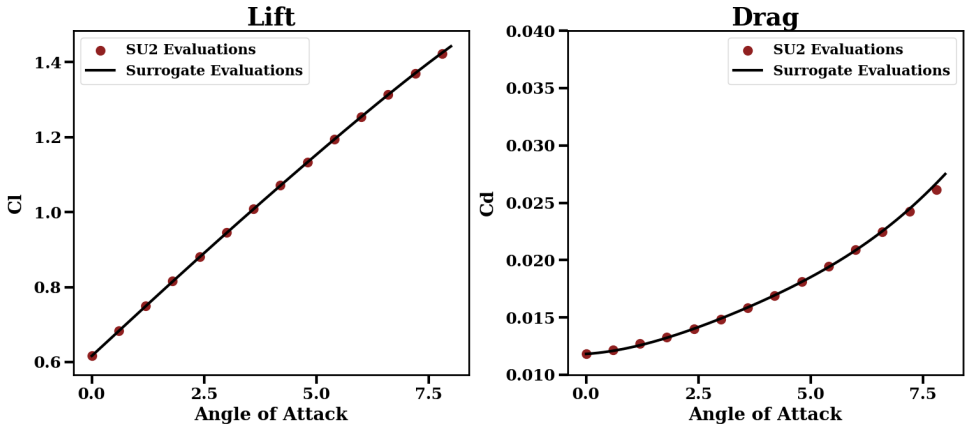

Figure A.39

Sparse Grids B-Spline, Index: 124

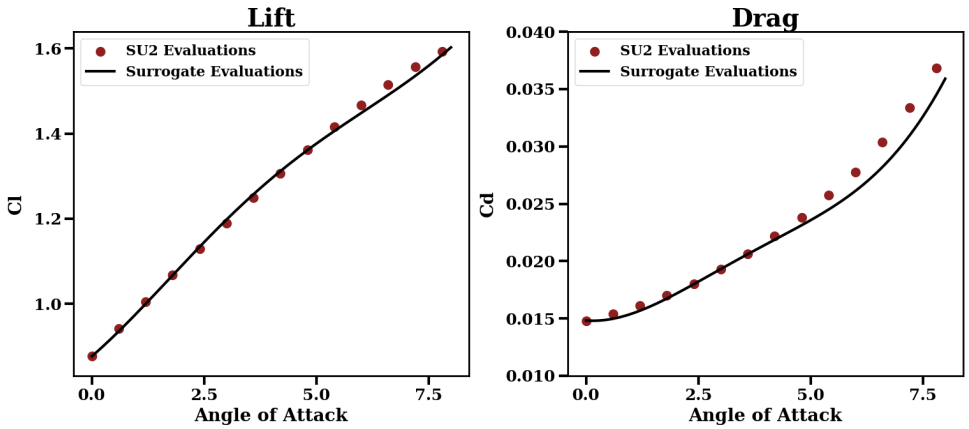

Figure A.40

Sparse Grids B-Spline, Index: 135

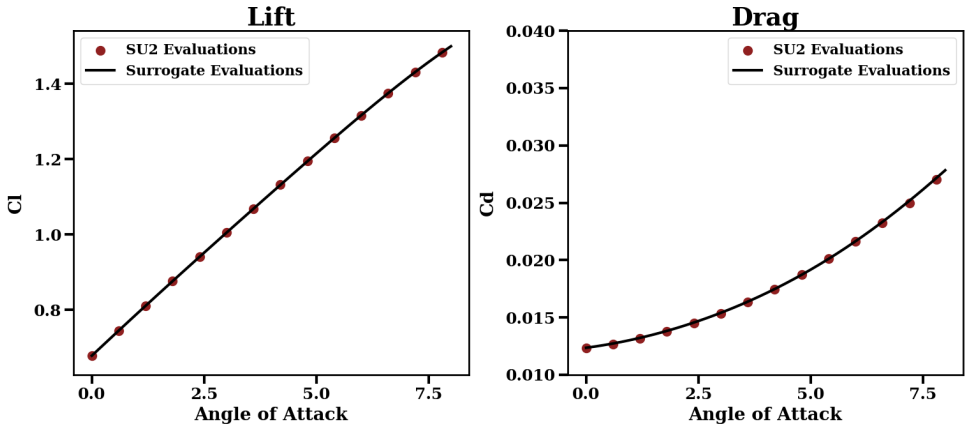

Figure A.41

Sparse Grids B-Spline, Index: 154

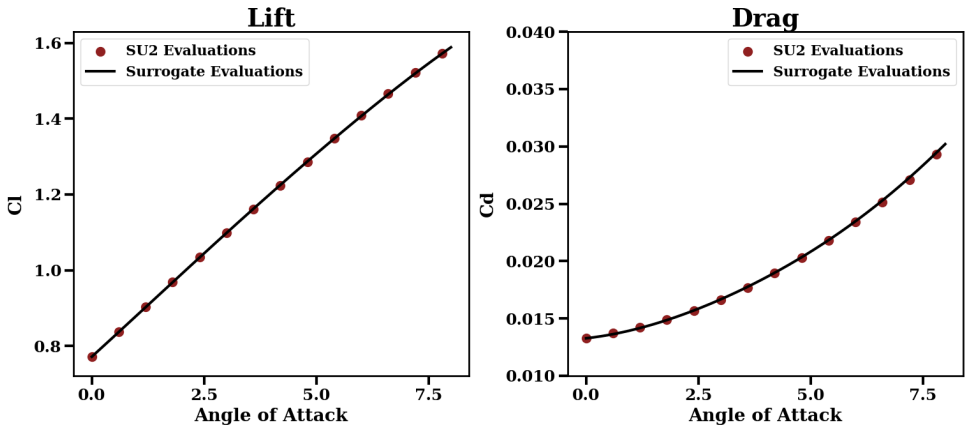

Figure A.42

Sparse Grids B-Spline, Index: 156

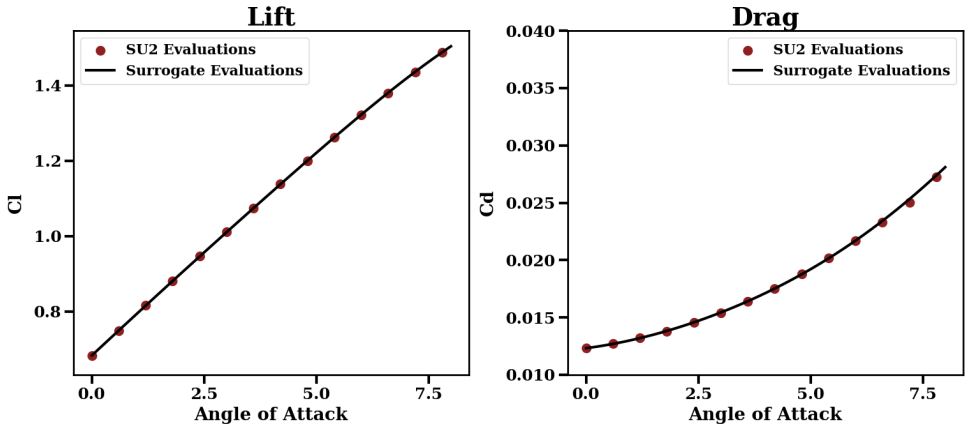

Figure A.43

Sparse Grids B-Spline, Index: 192

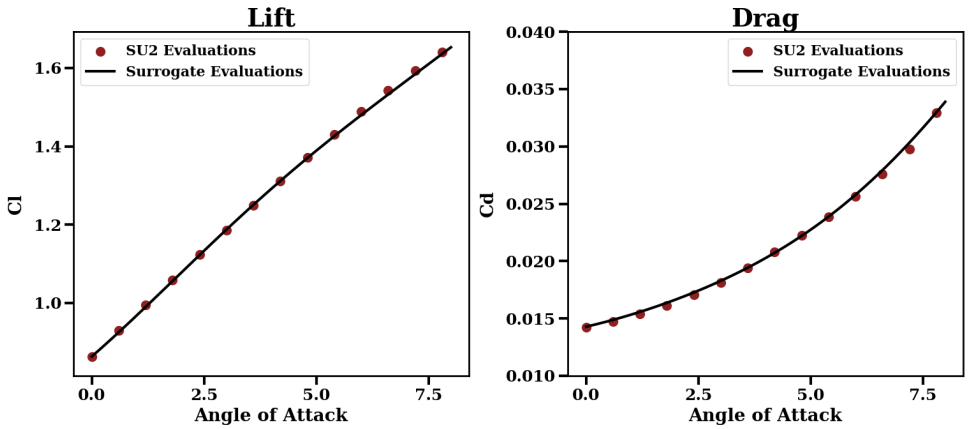

Figure A.44

Sparse Grids B-Spline, Index: 204

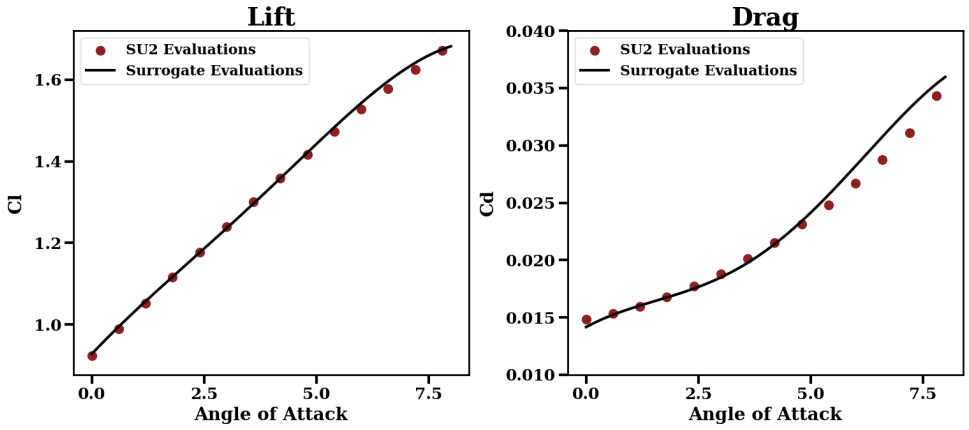

Figure A.45

Sparse Grids B-Spline, Index: 220

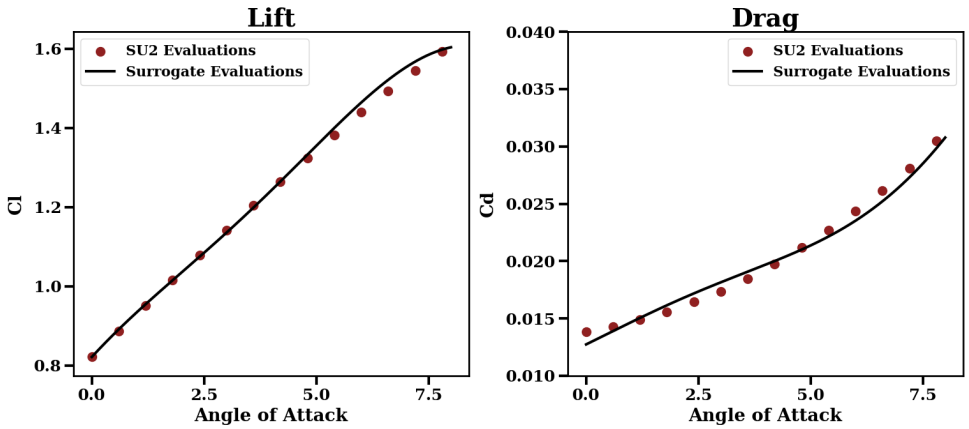

Figure A.46

Sparse Grids B-Spline, Index: 224

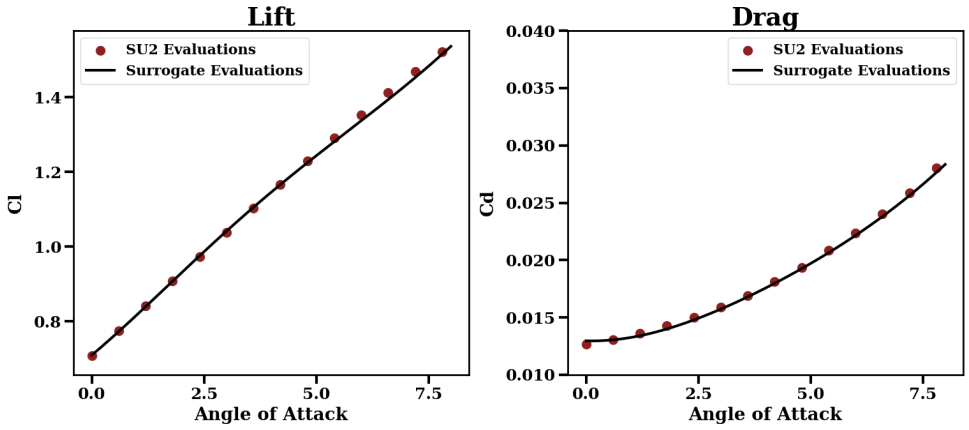

Figure A.47

Sparse Grids B-Spline, Index: 229

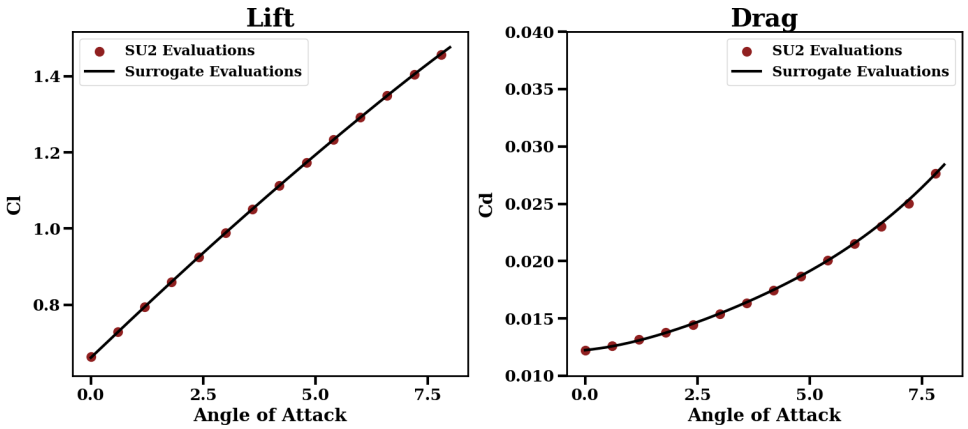

Figure A.48

Sparse Grids B-Spline, Index: 238

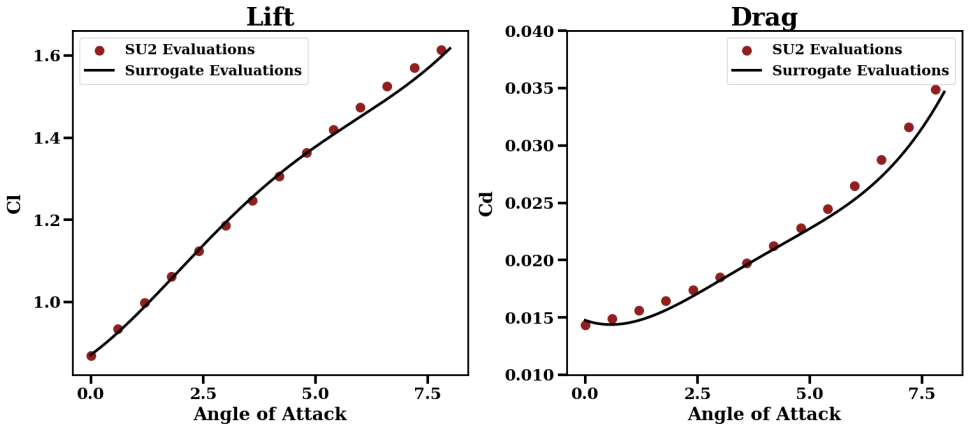

Figure A.49

Sparse Grids B-Spline, Index: 240

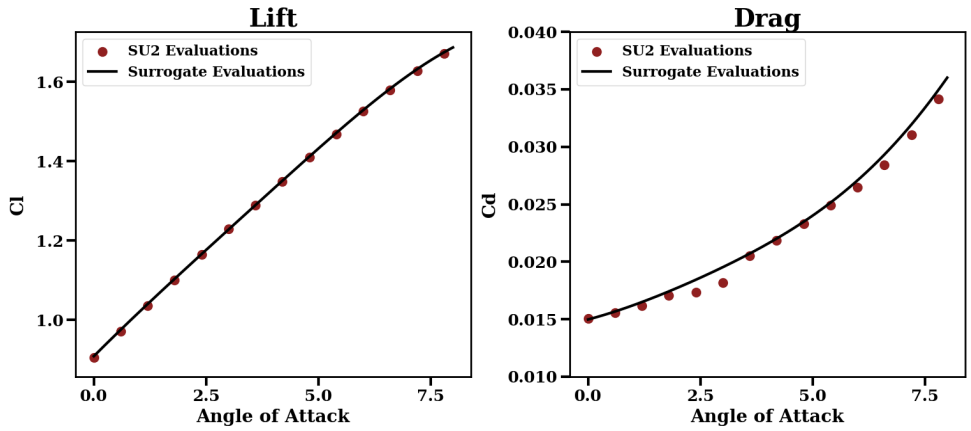

Figure A.50

Sparse Grids B-Spline, Index: 246

A.4 Kriging with Sparse Grids

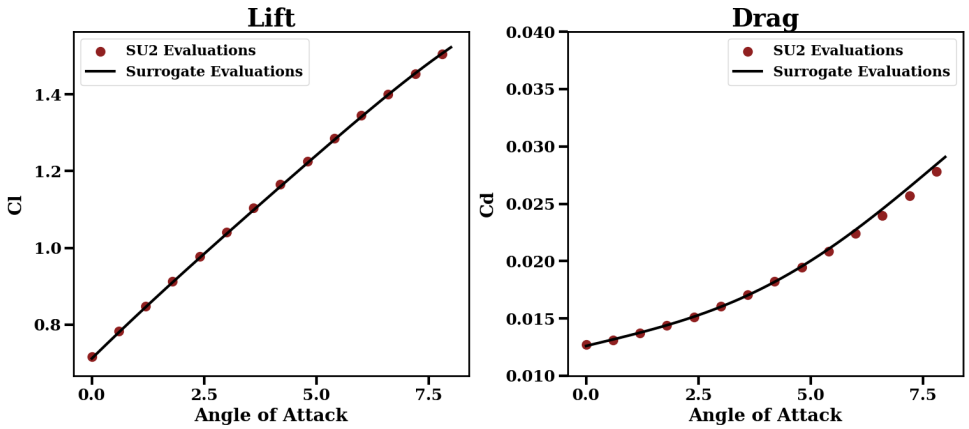

Figure A.51

Sparse Grids Kriging, Index: 0

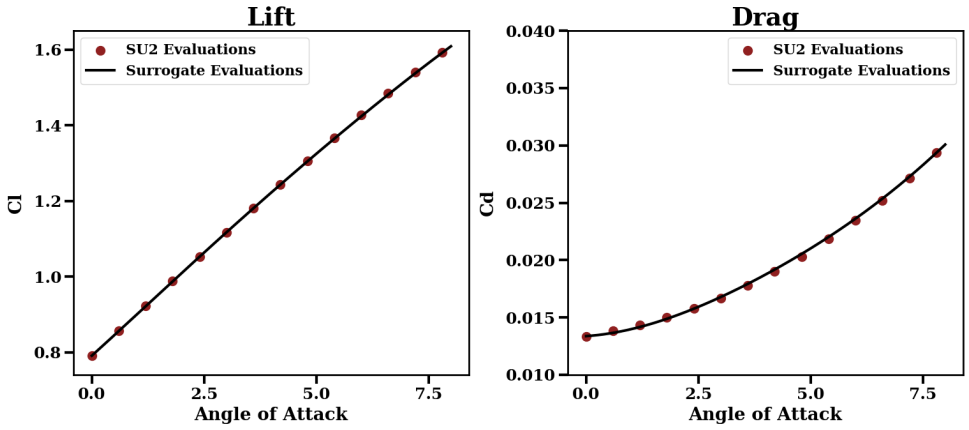

Figure A.52

Sparse Grids Kriging, Index: 10

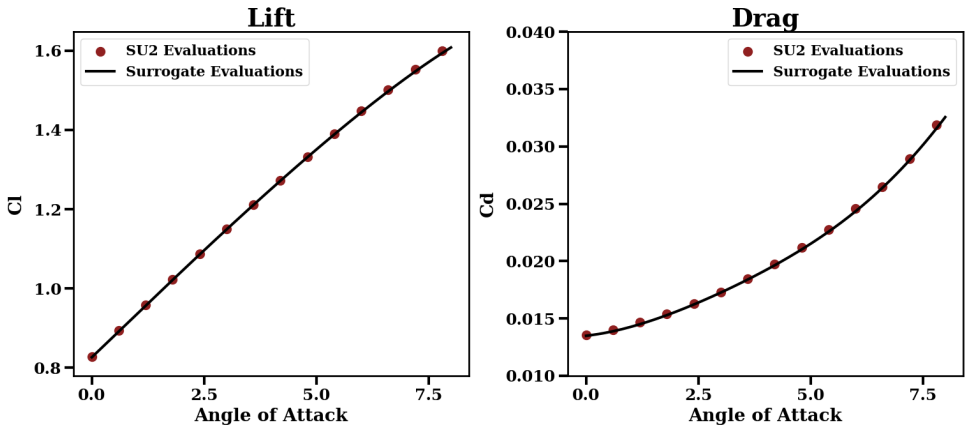

Figure A.53

Sparse Grids Kriging, Index: 13

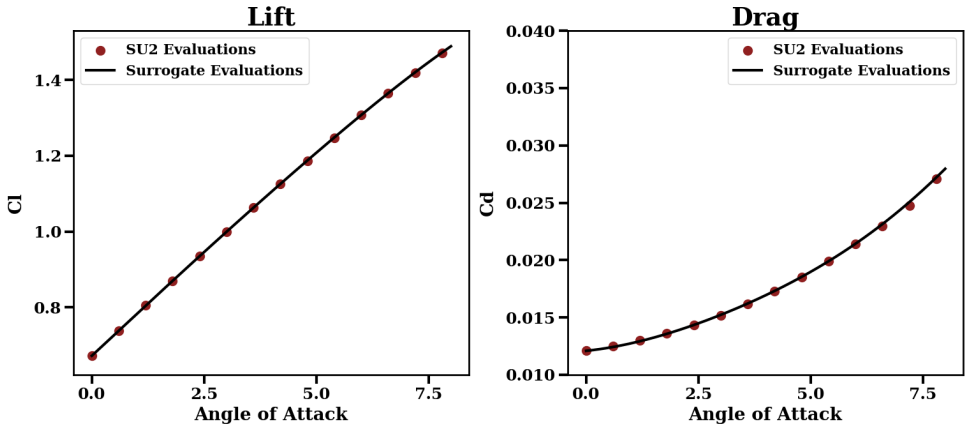

Figure A.54

Sparse Grids Kriging, Index: 18

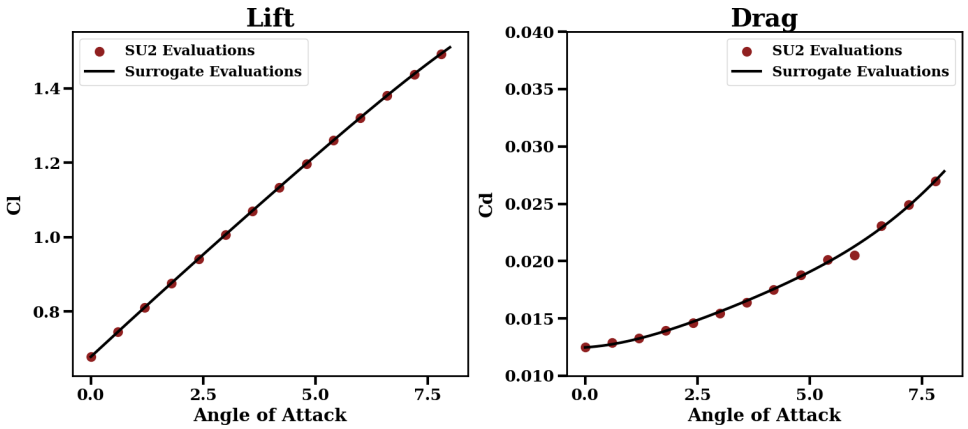

Figure A.55

Sparse Grids Kriging, Index: 50

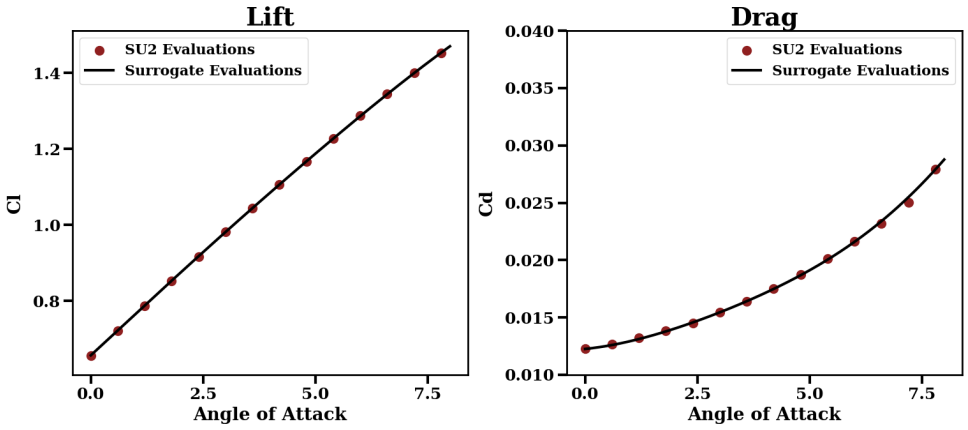

Figure A.56

Sparse Grids Kriging, Index: 60

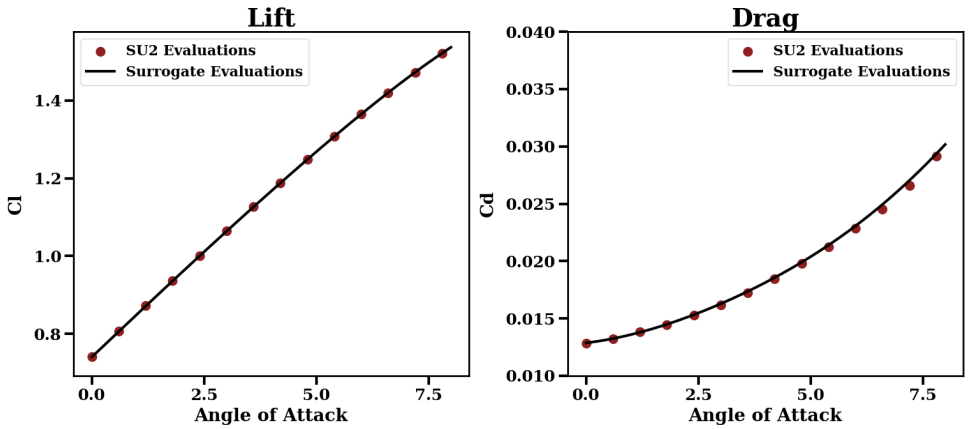

Figure A.57

Sparse Grids Kriging, Index: 64

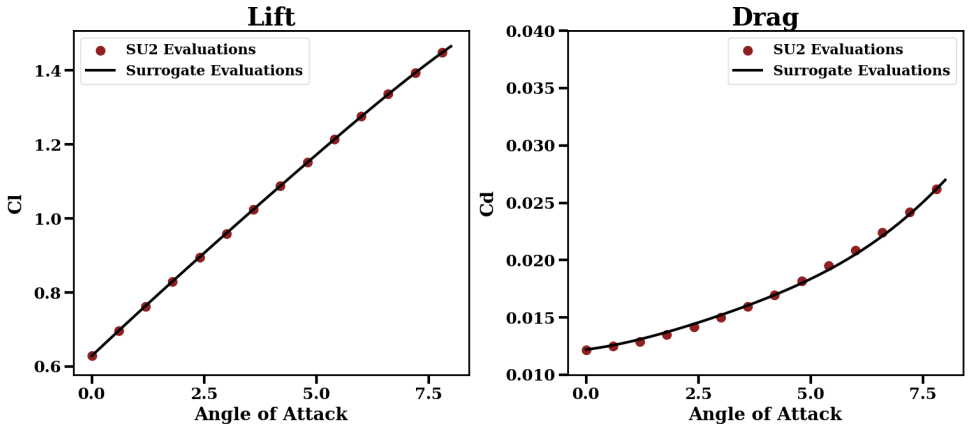

Figure A.58

Sparse Grids Kriging, Index: 70

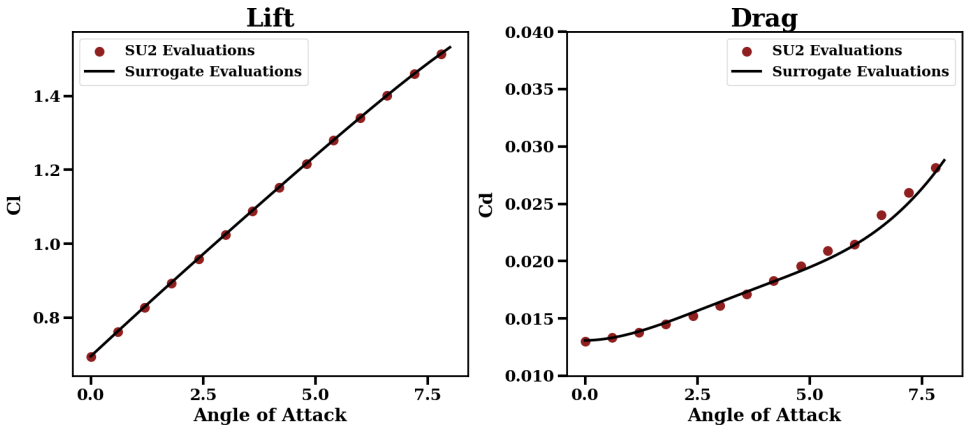

Figure A.59

Sparse Grids Kriging, Index: 73

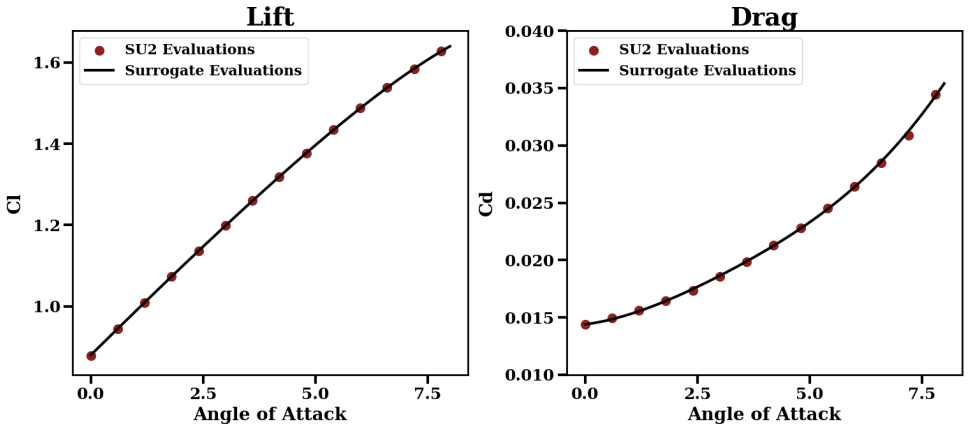

Figure A.60

Sparse Grids Kriging, Index: 76

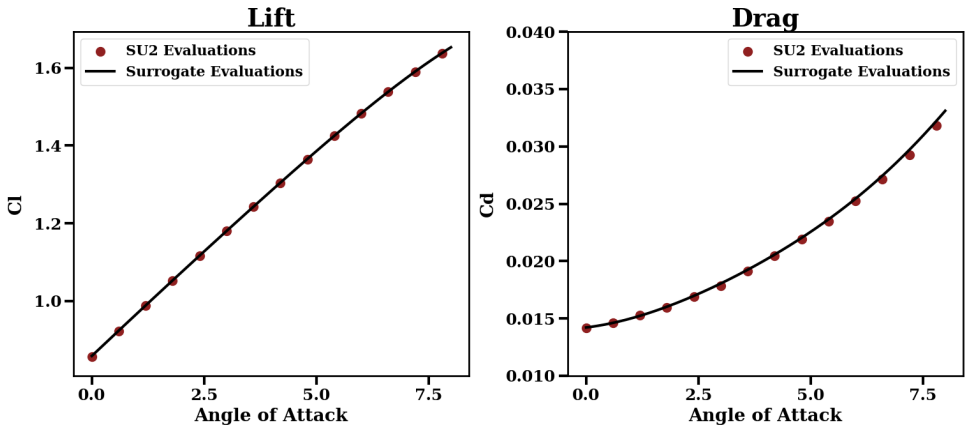

Figure A.61

Sparse Grids Kriging, Index: 87

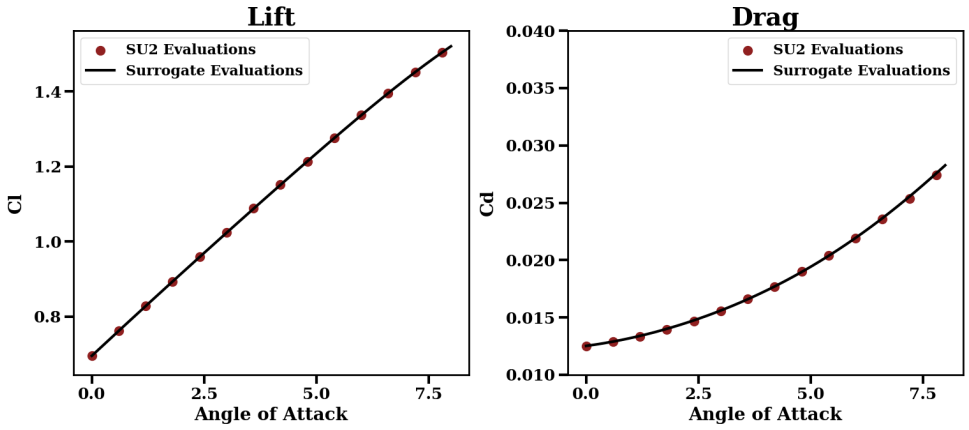

Figure A.62

Sparse Grids Kriging, Index: 88

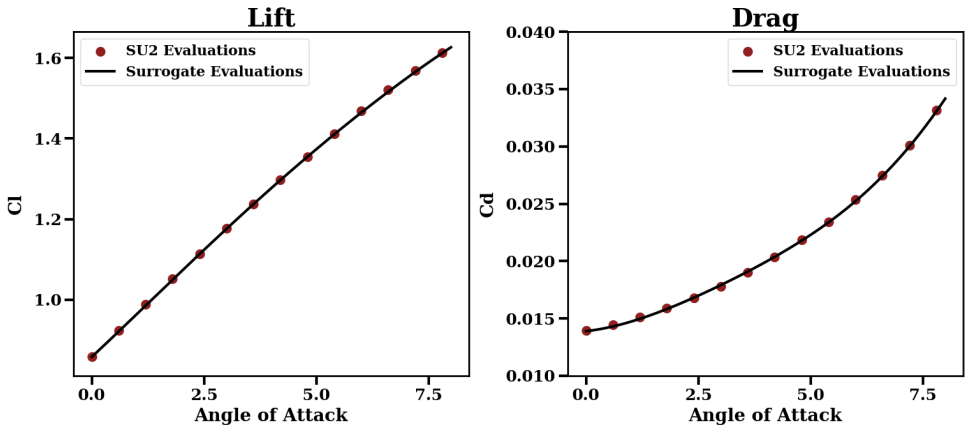

Figure A.63

Sparse Grids Kriging, Index: 106

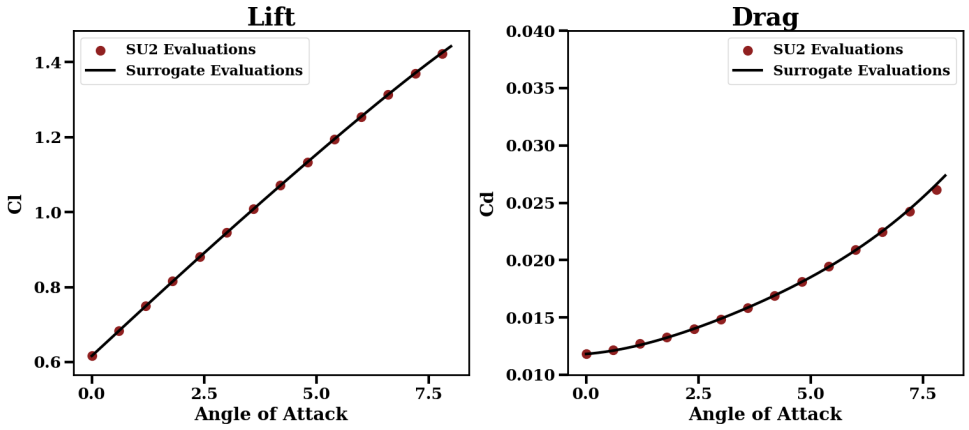

Figure A.64

Sparse Grids Kriging, Index: 124

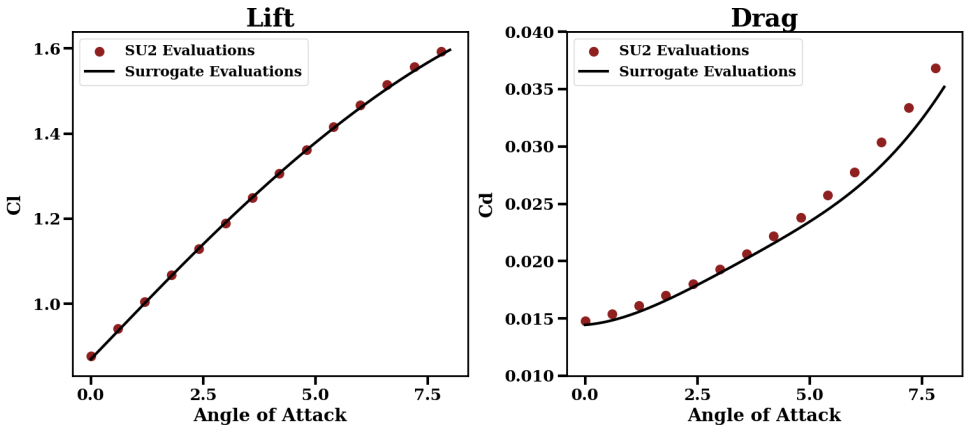

Figure A.65

Sparse Grids Kriging, Index: 135

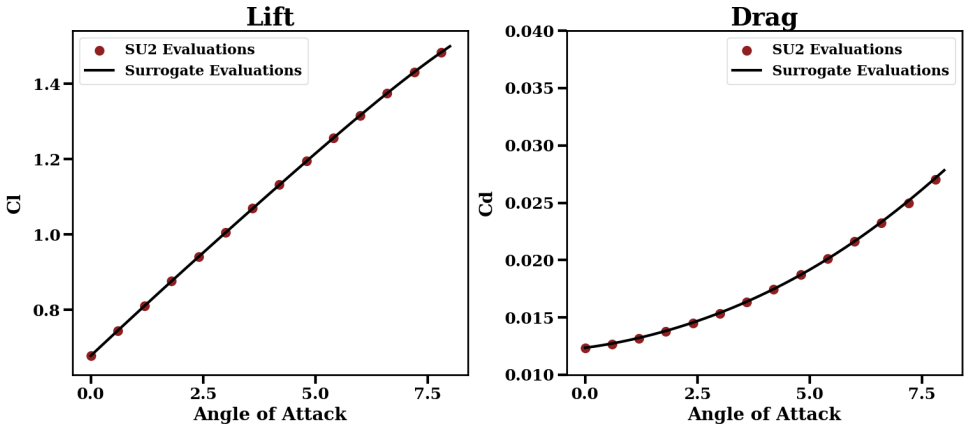

Figure A.66

Sparse Grids Kriging, Index: 154

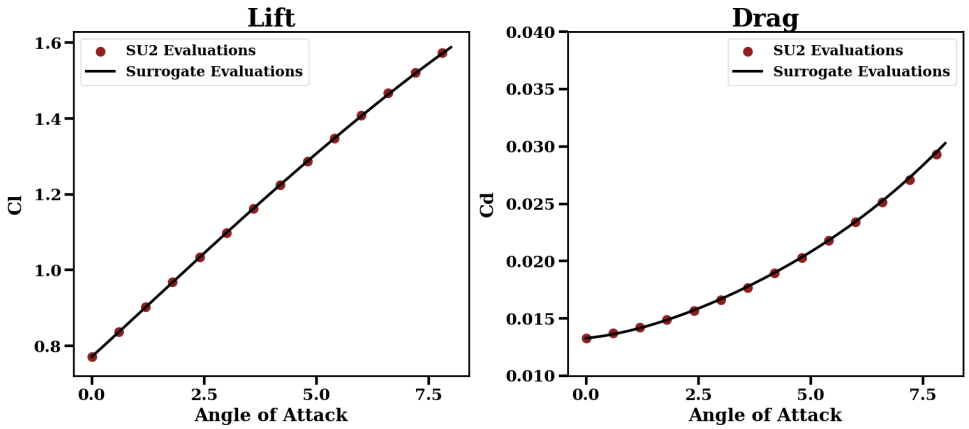

Figure A.67

Sparse Grids Kriging, Index: 156

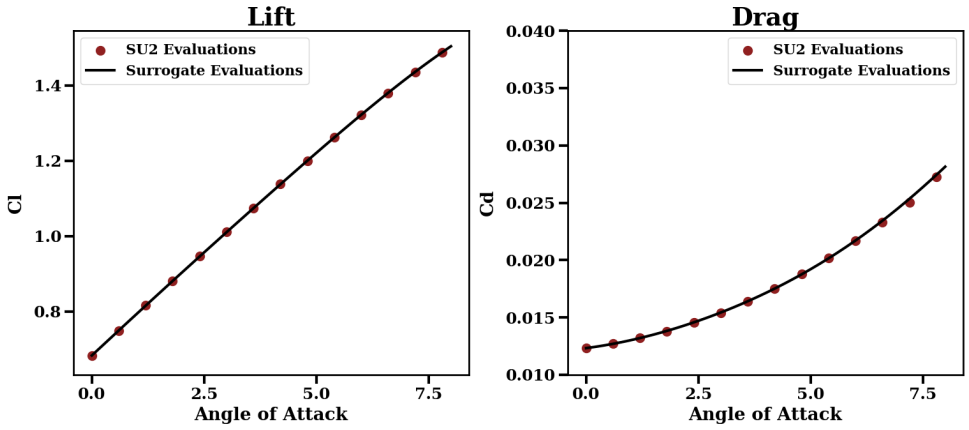

Figure A.68

Sparse Grids Kriging, Index: 192

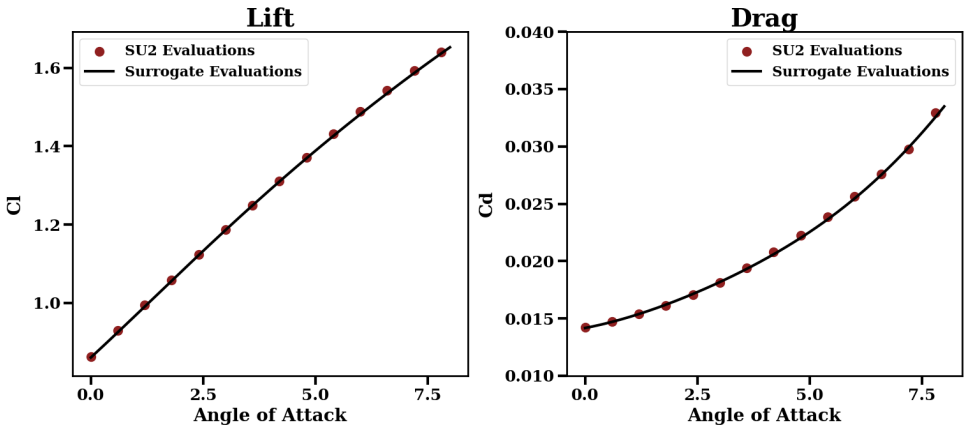

Figure A.69

Sparse Grids Kriging, Index: 204

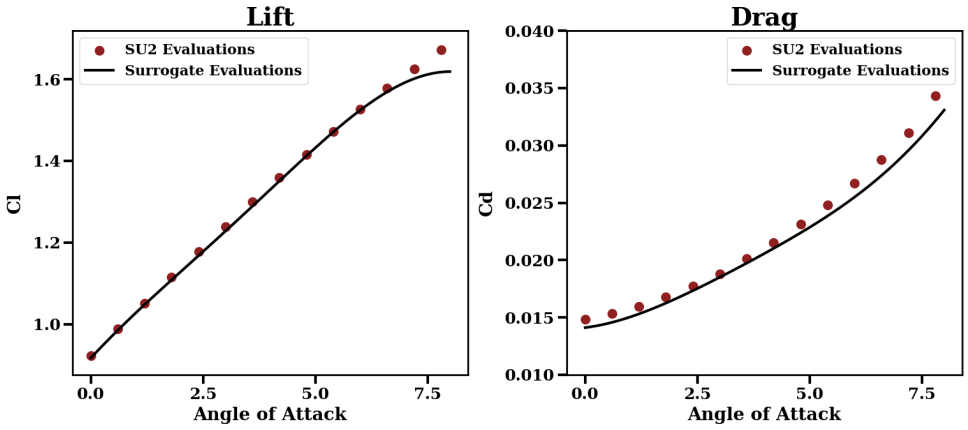

Figure A.70

Sparse Grids Kriging, Index: 220

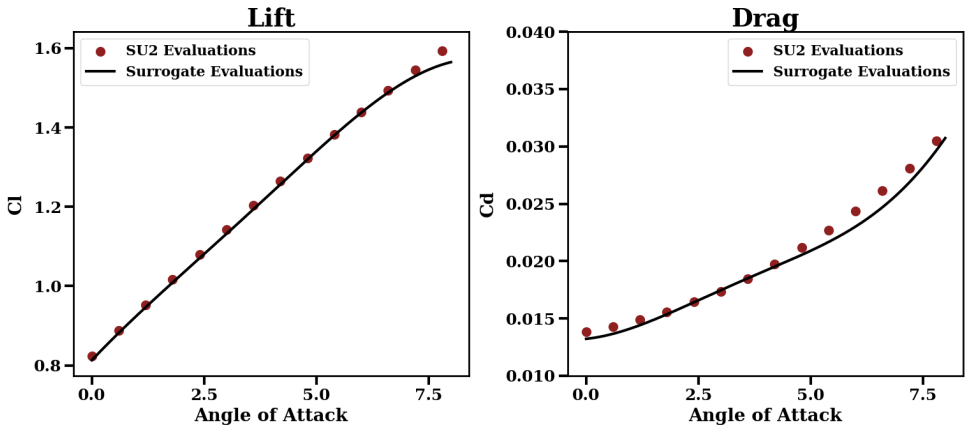

Figure A.71

Sparse Grids Kriging, Index: 224

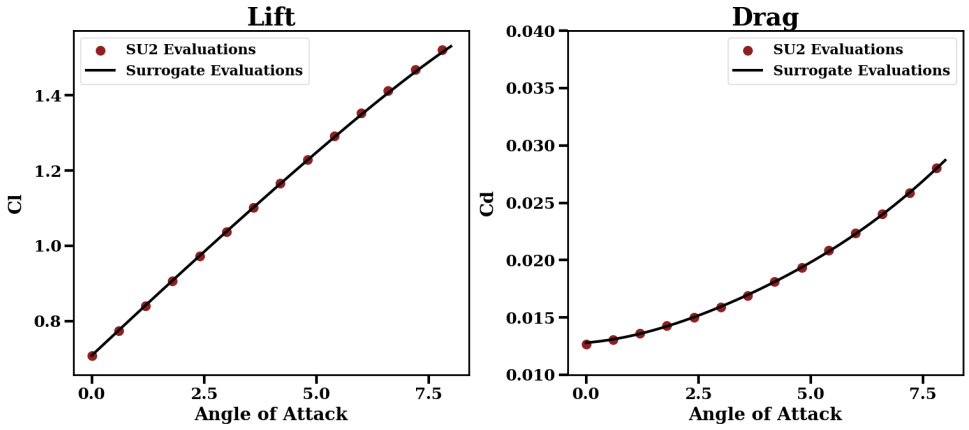

Figure A.72

Sparse Grids Kriging, Index: 229

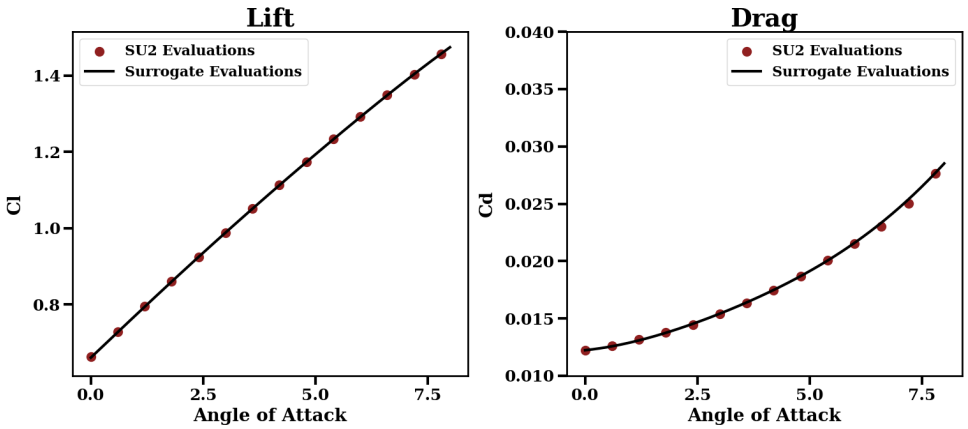

Figure A.73

Sparse Grids Kriging, Index: 238

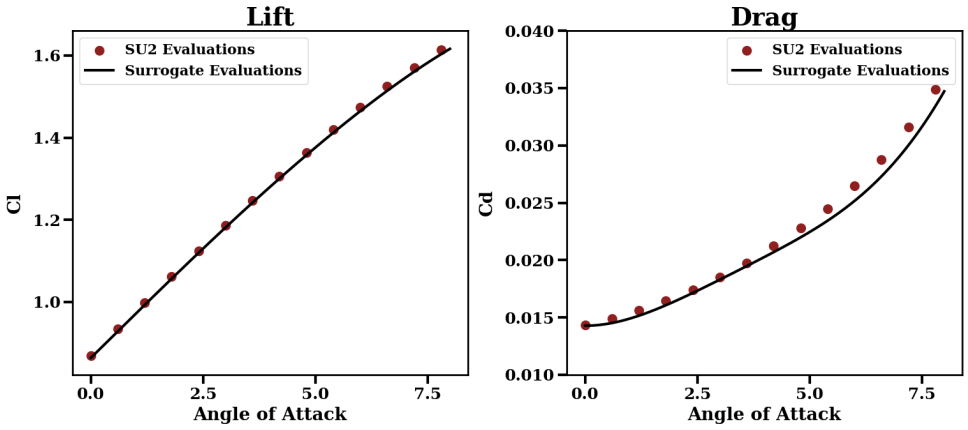

Figure A.74

Sparse Grids Kriging, Index: 240

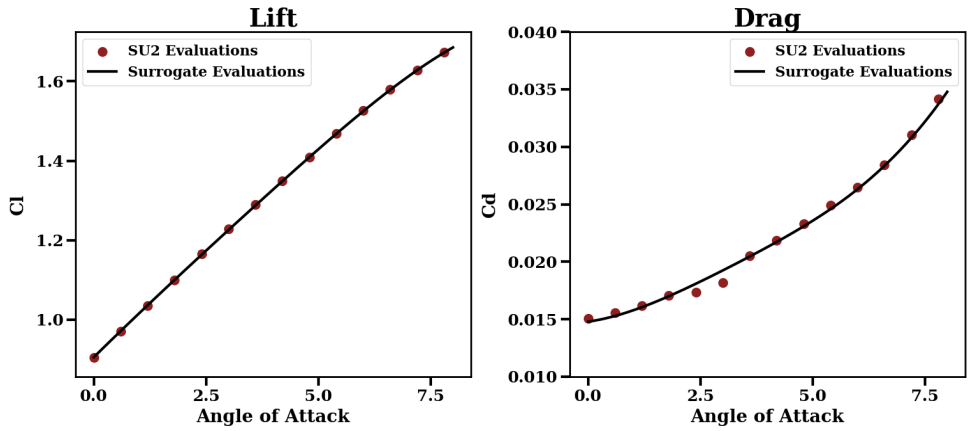

Figure A.75

Sparse Grids Kriging, Index: 246

A.5 Kriging with Latin Hypercubes

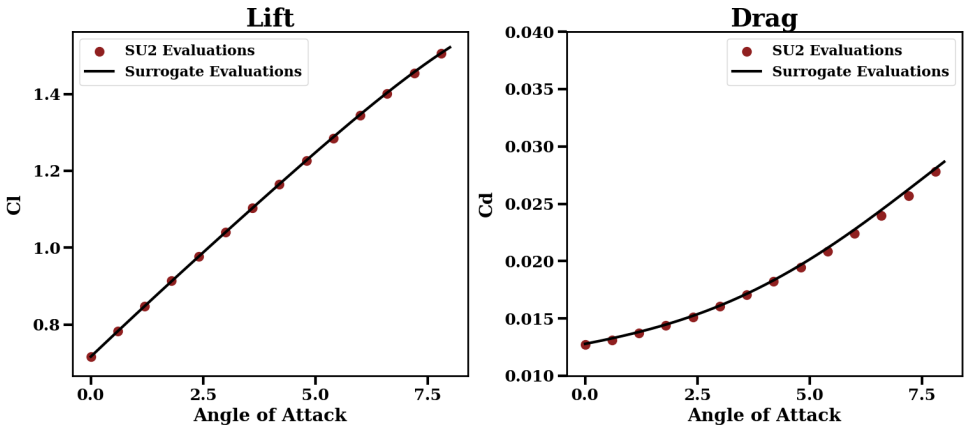

Figure A.76

Latin Hypercube Kriging, Index: 0

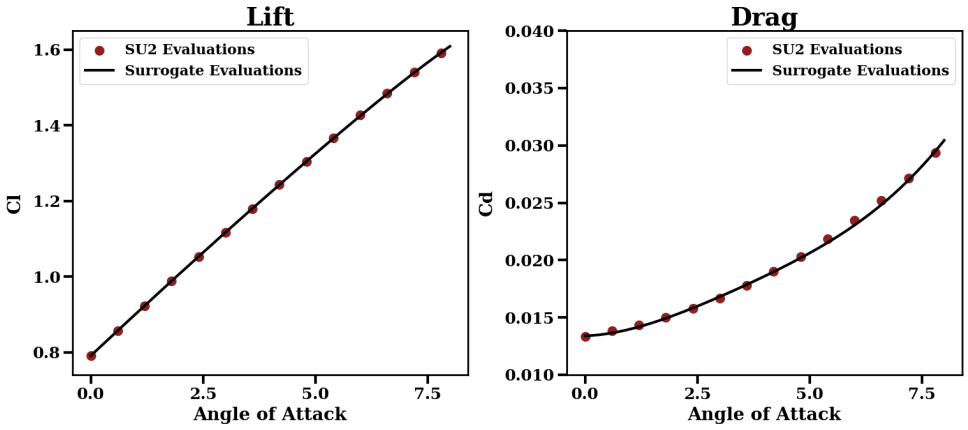

Figure A.77

Latin Hypercube Kriging, Index: 10

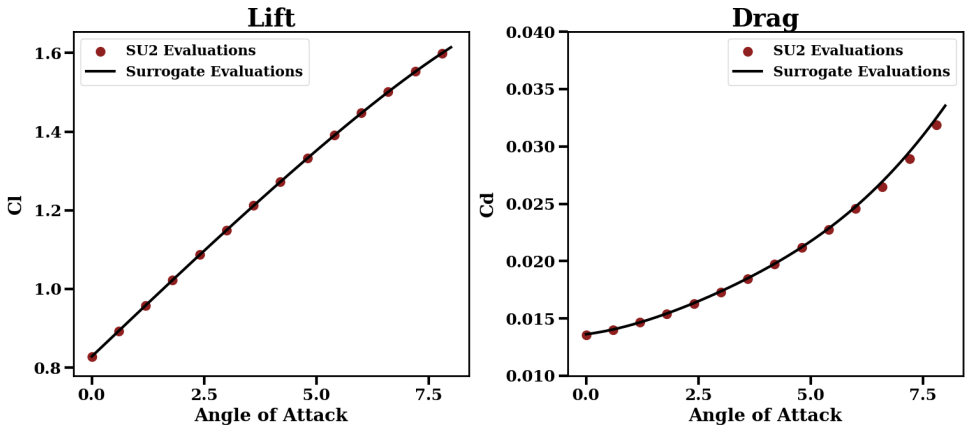

Figure A.78

Latin Hypercube Kriging, Index: 13

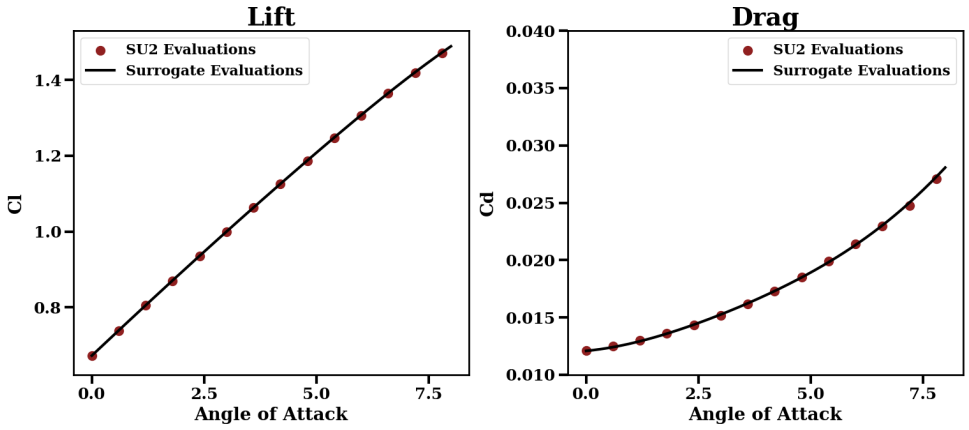

Figure A.79

Latin Hypercube Kriging, Index: 18

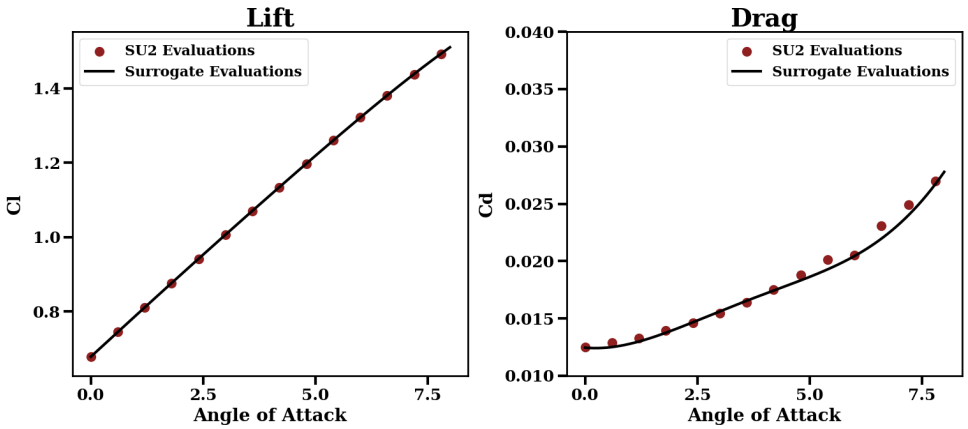

Figure A.80

Latin Hypercube Kriging, Index: 50

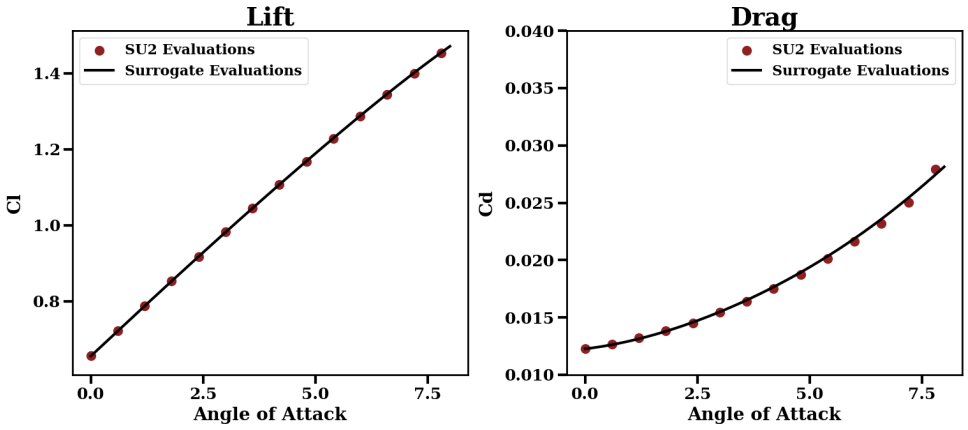

Figure A.81

Latin Hypercube Kriging, Index: 60

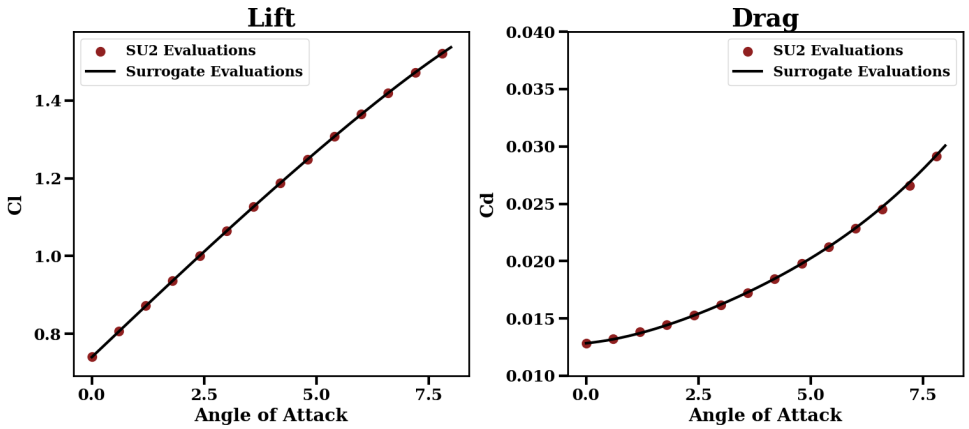

Figure A.82

Latin Hypercube Kriging, Index: 64

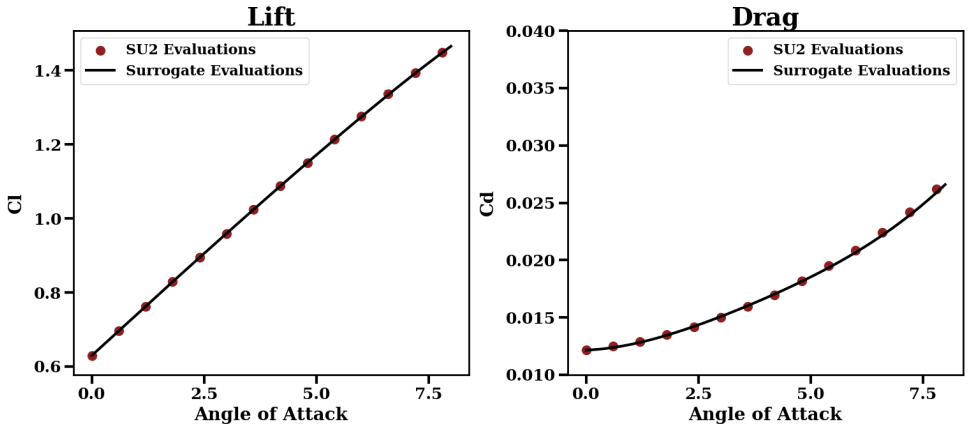

Figure A.83

Latin Hypercube Kriging, Index: 70

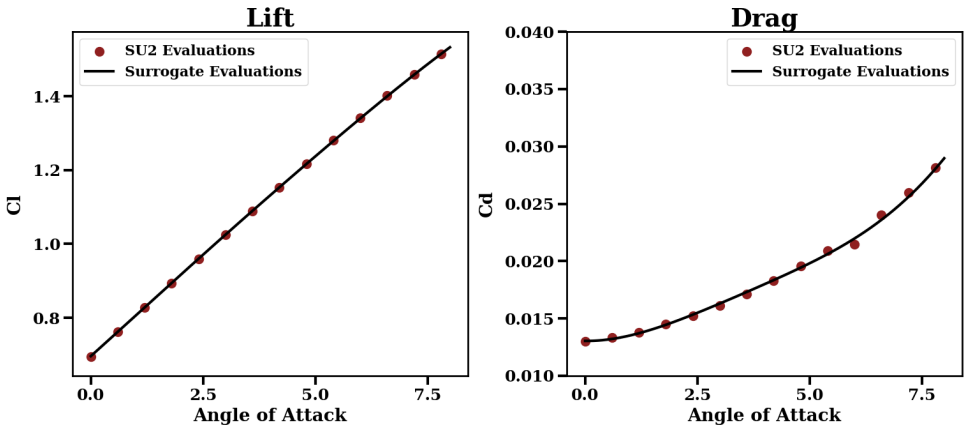

Figure A.84

Latin Hypercube Kriging, Index: 73

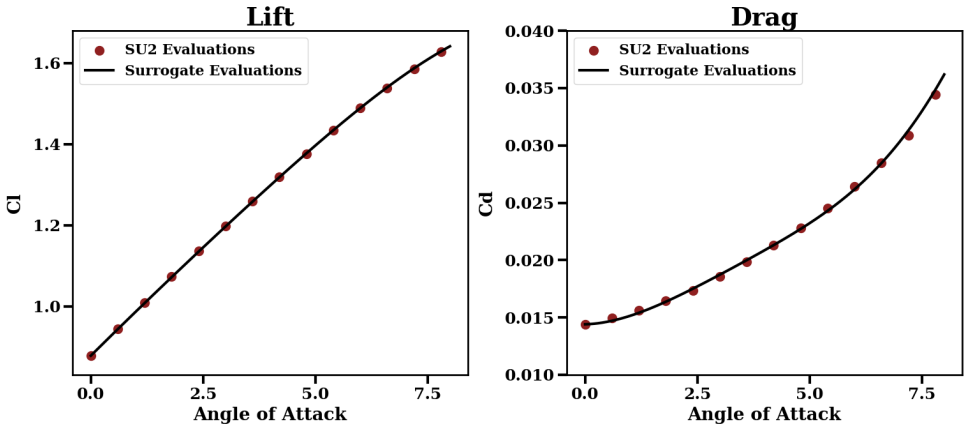

Figure A.85

Latin Hypercube Kriging, Index: 76

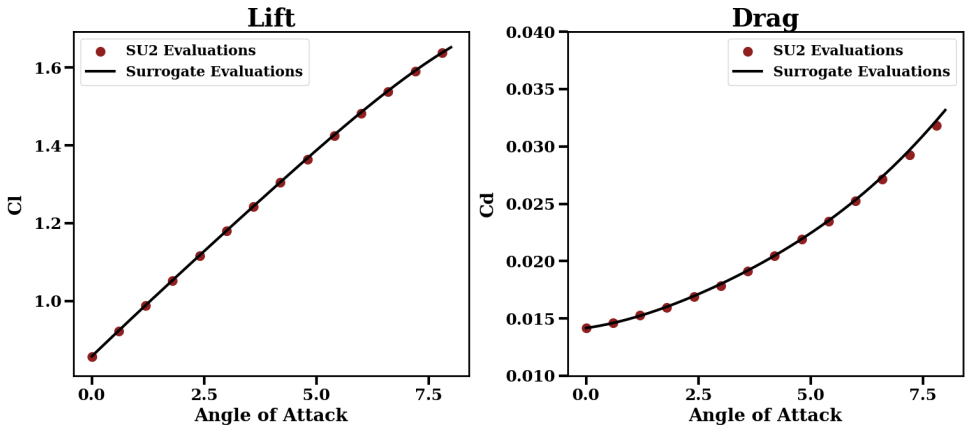

Figure A.86

Latin Hypercube Kriging, Index: 87

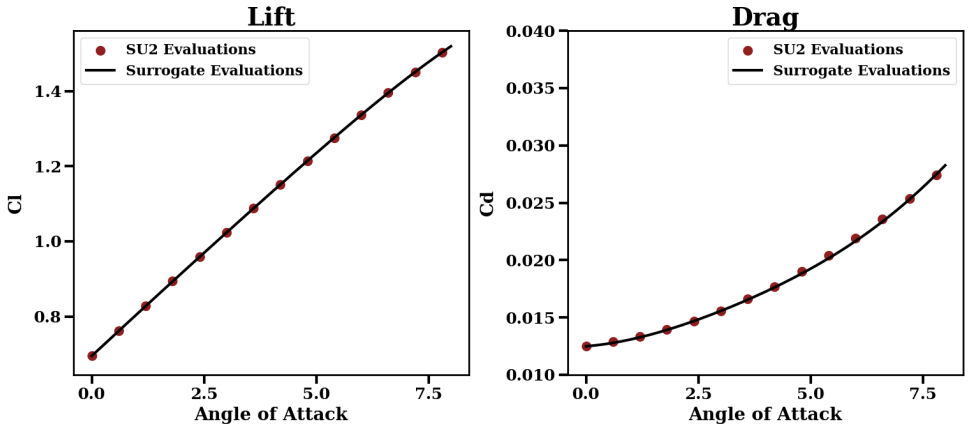

Figure A.87

Latin Hypercube Kriging, Index: 88

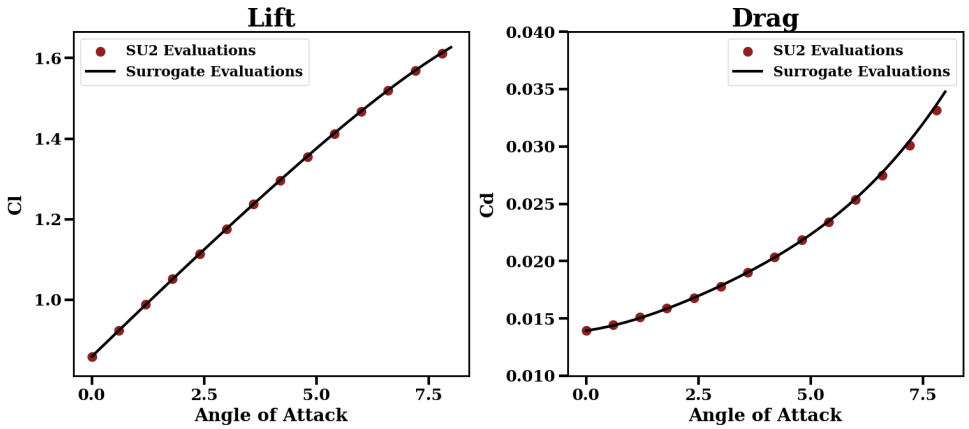

Figure A.88

Latin Hypercube Kriging, Index: 106

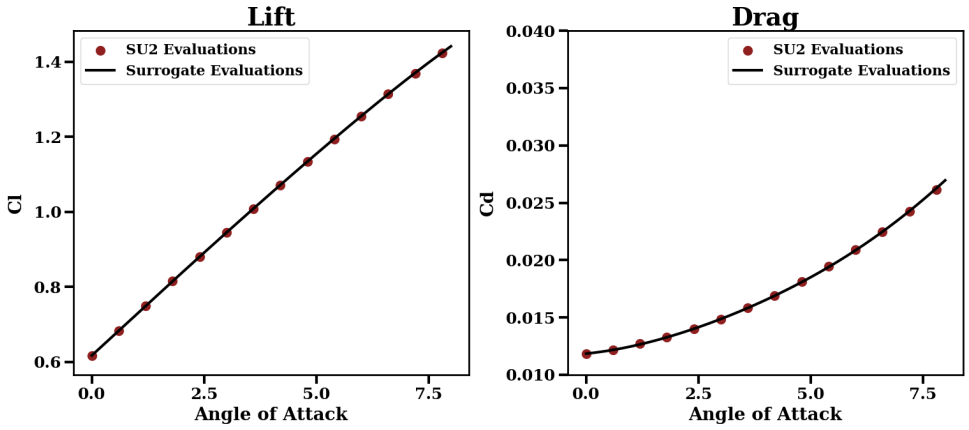

Figure A.89

Latin Hypercube Kriging, Index: 124

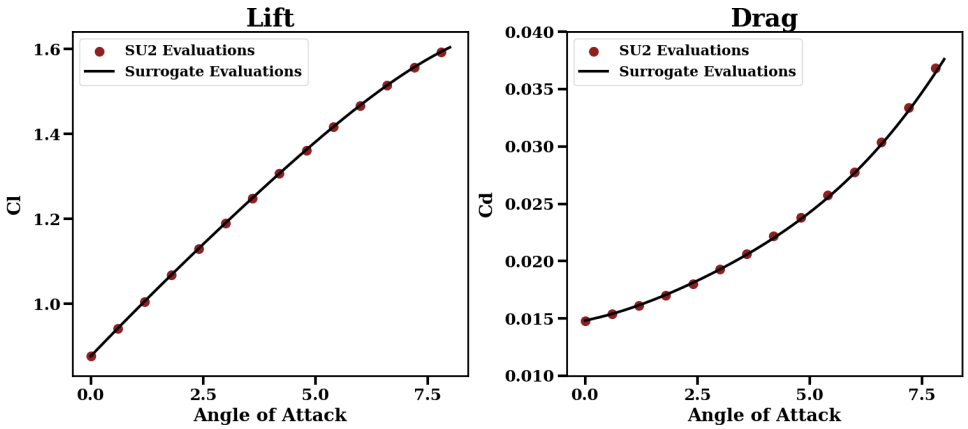

Figure A.90

Latin Hypercube Kriging, Index: 135

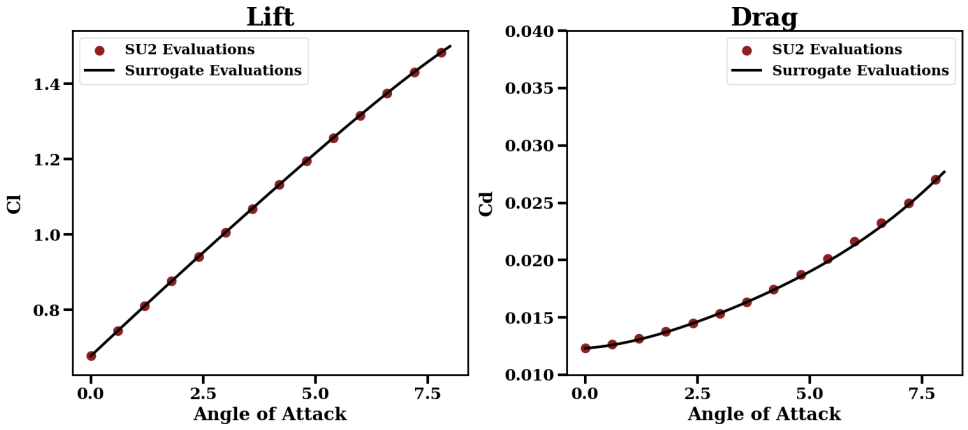

Figure A.91

Latin Hypercube Kriging, Index: 154

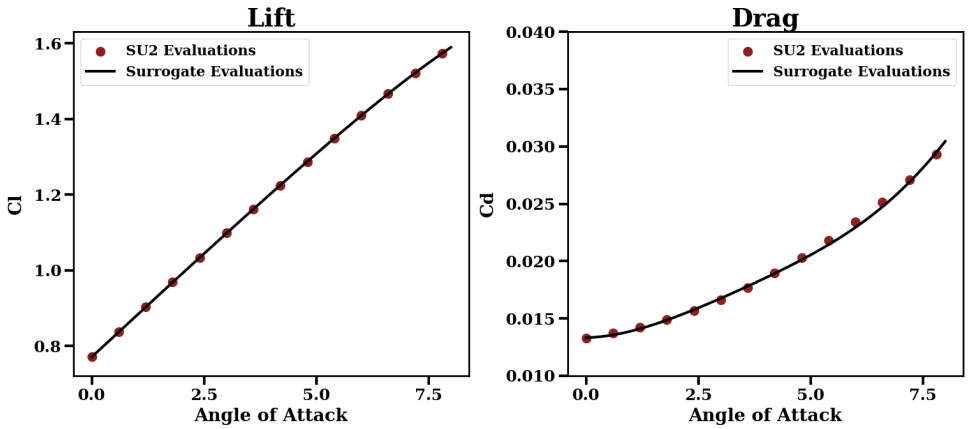

Figure A.92

Latin Hypercube Kriging, Index: 156

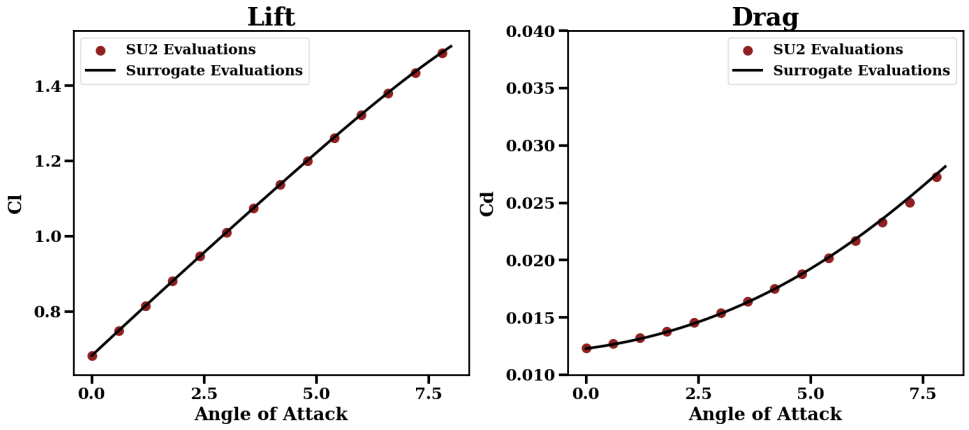

Figure A.93

Latin Hypercube Kriging, Index: 192

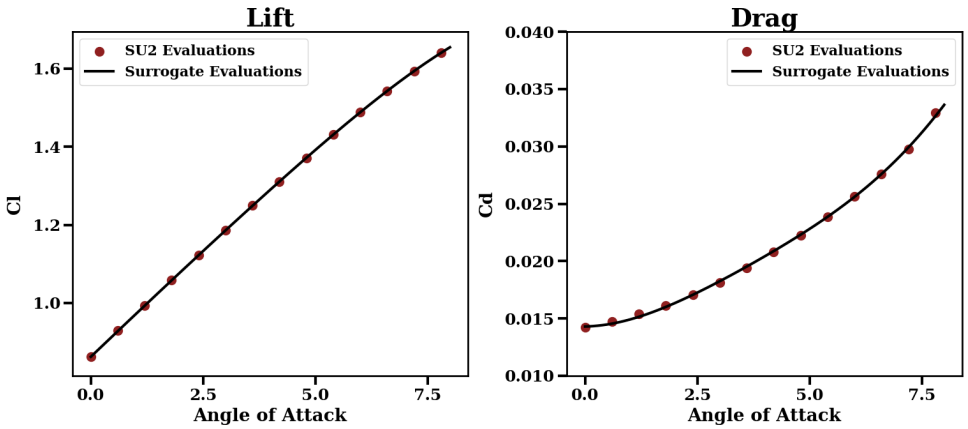

Figure A.94

Latin Hypercube Kriging, Index: 204

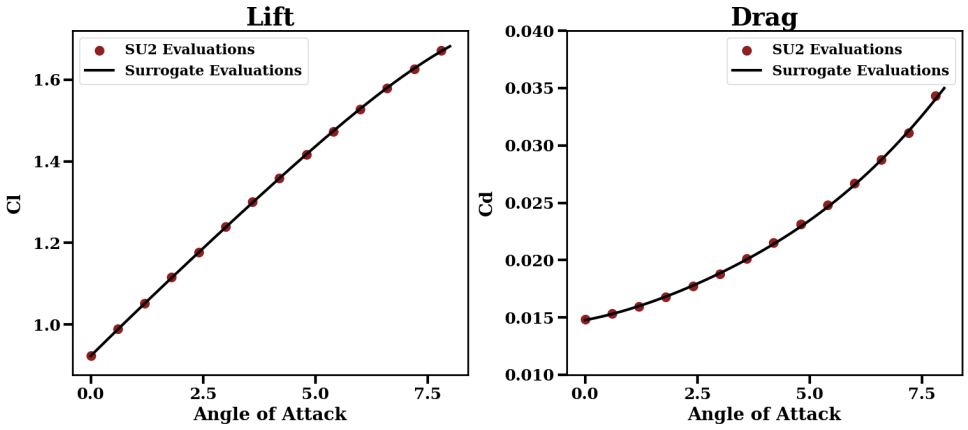

Figure A.95

Latin Hypercube Kriging, Index: 220

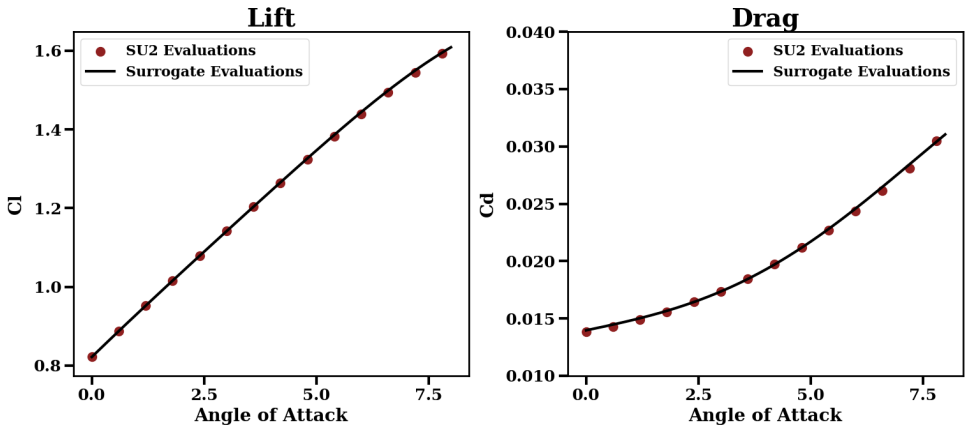

Figure A.96

Latin Hypercube Kriging, Index: 224

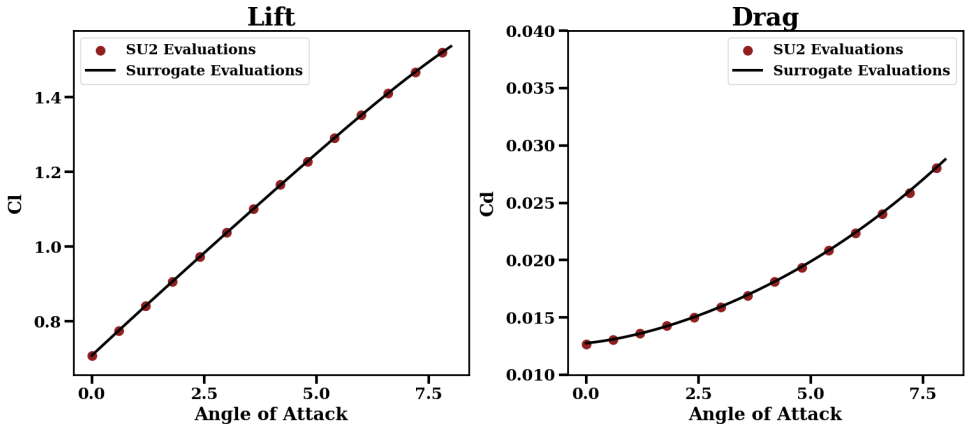

Figure A.97

Latin Hypercube Kriging, Index: 229

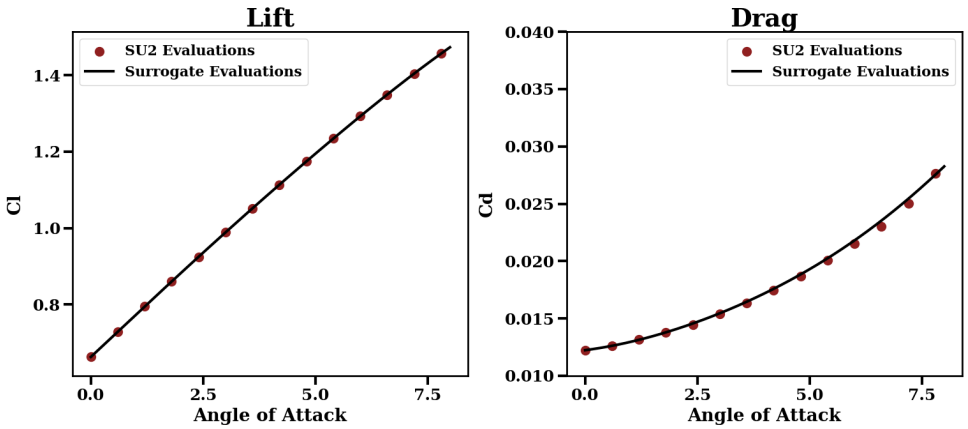

Figure A.98

Latin Hypercube Kriging, Index: 238

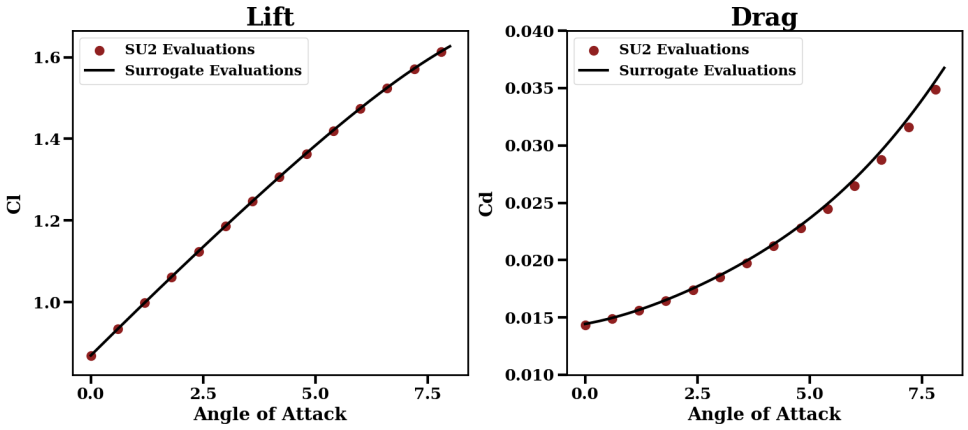

Figure A.99

Latin Hypercube Kriging, Index: 240

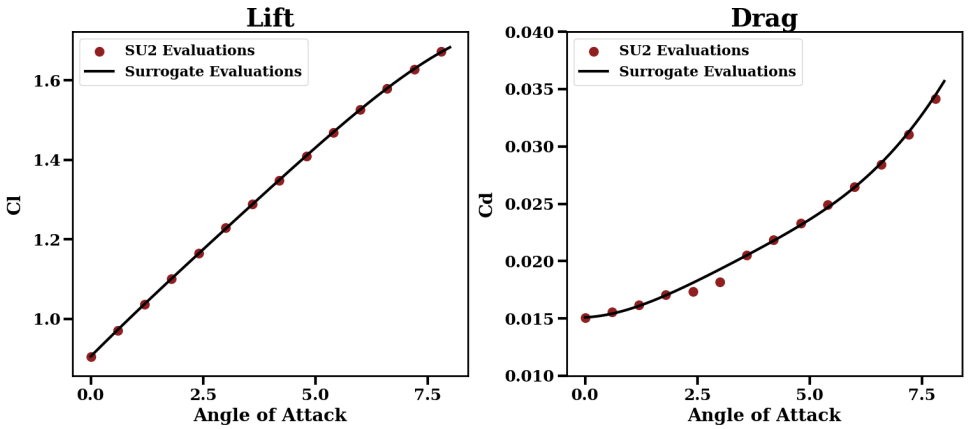

Figure A.100

Latin Hypercube Kriging, Index: 246