PROGRESSIVE VALIDITY METAMODEL TRUST REGION OPTIMIZATION

by

Quinn Parker Thomson

A thesis submitted in conformity with the requirements for the degree of Masters of Applied Science
Graduate Department of Aerospace Engineering
University of Toronto

Copyright © 2008 by Quinn Parker Thomson
Abstract

Progressive Validity Metamodel Trust Region Optimization

Quinn Parker Thomson
Masters of Applied Science
Graduate Department of Aerospace Engineering
University of Toronto
2008

The goal of this work was to develop metamodels of the MDO framework πMDO and to provide new research in metamodeling strategies. The theory of existing metamodels is presented and implementation details are given. A new trust region scheme — metamodel trust region optimization (MTRO) — was developed. This method uses a progressive level of minimum validity in order to reduce the number of sample points required for the optimization process. Higher levels of validity require denser point distributions, but the reducing size of the region during the optimization process mitigates an increase the number of points required. New metamodeling strategies include: inherited optimal latin hypercube sampling, hybrid latin hypercube sampling, and kriging with BFGS. MTRO performs better than traditional trust region methods for single discipline problems and is competitive against other MDO architectures when used with a CSSO algorithm. Advanced metamodeling methods proved to be inefficient in trust region methods.
Acknowledgements

I would first like to thank Dr. Martins for allowing me to work with him and more importantly for giving me special considerations. From allowing an early start to allowing me to finish from Alberta, Dr. Martins has shown an unyielding dedication to his students.

I would also like to thank my wife Kate and my daughter Isabella for giving me the support and motivation to finish my degree quickly.
Contents

1 Introduction 1
   1.1 Overview .................................................. 1
       1.1.1 Curse of Dimensionality ............................ 2
       1.1.2 Bilevel MDO Architectures ....................... 3
   1.2 Motivation ................................................. 3
   1.3 Research Overview ....................................... 4
   1.4 Contributions ............................................. 5

2 Background and Theory 6
   2.1 Metamodeling .............................................. 6
       2.1.1 Experimental Design .................................. 7
       2.1.2 Metamodel Type ....................................... 11
       2.1.3 Validation ............................................. 19
   2.2 Approximation Model Management ......................... 21
       2.2.1 Typical Model Management ......................... 21
       2.2.2 Progressive Validity Metamodel Management ....... 23

3 Implementation 29
   3.1 Outline of the Method .................................... 29
   3.2 Experimental Design — ED class ....................... 30
   3.3 Metamodel Construction — metamodel class ............. 31
List of Tables

4.1 Success rate of the metamodels during comparative trials of MTRO . . . 47
# List of Figures

1.1 Diagram of bilevel optimization architecture .............................................. 3

2.1 Early experimental designs used with physical trials [1] .............................. 7

2.2 Rectangular grid point sampling ................................................................. 8

2.3 Examples of valid LHS for a two dimensional design space ......................... 9

2.4 Optimal latin hypercube sampling [2] ......................................................... 10

2.5 Inherited latin hypercube sampling [3] ....................................................... 11

2.6 The time required to build each of the three metamodels listed with an increasing number of points ................................................................. 26

2.7 MTRO example with crazy function (described in results section): exact contour plot shown under an active trust region contour plot, outlines of previous trust regions, and progress of the optimization on each trust region 27

3.1 Example UML class ..................................................................................... 30

3.2 Experimental design UML diagram ............................................................ 31

3.3 Metamodel UML diagram .......................................................................... 32

3.4 2nd order PR UML diagram ...................................................................... 34

3.5 Kriging UML diagram ................................................................................ 35

4.1 Comparison of exact and approximate representations of the crazy function 40

4.2 Exact plot of the 2 dimensional Mullur function .......................................... 41

4.3 Exact plot of the 2 dimensional Ackley function .......................................... 41
4.4 Exact plot of the 2 dimensional quartic function ........................................ 42
4.5 Results of comparative experimental design trials ................................. 44
4.6 Results of graphical comparison of the crazy function ............................. 46
4.7 Results of comparative metamodel type trials using MTRO ...................... 48
4.8 Results of comparative metamodel type trials with traditional trust region optimization .......................................................... 51
4.9 Comparison between MTRO and traditional trust region method considering parallel computing and the three major types of metamodel ............ 52
4.10 Time of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS ................................. 54
4.11 Number of trust regions of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS ....................... 55
4.12 Number of original model computations of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS .............................................................................................................. 55
4.13 Time delay added to discipline analyses to simulate more costly computations with varying MDO optimizations of analytic function from Sellar et al. [4] .................................................................................................................. 58
Chapter 1

Introduction

1.1 Overview

Research and industry will always be finding more complex analyses which require vast amounts of computing power and time. As our understanding of the world increases and technology progresses, more variables enter into engineering equations and must be modelled in order to better reflect the actual phenomena. Consider computational fluid mechanics where even with use of large clusters of processors, the time required to reach a solution is often measure in hours and days. Metamodels are a tool intended to replace these expensive calculations with a less costly approximation based off the original model. It should be noted that metamodels will not typically be an efficient solution where the original model is relatively quick, other methods are better.

A metamodel is a “model of a model” [5, 6] and is an approximation to the original model; they are often called surrogate models or response surfaces. The original model is evaluated at a number of locations, and the resulting information is used to create the approximation. The sampling of the original model can easily be done in parallel, which is a distinct advantage of this method. Metamodelling can be divided into three distinct components: the experimental design, the metamodel type, and the validation...
method [6]. The experimental design is how points are selected in order to properly capture the desired properties. The metamodel type is how the points and the results therefrom are used to create an approximation. The validation method is how the accuracy of the metamodel is evaluated in order to ensure that the metamodel reflects the actual model. A detailed explanation of all components of metamodeling will be covered in Chapter 2.

Recent research in metamodels has focused on improving its accuracy and computational speed. Clarke et al. [7] uses support vector regression (SVR), a relatively new type of metamodel, to reduce the overall time of computation while increasing the accuracy of the method. Nelson et al. [8], Gano et al. [9], and Rodriguez et al. [10] both use variable fidelity models in the construction of their metamodels in order to reduce overall computation cost without sacrificing on accuracy. Mullur and Messac [11] extended the typical radial basis function (RBF) metamodel with non-radial functions to add more flexibility in the RBF method. Lee et al. [12] used the Broyden–Fletcher–Goldfarb–Shanno (BFGS) quasi-Newton method in order to reduce the number of points needed in the creation of their metamodel, resulting in a quicker method.

1.1.1 Curse of Dimensionality

The main challenge involved with metamodels is the curse of dimensionality [13], whereby the cost to build a metamodel is more than linearly proportional to the number of design variables. This escalation of cost with the number of design variables becomes even more important when multidisciplinary optimization (MDO) problems are considered, as they typically increase the number of variables being handled. Even the simplest metamodels used in practical applications require a trial set of points on the order of number of design variables squared. Lee et al. [12] used a BFGS method to reduce the number of points required for a polynomial regression metamodel to the order of number of variables, consequently decreasing the time required to build the metamodel. Nelson et al. [8] split
the design space into several sections, limiting the total number of points in each section, in order to decrease the total time required to create a global metamodel.

1.1.2 Bilevel MDO Architectures

Optimization problems typically consist of an optimizer dealing directly with the objective and constraint functions. When multiple disciplines are included in an MDO problem, this single level of interaction continues in the majority of MDO architectures. Metamodels change this relationship; they are used as an intermediary in the optimization process. Instead of the optimizer communicating directly with the target model, it communicates with the metamodel which provides much faster interactions. The target model is used to create the metamodel, but then is not used during the optimization process. This dual layer of communication creates what is called bilevel optimization architectures and is illustrated in Figure 1.1. Two examples of bilevel MDO architectures can be found in Sobieszczanski-Sobieski’s concurrent subspace optimization (CSSO) [14], and Sobieski and Kroo’s collaborative optimization using response surfaces (CORS) [15].

![Figure 1.1: Diagram of bilevel optimization architecture](image)

1.2 Motivation

The MDO framework $\pi$MDO, created by Martins, Marriage, and Tedford [16, 17], is a general MDO platform from which easy prototyping and testing can be done. This current work is to replace the limited metamodel functionality in $\pi$MDO with advanced
metamodels capable of giving the best performance available. With advanced metamodels added to πMDO, new work can be done using bilevel MDO architectures and accurate comparisons can be done between single level and bilevel methods. Now the task becomes to create a robust and efficient metamodel; unfortunately, no clear consensus exists on which metamodel is best [18].

Several major metamodel types have been studied and compared [5, 19, 20], but each offers different advantages. Furthermore, each major metamodel has had numerous variants, each of which supposes to offer improved performance. In order to find the best metamodeling strategy, several different experimental designs and metamodel types will be selected and tested. A definitive metamodel choice will be made for use with πMDO.

This work will make use of parallel computing. While many in academia prefer merely mentioning that parallelization is easy, it will actually be done here. The parallelization needs to be robust and capable of working with any hardware configuration. Parallel computing is essential to the viability of metamodels.

A companion to advanced metamodels in an optimization problem is a trust region update scheme. The trust region update controls the bounds of a metamodel throughout the optimization process. The behaviour of the trust region has enormous effects on the overall performance of a bilevel optimization method. Consequently, the search for the best metamodel for πMDO must also include treatment of the best trust region scheme. New work will be done in linking the validity of the metamodel to the trust region update.

### 1.3 Research Overview

The following document will provide details of the development and testing of advanced metamodels for the πMDO architecture. A complementary trust region update scheme will also be developed which will work in conjunction with the metamodel. The background and theory will be presented in Chapter 2. The details of the implementation
will be covered in Chapter 3. Sections 4.2.1 and 4.2.2 present the comparison of various experimental designs and metamodel types, resulting in a conclusive choice on the metamodeling strategy needed for πMDO. Sections 4.2 and 4.3 present the performance of the metamodel trust region optimization methods for single and multiple disciplinary optimization when compared to existing methods in πMDO.

1.4 Contributions

Two main contributions arise from this work. The first is the development of advanced metamodels for πMDO and the subsequent analysis to find the optimum metamodeling strategy. The second contribution is the development of a metamodel trust region optimization scheme which integrates the metamodel validity measurements into the trust region update cycle. The end result of this work is a version of πMDO which allows for prototyping of bilevel MDO architectures as well as the more efficient solution of expensive optimization problems through the use of parallel computing.
Chapter 2

Background and Theory

2.1 Metamodelling

This section is intended to provide some background to the metamodelling field as well as introduce the theory needed for the implementation of the current work. The experimental designs and metamodel types were chosen for the current purpose of an efficient bilevel optimization scheme which can avoid most of the influence of the curse of dimensionality.

Metamodelling can be separated into two separate eras: the physical testing era and the computational experiment era. Initially metamodels were used only to get variable influences, and were never intended to give an accurate representation of the actual system. This was done to limit the amount of expensive physical testing being done as well as to allow the solution to be found efficiently using non-computational methods (since they were not yet available). As computers became available and conducting many experiments became less costly, metamodelling started moving toward space filling data sets and being globally accurate. Now the metamodel was beginning to be capable of being a surrogate model for the original model in an optimization problem. This progression between the two eras can be seen through sections 2.1.1 and 2.1.2. The third component of metamodelling, validation, is covered in Section 2.1.3.
2.1.1 Experimental Design

Experimental design, also known as design of experiments, is the selection of a given number of points within a given set of bounds. The selection of points began with very few points located at the bounds, as can be seen in Figure 2.1. Initially this involved surveying all extremes of the design space, which can be seen in Figure 2.1(a), and is called a full factorial array.

The reduction of the number of sample points from this full factorial into a fractional factorial array. This degrades the accuracy of the metamodeling by potentially missing important trends with unbalanced point distribution. Taguchi arrays were the placing of a fractional factorial array of points into a balanced set of orthogonal points [1]. This limited balancing offers more certainty in capturing most of the important trends in the design space. Figure 2.1(b) shows an orthogonal array for three design variables.

![Figure 2.1: Early experimental designs used with physical trials [1]](image)

As computers became more readily available, experimentation became less costly and now space filling experimental designs started to be used. The simplest of space filling experimental designs is rectangular grid point sampling where the design space is filled with a grid of points, as can be seen in Figure 2.2. This primitive experimental design would require many more trial evaluations than what is necessary to capture the essential
behaviour of the model.

Similar to the physical testing era, scientists desired to reduce the number of points in the trial set. Plain Monte Carlo sampling involves using a random number generator to select the points and is the most basic reduction to the trial set [19]. While computationally efficient, Monte Carlo sampling provides no robustness in finding a space filling set of points.

![Rectangular grid point sampling](image)

Figure 2.2: Rectangular grid point sampling

Stratified sampling or sequences were the next sophistication of experimental design aimed at creating a balanced set of points. Sequencing selects its points based on a specified algorithm with are typically capable of spanning multiple dimensions. Nelson et al [8] provides a comparison of several types of sequence based point selection routines, including Halton, Sobol, Faure, and Niederreiter sequences. Nelson et al shows the degradation of the Halton, Faure, and Niederreiter sequences at higher dimensions; they fail to provide a balanced set of points. The Sobol sequence [21] was found to be fairly well balanced and was the best sequence in Nelson’s paper.

Another more sophisticated experimental design came in the form of stochastic processes, the most prominent of them being McKay et al’s latin hypercube sampling (LHS) [22] (also termed latin hypercube design). McKay proved that LHS provides more consistent point distributions than random number generators or a given stratified sam-
pling method. LHS is done by dividing each dimension into segments, and the number of segments will be the number of points desired in the trial set. One segment is chosen from each dimension at random to form each trial point, as shown in Figure 2.3(a). Note that this method offers no guarantee that the points will be a balanced set; for example, for a two dimensional problem, a valid LHS set of points is the set of diagonal points as seen in Figure 2.3(b).

![Figure 2.3: Examples of valid LHS for a two dimensional design space](image)

More researchers took note of McKay et al's paper [22] and extended upon his work. Two significant improvements in LHS are mentioned here as they apply to the current research. Park wrote an algorithm which creates an optimal latin hypercube sampling (OLHS) of points [2] which is guaranteed to span the design space and have a well balanced distribution. The algorithm consists of two steps: the filled cell distribution and the inner cell distribution. The first step iteratively tries to switch coordinates between two points, searching for the best distribution of filled cells. The objective function of this optimization is to minimize the logarithm of the determinant of the correlation matrix. This distributes the points throughout the design space with the points located at the center of the cells. The second stage relates to the placement of the points within the cells. While this algorithm does not find a unique optimal set of points for a given set of bounds [2], it
does offer an improved robustness in the distribution of the points. One of Park’s OLHS is shown in Figure 2.4.

![Image](image.png)

Figure 2.4: Optimal latin hypercube sampling [2]

Wang provided an improvement on LHS which facilitates the iterative overlapping metamodel construction typical of a trust region optimization process. Wang developed inherited latin hypercube sampling (ILHS) [3] whereby new metamodels inherit points and point evaluations from previous calculations which are within the bounds of the new region. By reusing previous results, fewer points need to be evaluated for a new set of bounds leading to significant cost reduction when considering that metamodels are typically intended for expensive models. In order to construct a new experimental design, the new region is divided into segments in the same fashion that LHS was. The unoccupied segments are placed in an auxiliary design space. Segments are added to the auxiliary design space in order to have the same number of segments in each dimension. A latin hypercube sampling is done in the auxiliary design space, then remapped to the original design space which can be see in Figure 2.5. Note that a non-latin hypercube sampling could result due to the uneven number of empty segments in the design space, also seen in Figure 2.5.
New Variants on LHS

Since both Park and Wang provided new types of LHS methods, the author of the current work created two new experimental designs in the hopes that the benefits of Park and Wang can be cumulative. These two hybrid experimental designs are created using the ideas of both OLHS and ILHS. The first is called inherited latin hypercube sampling (IOLHS) and uses the first part of Park’s algorithm to find an OLHS set of points in Wang’s auxiliary design space; this does not take into account the inherited points in the original design space. The second experimental design called hybrid latin hypercube sampling (hybridLHS) uses the inherited points in the original design space in Park’s optimization. The inherited points are not allowed to move, but they affect where the new points are placed.

2.1.2 Metamodel Type

The metamodel type and how the data generated from the experimental design is used forms the second component of metamodeling. While the metamodel type comes after the experimental design, the metamodel type requires a minimum amount of information in order to be solved and consequently drives the minimum number of points. Attention will placed on the minimum number of points as it is a main contributor to the growth of
the number of points with a growing number of dimensions; in other words it is a main
cause of the curse of dimensionality mentioned in Section 1.1.1.

Polynomial Regression

Since the earliest metamodels were done without computers, they needed to be mathe-
matically simple; consequently the earliest metamodels were polynomial regressions (PR).
Polynomial regression is often called response surface methodology (RSM), but since an
alternate name of metamodels is response surfaces, the terminology could get confusing
PR was not used. Polynomial regressions can be of any order, but typically they are
second order which makes the regression to be known as least squares regression. Second
order PR combined with a minimum of point evaluations can only give trends to the
model. PR continued to be of use into the computational era. More point evaluations
only made PR to be a better representation of the overall model, but despite additional
points PR is still limited by the order to the polynomial. The second order PR take the
following form:

$$\tilde{y} = \beta_o + \sum_{i=1}^{N} \beta_i x_i + \sum_{i=1}^{N} \beta_{ii} x_i^2 + \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_{ij} x_i x_j$$  (2.1)

Where $\tilde{y}$ is the approximation to the expensive function at $x$, $N$ is the number of design
variables, and $\beta_o, \beta_i, \beta_{ij}$ are the regression coefficients determined by the linear regression
of the polynomial model.

Polynomial regression is still used quite frequently in industry and academia as it is
simple to implement, understand, and compute. Unfortunately, PR still suffers from the
curse of dimensionality as the minimum number of points is $(N+1)(N+2)/2$. This means
that the cost of computation goes up quadratically with the number of dimensions.

As mentioned in Section 1.1.1, Lee et al. [12] provides a reduction to the mini-
mum number of points required. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-
Newton method that Lee uses provides an alternate computation for the cross-quadratic
terms in a second order PR. Mathematically this means that the $\beta_{ij}$ from equation (2.1)
are computed using the BFGS approximation in equation (2.2).

\[ B_k = B_{k-1} - \frac{(B_{k-1}\delta_k)(B_{k-1}\delta_k)^T}{\delta_k^T B_{k-1} \delta_k} + \frac{y_k y_k^T}{y_k^T \delta_k} \] (2.2)

Where \( B_k \) gives the approximation to the cross-quadratic terms \( \beta_{ij} \) where \( i \neq j \), \( \delta_k = x_k - x_{k-1} \) and \( y_k = g_k - g_{k-1} \). The extra information coming through the BFGS method is delivered through previous iterations of an iterative trust region method. BFGS reduces the minimum number of points down to \( 2N + 1 \), which means that the number of points required is now directly proportional to the number of dimensions, effectively dealing with the curse of dimensionality arising from the metamodel type.

**Kriging Metamodel**

Now that metamodeling was firmly established in the computer era, the capacity now existed to offer global metamodels. The kriging metamodel is widely used [5, 23, 2], and has been shown to be a generally accurate metamodel [24, 5]. The kriging metamodel was created by Georges Matheron in 1971 in the field of geostatistics and named after South African geologist by the name of Krige [9]. Kriging was first used in metamodeling by Currin et al. [25, 9] who termed design and analysis of computer experiments (DACE).

Three different types of kriging metamodels exist which differ in how they treat the mean and how the metamodel is constructed [26]. Ordinary kriging is composed of a single term and assumes a mean of zero. Universal kriging is a general form of kriging where an unknown linear combination of functions is used to predict the mean [26]. Detrended kriging is a type of universal kriging, but limits the determination of the mean to a linear regression. In order to create a metamodel tool complementary to \( \pi \)MDO capable of handling any function, no assumption the mean can be made, so ordinary kriging will not work. Computation has to be made automatically so the modeling of the mean has to be done by a known set of functions and linear regression is the simplest way to construct the mean, so detrended kriging is the method to use. More detailed information on kriging can be found in Cressie [27] and Olea [28].
Sacks et al. [29] provides a detailed layout of a computational kriging method, which will be used in the current work and presented here. Martin and Simpson also presented this method [26]. The metamodel approximation $Y(x)$ seen in equation (2.3) is a combination of a linear regression, which models the mean, and a stochastic process $Z(x)$, which has a mean of zero and models the deviations from the mean. The stochastic process has covariance $V(w, x) = \sigma^2 R(w, x)$ where $\sigma$ is the standard deviation and $R(x)$ is the correlation.

$$Y(x) = \sum_{j=1}^{k} \beta_j f_j + Z(x) \quad (2.3)$$

From this high level description, Sacks et al. [29] goes on to detail the implementation of the best linear unbiased predictor which fits this metamodel to the observed data points. A shortened explanation of the implementation follows:

The matrix $F$ encapsulates the results from the PR functions for the sample points used to construct the kriging surface, and the vector $f(x)$ contains the results from the PR functions for the untried data point $x$. The matrix $R = \{R(s_i, s_j)\}$ contains the stochastic-process calculations for the sample points, and the vector $r(x) = [R(x, s_i)]$ contains the stochastic process calculations for the untried data point. The kriging method prediction $\hat{y}$ comes in the following form:

$$\hat{y} = f'(x)\hat{\beta} + r'(x)R^{-1}(Y_s - F\hat{\beta}) \quad (2.4)$$

Where $Y_s$ is the vector containing the sample point evaluations by the expensive function, and $\hat{\beta} = (F' R^{-1} F)^{-1} F' R^{-1} Y_s$ is the generalized least-squares estimate of $\beta$. Unfortunately the calculation of the inverse of the correlation matrix unavoidable and contributes significantly to the cost of the method.

The most critical part of the kriging metamodel construction is the proper choice of correlation function, also known as the kernel function. These spatial correlation functions are crucial to the fit of the metamodel to the original data. Many different correlation functions have been investigated, but the Gaussian function is the most pop-
ular [9, 26]. Lophaven et al. [23] provided an in depth analysis of the Gaussian correlation function alongside an exponential, a spline, and a cubic correlation function. They found that the Gaussian function converges quicker and to a more accurate approximation. Nelson et al. [8] did an analysis between correlation functions during the kriging metamodel construction, from which the most accurate is chosen. They found that a better metamodel can be built with several options for the correlation function; the study did not include the performance consequence of this addition. The Gaussian correlation function can be seen in equation (2.5).

\[ R(w, x) = \prod e^{(-\theta_j|w_j-x_j|^p)} \] (2.5)

Where \(x_j\) and \(w_j\) are coordinates of design variable \(j\) at sample point \(x\) and \(w\). The \(\theta_j\)s are correlation parameters which can be optimized for a more accurate metamodel.

Sack et al. [29] demonstrated that a maximum likelihood estimation of the standard deviation, shown in equation (2.6), can be used in conjunction with the correlation matrix \(R = \{R(w, x)\}\) to find the best correlation parameters. The optimum correlation parameters are found be minimizing \((\det R)^{1/n}\hat{\sigma}^2\). Lophaven et al. [23] provides a pattern search optimization algorithm to find the best correlation parameters; Lophaven’s algorithm will be detailed in Chapter 3. Often, despite the impact on accuracy, the kriging metamodel will be implemented with static correlation parameters, which does decrease significantly the time needed to create the metamodel [9]. Gano et al. [9] determined that for a given minimum accuracy, the correlation parameters do not need to be changed at each iteration; consequently, a metamodel update management scheme was created to manage this choice.

\[ \hat{\sigma}^2 = \frac{1}{n}(y_s - F\hat{\beta})'R^{-1}(y_s - F\hat{\beta}) \] (2.6)

Adding BFGS to Kriging  Since the typical linear regression used to model the behavior of the mean is a second order polynomial regression, the minimum number of points required in Sacks et al.’s [29] kriging metamodel is still \((N + 1)(N + 2)/2;\)
the curse of dimensionality still affects this method. The author of the current work adapted the BFGS method used by Lee et al. [12] to the kriging method used by Sacks et al. [29]. In the new method, Lee’s reduced linear regression, which contains only the $\beta_o$ and the first two summation terms from equation (2.1), is used to create a smaller $F$ matrix from Sacks’ method. This reduced linear regression is used to find an intermediate optimum needed in the $\delta$ term of the BFGS method. Once this optimum is found, the BFGS method allows a computation of the missing cross-quadratic terms. These terms are remapped back to a full size $F$ matrix, which means Sacks’ typical method can be carried out. This method will reduce the minimum number of points required to build the metamodel to $2N + 1$.

**Radial Basis Functions**

Radial basis functions (RBF) are metamodels with a wide range of application fitting functions which are expressed in terms of the Euclidean distance (2.7) centered at the trial point locations $x_i$ [11].

$$r = \|x - x_i\|$$

(2.7)

The metamodel for RBF is constructed through a regression (2.8) of a set of basis functions (2.9) with the results of the original model evaluations (2.10) to find the regression coefficients $\sigma$. The multiquadratic function in equation (2.11) is potentially the most effective basis function used with RBF [11].

$$A\sigma = F$$

(2.8)

$$A_{ik} = \gamma(\|x - x_i\|); \quad i = 1, \ldots, n_p; \quad k = 1, \ldots, n_p$$

(2.9)

$$F = [f(x_1) \ f(x_2) \ \ldots \ f(x_p)]$$

(2.10)
\[ \gamma(r) = \sqrt{r^2 + c^2} \] (2.11)

**Benefits of Metamodels**

Metamodels are versatile and are well suited to a wide array of problems where the amount of information is uncertain or expensive to get. A metamodel is blind to any discontinuities found in the original and consequently smooths them out. This smoothing property of metamodels allows gradient based optimizers to be used with discontinuous problems. The metamodels presented in the previous sections do not require gradient information from the original model, which avoids potentially costly calculations or the need to create the algorithm in the first place. Other metamodels do exist though that can use gradient information to build the metamodel; for gradient use in the building of the kriging metamodel, see Morris et al. [30], van Keulen and Vervenne [31], or for the kriging metamodel, see Chung and Alonso [32]. Another advantage of metamodels is that gradient information can typically be gained from the metamodel, either analytically or through the use of finite difference or complex step.

The main flaw of metamodels is that no matter how many trial points you evaluate, an approximation created by a metamodel will never exactly mimic the original model. Despite this inaccuracy, a solution can be found at nearly the exact location for much less cost with a metamodel.

**Trade-Offs Between Metamodels**

Several comparative studies between metamodels have been carried out in recent years [24, 5, 7]; they will be summarized in this section.

Wang et al. [24] studied five metamodels: RBF, Gaussian processes (GP) (kriging is a type of GP [26]), MARS, PR, and adaptive weighted least squares (AWLS). They found that RBF and GP both produce accurate results, but RBF was noted to be slightly
better overall but GP performed better for noisy problems. In terms of efficiency, GP were found to take significantly longer to calculate than the other four methods.

Jin et al. [5] studied PR, RBF, kriging, and MARS. They found that RBF gave the most accurate results, was relatively efficient, and was the most robust. RBF was found to perform relatively the best for small and scarce sample sets. When considering the kriging metamodel, Jin et al found that kriging performed quite well for large sample sets but deteriorated with a decreasing number of sample points. Contrary to Wang’s findings [24], Jin found that kriging was particularly sensitive to noisy problems and performed poorly. In a quantitative study, Jin found that kriging took significantly longer to calculate than the other three metamodels tested.

Clarke et al. [7] introduced SVR and presented comparative studies to PR, RBF, kriging, and MARS. They found that SVR and kriging performed similarly in terms of accuracy and robustness with SVR being slightly better. Contrary to both Jin et al. [5] and Wang et al. [24], Clarke et al found that RBF gave the worst results for accuracy and was the second least robust. Clarke did note that their results contradicted other studies and were investigating them. Clarke referred to Jin et al’s study on performance [5], but noted that SVR’s efficiency was comparable to MARS which is worse than RBF and PR, but significantly better than kriging.

In terms of the current development of a trust region optimization method, kriging, RBF, and SVR offer different advantages. Out of the three, kriging is the least efficient but offers a robust, accurate method. RBF is generally accurate and is the most efficient, but Clarke et al’s results on accuracy are disconcerting [7]. SVR offers the best accuracy and is the most robust; on top of this, SVR is fairly efficient. Kriging was useful during the development of this project as it offered a robust, accurate solver but will be slower than the other methods. If Clarke et al’s study [7] is correct, SVR will likely be the best metamodel for the current work. RBF should be implemented for comparative studies.
Chapter 2. Background and Theory

2.1.3 Validation

Validation is the third step in metamodeling and is used to determine the accuracy, also known as validity, of the metamodel. While not a step in producing the metamodel, validation is crucial to knowing how much the approximation can be trusted. As statistical testing is considered inappropriate for deterministic computer modeling which lack random error [29], cross-validation methods and mean square error (MSE) methods are required [13].

The simplest method to determine the accuracy of a metamodel is to determine the root mean square error (RMSE), where they metamodel results, \( \hat{y}_j \), are compared to additional evaluations of the original model, \( y_j \), for all \( n \) points, as seen in equation (2.12). The smaller the value of RMSE, the more accurate the metamodel.

\[
RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (\hat{y}_j - y_j)^2}
\]  
(2.12)

Jin et al. [5] offers three additional error measurements, which offer different information each. The R square method seen in equation (2.13) uses the variance, which measures how irregular the problem is, combined with the MSE to produce an error measure. The larger the value of R square, the more accurate the metamodel.

\[
R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y}_i)^2} = 1 - \frac{MSE}{\text{variance}}
\]  
(2.13)

The next error calculation is the relative average absolute error (RAAE) seen in equation (2.14) and involves the standard deviation (STD). RAAE is typically correlated with MSE and the R square error. The smaller the value of RAAE, the more accurate the metamodel.

\[
RAAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n \times STD}
\]  
(2.14)

The last error calculation from Jin et al. [5] is the relative maximum absolute error (RMAE) seen in equation (2.15). RMAE can find large errors in a section of the design
space even though the overall accuracy is high. Note that RMAE does not find the overall performance, just the extreme errors. A small RMAE is preferred.

$$RMAE = \frac{\max\{|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \ldots, |y_n - \hat{y}_n|\}}{STD}$$ (2.15)

The four preceding error measures are useful in determining the accuracy of a given metamodel, but they all require additional evaluations of the original model. Consequently the cost can become significant if the original model is expensive, which is likely because metamodels are intended to be used with more costly models. The use of additional point evaluations makes these error measures only useful for development and comparative purposes, not in-the-loop processing of validity. A cross-validation method is needed to do this.

The most frequently used cross-validation method is the leave-$k$-out cross-validation method [13, 33, 26, 34, 35], or the sub-variant leave-one-out cross-validation. Meckesheimer et al. [6] determined that leave-$k$-out cross-validation provides a reasonable assessment of the validity of the metamodel. The basic procedure involves leaving $k$ points out of the sample set, then rebuilding the metamodel with $n - k$ points, where $n$ is the number of sample points. The new metamodel results are compared to the exact function evaluations from the $k$ points which were left out; a RMSE is found for these $k$ points. This step is repeated, where each sample point is included in the $k$ points for at least one iteration. The RMSE from all steps are averaged to give the final validity measurement. The value $k$ is suggested to be either $k = 0.1n$ or $k = \sqrt{n}$ for kriging and $k = 1$ is suggested for PR and RBF [6]. Leave-$k$-out cross-validation is shown in equation (2.16).

$$RMSE(i) = \sqrt{\frac{1}{k} \sum_{j=1}^{k} (\hat{y}_j - y_j)^2}$$ (2.16)

Wang and Shan [18] argue that leave-$k$-out cross-validation is not an accurate measure of validity, rather that it is a measure of insensitivity to losing points in the trial set. Despite Meckesheimer et al’s study [6] to the contrary, Wang and Shan’s point remains
valid. Wang and Shan give no cross-validation alternatives, and refer to the typical
validation schemes which require additional point evaluations.

2.2 Approximation Model Management

An approximation model management scheme is an algorithm in which the bounds of a
metamodel are managed through an optimization process. Typically, this management
scheme is based on the predicted reduction from the metamodel combined with the
actual reduction from the original model. The most common type of model management
was presented in Dennis and Torczon [36] and is shown below for the minimization of
\( f(x, y(x)) \).

2.2.1 Typical Model Management

Given the original model \( M \), an approximation model \( M^\alpha \), a starting location \( x_\alpha \), the
original model evaluation at the starting location \( y_\alpha^\alpha \), and a set of update parameters \( \eta_1 \)
and \( \eta_2 \) such that \( 0 < \eta_1 < \eta_2 \) for iterative regions \( k = 0, 1, \ldots \); do the following [36]:

1. Check for convergence, otherwise continue

2. Optimize using the approximation model to find \( x_{k+} \) for which a sufficient decrease
   condition is met
   
   Compute the predicted reduction \( \text{pred}_k = f(x_{k+}, y_{k+}^\alpha) - f(x_k, y_k) \)

3. Compute \( y_{k+} = M(x_{k+}) \) using the original model
   
   Compute the actual reduction \( \text{ared}_k = f(x_{k+}, y_{k+}) - f(x_k, y_k) \)
   
   Compute \( r_k = \frac{\text{ared}_k}{\text{pred}_k} \)

4. If \( r_k \leq 0 \) (improvement predicted but not achieved), then
   
   \bullet Set \( x_{k+1} = x_k \) (reject the step)
• Refine the model and allow less optimization to be done

Else if $0 < r_k < \eta_1$ (much more improvement was predicted then achieved), then

• Set $x_{k+1} = x_k$ (Accept the step)

• Allow less optimization to be done

Else if $\eta_1 \leq r_k < \eta_2$ (the prediction was satisfactory), then

• Set $x_{k+1} = x_k$ (Accept the step)

• Allow the same amount of optimization to be done

Else if $r_k \geq \eta_2$ (prediction was excellent, or more decrease was obtained than predicted), then

• Set $x_{k+1} = x_k$ (Accept the step)

• Relax the model and allow more optimization to be done

A trust region method is often considered a more specific type of model management scheme as they are typically related to 2nd order polynomial regression metamodels and use a varying set of bounds or radii in order to control the amount of optimization being done [36] (although trust regions are still used to refer to model management in general). Nocedal and Wright describes a similar trust region approach in their optimization text [37] as well as the use of Cauchy point to determine sufficient reduction in the optimization of a 2nd order PR model. Both Dennis and Turczon [36] and Booker et al. [38] use model management combined with pattern search optimization. The main goal of pattern search hybrid methods is to avoid the construction of a metamodel if one of the trial points produce a better result than the current step in the optimization process. Gano et al. [9] uses variable fidelity models in their model management method.
Problems with Typical Model Management and Pattern Search Optimization

Unfortunately, the typical model management and pattern search methods create inefficiencies when introduced to expensive original models and parallel processing. The typical model management method requires the calculation of the actual reduction for that iteration of the algorithm, shown in step 3 in the previous section. If all trial evaluations were done simultaneously using parallel processing, then this additional evaluation to get the actual reduction doubles the total time segments attributed to calculations involving the original model. With the understanding that metamodels are typically associated with expensive original models, this increase in model calculations virtually doubles the total time required.

Pattern search optimization’s procedure of avoiding the construction of the metamodel is inefficient if the time to construct the metamodel is insignificant when compared to the time to do a calculation with the original model. Since metamodels are intended for expensive functions, the use of pattern search optimization techniques is questionable. If the computational times of the original model and the metamodel construction are similar, then most likely either the original model is not costly enough to warrant metamodels, or the metamodel construction is too inefficient for optimization purposes.

2.2.2 Progressive Validity Metamodel Management

One of the goals of the current project is to introduce a new way to manage metamodels in optimization processes, which is called metamodel trust region optimization (MTRO). Rather than doubling the number of original model evaluations by computing the actual reduction, the new algorithm will use the validity of the metamodel in the update of the bounds of the current region. If leave-$k$-out cross-validation is used, the calculation of the validity will not require additional original model calculations. With the use of parallel computing, the time required for each step of the evaluation could be as low as a
single original model evaluation.

The cooperation between the metamodel and the trust region comes from linking the validity of the metamodel to the trust region progression in order to reduce the number of sample points taken from expensive evaluations. By only requiring a lower level of validity at early stages of optimization, a lower concentration sample points are needed over the trust region. As the optimization progresses, the number of points are increased at each trust region iteration until a minimum level of validity is achieved. Near the end of optimization, the required validity is highest, consequently a larger concentration of sample points are needed. This higher concentration of sample points does not necessarily mean higher numbers of points, as the trust region is getting smaller as the optimization progresses. The MTRO algorithm is as follows:

1. **Initialize**: Initialize the bounds of the trust region and starting point of the optimization.

2. **Construct**: Build metamodels for objective and constraint functions for region of size $h_k$ centered about the current point $x_k$. Use an efficient ED to increase the number of points until a minimum level of validity $V_{min_k}$ is acquired in all metamodels.
   - If too many points are needed, reduce the size of the trust region and restart metamodel construction.
   - If one of the ED points used to create the current metamodel is better than the current point, use it instead.

3. **Optimize over trust region**: Optimize over the bounds of the trust region with the given objective and constraint metamodels.

4. **Update trust region**: Update the size and location of the trust region based on the location of the current point $x_k$ within the current trust region where $h_k$ is
distance between the bounds of the trust region. The terms \( \eta_1 \) and \( \eta_2 \) are used to separate sections within the trust region, where \( 0 < \eta_1 < \eta_2 < 2 \), \( 0 < \xi_1 < \xi_2 < 1 < \xi_3 \), and \( 0 < \zeta_1 < \zeta_2 < 1 < \zeta_3 \).

- If \( x_k < \eta_1 \times h_k \), then \( h_{k+1} = h_k \times \xi_1 \) and \( V_{min_{k+1}} = V_{min_k} \times \zeta_1 \)
- If \( \eta_1 \times h_k < x_k < \eta_2 \times h_k \), then \( h_{k+1} = h_k \times \xi_2 \) and \( V_{min_{k+1}} = V_{min_k} \times \zeta_2 \)
- If \( x_k > \eta_2 \times h_k \), then \( h_{k+1} = h_k \times \xi_3 \) and \( V_{min_{k+1}} = V_{min_k} \times \zeta_3 \)

5. **Test for convergence**: If the conditions for convergence are satisfied, then stop. Otherwise, return to step 2.

The MTRO algorithm contains the same types of steps as the typical model management strategy; namely the initialize, build metamodel, optimize, update trust region, check for convergence. The changes come in steps 2 and 4. Step 2 is the new progressive validity component where points are added to the experimental design until a sufficient level of validity is reached according to the leave-\( k \)-out cross-validation method. Step 4 is a trust region update based on the final location of the optimizer in the trust region. An explanation of step 4 is as follows:

- **Update trust region**: Update size and location of trust region based on the progress from the previous trust region. If the progress is:
  - small: shrink the trust region and increase the needed validity both substantially
  - moderate: shrink the trust region and increase the needed validity both by a small margin
  - substantial: expand the the trust region and decrease needed validity

Note that the trust region never stays the same size, this is to ensure that infinite constructions of the same metamodel with inherited points does not occur.
The only modification to the MTRO algorithm comes from the use of the RBF metamodel. RBF did not work well with the leave-$k$-out cross-validation method and consequently bypasses step 2. To accommodate RBF in the algorithm, a set number of points per design variable are used to build the metamodel; sufficient points are chosen to ensure validity. This course of action is feasible with RBF because, as Figure 2.6 shows, RBF is significantly less costly than kriging as the number of points increases. In addition, RBF has no required minimum number of points. The PR used in MTRO does not use this strategy because of the inability to conform to more trial points and because the minimum number of points scales quadratically with number of design variables.

![Figure 2.6: The time required to build each of the three metamodels listed with an increasing number of points](image)

An example of this algorithm is shown in Figure 2.7.

Note that both the objective and the constraint functions need to be approximated by a metamodel since both may be expensive functions to calculated. It may be possible to have a method to detect whether or not a function requires a metamodel, but this is not included in MTRO.
The process of increasing the number of points and reconstructing the metamodel recursively until a desired level of validity is achieved can become expensive if the metamodel construction is costly. This cost can be alleviated by increasing the number of points added each time more validity is needed. A larger number of points can be added with little extra cost with the use of parallel processing. Further details on parallel processing is included in the implementation section of this document.

The trust region optimization algorithm fits perfectly with metamodeling as both use strict bounds in their execution. The trust region cannot extend past the metamodel bounds or else the optimization will be getting invalid results, and expensive function evaluations would be wasted. If the bounds are matched between metamodel and trust region, then no erroneous or inefficient processes take place due to the metamodel-optimization coupling.

In order to comply with the underlying goal of faster optimization, MTRO sacrifices
the ability to be an accurate global optimizer. No matter how large the initial trust region is, the sample point spacing is still very coarse and cannot guarantee that the global optima is found. If the validity required by the initial step is higher in order to create an accurate global optimizer, its cost could become prohibitive, and other strategies such as Nelson et al’s partitioning strategy [8] should be used. MTRO gains some level of global optimization by making the start point of the optimization of each trust region be the minimum of either the lowest exact value at the sample points or the end point of the previous optimization. While a typical optimization has no knowledge of the design space, metamodels know the exact value of the objective function at the sample points generated by the experimental design.
Chapter 3

Implementation

3.1 Outline of the Method

The MDO framework πMDO is an object oriented program with multiple levels of inheritance. It contains a single level optimization class named $\text{OptProb}$ which gets inherited by a general multidisciplinary optimization class named $\text{MdoProb}$. This MDO class gets inherited by a number of architectures which are used to solve MDO problems. Metamodels are used in two different ways within πMDO. They can be used in a single discipline optimization structure, which is also used in some MDO architectures for system level optimization or subsystem optimization. Metamodels are also used as a surrogate evaluation in simultaneous subsystem optimizations where dependency precludes non-metamodel methods.

Both of these uses could use all the components discussed in Chapter 2. The SDO structured problem is a straightforward application of the model management scheme with all the components of metamodeling. The surrogate use of metamodels may also require a model management method, but this method will be custom designed for the given MDO architecture. The three components of metamodeling are comprised into two sections. All experimental design types are included in one class, described in Section 3.2.
The metamodel construction and validation are contained within a generic metamodel class with derived children classes specific to each metamodel type; these are described in Section 3.3. The implementation of the model management is described in Section 3.4 with visualization described in Section 3.4.1.

The code will be represented through the use of unified modeling language (UML) class diagrams, as shown in Figure 3.1.

![Example UML class diagram](image)

Figure 3.1: Example UML class

### 3.2 Experimental Design — ED class

The experimental design class contains all the point selection techniques discussed in Section 2.1.1. The functions can be seen in the UML class diagram in Figure 3.2. The implementation of each ED is done how its described in Section 2.1.1, with the exception of OLHS. The implementation of OLHS only uses Park’s first stage of optimization [2] as it provides a well-balanced set of points without the cost of the second stage. Each of the methods normalizes the trial points to a specific range in order to avoid calculation overflows in the kriging correlation evaluations. The two private functions, `createLHS` and `createOMLHS`, are used by the other methods to create unnormalized sets of random and optimized points respectively; LHS and ILHS use `createLHS`, and OLHS, and one of the new variants use `createOMLHS`.

While both new ED use both optimal point distribution and inherited point reuse, they differ in how this is accomplished. The first hybrid method is called inherited optimal latin hypercube sampling (IOLHS), and uses `createOMLHS` on the auxiliary design space.
Chapter 3. Implementation

The second hybrid method is called hybrid Latin Hypercube sampling (hybridLHS) and uses the same algorithms found in both OLHS and ILHS, but combines them such that the inherited points affect the placement of the optimized points in the auxiliary design space.

![ED class function points_select](image)

**Figure 3.2: Experimental design UML diagram**

The ED class function `points_select` is a generic point selection function which allows the ED to be chosen from within the class, no matter how many times an experimental design is chosen. Once the best ED is found by experimentation, it can be set into `points_select` for use. If a new experimental design arises, it can be coded into just this one function and subsequently used in model management optimization schemes; in other words, `points_select` allows for rapid prototyping.

### 3.3 Metamodel Construction — metamodel class

The second and third components of metamodeling, the metamodel type and validation, are contained within the same class. The generic tools needed to all metamodels are
placed within the base class titled `metamodel`, as shown in Figure 3.3. The specifics of each metamodel type are then placed in the inherited classes `kriging` and `polyreg2`.

```plaintext
metamodel
+vars: dict
+dvar_keys: list
+mpi_eval_name: string
+Ndvars: int
+BFGS: bool
+RS_point_results: array
+norm_RS_point_results: array
+k_sat: int
+k_nonsat: int
+n: int
+results_mean: scalar
+results_scale: scalar
+num_meta: int
+old_x: array
+old_value: array
+mpi_eval_points(vars:dict, var_name: string, len_to_eval: int)
+function_eval(vars:dict)
+leave_k_out(k: int, point: ED, upperlimit: dict, lowerlimit: dict, scaled: bool=True)
+meta_contour(var1: string, index1: int, var2: string, index2: int, upperlimit: dict, lowerlimit: dict)
+meta_contour_inloop(var1: string, index1: int, var2: string, index2: int, upperlimit: dict, lowerlimit: dict, numContours: int, numPoints: int)
+infoUpdate(opt_x: dict, opt_value: real)
```

Figure 3.3: Metamodel UML diagram

The main attributes and functions from within the `metamodel` are related to the validation, visualization, parallel processing, or BFGS update. The leave-\(k\)-out cross-validation requires the rebuild of the metamodel with a reduced set of points, but the original metamodel is still desired after the validation has been completed. Consequently, the original metamodel information is stored before the validation is carried out, and copied back to the original variables afterwards. While the variables copied are not involved in any calculation in the `metamodel` class, they will be involved in every possible derived metamodel class. Any metamodel specific terms which would be modified by the rebuild need to copied and retrieved need to be placed in the functions `store_data` and
retrieve_data in every metamodel.

The visualization functions meta_contour and meta_contour_inloop are automatic contour plotters for the given dimensions and will be described in Section 3.4.1.

The two functions mpi_eval_points and function_eval relate to the parallel evaluation of the trial points and will be described in Section 3.3.4.

The boolean BFGS is flag on whether the BFGS approximation is used and the function infoUpdate updates the previous iteration information needed by the BFGS algorithm.

The metamodel construction is able to construct multiple models when an original model function provides multiple calculations which all need to be modelled. These multiple models are used to provide approximations to constraints in the optimization process. This metamodel class is also able to handle array input, which is also what πMDO is able to handle. Both the input and output of the computations are normalized in order to control overflow and underflow errors in this general metamodel framework. When point evaluations are made, the input values are converted to the normalized set and the output is converted back to the range of the original model.

3.3.1 Second Order Polynomial Regression — polyreg2 class

The polyreg2 class is a second order polynomial regression with optional implementation of Lee et al’s BFGS update [12] to approximate the cross-quadratic term, as shown in Figure 3.4. The Numpy [39] Python scientific computing package provides a least squares solver in order to compute the regression coefficients regCoeffs from the F_matrix and norm_RS_point_results. From these regression coefficients, the constant, linear, quadratic, and cross-quadratic terms are extracted to be used in the second order polynomial regression. If the BFGS boolean is true, the cross-quadratic terms are provided by the BFGS update.
3.3.2 Kriging Metamodel — kriging class

The kriging class starts with an identical implementation of the PR found in the previous section, including the BFGS update. The BFGS kriging implementation created by the current author begins with Lee et al’s reduced $F_{\text{matrix}}$ \[12\], which finds the constant, linear, and quadratic regression coefficients. The kriging method needs a full size $F_{\text{matrix}}$, which is found by converting the BFGS approximate cross-quadratic coefficients back to a full sized $F_{\text{matrix}}$, as described in Section 2.1.2. The UML diagram for kriging can be seen in Figure 3.5.

Pattern Search Optimization

Following the regression, the correlation parameters $\theta$ are found using Lophaven et al’s pattern search optimization \[23\]. This method optimizes the maximum likelihood estimation of the standard deviation, as shown in equation \(2.6\). A brief outline of the algorithm operates is shown below, Lophaven et al has a detailed explanation \[23\]:

1. Initialize required correlation parameters, bounds, and temporary parameters

2. Perform an exploratory step at a distance $\delta$ from the current set of parameters
using the function \texttt{theta\_explore}

3. If the step improves the maximum likelihood estimate of the standard deviation, keep stepping at step size \texttt{delta} in the same direction until the maximum likelihood estimate is not improved using function \texttt{theta\_move}

4. Reduce the step size \texttt{delta}

5. If the change in the correlation parameters is sufficiently small or a maximum number of iterations is reached, then stop. Otherwise return to step 2

The correlation parameters found by this optimization are then stored in \texttt{RS\_weights} and \texttt{Corr\_weights}. These two arrays are all that is needed in the \texttt{evaluate\_points}
3.3.3 Interface to pyACDT

The radial basis function implementation used in MTRO is found in the aircraft design toolbox pyACDT [40]. MTRO has a general interface to the metamodel component pyMMT of pyACDT. Currently, only RBF is available through this general interface but any metamodels can easily be added. The experimental design component of pyACDT titled pyDOE is also given a general interface within the ED class.

3.3.4 Parallel Processing

The parallel processing of the trial point evaluations are done using a master-slave algorithm. A master-slave algorithm employs a single processor as the master processor, which distributes trial points to the many slave processors and subsequently collects the results. The main advantage of the master-slave algorithm is that it can be used across a wide array hardware configurations without changes. This hardware flexibility coincides with one of the purposes of πMDO, which is that it is a general platform open to many different researchers at different locations.

The implementation of the parallel processing is accomplished using the message passing interface (MPI) method [41, 42], which is the standard method of parallel processing across industry and academia. A first release of the standard was done in 1994 with the release of MPI1 [41], which included basic communication tools. The second standard MPI2 [42] introduced dynamic process management in which MPI processes can create and kill other MPI processes.

A direct library between MPI and python is not available, but many python modules do exist which use C programming to establish the link. An MPI based python interpreter called pyMPI [43] is widely used, but only offers MPI1 functionality which limits the ability of a master-slave algorithm to generate a desired number of slaves. An MPI
python module which works with the normal python interpreter titled mpi4py adds the
dynamic process management contained in the MPI2 standard [44]. The dynamic process
management allows spawning and killing of new processes which means that processors
used in the master-slave algorithm are available when the algorithm is not running.

3.4 Trust Region Optimization

The single discipline optimization using metamodels is done using a metamodel version
of πMDO’s OptProb. The SNOPT optimizer [45], a sequential quadratic programming
(SQP) optimization algorithm, is used in conjunction with its Python wrapper pySNOPT
to perform an optimization on the metamodel. The implementation of SNOPT into a
model management was relatively simple as SNOPT is already used in πMDO. SNOPT
requires the setting of bounds on the optimization, consequently using the bounds of the
trust regions fits perfectly with SNOPT. The trust region optimization scheme replaces
the simpler optimization scheme of the original OptProb. Other than the optimization,
the new OptProbMeta uses all the functions previously found in the original OptProb in
πMDO. For more information on πMDO, refer to Tedford’s thesis [46].

The traditional trust region optimization method constructed for comparison pur-
poses comes directly from Nocedal and Wright [37] and follows the method described in
Section 2.2.1.

3.4.1 Visualization of Metamodels and Trust Regions

The real time visualization throughout the optimization process aids in the monitoring
of the progression of the optimization and in the diagnosis of unwanted behaviour. The
general metamodel functions meta_contour and meta_contour_inloop both provide con-
tour plotting through the Python module matplotlib [47]. They have arguments which
select the dimensions which are to be placed on the contour plot. The difference between
the two contour plotting functions is that \texttt{meta\_contour} is used for detailed plotting and requires hardcoded options for the number of contours and number of points in the metamodel point evaluations, and \texttt{meta\_contour\_inloop} provides real time contour plots for the given trust region with arguments for number of contours and point evaluations.

Two other plotting functions are contained in the \texttt{OptProbMeta} which provide plotting of the bounds of the trust region, plotting of the SNOPT progress, and management of the contour plotting. The function \texttt{stepplot} plots the progress of the SNOPT optimization. The function \texttt{trustRegionPlot} offers many different options for plotting; it can plot boxes around the trust region overlapping on top of the contour plots for the metamodel, plot a transparent copy of the exact function contour plot, and manage what the plot window is showing.
Chapter 4

Results

4.1 Evaluation Problems

Four different functions are used to compare the various elements of metamodeling presented in this document. The first, the *crazy function*, is a creation of the Dr. Martins and is defined in equation (4.1). This is a piecewise function with a discontinuity between the Rosenbrock function and a linear combination of a quadratic function and a trigonometric function. This problem shows that metamodeling inherently smoothes discontinuities, which allows the use of gradient based optimizers, such as SNOPT. The exact crazy function contour plot can be seen in Figure 4.1(a), while a 40 point OLHS kriging metamodel shown in Figure 4.1(b) demonstrates the smoothed discontinuity. The crazy function is bounded by \(-1.5 < x_i < 1.5\) with a minimum at \(x_i = 1.5\). The starting point was chosen to be \(x_0 = -0.5\) and \(x_1 = 0.5\).

\[
f(x) = \begin{cases} 
100(x_1 - x_0^2)^2 + (1 - x_0)^2 - 200 & x_1 < -|0.8x_0 - 0.4| \\
400(x_0^2 - x_0x_1 - 0.2x_1^4) + 1000 \sin(x_1^2 - x_0^3) & \text{else}
\end{cases} \tag{4.1}
\]

The second function used comes from Mullur and Messac [11] and is a highly nonlinear function. This function is referred to as the Mullur function, and is shown in equation (4.2)
Chapter 4. Results

Figure 4.1: Comparison of exact and approximate representations of the crazy function

(a) Exact crazy function

(b) Kriging metamodel of crazy function with a 40 point OLHS ED

and Figure 4.1. This function is an n-dimensional problem, which means it can be
scaled to any number of dimensions. The bounds are \(-1 < x_i < 1\) with a minimum at
\(x_i = 0.446\). The starting point was chosen to be \(x_i = -0.4\).

\[
f(x) = \sum_{j=1}^{n} \left[ \frac{3}{10} + \sin \left( \frac{16}{15} x_j - 1 \right) \sin^2 \left( \frac{16}{15} x_j - 1 \right) \right]
\] (4.2)

The third function is the Ackley function, as shown in equation (4.3) and Figure 4.1.
The Ackley function is used by Affenzeller and Wagner [48] to test genetic algorithms,
and is bounded by \(-32.768 < x_i < 32.768\) with a minimum at \(x_i = 0\). It is an n-
dimensional problem with plenty of local minima to test the ability to globally optimize.
The starting point was chosen to be \(x_i = -25.0\).

\[
f(x) = 20 + e^{-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n} \sum_{i=1}^{n} \cos 2\pi x_i}
\] (4.3)

The fourth function is used to test how MTRO and the various metamodels perform
with an increasing number of dimensions. This function is a simple n-dimensional quartic
function given in equation (4.4) and Figure 4.1. The quartic function is bounded by
Figure 4.2: Exact plot of the 2 dimensional Muller function

Figure 4.3: Exact plot of the 2 dimensional Ackley function
Chapter 4. Results

42

Figure 4.4: Exact plot of the 2 dimensional quartic function

\[-1.5 < x_i < 1.5 \text{ with a minimum at } x_i = 0. \text{ The starting point was chosen to be } x_i = 1.0.\]

\[f(x) = \sum_{i=1}^{n} x_i^4\]  \hspace{1cm} (4.4)

An important thing to note is that these functions are not expensive to calculate, contrary to the main purpose of metamodels. Any time measurements made of the optimization progress reflects the cost of the metamodeling and trust region management, and not the original model evaluations. The expected cost of similar but more expensive functions can be approximated by the number of trust regions and original model evaluations required to complete the optimization.

The following trials were done multiple times, then averaged to find a typical behaviour of the method. This is needed due to the stochastic nature of the experimental design. The method is not perfected yet, and rarely results in excessive computation times. To demonstrate the expected behaviour of the method, these excessively long trials were not included in the following results. Parallel processing was not used in the
computation of the following results except where specifically mentioned.

4.2 Comparing Single Discipline Optimization Problems

4.2.1 Effect of Experimental Designs on MTRO

The five latin hypercube sampling routines are tested using the MTRO algorithm on the crazy, two dimensional Mullur, and the two dimensional Ackley. Note that the LHS and OLHS ED methods do not benefit from inheriting old points from the metamodels constructed with insufficient validity, rather a completely new set is evaluated every time more points are needed. This method would be costly in practical applications, as can be seen from the higher number of original model evaluations in Figure 4.5(c), but in this setting it allows the assessment of the inherited EDs. As can be seen in Figure 4.5(b), the number of trust regions for all EDs is relatively similar. Consequently, it can be concluded that the inheritance of points and occasional improper latin hypercube set of points has little effect on the overall process of MTRO.

The main difference between the three inherited latin hypercube sampling methods occurs in the time taken for the MTRO process for the crazy function, as shown in Figure 4.5(a). Both ILHS and hybridLHS took considerably longer than the other three EDs. Since the problem, metamodel type, and optimization method were all identical between the trials, it can be concluded that IOLHS is the best inherited sampling scheme presented in this paper. One possible cause of IOLHS’s better performance could rest in the optimization of the points ensuring a more balanced set of design points, resulting in better predictions of the design space.
(a) Time of MTRO evaluations on various problems with varying EDs using the kriging metamodel

(b) Number of trust regions of MTRO evaluations on various problems with varying EDs using the kriging metamodel

(c) Number of original model computations of MTRO evaluations on various problems with varying EDs using the kriging metamodel

Figure 4.5: Results of comparative experimental design trials
4.2.2 Comparing Metamodels

Graphical Comparisons of Metamodel Types

This section gives a graphical comparison of the three major metamodel types available in the current work. Each metamodel is given the same experimental design to fit the crazy function, given in Section 4.1. While this is not a definitive test, it does demonstrate some key features of the metamodels.

The most important feature to note is the inability of the PR metamodel to fit more points. While the PR metamodel only captures the most basic features of the function, both the kriging and RBF metamodels are able to approximate more features of the original function. This differentiates between simple metamodels and advanced metamodels.
Chapter 4. Results

Figure 4.6: Results of graphical comparison of the crazy function

(a) PR with 10 points  
(b) Kriging with 10 points  
(c) RBF with 10 points

(d) PR with 20 points  
(e) Kriging with 20 points  
(f) RBF with 20 points

(g) PR with 30 points  
(h) Kriging with 30 points  
(i) RBF with 30 points

(j) PR with 40 points  
(k) Kriging with 40 points  
(l) RBF with 40 points
Effect of Metamodel Type on MTRO

Five metamodels are being tested with MTRO, two distinct types, their BFGS counterparts, and RBF. The first result to note in Figure 4.7(a) is the difference in computation time required between PR and kriging. This is expected due to the explicit inverse in the construction of the kriging metamodel. The interesting result here is that the number of trust regions (Figure 4.7(b)) are relatively similar between all five methods for each problem and the number of original model evaluations (Figure 4.7(c)) only varied significantly for RBF, owing to its inability to use the cross-validation portion of the MTRO algorithm. The higher accuracy and versatility of the kriging or RBF metamodels did not reduce the number of trust regions in MTRO. Given the huge disparity in computation time, it can be concluded that PR is the better metamodel type for MTRO. The RBF metamodel may become more competitive if more processors are available. It is uncertain from these results whether the BFGS update is useful or not.

All five metamodels proved fairly robust in finding the desired minimum despite the challenges of the problems listed in Section 4.1. Each of the metamodels were tested 10 times with the same starting point on the same problem in order to get a larger sample of this stochastic process. The success rates are given in table 4.1.

<table>
<thead>
<tr>
<th>Metamodel</th>
<th>Crazy</th>
<th>Mullur</th>
<th>Ackley</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>100%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>PR with BFGS</td>
<td>100%</td>
<td>80%</td>
<td>100%</td>
</tr>
<tr>
<td>Kriging</td>
<td>100%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>Kriging with BFGS</td>
<td>100%</td>
<td>90%</td>
<td>40%</td>
</tr>
<tr>
<td>RBF</td>
<td>70%</td>
<td>80%</td>
<td>90%</td>
</tr>
</tbody>
</table>

Table 4.1: Success rate of the metamodels during comparative trials of MTRO

According to these trials, the most robust methods are the non-BFGS versions of PR
Chapter 4. Results

(a) Time of MTRO evaluations on various problems with varying metamodel types using IOLHS

(b) Number of trust regions of MTRO evaluations on various problems with varying metamodel types using IOLHS

(c) Number of original model computations of MTRO evaluations on various problems with varying metamodel types using IOLHS

Figure 4.7: Results of comparative metamodel type trials using MTRO
and kriging, and PR with BFGS not being far behind. The RBF metamodel proved to be less reliable for all three problems tested. While failed trials did not find the global minimum, they did find local minima. These results show the importance of choosing the correct starting point in a gradient based optimization and if the starting point were changed, these success rates could change dramatically.
4.2.3 Comparing Model Management

The traditional trust region method implemented here did not perform well, and failed to converge a majority of the time. While this renders accurate appraisals of the overall operation of traditional trust region methods impossible, there is still enough information to draw conclusions from the comparison of MTRO to this traditional trust region.

The information garnered directly from the plots gives similar results to that of MTRO. Kriging took significantly longer to complete the optimization process than PR, as seen in Figure 4.8(a). The number of trust regions (Figure 4.8(b)) and the number of original model evaluations (Figure 4.8(c)) are similar in each of the problems. These results also show that PR is better than kriging, while giving no indication of the usefulness of the BFGS approximation.

The comparison between MTRO and this traditional trust region method demonstrate a clearly better method, especially in consideration of parallel processing. The number of original model evaluations (Figure 4.8(c)) only shows the number needed to build the metamodels at each stage, and not the one original model needed for each trust region to determine the actual reduction. Figure 4.9 details the implications of using parallel computing with the ideal number of processors for each trust region iteration. All three subfigures show that parallel computing is extremely advantageous when used with metamodels. While no clear benefit is evident in using MTRO in serial processing, parallel computing shows a clear advantage.

The ideal parallel processing implementation of both MTRO and traditional trust regions would result in all the points needed for a metamodel being computed at the same time. Consequently the number of trust regions needed for an optimization is the minimum number of iterations to the trust region update. Assuming the time to construct the metamodel is insignificant compared to the time to compute the original model (as is the expected case with metamodels), the overall cost can be approximated by the number of sequential evaluations of the original model. In MTRO, the number of
Chapter 4. Results

(a) Time of traditional trust region evaluations on various problems with varying metamodel types using IOLHS

(b) Number of trust regions of traditional trust region evaluations on various problems with varying metamodel types using IOLHS

(c) Number of original model computations of traditional trust region evaluations on various problems with varying metamodel types using IOLHS

Figure 4.8: Results of comparative metamodel type trials with traditional trust region optimization
Figure 4.9: Comparison between MTRO and traditional trust region method considering parallel computing and the three major types of metamodel
sequential evaluations of the original model is the number of trust regions whereby all the points for each region are computed in parallel. For traditional trust regions, the parallel evaluations of the original model for the construction of the metamodel cannot occur at the same time as the original model evaluation for the actual reduction of that trust region update step. This means that the time for two original model evaluations have to occur for each traditional trust region iteration. Even if MTRO and this traditional region converged to the same solution in the same number of trust regions, MTRO would be twice as fast, under the assumption that the original model evaluation is much more expensive than the metamodeling costs.

The comparison between MTRO and traditional trust region methods results in a clearly superior method when parallel computing is used, according to these results. MTRO had a smaller number of trust regions for most of the optimizations than traditional trust regions (as seen in Figure 4.9), and coupled with parallel processing which makes each trust region for the traditional method twice as expensive as MTRO’s, MTRO is certainly more efficient than traditional trust regions. MTRO converges to the proper solution most of the time (as seen in Table 4.1), while this traditional trust region method typically failed to converge to the proper solution. In both accuracy and efficiency, MTRO is the better solution.

4.2.4 Effect of the Number of Dimensions on MTRO

The quartic function used in these trials is very simple, but it clearly shows the effect that dimensionality has on metamodeling methods. Figure 4.10, Figure 4.11, and Figure 4.12 all require a semilog plot in order to give a decent representation of the results. None of the metamodels were able to overcome dimensionality. Figure 4.10 does show the extent of the additional cost of kriging over the second order PR; kriging is approximately two orders of magnitude more costly than PR.

The most interesting feature of these results is the behaviour of the BFGS meth-
Figure 4.10: Time of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS

methods with increasing dimension. Both kriging with BFGS and PR with BFGS started needing significantly more original model evaluations (Figure 4.12) and trust regions (Figure 4.11) as the number of dimensions increased. These results seem to indicate that the metamodels generated by the BFGS methods seemed accurate to the cross-validation method, but in reality they were poor representations; resulting in more trust regions and consequently original model evaluations being needed to reach the optimum. This larger amount of trust regions would significantly degrade the operation of these BFGS metamodels when used in parallel computing. These results clearly eliminate the BFGS methods as a possible metamodel method with MTRO.

4.2.5 Effect of the Number of Processors on MTRO

Tests were run with an increasing number of processors for the three dimensional Ackley function in order to determine what the influence of parallel processing had on the overall time. Since the Ackley function is analytic and requires little computation time, the main
Figure 4.11: Number of trust regions of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS

Figure 4.12: Number of original model computations of MTRO evaluations on quartic problem with varying number of dimensions and metamodel types using IOLHS
contributions to the overall time come from the ED, metamodel construction, and trust region optimization. In increasing the number of processors, the overhead of the parallel processing was evaluated. From the tests done, there was no indication that the parallel processing added time to the overall operation of MTRO.

4.3 Comparing Multidisciplinary Optimization Problems

The behaviour of MTRO when combined into a multidisciplinary optimization (MDO) problem is evaluated using an analytic problem introduced by Sellar et al. [4] and has been used in πMDO before [16]. Despite the problems low dimensionality, the problem exhibits characteristics found in larger MDO problems [16]. The problem is as follows:

minimize: \( x_1^2 + x_2 + y_1 + e^{y_2} \)

w.r.t.: \( z_1, x_1, x_2 \)

s.t.: \( 1 - y_1/3.16 \leq 0 \)
\( y_2/24 - 1 \)
\( -10 \leq z_1 \leq 10 \)
\( 0 \leq x_1 \leq 10 \)
\( 0 \leq x_2 \leq 10 \)

The MDO architecture chosen to test MTRO is called concurrent subspace optimization (CSSO), which uses metamodels as an approximation to the coupling variables. By estimating the coupling variables, sub-optimizations can be carried out on individual disciplines in parallel and the main objective function uses the coupling variable metamodels to speed up the overall optimization. For further information on CSSO, refer to Martins and Marriage [16].
The CSSO architecture comprising MTRO is compared to three other MDO architectures found in πMDO: multidisciplinary feasible (MDF), individual discipline feasible (IDF), and simultaneous analysis and design (SAND). MDF takes the entire MDO problem as a single discipline optimization, where the single discipline is an iterative solver for the multidisciplinary analysis of all disciplines. IDF solves each discipline independently where the system level optimizer is responsible for ensuring convergence of the multidisciplinary state. SAND decomposes the MDO problem into a single discipline problem by setting the governing equations of the discipline analyses as equality constraints. Refer to Martins and Marriage [16] for further description of each of these architectures.

The comparison between the architectures is done by artificially elongating the computation for the discipline analyses through a time delay. Time is added incrementally to the discipline analyses and the optimization is carried out at each step. These trials simulate what would happen if a more expensive MDO problem which behaves as the analytic function were to be optimized with the four architectures described above. This time delay is used to test the use of CSSO in the capacity it was intended for; expensive discipline analyses. MDF, IDF, and SAND were computed in serial, which is the current limitation of πMDO. The CSSO results were found with a simulated parallel solution, where the time shown is equivalent to what a parallel solution would be. The results of these trials are shown in Figure 4.13.

These trials indicate that even with only two disciplines, the CSSO algorithm can be competitive with other architectures. While CSSO performs better than MDF as the time delay increases, it still performs worse than IDF and SAND. CSSO should perform better in problems where more disciplines are present since the sub-optimizations can be performed in parallel.
Figure 4.13: Time delay added to discipline analyses to simulate more costly computations with varying MDO optimizations of analytic function from Sellar et al. [4]
Chapter 5

Summary of Research

5.1 Conclusions

The two main goals of this work have been accomplished. Advanced metamodeling support was added to \(\pi\)MDO; consequently metamodel based MDO architectures can now be compared to more conventional MDO methods. The second goal, to establish a new trust region update strategy, has been accomplished through the introduction of the metamodel trust region optimization (MTRO) algorithm.

The intention of \(\pi\)MDO being a tool for development of MDO architectures across computing platforms has not been compromised. The parallel processing of the metamodel sample points is done with a master-slave algorithm with a user defined number of processors available, and uses dynamic process spawning and killing to allow other portions of \(\pi\)MDO to be parallelized and use the same processors that the metamodel code is using.

Five experimental designs have been compared at the optimization level in order to determine their effect on the overall performance. The five experimental designs include three previously developed algorithms, the original latin hypercube sampling [22] (LHS), Park’s [2] optimal latin hypercube sampling (OLHS), and Wang’s [3] inherited
latin hypercube sampling (ILHS), as well as two newly developed methods. One new experimental design, inherited optimal latin hypercube sampling (IOLHS), was developed that combines uses part of Park’s [2] OLHS to optimize the point distribution in the auxiliary design space of Wang’s [3] ILHS method. The second new algorithm, hybrid latin hypercube sampling (hybridLHS), also uses OLHS and ILHS but considers the locations of the inherited points during the optimization process. The new method IOLHS was determined to the best overall as it provides support for the reuse of previously evaluated sample points as well as optimizing the space filling set of trial points.

Five metamodel types have also been compared at the optimization level in order to determine their effect on the overall performance. The metamodel types include four previously developed methods, basic second order polynomial regression (PR), Lee et al.’s PR with BFGS quasi-Newton algorithm [12], Sacks et al.’s kriging method [29], and a typical radial basis function (RBF) implementation. The new metamodel type combines the kriging method with Lee et al.’s [12] BFGS approximation in an attempt to lower the cost of kriging and make it a feasible metamodel for trust region optimization. This new method did not perform well for increasing dimension. Polynomial regression (PR) was found to offer the best performance when fewer numbers of processors were available while radial basis functions (RBF) perform better if there are sufficient numbers of processors.

The “curse of dimensionality” is still evident in all metamodels and trials were limited to low numbers of dimensions. The attempt of the BFGS method to reduce the effect of dimensionality did not succeed and resulted in higher computation times for higher dimension problems. It is suspected that while a lower number of points are used for each trust region, the reduced accuracy caused by fewer points results in more trust regions being needed; more trust regions with fewer points still results in a greater overall number of points being needed.

A new trust region algorithm, metamodel trust region optimization (MTRO), uses progressive validity requirements to ensure a minimum level of validity for each meta-
model used in the trust region optimization. Lower validity metamodels are used for the larger trust regions at the beginning of the optimization, and higher validity metamodels are used for the smaller trust regions at the end of optimization. This progressive validity method minimizes the number of points in each stage of the metamodel. Calls to potentially expensive discipline analyses are avoided through the use of leave-$k$-out cross-validation. In addition, the computation of the actual reduction found in traditional trust region methods is eliminated to allow the removal of the original model analysis contained in the actual reduction which cannot be parallelized. It is thought that the guaranteed validity of the metamodel will allow a trust region update based solely on location within the region to be successful. In single discipline optimization trials, MTRO proved to be a more efficient solution than traditional trust region methods when parallel computing is considered. Neither method is clearly superior than the other in non-parallel cases.

Comparative trials were conducted between concurrent subspace optimization (CSSO) [14] and three other πMDO architectures: multidisciplinary feasible (MDF), individual discipline feasible (IDF), and simultaneous analysis and design (SAND). The CSSO method uses the newly developed MTRO algorithm as a model management method. These four architectures were tested with the analytic function found in Sellar et al. [4]. A time delay was incrementally added to the analytic function to simulate a progressively more expensive function. While MTRO-based CSSO was less efficient than IDF and SAND, it performed better than MDF and proved to be a competitive method. Further trials with an increasing number of disciplines would suit the CSSO algorithm better and may prove to be a superior method for these cases.

\section{5.2 Future Research}

While the current metamodel implementation is fully functional, there are many areas where research could be directed to advance bilevel MDO architectures within πMDO.
The following items are suggested for the development bilevel MDO architectures and \(\pi\)MDO in general:

- Perform trials with increasing numbers of disciplines and dimensions in order to determine which types of problems have expensive enough analyses while avoiding the increasing cost of dimensionality associated with metamodels.

- Use MTRO in other bilevel MDO architectures, such as bilevel integrated system synthesis (BLISS) [49] and collaborative optimization with response surfaces (CORS) [15].

- Parallelize CSSO and other non-metamodel based architectures, such as IDF.

The first item will allow a clear indication of the practical application of bilevel methods and either direct the required research or provide enough evidence to preclude any further research in this area. If there are no problems which would result in a bilevel method being the most efficient, then no further work should be directed towards these methods. The second and third items on this would allow more development into parallel computing methods, which seems to be the direction the scientific community is heading.

In addition to improvements to \(\pi\)MDO, there are areas where the metamodeling can be explored and potentially improved. The following work could be beneficial:

- Determine how much the experimental design contributes to dimensionality problems. Would a less robust experimental design such as LHS provide enough performance gains while still providing sufficient success rates?

- Test non latin hypercube based experimental designs, such as the Sobol sequence [21].

- Explore the use of variable and multi-fidelity metamodeling strategies especially in the area of aerostructural optimization, considering the availability of multiple levels of fidelity in pyACDT.
• Explore the use of patching multiple metamodels together to lower overall cost of the metamodels.

• Program the support vector regression metamodel and perform comparative analysis.

The first two items respond to potential inefficiencies within the current work. The last three items are areas of potential research, all of which were presented numerous times at the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, held 10-12 September 2008, at Victoria, British Columbia, Canada.
Bibliography


