Reduced order techniques for sensitivity analysis and design optimization of aerospace systems

By

Jefferson Carter Parrish

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computational Engineering
in the Bagley College of Engineering

Mississippi State, Mississippi

May 2014

Reduced order techniques for sensitivity analysis and design optimization of

aerospace systems

By

Jefferson Carter Parrish

Approved:

Masoud Rais-Rohani
(Major Professor)

J. Mark Janus
(Co-Major Professor)

James C. Newman, III
(Committee Member)

Ioana Banicescu
(Committee Member)

Roger L. King
(Graduate Coordinator)

Jason M. Keith
Interim Dean
Bagley College of Engineering

Name: Jefferson Carter Parrish

Date of Degree: May 16, 2014

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Masoud Rais-Rohani and Mark Janus

Title of Study: Reduced order techniques for sensitivity analysis and design optimization of aerospace systems

Pages in Study: 183

Candidate for Degree of Doctor of Philosophy

This work proposes a new method for using reduced order models in lieu of high fidelity analysis during the sensitivity analysis step of gradient based design optimization. The method offers a reduction in the computational cost of finite difference based sensitivity analysis in that context.

The method relies on interpolating reduced order models which are based on proper orthogonal decomposition. The interpolation process is performed using radial basis functions and Grassmann manifold projection. It does not require additional high fidelity analyses to interpolate a reduced order model for new points in the design space. The interpolated models are used specifically for points in the finite difference stencil during sensitivity analysis.

The proposed method is applied to an airfoil shape optimization (ASO) problem and a transport wing optimization (TWO) problem. The errors associated with the reduced order models themselves as well as the gradients calculated from them are evaluated. The effects of the method on the overall optimization path, computation times, and function counts are also examined.

The ASO results indicate that the proposed scheme is a viable method for reducing the computational cost of these optimizations. They also indicate that the adaptive step is an effective method of improving interpolated gradient accuracy. The TWO results indicate that the interpolation accuracy can have a strong impact on optimization search direction.

ACKNOWLEDGEMENTS

I am deeply grateful to my committee and advisors for their continued support and guidance. I would also like to thank my family and friends for their patience and understanding during this endeavor. This has been a long path filled with unexpected turns and obstacles, and without these people, traveling it would not have been possible.

The Bagley College of Engineering and the Center for Advanced Vehicular Systems have also made this work possible through their extensive support and resources. I would also like to acknowledge the NASA Mississippi Space Grant Consortium for their support. Vanderplaats Research and Development was kind enough to provide guidance and reference for the second test problem in this work, which was especially appreciated.

Finally, no engineering thesis would be complete without acknowledging the contributions of the great M. Scott, who has inspired more than one engineer to pursue their interests in space.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AoA	Angle of Attack
ASO	Airfoil Shape Optimization
BC	Boundary Condition
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
CSM	Computational Structural Mechanics
DoE	Design of Experiments
ERBF	Extended Radial Basis Functions
FEA	Finite Element Analysis
FEM	Finite Element Model
FSI	Fluid Structure Interaction
FVM	Finite Volume Method
GMP	Grassmann Manifold Projection
GSE	Global Sensitivity Equations
IROM	Interpolated Reduced Order Model
KER	k-ε Realizable (Turbulence Model)
MDA/MDO	Multidisciplinary Design Analysis / Optimization
MDF	Multiple Discipline Feasible
PCA	Principle Components Analysis

POD	Proper Orthogonal Decomposition
RBF	Radial Basis Functions
RNG	k-ε Renormalization Group (Turbulence Model)
ROM	Reduced Order Model
SA	Sensitivity Analysis
SQP	Sequential Quadratic Programming
SST	k-ω Shear Stress Transport (Turbulence Model)
SVD	Singular Value Decomposition
TWO	Transport Wing Optimization

CHAPTER I

INTRODUCTION

In a typical gradient-based optimization, one of the costly steps involves calculating the gradients of the objective and constraint functions with respect to the design variables at various points in the design space in search of the optimum design point. This is typically accomplished by using a finite difference scheme, requiring a minimum of two function evaluations to calculate an approximate gradient of the function at each design point. When the objective and constraint functions are based on the results of expensive high fidelity analyses, this method of gradient calculation for sensitivity analysis becomes very costly in terms of both time and computer resources.

In this dissertation research, a new methodology is developed to reduce the computational cost of finite difference-based sensitivity analysis by relying on interpolation of disciplinary reduced-order models (ROMs) based on proper orthogonal decomposition (POD). Since the interpolation scheme does not require additional high fidelity simulations to construct new reduced order models, the cost of evaluating design sensitivities is significantly reduced. Rather than using the reduced-order models for general design point evaluation as is typically done in the literature, they are interpolated only for the sensitivity analysis stencil (i.e., function evaluation at the perturbation point) while the high fidelity analysis for the design point of interest is retained. This allows the optimization to utilize more accurate (high fidelity) analyses for each design point along the optimization path while greatly reducing the cost of sensitivity analysis evaluations by use of interpolated ROMs (IROMs). It also allows the interpolation of ROMs for use in larger multidisciplinary systems. Further, the particular method described here does not require additional high fidelity evaluations to construct the interpolated ROMs, thus greatly reducing the costs associated with their construction and evaluation.

1.1 Background and Related Work

This work draws on several different areas of work in surrogate modeling, reduced-order modeling, high fidelity engineering analysis, and optimization of complex systems. Surrogate modeling in particular serves as a useful context for reduced order modeling.[1,2] Methods such as polynomial regression, Kriging, multivariate adaptive regression splines, radial basis functions, artificial neural networks, and support vector machines are all common approaches to providing more economic evaluations of a function.[3–16] The effectiveness of these techniques is closely tied to topics such as design of experiments (DOE), both from general statistics as well as the specific circumstances of computational experiments, as can be seen in various example applications. [15,17–33]

Multidisciplinary analysis and optimization (MDA/MDO), including formal optimization, coupled system theory, and sensitivity analysis, is also relevant to this research work. Traditional optimization has a long history, rooted in mathematical programming, and has well developed tools.[34–36] A good overview of MDA/O technique and problem formulation can be found in Cramer et al.[37] and especially Sobieszczanski-Sobieski and Haft[38], among other overviews.[39–41] The theory and analysis of coupled systems forms much of the mathematical underpinning of MDO techniques.[42] Problem partitioning is also a key concept, especially for hierarchical frameworks.[43] Some examples of MDO applications include aeroelastic optimization,[44–46] multilevel wing box optimization,[47] aerostructural optimization,[25,48] space structures,[49] and others.[50–54] There have been several attempts at providing multidisciplinary problem test suites, which are also useful as examples; these include efforts by the Hulme and Bloebaum[55–58] and Padula.[59] Bandwidth reduction is a common issue with MDO applications, and several techniques have been suggested for addressing it; this includes intermediate models,[1,60] reduced order data sets,[61,62] and other approaches.[63,64] Another issue is dealing with blackbox analysis functions.[65] Sensitivity analysis for these problems is a well-studied topic.[66–69] MDA and MDO have also benefited from the development of the global sensitivity equations (GSEs) , which allow determination of global sensitivities using local derivative information.[70–73]

MDO strategies such as multiple discipline feasible,[52] all-at-once, individual discipline feasible, collaborative optimization,[74] collaborative subspace optimization,[75] bi-level integrated system synthesis,[76] the collaboration pursuing method,[77–80] analytical target cascading, and others[50,81,82] are examples of common MDO schemes into which this work may be integrated in the future.[61] There have been several studies comparing various frameworks[40,53,83–87] and presenting general descriptive schemes for the various techniques.[40,88,89]

Process modeling frameworks are closely related to these schemes, being the practical implementations that ultimately determine how useful the techniques are to end users.[90] Often these frameworks provide the capability to rapidly address multiple design concepts, which is an important part of real-world design work.[91–93] Due to their nature, they are often designed with extensibility and integration with other tools in mind,[94] as the frameworks themselves often become major projects.[95] These include frameworks to address concurrency,[96] strongly coupled analysis,[45] the development environment,[97–101] and other practical coordination aspects that are often overlooked by more theoretical explorations of MDO.[54,102,103] Many such process frameworks have already been extensively developed and commercialized.[104–108]

There has been a great deal of application of reduced order techniques as surrogate models for high-fidelity, computationally expensive analyses.[27,76,109–114] Jones et al in particular present an interesting approach of correlating errors instead of values, to capture function behavior over function magnitude.[65] LeGresley uses POD-based surrogate models for optimization, while switching back to the full-order models for sensitive areas of the design space.[115,116] Some work involving variants of particular methods, such as constrained POD, can delve quite deeply into the underlying mathematics.[117–120] Other work focuses on incorporating sensitivity information into the POD models,[121] or expanding the valid parameter space by analyzing the model's sensitivity to various inputs.[122] A more general overview of POD for model reduction can be found from Volkwein.[123] An excellent report on various model reduction techniques, as well as examples of their applications, can be found in reports by Newman.[124,125]

This research work is also concerned with reduced order model interpolation, especially the elements of radial basis functions, POD-based ROMs, and their potential role with optimization and sensitivity analysis. Volterra-based ROMs were also considered, but discarded by comparison with POD-based ROMs during the preliminary problem work. Most of this investigation was based on Silva's work,[126–133] with some additional reference regarding application of Volterra kernels and their requisite components.[134–136] Although effective when applied appropriately, it was determined that they were not well suited for the goals of this research.

There has been a moderate amount of prior work specifically on interpolating ROMs for application to analysis or optimization. Those most relevant to the current work are highlighted here. Investigation into various POD basis vector interpolation schemes ultimately resulted in the selection of Grassmann Manifold Projection (GMP) as the ideal basis interpolation method. Most interpolation approaches for POD tend to be analytical. Statespace based representations are common for these approaches.[137] Lieu et al proposed a method for interpolating POD subspaces instead of the basis vectors themselves.[138] Naets et al compared several interpolation strategies for structural optimization, including one operating in the eigenspace of the relevant matrices.[139]

GMP is a projection applied to POD bases which is then used in concert with a direct interpolation method. It is explained in excellent detail in the work by Vetrano et al.[140] Amsallem et al apply GMP and splines to POD before projecting the known system equations onto the interpolated basis.[141–144] Degroote et al demonstrate POD interpolation with GMP and splines, and like this work do not require full order data to construct the new ROMs; however, it does assume that a statespace representation of the system of interest is available.[145] Using a global POD basis with interpolated coordinates is also a somewhat common approach,[62,139,146,147] and has been applied to determination of Pareto frontiers for multiobjective problems.[119,120] Very closely related is using POD to determine system modes before applying a coordinate interpolation scheme,[147] as well as projecting full order data.[148]

Each proposed ROM interpolation scheme was examined closely for similarities to the proposed IROM method, in order to establish the novelty of this work. While each of the relevant components of this work have been used elsewhere, and similar interpolation methods have been developed, none found have integrated these methods in precisely this manner or applied the method strictly to the sensitivity analysis in the manner proposed in this work.

In their work, Degroote et al.[145] utilized POD to produce ROMs that are interpolated using Grassmann Manifold Projection (GMP) and spline interpolation to determine bases for new design points, either for analysis or optimization. The ROMs were constructed by assuming that a state-space representation of the system dynamics is readily available, and thus the known system matrices are projected onto the interpolated basis to complete the ROM. In contrast, this work assumes nothing about the system is known except for a small set of inputs and outputs. ROMs are comprised of a set of basis vectors and a set of coordinates corresponding to snapshots of the system output. This work also uses Radial Basis Functions (RBF) as described by Mullur and Messac[10] rather than spline fits. GMP was proposed as a method for POD basis interpolation by Amsallem et al.[141–144] who reported several applications which utilized it for ROM interpolation. Initially focused on extending the valid analysis space, later work developed ROMs by projecting the system equations analytically onto the interpolated POD bases, and more recently by rerunning the fluid analysis code to project the results onto the new basis. Other works such as that by Naets[139] focused on analytical reduction of the governing systems, or more commonly by using a global POD basis and then interpolating the coefficients. Coelho et al.[62,147] applied this approach to examine the impact of ROMs in Multidisciplinary Design Optimization (MDO), although they did not iterate the fluid-structural system and commented on the challenges this creates. Xiao et al.[119,120] applied a similar methodology to automotive applications, attempting to determine the Pareto frontier for that class of problems. None of the cited works encompass ROM interpolation for black-box systems.

Prior work performed by LeGresley[115] is also worth mentioning, as it integrates POD into the BLISS MDO architecture, primarily for reducing coupling bandwidth and dealing with the interaction variables. This may prove a useful starting point to investigate integrating the proposed ROM interpolation scheme with existing MDO methods in future work. Finally, Vetrano et al.[140] compared several POD interpolation schemes, which contains an excellent overview of the dominant methodology and concludes that GMP is a powerful approach to that problem.

1.2 Dissertation Organization

This dissertation is structured as follows. Following the introduction in this chapter, the component methodologies that this research draws on are presented in CHAPTER II. After presenting the relevant background, the method is developed along with its component methods. The proposed interpolation method is discussed in CHAPTER III. It is then applied to an Airfoil Shape Optimization (ASO) and a Transport Wing Optimization (TWO) test problem to evaluate the method's effectiveness for more complex optimization problems. The descriptions of the test problems are given in CHAPTER IV. Although the ASO problem is a single discipline, multipoint problem, it is easy to see that the proposed IROM scheme has natural benefits for a multidisciplinary analysis. This is demonstrated by the structure of the TWO problem, which was selected with MDA problems in mind. The computational framework used to apply the interpolation method to each test problem is described in CHAPTER V. The results of each problem are presented in CHAPTER VI. There errors associated with the interpolated ROMs as well as a discussion of the impact on the overall interpolation procedure are presented. The emphasis for the ASO problem is on demonstrating the feasibility of the method, while the TWO problem is geared to illustrating its applications in a multidisciplinary structure. Finally, conclusions are discussed in CHAPTER VII.

CHAPTER II

METHODOLOGY

This chapter will detail the component methodologies used for the IROM scheme, before detailing the scheme itself. A discussion of Sequential Quadratic Programming (SQP) will first provide some context for this work, by illustrating a common gradient-based optimization process. Following this, a summary of Computational Fluid Dynamics (CFD) and Computational Structural Mechanics (CSM) is presented as the example of high-fidelity analysis used here. An overview of Design of Experiments (DOE) methodology used for design space sampling follows before a more detailed discussion of the Reduced Order Models (ROMs) used in this work.

2.1 Engineering Design Optimization

Optimization is a critical part of the design of engineering systems, particularly the complex systems found in aerospace. While, to some extent, the design of any aerospace system involves multidisciplinary trades and optimizations, the process has only been formalized in the last few decades. Particularly, optimization of vehicles during the preliminary design stage involves multiple high fidelity disciplinary analyses in a strongly coupled system. Traditional optimization schemes which are derived from the field of mathematical programming are applicable to these multidisciplinary systems by operating on the results of multidisciplinary analyses, which account for interdisciplinary interactions during the coupled analysis stage. Multidisciplinary optimization schemes, by contrast, incorporate these interactions into the optimization scheme itself.

Optimization of complex aerospace systems tends to be very costly in terms of time and compute resources, and often in human effort as well. The high fidelity analyses utilized during the detailed design phase are often resource-limited on their own, a cost which is compounded both by the numerous analyses required for an optimization as well as the costs associated with coupling multiple high fidelity analyses together. A classic example of this is found in high-fidelity aeroelastic optimization, where detailed fluid and structural simulations must interact to perform a single analysis, and many analyses are required to perform the optimization. Not only are the individual simulations resource-expensive, the coupled analysis incurs additional costs due to data coupling between the simulations, grid deformation, and additional convergence cost of the coupled system, among other factors. The optimization spaces are also complicated due to multidisciplinary interaction effects, which are not observed in the systems individually, a concept familiar to those working with design of experiments (DOE).

A note on terminology is appropriate here: when we are referring to the system as being “weakly” or “strongly” coupled, we are referring to the strength of the interaction effects of the multidisciplinary system itself. By way of example, a large heat sink with cooling fins at low heat flux may demonstrate weak coupling between the thermal analysis and the structural deformation. A wing near flutter conditions, on the other hand, would demonstrate strong coupling between aerodynamic and structural analyses. By contrast, when we refer to the problem as being “loosely” or “tightly” coupled, we are referring to the structure of the integration of the analysis codes. Tightly coupled generally refers to analyses which are integrated on an analytical or code level, such as utilizing equations from the field of aeroelasticity. Loosely coupled indicates disparate analysis modules, such as three separate fluid, structural, and thermal analysis codes which have their inputs and outputs coordinated for a larger multidisciplinary analysis.

It is worth noting that here we are concerned with high-fidelity, loosely-coupled multidisciplinary optimizations, where the individual analysis codes are considered black-box and possibly proprietary, and their direct coupling is not possible. It is often the case that either the fluid or structural models will be severely simplified to allow for low-order interactions to be accounted for while incurring minimal additional costs. It is also common to avoid the additional convergence cost by assuming small deformations, allowing a fluid solution to be transferred directly to the structural model without requiring iteration between the two. Both of these practices are effective under appropriate circumstances for providing some interaction effects at a minimum additional cost; however we are interested in higher fidelity cases where these simplifications are unacceptable. It is also common for aerostructural analyses to directly couple the analytical equations of the individual disciplinary codes or to utilize analytically derived equations from the field of aeroelasticity, which has been extensively developed on its own.[149] These practices are again appropriate under certain circumstances, but they do not address the issue of black-box, proprietary codes. More importantly, with the addition of more disciplinary analyses, the analytical task of coupling the individual equations quickly becomes impractical. For example, a hypersonic magnetohydrodynamic aerothermoelastic problem will involve coupling between aerodynamic, structural, thermal, chemical, electromagnetic, and radiation models. The ability to offload the coupling workload to a computational system, as opposed to human or human-guided effort, presents obvious benefits.

Our optimization scheme will treat this coupled analysis as a single system, as in Figure 2.1. Much research has been expended on integrating the interdisciplinary analysis and coupling into the optimization framework itself, i.e., the field of MDO. This is a very promising research area and provides an efficient theoretical framework for exploring more efficient optimization methods of complex coupled systems. However, for reasons of simplicity, this work will utilize the scheme known as multiple discipline feasible (MDF) optimization. Essentially, the optimization scheme is identical to the well-studied single-discipline optimization schemes, where the objective and constraint functions are concerned only with the result of the multidisciplinary analysis (MDA), and not with the mechanisms of obtaining it. To the MDF scheme, there is no difference between single- and multiple-discipline analysis. This allows us to utilize classical optimization schemes such as the modified methods of feasible directions, SQP, etc.

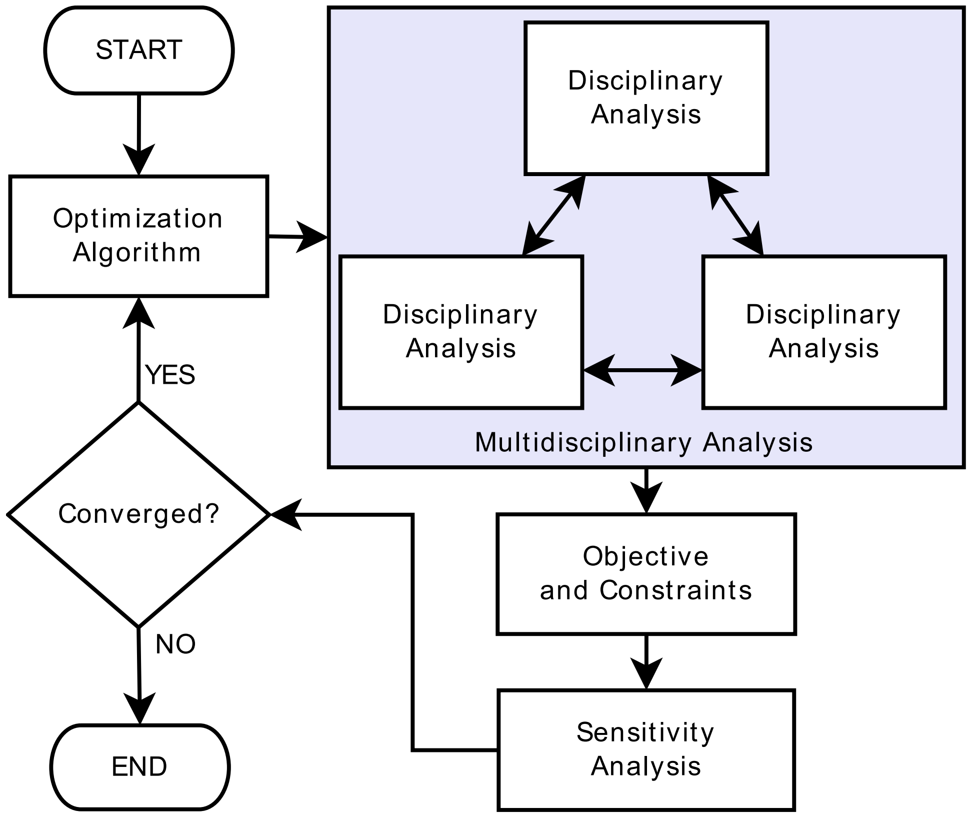

Figure 2.1 Optimization / Analysis System Structure

A full multidisciplinary analysis is performed for the design point X during each iteration of the optimization. A set of perturbation points $\{X + \Delta X\}$ based on the finite difference stencil is also evaluated for gradient estimation and sensitivity analysis. The results of the analyses, Y, are utilized as input to the objective and constraint functions, F and G, which are returned to the optimizer in order to evaluate the next design point. Note that the sensitivity analysis routine computes the sensitivities of the objective and constraint functions to changes in the design point, dF/dX and dG/dX.

In this case, the optimization method selected is SQP, a nonlinear optimization method. Details of SQP are available from a number of resources, and in this work the built-in SQP functionality from MATLAB is utilized.[150,151] To summarize, starting from an arbitrary initial design point, a quadratic subproblem is formulated and solved for finding a search direction. While the objective function is approximated with a quadratic function, all the design constraints are linearized using gradient information of the objective and constraints. Only the first-order derivatives are used, even in the case of approximating the Hessian of the Lagrange function. A simple line search is performed, as a second step, using a one-dimensional minimization technique to find the optimum step size along the search direction to determine the next point along the optimization path. The two-step procedure is repeated until a local optimum design point is found.

2.1.1 Finite Difference Based Sensitivity Analysis

Sensitivity analysis (SA) is a general term describing the process of determining the partial derivative of the function with respect to each input at a given point in the parameter space. In the context of this research, and in optimization in general, it refers to determining the gradients of the objective and constraint functions with respect to the design variables. In a full multidisciplinary context, the global sensitivity equations (GSE) are often applied to determine the full system sensitivity based on the individual disciplinary derivatives.[152] For a set of disciplinary analyses, f_i, with associated outputs, $\boldsymbol{Y}_i$, which may serve as inputs to other analyses, the derivative of all system outputs with respect to system inputs, $\boldsymbol{X}$, can be determined using the derivative information of each analysis to its inputs. As an example, for a three discipline system:

$$\begin{bmatrix} I & -\partial f_1/\partial \boldsymbol{Y}_2 & -\partial f_1/\partial \boldsymbol{Y}_3 \\ -\partial f_2/\partial \boldsymbol{Y}_1 & \boldsymbol{I} & -\partial f_2/\partial \boldsymbol{Y}_3 \\ -\partial f_3/\partial \boldsymbol{Y}_1 & -\partial f_3/\partial \boldsymbol{Y}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \partial \boldsymbol{Y}_1/\partial X_i \\ \partial \boldsymbol{Y}_2/\partial X_i \\ \partial \boldsymbol{Y}_3/\partial X_i \end{bmatrix} = \begin{bmatrix} \partial f_1/\partial X_i \\ \partial f_2/\partial X_i \\ \partial f_3/\partial X_i \end{bmatrix} \tag{2.1}$$

While information from these disciplinary modules is sometimes available analytically or as part of the module output, this research assumes that such modules are black-box functions which do not supply this information. Furthermore, by adopting an MDF approach, the values and thus the sensitivities of the objective and constraint functions are dependent on the full MDA analysis; thus, the GSEs are not applied here, merely the single-function finite difference formula for a given degree of accuracy. In this research, finite differencing is achieved directly and numerically by varying the design variables and operating on the objective and constraint outputs. In the cases described here, the standard forward difference for the first derivative is used. Finite difference approximations for gradient calculations are well established procedures, and the reader is referred to the references for further information.[67,153]

2.2 Computational Analyses

For the optimization structure this work utilizes, there is a coupled multidisciplinary analysis involving individual high-fidelity codes. In particular, for most of the intended problem systems, there is a coupling between separate aerodynamic and structural codes for aeroelastic or aerothermal solutions. In the simplified test case proposed here, the individual solver is ANSYS Fluent; a combination of Mathworks MATLAB and various journal/shell scripts will be applied as well. This section gives a brief overview of CFD and CSM for this work; a more comprehensive description is available in the literature.[154]

CFD is primarily concerned with the solution of the Navier-Stokes equations of fluid dynamics and their related counterparts (such as the Euler equations). These equations model the dynamics of a fluid's mass, energy, and momentum across a defined volume of space. The numerical solution to these equations in various formulations and algorithms has been the subject of decades of work. Computational limitations and increased problem complexity have served as impetus for the integration of many other models into CFD solvers, ranging from turbulence models of varying fidelity, to chemistry and species transport, to accounting for electromagnetic forces in magnetohydrodynamic flows, to mixed phase flows involving gases, liquids, and solids.

Of prime importance when developing a CFD simulation case is the numerical grid describing the volume of interest. The geometric properties of this grid have a strong impact on both the solution's resolution and its numerical stability and convergence. There are essentially two approaches to defining that volume of interest: the finite volume method (FVM), which defines regions which fluid passes through, and the finite element method (FEM), which utilizes deforming geometric elements. Mathematically, the FVM can be shown to be a special case of FEM. Most CFD solvers utilize the finite volume formulation, although many exist which rely on the FEM approach. This is especially true for solvers that are intended to interface with structural solvers, which are almost exclusively developed using a finite element formulation. ANSYS Fluent primarily utilizes finite volume grids. In general, grids and operations performed on them (such as grid repair, automatic meshing, grid adaptation and deformation) are a major research area in their own right.

Typically, a case setup for a CFD simulation first involves defining the geometry and domain of interest and meshing it into an appropriate grid. The boundaries of the grid are assigned various boundary conditions as the problem and solution models require

– an example would be a wall condition, which allows no fluid flow across the boundary, or a pressure inlet condition, which defines specific conditions that must be maintained on the boundary. These boundary conditions (BCs) are used to close the set of equations describing the flow solution. The particular models and solver settings can then be selected, and various parameters related to the numerics of the solution set appropriately. It is typically important to verify that the chosen models, settings, grid, and BCs are appropriate for the problem of interest and are numerically compatible with one another. For example, it would be inappropriate to utilize the incompressible ideal gas equation for a supersonic flow. Understanding these various settings, the applicability to particular problem cases, the idiosyncrasies of arriving at a converged and realistic solution, and maintaining a balance between fidelity and computational cost comprise a large part of the "art" of performing CFD.

Computational structural mechanics is, naturally, concerned with the behavior of structures under loads. These loads may be concentrated or distributed forces, thermal loads, static or time-varying, and so on. Generally, CSM simulations are mostly concerned with the deformations and stresses within a structure under a certain set of loading conditions. They are also often concerned with issues such as fatigue, cracks, and failure modes, amongst many others. As noted above, CSM simulations typically rely on a finite element grid to describe the problem of interest. Generally a more complex structure is modeled using many smaller, simpler elements.

Case setup for CSM generally begins by defining the geometry of the structure of interest, usually via Computer Aided Design (CAD). Various parts or sections of parts may then be assigned material types, which is a shorthand way of specifying properties as required for the particular solution. Boundary conditions, usually loads of varying sorts or restrained degrees of freedom at the support points or friction coefficients (as in sliding interfaces) may also be specified. Meshing a CSM geometry typically refers to decomposing the geometry into numerous smaller finite elements. CSD grids tend to be far more regular than their CFD counterparts, although unstructured meshing elements are also used. Developing a CSM case typically requires basic CAD skills and an understanding of the structural analysis of interest (e.g., static, dynamic, buckling), as well as a grasp of FEM and other assorted terminology and concepts.

Coupling CFD and CSM solvers refers to the passing of relevant information between the two solvers while maintaining some sense of synchronicity in the solutions (which may or may not be temporal or algorithmic in nature). The traditional example case is an aeroelastically deforming wing, a case of fluid-structure interaction (FSI). The CFD code generates pressures over the surface of the wing as part of the fluid solution. These pressures are passed to the structural code, which must translate them across the fluid-structure interface and calculate the deformation of the structure in response to those loads. These deformations are passed to the fluid code, which updates the geometry of the wing accordingly. The change in geometry changes the flow solution, necessitating a recomputation. The basic system can be seen in Figure 2.2.

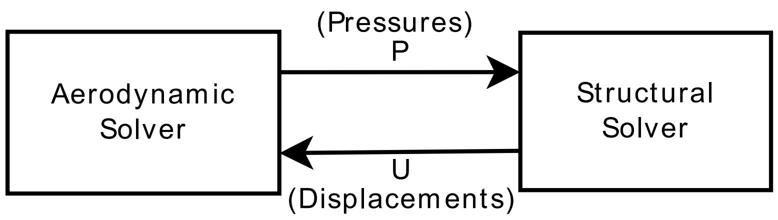

Figure 2.2 Simple Aeroelastic System Coupling

It is clear that when one or both of these solvers is expensive to run, this iterative coupling (referred to as loose coupling, as described in the optimization section above) can become very computationally intensive. It is also complicated by whether the solution is time-independent or transient. There are a variety of other methods, such as tight coupling (utilizing analytical or code-level coupling of the fluid and structural equations), various schemes to reduce the required algorithmic synchronicity, or approximating one of the solvers with a lower fidelity model (such as a reduced fluid equation set or utilizing structural modes). Some cases may make the assumption that deformation is very small and does not have a large impact on the fluid solution, in which case they may solve for a flow solution and then apply those loads to the structure without iterating. What is clear, however, is that the fluid and structural analyses form a coupled system that involve coordinated data passing. Given the large number of problems in aerospace and other fields that must take these interactions into consideration, it is clear why FSI is another intensive research area.

It is also clear that in an aeroelastic problem, a fluid-structure interface must be maintained such that the spatial relation of data in the two solvers can be determined. There is also a need for deformation or remeshing of the fluid geometry. Both of these are very technically intensive undertakings and represent a nontrivial investment of effort.

2.2.1 CFD Validation

Validation of the aerodynamics code was performed using ANSYS Fluent to select the appropriate grid parameters and solver settings. This took the form of a standard airfoil coefficients study, performed for a NACA 2412 airfoil. Validation was performed on a hybrid structured/unstructured grid with the pressure-based Navier-Stokes solver and the SST k-ω turbulence model. Results indicated good overall coefficient agreement, with some discrepancy in predicting the separation point of the airfoil. This was considered acceptable for the relatively small angles of attack considered in the ASO problem and the testing purpose of the problem. More specifically, the agreement between the simulation and experimental reference is good for the linear portion of the lift coefficient curve, and the max $c_L$also agrees well with experiment, although separation is predicted somewhat late. The drag coefficient, usually somewhat more difficult to predict, also indicates good agreement for the lower angles of attack. The results from this validation can be seen in Figure 2.3 and Figure 2.4.

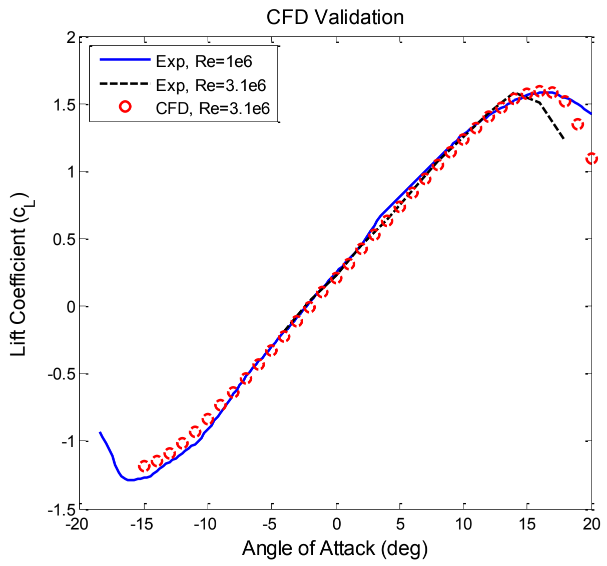

Figure 2.3 CFD Validation Results (NACA 2412 Lift Coefficient)

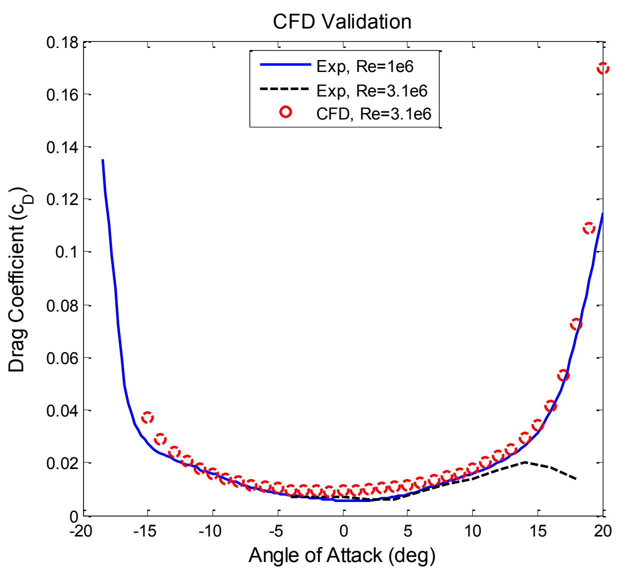

Figure 2.4 CFD Validation Results (NACA 2412 Drag Coefficient)

2.3 Design of Experiments

From its origins in statistics, experimental design has been applied in one form or another to a wide range of fields, from medical studies to manufacturing process control. In the present context, design of experiments (DoE) refers to the selection of inputs to a system of interest, chosen such that the system outputs will yield information on dynamics and behavior without detailed knowledge of the system itself. These outputs, or responses, or generally applied to create simplified models of the output space with regard to the inputs, thus leading to response surface methodologies and surrogate modeling. Here, we use DoE to refer specifically to the input selection scheme.

The most straightforward example of a DoE scheme would be random sampling. Given N inputs to the system, and a specified domain for those inputs, a set of random inputs is selected and the system response measured. Alternatively, the minimum and maximum of each parameter could be sampled, or all combinations of such. The latter is an example of a full factorial design: each input parameter p_i is allowed to assume m_i values, resulting in a total of $\prod_i m_i$ possible combinations. This highlights one of the primary properties of an experimental design – the number of sample points required. Obviously in a high-dimensional context, even restricting each of N parameters to two values will still result in 2^N samples. When the evaluation of a sample is expensive, or in the common engineering case where hundreds or thousands of dimensions may be considered, full factorial designs quickly accrue unacceptable costs.

These two schemes – one dimension at a time, and full factorial – represent two extremes of sampling designs. While measuring the response for each dimension independently provides efficiency in the total number of sample points, it ignores the possibility that parameters may not be fully independent – in a coupled system, especially the strongly coupled systems found in engineering, there are effects on the system response that cannot be accounted for by varying a single dimension at a time. These effects are only fully accounted for by a full factorial design, which is inherently expensive. This contrast, between minimizing the number of sample points, while maximizing the ability of the scheme to capture coupled system effects, is at the heart of most DoE schemes.

While a full explanation of DoE methodology is beyond the scope of this document, it is worth mentioning a few common schemes. Normal design of experiments includes the assumption of independent, randomly distributed errors. This assumption does not typically apply to computer experiments, which are generally deterministic and in which errors are usually nonrandom and correlated.[18,29] When dealing with very expensive functions such as those found in high-fidelity engineering analysis, full factorial or even partial factorial designs may be prohibitively time-consuming. Most scholars agree that the use of space-filling designs are most appropriate for computer experiments.[15] These designs attempt to gain as large a spread in the design space as possible, subject to other competing goals. Most engineering literature uses orthogonal arrays, Latin hypercubes, Hamersley sequences or uniform designs.[15] Latin hypercubes have several varieties, including normal, orthogonal, symmetric, and various combinations. Further, each sampling strategy may be developed to be optimal for the purpose of providing the most information about the design space.[23,32,33] Additionally, Taguchi methods are popular due to their origins in robust design. The emphasis on low objective sensitivity to design space perturbations also developed a signal-to-noise analysis which can be useful for certain applications.[31]

In this work, generally Latin Hypercubes are used for their good space-filling and efficiency properties. Designs are selected according to maximin criteria, i.e. maximizing the minimum distance between sample points.

2.4 Reduced Order Models

A common approach to reducing the computational expense associated with the complex spaces found in high-fidelity analysis and optimization is utilization of surrogate models. These are simplified representations of the space of interest, often fit to a sampling of the data in that space, which are much cheaper to evaluate and operate on than the original complex space. Like using simplified disciplinary equations, they trade off fidelity for speed, but since they are to some extent fitted to the actual space of interest, they are often able to represent the specific space of interest to a higher degree of accuracy than applying more general simplified equations. Thus, they are more efficient in their tradeoff of fidelity for reduced cost.

At its simplest, a surrogate model may be nothing more than a simple curve or polynomial surface fit. More complex models tend to provide better fits to the space of interest, but at the expense of construction or evaluation costs. These models are typically applied to replace a space of interest, such as replacing a complex objective space with a surrogate model that provide smoother gradients and much faster evaluation. Examples include multiquadratic surfaces, radial basis functions (RBF), kriging, neural networks, induction based learning, genetic algorithms, splines, and regression fits. The term surrogate model is often used as a blanket category for these models which replace a space of interest. Other terms include response surfaces and metamodels, with the precise definition of each term and its constituent methodologies varying somewhere from author to author.

In general, surrogate models tend to be very sensitive to the input data they are provided which describes the space of interest they are intended to approximate. While details vary from method to method, proper use of design of experiments (DoE) sampling methodologies is generally a key theme. The robustness of a model's approximation surface with respect to its input sample is a common point of comparison between methods. Additionally, the associated cost of generating the input sample datasets is usually a driving factor of the selection of a particular method, as these inputs require evaluation of the expensive high fidelity analysis functions. While some applications of surrogate models such as control systems are able to perform this initial sample calculation “off-line,” or in a non-time-critical environment, normal analyses and optimizations are generally desired with minimum turnaround time.

Some surrogate models are applied at the optimization level, approximating objective or constraint spaces and thus allowing the optimization to skip the full expensive analyses for some parts of the optimization. This application is typically associated with the use of trust-region and error-estimation metrics, which are a field unto themselves.

The other application we are interested in is the utilization of surrogate models to replace individual analyses during the multidisciplinary analysis stage. For example, replacing a computational fluid dynamics (CFD) code with a surrogate model representing the change in surface pressures and temperatures with respect to angle of attack might be used to speed up the overall coupled multidisciplinary analysis in an aerothermoelastic simulation. While input samples (in this case, solutions for various angles of attack) are still required to compute the surrogate model, the number of samples required may be far less than the number of evaluations otherwise required to converge the full multidisciplinary system. On the other hand, if the geometry of the case is changing as during a shape optimization, the surrogate model built for a single design point is not valid for application to other design points. Thus, we must rebuild the surrogate model for each individual design point. A variation on this application is to use the surrogate model until we are close to the converged solution, and then use the high fidelity code to home in on the final value.

A category of models related to surrogate models is that of reduced order models (ROM). Both surrogate models and ROMs are sufficiently general in design and application that they may be classified separately, as subsets of one another, identically, or as close cousins. The difference between them is primarily one of mindset. Surrogate models are essentially attempting to produce a lower-dimensional surface which approximates a surface of higher dimensionality based on a set of sample points. It is very much engrained in a geometrical conceptualization of the approximation and problem spaces. Reduced order models, however, are focused on producing a model which recreates the dominant system behaviors while discarding those less important. Instead of viewing the problem space as a geometrical output space correlated to an input space, ROMs view the problem space as a black box, input-output system. While surrogate models think in terms of surfaces, ROMs think in terms of systems. Both concepts can be applied to describe the same problems, but the differences in their mindsets can cause some confusion if left unstated.

There are generally two approaches to constructing ROMs. The first is to take the fundamental system equations for the discipline of interest and analytically reduce them, usually via system mode shapes (i.e., eigenanalysis) for the problem of interest. This is common in structural analysis by taking advantage of structures with linear responses and superimposing individual vibration modes. It is also applicable to aerodynamic systems, although the responses in aerodynamics tend to be strongly nonlinear. Regardless, this approach tends to require a great deal of human effort to reduce the analytical equations for the problem at hand. Further, if one wishes to reduce a system for which the analytical equations are difficult to couple, as illustrated above, there may not be a convenient set of equations from which to start the reduction at all.

The second approach is to use system identification methods to determine system mode shapes from sample input-output sets. This has the advantage of operating numerically, as well as requiring no knowledge of the underlying system equations. Perhaps the most common of these methods, and the one utilized in this work, is that of *proper orthogonal decomposition*. By itself, POD merely identifies the system modes and the corresponding mode coefficients for each input sample, but these are used to form the basis of a reduced order model that represents the original system. The advantages to POD are that it is guaranteed to generate optimal system mode shapes, as well as automatically ordering and ranking the mode shapes according to their relative influence on the overall system dynamics. This information is highly convenient for discarding mode shapes which have very weak contributions, allowing for an efficient reduction in the model's dimensionality versus fidelity lost.

2.4.1 Proper Orthogonal Decomposition

The cores of ROMs constructed in this research are based on the proper orthogonal decomposition (POD). It is what determines the ROM basis vectors. POD is a numerical procedure analogous to eigensystem analysis for square matrices, which determines a set of basis vectors, analogous to eigenvectors, of the system modes. These modes are also weighted by their singular values, analogous to eigenvalues, which indicate the relative importance of each mode. POD can operate on any rectangular matrix, but in this context we utilize a technique known as the method of snapshots. A matrix of system outputs corresponding to various parameter inputs is constructed, a series of "snapshots" of the system state. This is often used for a system which evolves in time, hence the snapshot terminology, but is equally applicable to other parameters.

The points for the snapshots do not need to follow any particular pattern in parameter space, as the corresponding input parameter values attached to each snapshot are retained separately. This snapshot matrix is often centered to an average snapshot before it is decomposed via POD. The resulting basis vectors are then retained or truncated based on their relative importance as determined by the corresponding singular values. It is important to note that the singular values, and the basis vectors, are ordered by the POD method in decreasing order of importance, and thus retaining the first k modes corresponds to a k-order approximation to the original system containing the k most important modes. Hence the term, reduced order basis. Because of the importance of POD in this work, this section will go into relatively more detail than some of the other background topics.

Proper Orthogonal Decomposition refers to a matrix decomposition method that is used to determine an optimal set of orthogonal basis vectors for the matrix. It also determines the corresponding influence of each basis vector on describing the matrix, and orders the basis vectors accordingly. This process is also called (with more or less accuracy) Singular Value Decomposition (SVD), Principle Components Analysis (PCA), or Karhunen-Loéve Decomposition. Some names imply slightly different intended uses or additional processing, but they are all generally used for describing a similar procedure. The basis vectors and singular values produced are analogous to eigenvectors and eigenvalues for square matrixes, except that the procedure can be applied to rectangular matrices. The method is applied across a wide variety of fields, notably in circuit design.

The Method of Snapshots is a method used to model the behavior of a system by building a matrix out of corresponding sets of samples. These sets are often sensors sampled through time, for example, pressure or temperature sensors measuring a field which changes in time. These sets are referred to as snapshots of the system. The snapshot matrix is decomposed into a set of basis vectors using POD, and the snapshots projected onto the new basis. The system behavior can then be predicted from these projected coordinates and the basis vectors.

Since the basis vectors are ordered according to their importance in describing the data, it is common practice to truncate a possibly large set of basis vectors (i.e., many hundreds or thousands) and retain only the most important vectors, or mode shapes. This introduces some error into the resulting system behavior, but that error can be estimated by using the singular values, and thus limited. This is the basis of most POD-based ROMs.

Before detailing the mathematics of the procedure, it may be illustrative to view an example of the method. Consider constructing a model representing the pressure over the surface of an airfoil with respect to angle of attack. A set of snapshots of the pressure values at 100 nodes over the airfoil surface has been generated for a series of angles of attack and may be seen in Figure 2.5.

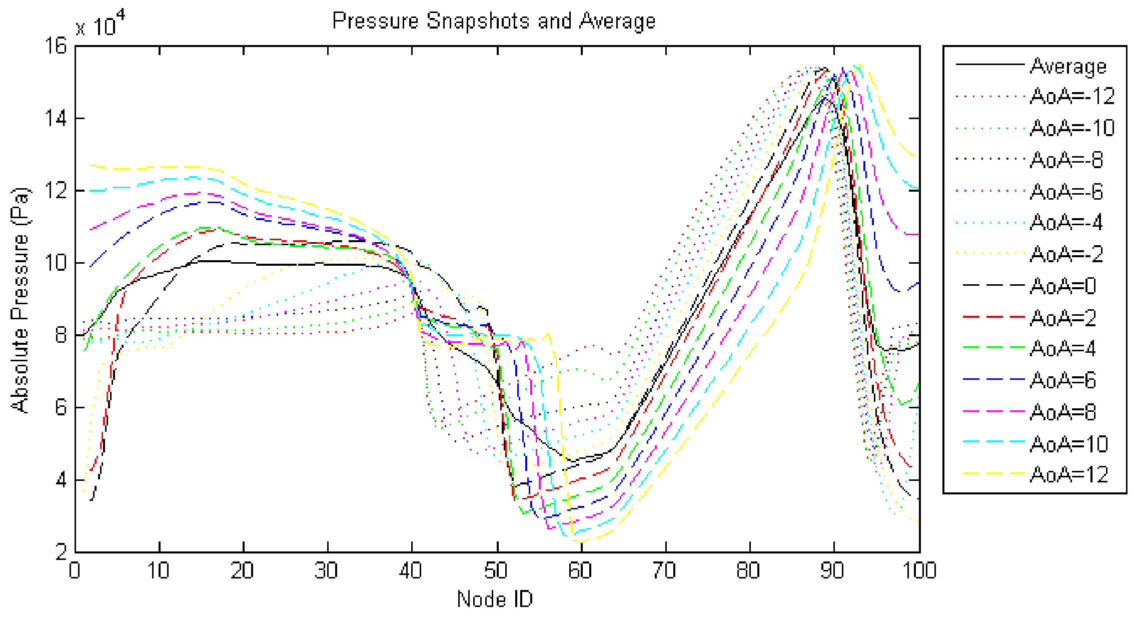

Figure 2.5 Airfoil Surface Pressure Snapshots and Average

A matrix is built of each snapshot. SVD is performed on the snapshot matrix, and the left-hand basis vectors taken as the new orthogonal basis set, or mode shapes. The first 12 shapes, and the corresponding singular values, can be seen in Figure 2.6 and Figure 2.7.

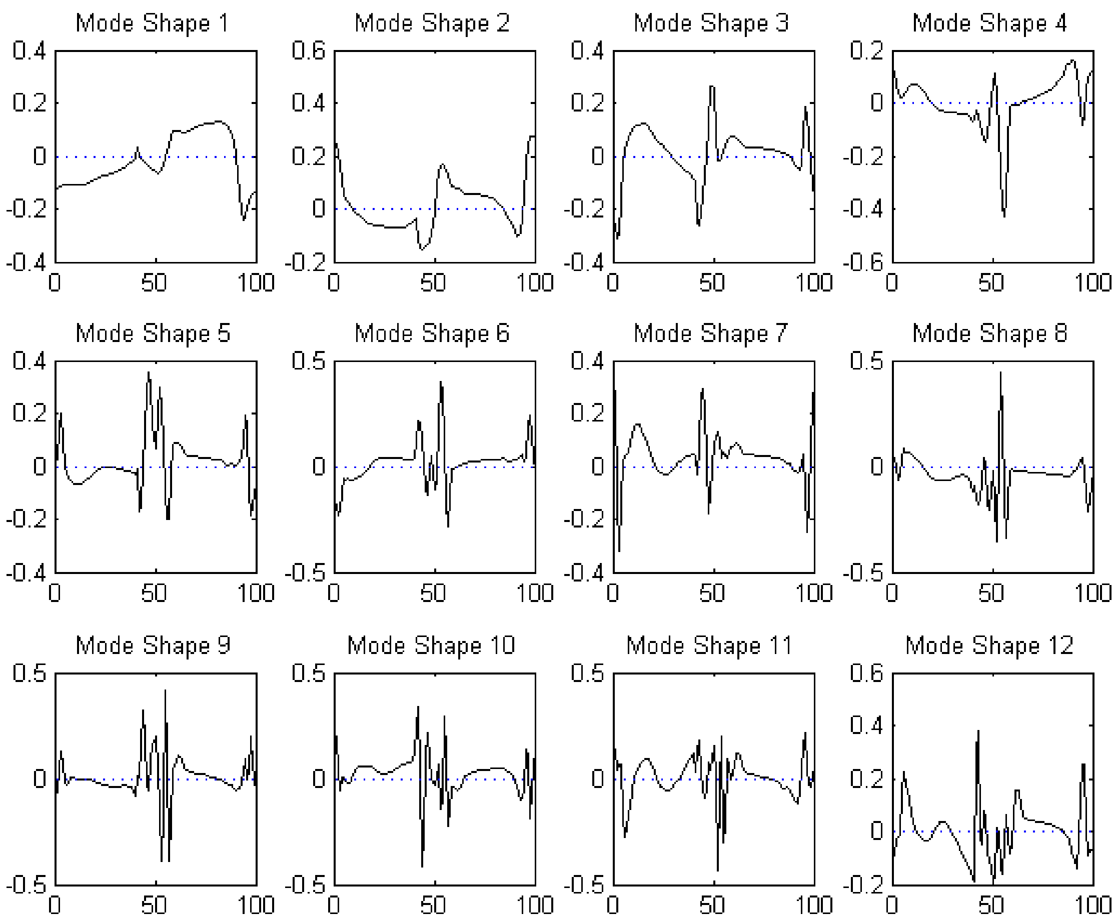

Figure 2.6 First 12 Mode Shapes of the Pressure System

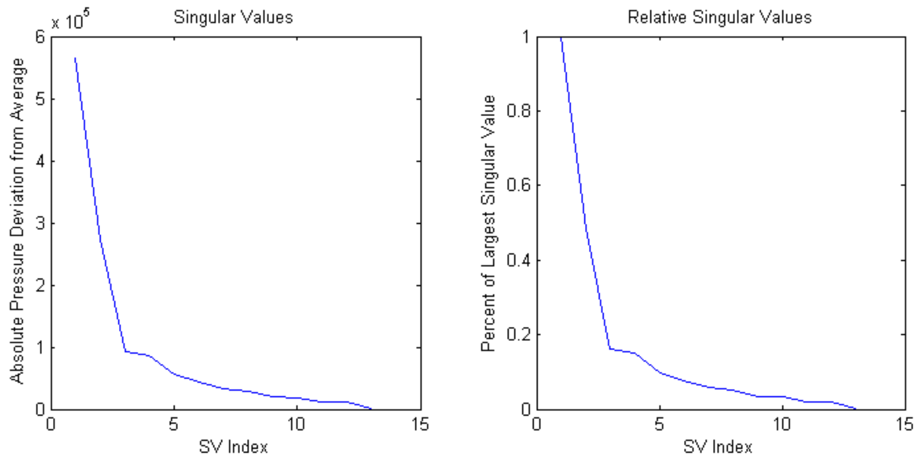

Figure 2.7 Singular Values of the Pressure System

The original snapshots can be projected onto the mode shapes to determine their coordinates in the new basis. By including all the mode shapes calculated, we can gain an exact reconstruction of each snapshot, as in Figure 2.8. However, if we truncate more of the modes, we introduce some error, although we still retain a large degree of accuracy, as seen in Figure 2.9. The error for the first 12 snapshots versus the number of modes retained can be seen in Figure 2.10.

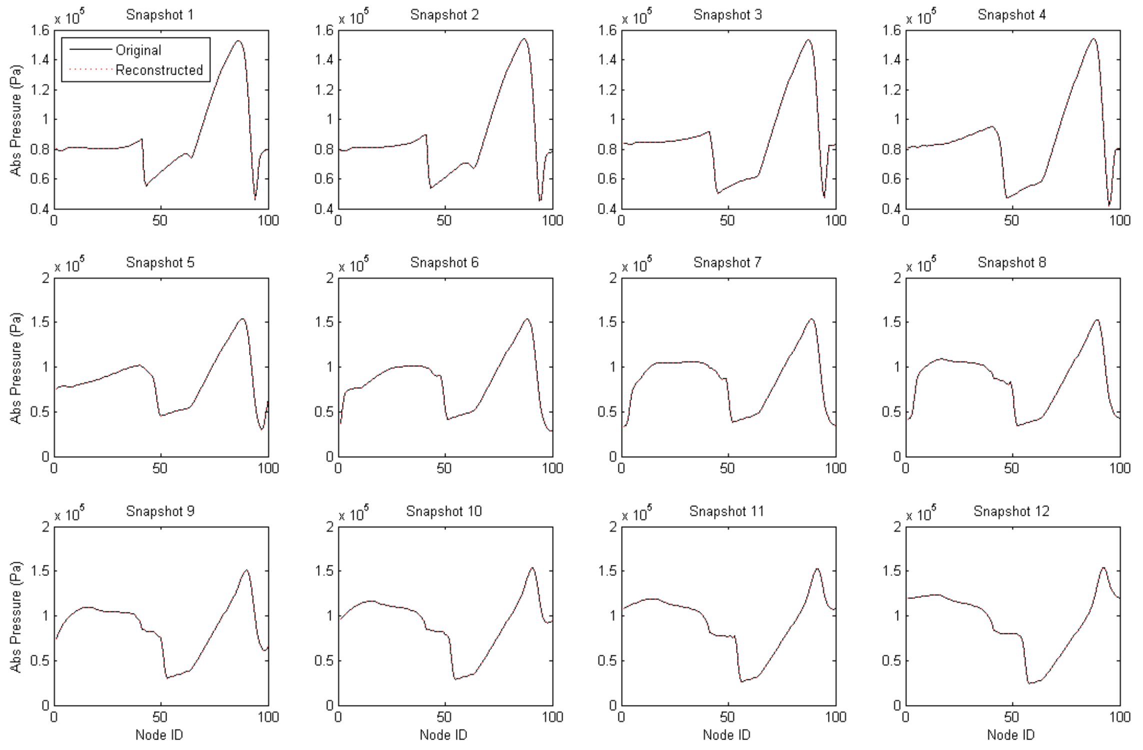

Figure 2.8 First 12 Snapshot Reconstructions (All Modes Included)

If we desire a pressure distribution for an intermediate angle of attack, we can interpolate the snapshot coordinates according to their corresponding angles of attack. A new pressure profile based on the basis mode shapes can then be reconstructed. We have now created an aerodynamic ROM for determining pressure distribution in response to an input of angle of attack.

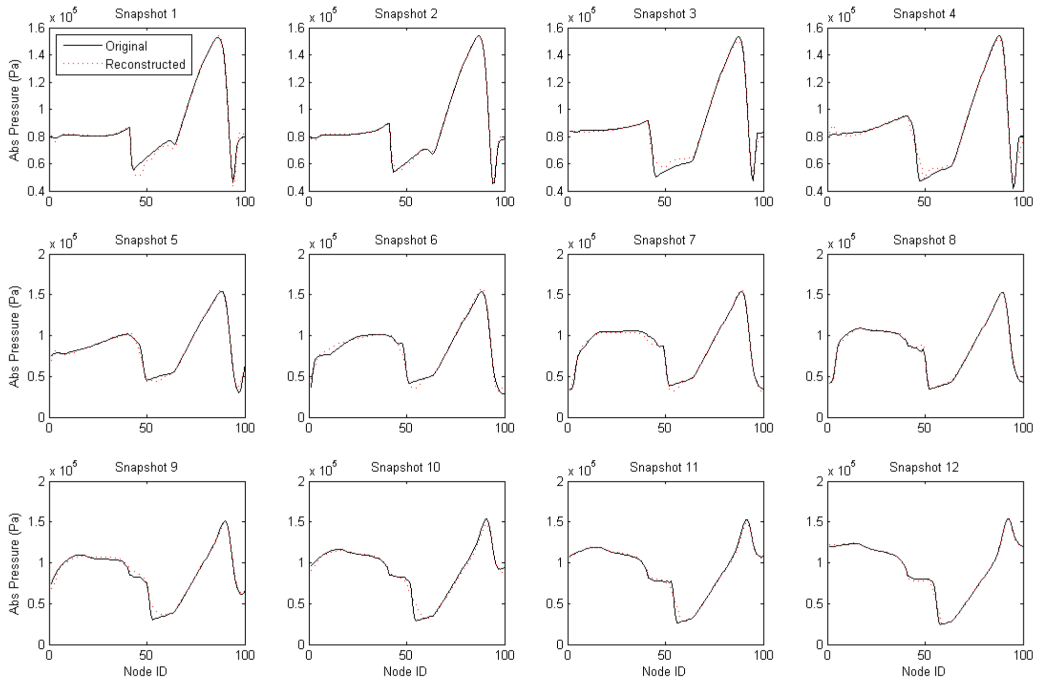

Figure 2.9 First 12 Snapshot Reconstructions (3 Modes Included)

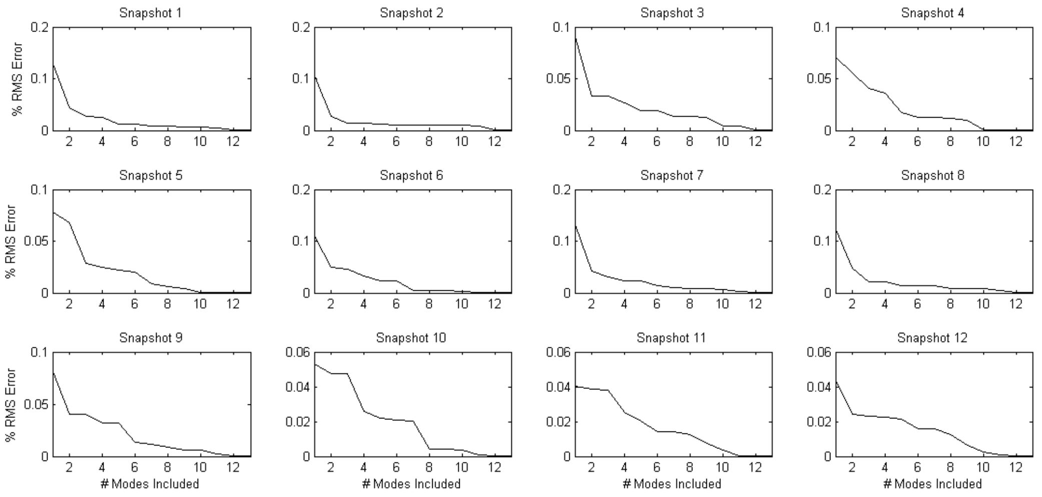

Figure 2.10 RMS Error for First 12 Snapshots vs. Number of Included Modes

SVD begins with a real or complex matrix, usually describing some dataset. For our purposes, we will be working with a real matrix. The procedure then decomposes the matrix into orthogonal bases in row and column space that are related by scaling factors. That is, for matrix A:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{-1} \tag{2.2}$$

If $\boldsymbol{A}$ were a square matrix, $\boldsymbol{U}$ would correspond to left eigenvectors arranged columnwise, $\boldsymbol{V}^{-1}$to right eigenvectors arranged rowwise, and $\boldsymbol{\Sigma}$ a diagonal matrix corresponding to the eigenvalues. For rectangular matrices, the values in $\boldsymbol{\Sigma}$ are referred to as singular values (hence the name of the method). The procedure for determining $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{\Sigma}$ is fairly straightforward. The following description is drawn largely from a recorded MIT lecture by Gilbert Strang.[155]

Since $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal bases, their inverse and transpose are identical. Thus we may write the above as:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{T} \tag{2.3}$$

We can find $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$ by premultiplying by $\boldsymbol{A}^{T}$:

$$\boldsymbol{A}^{T}\boldsymbol{A} = \boldsymbol{V\Sigma}^{T}\boldsymbol{U}^{T}\boldsymbol{U\Sigma V}^{T} = \boldsymbol{V\Sigma}^{2}\boldsymbol{V}^{T} \tag{2.4}$$

Which forms an eigenvalue problem that can be solved in the typical fashion. The procedure is similar for determining $\boldsymbol{U}$. $\boldsymbol{\Sigma}$ from each calculation ($\boldsymbol{U}$ and $\boldsymbol{V}$) should be identical, a convenient sanity check. We may then order the basis vectors such that the singular values proceed in decreasing value. For the purposes of this research, the built in SVD solver available in MATLAB is used, and the basis for the POD method taken to be $\boldsymbol{U}$.

To reduce the dimensionality of the system, we may simply truncate the number of vectors and singular values to retain or discard however many we desire. Thus, an order k approximation of the system will retain only the first k basis vectors and singular values. This reduced-order basis is at the heart of most POD-based reduced order models. The truncated order k set of basis vectors and singular values is thus denoted $\boldsymbol{U}_k$ and $\boldsymbol{\Sigma}_k$.

The method of snapshots is a common POD based ROM, and is the method considered here. First, a series of system "snapshots" with respect to time or some other indexing factor (such as angle of attack in the example above) is computed. The average of these snapshots is calculated and subtracted from each individual snapshot, resulting in a set of vectors representing the deviation of each snapshot from average. These adjusted snapshot vectors are then composed columnwise into a snapshot matrix, $\boldsymbol{M}$.

$$S_{avg} = \left(\tfrac{1}{n}\right) \textstyle\sum_{i=1}^{n} S_i \tag{2.5}$$

$$\boldsymbol{M} = \left[(S_1 - S_{avg}) \quad \cdots \quad (S_n - S_{avg}) \right] \tag{2.6}$$

SVD is performed on the snapshot matrix as described above, and truncated if desired to form an order k approximation. For this work, the columnwise basis vectors (denoted as $\boldsymbol{U}$ above) are taken to form the model basis. A check of the mutual orthogonality of each basis vector can be used as a sanity check at this stage. Each snapshot is then projected onto the new basis, resulting in a set of coordinates representing each individual snapshot as a linear combination of the new mode shapes.

New snapshots for arbitrary values of the input variable (time, angle of attack, etc.) can be calculated by interpolating coordinates for the snapshots. For example, if snapshots are available at 2° and 6° angle of attack, a snapshot at 3.44° could be calculated by linearly interpolating the new basis coordinates for the 2° and 6° snapshots. That is:

$$p = (3.44° - 2°)/(6° - 2°) \tag{2.7}$$

$$C_{3.44°} = (1 - p)C_{2°} + (p)C_{6°} \tag{2.8}$$

where C_i is the coordinates for a system snapshot at $i°$ angle of attack, and p is a normalized linear scaling factor. The new coordinates $C_{3.44°}$ can then be multiplied by the model basis and added to the snapshot average to determine the reconstructed snapshot.

$$S_{3.44°} = \boldsymbol{U}_k C_{3.44°} + S_{avg} \tag{2.9}$$

The generality of POD based models can make it very easy to become confused about inputs and outputs to the model system, particular when a portion of a coupled system which evolves in time is the subject of the model (as the aeroelastic system modeled here is). It may be helpful to examine a few related systems in order to more clearly illustrate their inputs and outputs.

For example, in the pressure model given as an example above, the steady-state pressure over the surface of the airfoil (output) is determined as a function of angle of attack (input). A few samples of the system state (surface pressures) are taken at a range of the input value (angles of attack). These sample snapshots are then used to construct a model and determine the output pressure for any input angle of attack by interpolating between the system mode shapes identified by the model. Note however that the model can only be expected to be valid for angles of attack within the range of the original dataset; this is an important point.

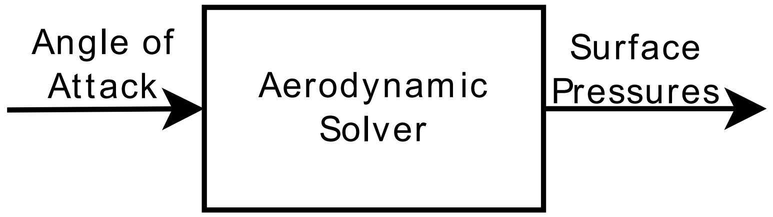

Figure 2.11 Example Angle-of-Attack/Surface Pressure System

The system considered in this preliminary problem concerns itself with modeling the aerodynamic forces generated by a physical displacement of the geometry. Thus, the input to the system is displacement of geometry, whether that is represented as a scalar (as for the angle of attack in the preliminary problem), a vector of displacements of mesh nodes (as in the larger 3D research problem), as an analytical description of the displacement (such as a combination of structural modes), or some other representation. The key point is that the input to the aerodynamic system is the geometric displacement. Likewise, the output of that system which is of interest to an aeroelastic problem will be the aerodynamic forces (and perhaps temperatures) on the surfaces of interest. Here, note that we do not concern ourselves with the evolution of the system in time, only with the system describing the response of surface forces to displacements. This type of system may be used in speeding up an aeroelastic analysis by replacing the aerodynamic solver with the faster ROM.

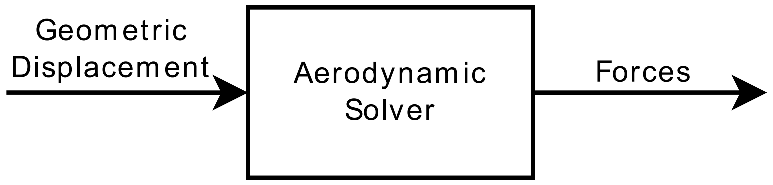

Figure 2.12 Example Displacement/Forces System

This can be contrasted with a model describing a full aeroelastic system. Here, the input to the system may be a description of initial displacement, flow conditions, and/or forces acting on the design of interest. The output of the system will be a time history of the forces and displacements of the design. Here, our snapshots would take the form of time histories, sampled at varying values of initial displacements or flow conditions. This might be used as part of a control system, producing a faster ROM capable of evaluating the response of a system in realtime.

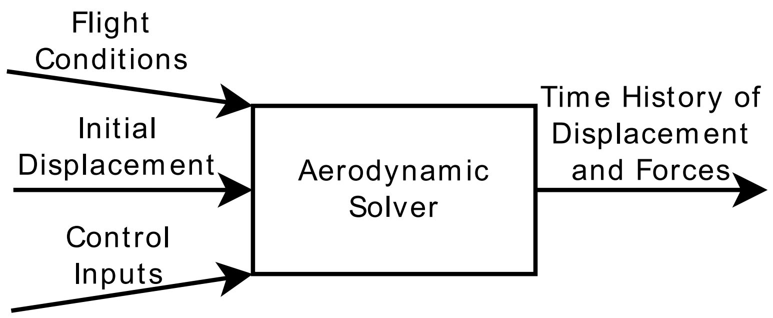

Figure 2.13 Example Aeroelastic System

A fourth model may take a design description of external geometry and structure as input, and determine as output an evaluation of the aeroelastic characteristics of the system (such as flutter speed). Here our snapshots may be scalar in nature, while our input samples may be very complicated. This type of model may be used to assist in optimization of a design by providing cheaper evaluations of the impact of design decisions.

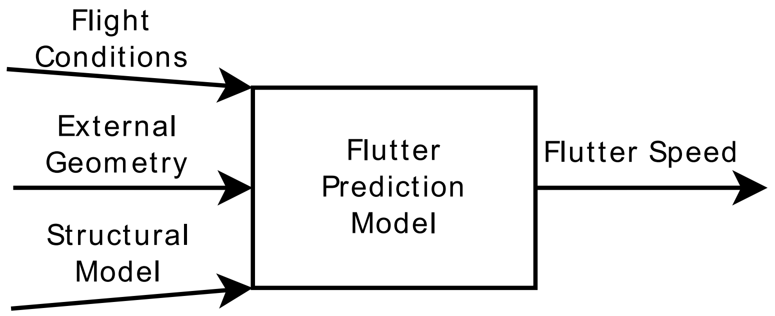

Figure 2.14 Example Flutter Predictor System

As can be seen in the above examples, it is critical to clearly identify the inputs and outputs of the system being modeled, and understand the context in which the model itself will be applied. The potential for confusion and complexity can be easily seen, especially when one considers that many of the above models could be nested within one another.

It should be noted that the model construction described above takes an interpolative approach to predicting output responses to input parameters. Alternatively, we could use POD to determine a reduced state equation for the system, provided the system is easily modeled with a state equation. Since we are primarily interested in dealing with complex, black-boxed multidisciplinary systems, this approach will not be addressed further here. For more details, please see Antoulas, Sorenson, and Gugorcin 2001.[4] Example MATLAB code for build POD-based ROMs is given in APPENDIX B.

2.4.1.1 Grassmann Manifold Projection

Interpolation of ROM basis vectors as well as coordinates is a very attractive idea, as it removes the necessity to calculate a new snapshot matrix for each design point. Unfortunately, interpolating the basis vectors between ROMs is somewhat more complicated than in the case of coordinates. Basis vector interpolation is still an active area of research.[118,137,139–143,145,156] It's useful to note that in this research, we utilize the same model order and snapshot length globally, which simplifies many of the interpolation considerations, but does not eliminate some of the fundamental problems. Basis vectors in our ROMs are right-handed and orthonormal, and preserving these properties proves to be a major challenge. The issue is compounded by attempting to interpolate with information from many different ROM models of arbitrary dimension.

The problem is readily apparently from attempting to interpolate an arbitrary number of basis vectors in three dimensional space; the results from direct interpolation are not guaranteed to produce a new orthonormal basis, nor even a basis which spans the original space at all. In three dimensions, specialized approaches have been developed such as decomposing into Euler angles or projecting into quaternion space. Neither approach is extensible to an arbitrary number of dimensions. However, quaternion projection provides a useful analog for the procedure considered here; the original basis vectors are projected into a different space which allows for straightforward interpolation, the result of which is then projected back into the original space.

Grassmann Manifold Projection (GMP) is very similar in this concept. A set of bases is projected into a tangent space which allows for direct interpolation methods, and then reprojected back to the original basis space. The result is a straightforward and efficient basis interpolation method which preserves the important properties of the bases for our applications: orthogonality, orthonormality, and span.

The algorithm for GMP used in this work comes primarily from Amsallem and Farhat 2008.[141] For more details on the mathematical background beyond the algorithm summarized here, the reader is referred to their work.

The setting for the algorithm begins with a set of design points, λ_i, for which we have constructed corresponding sets of reduced order bases, $\boldsymbol{\phi}_i$, of order k and dimension N_R. We are interested in determining an interpolated basis $\boldsymbol{\phi}_R$ at a new design point λ_R. First, we select one of the design points and bases as a reference and origin, λ_0 and $\boldsymbol{\phi}_0$. We then develop a projection into the tangent space, $\boldsymbol{\Gamma}_i$, for every other basis as follows:

$$(\boldsymbol{I} - \boldsymbol{\phi}_0\boldsymbol{\phi}_0^T)\boldsymbol{\phi}_i(\boldsymbol{\phi}_0^T\boldsymbol{\phi}_i)^{-1} = \boldsymbol{U}_i\boldsymbol{\Sigma}_i\boldsymbol{V}_i^T \tag{2.10}$$

$$\boldsymbol{\Gamma}_i = \boldsymbol{U}_i \tan^{-1}(\boldsymbol{\Sigma}_i)\,\boldsymbol{V}_i^T \tag{2.11}$$

We may now interpolate these gamma matrices element-wise based on their corresponding design points. In our case, we will utilize radial basis functions. The resulting interpolated matrix, $\boldsymbol{\Gamma}_R$, can be mapped back to the basis space as follows:

$$\boldsymbol{\Gamma}_R = \boldsymbol{U}_R\boldsymbol{\Sigma}_R\boldsymbol{V}_R^T \tag{2.12}$$

$$\boldsymbol{\phi}_R = \boldsymbol{\phi}_0\boldsymbol{V}_R\cos(\boldsymbol{\Sigma}_R) + \boldsymbol{U}_R\sin(\boldsymbol{\Sigma}_R) \tag{2.13}$$

It is worth noting that the set of bases for a given interpolation is sometimes restricted to bases that are relatively near to the interpolated point of interest; this is an effective strategy for Lagrange- or spline-based interpolation schemes. We will not apply this filtering approach here, as the use of RBFs should minimize the impact of distant bases on the interpolated result.

2.4.2 Radial Basis Functions

In the IROM method, the design of parameter inputs which are sampled for the snapshot matrix, used to generate the ROM corresponding to each design point, is applied identically to all points. Thus the same sample range is used for all ROMs. This allows us to interpolate coordinates vectors between the ROMs. This is accomplished using radial basis functions.

Radial basis functions, especially extended RBF, are very accurate for any scale with highly-nonlinear problems.[8] They do not require any systematic sampling scheme, although an appropriate experimental design can increase their accuracy. They operate by assuming that each sampled point is a basis function and that the value of the function at any given point is the result of a combination of all these bases, weighted by distance. Unlike polynomial regression, Kriging, or a variety of other methods, this method does not develop an intermediate approximation to the surface; it predicts values directly. This also has the advantage of making evaluations largely inexpensive. RBF are very popular in the literature.[3,8,10]

RBF begins with the basic response surface problem, that is, to develop an approximation $\bar{F}$ to a set of true and expensive function evaluations F corresponding to a set of n_p design points $x \in \mathbb{R}^m$. It does this by weighting a linear combination of radial functions, which operate on the distance between the data points and the evaluation point of interest. More explicitly:

$$\bar{F} = \sum_{i=1}^{n_p} \sigma_i \varphi(\left||x - x_i|\right|) \tag{2.14}$$

The various σ_i are referred to as the RBF coefficients, while φ is the radial function. There are several common choices, including:

Quadratic $$\varphi_Q(x) = x^2 + c^2 \tag{2.15}$$

Inverse Quadratic $$\varphi_{IQ}(x) = 1/(x^2 + c^2) \tag{2.16}$$

Multiquadratic $$\varphi_{MQ}(x) = \sqrt{x\hat{}2 + c\hat{}2} \tag{2.17}$$

Inverse Multiquadratic $$\varphi_{IMQ}(x) = 1/\sqrt{x\hat{}2 + c\hat{}2} \tag{2.18}$$

Here, $c > 0$ is an arbitrary parameter. By applying the constraint that the model must match the true function values at all sample points, we arrive at the set of equations that can be used to determine the approximation coefficients:

$$\sum_{i=1}^{n_p} \sigma_i \, \varphi(\left||x_k - x_i|\right|) = F(x_k) \qquad k = 1{:}\,n_p \tag{2.19}$$

Or, in matrix form:

$$\boldsymbol{A}\sigma = F \tag{2.20}$$

$$A_{ik} = \varphi(\left||x_k - x_i|\right|) \tag{2.21}$$

This can be solved for σ in a straightforward manner. Example MATLAB code providing this functionality is given in APPENDIX C. In this work, the IMQ radial function is favored. An example of a radial basis function can be seen in Figure X. Here, random curves are generated representing an aerodynamic coefficient measured across a two dimensional parameter space – airfoil thickness (T), and angle of attack (AoA). The

RBF is then sampled at regular angle of attack intervals and random thickness intervals and plotted against the snapshot curves.

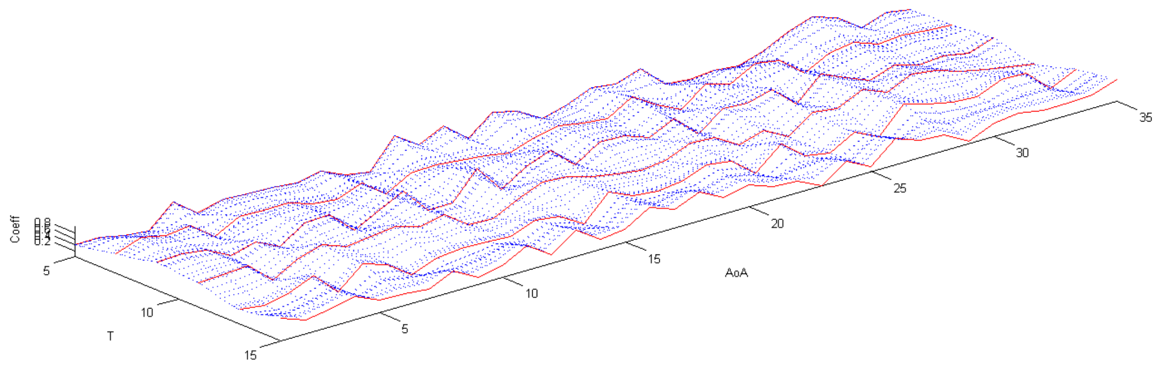

Figure 2.15 Radial Basis Function Example

A notable derivative method is called extended RBFs (E-RBFs). This method adds an additional term to the approximation involving the use of non-radial functions. These nonradial functions as given by Mullur and Messac are a piecewise function with arbitrary parameters controlling the divisions between the different regions and their behavior. The combined function is linear on the outer regions and has controllable nonlinear behavior in the inner regions. The resulting system is underdetermined and has a family of solutions, however the simple pseudoinverse provides an efficient and least-norm solution. This allows the imposition of additional constraints such as smoothness and convexity, which can be useful for gradient based optimizations. The reader is referred to Mullur and Messac's work for further details on those aspects.[10]

2.4.3 Extended Radial Basis Functions

ERBFs were developed by Mullur and Messac[10] to incorporate non-radial basis functions as well as radial basis functions. This increases the overall ROM accuracy while allowing the ROMs to accurately represent linear portions of a function, a classic weakness of regular RBF. The tradeoff is that the extra NRBFs introduce many more unknown coefficients, resulting in an underdetermined set of equations which must be solved by a constrained linear programming subproblem. If the subproblem fails to find a feasible solution, the system may be solved by a pseudo inverse procedure, which is equivalent to the typical RBF solution method. Thus, ERBFs provide at least as much accuracy as regular RBF, with the capability of improving that accuracy significantly. For a detailed description, the reader is referred to their work, however we will provide a summary overview here for completion. Example MATLAB code implementing ERBFs is available in 0.

Table 2.1 Definitions of ERBF Coefficients

ξ_d^i	Φ^L	Φ^R	Φ^β
$\xi_d^i \leq -\gamma$	$(-n\gamma^{n-1})\xi_d^i + \gamma^n(1-n)$	0	ξ_d^i
$-\gamma \leq \xi_d^i \leq 0$	$(\xi_d^i)^n$	0	ξ_d^i
$0 \leq \xi_d^i \leq \gamma$	0	$(\xi_d^i)^n$	ξ_d^i
$\gamma \leq \xi_d^i$	0	$(n\gamma^{n-1})\xi_d^i + \gamma^n(1-n)$	ξ_d^i

ERBF begins with the same inputs as RBF, namely the input points X, the output samples Y, and a radial basis function φ. ERBF also uses two additional parameters, a smoothing factor γ, and a nonlinear order factor n. Using these parameters, the approximation to the original system function is given by

$$\tilde{F}(X) = \sum_{i=1}^{n_p} \sigma^i \varphi(\|X - X^i\|) + \sum_{i=1}^{n_p} \sum_{d=1}^{n_{xd}} [\alpha_d^{Li} \Phi^L(\xi_d^i) + \alpha_d^{Ri} \Phi^R(\xi_d^i) + \beta_d \Phi^\beta(\xi_d^i)] \tag{2.22}$$

where σ, α^L, α^R, and β are the unknown coefficients. Here, the first term represents the normal RBF contribution, while the second term represents the NRBF contribution. The functions Φ^L, Φ^R, and Φ^β are defined as in Table 1, where ξ^i is defined as $X - X^i$, that is, ξ_d^i is the difference between X and X^i along dimension d. To construct the set of equations we will solve, we first construct the RBF weighting matrix as per regular RBF

$$A_{ij} = \varphi(\|X^j - X^i\|) \tag{2.23}$$

Additionally, we construct a NRBF weighting matrix $\boldsymbol{\bar{B}}$. This matrix is constructed from rows $\bar{B}^k$ where the structure of the kth row is

$$\bar{B}^k = [\bar{B}^{Lk}\ \bar{B}^{Rk}\ \bar{B}^{\beta k}] \tag{2.24}$$

Given $\xi^{ij} = X^j - X^i$, each term in this structure is a row vector given by

$$\bar{B}^{\cdot k} = \left[\Phi^{\cdot k}(\xi_1^{k1})\ \Phi^{\cdot k}(\xi_2^{k1}) \cdots \Phi^{\cdot k}(\xi_{n_{xd}}^{k1})\ \Phi^{\cdot k}(\xi_1^{k2}) \cdots \Phi^{\cdot k}\left(\xi_{n_{xd}}^{kn_p}\right)\right] \tag{2.25}$$

$$\cdot \equiv L, R, \beta$$

These two matrices $\boldsymbol{A}$ and $\boldsymbol{\bar{B}}$ are combined into a matrix $\boldsymbol{\bar{A}} = [\boldsymbol{A} \quad | \quad \boldsymbol{\bar{B}}]$ as an underdetermined system. We construct our vector of unknowns as a column vector $\bar{\alpha}_{yd} = [\sigma_{yd} \quad \alpha_{yd}^L \quad \alpha_{yd}^R \quad \beta_{yd}]^T$ for each dimension yd of the output space, and set it equal to the column vector of samples for that dimension, F_{yd}. This results in the system

$$\bar{\boldsymbol{A}}\bar{\alpha}_{yd} = F_{yd} \tag{2.26}$$

This is an underdetermined system that we can solve via a constrained linear programming subproblem given by

$$\min_{\bar{\alpha}_{yd}} b^T \bar{\alpha}_{yd} \mid \bar{\boldsymbol{A}}\bar{\alpha}_{yd} = F_{yd};\ \bar{\alpha}_{yd} \geq 0 \tag{2.27}$$

where b^T is taken to be a vector of ones. If no feasible solution is found, we solve for $\bar{\alpha}_{yd}$ using the pseudo inverse procedure, i.e. $\bar{\alpha}_{yd} = \bar{\boldsymbol{A}} \backslash F_{yd}$, which will yield a regular RBF solution. Once the vector of unknowns is calculated, we can evaluate a new parameter point P according to the approximation formula given above.

CHAPTER III

ROM INTERPOLATION SCHEME

3.1 IROM Scheme

Generally, the most computationally expensive component of an optimization is the full-order analysis for evaluations of the objective function(s) and design constraint(s). Any reduction in the computational cost of performing this analysis, or the number of times it must be performed, has the potential to provide significant speedup benefits to the optimization. This expense is particularly pronounced when considering the finite difference stencils.

In a typical gradient-based optimization, there is a clear analysis step where the partial derivatives of objective and constraints are evaluated across a finite difference stencil of design points. The strategy adopted in this work is to apply the proposed interpolation procedure to the gradient evaluation / sensitivity analysis (SA) step. That is, new ROMs will be interpolated for each of the SA stencil points instead of performing FOAs. This will allow the evaluation at each stencil point to be performed using the less expensive interpolated ROMs. To differentiate interpolated ROMs from those constructed using full order data, they will be referred to here as IROMs. Unlike previous interpolation work, this method does not require additional FOAs and operates with black box analyses. An illustration of the interpolation process is shown in Figure 3.1.

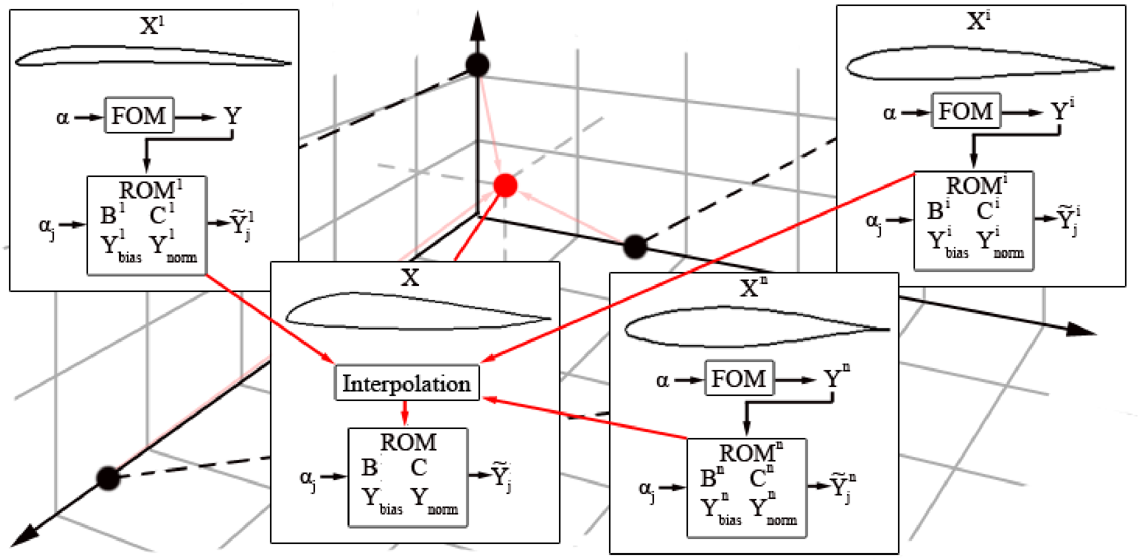

Figure 3.1 Illustration of ROM Interpolation Concept

The first part of the scheme is an offline computation. Before beginning the optimization, a database of ROMs is constructed throughout the design space. To do this, first a set of training points is constructed via a DOE design. In this work, Latin hypercube designs are generated and selected according to maximin criteria, which maximizes the minimum distance between sample points, leading to good space filling properties. At each training point, data from an FOA at that point is used to construct a ROM. These will form a reference set of ROMs to be utilized during interpolation.

The online portion of the scheme occurs during the optimization procedure itself. During sensitivity analysis for each design point, the reference ROMs are interpolated to create new IROMs associated with each SA stencil point. These are then used to evaluate the objective and constraint functions at those points. For the current design point itself, an FOA is performed. The results are then used to compute the gradient at the center point.

Optionally, the FOA data for the center point can be used to construct a new ROM for that design point. This ROM can be added to the database of reference ROMs and is subsequently available during any ROM interpolation. This is referred to as an adaptive step.

While this may increase accuracy, it also requires updating the interpolation model. For a given set of reference ROMs, much of the information required to perform the interpolation is independent of the target design point. This information can be stored to save computational time, and is referred to here as the interpolation model. However, when the reference set of ROMs changes, this model must be rebuilt, incurring an additional cost each time the reference set is updated.

The interpolation process itself is based on the component methods discussed previously in CHAPTER II. In this work, the ROMs are based on POD, which means that they are primarily represented by a set of basis vectors, $\boldsymbol{B}$, a set of coordinates $\boldsymbol{C}$ and any associated bias and normalization information. To interpolate between these ROMs, their components are interpolated separately.

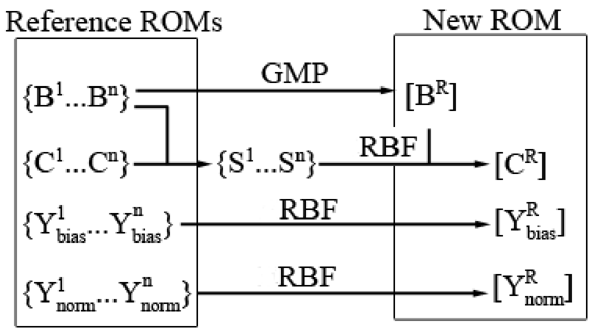

Figure 3.2 Illustration of ROM Interpolation Procedure

The basis vectors are interpolated using Grassmann manifold projection (GMP), using the nearest point as the reference basis. The snapshot coordinates of each ROM are in potentially in terms of a different basis; thus they are first reconstructed into the original snapshot space before being interpolated elementwise using RBF. The interpolated snapshots are then reprojected onto the new interpolated basis. Any bias and normalization information is also interpolated elementwise using RBF. This process is summarized in Figure 3.2.

3.1.1 IROM Algorithm

A more formal specification of the algorithm is given below:

1. Reference Database Generation
    a. Before the optimization, generate a set of design points XT, preferably spanning the design space in a manner which is optimal by some DOE criteria.
    b. As per standard POD method of snapshot procedure, create a ROM, ROM^i, for the analysis to be replaced. Retain the snapshot coordinates, $\boldsymbol{C}^i$, and the first k basis vectors, $\boldsymbol{\Phi}_k$, for each ROM. Also retain any biasing and normalization information. Thus, the data for a single ROM in the database consists of:$[X, \boldsymbol{C}, \boldsymbol{B}, Y_{bias}, Y_{norm}, P_{bias}, P_{norm}]$

       Store in a database for reference.
2. ROM Interpolation

   a. During the optimization, for each point along the optimization path X, perform an FOA. Optionally, construct a new ROM, ROM^X, and add this ROM to the reference database.
   b. For sensitivity analysis, generate a set of design points $\{X^{FD}\}$ as per the desired finite difference scheme.
   c. For each of the points $X_R \in \{X\}_{SA} X^R \in \{X^{FD}\}$, interpolate a new ROM, $IROM^R$:
      i. Interpolate the ROM bases using GMP and RBF, choosing the nearest ROM as reference.
      ii. Interpolate the bias and normalization information using RBF.
      iii. Reconstruct the snapshots for each reference ROM, interpolate using RBF, and then project onto the interpolated basis.
3. Reduced Order Analysis
   a. Using the new IROM data, $[X^R, \boldsymbol{C}^R, \boldsymbol{B}^{\boldsymbol{R}}, Y_{bias}^R, Y_{norm}^R,\ P_{bias}^R, P_{norm}^R]$, perform the ROA utilizing the IROM in place of the selected disciplinary analysis
   b. Evaluate the objective and constraint functions for the ROA output, $f(\tilde{Y}(X^R))$ and $\{g_i(\tilde{Y}(X^R))\}$.
4. Sensitivity Analysis
   a. Using the resulting evaluations, perform finite difference calculations to determine an estimated value of $\partial f/\partial X$ and $\left\{\frac{dG}{dX}\right\}$. Proceed with the optimization as per usual.

3.2 Error Metrics

In order to evaluate the error associated with using the IROM scheme in optimization, as opposed to the normal FOM optimization, it is useful to quantify several different error metrics. For most of these, the basic error measure will be that of Normalized Root Mean Square Deviation (NRMSD). Normalized Maximum Deviation (NMAX) will also provide some useful insight. Generally, NRMSD will be used to quantify the general accuracy while NMAX provides an indicator as to the magnitude of outliers. The specific formulation of these metrics for this work are given as

$$NRMSD = \sqrt{\sum_i \left\| Y^i - \tilde{Y}^i \right\|_2^2 \Big/ \sum_i \| Y^i \|_2^2} \tag{3.1}$$

$$NMAX = \frac{\max_i \left\{ \left\| Y^i - \tilde{Y}^i \right\|_2 \right\}}{\sqrt{1/n_p \sum_i \left\| Y^i - \tilde{Y}^i \right\|_2^2}} \tag{3.2}$$

Here, Y is the true value, or in our case the full-order value, and $\tilde{Y}$ is the approximation. n_p simply denotes the number of points.

The first errors to address are specific to the ROMs being used. POD-based ROMs may have a truncated set of basis vectors, which introduces truncation error to model outputs. In this work, the impact of truncation is not addressed, and all POD-based ROMs retain their full basis vector sets. Thus, truncation error is expected to be zero to machine accuracy; this is verified via NRMSD as a sanity check when constructing the initial ROM database.

Both POD- and RBF-based ROMs have error associated with predicting the function values for new input points. This represents the accuracy the model fit to the true function. These can be evaluated against the true function output with both NRMSD and NMAX. Generally in this work, a good NRMSD fit is considered the primary indicator, and NMAX is considered supplemental information about significant outliers.

When discussing the error associated with the interpolation procedure itself, it is useful to differentiate between global and local accuracy. Global accuracy refers to the ability of the interpolation model to match the full order model throughout the design space. This is a measure of the interpolation model's general fit. Local accuracy refers to the ability of the interpolation model to capture variations of the full-order model in the vicinity of a selected point. This is important when considering the ability of the interpolation procedure to properly approximate gradients.

The global error in this work is estimated after the initial ROM database construction by selecting a set of random test points throughout the design space. For each of these points, the results of the FOA and an IROM interpolated to that point are compared with NRMSD and NMAX.

The local error is evaluated in the same manner, on a set of random test points. Finite different stencils are generated for each test point, and FOA and IROM evaluations performed for each stencil point. The spectral angle between the computed gradients can then be used to measure the local error. By constructing a ROM for the test point and updating the interpolation model, the impact of the adaptive step can also be measured. The local error is sometimes referred to here as the spectral error (SPERR). The formula used here scales the spectral angle by 180 degrees, in order to report the error as a percentage. That is, 100% error would be a gradient pointing directly opposite the true gradient. The specific formula used in this work is given as

$$SPERR = \cos^{-1}\left(\frac{Y^i.\tilde{Y}^i}{|Y^i||\tilde{Y}^i|}\right)/\pi \tag{3.3}$$

Since the work here is concerned primarily with sensitivity analysis, the local error is considered the primary indicator of a good interpolation model. The remaining errors provide additional information and insight into the behavior and characteristics of the model. All of these errors can be evaluated after constructing the initial ROM database and interpolation model, and provide valuable feedback about the choice of DOE design.

For the effect of the model on the optimization itself, this is primarily considered through comparing the objective histories and design point paths between the two optimizations. Since it is expected that the paths will not be exactly the same, evaluating the local accuracy at each point along the optimization path would involve either maintaining an interpolation model (for the FOM optimization) or evaluating the FOM model at each point (for the IROM optimization). This is not performed in the current work due to time limitations and the difficulty it introduces to obtaining accurate timing and function count information for the respective methods.

CHAPTER IV

TEST PROBLEMS

4.1 Airfoil Shape Optimization

The airfoil shape optimization (ASO) test problem was selected as a simple test problem for illustrating the viability of the IROM method and potential speedup. The problem is based loosely on a problem by Vanderplaats and Hicks.[157]

The goal of the ASO problem is to maximize the average lift-to-drag ratio of an airfoil over several angles of attack. The airfoil must also meet a minimum lift coefficient at all angles of attack and a minimum quarter chord thickness. The flow conditins are chosen to represent standard sea-level with a velocity of 50 m/s.

The design airfoil shape is represented by blending or combining several different basis airfoil shapes.

More specifically, the upper and lower surfaces of four standard NACA airfoils are used for the basis weight factors: NACA 2412, NACA 64A215, NACA 65-415, and NACA 64-412. The weight factor given to each particular basis airfoil is treated as a design variable that ultimately determines the resultant shape of the design airfoil. Thus, a design point X is defined by the magnitude of the weight corresponding to each basis airfoil, and is synonymous with a particular airfoil shape. This results in eight total weight factors, four for the upper surfaces, and four for the lower surfaces. The weight factors are allowed to vary from 0.0 to 1.5, with a minimum bound imposed on the sum of weight factors for the upper and lower surfaces. The lower bound on the sum is to avoid excessively flat surfaces. An illustration of the airfoil shapes and blending them is given in Figure 4.1.

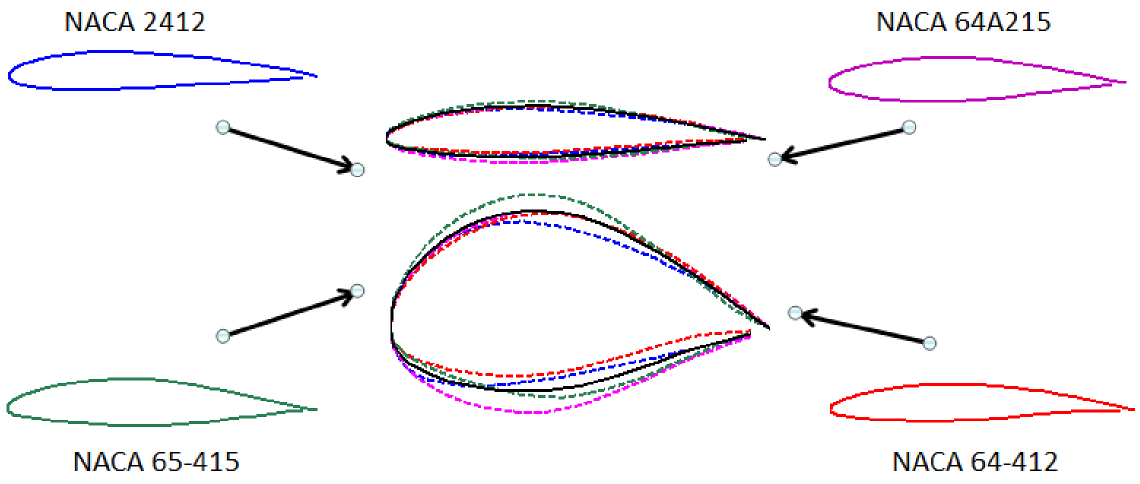

Figure 4.1 Illustration of Blended Airfoil Shapes

The ASO problem can be more formally specified as a constrained nonlinear programming problem expressed as

$$\min_{\boldsymbol{X}=[w_1 \cdots w_{xd}]} \quad F(Y(\boldsymbol{X})) = -\frac{1}{n_\alpha}\sum_{\alpha_j} L(\boldsymbol{X},\alpha_j)/D(\boldsymbol{X},\alpha_j)$$

$$\text{s.t.} \quad g_1(Y(\boldsymbol{X})) = c_L^{min} - c_L \leq 0$$

$$g_2(Y(\boldsymbol{X})) = t_{QC}^{min} - t(0.25) \leq 0$$

$$g_3(Y(\boldsymbol{X})) = w_{upper}^{min} - \sum_{i=1}^{4} w_i \leq 0$$

$$g_4(Y(\boldsymbol{X})) = w_{lower}^{min} - \sum_{i=5}^{8} w_i \leq 0$$

$$w_i^{min} \leq w_i \leq w_i^{max}; i = 1, n_{xd} \tag{3.1}$$

Here, $\boldsymbol{X}$ represents the design variables, i.e. the collection of weight factors $w_1 \ldots w_{xd}$. $Y(\boldsymbol{X})$ indicates the analysis required to determine the lift, $L(\boldsymbol{X}, \alpha_j)$ and drag, $D(\boldsymbol{X}, \alpha_j)$ characteristics for each of n_α angles of attack. These responses are used to formulate the objective function, F as the inverse of the average lift-to-drag ratio. The constraints are formulated to enforce a minimum lift coefficient, a minimum quarter chord thickness, bounds on the summed weights for the upper and lower surfaces, and side constraints on the values of the weights.

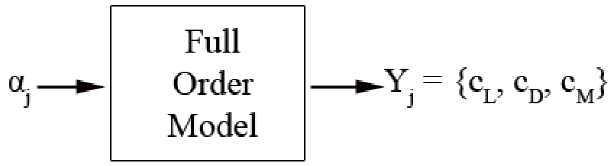

Figure 4.2 Full Order Model for ASO Problem

To apply the IROM scheme, it is critical to identify which system the ROMs are built to replace. For the ASO problem, the full order analysis as shown in Figure 4.2 is taken to be the aerodynamic solver. This system takes an input angle of attack and produces an output set of airfoil coefficients. The ROMs are constructed to replace this system, taking the same inputs and outputs as shown in Figure 4.3.

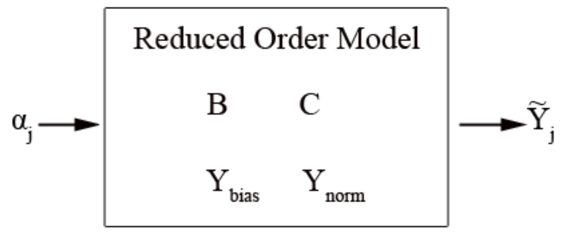

Figure 4.3 Reduced Order Model for ASO Problem

The fluid solver used for this problem is ANSYS Fluent, with grids generated through ANSYS ICEM-CFD. The grid is a structured quad grid with a typical C-grid topology and approximately fifteen chord lengths between the airfoil and the external boundary. The simulation is performed with the pressure-based Navier-Stokes solver and the SST k-ω turbulence model, with pressure far field boundary conditions at a Reynolds number of approximately 3.1e6. The overall optimization and analysis automation was performed in MATLAB.

4.2 Transport Wing Optimization

The second test problem, Transport Wing Optimization (TWO), is taken largely from papers by Garcelon et al and Venter and Sobieski.[158–160] It is a simplified optimization of the airfoil thickness and aspect ratio for a transport aircraft wing similar to the early Boeing 767 class. Venter and Sobieski test a particle swarm optimization technique on the problem; in contrast, this work applies the more traditional gradient-based sequential quadratic programming (SQP) algorithm. Since the objective space is fairly noisy for gradient-based optimization, the starting point in this work is altered to be in the vicinity of the gradient-based optimum found in the previous works. This allows a comparison of the final optimum points by way of a sanity check on the problem implementation, although it is not expected that they will be exactly the same due to key changes in the problem. These changes include use of different panel materials, slightly different load factors, and different buckling equations, among others. The interpolated ROMs are used to replace the finite element analysis in the structural subproblem.

Whereas the ASO problem was chosen as a proof-of-concept case to explore the speedup benefits of the IROM method, the TWO problem was chosen to examine the impact of utilizing IROMs within a larger system. Although the fluid analysis for this problem is simplified, the structure of the problem matches well with that of a multidisciplinary problem.

The TWO problem is a structured as a multilevel problem involving a system level range optimization of the wing by varying airfoil thickness, t/c, and the overall aspect ratio, AR. The wing reference area, sweep angle, and taper ratio, as well as the aircraft properties such as takeoff gross weight (TOGW) and ratio of nonstructural weight to gross takeoff weight, are held fixed.

For each system level design point, the minimum wing box weight is found by running a subproblem optimization on thicknesses of the wing spars, ribs, and surface panels. The structural design must meet stress constraints for two load factors, -1.5 g and +3.5 g. These load factors are distributed by a normalized combination of spanwise and chordwise loading distributions, shown in Table 4.1, from root to tip and leading to trailing nodes, respectively. The loads are applied on the bottom nodes only. The loading on each node is determined as a combination of its spanwise and chordwise location index found as

$$F_{ij} = WL \cdot DS_i \cdot DC_j \cdot A_{ij} \cdot LF \cdot \frac{\alpha+\beta_j}{\alpha_{ref}} \tag{4.2}$$

where F_{ij} is the force on the node, WL is the wing loading, DS_i and DC_j are the distribution factors for spanwise and chordwise directions, respectively, A_{ij} is an area associated with each node, LF is the current load factor, and the remaining term is a linear scaling factor based on the current angle of attack and displacement.

Table 4.1 TWO Load Distribution Factors

Location	1	2	3	4	5	6	7	8	9
Span	0.1786	0.1696	0.1518	0.1429	0.1250	0.1071	0.0893	0.0268	0.0089
Chord	0.1270	0.3175	0.3175	0.1587	0.0794				

The previous work also varied the number of internal spars and ribs, which will not be performed here. The spars and ribs are modeled as Al 6061, while the surface panels are modeled as aluminum sandwich panels consisting of Al 6061 face sheets with Divinycell F40 foam as the core material. The relevant material properties used for each of these can be found in APPENDIX F.

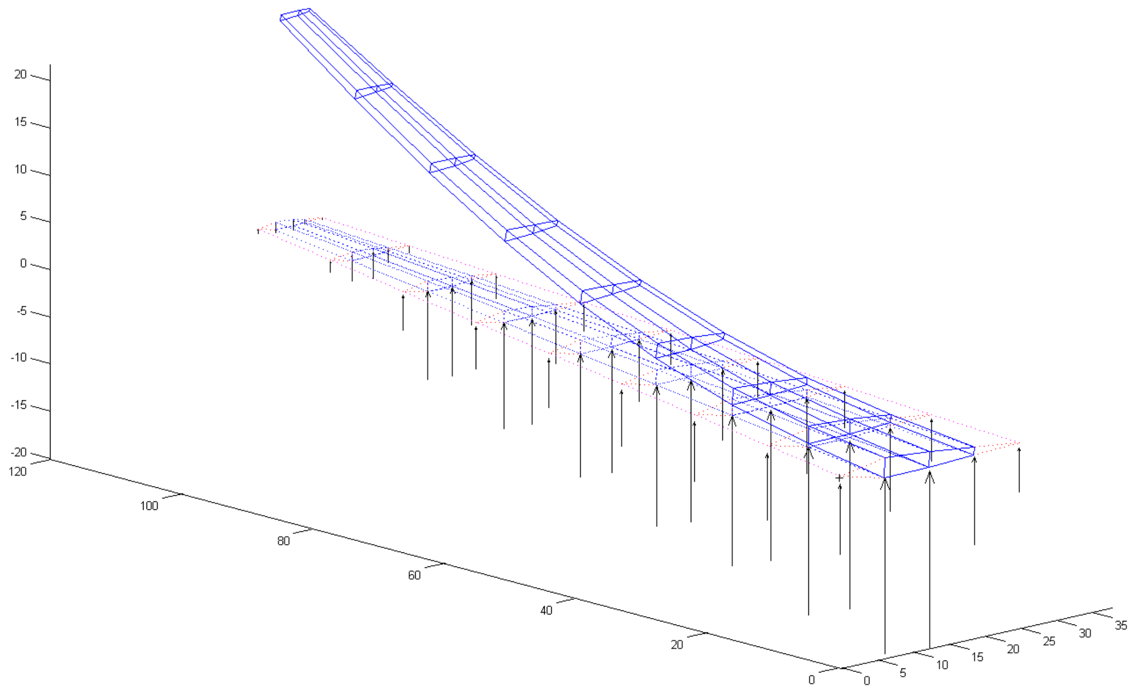

Figure 4.4 Example of Displaced Wing Box Model

Only the wing box and the upper and lower surface panels framed by the spars and ribs are modeled; the surface panels of the leading and trailing edges of the wing are not modeled. There is no rib in the root chord, as those nodes are fully constrained. There are eight ribs at equal spacing, including the tip rib, which divide the wing box spanwise into eight sections. The three spars are placed at the 25%, 50%, and 75% chord positions. Additionally, triangular "rigid" elements are used to transfer forces from the leading and training edges to the wing box; these are RBE3 elements from the Nastran analysis program, which do not add stiffness to the model but instead act to distribute forces. All other elements are modeled with 2D quadrilateral membrane elements, specifically CQUAD4 elements. The aluminum sandwich material is specified using a PCOMP property card, which converts the material internally into an equivalent MAT2 structure. An example of the wing box displaced under loading is given in Figure 4.4, with exaggerated displacements.

For each spanwise region of the wing box framed by ribs, the spars share a single thickness design variable, the associated outboard rib has an independent design variable, and each surface panel (four total per section) has independent core and total thickness design variables. This creates a total of eighty thickness design variables, ten per wing box section. An illustration of the wing box structural elements for one wing section, omitting three of the skin panels, is show in Figure 4.5.

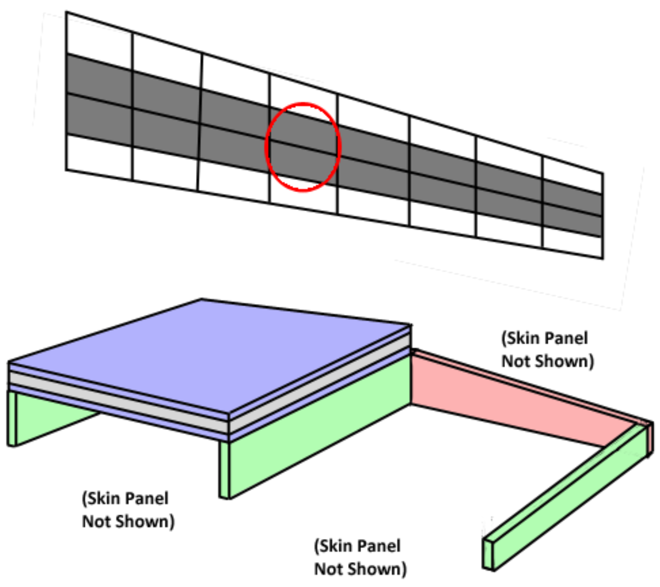

Figure 4.5 Illustration of wing box structural elements

The system level problem is a search for the maximum range configuration, constrained only by upper and lower bounds on the system level design variables. The structural subproblem is constrained by upper and lower thickness bounds, von Mises stress constraints on the aluminum elements, and buckling constraints on the sandwich skin panels. Additionally, each panel core thickness is constrained to be at least two minimum skin thicknesses below the total panel thickness.

The specification for the system level problem is given as

$$\begin{aligned}
&\text{min} && -R(\boldsymbol{X}) = -c_R \cdot \frac{TOGM \cdot g}{D} \cdot \ln \frac{TOGW}{c_{ns} \cdot TOGW + w_{str}} \\
&\text{over} && \boldsymbol{X} = \begin{bmatrix} \frac{t}{c} & AR \end{bmatrix} \\
&\text{s.t.} && 0.08 \le \frac{t}{c} \le 0.15 \\
& && 4 \le AR \le 15 \qquad (4.3)
\end{aligned}$$

where $R(\boldsymbol{X})$ is the simplified Breguet range equation, c_R is a constant designed to normalize the reference wing range to 9260 km (5000 nm), D is the wing drag force, c_{ns} is a constant representing nonstructural weight equal to 0.61, and w_{str} is the weight from the structural suboptimization multiplied by a structural overhead factor of 1.3. The drag force D is determined analytically as a function of $\boldsymbol{X}$

$$D_i = D_{ref} c_{di} AR_{ref} / AR \qquad (4.4)$$

$$D_w = D_{ref} c_{dw} \cdot \left({}^{t}/_{c}\right) / \left({}^{t}/_{c}\right)_{ref} \qquad (4.5)$$

$$D = D_i + D_w + (1 - c_{di} - c_{dw}) * D_{ref} \qquad (4.6)$$

where c_{di} and c_{dw} are coefficients representing the portion of drag corresponding to induced and wave drag, respectively.

The structural subproblem is given as:

min $W_{opt} = W(\boldsymbol{T})$

over $\boldsymbol{T}$

s.t. $\frac{\sigma_1}{\sigma_{cr}} - 1 < 0$ [1]

$\frac{\tau_{12}}{\tau_{cr}} - 1 < 0$ [1]

$\frac{\sigma_1}{\sigma_{cr}} + \left(\frac{\tau_{12}}{\tau_{cr}}\right)^2 - 1 < 0$ [1]

$t_{core} - t_{panel} + 2t_{skin}^{min} < 0$ [1]

$\sigma_{VM} - M_{TS} < 0$ [2]

$-\sigma_{VM} - M_{CS} < 0$ [2]

$U_z^{tip} - 2.0 < 0$

$0.0001\,m \leq t_{skin} \leq 0.0035\,m$

$0.0001\,m \leq t_{core} \leq 0.0100\,m$

$0.0050\,m \leq t_{spar} \leq 0.1000\,m$

$0.0005\,m \leq t_{rib} \leq 0.0250\,m$ (4.7)

[1] For each sandwich element

[2] For each aluminum element

where $W(\boldsymbol{T})$ is the total mass of the wing box, determined by multiplying each element area by the thickness and density of each material layer for that element, and summing over all elements. $\boldsymbol{T}$ is a vector of eighty elements, ten per wing section, with eight corresponding to the total and core thicknesses of the four cover panels, one corresponding to the spars in the section, and one corresponding to the outboard rib for that section. The order of these variables for a single section are show in Table 4.2. The overall vector $\boldsymbol{T}$ is simply the concatenation of the variables for all eight sections. t_{skin} for each panel is simply the difference $t_{panel} - t_{core}$. U_z^{tip} is the tip deflection, which is constrained to be less than two meters.

Table 4.2 Example of Thickness Variables for a Wing Section

Panel 1		Panel 2		Panel 3		Panel 4		Spars	Ribs
$t_{panel,1}$	$t_{core,1}$	$t_{panel,2}$	$t_{core,2}$	$t_{panel,3}$	$t_{core,3}$	$t_{panel,4}$	$t_{core,4}$	t_{spar}	t_{rib}

The spanwise normal stress σ_1, the panel shear stress τ_{12}, and the von Mises stresses σ_{VM} are determined directly by the structural analysis for each element (depending on whether it is a sandwich element or an aluminum element). The other variables for the buckling constraints are determined analytically, using the conservative assumption of simply supported edges. For simplification, the average edge loads are considered to be uniform loads, and buckling is assumed to be elastic. Specifically:

$$\sigma_{cr} = \frac{k_c \pi^2 E_1}{12(1-\nu^2)} \left(\frac{t_{eq}}{b}\right)^2 \tag{4.8}$$

$$\tau_{cr} = \frac{k_s \pi^2 E_1}{12(1-\nu^2)} \left(\frac{t_{eq}}{b}\right)^2 \tag{4.9}$$

$$t_{eq} = (6t_F(t_c + t_F)^2 + 2{t_F}^3)^{1/3} \tag{4.10}$$

Here, k_c and k_s are compressive and shear buckling coefficients, which are taken to be 4.0 and 6.0, respectively. Although the coefficients are dependent on panel aspect ratio, these are the minimum values over the range of aspect ratios considered in this work.[161] Thus, they represent a conservative simplification for the purposes of this work. In the compressive buckling equation, b is the length of the loaded panel edge, while in the shear buckling equation, it is the shortest edge. In this work, these edges are the same. The thickness t_{eq} is that of an equivalent solid plate with matching bending stiffness. The derivation of this equation is given in APPENDIX G.

The reference wing parameters are given in APPENDIX H.

CHAPTER V

COMPUTATIONAL FRAMEWORK

5.1 Implementation

The IROM method was implemented primarily in MATLAB. Each test problem's optimization was managed within MATLAB, which also acted to coordinate processes for any analysis and pre-processing codes. These codes included ANSYS ICEM-CFD, ANSYS Fluent, and MSC Nastran.

The primary MATLAB code was to be executed on a workstation with multiple cores available. Massively parallel computational clusters were also available and were utilized for selected sections of the code, as well as the CFD analysis. This parallelization is noted in the discussion of the results for each test problem.

The code was structured to allow the optimization problem and analyses to be easily altered. The master execution script (Master File) managed all parameters, initial DOE construction setup and pre-optimization error testing, and the primary optimization. Separate scripts for the objective and constraint functions were implemented, and were responsible for computing gradients when requested. For the TWO problem, the system objective script was responsible for managing the suboptimization, as well as interpolating any IROMs for the relevant design points and passing those to the suboptimization. This structure allowed for problem-specific objective and constraint formulation.

The IROM interpolation model was maintained as a global object, which could be updated when necessary. The interpolation script, which implements the primary IROM algorithm, was kept isolated from the problem specific structures. Diagrams for the general structure of the ASO and TWO test problems are shown in Figure 5.1 and Figure 5.2, respectively.

Critical data structures such as the testing points and results, the interpolation model, databases of FOM results, ROMs, and valid IROMs, and the optimization path, are stored as separate files and updated as needed. This organization provides restart capability in the event of unforeseen failures, as well as minimizing any necessary recomputation as a result of changes to the problem definitions or structures. Many critical parameters and settings are stored in the master file, to provide ease of modification.

Monitoring of the problems is accomplished via text log files, which are also printed to the command window. These log files are timestamped with both local CPU and wallclock information, and key sections of code print unique identifiers and timing information to allow later analysis of function calls and timing information from the log file. MATLAB's built-in diary functionality is also used to capture any messages, which appear only in the command window.

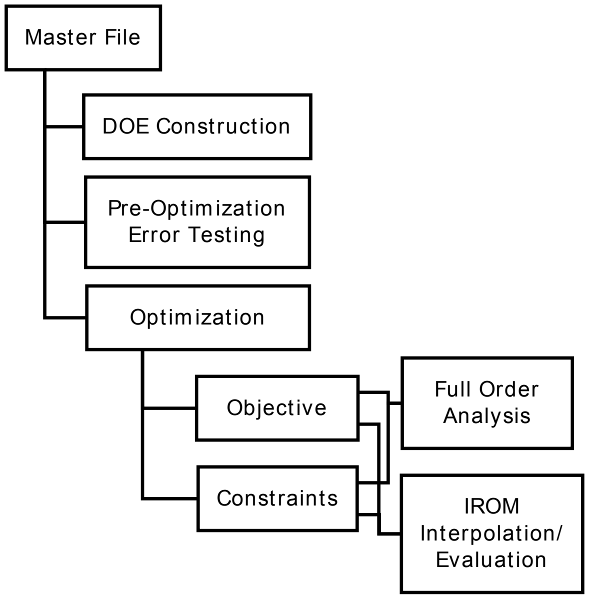

Figure 5.1 ASO Implementation Structure

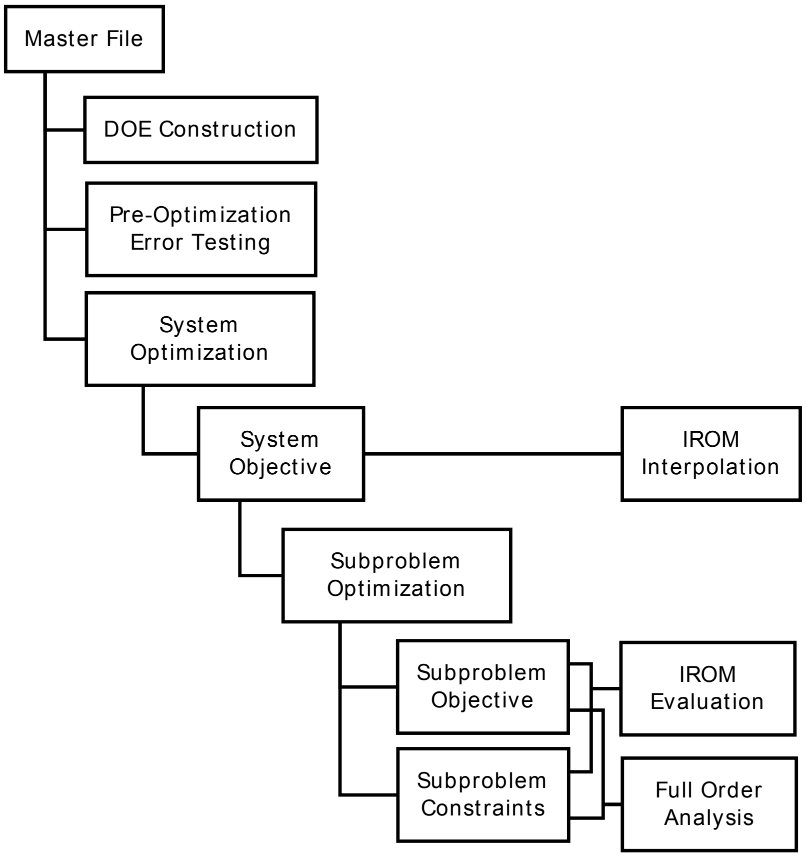

Figure 5.2 TWO Implementation Structure

Several forms of parallelization are utilized. The local workstation has multiple cores available. These are utilized via MATLAB's built in local worker pool capability and the *parfor* structure, which allows easy parallelization of a *for* loop. Some sections of code, such as the evaluation of snapshot ROMs during the interpolation process, are manually split into sections and then submitted to the clusters as individual jobs. Coordination of these workers is accomplished through input/output files, while the main code waits for completion of all jobs. Finally, analyses may be conducted in coarse- and fine-grained parallelism; for example, all angles of attack for the ASO problem are evaluated as separate cluster jobs while each job utilizes multiple processors from within Fluent.

Post-processing of the test problem results is also performed in MATLAB, either as part of the problem run through analysis of the log files, or as analysis of the output data structures.

Relevant specifics regarding the computational environments for each test problem are discussed along with their results in CHAPTER VI. Although the memory and storage requirements for these problems are nontrivial, they are well within workstation capabilities. Thus, the remainder of this chapter is concerned with general time estimates of the major problem steps. This analysis may be useful for estimating the time investment each step requires during application. Examples of parallelization are included here to illustrate some of the approaches used in this work; however, in general there are many different ways to parallelize each of the major processes. The term ROM is used here sometimes to refer to the RBF models, though it should be clear from the context whether reference is to RBF- or POD-based ROMs.

5.2 DOE Construction

This step involves generating a DOE design for the initial ROM database. In this work, Latin hypercube design points selected by maximin criteria are used. Designs for both the design variables, $\boldsymbol{X}$, and the ROM parameters, $\boldsymbol{P}$, are generated. The full order model is evaluated for each design point over each parameter input, and the results are used as snapshot sets to create ROMs. Thus, the resulting time for this step given N_X sample design points and N_P parameter samples is a function of the time for FOM evaluations and the time for constructing a POD-based ROM. These times are generally small enough to be measured directly. This step may also be parallelized by a factor p. Therefore, the total DOE computational time can be determined as

$$t_{DOE} = \left(N_x N_p (t_{FOM} + t_{POD})\right)/p \tag{5.1}$$

5.3 Pre-Optimization Testing

The testing prior to beginning optimization involves estimating the error associated with the chosen DOE designs, as well as constructing the initial interpolation model. The specific errors are truncation error, ROM error, global error, non-adaptive local error, and adaptive local error.

5.3.1 Truncation Error

Truncation error is a straightforward evaluation, requiring a negligible time investment, and in this work serves primarily as a sanity check. The time estimate for truncation error is calculated as

$$t_{trunc} = N_X N_s t_{evalPOD} \tag{5.2}$$

where N_X is the number of design points in the ROM database, N_s is the number of parameter snapshots, and $t_{evalPOD}$ is a representative time for evaluating a POD-based ROM.

5.3.2 ROM Error

ROM error is evaluated by selecting a number of random test parameters, for which the full order model is evaluated and compared against the ROM output for that design point. This provides an estimate of how well the ROM approximates the full order analysis at that particular design point. The time required to evaluate this error is also generally minor, and is found as

$$t_{ROMerror} = N_P N_{P,test} t_{FOM} + N_p N_{p,test} t_{evalPOD} \tag{5.3}$$

5.3.3 Interpolation Model

The interpolation model must be constructed in order to evaluate the global and local errors. There are three steps to that process which occupy the bulk of the interpolation time: (1) constructing and evaluating the basis interpolation ROM, (2) constructing and evaluating the ROMs for the bias and normalization vectors, and (3) constructing and evaluating the snapshot interpolation.

5.3.3.1 Basis Interpolation ROM

The basis interpolation ROM is a single RBF model interpolating each projected basis vector across the design space. That is, for a set of k-order bases, N_X vectors of length $n_{pdim} \cdot k$ are used as samples for a new RBF model, and that model is then evaluated at the target design point. Here, n_{pdim} represents the dimensionality of the snapshots. Since this work does not truncate the bases, the dimensionality becomes n_{pdim}^2.

In this work, the dominant cost of RBF construction is the leave-one-out cross-validation fitting procedure. An upper bound parameter, $maxcross$, is available for limiting the growth of this procedure. The cost of performing cross validation is a function of the time required to construct an RBF model for a given kernel function, ϕ, and shape parameter, c. This time varies with the dimensionality of the samples, but is generally small enough to be measured directly. The kernel functions and shape parameters are also searched over their valid ranges, with N_ϕ functions and N_c samples considered. The cross validation may also be locally parallelized. Thus, the total time to construct the basis interpolation RBF model is given by the time required to perform cross-validation plus the time to reconstruct the best fit as

$$t_{BasisRBF} = \left(N_\phi N_c \min\{maxcross, N_X\}\, t_{buildRBF}\right)/p + t_{buildRBF} \tag{5.4}$$

The total time for this step is then simply the time to construct and evaluate the basis interpolation RBF model given as

$$t_{ba} = t_{BasisRBF} + t_{evalRBF} \tag{5.5}$$

5.3.3.2 Bias and Normalization Vectors

The snapshots for the ROMs may be biased and normalized, resulting in a Y_{bias} vector with n_{pdim} elements, and a scalar Y_{norm}. The ROMs are interpolated elementwise with RBF models. Thus the time to create each ROM, t_{RBF}, and the time to evaluate them, $t_{evalRBF}$, are used to evaluate the bias and normalization time as

$$t_{bn} = n_{pdim}(t_{RBF} + t_{evalRBF})/p \tag{5.6}$$

5.3.3.3 Snapshot Interpolation

The interpolation of the snapshots themselves is also performed elementwise. This is generally the most expensive step, involving the construction and evaluation of $N_X \cdot N_P$ RBF models. The time required can be estimated as

$$t_{si} = N_X N_P(t_{RBF} + t_{evalRBF})/p \tag{5.7}$$

5.3.3.4 IROM Evaluation

The cost of constructing the interpolation model is simply the sum of the previous major components found as

$$t_{IROM} = t_{ba} + t_{bn} + t_{si} \tag{5.8}$$

To evaluate an interpolation model, which does not require an update, most of the RBF models are already constructed and stored. The exception is the basis interpolation models, which must be recomputed for each target design point because they are dependent on the choice of reference basis. There is a small additional fixed cost representing the other steps of the interpolation function, but this is generally negligible compared to the main steps. Thus, the relevant times for each section, and the total, can be estimated as

$$t_{ba} = t_{BasisRBF} + t_{evalRBF} \tag{5.9}$$

$$t_{evalbn} = n_{pdim}(t_{evalRBF})/p \tag{5.10}$$

$$t_{evalsi} = N_X N_P (t_{evalRBF})/p \tag{5.11}$$

$$t_{evalIROM} = t_{ba} + t_{evalbn} + t_{evalsi} \tag{5.12}$$

5.3.4 Global Error

The global error is a measure of the interpolation model's general goodness of fit for matching the full order analysis throughout the design space. Thus, a set of N_G test points must be evaluated with the full order model, IROMs interpolated to those target points, and then evaluated. The results are compared at each of the common snapshot parameters to avoid influence from ROM error. This leads to the time estimate as

$$t_G = N_G(N_P t_{FOM} + t_{evalIROM} + N_P t_{evalPOD}) \tag{5.13}$$

5.3.5 Non-Adaptive Local Error

The local error is a measure of the error of the gradient calculation with the full order model and the IROMs for the same stencil. This is performed with the objective function for the problem of interest; thus the time estimate for evaluating the objective with the full-order and reduced-order models is obviously problem dependent. In general, for N_{NL} test points, each of which is of dimensionality n_{xdim}, each test point must be evaluated with the objective function utilizing the full-order model, another evaluation for each stencil point, and IROM interpolations and evaluations for each point as well. In this work, the stencil evaluation may be parallelized. Note that depending on the finite difference method used, there may be one or two stencil points per dimension, represented here as c_{FD}. Thus the time is generally estimated as

$$t_{NL} = N_{NL}\left(t_{objFOM} + c_{FD} n_{xdim}\left(t_{objFOM} + t_{evalIROM} + t_{objIROM}\right)/p\right) \tag{5.14}$$

5.3.6 Adaptive Local Error

The adaptive local error is a measure of the error of the gradient calculation as well, but it also takes into account the effect of building a ROM for the target design point and updating the interpolation model accordingly. To avoid influence from other test points, the reference interpolation model is stored and updated independently for each test point; thus, the interpolation model used for each test point consists of the reference model plus the ROM for that point only.

The cost is similar to the non-adaptive local error, with the added cost of constructing ROMs and rebuilding the interpolation model for each test point found as

$$t_{NL} = N_{NL}\begin{pmatrix} t_{objFOM} + t_{POD} + t_{IROM} + \\ c_{FD} n_{xdim}\left(t_{objFOM} + t_{evalIROM} + t_{objIROM}\right)/p \end{pmatrix} \tag{5.15}$$

5.4 Objective, Constraints and Gradient Evaluation

5.4.1 ASO Problem

Evaluating the objective and constraints for the ASO problem involves performing n_α CFD simulations for the full-order case. If a gradient is requested, $n_{xdim} \cdot n_\alpha$ additional simulations must be performed in the full order case, or n_{xdim} IROM interpolations followed by $n_{xdim} \cdot n_\alpha$ POD evaluations of the IROMs. Note that generally, once a CFD analysis, ROM construction, or IROM interpolation has been evaluated, it is stored and does not need to be performed again.

In this work, the CFD simulations were parallelized in two levels, a coarse-grained parallelism of $p = n_\alpha$, and a fine-grained parallelism that was used to speed up each respective simulation, reducing t_{FOM}. Generally, the cost of evaluating the objectives and constraints was dominated by the cost of performing the full or reduced order analysis.

For the ASO problem, the adaptive step was enabled. Thus, the time to evaluate the objective and constraint functions at the target point is mainly just the cost of performing the FOA. The cost of evaluating the gradients of both is then a function of the number of stencil points, the IROM interpolations required, and the times to evaluate either the FOM or the IROM at each stencil point. That is

$$t_{ASO,FOM} = (n_{xdim} + 1)(n_\alpha t_{FOM}/p) \tag{5.16}$$

$$t_{ASO,IROM} = n_\alpha t_{FOM}/p + n_{xdim}(t_{evalIROM} + n_\alpha t_{evalPOD}) + t_{IROM} \tag{5.17}$$

5.4.2 TWO Problem

In the case of the TWO problem, evaluating the system objective depends on performing the structural suboptimization. Although a general estimate of this time can be developed through testing, it is not a fixed quantity and varies throughout the overall optimization process. The suboptimization is also the dominant cost of evaluating the objective and its gradients.

For this problem, the objective function was responsible for interpolating the IROM models to replace the finite element analysis at each stencil design point. These models were then passed to the suboptimization script. Also note that the adaptive step was not enabled for this problem. Thus, given representative suboptimization times for the full-order and IROM-based analyses, the system level objective and gradient cost is essentially found as

$$t_{SYS,FOM} = (n_{xdim} + 1)t_{SUB,FOM} \tag{5.18}$$

$$t_{SYS,IROM} = t_{SUB,FOM} + n_{xdim}(t_{evalIROM} + t_{SUB,IROM}) \tag{5.19}$$

Since the suboptimization objective is a simple analytical expression, and the suboptimization constraint function is evaluated simply with the FOM or IROM as requested, the expected relation between $t_{SUB,FOM}$ and $t_{SUB,IROM}$ is given as

$$t_{SUB,IROM} = \left(\frac{t_{evalPOD}}{t_{FOM}}\right) t_{SUB,FOM} \tag{5.20}$$

CHAPTER VI

RESULTS AND DISCUSSION

6.1 ASO Problem

The ASO problem was implemented in MATLAB, utilizing ANSYS ICEM-CFD for grid generation and ANSYS Fluent for CFD analysis. Both the FOM and IROM optimizations were performed via SQP under identical computational environments, operating on a four-core workstation to manage the optimization and interpolation model; two of these cores were used. A high performance massively parallel system, Raptor, was for the CFD simulations. Each angle of attack was submitted as a separate job to the cluster, each of which utilized twelve cores for fine-grained parallelism managed within Fluent. Each optimization started with 0.5 for each weight.

Latin hypercube experimental designs were generated for the design space (airfoil weights). Since the ROMs did not need to be evaluated at locations other than the original snapshot angles of attack, no DOE was needed for the parameter space. The DOE for the design space was selected based on maximin criteria. A 50-point design and a 100-point design were compared prior to optimization to gauge the relative benefit of increasing the DOE design size.

The initial solution and ROM databases were constructed according to the DOE designs, an initial interpolation model for the IROMs built, and pre-optimization error testing was performed. The 50-point DOE design was selected based on the results. The FOM and IROM optimizations were then performed.

6.1.1 Interpolation Error

A set of ten random test points was used to evaluate the 50-point DOE design and interpolation errors. The points along with their finite difference stencils were evaluated using the FOM. A sanity check of the truncation error for each ROM yielded a maximum NRMSD of 1.4E-12. Since the ROMs were not utilized at parameters other than the snapshots, no ROM error was required. A stepsize of 0.0005 was selected through a stepsize study. IROMs were also interpolated to each test point and its associated stencil for use in the non-adaptive local error estimate.

For the adaptive local error, the reference interpolation model was stored separately. A ROM was then constructed for each test point and used to update the interpolation model before interpolating the IROMs for the stencil points. Each update operated strictly with the original interpolation model plus the test point, to avoid the influence of other test points. The 100-point design was also used to evaluate adaptive local error in this fashion, to gauge the benefit of a larger DOE design.

6.1.1.1 Global Error

The global interpolation error is summarized in Table 6.1. The maximum NRMSD was 0.00827, indicating a good overall fit throughout the design space. The NMAX metric indicates the presence of outliers.

Table 6.1 ASO Global Error

ID	1	2	3	4	5	6	7	8	9	10
NRMSD	0.0041	0.0072	0.0003	0.0046	0.0036	0.0024	0.0041	0.0082	0.0082	0.0006
NMAX	17.35	7.95	217.50	15.34	18.64	28.02	15.25	8.54	9.84	11.01

6.1.1.2 Local Error

The non-adaptive and adaptive local errors are summarized in Table 6.2, expressed as percentage spectral angle error.

Table 6.2 ASO Local Error

ID	1	2	3	4	5	6	7	8	9	10
SPERR (non)	60.5%	61.9%	54.5%	55.0%	41.4%	66.2%	51.8%	52.1%	37.7%	47.7%
SPERR (adapt)	9.2%	11.5%	9.2%	12.4%	4.3%	9.6%	7.9%	19.3%	14.0%	10.6%
SPERR (100pt)	4.2%	7.3%	10.7%	10.6%	7.3%	7.3%	10.6%	13.5%	15.4%	13.6%

In general, the non-adaptive errors were quite large, ranging from 68 - 119 degrees, or 37.7 - 66.2%. This reflects the sensitivity of the objective function to errors in the aerodynamic coefficients, which were on the order of 1e-3.

The adaptive local error was also evaluated, for both the 50-point and 100-point DOE designs. In the 50-point case, the local error ranged from 7 - 34 degrees, or 4.3 - 19.3%. While this error is relatively high, it is also greatly improved over the non-adaptive case.

The mean adaptive local error for the 50-point design was 10.8%, while increasing the size to 100-points only improved the mean error to 10.0%, while incurring substantial additional computation time. Based on this, the 50-point error was deemed sufficient. It was also of interest to examine how the optimization would perform with a small error in the gradient.

6.1.2 Optimization Paths

The optimization paths and objective histories between the FOM and IROM optimizations are compared in Figure 6.1 and Figure 6.2. The FOM run completed in 7 iterations, while the IROM run completed in 6. At the end of the last iteration, the objective values were -41.49 for the FOM optimization, and -41.42 for the IROM optimization. These represent 24.1% and 23.9% improvement over the original value of -33.43.

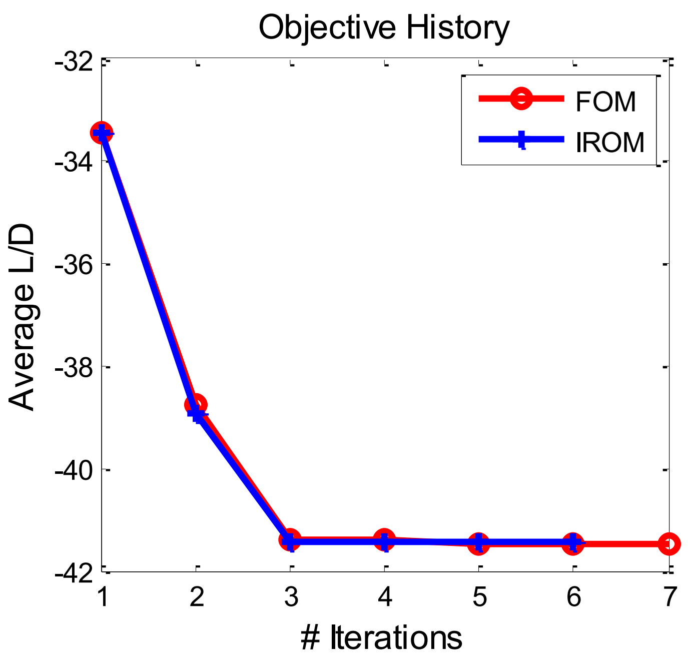

Figure 6.1 ASO Optimization Objective History

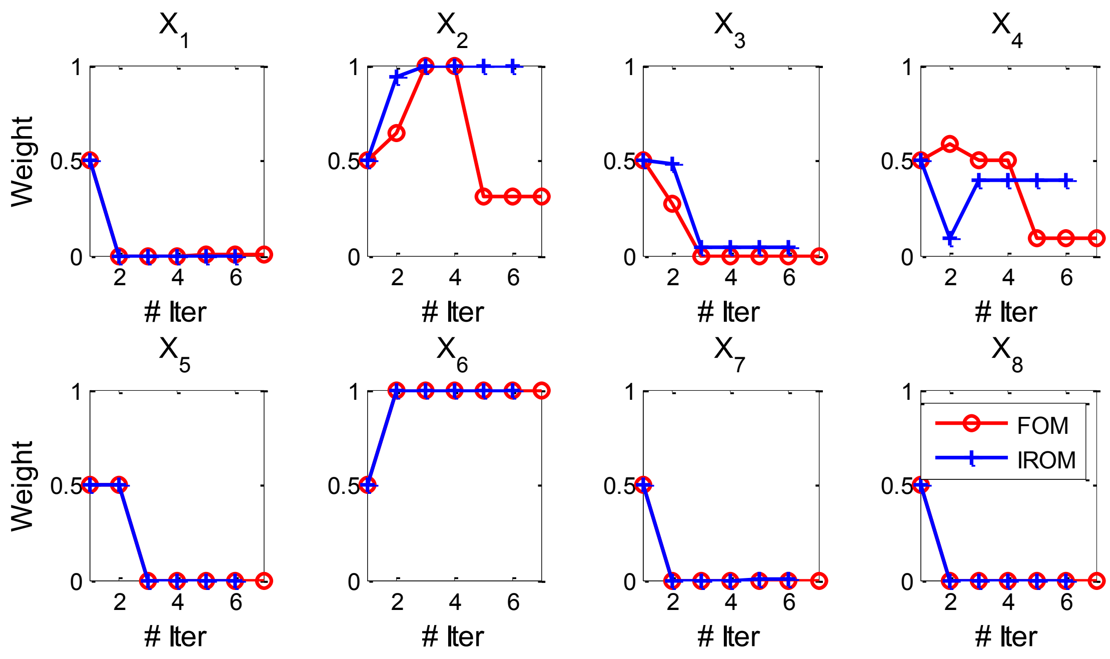

Figure 6.2 ASO Optimization Path

6.1.3 Function Calls

One of the ways to evaluate the computational cost of solution procedures for different problem (i.e., analysis functions) is to compare the number of function calls. In this case, the primary functions of interest are the number of objective and constraint evaluations, the number of FOM evaluations, and the number of IROM interpolations. The objective and constraint functions may be requested with or without gradients, and the interpolation function may or may not require an update for the adaptive step; the number of calls requiring these specific steps is also noted. Since the number of optimization iterations is difficult to predict a priori, the total counts for each optimization, as well as the average counts per optimization iteration, are reported in Table 6.3.

Table 6.3 ASO Function Calls

Function	Total (FOM)	Total (IROM)	Avg/It (FOM)	Avg/It (IROM)
Objective	154	183	22.0 (154/7)	30.5 (183/6)
(w/ Gradient)	80	94	11.4 (80/7)	15.6 (94/6)
Constraints	80	94	11.4 (80/7)	15.6 (94/6)
(w/ Gradients)	80	94	11.4 (80/7)	15.6 (94/6)
Full Order Analysis	675	90	96.4 (675/7)	15.0 (90/6)
IROM Interpolation	0	720	0.0	120.0 (720/6)
(w/Model Update)	0	90	0.0	15.0 (90/6)

6.1.4 CPU and Wallclock Times

A more explicit metric for computational cost is a direct measure of both CPU and wallclock time required by the various optimization components. For these calculations, parallelism must be considered. For these problems, the FOM evaluated each parameter value in an independent simulation, each of which utilized twelve cores on a computing cluster. Stencil points were also evaluated in parallel for the IROM optimization, but in serial for the FOM optimization due to license limitations for the fluid solver. When updating the interpolation model, the elements of Y_{bias} and the outputs are modeled independently; these models were also constructed in parallel. Finally, RBF cross-validation was performed in parallel with two cores whenever the model construction was not within another level of parallelism. Some times are estimated due to inherent variability, such as simulation time, or unpredictable delays, such as cluster queue wait time. These times are summarized in Table 6.4. Note that these are representative times based on the optimization logs; some minor variability is expected, and some discrepancy due to the timing of log printouts may exist. CPU time for components utilizing computational clusters are estimated based on the wallclock time and parallelism, since the recorded CPU time is not reflective of the cluster's workload.

To interpret these times, consider total CPU time to be reflective of the cost of the operation; wallclock time is then illustrative of real world cost which takes advantage of parallelism, to the indicated extent. Also note that the FOM optimization was running for approximately 107.9 hours, while the IROM optimization ran for approximately 71.4 hours. This results in average wallclock time per iteration of 15.4 hours and 11.9 hours for the FOM and IROM optimizations, respectively.

Table 6.4 ASO CPU and Wallclock Times

Component	Total CPU Time (s)	Parallelism (Coarse / Fine)	Wallclock Time (s)
Full Order Analysis	~8,640	4 / 12	~180
RBF Construction (3 kernels, 50 pts)	6	2	~3
IROM Interpolation (total, w/ update)	~5876	3 (Y_{bias}), 12 ($\tilde{Y}$)	649
Y_{bias} Models	~339	3	113
$\tilde{Y}$ Models	~4,380	12	365
Stencil Evaluation Only (FOM)	~69,120	1 / 12	~1140
Stencil Evaluation Only (IROM)	~1,176	8	147

6.1.5 Discussion

From the optimization runs and the plots in Figure 6.1 and Figure 6.2, it is possible to make a few observations. First, it is clear that the optimization path taken by the IROM optimization differs from that taken by the FOM case. This is likely due to the ~10% mean error in the interpolated gradient calculations. The impact of this error, and of the alternative path, is expected to be highly problem dependent. For this case, it appears from the objective history that the impact is minimal, resulting in approximately the same improvement and convergence rate. Note that since the objective values are the results of FOAs, the error lies mainly in the search direction.

The purpose of dealing with the errors, of course, is the computational advantage. Even for the relatively inexpensive CFD analysis used in this problem, the IROM optimization achieved a comparable degree of progress in about 67% of the total wallclock time of the FOM optimization. It is also clear from the function counts that far fewer FOAs are required for the IROM case, approximately 13.3% (20.7% if including the offline DOE calculations). This is the case even though the average evaluations per iteration suggest that the line searches were somewhat longer in the IROM optimization. With an expensive FOA, this suggests the potential for significant speedup.

The CPU times provide further support for this potential, illustrating that the IROM update and interpolation costs are moderate with respect to the FOA. However, it should also be noted that the costs of constructing IROMs and updating the interpolation model are dependent on the dimensionality of the parameter and snapshot spaces, and thus the real world cost would likely be a strong function of the ability to parallelize the process. For an ideal speedup, the choice of reduced system that the IROM will interpolate should be made carefully.

There are a few caveats to note. Like most surrogate model schemes, this scheme requires an initial offline computation of an experimental design. Associated with this are DOE size studies, stepsize studies, and initial error estimates to evaluate candidate designs. In some cases this setup could become expensive, and should be considered when determining the speedup advantages versus a more standard optimization approach.

The sensitivity of the objective and constraint gradient error to the analysis outputs is also problem-dependent, and may determine a minimum level of accuracy.

6.2 TWO Problem

The TWO problem was also implemented in MATLAB, by modifying the analysis and problem portions of the ASO code. MSC/NASTRAN was used as the analysis code that the ROMs replaced. A 50-point Latin Hypercube design was generated for both the design and parameter spaces, selected via maximin criteria. Due to technical challenges, automatic cluster submission was not available for this problem. Thus, all test optimizations were performed on local workstations, with equal numbers of processors allocated to each job.

This problem is structured as a two-level multidisciplinary problem, involving interaction between a system level problem, with design variables of airfoil thickness and wing aspect ratio, and a sublevel problem, with design variables of wall thicknesses for the wing structural elements. The system level problem is responsible for the aerodynamic analysis and range calculations, while the sublevel problem is responsible for determining the optimal wing weight according to material limits, buckling constraints, and tip deflection limits, which must be satisfied under two loading conditions equivalent to -1.5g and +3.5g.

The primary coupling between the problem levels consists of the system level passing the wing geometric parameters $\left(\frac{t}{c}\right)$ and AR to the sublevel, which determines and returns the optimal wing weight, W_{opt}. The code is structured to allow iteration between the aerodynamic and structural solvers to determine the convergent root angle of attack;

however, in this work, this iteration is omitted due to time considerations. Thus, the problem analysis is that of a rigid wing. The reference implementation also makes this change, noting that the actual wing could be built to a jig shape to offset deformation due to aerodynamics.

The main emphasis for the TWO problem is on examining the IROM scheme's capability to operate in a multidisciplinary context. In that environment, the IROM would replace one of the disciplinary analyses involved in a larger multidisciplinary analysis. In this case, the IROM must interact with other analyses as a more traditional surrogate, albeit one constructed without relying on additional full-order data. In the TWO problem, this is represented by the suboptimization being performed entirely with either with the FOM or the IROM.

The TWO problem is dominated by the dynamics of the structural suboptimization, in terms of accuracy as well as computational cost. Additionally, the suboptimization problem is a highly challenging problem for gradient-based optimization, with numerous local minima. The original reference work uses particle swarm optimization for the subproblem, noting difficulties for gradient-based methods.

The initial test run for the TWO problem used the reference wing as a starting point. Subsequently, and based on the results of each previous test, additional test points were selected and examined. The optimization tolerances, which impact stopping conditions based on optimality criteria, changes in the design variables, and violations of constraints, were adjusted with each successive test point based on the optimization behavior observed. The data quantity and quality were also improved with each revision in order to deepen the investigation into the behavior of the IROM scheme and the optimizations.

6.2.1 Interpolation Error

A set of ten random test points was used to evaluate DOE design and interpolation errors. The points along with their finite difference stencils were evaluated using the FOM. A sanity check of the truncation error for each ROM yielded a maximum NRMSD of 5.0E-16 for the first load case, and 6.0E-16 for the second.

The error for each ROM was estimated through a set of 10 random test points in the parameter (thickness) space. The mean ROM NRMSD error was 0.09227 for both load cases, and the median NRMSD error was 0.0892. Differences between the load cases began in the eighth or ninth decimal place.

A stepsize of 0.001 was selected through a preliminary stepsize study. IROMs were also interpolated to each test point and its associated stencil for use in the non-adaptive local error estimate.

6.2.1.1 Global Error

The global error estimate is summarized in Table 6.5 for the first load case (-1.5g) and Table 6.6 for the second load case (+3.5g). The values are plotted in Figure 6.3.

Table 6.5 TWO Global Error for Load Case 1

ID	1	2	3	4	5	6	7	8	9	10
NRMSD	0.1650	0.0038	0.2722	0.4372	0.2635	0.0932	0.1807	1.2155	0.0184	0.4376
NMAX	1.2781	1.2982	1.2815	1.2715	1.2831	1.2689	1.2759	1.2711	1.2938	1.2772

Table 6.6 TWO Global Error for Load Case 2

ID	1	2	3	4	5	6	7	8	9	10
NRMSD	0.1650	0.0038	0.2722	0.4372	0.2635	0.0932	0.1807	1.2155	0.0184	0.4376
NMAX	1.2781	1.2983	1.2815	1.2715	1.2831	1.2689	1.2759	1.2711	1.2838	1.2772

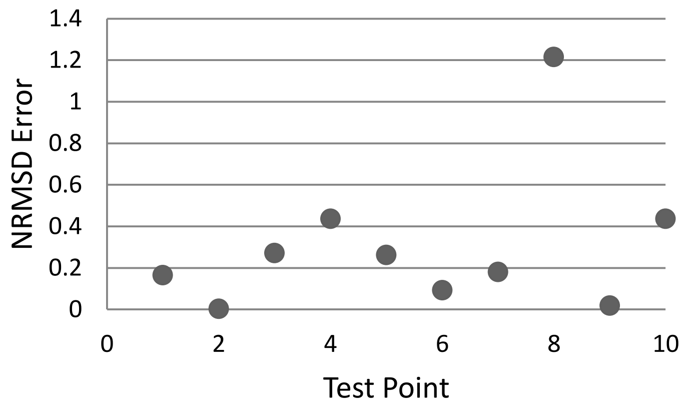

Figure 6.3 TWO Global Error

There was very little variation between the load cases, with differences occurring in the fifth or sixth decimal place. For both load cases, the mean global error for was 0.3087, and the median error was 0.2221. There was also a significant outlier of 1.2155.

6.2.1.2 Local Error

The non-adaptive local error is summarized in Table 6.7. The mean spectral error was 50.81%, while the median was 43.42% error. Due to limited computational resources, the adaptive step was not tested for this problem.

Table 6.7 TWO Local Error

ID	1	2	3	4	5	6	7	8	9	10
SPERR (%)	73.42	34.51	19.08	93.12	19.63	43.09	38.98	70.30	72.30	43.76

6.2.2 Test Point 1

The first test optimization began from the reference airfoil, with a airfoil thickness of 12.00% and an aspect ratio of 6.8751.

6.2.2.1 Optimization Paths

The optimization for the FOM case completed in 2 iterations and the IROM case completed in 4 iterations. The paths (up to 2 iterations) were nearly identical to within reporting accuracy. The path is represented in Figure 6.4 and Figure 6.5.

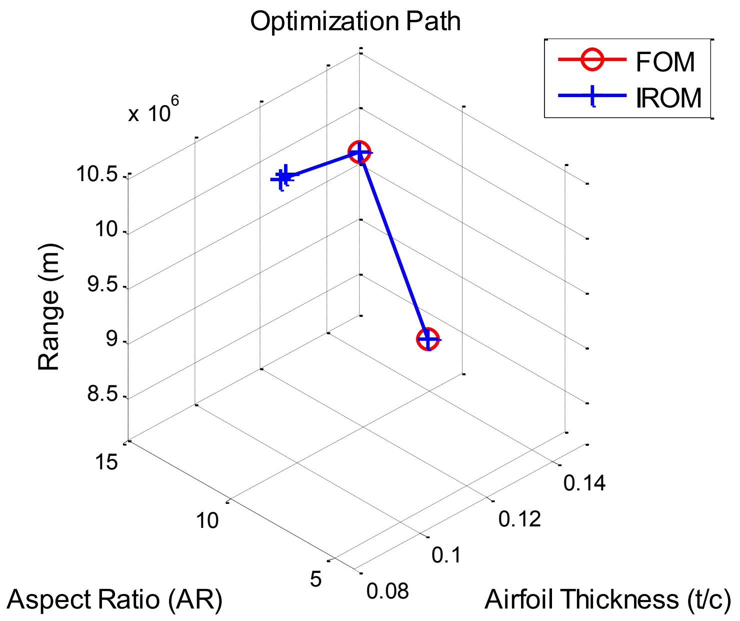

Figure 6.4 TWO Optimization Path (3D) , Test Point 1

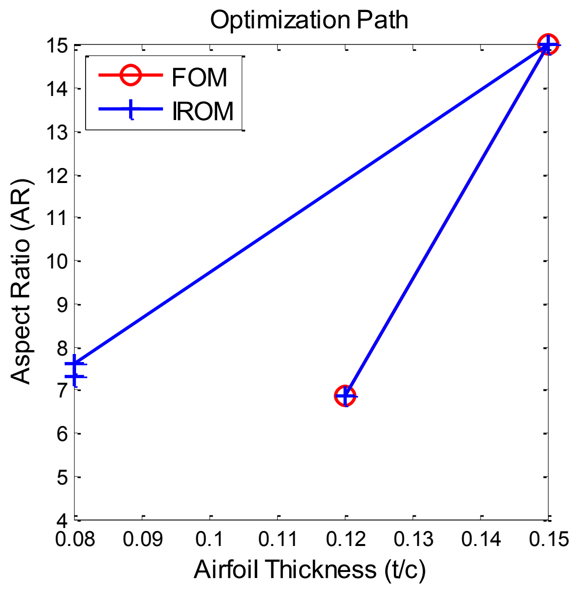

Figure 6.5 TWO Optimization Path (2D) , Test Point 1

The design variable histories, and the objective history, in terms of range, are shown in Figure 6.6. The FOM optimum was found at 15% airfoil thickness and an aspect ratio of 15.000. The final range for the FOM optimization was 9,596 km (5,181 nm), a 3.63% improvement over the original value of 9,260 km (5,000 nm). The IROM optimum was found at 8% airfoil thickness and an aspect ratio of 7.3228. The IROM optimum range was 11,358 km (6,133 nm), an improvement of 22.66%. An illustration of the wing planforms for the respective optimums is given in Figure 6.7

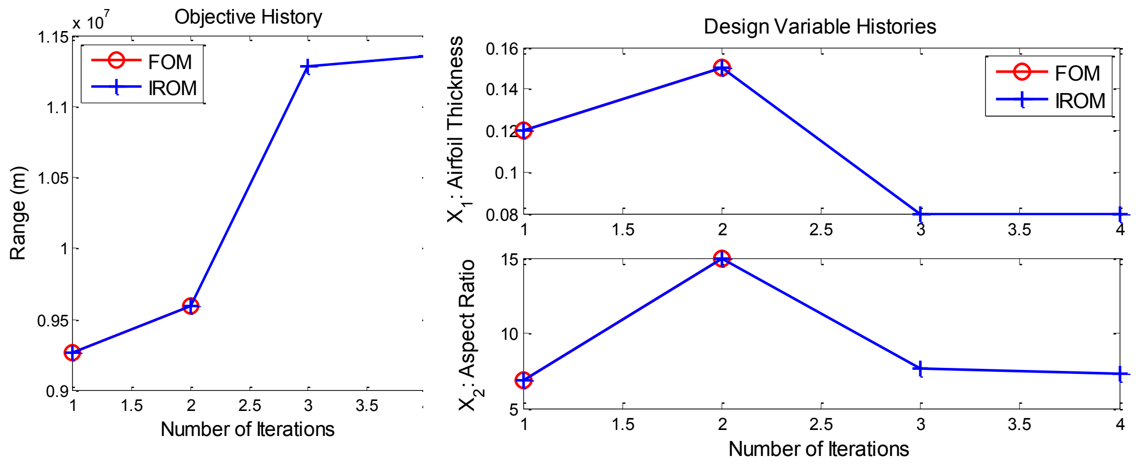

Figure 6.6 TWO Objective and Design Variable History, Test Point 1

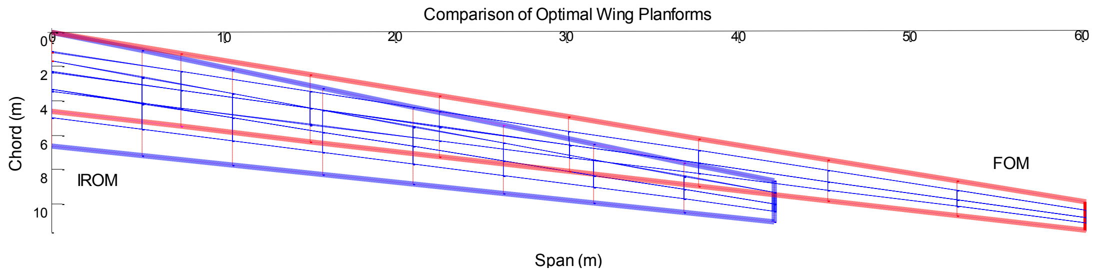

Figure 6.7 TWO Optimum Wing Planforms, Test Point 1

6.2.2.2 Function Calls

The total number of function calls for this optimization are given in Table 6.8.

Table 6.8 TWO Function Calls, Test Point 1

Function	Total (FOM)	Total (IROM)	Avg/Obj (FOM)	Avg/Obj (IROM)
Objective	4	14	1.0	1.0
(w/ Gradient)	2	4	0.5	0.3
Suboptimization	8	22	2.0	1.6
(w/ FOM)	8	14	2.0	1.0
(w/ IROM)	0	8	0.0	0.6
Subopt Objective	36,054	120,754	9,013.5	8,625.3
Subopt Constraints	970	3,417	242.5	244.1
(w/ FOM)	970	2,184	242.5	156.0
(w/ IROM)	0	1,233	0.0	88.1
(w/ Gradient)	970	3,417	242.5	244.1
Full Order Analysis	36,498	77,166	9,124.5	5,511.9
IROM Interpolation	0	16	0.0	1.1

6.2.3 Test Point 2

The second test point was randomly selected with an airfoil thickness of 12.43% and an aspect ratio of 5.0729.

6.2.3.1 Optimization Paths

The optimization for the FOM case completed in 3 iterations and the IROM case completed in 4 iterations. The paths (up to 2 iterations) were nearly identical to within reporting accuracy. The path is represented in Figure 6.8 and Figure 6.9.

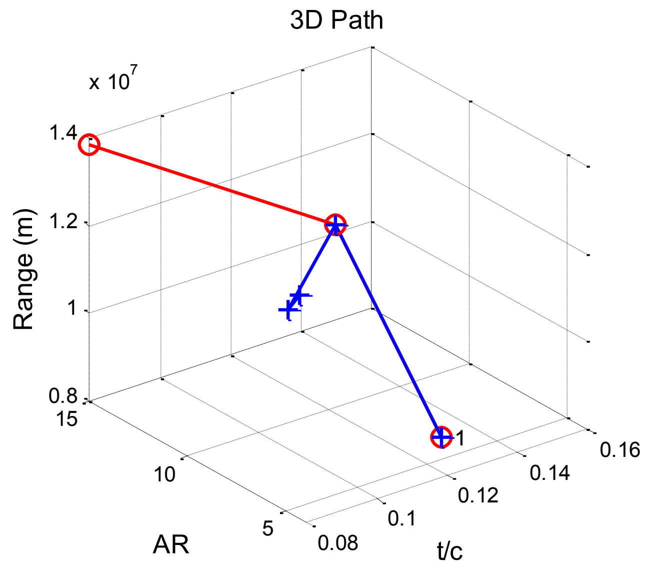

Figure 6.8 TWO Optimization Path (3D) , Test Point 2

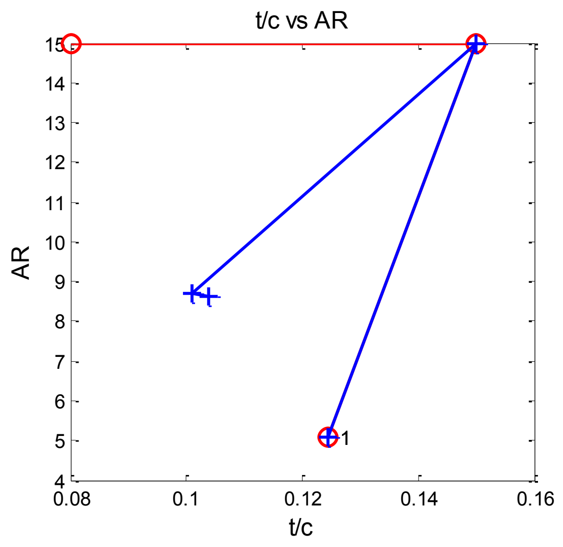

Figure 6.9 TWO Optimization Path (2D) , Test Point 2

The design variable histories and the objective history, in terms of range, are shown in Figure 6.10. The FOM optimum was found at 8.00% airfoil thickness and an aspect ratio of 15.0. The final range for the FOM optimization was 13,419 km (7,245 nm), a 44.90% improvement over the reference value of 9,260 km (5,000 nm). The

IROM optimum was found at 10.38% airfoil thickness and an aspect ratio of 8.6260. The IROM optimum range was 11,042 km (5,962 nm), an improvement of 19.24% over the reference value.

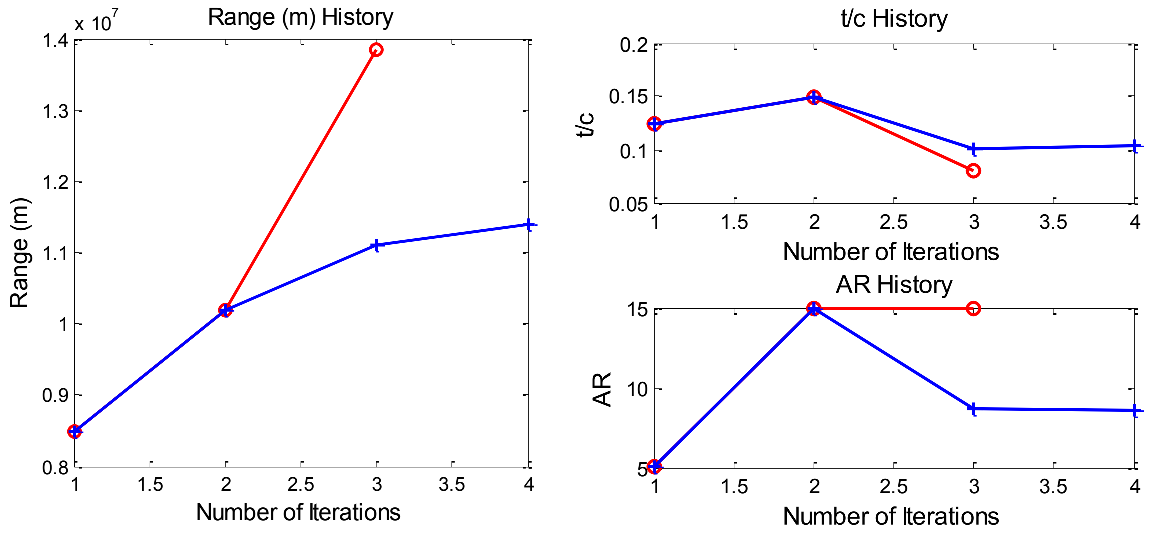

Figure 6.10 TWO Design Variable and Objective Histories, Test Point 2

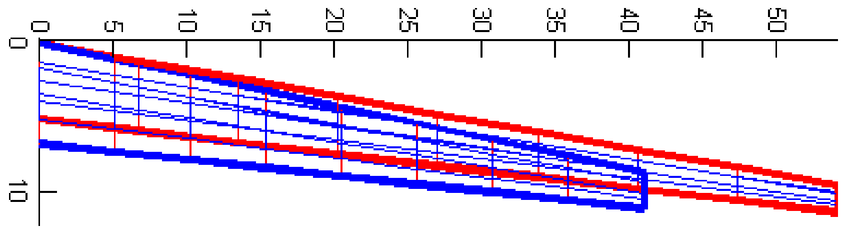

Figure 6.11 TWO Optimum Wing Planforms, Test Point 2

6.2.3.2 Function Calls

Table 6.9 TWO Function Calls, Test Point 2

Function	Total (FOM)	Total (IROM)	Avg/Obj (FOM)	Avg/Obj (IROM)
Objective	5	34	1.0	1.0
(w/ Gradient)	3	4	0.6	0.1
Suboptimization	11	42	2.2	1.2
(w/ FOM)	11	34	2.2	1.0
(w/ IROM)	0	8	0.0	0.2
Subopt Objective	70,019	218,893	14,003.8	6,438.0
Subopt Constraints	2,042	6,262	408.4	184.1
(w/ FOM)	2,042	4,880	408.4	143.5
(w/ IROM)	0	1,382	0.0	40.6
(w/ Gradient)	2,042	6,262	408.4	184.1
Full Order Analysis	70,852	170,202	14,170.4	5,005.9
IROM Interpolation	0	16	0.0	0.4

6.2.4 Test Point 3

The third test optimization began from a randomly selected test point, with an airfoil thickness of 14.7% and an aspect ratio of 9.3391.

6.2.4.1 Optimization Paths

The optimization for the FOM case completed in 5 iterations and the IROM case completed in 3 iterations. The path is represented in Figure 6.12 and Figure 6.13.

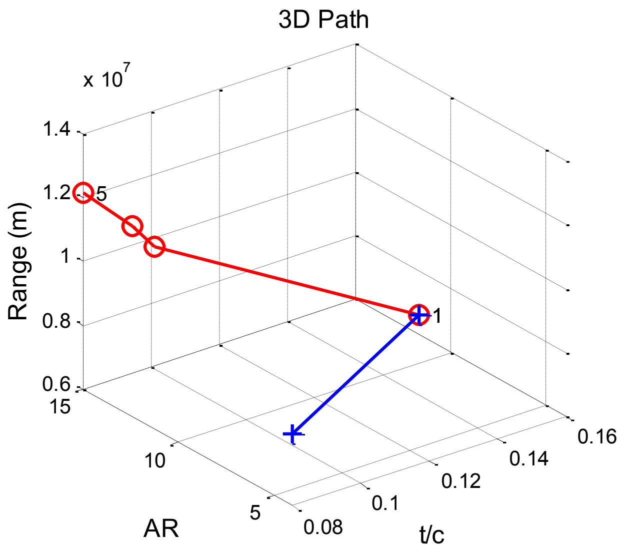

Figure 6.12 TWO Optimization Path (3D) , Test Point 3

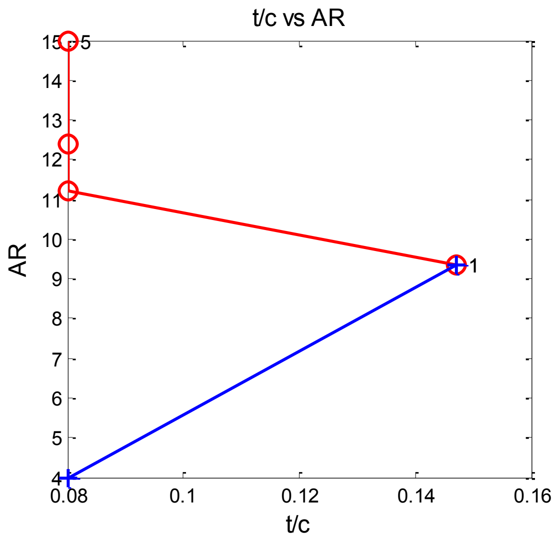

Figure 6.13 TWO Optimization Path (2D) , Test Point 3

The design variable and objective histories are shown in Figure 6.14.

The FOM optimum was found at 8% airfoil thickness and an aspect ratio of 15.0. The final range for the FOM optimization was 11,765 km (6,352 nm), a 27.05% improvement over the reference value of 9,260 km (5,000 nm).

The IROM optimum was found at 8% airfoil thickness and an aspect ratio of 4.0. The IROM optimum range was 8,004 km (4,321 nm), a decrease of 13.56% from the reference value (and an increase of 5.05% over the starting value of 7,619 km, or 4,113 nm).

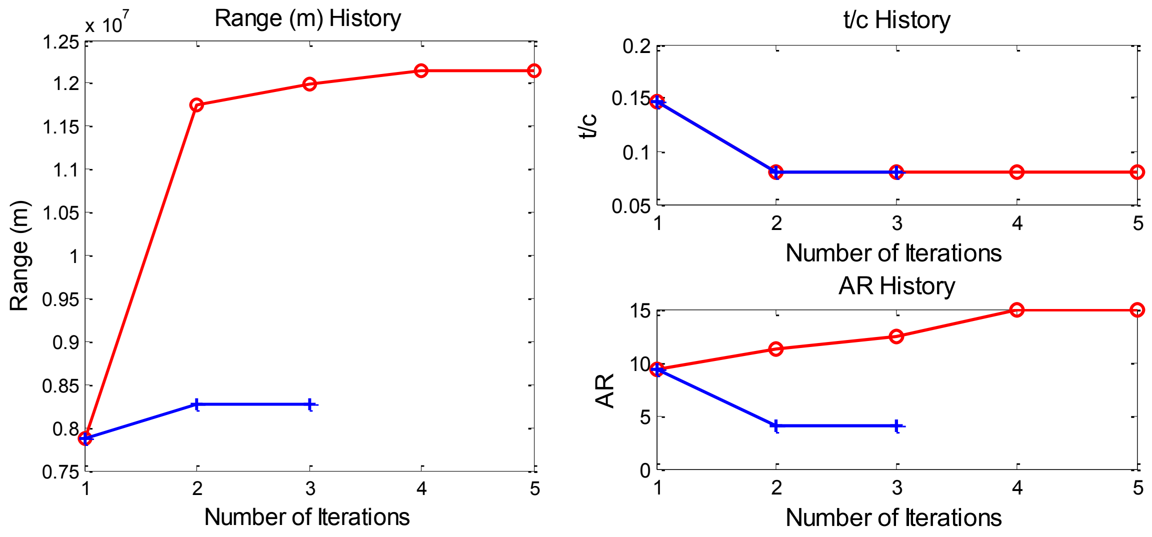

Figure 6.14 TWO Design Variable and Objective Histories, Test Point 3

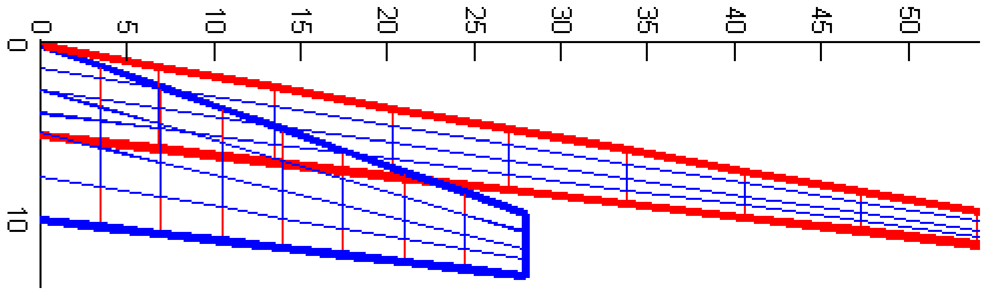

Figure 6.15 TWO Optimum Wing Planforms, Test Point 3

6.2.4.2 Function Calls

The total number of function calls for this optimization are given in Table 6.10.

Table 6.10 TWO Function Calls, Test Point 3

Function	Total (FOM)	Total (IROM)	Avg/Obj (FOM)	Avg/Obj (IROM)
Objective	9	5	1.0	1.0
(w/ Gradient)	5	3	0.5	0.6
Suboptimization	19	11	2.1	2.2
(w/ FOM)	19	5	2.1	1.0
(w/ IROM)	0	6	0.0	1.2
Subopt Objective	10,692	27,191	1,188.0	5,438.2
Subopt Constraints	261	595	29.0	119.0
(w/ FOM)	261	139	29.0	27.8
(w/ IROM)	0	456	0.0	91.2
(w/ Gradient)	261	595	29.0	119.0
Full Order Analysis	10,824	5,190	1,202.6	1,038.0
IROM Interpolation	0	12	0.0	2.4

6.2.5 Revised Test Point 1

Information from the first three test points was used to revise the optimization tolerances. Additionally, a more detailed stepsize study was performed. The revisions were tested at the first test point (the reference wing).

6.2.5.1 Stepsize Study

A more detailed stepsize study was conducted at a design point of 8.89% airfoil thickness and an aspect ratio of 14.0471. The resulting partial derivative for the objective function with respect to the airfoil thickness can be seen in Figure 6.16, and the absolute relative error of the IROM approach in Figure 6.17. Similarly, the partial derivative for the objective function with respect to the aspect ratio can be seen in Figure 6.18, and a plot of the absolute relative error in Figure 6.19.

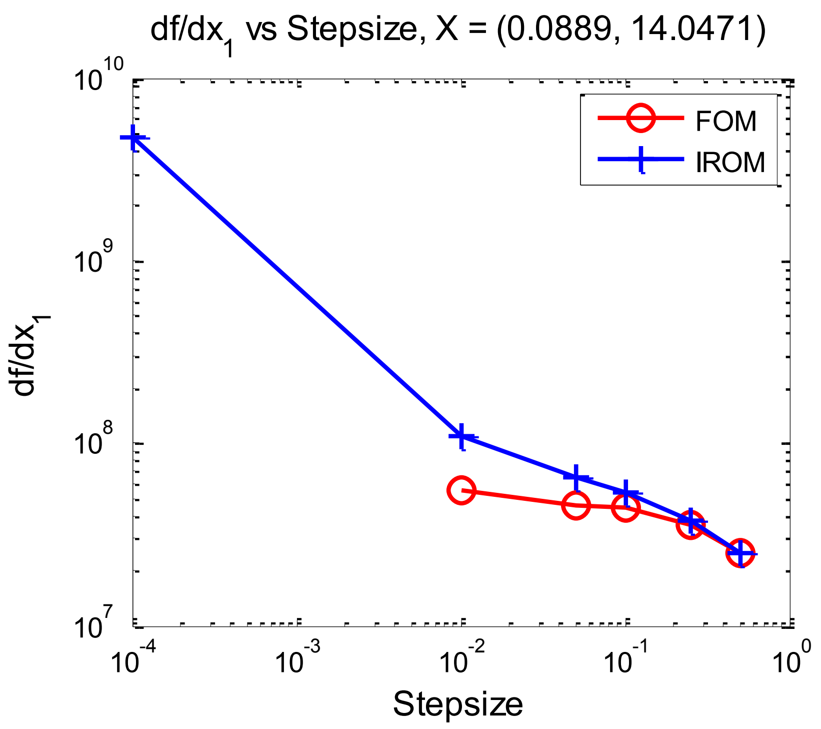

Figure 6.16 Airfoil Thickness Stepsize Study

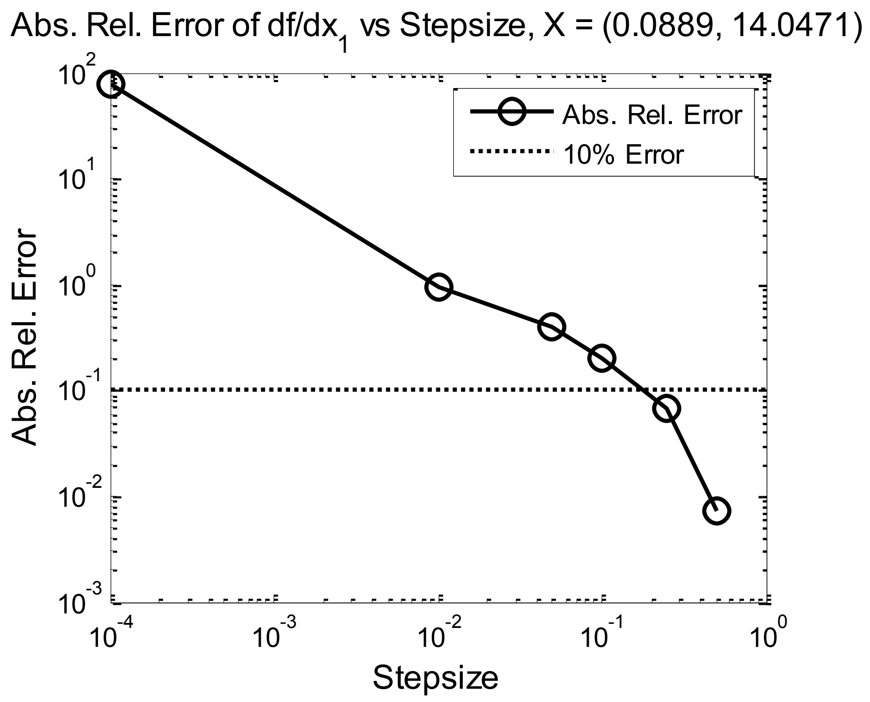

Figure 6.17 Airfoil Thickness Stepsize Study Error

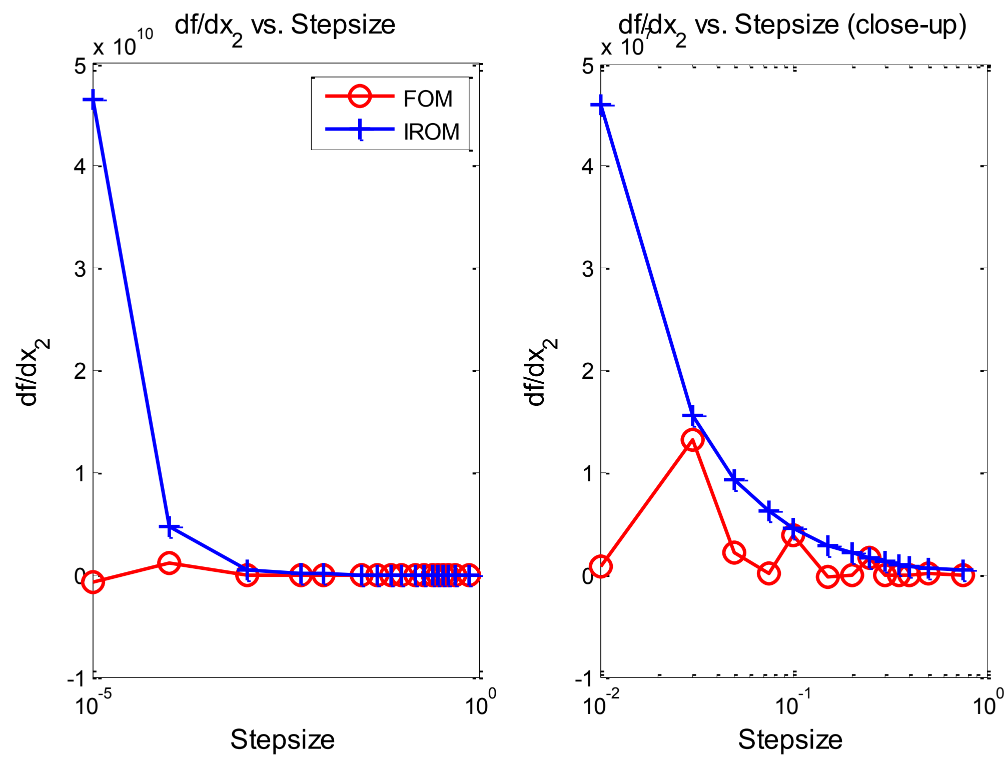

Figure 6.18 Aspect Ratio Stepsize Study

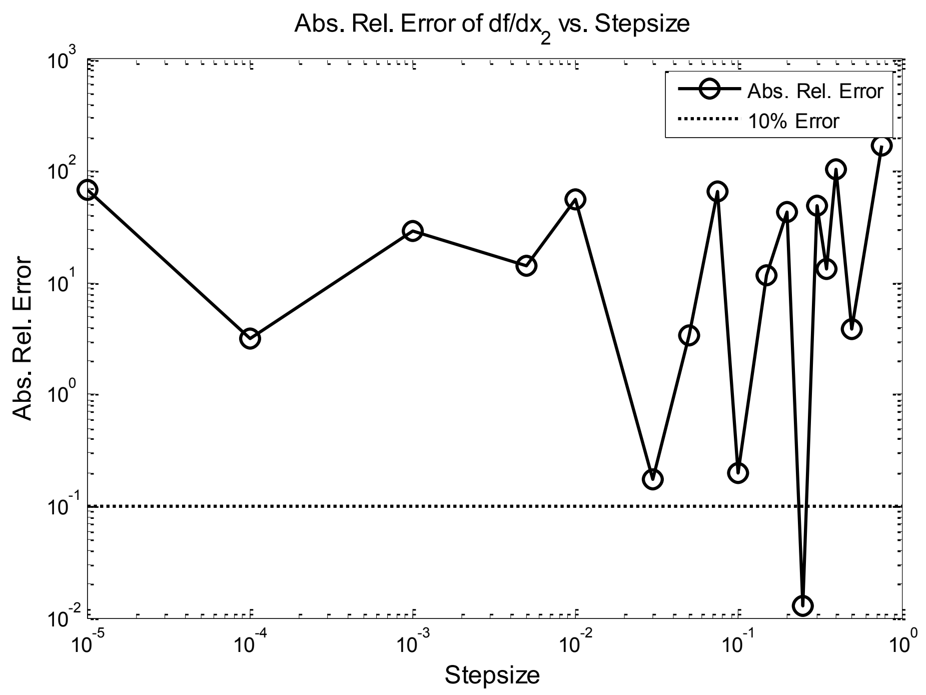

Figure 6.19 Aspect Ratio Stepsize Study Error

Four additional points throughout the design space were selected, and smaller stepsize studies conducted for the partial derivative with respect to aspect ratio. The partial derivatives can be seen in Figure 6.20 and the errors are shown in Figure 6.21.

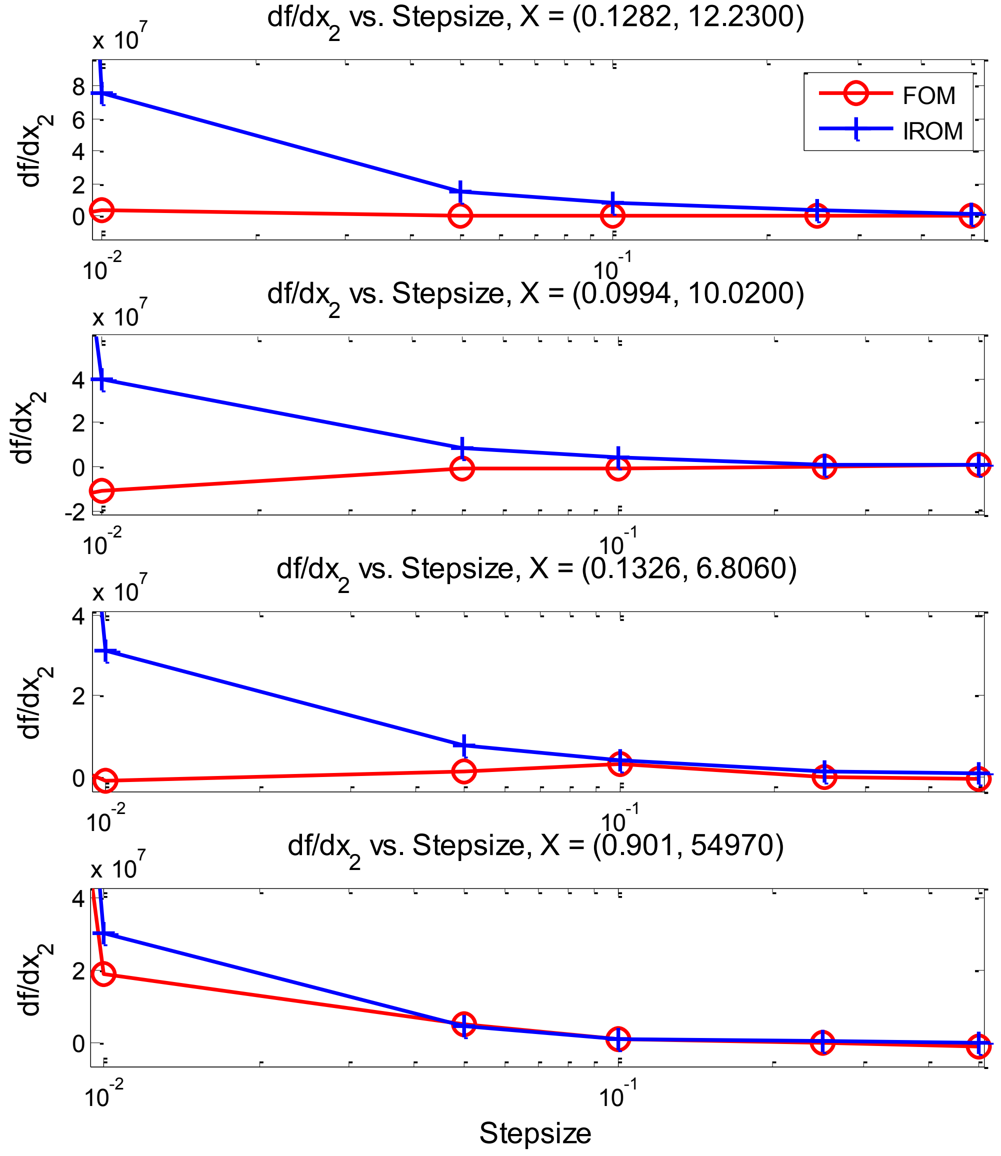

Figure 6.20 Aspect Ratio Stepsize Study (Additional Points)

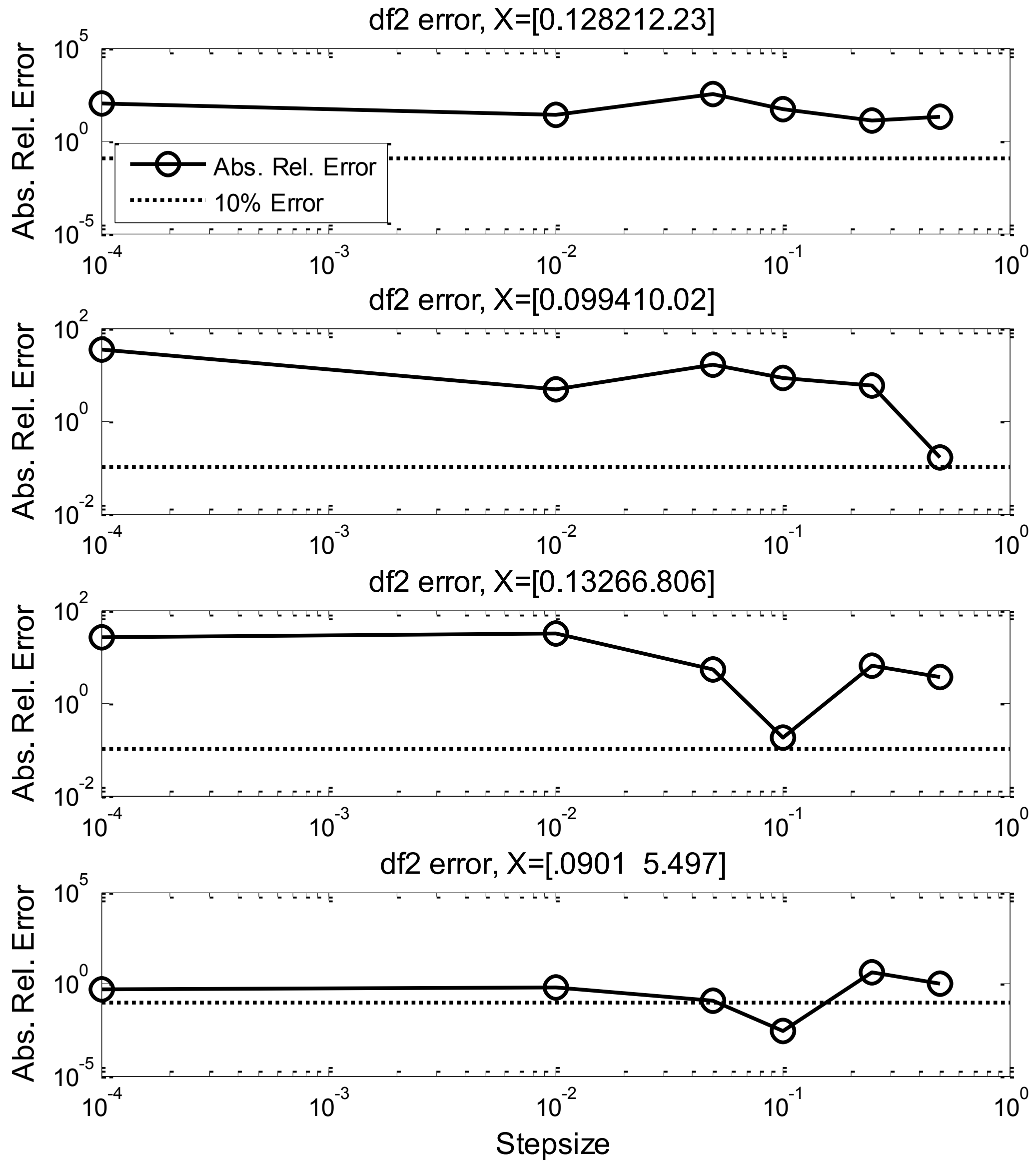

Figure 6.21 Aspect Ratio Stepsize Study Error (Additional Points)

As a result of this study, a stepsize of 0.01 was selected for the airfoil thickness and 0.1 for the aspect ratio.

6.2.5.2 Non-adaptive Local Error

The local error points were retested using the new stepsizes. Test point 9 did not complete due to a technical error. The results are shown in Table 6.11.

Table 6.11 TWO Local Error (Revised)

ID	1	2	3	4	5	6	7	8	9	10
SPERR (%)	6.39	97.12	2.34	3.02	1.77	76.10	0.21	3.74	-	0.25

6.2.5.3 Optimization Path

The optimization path for the revised first test point is shown in Figure 6.22. The objective and design variable histories are show in Figure 6.23. The FOM reached an optimum at 8% airfoil thickness and an aspect ratio of 15, while the IROM reached an airfoil thickness of 8% and an aspect ratio of 6.0002. The FOM optimum had a range of 13,441 km (7,257 nm), while the IROM optimum had a range of 10,179 k m (5,496 nm).

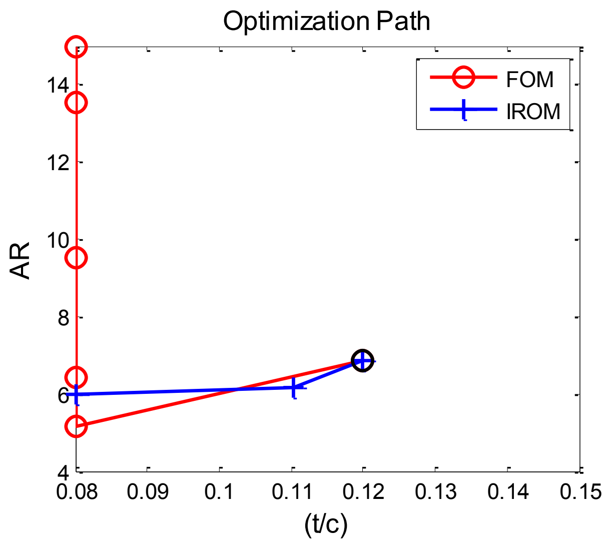

Figure 6.22 Optimization Path, Test Point 1 (Revised)

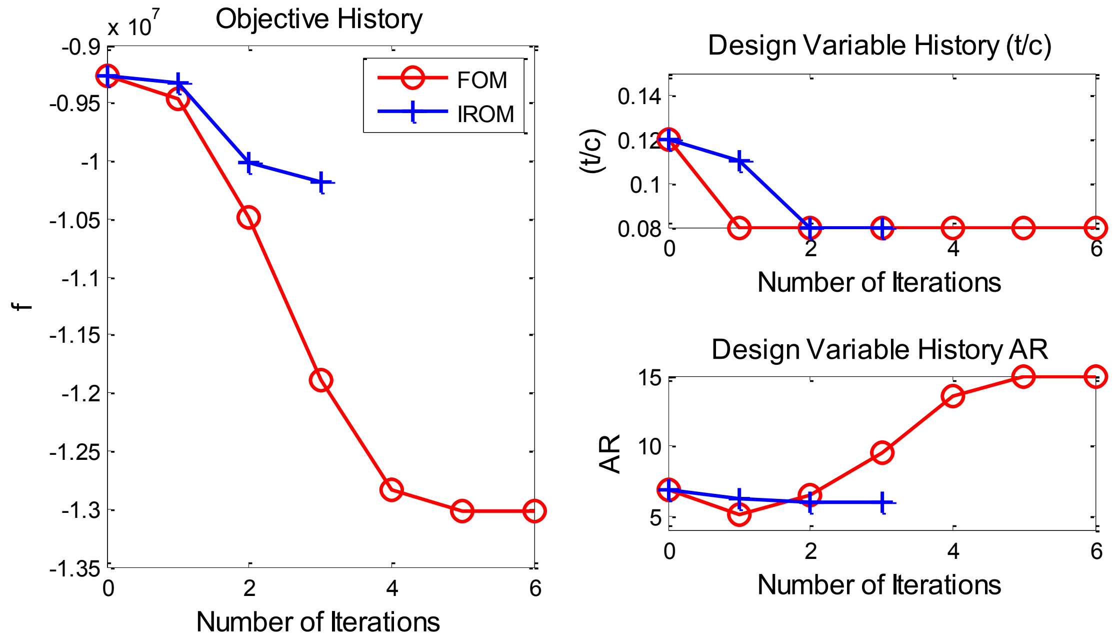

Figure 6.23 Objective and Design Histories, Test Point 1 (Revised)

Additionally, for this run information on the gradient histories and the search directions was retained. These histories are shown in Figure 6.24 and Figure 6.25, respectively.

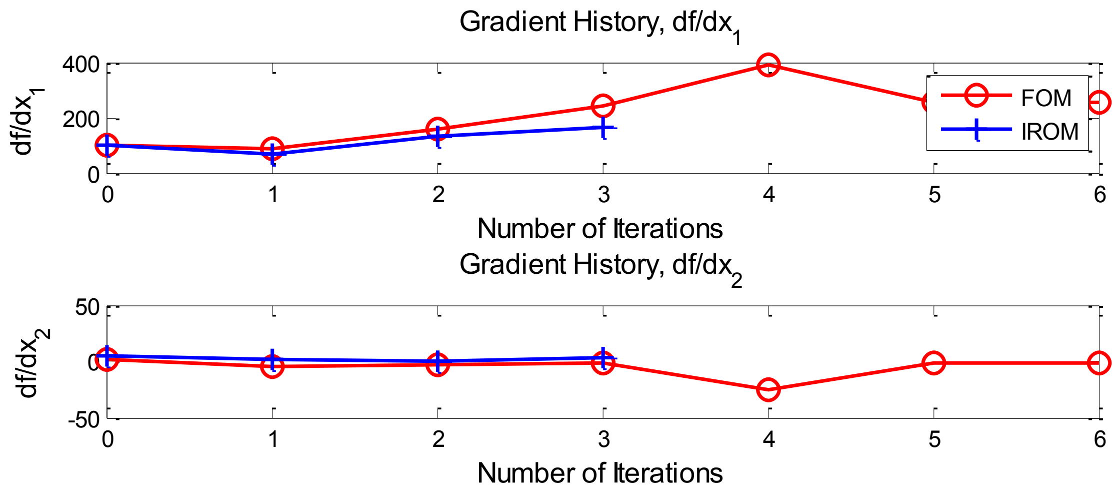

Figure 6.24 Gradient History, Test Point 1 (Revised)

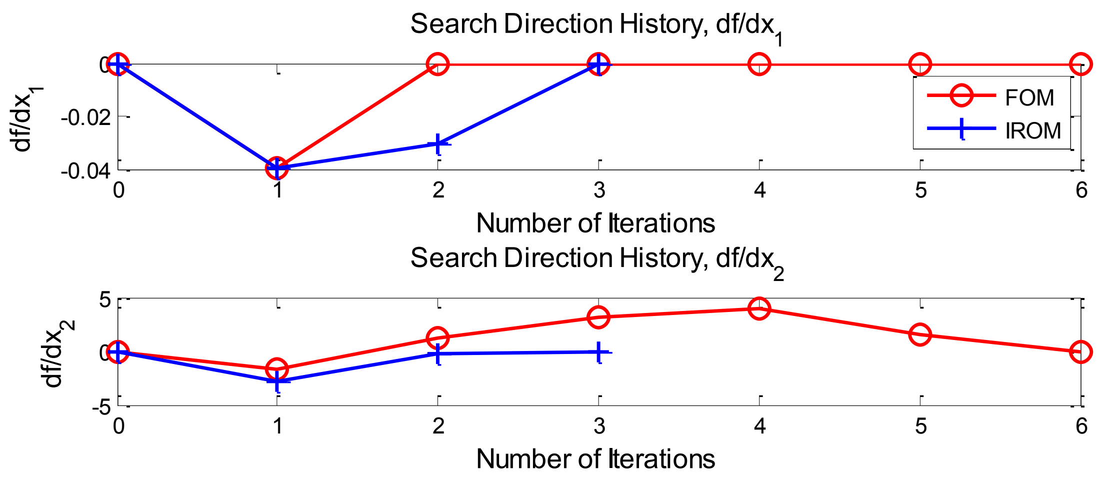

Figure 6.25 Search Direction History, Test Point 1 (Revised)

6.2.5.4 Function Counts

The total number of function calls for the revised first test point is given below in Table 6.12.

Table 6.12 TWO Function Calls, Test Point 1 (Revised)

Function	Total (FOM)	Total (IROM)	Avg/Obj (FOM)	Avg/Obj (IROM)
Objective	13	11	1.00	1.00
(w/ Gradient)	7	4	0.53	0.36
Suboptimization	18	16	1.38	1.45
(w/ FOM)	18	8	1.38	0.72
(w/ IROM)	0	8	0.00	0.72
Subopt Objective	65,914	47,611	5,070.30	4,328.27
Subopt Constraints	1,669	1,222	128.38	111.09
(w/ FOM)	1,669	568	128.38	51.63
(w/ IROM)	0	654	0.00	59.45
(w/ Gradient)	1,669	1,222	128.38	111.09
Full Order Analysis	66,727	21,064	5,132.84	1,914.90
IROM Interpolation	0	16	0.00	1.45

6.2.6 CPU and Wallclock Times

Summaries of the total CPU and wallclock times are given in Table 6.13.

Table 6.13 TWO CPU and Wallclock Times

Component	Total CPU Time (s)	Parallelism	Wallclock (s)
Full Order Analysis	~11	1	~11.0
RBF Construction (2 kernels, 50 pts)	~6.80	4	1.7
Suboptimization (FOM)	~45,440	4	~11,360.0
Suboptimization (IROM)	~781	4	~195.0
IROM Interpolation (no update)	~6,823	1	2,078.0
Y_{bias} Models	~0	4	~0.0
$\tilde{Y}$ Models	~6,326	4	1,581.0
Stencil Evaluation Only (FOM)	24,291	1	24,291.0
Stencil Evaluation Only (IROM)	7,360	1	7,360.0

6.2.7 Discussion

6.2.7.1 Preoptimization error

The initial error estimation for the 50-point Latin hypercube DOE design exhibited moderate to high error. The global error had a mean of about 30%, and the local error closer to 50%. Later examination of the test set indicated that the randomly generated points were not well-distributed, favoring the lower aspect ratios. Due to limited computational resources, the adaptive local error could not be explored in a reasonable timeframe.

The revised stepsizes yielded greatly improved local error, with a mean of about 20%, and a median of 3.02%. It is also clear that there were outlier points with significant error.

Based on this information, steps to improve the accuracy such as a new DOE design, or incorporation of the adaptive step, would be advised. However, these steps were not taken here due to time limitations.

6.2.7.2 IROM and SQP Implementation

Before discussing the test points, it is useful to overview MATLAB's implementation of SQP and tolerances. MATLAB breaks its implementation into three main stages: updating the Hessian approximation, solving the direction finding (QP) subproblem, and performing a line search. The tolerances involve an optimality criterion, TolFun, change in the design variables, TolX, and violation of the constraints, TolCon.

The Hessian approximation is maintained as the solver progresses, and updated at each iteration. The general form for this update is

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{\boldsymbol{H}_k s_k s_k^T \boldsymbol{H}_k^T}{s_k^T \boldsymbol{H}_k s_k} \qquad (6.1)$$

where the Hessian approximation $\boldsymbol{H}$ at iteration $k+1$ is based on an update from the previous data. The update parameters are defined as

$$s_k = x_{k+1} - x_k \qquad (6.2)$$

and

$$q_k = \left[\nabla f(x_{k+1}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x_{k+1})\right] - \left[\nabla f(x_k) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x_k)\right] \qquad (6.3)$$

For this work, it is useful to be aware that the error in the Hessian approximation will be affected by the error in the objective and constraint gradients; more specifically, by the difference in error between iteration points. For the TWO system-level problem, only the objective gradient has error introduced by the IROM method, since the only constraints at the system level are the side bounds on the design variables.

The approximate Hessian and the linearized constraints are used to form a quadratic programming subproblem, which is responsible for determining the search direction. This problem has the general form

$$\begin{array}{llll} \text{min} & q(d) = \frac{1}{2} d^t \boldsymbol{H} d + c^T d & & \\ \text{over} & d & & \\ \text{s.t.} & A_i d = b_i & i = 1{:}\, m_{eq} & \\ & A_i d \leq b_i & i = \left(m_{eq} + 1\right){:}\, m & (6.4) \end{array}$$

That is, find the direction d which decreases the objective function while satisfying the (linearized) constraints. This problem is solved using an active set method, where feasible QP search directions are defined by the set of active constraints.

For this work, it is important to note that error in the Hessian approximation with naturally affect the results of this QP subproblem. Additionally, if gradients are used in linearizing the constraints, gradient error will affect those as well. In this work, the system constraint gradients (upper and lower bounds) are not affected by the IROM error.

After a suitable search direction has been found, the SQP algorithm proceeds with a line search along that direction. This search procedure is not seeking to directly minimize the objective function, but rather to minimize a merit function. This function adds penalty parameters as multipliers for the constraints. The form of this merit function is

$$\varphi(x) = f(x) + \sum_{i=1}^{m_{eq}} r_i g_i(x) + \sum_{i=m_{eq}+1}^{m} r_i \max\{0, g_i(x)\} \tag{6.5}$$

where the penalty parameter, r_i, is set iteratively as

$$r_i = (r_{k+1})_i = \max_i \left\{\lambda_i, \frac{(r_k)_i + \lambda_i}{2}\right\} \tag{6.6}$$

More importantly, the initial estimate for r_i is determined as

$$r_i = \frac{\|\nabla f(x)\|_2}{\|\nabla g_i(x)\|_2} \tag{6.7}$$

which shows that gradient error will impact the initial merit estimate, as well as subsequent updates to the merit function. For the TWO system-level problem, the constraint gradients are computed directly by MATLAB and are constants; thus the error in the initial penalty parameter calculation will be error in the magnitude of the objective gradient, scaled by the appropriate factor. The propagation of this error through the iterative updates may explain why the IROM line searches tend to take somewhat longer than the FOM line searches.

MATLAB has several tolerance settings for use with the SQP-based optimizer. The primary tolerances of concern here are change in design variables, TolX, the first order optimality criterion, TolFun, and the constraint violation tolerance, TolCon.

TolX is used as a stopping condition; if the magnitude of change in design variables is less than TolX, the optimization is stopped. It also has an effect on the line search by determining some of the step sizes used, with larger tolerances allowing for larger adjustments in step size.

TolFun is used to determine when the optimizer is no longer able to improve the design by a meaningful amount. Instead of relying on the objective function directly, a first order optimality criterion (FOOC) rooted in the augmented Lagrangian equation is used. Additionally, in the solver implementation, side bounds, linear, and nonlinear constraints are separated. The result is two criteria which are combined into the FOOC, namely

$$\|\nabla_x L(x,\lambda)\|_\infty = \left\| \begin{matrix} \nabla f(x) + A^T\lambda_{in,lin} + A_{eq}^T\lambda_{in,eq} \\ + \sum \lambda_{in,non}\nabla c_i(x) + \sum \lambda_{eq,non}\nabla c_{eq,i}(x) \end{matrix} \right\|_\infty \tag{6.8}$$

where the subscripts in and eq denote variables associated with inequality and equality constraints, respectively, and lin and non denote linear and nonlinear constraints, respectively. The other criterion is given by

$$\left\|\left\lceil \lambda_g g(x) \right\rceil\right\|_\infty = \left\| \begin{matrix} \left\lceil |l_i - x_i|\lambda_{lower,u} \right\rceil, \ \left\lceil |x_i - l_i|\lambda_{upper,i} \right\rceil, \\ \left\lceil |(Ax-b)_i|\lambda_{in,lin,i} \right\rceil, \ \left\lceil |c_i(x)|\lambda_{in,non,i} \right\rceil \end{matrix} \right\|_\infty \tag{6.9}$$

The maximum infinity norm of these criteria forms the combined optimality criterion, the FOOC:

$$\max\left\{ \left\|\left\lceil \lambda_g g(x) \right\rceil\right\|_\infty, \ \|\nabla_x L(x,\lambda)\|_\infty \right\} \tag{6.10}$$

For the IROM method, this means that errors in the objective and constraint gradients can have a significant impact on the optimality criterion.

The constraint violation tolerance, TolCon, is largely a measure of the distance between and design point and a constraint before the constraint is considered active or violated. It has an impact both on the main SQP iterations and the QP subproblem.

From this it is clear that, as expected, the main impact the IROM scheme has on the SQP scheme is due to the introduction of error into the gradient calculations. This is particularly evident in the Hessian update and constraint linearizations (forming the basis of the QP subproblem, and hence search direction determination), the line search merit function, and the first order optimality criterion.

6.2.7.3 Test Point 1

The first optimization was performed starting from the reference wing configuration. The FOM and IROM exhibit significantly different paths after the first two iterations, with the FOM reaching a stopping condition at a corner of the design space and the IROM continuing on. The FOM stoppage is largely due to the impact of convergence tolerances, which were overly tight (1e-6) for this problem.

The IROM takes a different path due to high gradient error at the corner point, which impacts the search direction as well as the convergence metrics used in MATLAB. At the first two iterations, the finite difference stencils are identical. This allows the comparison of both the computed range (objective function) at those stencil points as well as the gradients. The range error at the initial point is 2.80% and 0.71% for each stencil point, respectively. This results in a local error of 13.77%. However, at the corner point, the range errors are 14.89% and 14.68%, resulting in a local error of 86.58%. Although the IROM ends up finding a better optimum than the FOM solution, it is largely chance for this case.

As previously mentioned, the convergence tolerances were tight for this case, at 1e-6 for all tolerances; this resulted in an excessive number of objective constraints during the line search, as well as affecting the stopping conditions for the optimization. The subsequent test points each loosened the tolerances until an adequate balance was determined.

The FOM optimization for this point was then rerun with the looser tolerances. In this case, the FOM optimization finds an optimum at 8% airfoil thickness and an aspect ratio of 15.0.

Although it was not a primary objective for this test problem, the function counts for this case do indicate a significant reduction in the number of calls to the full order analysis.

6.2.7.4 Test Point 2

The second test point was selected at random, and ended up somewhat close to the reference wing. It also implemented somewhat looser tolerances, with most constraints set to 1e-4 and the constraint tolerance (TolCon) for the suboptimization problem set to 1e-6. The function counts indicate that there is still an excessive number of evaluations during the line search, suggesting the system-level design point tolerance (TolX) could be loosened.

The FOM optimization for this point finds a similar optimum to the first test point with looser tolerances, ending at an airfoil thickness of 8% and an aspect ratio of 15.0. The IROM path again follows the FOM path accurately from the initial point, although it then diverges to an optimum with similar aspect ratio and greater airfoil thickness than that found in the first optimization. A significant reduction in the number of full order analysis calls is also shown.

6.2.7.5 Test Point 3

The third test point was another randomly selected starting point. The tolerances were again loosened, to 1e-3 for most and 1e-4 for the suboptimization constraints (TolCon). The FOM optimization again finds 8% airfoil thickness and an aspect ratio of 15.0 to be the optimum. For this point, the IROM exhibits a gradient error at the starting point of 16.47%, leading to an incorrect search direction. A slight decrease in the number of full order analysis calls is also observed.

For the third test point, higher quality data was collected about the system and sublevel optimizations. Comparing the gradients and search directions for the system problem, it appears that even with the interpolation model error, the partial derivative with respect to $\left(\frac{t}{c}\right)$ is generally correct; however, the partial derivative with respect to AR is often not. This is probably due to the choice of stepsize, which was selected as 0.001 for both variables. This is equivalent to a change of 0.1% airfoil thickness, but only 0.001 in aspect ratio. This change in AR is probably too small to escape the local noise introduced by the interpolation error. A larger stepsize for AR, such as 0.01 or even 0.1, may provide better global trend information. This motivated the more detailed stepsize study and revision of the first test point.

6.2.7.6 Test Point 1 (Revised)

The stepsize study indicates that the previous stepsize of 0.001 for both variables yielded a high degree of inaccuracy. It can also be seen from the detailed views of the objective function's partial derivative with respect to aspect ratio that the function is quite noisy. Although the IROM and FOM derivatives do have a region of relative convergence, this noise makes a good match quite difficult.

The revised stepsizes and optimization tolerances seem to have greatly improved the results for the first test problem. From the optimization path, as well as the gradient and search direction histories, it is clear that the IROM approximates the FOM path. However, it then terminates prematurely at a low aspect ratio, while the FOM optimization continues to the corner of the design space. This likely indicates that the optimization tolerances require further adjustment.

6.2.7.7 General Test Point Discussion

In retrospect, the FOM optimums of 8% airfoil thickness and an aspect ratio of 15.0 is not overly surprising. Looking at the problem formulation, there is little benefit to having a larger $\left(\frac{t}{c}\right)$. It only adds weight and drag penalties, and the structural model is likely too simplistic to accurately capture the impact of airfoil thickness on structural loading. Furthermore, the airfoil sampled in this problem is a simple NACA 0012 scaled in thickness, and the wing is held fixed at a very low angle of attack (1.5 deg). A minimum volume constraint, a higher fidelity finite element model, or a different airfoil (such as one with camber) may improve this aspect of the problem.

Based on the FOM paths, especially comparing aspect ratio and range for the third test point, there is an indication that the aspect ratio may be converging to a point beyond the upper bound of 15.0. This is shown in Figure 6.26. Given the numerous simplifications in this problem, especially as weights and buckling analysis are concerned, this seems likely. Although this high aspect ratio is probably not the real-world optimum for a wing of this type, it may reflect the optimum for the very simplified problem formulation used here. To test this, an additional FOM run with the upper bound on aspect ratio raised to 20 was performed. From the results in Figure 6.26, the problem does indeed trend toward an optimum at an aspect ratio of approximately 19. The IROM could not be compared, since extending the valid design space would entail extending the DOE design and rebuilding the interpolation model.

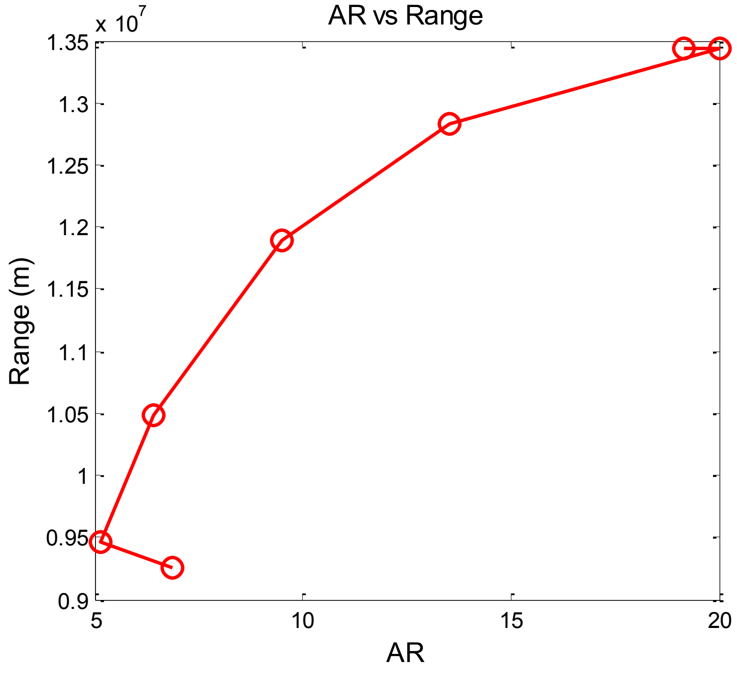

Figure 6.26   Aspect Ratio vs. Range, FOM Optimization, Test Point 3

There are only a few test points during the optimizations where the IROM capability to serve as a surrogate is directly evaluated, namely where the FOM and IROM optimizations evaluate gradients at the same design points. The local error test points also provide information about this capability. The error in range (objective function) for these points is a convenient metric, as it relies on the suboptimization for that point using the FOM or IROM models. These are summarized in Table 6.14. In general, the accuracy is fairly good, with a mean of 6.86% and a median of 4.65%. Although higher accuracy is desirable for the gradient calculations, this data illustrates that the IROM can serve as a viable surrogate model for part of a larger multidisciplinary system.

Table 6.14 Comparison of FOM/IROM Range Errors for Incidental Design Points

(t/c)	AR	R_{FOM} (m)	R_{IROM} (m)	**Abs. Rel Error**
0.1210	6.8571	9,327,408	9,066,227	**2.8%**
0.1200	6.8581	9,164,899	9,099,163	**0.7%**
0.1510	15.0000	9,319,601	7,931,061	**14.8%**
0.1500	15.0010	9,353,585	7,980,037	**14.6%**
0.1253	5.0729	8,468,909	8,321,133	**1.7%**
0.1243	5.0739	8,494,590	8,344,325	**1.7%**
0.1480	9.3391	7,606,027	6,932,099	**8.8%**
0.1470	9.3401	7,619,787	7,245,046	**4.9%**
0.1470	5.7337	7,885,270	7,395,990	**6.2%**
0.1370	5.8337	7,363,334	7,590,154	**3.0%**
0.1534	14.6765	8,817,037	7,225,864	**18.0%**
0.1434	14.7765	8,822,910	7,746,912	**12.1%**
0.0989	14.5288	10,231,697	10,413,650	**1.7%**
0.0889	14.6288	11,897,543	11,067,975	**6.9%**
0.1539	9.3391	7,162,996	7,722,964	**7.8%**
0.1439	9.4391	7,308,198	8,061,679	**10.3%**
0.1343	12.8031	9,261,435	8,337,830	**9.9%**
0.1243	12.9031	8,963,434	8,867,045	**1.0%**
0.0968	5.5607	9,161,980	8,844,721	**3.4%**
0.0868	5.6607	9,520,005	9,256,519	**2.7%**
0.1095	8.6394	9,727,857	9,417,382	**3.1%**
0.0995	8.7394	10,131,166	9,829,401	**2.9%**
0.1283	14.0731	7,456,930	8,564,667	**14.8%**
0.1183	14.1731	10,425,793	8,740,012	**16.1%**
0.1575	14.5544	7,550,696	7,217,357	**4.4%**
0.1475	14.6544	8,894,204	8,526,536	**4.1%**

6.2.7.8 CPU/Wallclock Time

Although the TWO problem was not intended to demonstrate computational speedup, representative times were still measured for the first test point run. It is clear that the suboptimization time using an IROM is significantly faster than using the FOM.

The stencil evaluation indicates significant speedup for the evaluation itself, due to the decreased suboptimization time; however, in this case, there are four IROM interpolations per system objective gradient evaluations (two load cases times two design variables). Each of these interpolations involves approximately 20,000 RBF models; thus the CPU time for evaluating the objective gradient with IROM models is actually greater than the FOM models for this problem, approximately 35,000 seconds for the IROM as opposed to approximately 24,000 seconds for the FOM. However, it is important to note that the base finite element analysis takes approximately 12 seconds of CPU time in this problem; it is easy to see that with a more expensive analysis, the IROM speedup benefit would improve significantly.

CHAPTER VII

CONCLUSIONS

7.1 Summary

A new method was developed for using reduced order models in lieu of high fidelity analysis during the sensitivity analysis step. A combination of proper orthogonal decomposition and radial basis functions was used to develop ROMs. Optimization with the full-order and interpolated reduced-order models was performed on airfoil shape optimization and transport wing optimization test problems. The errors associated with the ROMs themselves as well as the gradients calculated from them were also compared. The effects of each approach on the overall optimization paths and function counts were also examined.

The ASO results illustrated that the proposed interpolation scheme is a viable candidate for significantly reducing the computational cost of performing optimization with expensive analyses. It also revealed several challenges and caveats involved with applying the interpolation scheme.

The TWO results indicate that the IROM is capable of fitting into a multidisciplinary analysis structure as a more traditional surrogate model. It also indicates that accuracy remains a highly problem-dependent challenge.

7.2 Future Work

There is a great deal of future work which can be performed. For example, investigating the impact of the interpolation process on the ROM approximation accuracy would be useful for further investigating the replacement an analysis in a MDO context. The impact of reduced basis sets in the POD models is also a subject for future investigation. Given the natural benefits of this research work for MDO problems, integrating the method into MDO frameworks may also be a productive research path.

REFERENCES

1 Jamshid A. Samareh, and Kumar G. Bhatia, "A Unified Approach to Modeling Multidisciplinary Interactions," 2000.

2 Dowell, E. H., Hall, K. C., Thomas, J. P., Kielb, R. E., Spiker, M. A., Li, A., and Denegri, C. M., "Reduced order models in unsteady aerodynamic models, aeroelasticity and molecular dynamics," *Proceedings of the 26th ICAS Congress, Anchorage, Alaska, USA*, 2008.

3 Acar, E., and Rais-Rohani, M., "Ensemble of metamodels with optimized weight factors," *Structural and Multidisciplinary Optimization*, vol. 37, Feb. 2008, pp. 279–294.

4 Antoulas, A. C., Sorensen, D. C., and Gugercin, S., "A survey of model reduction methods for large-scale systems," *Structured matrices in mathematics, computer science, and engineering: proceedings of an AMS-IMS-SIAM joint summer research conference, University of Colorado, Boulder, June 27-July 1, 1999*, 2001, p. 193.

5 Burges, C. J. C., "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, 1998, pp. 121–167.

6 Friedman, J. H., "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, Mar. 1991, pp. 1–67.

7 Giunta, A. A., Swiler, L. P., Brown, S. L., Eldred, M. S., Richards, M. D., and Cyr, E. C., "The Surfpack Software Library for Surrogate Modeling of Sparse Irregularly Spaced Multidimensional Data," Portsmouth, Virginia: 2006.

8 Jin, R., Chen, W., and Simpson, T. W., "Comparative studies of metamodelling techniques under multiple modelling criteria," *Structural and Multidisciplinary Optimization*, vol. 23, 2001, pp. 1–13.

9 Jones, D. R., "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *Journal of Global Optimization*, vol. 21, 2001, pp. 345–383.

10 Mullur, A. A., and Messac, A., "Extended radial basis functions: more flexible and effective metamodeling," *AIAA Journal*, vol. 43, Jun. 2005, pp. 1306–1315.

[11] Rumelhart, D. E., Widrow, B., and Lehr, M. A., "The Basic Ideas in Neural Networks," *Communications of the ACM*, vol. 37, Mar. 1994, pp. 86–92.

[12] Qian, Z., Seepersad, C. C., Joseph, V. R., Allen, J. K., and Wu, C. F. J., "Building Surrogate Models Based on Detailed and Approximate Simulations," *Journal of Mechanical Design*, vol. 128, Jul. 2006, pp. 668–697.

[13] Richard, R. E., Rule, J. A., and Clark, R. L., "Genetic Spatial Optimization of Active Elements on an Aeroelastic Delta Wing," *Journal of Vibration and Acoustics*, vol. 123, 2001, p. 466.

[14] Simpson, T. W., Peplinski, J. D., Koch, P. N., and Allen, J. K., "Metamodels for Computer-based Engineering Design: Survey and recommendations," *Engineering with Computers*, vol. 17, 2001, pp. 129–150.

[15] Wang, G. G., and Shan, S., "Review of Metamodeling Techniques in Support of Engineering Design Optimization," *Journal of Mechanical Design*, vol. 129, Apr. 2007, pp. 370–380.

[16] Zhou, Z., Ong, Y. S., Nair, P. B., Keane, A. J., and Lum, K. Y., "Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization," 2005.

[17] Alexandrov, N. M., Lewis, R. M., Gumbert, C. R., Green, L. L., and Newman, P. A., *Optimization with Variable-fidelity Models Applied to Wing Design*, 1999.

[18] Booker, A. J., "Design and Analysis of Computer Experiments," *AIAA Journal*, 1998, pp. 118–128.

[19] Davis H. Crawford, *Investigation of the flow over a spiked-nose hemisphere-cylinder at a mach number of 6.8*, 1959.

[20] Culler, A. J., and McNamara, J. J., "Impact of Fluid-Thermal-Structural Coupling on Response Prediction of Hypersonic Skin Panels," *AIAA Journal*, vol. 49, Nov. 2011, pp. 2393–2406.

[21] Deveikis, W. D., and Sawyer, J. W., *NASA TN D-6281*, National Aeronautics and Space Administration, 1971.

[22] Jack P. C. Kleijnen, "An overview of the design and analysis of simulation experiments for sensitivity analysis," *European Journal of Operational Research*, 2004.

[23] Leary, S., Bhaskar, A., and Keane, A., "Optimal orthogonal-array-based latin hypercubes," *Journal of Applied Statistics*, vol. 30, 2003, pp. 585–598.

[24] Liu, F., Cai, J., Zhu, Y., Tsai, H. M., and Wong, A. S. F., "Calculation of wing flutter by a coupled fluid-structure method," *Journal of Aircraft*, vol. 38, 2001, pp. 334–342.

[25] Joaquim R. R. A. Martins, Juan J. Alonso, and James J. Reuther, "Complete configuration aero-structural optimization using a couple sensitivity analysis method," *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, USA: AIAA, 2002.

[26] Mullur, A. A., and Messac, A., "Metamodeling using extended radial basis functions: a comparative approach," *Engineering with Computers*, vol. 21, 2006, pp. 203–217.

[27] Olds, J. R., "Multidisciplinary Design Techniques Applies to Conceptual Aerospace Vehicle Design," Dissertation, North Carolina State University, 1993.

[28] Ong, Y. S., "Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling," *AIAA Journal*, vol. 41, 2003, pp. 687–696.

[29] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., "Design and Analysis of Computer Experiments," *Statistical Science*, vol. 4, Nov. 1989, pp. 409–423.

[30] Srivastava, A., Hacker, K., Lewis, K., and Simpson, T. W., "A method for using legacy data for metamodel-based design of large-scale systems," *Structural and Multidisciplinary Optimization*, vol. 28, 2004, pp. 146–155.

[31] Stanley, D. O., Unal, R., and Joyner, C. R., "Application of Taguchi Methods to Propulsion System Optimization for SSTO Vehicles," *Journal of Spacecraft and Rockets*, vol. 29, Aug. 1992, pp. 453–459.

[32] Tang, B., "Orthogonal Array-Based Latin Hypercubes," *Journal of the American Statistical Association*, vol. 88, Dec. 1993, pp. 1392– 1397.

[33] Ye, K. Q., Li, W., and Adjianto, A., "Algorithmic construction of optimal symmetric Latin Hypercube Designs," *Journal of Statistical Planning and Inferences*, vol. 90, 2000, pp. 145–159.

[34] *VisualDOC*, Novi, MI, USA.: Vanderplaats Research & Development, Inc, 2013.

[35] *Matlab Optimization Toolbox*, Natick, MA, USA: MathWorks, 2013.

[36] A.R. Parkinson, and R.J. Balling, "The OptdesX design optimization software," *Structural Multidisciplinary Optimization*, vol. 23, 2002, pp. 127–139.

[37] Cramer, E. J., Dennis Jr, J. E., Frank, P. D., Lewis, R. M., and Shubin, G. R., "Problem formulation for multidisciplinary optimization," 1993.

38 Sobieszczanski-Sobieski, J., and Haft, R. T., “Multidisciplinary aerospace design optimization: survey of recent developments,” *Structural Optimization*, vol. 14, 1997, pp. 1–23.

39 Jeremy Agte, Olivier de Weck, Jaroslaw Sobieszczanski-Sobieski, Paul Arendsen, Alan Morris, and Martin Spieck, “MDO: assessment and direction for advancement - an opinion of one international group,” *Structural Multidisciplinary Optimization*, vol. 40, 2010, pp. 17–33.

40 Philippe Depince, Benoit Guedas, and Jerome Picard, “Multidisciplinary and multiobjective optimization: comparison of several methods,” *7th World Congress on Structural and Multidisciplinary Optimization*, Seoul, Republic of Korea: 2007.

41 Thomas A Zang, and Lawrence L. Green, “Multidisciplinary design optimization techniques: implications and opportunities for fluid dynamics research,” *Proceedings of the 30th AIAA Fluid Dynamics Conference*, Norfolk, VA, USA: AIAA, 1999.

42 Terrance C. Wagner, and Panos Y. Papalambros, “A general framework for decomposition analysis in optimal design,” *DE*, vol. 65-2, 1993.

43 Nestor Michelena, and Panos Papalambros, “A hypergraph framework for optimal model-based decomposition of design problems,” *Computational Optimization and Applications*, vol. 8, Sep. 1997, pp. 173–196.

44 R. Ganguli, “Optimum design of a helicopter rotor for low vibration using aeroelastic analysis and response surface methods,” *Journal of Sound and Vibration*, vol. 58, 2002, pp. 327–344.

45 Nikbay, M., Öncü, L., and Aysan, A., “Multidisciplinary Code Coupling for Analysis and Optimization of Aeroelastic Systems,” *Journal of Aircraft*, vol. 46, Nov. 2009, pp. 1938–1944.

46 Massimiliano Zingales, and Isaac Elishakoff, “Hybrid Aeroelastic Optimization and Antioptimization,” *AIAA journal*, vol. 39, Jan. 2001, pp. 161–175.

47 Gasbarri, P., Chiwiacowsky, L. D., and de Campos Velho, H. F., “A hybrid multilevel approach for aeroelastic optimization of composite wing-box,” *Structural and Multidisciplinary Optimization*, vol. 39, 2009, pp. 607–624.

48 Martins, J., Alonso, J. J., and Reuther, J. J., “High-fidelity aerostructural design optimization of a supersonic business jet,” *Journal of Aircraft*, vol. 41, 2004, pp. 523–530.

49 Sharon L Padula, Benjamin B James, Philip C Graves, and Stanley E Woodard, *Multidisciplinary optimization of controlled space structures with global sensitivity equations*, Hampton, VA, USA: NASA Langley Research Center, 1991.

50 Gumbert, C. R., Hou, G. J. W., and Newman, P. A., "Simultaneous aerodynamic analysis and design optimization (SAADO) for a 3-D flexible wing," *AIAA Paper*, vol. 1107, 2001, p. 2001.

51 Bartholomew, P., "The Role of MDO Within Aerospace Design and Progress Towards an MDO Capability," *AIAA Journal*.

52 J.J. Korte, R.P. Weston, and T.A. Zang, "Multidisciplinary optimization methods for preliminary design."

53 H.C. Ajmera, P.M. Mujumdar, and K. Sudhakar, "MDO architectures for coupled aerodynamic and structural optimization of a flexible wing," *Proceedings of the 45th AIAA/ASME/ASCE/ARS/ASC Structures, Structural Dynamics & Materials Conference*, Palm Springs, CA, USA: AIAA, 2004.

54 John K. Lytle, "The Numerical Propulsion System Simulation: A Multidisciplinary Design System for Aerospace Vehicles," *Proceedings of the 14th International Symposium on Air Breathing Engines*, Florence, Italy: NASA, 1999.

55 K.F. Hulme, and C.L. Bloebaum, "Development of CASCADE: a multidisciplinary design test simulator," *AIAA*, 1996.

56 Kevin F. Hulme, "Development of CASCADE - A Test Simulator for Modelling Multidisciplinary Design Optimization Problems in Distributed Computing Environments," Masters Thesis, State University of New York at Buffalo, 1996.

57 K.F. Hulme, and C.L. Bloebaum, "Development of a simulation-based framework for exploting new tools and techniques in multidisciplinary design optimization," 1999.

58 Kevin F. Hulme, "The design of a simulation-based framework for the development of solution approaches in multidisciplinary design optimization," Doctoral Thesis, State University of New York at Buffalo, 2000.

59 Sharon L. Padula, Natalia Alexandrov, and Lawrence L. Green, "MDO test suite at NASA Langley research center," *Proceedings of the 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, WA, USA: AIAA, 1996.

[60] S. Parashar, K. English, and C.L. Bloebaum, “Data transmission in multidisciplinary design optimization using a platform-independent data structure,” *Proceedings of the 42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA: AIAA, 2004.

[61] Patrick A LeGresley, and Juan J Alonso, “Improving the performance of design decomposition methods with POD,” Aug. 2004.

[62] Rajan Filomeno Coelho, Piotr Breitkopf, and Catherine Knopf-Lenoir, “Model reduction for multidisciplinary optimization - application to a 2D wing,” *Structural and Multidisciplinary Optimization*, vol. 37, Apr. 2008, pp. 29–48.

[63] G. Agrawal, S. Parashar, K.W. English, and C.L. Bloebaum, “Web-based visualization framework for decision-making in multidisciplinary design optimization,” *ASME Journal of Computing and Information Science in Engineering*.

[64] Arun N. Nambiar, “Data exchange in multi-disciplinary optimization frameworks,” Masters of Science, Ohio University, 2004.

[65] Donald R. Jones, Matthias Schonlau, and William J. Welch, “Efficient Global Optimization of Expensive Black-Box Functions,” *Journal of Global Optimization*, vol. 13, 1998, pp. 455–492.

[66] J. Olds, “System Sensitivity Analysis Applied to the Conceptual Design of a Dual-Fuel Rocket SSTO,” *Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, FL, USA: AIAA, 1994.

[67] Joaquim R. R. A. Martins, “Sensitivity Analysis,” *Multidisciplinary Design Optimization*.

[68] J.L. Rogers, and C.L. Bloebaum, “Ordering design tasks based on coupling strengths,” *Proceedings of the Fifth AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, Florida, USA: AIAA, 1994.

[69] James C. Newman III, Arthur C. Taylor lil, Richard W. Barnwell, Perry A. Newman, and Gene J.-W. Hou, “Overview of Sensitivity Analysis and Shape Optimization for Complex Aerodynamic Configurations,” *Journal of Aircraft*, vol. 36, Jan. 1999, pp. 87–96.

[70] Malone, B., and Mason, W. H., “Multidisciplinary Optimization in Aircraft Design Using Analytic Technology Models,” *Journal of Aircraft*, vol. 32, Apr. 1995, pp. 431–438.

[71] Maute, K., Nikbay, M., and Farhat, C., "Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems," *AIAA journal*, vol. 39, 2001, pp. 2051–2061.

[72] Pradeep Raj, "Aircraft design in the 21st century: implications for design methods," Nov. 1998.

[73] Sulaiman F. Alyaqout, Panos Y. Papalambros, and A. Galip Ulsoy, "Quantification and use of system coupling in decomposed design optimization problems," *Proceedings of IMECE2005*, Orlando, Florida, USA: ASME, 2005.

[74] I. Kroo, "Distributed multidisciplinary design and collaborative optimization," Nov. 2004.

[75] Tappeta, R. V., Nagendra, S., and Renaud, J. E., "A multidisciplinary design optimization approach for high temperature aircraft engine components," *Structural and Multidisciplinary Optimization*, vol. 18, Oct. 1999, pp. 134–145.

[76] Srinivas Kodiyalam, and Jaroslaw Sobieszczanski-Sobieski, "Bi-Level Integrated System Synthesis with Response Surfaces," *Proceedings of the 40th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, St. Louis, MO, USA: AIAA, 1999.

[77] Dapeng Wang, G. Gary Wang, and Greg F. Naterer, "Collaboration pursuing method for MDO problems," *Proceedings of the 1st AIAA Multidisciplinary Design Optimization Specialist Conference*, Austin, TX, USA: AIAA, 2005.

[78] Dapeng Wang, G. Gary Wang, and Greg F. Naterer, "Advancement of a Collaboration Pursuing Method (CPM)," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, Renu, NV, USA: AIAA, 2006.

[79] Dapeng Wang, G. Gary Wang, and Greg. F. Naterer, "Collaboration Pursuing Method for Multidisciplinary Design Optimization Problems," *AIAA Journal*, vol. 45, May 2007.

[80] Dapeng Wang, G. Gary Wang, and Greg F. Naterer, "Extended Collaboration Pursuing Method for Solving Larger Multidisciplinary Design Optimization Problems," *AIAA Journal*, vol. 45, Jun. 2007.

[81] Xiaoyu Gu, and John E. Renaud, "Implicit uncertainty propogation for robust collaborative optimization," *Proceedigs of DETC'01*, Pittsburgh, PA, USA: ASME, 2001.

[82] Harrison M. Kim, Wei Chan, and Margaret M. Wiecek, "Lagrangian Coordination for Enhancing the Convergence of Analytical Target Cascading," *AIAA Journal*, vol. 44, Oct. 2006, pp. 2197–2207.

83 N.M. Alexandrov, and S. Kodiyalam, “Initial Results of an MDO Method Evaluation Study,” *Proceedings of the Seventh AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, USA: AIAA, 1998.

84 Natalia M. Alexandrov, and Robert Michael Lewis, *Comparative properties of collaborative optimization and other approaches to MDO*, ICASE, 1999.

85 Kamran Behdinan, Ruben E. Perez, and Hugh T. Liu, “Multidisciplinary design optimization of aerospace systems,” *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY, USA: AIAA, 2004.

86 Srinivas Kodiyalam, *Evaluation of Methods for Multidisciplinary Design Optimization (MDO), Phase I*, NASA, 1998.

87 Srinivas Kodiyalam, and Charles Yuan, *Evaluation of Methods for Multidisciplinary Design Optimization (MDO), Part II*, NASA, 2000.

88 R.J. Balling, and J. Sobieszczanski-Sobieski, *Optimization of coupled systems: a critical overview of approaches*, 1994.

89 Nathan Tedford, “Comparison of MDO Architectures within a Universal Framework,” Masters of Applied Science, University of Toronto, 2006.

90 Andrea Salas, “Framework Activities at NASA LaRC under the HPCC Program,” Aug. 1998.

91 Krammer, J., Sensburg, O., Vilsmeier, J., and Berchtold, G., “Concurrent Engineering in Design of Aircraft Structures,” *Journal of Aircraft*, vol. 32, Apr. 1995.

92 Srinivas Kodiyalam, and Jaroslaw Sobieszczanski-Sobieski, “Multidisciplinary design optimization - some formal methods, framework requirements, and application to vehicle design,” *International Journal of Vehicle Design*, vol. 25, 2001, pp. 3–32.

93 H.G. Hoenlinger, J. Krammer, and M. Stettner, “MDO technology needs in aeroelastic structural design,” *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, USA: AIAA, 1998.

94 Masoud Rais-Rohani, *A Framework for Preliminary Design of Aircraft Structures Based on Process Information*, Starkville, MS, USA: Mississippi State University, 1998.

95 James C. Townsend, Andrea O. Salas, and M. Patricia Schuler, *Configuration Management of an Optimization Application in a Research Environment*, Hampton, VA, USA: NASA Langley Research Center, 1999.

96 Chapman, B., Mehrotra, P., Rosendale, J. V., and Zima, H., *A Software Architecture for Multidisciplinary Applications: Integrating Task and Data Parallelism*, 1994.

97 Vogels, M. E. S., Arendsen, P., Krol, R. J., Laban, M., and Pruis, G. W., *From a mono-disciplinary to a multi-disciplinary approach in aerospace: as seen from information and communication technology perspective*, Nationaal Lucht- en Ruimtevaartlaboratorium (National Aerospace Laboratory), 1998.

98 Raj Sistla, Gus Dovi, and Phillip Su, "A Distributed, Heterogenous Computing Environment for Multidisciplinary Design & Analysis of Aerospace Vehicles," Oct. 1999.

99 J.C. Townsend, J.A. Samareh, R.P. Weston, and W.E. Zorumski, *Integration of a CAD System Into an MDO Framework*, Hampton, VA, USA: NASA Langley Research Center, 1998.

100 R.P. Weston, J.C. Townsend, T.M. Eidson, and R.L. Gates, "A Distributed Computing Environment for Multidisciplinary Design," 1994.

101 E.H. Winer, M.K. Abdul-Jalil, and C.L. Bloebaum, "Development of a geographic independent virtual design environment for large-scale design," *AIAA*, 1998.

102 Stephen T. LeDoux, William W. Herling, Gasper J. Fatta, and Robert R. Ratcliff, "MDOPT - A Multidisciplinary Design Optimization System Using Higher Order Analysis Codes," *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY, USA: 2004.

103 S.L. Padula, J.J. Korte, H.J. Dunn, and A.O. Salas, *Multidisciplinary Optimization Branch Experience Using iSIGHT Software*, Hampton, VA, USA: NASA Langley Research Center, 1999.

104 Phoenix Integration, "Phoenix Integration - ModelCenter 9.0 - General," *Phoenix Integration - ModelCenter 9.0 - General* Available: http://www.phoenix-int.com/software/phx_modelcenter.php.

105 SIMULIA, "SIMULIA > Products > iSIGHT," *SIMULIA > Products > iSIGHT* Available: http://www.simulia.com/products/isight.html.

106 "Esteco - Leader in engineering design optimization software, process integration and multidisciplinary optimization with modeFRONTIER" Available: http://www.esteco.com/.

107 Altair, "Altair HyperStudy - Optimization of Design Performance & Robustness," *Altair* Available: http://www.altairhyperworks.com/Product,10,HyperStudy.aspx.

108 Sigma Technology, "'Sigma Technology'. Novel Optimization Strategy - IOSO," *"Sigma Technology". Novel Optimization Strategy - IOSO* Available: http://www.iosotech.com/.

109 Audet, C., "A surrogate-model-based method for contrained optimization," Long Beach, CA: 2000.

110 Booker, A. J., Jr., J. E. D., Frank, P. ., Serafini, D. B., Torezon, V., and Trosset, M. W., "A rigorous framework for optimization of expensive functions by surrogates," *Structural Optimization*, vol. 17, 1999, pp. 1–13.

111 J. Olds, "The suitability of selected multidisciplinary design and optimization techniques to conceptual aerospace vehicle design," *Proceedings of the 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, Cleveland, OH: AIAA, 1992.

112 Simpson, T. W., Mauery, T. M., Korte, J. J., and Mistree, F., "Comparison of response surface and kriging models for multidisciplinary design optimization," *AIAA paper 98*, vol. 4758, 1998.

113 Tenne, Y., and Armfield, S. W., "A framework for memetic optimization using variable global and local surrogate models," *Soft Computing*, vol. 13, 2009, pp. 781–793.

114 Berkooz, G., Holmes, P., and Lumley, J. L., "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual review of fluid mechanics*, vol. 25, 1993, pp. 539–575.

115 LeGresley, "Application of proper orthogonal decomposition (POD) to design decomposition methods.," Dissertation, Stanford University, 2005.

116 LeGresley, P. A., and Alonso, J. J., "Investigation of non-linear projection for POD based reduced order models for aerodynamics," *AIAA paper*, vol. 926, 2001, p. 2001.

117 Or, A. C., Speyer, J. L., and Kim, J., "Reduced Balancing Transformations for Large Nonnormal State-Space Systems," *Journal of Guidance, Control, and Dynamics*, vol. 35, Jan. 2012, pp. 129–137.

118 Chaturantabut, and Sorensen, *Discrete empirical interpolation for nonlinear model reduction*, Rice University, 2009.

119 Manyu Xiao, Piotr Breitkopf, Rajan Filomeno Coelho, Catherine Knopf-Lenoir, Maryan Sidorkiewicz, and Pierre Villon, “Model reduction by CPOD and Kriging,” *Structural and Multidisciplinary Optimization*, vol. 41, Oct. 2009, pp. 555–574.

120 Xiao, M., Breitkopf, P., Filomeno Coelho, R., Knopf-Lenoir, C., and Villon, P., “Enhanced POD projection basis with application to shape optimization of car engine intake port,” *Structural and Multidisciplinary Optimization*, vol. 46, Jan. 2012, pp. 129–136.

121 Carlberg, K., and Farhat, C., “A low-cost, goal-oriented ‘compact proper orthogonal decomposition’basis for model reduction of static systems,” *International Journal for Numerical Methods in Engineering*, vol. 86, 2011, pp. 381–402.

122 Hay, A., Borggaard, J., Akhtar, I., and Pelletier, D., “Reduced-order models for parameter dependent geometries based on shape sensitivity analysis,” *Journal of Computational Physics*, vol. 229, Feb. 2010, pp. 1327–1352.

123 S. Volkwein, “Model reduction using proper orthogonal decomposition.”

124 Newman, A. J., “Model reduction via the Karhunen-Loeve expansion Part I: An exposition,” 1996.

125 Newman, A. J., “Model reduction via the Karhunen-Loeve expansion Part II: Some elementary examples,” 1996.

126 Silva, “Application of nonlinear systems theory to transsonic unsteady aerodynamic responses,” *Journal of Aircraft*, vol. 30, 1993, pp. 660–668.

127 Silva, “Extension of a nonlinear systems theory to general-frequency unsteady transonic aerodynamic responses,” La Jolla, CA, USA: 1993.

128 Silva, “Discrete-time linear and nonlinear aerodynamic impulse responses for efficient CFD analyses,” Dissertation, 1997.

129 Silva, W. A., *Identification of linear and nonlinear aerodynamic impulse responses using digital filter techniques*, Citeseer, 1997.

130 Walter A. Silva, “Reduced order models based on linear and nonlinear aerodynamic impulse responses,” *Proceedings of the AIAA/ASE/ASCE/AHS/ASC 40th Structures, Structural Dynamics, and Materials Conference*, St. Louis, MO, USA: AIAA, 1999, p. 12.

131 Silva, W. A., and Bartels, R. E., “Development of reduced-order models for aeroelastic analysis and flutter prediction using the CFL3Dv6. 0 code,” *Journal of fluids and structures*, vol. 19, 2004, pp. 729–745.

[132] Walter A. Silva, “Recent Enhancements to the Development of CFD-Based Aeroelastic Reduced-Order Models,” *Proceedings of the 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, Hawaii: AIAA, 2007.

[133] Silva, W. A., “Simultaneous Excitation of Multiple-Input/Multiple-Output CFD-Based Unsteady Aerodynamic Systems,” *Journal of Aircraft*, vol. 45, Jul. 2008, pp. 1267–1274.

[134] Falkowski, B. J., and Sasao, T., “Unified algorithm to generate Walsh functions in four different orderings and its programmable hardware implementations,” *IEE Proceedings - Vision, Image, and Signal Processing*, vol. 152, 2005, p. 819.

[135] Adam Jirasek, and Russell M. Cummings, “Application of Volterra functions to X-31 aircraft model motion,” San Antonio, TX, USA: 2009.

[136] J. Leo Van Hemmen, Werner M. Kistler, and Erik G. F. Thomas, “Calculation of Volterra kernels for solutions of nonlinear differential equations,” *Society for Industrial and Applied Mathematics*, vol. 61, 2000, pp. 1–21.

[137] Vandendorpe, A., “Model reduction of linear systems, an interpolation point of view,” *PhD thesis*, 2004.

[138] Lieu, T., Farhat, C., and Lesoinne, M., “Reduced-order fluid/structure modeling of a complete aircraft configuration,” *Computer Methods in Applied Mechanics and Engineering*, vol. 195, Aug. 2006, pp. 5730–5742.

[139] Naets, F., Heirman, G., and Desmet, W., “Reduced-order-model interpolation for use in global modal parameterization,” *Proceedings of the International Conference on Noise and Vibration Engineering, ISMA 2010*, 2010, pp. 3021–3032.

[140] Fabio Vetrano, Christophe Le Garrec, Guy D.Mortchelewicz, and Roger Ohayon, “Assessment of dtrategies for interpolating POD based reduced order model and application to aeroelasticity,” vol. 2, 2012, pp. 85–104.

[141] David Amsallem, and Charbel Farhat, “An interpolation method for adapting reduced-order models and application to aeroelasticity,” *AIAA Journal*, vol. 46, Jul. 2008, pp. 1803–1813.

[142] Amsallem, D., Farhat, C., Cortial, J., and Carlberg, K. T., “A class of high-order and multivariate interpolation methods for adapting recuded-order models to continuous parameter changes,” Venice, Italy: 2008.

[143] Amsallem, D., Cortial, J., Carlberg, K., and Farhat, C., "A method for interpolating on manifolds structural dynamics reduced-order models," *International Journal for Numerical Methods in Engineering*, vol. 80, 2009, pp. 1241–1258.

[144] Amsallem, D., Cortial, J., and Farhat, C., "Towards real-time computational-fluid-dynamics-based aeroelastic computations using a database of reduced-order information.," *AIAA Journal*, vol. 48, 2010, pp. 2029–2037.

[145] Joris Degroote, Jan Vierendeels, and Karen Willcox, "Interpolation among reduced-order matrices to obtain parameterized models for design, optimization and probabilistic analysis," *International Journal for Numerical Methods in Fluids*, vol. 63, 2010, pp. 207–230.

[146] Michael Mifsud, "Reduced-order modelling for high-speed aerial weapon aerodynamics," Doctoral Thesis, Cranfield University, 2008.

[147] Rajan Filomeno Coelho, Piotr Breitkopf, Catherine Knopf-Lenoir, and Pierre Villon, "Bi-level model reduction for coupled problems," *Structural and Multidisciplinary Optimization*, vol. 39, Nov. 2008, pp. 401–418.

[148] Weickum, G., Eldred, M. S., and Maute, K., "A multi-point reduced-order modeling approach of transient structural dynamics with application to robust design optimization," *Structural and Multidisciplinary Optimization*, vol. 38, Sep. 2008, pp. 599–611.

[149] Dewey H Hodges, and G. Alvin Pierce, *Introduction to Structural Dynamics and Aeroelasticity*, 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2002.

[150] Raphael T. Haftka, and Zafer Gurdal, *Elements of Structural Optimization*, Kluwer Academic Publishers, 1992.

[151] Garret N. Vanderplaats, *Multidiscipline Design Optimization*, Vanderplaats Research and Development, Inc, 2007.

[152] Sobieszczanski-Sobieski, J., "Sensitivity of Complex, Internally Coupled Systems," *AIAA Journal*, vol. 28, Jan. 1990, pp. 153–160.

[153] Richard L. Burden, and J. Douglas Faires, *Numerical Analysis*, Thomson Brooks/Cole, 2005.

[154] Charles Hirsch, *Numerical Computation of Internal and External Flows: Fundamentals of Computational Fluid Dynamics*, Butterworth-Heinemann, Elsevier, 2007.

155 Gilbert Strang, "MIT OpenCourseWare | Mathematics | 18.06 Linear Algebra, Spring 2010 | Video Lectures | Lecture 29: Singular value decomposition," 1999.

156 Lieu, "Adaption of reduced order models for applications in aeroelasticity," Dissertation, University of Colorodo at Boulder, 2004.

157 Garret N. Vanderplaats, and Raymond M. Hicks, "Numerical airfoil optimization using a reduced number of design coordinates," Jul. 1976.

158 John Garcelon, Vladimir Balabanov, and Jaroslaw Sobieski, "Multidisciplinary optimization of a transport aircraft wing using visualDOC," *40th Structures, Structural Dynamics, and Materials Conference and Exhibit*, American Institute of Aeronautics and Astronautics, 1999.

159 Gerhard Venter, and Jaroslaw Sobieszczanski-Sobieski, "Multidisciplinary Optimization of a Transport Aircraft Wing Using Particle Swarm Optimization," *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, 2002.

160 Venter, G., and Sobieszczanski-Sobieski, J., "Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations," *Journal of Aerospace Computing, Information, and Communication*, vol. 3, Mar. 2006, pp. 123–137.

161 E. F. Bruhn, *Analysis and Design of Flight Vehicle Structures*, 9135 N. Meridian Street, Suite B6, Indianapolis, IN 40260: S.R. Jacobs and Associates, Inc., 1973.

APPENDIX A

DERIVATION OF ANALYTICAL DERIVATIVE TO RBF-BASED OBJECTIVE APPROXIMATION

The objective to the ASO problem is given as:

$$\mathrm{F}(\bar{\mathrm{X}}) = \frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} c_L^j / c_D^j \tag{A.1}$$

Note that c_L^j and c_D^j are results from a complex analysis, $\boldsymbol{Y}(X)$, covering $j = 1{:}\, n_\alpha$ angles of attack. We store the analysis results in a matrix, $\boldsymbol{Y}$, such that each column represents results for an angle of attack, and the rows correspond to c_L, $c_D$and c_M. That is:

$$\boldsymbol{Y} = \begin{bmatrix} c_L^1 & c_L^2 & \cdots & c_L^{n_\alpha} \\ c_D^1 & c_D^2 & \cdots & c_D^{n_\alpha} \\ c_M^1 & c_M^2 & \cdots & c_M^{n_\alpha} \end{bmatrix} \tag{A.2}$$

This is just to say that we can refer to c_L^j as element $y^{(1j)}$ and c_D^j as element $y^{(2j)}$. This will avoid confusion between the aerodynamic coefficients and the RBF shape parameter c, as well as reflecting a more general formulation. Thus we can write the objective as:

$$\mathrm{F}(\bar{\mathrm{X}}) = \frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} y^{(1j)} / y^{(2j)} \tag{A.3}$$

To approximate F, we use RBF models to approximate each element of $\boldsymbol{Y}$, thus:

$$\tilde{\mathrm{F}}(\bar{\mathrm{X}}) = \frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} \tilde{y}^{(1j)} / \tilde{y}^{(2j)} \tag{A.4}$$

The approximation for $y^{(1j)}$, and similarly for $y^{(2j)}$, is given as:

$$y^{(1j)} = \sum_{i=1}^{n_p} \sigma_i^{(1j)} \varphi^{(1j)}(r_i^{(1j)}(X)) \tag{A.5}$$

Here, $\sigma_i^{(1j)}$ are the model coefficients, and n_p is the number of sample points used to construct the model, which in this case is identical for all the ROMs. $\varphi^{(1j)}$ and $r_i^{(1j)}$ are given by:

$$\varphi^{(1j)} \in \begin{cases} \varphi_{FLIN}(r) \equiv cr \\ \varphi_{MQ}(r) \equiv \sqrt{1+cr^2} \\ \varphi_{IMQ}(r) \equiv 1/\sqrt{1+cr^2} \end{cases} \tag{A.6}$$

$$r_i^{(1j)}(X) = \sqrt{\sum_{k=1}^{m}\left(X_k - X_k^i\right)^2} \tag{A.7}$$

Where $m = \dim(X)$. We will let ξ_k^i denote $X_k - X_k^i$ for simplicity of notation. We can derive (4) with respect to X_d as:

$$\frac{\delta \tilde{F}}{\delta X_d} = \frac{\delta}{\delta X_d}\left[-\frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} \tilde{y}^{(1j)}(X) \cdot (\tilde{y}^{(2j)}(X))^{-1}\right] \tag{A.8}$$

Which, by the product rule:

$$\frac{\delta \tilde{F}}{\delta X_d} = -\frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} \frac{\delta \tilde{y}^{(1j)}}{\delta X_d}\frac{1}{(\tilde{y}^{(2j)}(\bar{X}))} + \frac{\delta \tilde{y}^{(2j)}}{\delta X_d}\left(\frac{\tilde{y}^{(1j)}(\bar{X})}{(\widetilde{y}^{(2j)}(\bar{X}))^2}\right) \tag{A.9}$$

The first partial is given by:

$$\frac{\delta \tilde{y}^{(1j)}}{\delta X_d} = \frac{\delta}{\delta X_d}\sum_{i=1}^{n_p} \sigma_i^{(1j)} \varphi^{(1j)}(r_i^{(1j)}(X)) \tag{A.10}$$

Which by the chain rule gives:

$$\frac{\delta \tilde{y}^{(1j)}}{\delta X_d} = \sum_{i=1}^{n_p} \sigma_i^{(1j)} \frac{\delta \varphi^{(1j)}}{\delta r}\frac{\delta r_i^{(1j)}}{\delta X_d} \tag{A.11}$$

The second partial, $\frac{\delta \tilde{y}^{(2j)}}{\delta X_d}$, is given similarly to (10). The latter partial is straightforward, given by:

$$\frac{\delta r_i}{\delta X_d} = \frac{\delta}{\delta X_d}\sqrt{\sum_{\substack{k=1 \\ k\neq d}}^{m}\left(\xi_k^i\right)^2 + \left(\xi_d^i\right)^2} = \frac{\xi_d^i}{\sqrt{\sum_{\substack{k=1 \\ k\neq d}}^{m}\left(\xi_k^i\right)^2 + \left(\xi_d^i\right)^2}} \tag{A.12}$$

Where we are separating the terms held constant during the partial derivation for clarity. The first partial of the right hand side of (10) may take different forms depending on which kernel function was selected as the best fit for the particular ROM, and this may not match between different elements of $\boldsymbol{Y}$. For each included kernel function, the relevant partials are given by:

$$\frac{\delta \varphi_{FLIN}}{\delta r} = c \tag{A.13}$$

$$\frac{\delta \varphi_{MQ}}{\delta r} = \frac{cr}{\sqrt{1+cr^2}} \tag{A.14}$$

$$\frac{\delta \varphi_{IMQ}}{\delta r} = -\frac{cr}{(1+cr^2)^{3/2}} \tag{A.15}$$

Putting this all together, the overall sensitivity is given by:

$$\frac{\delta \tilde{F}}{\delta X_d} = -\frac{1}{n_\alpha}\sum_{j=1}^{n_\alpha} \left\lfloor \begin{array}{c} \left(\frac{1}{\left(\tilde{y}^{(2j)}(\bar{X})\right)}\right)\sum_{i=1}^{n_p}\left[\sigma_i^{(1j)}\left(\frac{\delta\varphi^{(1j)}}{\delta r}\right)\left(\frac{\xi_d^i}{\sqrt{\sum_{k=1}^{m}\left(\xi_k^i\right)^2}}\right)\right] + \\ \left(\frac{\tilde{y}^{(1j)}(\bar{X})}{(\widetilde{y}^{(2j)}(\bar{X}))^2}\right)\sum_{i=1}^{n_p}\left[\sigma_i^{(2j)}\left(\frac{\delta\varphi^{(2j)}}{\delta r}\right)\left(\frac{\xi_d^i}{\sqrt{\sum_{k=1}^{m}\left(\xi_k^i\right)^2}}\right)\right] \end{array} \right\rfloor \tag{A.16}$$

This analytical derivative is evaluated for the ASO problem by using the following MATLAB code:

```
% Analytical Derivative for ASO Objective using IROM models
function [dF] = drbf(IROM,X)
load param.mat Param
dF = zeros(1,size(IROM.ROMC(1,1).ROM.X,2));
for d=1:size(IROM.ROMC(1,1).ROM.X,2) %m
    sns = 0;
    for j=1:size(IROM.ROMC,2) %ns
        snp = 0;
        y1ja = eval_rom(IROM.ROMC(1,j).ROM,X);
        y2ja = eval_rom(IROM.ROMC(2,j).ROM,X);
        c1j = IROM.ROMC(1,j).ROM.c;
        c2j = IROM.ROMC(2,j).ROM.c;
        for i=1:size(IROM.ROMC(1,j).ROM.X,1) %np
            s1ji = IROM.ROMC(1,j).ROM.sig(i);
            s2ji = IROM.ROMC(2,j).ROM.sig(i);
            r1 = norm(X-IROM.ROMC(1,j).ROM.X(i,:),2);
            r2 = norm(X-IROM.ROMC(2,j).ROM.X(i,:),2);
            rd1 = X(d)-IROM.ROMC(1,j).ROM.X(i,d);
            rd2 = X(d)-IROM.ROMC(2,j).ROM.X(i,d);
            % Adjust dphi/dr for the ROM-specific kernel
            if(strcmp(IROM.ROMC(1,j).ROM.phiname, 'FLIN')==1)
                d1 = c1j;
            elseif(strcmp(IROM.ROMC(1,j).ROM.phiname, 'MQ')==1)
                d1 = (c1j*r1)/sqrt(1+c1j*r1*r1);
            elseif(strcmp(IROM.ROMC(1,j).ROM.phiname, 'IMQ')==1)
                d1 = (-c1j*r1)/((1+c1j*r1*r1)^1.5);
            else
                error('ROM(1,%d) kernel function is not recognized',j);
            end
            if(strcmp(IROM.ROMC(2,j).ROM.phiname, 'FLIN')==1)
                d2 = c2j;
            elseif(strcmp(IROM.ROMC(2,j).ROM.phiname, 'MQ')==1)
                d2 = (c2j*r2)/sqrt(1+c2j*r2*r2);
            elseif(strcmp(IROM.ROMC(2,j).ROM.phiname, 'IMQ')==1)
                d2 = (-c2j*r2)/((1+c2j*r2*r2)^1.5);
            else
                error('ROM(2,%d) kernel function is not recognized',j);
            end
            xd1 = X(d)*ones(size(IROM.ROMC(1,j).ROM.X,1),1);
            xd2 = X(d)*ones(size(IROM.ROMC(2,j).ROM.X,1),1);
            d3 = rd1/sqrt(dot(xd1-IROM.ROMC(1,j).ROM.X(:,d),xd1-
IROM.ROMC(1,j).ROM.X(:,d)));
            d4 = rd2/sqrt(dot(xd2-IROM.ROMC(2,j).ROM.X(:,d),xd2-
IROM.ROMC(2,j).ROM.X(:,d)));
            snp = snp + (1/y2ja)*(s1ji)*(d1)*(d3) + (-
y1ja/(y2ja*y2ja))*(s2ji)*(d2)*(d4);
        end %i
        sns = sns + snp;
    end %j
    dF(d) = (-1/size(IROM.ROMC,2))*sns;
end %d
```

APPENDIX B

POD BASED ROM (MATLAB CODE)

B.1 Construction

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Construct a POD-based ROM object
% Jeff Parrish
% Mississipi State University
%
% Standard POD formulation with options for biasing and normalizing
inputs
% of the system of interest.  Outputs are always normalized.  Flags are
0 or 1.
%
% Inputs:
%   X is row-ordered parameter vectors of snapshots
%   Y is column-ordered snapshots
%   k is the POD model order
%   P is the global parameter structure, optionally passed as a direct
copy
%
% Outputs:
%   ROM is a structure with the following fields:
%     ROM.type      - Text field denoting ROM type, 'POD' for this
%     ROM.X         - Parameters (possibly biased/normalized)
%     ROM.basis     - Basis vectors (columnwise, k cols)
%     ROM.C         - Coordinates of snapshots, snapshot i in row i,
basis vector j in col j
%     ROM.k         - Model Order (as constructed, not requested)
%     ROM.RBF       - RBF ROM for interpolating coordinates by
parameter
%     ROM.xbias     - X-bias vector, if any
%     ROM.xscale    - X-scaling term, if any
%     ROM.ybias     - Y-bias vector, if any
%     ROM.yscale    - Y-scaling term, if any
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [ROM] = build_pod(X,Y,k,P)
global Param;
if(nargin == 4)
    Param = P;
end
% Check for problems
if(size(X,1) ~= size(Y,2))
    error('build_pod(): Row count of X must match column count of Y.')
end

% vprint(sprintf('
build_pod(X[%dx%d],Y[%d,%d],%d)',size(X),size(Y),k));

% Peform Biasing/Normalization
% Note that biasing is performed prior to normalization
% Thus to reconstruct a value, first scale and then add the bias

% Bias/Norm Defaults:
```

```
xbias=zeros(1,size(X,2));
xscale=1;
ybias=zeros(size(Y,1),1);
yscale=1;
% Bias:
if(Param.podinbias == 1)
    xbias = mean(X);            % mean of each column (dim) of X, as
row vector -> average coordinate vector
    for i=1:size(X,1)           % bias each parameter row to center on
xbias average
        X(i,:) = X(i,:)-xbias;
    end
end
% Normalize:
if(Param.podinnorm == 1)
    xscale = norm(X(1,:),2);                % Find scale
    for i=2:size(X,1)
        if(norm(X(i,:),2) > xscale)
            xscale = norm(X(i,:),2);
        end
    end
    for i=1:size(X,1)                       % Scale each parameter
        X(i,:) = X(i,:)/xscale;
    end
end
% Bias/Normalized Snapshots
if(Param.podoutbias == 1)
    ybias = mean(Y,2);          % mean of each row (dim) of Y, as
column vector -> average snapshot vector
    for i=1:size(Y,2)           % bias each snapshot column to center
on ybias average
        Y(:,i) = Y(:,i)-ybias;
    end
end
if(Param.podoutnorm == 1)
    yscale = norm(Y(:,1),2);                % Find scale
    for i=2:size(Y,2)
        if(norm(Y(:,i),2) > yscale)
            yscale = norm(Y(:,i),2);
        end
    end
    for i=1:size(Y,2)                   % Scale each snapshot
        Y(:,i) = Y(:,i)/yscale;
    end
end

% Perform SVD
[U,S,V] = svd(Y);

% Basis Vectors
if(k<=0)
    mk = size(U,2);
else
    mk = min(k,size(U,2));
end
```

```
if(mk<k)
    warning('build_pod(): Requested model order k=%d is greater than
maximum model dimensionality %d; reducing model order.',k,mk);
end
basis = U(:,1:mk);
% Normalize the basis vectors
for i=1:mk
    basis(:,i) = basis(:,i)/norm(basis(:,i),2);
end

% Project snapshots onto basis vectors, get coordinates
for si=1:size(Y,2)
    for bi=1:mk
        C(si,bi) = dot(basis(:,bi), Y(:,si)');
    end
end
size(C);

% Construct RBF model for parameter/coordinate interpolation
ERBF = build_rbf(X, C');

% Pack ROM
ROM.type = 'POD';
ROM.X = X;
ROM.basis = basis;
ROM.C = C;
ROM.k = mk;
ROM.RBF = ERBF;
ROM.xbias = xbias;
ROM.xscale = xscale;
ROM.ybias = ybias;
ROM.yscale = yscale;

end
```

B.2 Evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Evaluate POD-based ROM
% Jeff Parrish
% Mississipi State University
%
% Evaluates ROMs constructed with build_pod().  X is a row-wise
ordering of
% evaluation points; to evaluate multiple points, place one parameter
% vector per row.
%
% Inputs:
%   ROM - ROM Model built with build_pod();
%   X - Parameter vectors to evaluate, row ordered.  Multiple rows will
be
%   evaluated separately.  That is, X is [nsamp x ndim].
%   P - Global Parameter structure, optionally passed as direct copy
%
% Outputs:
%   Y - ROM Outputs, column wise - col Y_i corresponds to row X_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [Y] = eval_pod(ROM,X,P)
global Param;
if(nargin == 3 && isempty(Param))
    Param = P;
end
% Check for problems
if(strcmp(ROM.type,'POD') ~= 1)
    error('eval_pod(): ROM is not a standard POD-based model.')
end

% vprint(sprintf('     eval_pod(ROM,X[%d,%d]))',size(X)));

% If ROM biased/normalized parameters, do the same to the new X
Xset = X;
for i=1:size(X,1)
    if(Param.podinbias)
        Xset(i,:) = X(i,:)-ROM.xbias;
    end
    if(Param.podinnorm)
        Xset(i,:) = X(i,:)/ROM.xscale;
    end
end

Y = zeros(length(ROM.ybias),size(X,1));
for i=1:size(X,1)
% For each new evaluation point
    % Construct (possibly) biased/normalized output vector
    Xbn = Xset(i,:);
    Ybn = zeros(length(ROM.ybias),1);
    Cbn = eval_rom(ROM.RBF,Xbn);
% Interpolate coordinates, store as row vector
    % Unbias/unnormalize
      Y(:,i) = ROM.basis*Cbn;
      % If ROM biased/normalized parameters, do the same to the new X
      if(Param.podoutnorm)
            Y(:,i) = Y(:,i)*ROM.yscale;
      end
      if(Param.podoutbias)
            Y(:,i) = Y(:,i)+ROM.ybias;
      end
end

end
```

APPENDIX C

RBF BASED ROM (MATLAB CODE)

C.1 Construction

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Construct a RBF-based ROM object
% Jeff Parrish
% Mississipi State University
%
% Standard RBF formulation with options for biasing and normalizing
inputs
% and outputs of the system of interest.  Flags are 0 or 1.
%
% Inputs:
%   X is row-ordered parameter vectors of snapshots
%   Y is column-ordered snapshots
%   P - Global Parameter structure, optionally passed as direct copy
%
% Outputs:
%   ROM is a structure with the following fields:
%     ROM.type      - Text field denoting ROM type, 'RBF' for this
%     ROM.sig       - Weighting Matrix
%     ROM.X         - Parameters (possibly biased/normalized)
%     ROM.phi       - RBF
%     ROM.xbias     - X-bias vector, if any
%     ROM.xscale    - X-scaling term, if any
%     ROM.ybias     - Y-bias vector, if any
%     ROM.yscale    - Y-scaling term, if any
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [ROM] = build_rbf(X,Y,P)
global Param;
if(nargin == 3)
    Param = P;
end
% tic
% vprint(sprintf('
builf_rbf(X[%dx%d],Y[%d,%d]))',size(X),size(Y)));
useparfor=0;

% Check for problems
if(size(X,1) ~= size(Y,2))
    error('build_rbf(): Row count of X must match column count of Y.')
end

% Peform Biasing/Normalization
% Note that biasing is performed prior to normalization
% Thus to reconstruct a value, first scale and then add the bias

% Bias/Norm Defaults:
% if(matlabpool('size') <= 1)
%     vprint(sprintf('\t\tRBF: Bias/Norm...'));
% end
xbias=zeros(1,size(X,2));
xscale=1;
```

```
ybias=zeros(size(Y,1),1);
yscale=1;
% Bias:
if(Param.rbfinbias == 1)
    xbias = mean(X);            % mean of each column (dim) of X, as
row vector -> average coordinate vector
    for i=1:size(X,1)           % bias each parameter row to center on
xbias average
        X(i,:) = X(i,:)-xbias;
    end
end
if(Param.rbfoutbias == 1)
    ybias = mean(Y,2);          % mean of each row (dim) of Y, as
column vector -> averaege snapshot vector
    for i=1:size(Y,2)           % bias each snapshot column to center
on ybias average
        Y(:,i) = Y(:,i)-ybias;
    end
end
% Normalize:
if(Param.rbfinnorm == 1)
    xscale = norm(X(1,:),2);                % Find scale
    for i=2:size(X,1)
        if(norm(X(i,:),2) > xscale)
            xscale = norm(X(i,:),2);
        end
    end
    for i=1:size(X,1)                       % Scale each parameter
        X(i,:) = X(i,:)/xscale;
    end
end
if(Param.rbfoutnorm == 1)
    yscale = norm(Y(:,1),2);                % Find scale
    for i=2:size(Y,2)
        if(norm(Y(:,i),2) > yscale)
            yscale = norm(Y(:,i),2);
        end
    end
    for i=1:size(Y,2)                   % Scale each snapshot
        Y(:,i) = Y(:,i)/yscale;
    end
end

% Search kernel functions for best fit
bestphi = Param.phirbf(1).phi;
bestc = Param.rbf_crng(1);
besterr = 10;
Prm = Param;
if(size(X,1) > Param.rbfmaxcross)
    cvi(:,1) = randperm(size(X,1));
    cvi(Param.erbfmaxcross+1:end) = [];
else
    cvi = 1:size(X,1);
end
err = ones(length(cvi),1)*10000;
```

```
for p = 1:length(Param.phirbf)
    phi = Param.phirbf(p).phi;
    % Search c-calues to fit the model
    for c = Param.rbf_crng
        % Perform cross validation to gauge model fit
%         if(matlabpool('size')<=1)
%             vprint(sprintf('RBF: Performing Cross Validation for phi
%d/%d and c=%0.4f (max %0.4f)',p,length(Param.phirbf),c,
Param.rbf_crng(end)));
%             vprint(sprintf('RBF: Cross Validation will be performed
for %d samples',length(cvi)));
%         end
        ROMcv.type = 'RBF';
        ROMcv.sig = zeros(size(X,1)-1,size(Y,2));  % nsamp x ydim
        ROMcv.c = 0;
        ROMcv.X = zeros(size(X)-[1 1]);
        ROMcv.phi = phi;
        ROMcv.xbias=zeros(1,size(X,2));
        ROMcv.xscale=1;
        ROMcv.ybias=zeros(size(Y,1),1);
        ROMcv.yscale=1;
        err = 10000*ones(length(cvi));
        rbfib = Param.rbfinbias;
        rbfin = Param.rbfinnorm;
        rbfob = Param.rbfoutbias;
        rbfon = Param.rbfoutnorm;
        nw = matlabpool('size');
        if(useparfor)
            top=tic;
            parfor op = 1:length(cvi)
                Rcv = ROMcv;
%                 if(nw<=1)
%                     fprintf(1,'.');
%                 end
                warning('off','all');
                opi = cvi(op);
                % Omit the opi-th point
                Xcv = X;
                Xcv(opi,:) = [];
                Ycv = Y;
                Ycv(:,opi) = [];
                % bias and scaling will change depending on omitted
points!
                % Bias:
                if(rbfib == 1)
                    Rcv.xbias = mean(Xcv);            % mean of each
column (dim) of X, as row vector -> average coordinate vector
                    for i=1:size(Xcv,1)           % bias each parameter
row to center on xbias average
                        Xcv(i,:) = Xcv(i,:)-Rcv.xbias;
                    end
                end
                if(rbfob == 1)
                    Rcv.ybias = mean(Ycv,2);          % mean of each
row (dim) of Y, as column vector -> averaege snapshot vector
```

```
                for i=1:size(Ycv,2)          % bias each snapshot
column to center on ybias average
                    Ycv(:,i) = Ycv(:,i)-Rcv.ybias;
                end
            end
            % Normalize:
            if(rbfin == 1)
                Rcv.xscale = norm(Xcv(1,:),2);                %
Find scale
                for i=2:size(Xcv,1)
                    if(norm(Xcv(i,:),2) > Rcv.xscale)
                        Rcv.xscale = norm(Xcv(i,:),2);
                    end
                end
                for i=1:size(Xcv,1)                        % Scale
each parameter
                    Xcv(i,:) = Xcv(i,:)/Rcv.xscale;
                end
            end
            if(rbfon == 1)
                Rcv.yscale = norm(Ycv(:,1),2);                %
Find scale
                for i=2:size(Ycv,2)
                    if(norm(Ycv(:,i),2) > Rcv.yscale)
                        Rcv.yscale = norm(Ycv(:,i),2);
                    end
                end
                for i=1:size(Ycv,2)                    % Scale each
snapshot
                    Ycv(:,i) = Ycv(:,i)/Rcv.yscale;
                end
            end
            % Build distance weighting matrix
            A = zeros(size(Xcv,1));
            for i=1:size(Xcv,1)
                for j=(i):size(Xcv,1)
                    A(i,j) = phi(norm(Xcv(j,:)-Xcv(i,:)), c);
                    A(j,i) = A(i,j);
                end
            end
            % Solve for weighting coefficients
            % Using mldivide (\) for solving A*sig = Y using
pseudoinverse
            sig = A\(Ycv');
            % Pack for testing
            Rcv.sig = sig;
            Rcv.c = c;
            Rcv.X = Xcv;
            % Evaluate at omitted point
            yop = eval_rbf(Rcv, X(opi,:), Prm);
            if(max(isnan(yop)) >= 1)
                error('RBF: (parfor) yop contains NAN values.
Cannot continue.')
            end
            % Record error
```

```
                err(op) = nrmsd(Y(:,op), yop);
            end % op
            ttop = toc(top);
%             if(matlabpool('size')<=1)
%                 fprintf(1,'\n');
%                 vprint(sprintf('RBF: Cross Validation took %0.8f
seconds',ttop));
%             end
        else
            top=tic;
            Rcv = ROMcv;
            Afull = zeros(size(X,1));
            A = zeros(size(X,1)-1);
            for i=1:size(X,1)
                for j=(i):size(X,1)
                    Afull(i,j) = phi(norm(X(j,:)-X(i,:)), c);
                    Afull(j,i) = Afull(i,j);
                end
            end
            for op = 1:length(cvi)
%                 tic
%                 if(nw<=1)
%                     fprintf(1,'.');
%                 end
                warning('off','all');
%                 fprintf(1,'(1)'); toc, tic
                opi = cvi(op);
%                 fprintf(1,'(2)'); toc, tic
                % Omit the opi-th point
                Xcv = X;
                Xcv(opi,:) = [];
                Ycv = Y;
                Ycv(:,opi) = [];
%                 fprintf(1,'(3)'); toc, tic
                % bias and scaling will change depending on omitted
points!
                % Bias:
                if(rbfib == 1)
                    Rcv.xbias = mean(Xcv);             % mean of each
column (dim) of X, as row vector -> average coordinate vector
                    for i=1:size(Xcv,1)            % bias each parameter
row to center on xbias average
                        Xcv(i,:) = Xcv(i,:)-Rcv.xbias;
                    end
                end
%                 fprintf(1,'(4)'); toc, tic
                if(rbfob == 1)
                    Rcv.ybias = mean(Ycv,2);           % mean of each
row (dim) of Y, as column vector -> averaege snapshot vector
                    for i=1:size(Ycv,2)            % bias each snapshot
column to center on ybias average
                        Ycv(:,i) = Ycv(:,i)-Rcv.ybias;
                    end
                end
%                 fprintf(1,'(5)'); toc, tic
```

```
                % Normalize:
                if(rbfin == 1)
                    Rcv.xscale = norm(Xcv(1,:),2);                %
Find scale
                    for i=2:size(Xcv,1)
                        if(norm(Xcv(i,:),2) > Rcv.xscale)
                            Rcv.xscale = norm(Xcv(i,:),2);
                        end
                    end
%                     fprintf(1,'(6)'); toc, tic
                    for i=1:size(Xcv,1)                         % Scale
each parameter
                        Xcv(i,:) = Xcv(i,:)/Rcv.xscale;
                    end
                end
%                 fprintf(1,'(7)'); toc, tic
                if(rbfon == 1)
                    Rcv.yscale = norm(Ycv(:,1),2);                %
Find scale
                    for i=2:size(Ycv,2)
                        if(norm(Ycv(:,i),2) > Rcv.yscale)
                            Rcv.yscale = norm(Ycv(:,i),2);
                        end
                    end
%                     fprintf(1,'(8)'); toc, tic
                    for i=1:size(Ycv,2)                     % Scale each
snapshot
                        Ycv(:,i) = Ycv(:,i)/Rcv.yscale;
                    end
                end
%                 fprintf(1,'(9)'); toc, tic
                % Build distance weighting matrix
%                 A = zeros(size(Xcv,1));
%                 for i=1:size(Xcv,1)
%                     for j=(i):size(Xcv,1)
%                         A(i,j) = phi(norm(Xcv(j,:)-Xcv(i,:)), c);
%                         A(j,i) = A(i,j);
%                     end
%                 end
                A = Afull;
                A(opi,:) = [];
                A(:,opi) = [];
%                 fprintf(1,'(10)'); toc, tic
                % Solve for weighting coefficients
                % Using mldivide (\) for solving A*sig = Y using
pseudoinverse
                sig = A\(Ycv');
%                 fprintf(1,'(11)'); toc, tic
                % Pack for testing
                Rcv.sig = sig;
                Rcv.c = c;
                Rcv.X = Xcv;
%                 fprintf(1,'(12)'); toc, tic
                % Evaluate at omitted point
                yop = eval_rbf(Rcv, X(opi,:), Prm);
```

```
%                  fprintf(1,'(13)'); toc, tic
%                  if(max(isnan(yop)) >= 1)
%                      error('RBF: (parfor) yop contains NAN values.
Cannot continue.')
%                  end
%                  fprintf(1,'(13)'); toc, tic
                 % Record error
                 err(op) = nrmsd(Y(:,op), yop);
%                  fprintf(1,'(14)'); toc, tic
             end % op
             ttop = toc(top);
%              if(matlabpool('size')<=1)
%                  fprintf(1,'\n');
%                  vprint(sprintf('RBF: Cross Validation took %0.8f
seconds',ttop));
%              end
         end % useparfor
         % If better average error, save new best fit
         if(mean(abs(err)) < besterr)
             besterr = mean(err);
             bestphi = phi;
             bestc = c;
         end
     end % c
end % p

% Build final fitted ROM
% if(matlabpool('size')<=1)
%     vprint(sprintf('RBF: Building Selected ROM'));
% end
% Build distance weighting matrix
A = eye(size(X,1));
for i=1:size(X,1)
    for j=(i):size(X,1)
        A(i,j) = bestphi(norm(X(j,:)-X(i,:)), bestc);
        A(j,i) = A(i,j);
    end
end
% Solve for weighting coefficients
% Using mldivide (\) for solving A*sig = Y using pseudoinverse
sig = A\(Y');
% sig = inv(A)*(Y');

% Pack ROM
% if(matlabpool('size')<=1)
%     vprint(sprintf('RBF: Packing'));
% end
ROM.type = 'RBF';
ROM.sig = sig;
ROM.c = bestc;
ROM.X = X;
ROM.phi = bestphi;
ROM.xbias = xbias;
ROM.xscale = xscale;
ROM.ybias = ybias;
ROM.yscale = yscale;
ROM.err = besterr;

% toc
% vprint(sprintf( '\n')); % DEBUG
end
```

C.2 Evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Evaluate RBF-based ROM
% Jeff Parrish
% Mississipi State University
%
%  Evaluate a ROM constructed with build_rbf().
%
% Inputs:
%   ROM - ROM Model build with build_rbf();
%   X - Parameter vectors to evaluate, row ordered.  Multiple rows will
be
%   evaluated separately.  That is, X is [nsamp x ndim].
%   P - Global Parameter structure, optionally passed as direct copy
%
% Outputs:
%   Y - ROM Outputs, column wise - col Y_i corresponds to row X_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [Y] = eval_rbf(ROM,X,P)
global Param;
if(nargin>2 && isempty(Param))
    Param = P;
end

% vprint(sprintf('     eval_rbf(ROM,X[%d,%d]))',size(X)));

% Check for problems
if(strcmp(ROM.type,'RBF') ~= 1)
    error('eval_rbf(): ROM is not a standard RBF-based model.')
end

% If ROM biased/normalized parameters, do the same to the new X
for i=1:size(X,1)
    if(Param.rbfinbias)
        X(i,:) = X(i,:)-ROM.xbias;
    end
    if(Param.rbfinnorm)
        X(i,:) = X(i,:)/ROM.xscale;
    end
end

% Construct (possibly) biased/normalized output vector
Y = zeros(length(ROM.ybias),size(X,1));
for i=1:size(X,1)
    Ybn = zeros(1,length(ROM.ybias));
    Xbn = X(i,:);
    for p=1:size(ROM.X,1)
        pr = ROM.phi(norm(Xbn-ROM.X(p,:),2), ROM.c);
        Ybn = Ybn + (ROM.sig(p,:) * pr);
% For multidimensional Y, use rows of sigma -> the # cols of sigma
equal the # cols of Y
    end
      Y(:,i) = Ybn';
      % If ROM biased/normalized parameters, do the same to the new X
      if(Param.rbfoutnorm)
            Y(:,i) = Y(:,i)*ROM.yscale;
      end
      if(Param.rbfoutbias)
            Y(:,i) = Y(:,i)+ROM.ybias;
      end
end

end
```

APPENDIX D

ERBF BASED ROM (MATLAB CODE)

D.1 Construction

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Construct a ERBF-based ROM object
% Jeff Parrish
% Mississipi State University
%
% Extended RBF formulation with options for biasing and normalizing
inputs
% and outputs of the system of interest.  Flags are 0 or 1.
% Based on the ERBF formulation proposed by Messac et al.
%
% Inputs:
%   X is row-ordered parameter vectors of snapshots
%   Y is column-ordered snapshots
%   P - Global Parameter structure, optionally passed as direct copy
%
% Outputs:
%   ROM is a structure with the following fields:
%     ROM.type      - Text field denoting ROM type, 'RBF' for this
%     ROM.sig       - Weighting Matrix
%     ROM.X         - Parameters (possibly biased/normalized)
%     ROM.phi       - RBF
%     ROM.gamma     - Smoothness Parameter
%     ROM.n         - Nonlinear NRBF Order
%     ROM.xbias     - X-bias vector, if any
%     ROM.xscale    - X-scaling term, if any
%     ROM.ybias     - Y-bias vector, if any
%     ROM.yscale    - Y-scaling term, if any
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [ROM] = build_erbf(X,Y,P)
global Param;
if(nargin == 3)
    Param = P;
end
tic
% vprint(sprintf('
build_erbf(X[%dx%d],Y[%d,%d]))',size(X),size(Y)));

% Check for problems
if(size(X,1) ~= size(Y,2))
    error('build_erbf(): Row count of X must match column count of Y.')
end

% %If X is one-dimensional, use linear RBF
% if(size(X,2) == 1)
%     phi = @(r)(r);
% end

% Peform Biasing/Normalization
% Note that biasing is performed prior to normalization
% Thus to reconstruct a value, first scale and then add the bias
% Bias/Norm Defaults:
xbias=zeros(1,size(X,2));
xscale=1;
ybias=zeros(size(Y,1),1);
yscale=1;
% Bias:
if(Param.erbfinbias == 1)
    xbias = mean(X);            % mean of each column (dim) of X, as
row vector -> average coordinate vector
    for i=1:size(X,1)           % bias each parameter row to center on
xbias average
        X(i,:) = X(i,:)-xbias;
    end
end
if(Param.erbfoutbias == 1)
    ybias = mean(Y,2);          % mean of each row (dim) of Y, as
column vector -> averaege snapshot vector
    for i=1:size(Y,2)           % bias each snapshot column to center
on ybias average
        Y(:,i) = Y(:,i)-ybias;
    end
end
% Normalize:
if(Param.erbfinnorm == 1)
    xscale = norm(X(1,:),2);                % Find scale
    for i=2:size(X,1)
        if(norm(X(i,:),2) > xscale)
            xscale = norm(X(i,:),2);
        end
    end
    for i=1:size(X,1)                       % Scale each parameter
        X(i,:) = X(i,:)/xscale;
    end
end
if(Param.erbfoutnorm == 1)
    yscale = norm(Y(:,1),2);                % Find scale
    for i=2:size(Y,2)
        if(norm(Y(:,i),2) > yscale)
            yscale = norm(Y(:,i),2);
        end
    end
    for i=1:size(Y,2)                   % Scale each snapshot
        Y(:,i) = Y(:,i)/yscale;
    end
end

% Search kernel functions for best fit
vprint(sprintf('\t\tFitting ERBF Rom: '));
bestphi = Param.phirbf(1).phi;
bestc = Param.rbf_crng(1);
bestgamma = Param.erbf_grng(1);
bestn = Param.erbf_nrng(1);
besterr = 10;
Prm=Param;
```

```
if(size(X,1) > Param.erbfmaxcross)
    cvi(:,1) = randperm(size(X,1));
    cvi(Param.erbfmaxcross+1:end) = [];
else
    cvi = 1:size(X,1);
end
err = ones(length(cvi),1)*10000;
for ph = 1:length(Param.phirbf)
    phi = Param.phirbf(ph).phi;
    % Search n-values to fit the model
    for n = Param.erbf_nrng
        % Search gamma-values to fit the model
        for gamma = Param.erbf_grng
            % Search c-values to fit the model
            for c = Param.rbf_crng
                vprint(sprintf('_')); % DEBUG
                err = [];
                % Perform cross validation to gauge model fit
                parfor op = 1:length(cvi)
                    vprint(sprintf('.')); % DEBUG
                    opi = cvi(op);
                    ROMcv = [];
                    % Omit the op-th point
                    Xcv = X;
                    Xcv(opi,:) = [];
                    Ycv = Y;
                    Ycv(:,opi) = [];
                    % Build distance weighting matrix
                    A = eye(size(Xcv,1));
                    for i=1:size(Xcv,1)
                        for j=i:size(Xcv,1)
                            A(i,j) = phi(norm(Xcv(j,:)-Xcv(i,:)),c);
                            A(j,i) = A(i,j);
                        end
                    end
                    % Build NRBF Coefficient Matrix
                    m = size(Xcv,2);
                    np = size(Xcv,1);
                    B = zeros(np, 3*m*np);
                    BL = zeros(1, m*np);
                    BR = zeros(1, m*np);
                    BB = zeros(1, m*np);
                    for r=1:np
% Per Row of B
                        for p=1:np
% Per Sample Point
                            for d=1:m
% Per Dimensiona of Parameter Space
                                % Difference in dimension d between
point r and point p
                                dx = Xcv(r,d)-Xcv(p,d);
                                % Set phi_ functions based on
dimensional distance
                                if(dx <= -gamma)
% Region I
```

```
                                    phiL = (-n*gamma^(n-1))*dx + (1-
n)*gamma^n;
                                    phiR = 0;
                                    phiB = dx;
                                elseif(dx > -gamma && dx <= 0)
% Region II
                                    phiL = dx^n;
                                    phiR = 0;
                                    phiB = dx;
                                elseif(dx > 0 && dx <= gamma)
% Region III
                                    phiL = 0;
                                    phiR = dx^n;
                                    phiB = dx;
                                else
% Region IV
                                    phiL = 0;
                                    phiR = (n*gamma^(n-1))*dx + (1-
n)*gamma^n;
                                    phiB = dx;
                                end
                                % Set in appropriate index of BL, BR,
BB
                                BL((p-1)*m+d) = phiL;
                                BR((p-1)*m+d) = phiR;
                                BB((p-1)*m+d) = phiB;
                            end
                        end
                        % Concatenate into row Br
                        B(r,:) = [BL BR BB];
                    end
                    % Build Combined System Matrix
                    % Note that Abar is [np] x [np + 3*xdim*np]
                    Abar = [A B];
                    % Solve Linear Programming Subproblem
                    % (b assumed to be vector of ones, so omitted here
                    % min (over lpx): b*lpx
                    % st: Abar*lpx = F
                    %     lpx_i >= 0
                    lpx = ones(np+3*m*np, size(Ycv,1));
                    F = Ycv';
                    options = optimset('Display', 'off', 'UseParallel',
'always', 'MaxIter', 200);
                    % Constrained LP Subproblems
                    for yd = 1:size(Ycv,1)
% For each y dimension
                        b = ones(size(lpx(:,yd)));
% System multiplier is just ones
                        Fi = F(:,yd);
% System output samples for dimension d
                        [lpx(:,yd), fval, exitflag] = linprog(b, [],
[], Abar, Fi, zeros(size(lpx,1),1), [], [], options);
                        % If no feasible solution is found, use the
pseudoinverse method
                        if(exitflag ~= 1)
```

```
                            %warning('build_erbf(): LP Subproblem did
not find a feasible solution.  Defaulting to pseudoinverse method.');
                            lpx(:,yd) = Abar\Fi;
                        end
                    end
                    % Split coefficients into component vectors
                    % Note that each vector has pointwise components in
rows, and y-dimension
                    % components along columns
                    sig = lpx(1:np, :);
                    alpL = lpx((np+1):((m+1)*np), :);
                    alpR = lpx(((m+1)*np+1):((2*m+1)*np), :);
                    beta = lpx(((2*m+1)*np+1):((3*m+1)*np), :);
                    % Pack ROM
                    ROMcv.type = 'ERBF';
                    ROMcv.sig = sig;
                    ROMcv.alpL = alpL;
                    ROMcv.alpR = alpR;
                    ROMcv.beta = beta;
                    ROMcv.X = Xcv;
                    ROMcv.phi = phi;
                    ROMcv.gamma = gamma;
                    ROMcv.n = n;
                    ROMcv.c = c;
                    ROMcv.xbias = xbias;
                    ROMcv.xscale = xscale;
                    ROMcv.ybias = ybias;
                    ROMcv.yscale = yscale;
                    % Evaluate at omitted point
                    yop = eval_erbf(ROMcv, X(opi,:), Prm);
                    % Record error
                    err(op) = nrmsd(Y(:,opi), yop);
                end % op
                % If better average error, save new best fit
                if(mean(abs(err)) < besterr)
                    besterr = mean(err);
                    bestphi = phi;
                    bestc = c;
                    bestgamma = gamma;
                    bestn = n;
                end
            end % c
        end % gamma
    end % n
end % ph

% Build ROM
phi = bestphi;
c = bestc;
n = bestn;
gamma = bestgamma;
% Build distance weighting matrix
A = eye(size(X,1));
for i=1:size(X,1)
    for j=i:size(X,1)
```

```
        A(i,j) = phi(norm(X(j,:)-X(i,:)),c);
        A(j,i) = A(i,j);
    end
end
% Build NRBF Coefficient Matrix
m = size(X,2);
np = size(X,1);
B = zeros(np, 3*m*np);
BL = zeros(1, m*np);
BR = zeros(1, m*np);
BB = zeros(1, m*np);
for r=1:np                                                      %
Per Row of B
    for p=1:np                                                  %
Per Sample Point
        for d=1:m                                               %
Per Dimensiona of Parameter Space
            % Difference in dimension d between point r and point p
            dx = X(r,d)-X(p,d);
            % Set phi_ functions based on dimensional distance
            if(dx <= -gamma)
% Region I
                phiL = (-n*gamma^(n-1))*dx + (1-n)*gamma^n;
                phiR = 0;
                phiB = dx;
            elseif(dx > -gamma && dx <= 0)
% Region II
                phiL = dx^n;
                phiR = 0;
                phiB = dx;
            elseif(dx > 0 && dx <= gamma)
% Region III
                phiL = 0;
                phiR = dx^n;
                phiB = dx;
            else
% Region IV
                phiL = 0;
                phiR = (n*gamma^(n-1))*dx + (1-n)*gamma^n;
                phiB = dx;
            end
            % Set in appropriate index of BL, BR, BB
            BL((p-1)*m+d) = phiL;
            BR((p-1)*m+d) = phiR;
            BB((p-1)*m+d) = phiB;
        end
    end
    % Concatenate into row Br
    B(r,:) = [BL BR BB];
end
% Build Combined System Matrix
% Note that Abar is [np] x [np + 3*xdim*np]
Abar = [A B];
% Solve Linear Programming Subproblem
% (b assumed to be vector of ones, so omitted here
% min (over lpx): b*lpx
% st: Abar*lpx = F
%     lpx_i >= 0
lpx = ones(np+3*m*np, size(Y,1));
F = Y';
options = optimset('Display', 'off');
% Constrained LP Subproblems
for yd = 1:size(Y,1)
% For each y dimension
    b = ones(size(lpx(:,yd)));
% System multiplier is just ones
    Fi = F(:,yd);
% System output samples for dimension d
    [lpx(:,yd), fval, exitflag] = linprog(b, [], [], Abar, Fi, 
zeros(size(lpx,1),1), [], [], options);
    % If no feasible solution is found, use the pseudoinverse method
    if(exitflag ~= 1)
        %warning('build_erbf(): LP Subproblem did not find a feasible 
solution.  Defaulting to pseudoinverse method.');
        lpx(:,yd) = Abar\Fi;
    end
end
% Split coefficients into component vectors
% Note that each vector has pointwise components in rows, and y-
dimension
% components along columns
sig = lpx(1:np, :);
alpL = lpx((np+1):((m+1)*np), :);
alpR = lpx(((m+1)*np+1):((2*m+1)*np), :);
beta = lpx(((2*m+1)*np+1):((3*m+1)*np), :);
% Pack ROM
ROM.type = 'ERBF';
ROM.sig = sig;
ROM.alpL = alpL;
ROM.alpR = alpR;
ROM.beta = beta;
ROM.X = X;
ROM.phi = phi;
ROM.gamma = gamma;
ROM.n = n;
ROM.c = c;
ROM.xbias = xbias;
ROM.xscale = xscale;
ROM.ybias = ybias;
ROM.yscale = yscale;
ROM.err = besterr;

toc
vprint(sprintf( '\n')); % DEBUG
```

D.2 Evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Evaluate ERBF-based ROM
% Jeff Parrish
% Mississipi State University
%
% Inputs:
%   ROM - ROM Model build with build_erbf();
%   X - Parameter vectors to evaluate, row ordered.  Multiple rows will
be evaluated separately.
%   P - Global Parameter structure, optionally passed as direct copy
%
% Outputs:
%   Y - ROM Outputs, column wise - col Y_i corresponds to input
parameter X_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [Y] = eval_erbf(ROM,X,P)
global Param;
if(nargin>2)
    Param = P;
end

% vprint(sprintf('     eval_erbf(ROM,X[%d,%d]))',size(X)));

% Check for problems
if(strcmp(ROM.type,'ERBF') ~= 1)
    error('eval_erbf(): ROM is not a standard ERBF-based model.')
end

% If ROM biased/normalized parameters, do the same to the new X
for i=1:size(X,1)
    if(Param.erbfinbias)
        X(i,:) = X(i,:)-ROM.xbias;
    end
    if(Param.erbfinnorm)
        X(i,:) = X(i,:)/ROM.xscale;
    end
end

% Model parts
np = size(ROM.X,1);
m = size(ROM.X,2);
sig = ROM.sig;
alpL = ROM.alpL;
alpR = ROM.alpR;
beta = ROM.beta;
phi = ROM.phi;

% Construct (possibly) biased/normalized output vector
Y = zeros(length(ROM.ybias),size(X,1));
for xi=1:size(X,1)                                      % For each
evaluation point
    Xi = X(xi,:);                                       % Current
Evaluation Point
    for yd = 1:size(ROM.ybias,1)                        % For each
output dimension
```

```
        for p=1:np                                      % For each
sample point (from original model)
            dx = Xi-ROM.X(p,:);                         % Get
difference vector, evaluation point from sample point
            for xd=1:m                                  % For each x
dimension, determine nrbf contribution; want as rows to add to y-dim
                % Build vectors of phi_ along x-dim, for this sample
point
                if(dx(xd) <= -ROM.gamma)
% Region I
                    phiL(1,xd) = (-ROM.n*ROM.gamma^(ROM.n-1))*dx(xd) +
(1-ROM.n)*ROM.gamma^ROM.n;
                    phiR(1,xd) = 0;
                    phiB(1,xd) = dx(xd);
                elseif(dx(xd) > -ROM.gamma && dx(xd) <= 0)
% Region II
                    phiL(1,xd) = dx(xd)^ROM.n;
                    phiR(1,xd) = 0;
                    phiB(1,xd) = dx(xd);
                elseif(dx(xd) > 0 && dx(xd) <= ROM.gamma)
% Region III
                    phiL(1,xd) = 0;
                    phiR(1,xd) = dx(xd)^ROM.n;
                    phiB(1,xd) = dx(xd);
                else
% Region IV
                    phiL(1,xd) = 0;
                    phiR(1,xd) = (ROM.n*ROM.gamma^(ROM.n-1))*dx(xd) +
(1-ROM.n)*ROM.gamma^ROM.n;
                    phiB(1,xd) = dx(xd);
                end
            end % xd
            % Build vectors of alpha_ and beta, for this sample point,
for this y-dimension
            alpLp = alpL(((p-1)*m+1):(p*m), yd);
            alpRp = alpR(((p-1)*m+1):(p*m), yd);
            betap = beta(((p-1)*m+1):(p*m), yd);
            % Sum radial and nonradial contributions for this sample
point
            dot1 = dot(alpLp,phiL);
            dot2 = dot(alpRp,phiR);
            dot3 = dot(betap,phiB);
            Y(yd,xi) = Y(yd,xi) + (sig(p,yd)*ROM.phi(norm(dx,2),ROM.c))
+ dot1 + dot2 + dot3;
            % If ROM biased/normalized parameters, do the same to the
new Y
            if(Param.erbfoutnorm)
                Y(:,xi) = Y(:,xi)*ROM.yscale;
            end
            if(Param.erbfoutbias)
                Y(:,xi) = Y(:,xi)+ROM.ybias;
            end
        end % p
    end % yd
end % xi

end
```

APPENDIX E

INTERPOLATION METHOD (MATLAB CODE)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Interpolate a ROM object, sanpshots elementwise
% Jeff Parrish
% Mississipi State University
%
% Interpolate a ROM model's elements directly.  Interpolation uses
Grassman
% Manifold Projection for matrices to ensure preservation of
orthonormal
% properties, and standard Radial Basis Functions for direct
interpolation
% of scalars or vectors.  (Note that GMP vectorizes the projected
matrices
% for direct element-wise interpolation via RBF, and then reconstructs
the
% interpolated matrix for reprojection back into basis space.)
%
% Currently, ROMs based on POD and RBF are supported.  Extended RBFs
may be
% incorporated in the future.
%
% Inputs:
%   X               - Row vector coresponding to a single design point
%
% POD-Based ROMs:
%   ROM is a structure with the following fields:
%     ROM.type      - Text field denoting ROM type, 'POD' for this
%     ROM.X         - Parameters (possibly biased/normalized)
%     ROM.basis     - Basis vectors (columnwise, k cols)
%     ROM.C         - Coordinates of snapshots, snapshot i in row i,
basis vector j in col j
%     ROM.k         - Model Order (as constructed, not requested)
%     ROM.RBF       - RBF ROM for interpolating coordinates by
parameter
%     ROM.xbias     - X-bias vector, if any
%     ROM.xscale    - X-scaling term, if any
%     ROM.ybias     - Y-bias vector, if any
%     ROM.yscale    - Y-scaling term, if any
%
% RBF-Based ROMs:
%   ROM is a structure with the following fields:
%     ROM.type      - Text field denoting ROM type, 'RBF' for this
%     ROM.sig       - Weighting Matrix
%     ROM.X         - Parameters (possibly biased/normalized)
%     ROM.phi       - RBF
%     ROM.xbias     - X-bias vector, if any
%     ROM.xscale    - X-scaling term, if any
%     ROM.ybias     - Y-bias vector, if any
%     ROM.yscale    - Y-scaling term, if any
%-----------------------------------------------------------------------
----
function [I TR B0 CV] = interprom(X, rdb, prm, irm, irmdb, useclust)
global romdb;
global Param;
global irom;
global iromdb;
if(nargin >= 2)
    RDB = rdb;
else
    RDB = romdb;
end
if(nargin >= 3)
    Prm = prm;
else
    Prm = Param;
end
if(nargin >= 4)
    irom = irm;
end
if(nargin >= 5)
    iromdb = irmdb;
end
if(nargin >= 6)
    usecluster = useclust;  % use cluster for snapshot roms; 0 or
nworkers per Y-col (irom update)
else
    usecluster=0;
end
IRM = irom;
heal_irom = 0;  % Heal flag for any missing ROMs
debug=1; % DEBUG!  Obviously.
clust = 'raptor';
nodes = 1;
wallhours = 1;
useparfor=1; % Use parfor loops or not
if(debug)
    vprint(sprintf('\t\t050 interprom(%0.4f %0.4f %0.4f %0.4f %0.4f
%0.4f %0.4f %0.4f)',X));
end

% Compile list of X-coords in romdb for comparison to IROM model.
% Update model if different.
XR = zeros(length(RDB), size(RDB(1).X,2));
for i=1:length(RDB)
    XR(i,:) = RDB(i).X;
end
if(isequal(size(irom.X),size(XR)))
    if(isequal(irom.X, XR))
        update_irom = 0;
    else
        update_irom = 1;
        if(debug)
            vprint(sprintf('\t\t\t055 Updating IROM...'));
        end
        vprint('EQUAL SIZE, NONEQUAL VALUE');
        vprint(sprintf('>>> [%0.4f %0.4f], [%0.4f %0.4f], [%0.4f
%0.4f], [%0.4f %0.4f], [%0.4f %0.4f]', irom.X));
                vprint(sprintf('>>> [%0.4f %0.4f], [%0.4f %0.4f],
[%0.4f %0.4f], [%0.4f %0.4f], [%0.4f %0.4f]', XR));
```

```
        % Clear temporary interpolated ROM storage
        iromdb(:) = [];
    end
else
    vprint('NONEQUAL SIZE');
    vprint(sprintf('irom.X[%d, %d]: ',size(irom.X)))
    vprint(sprintf('\n[%0.4f %0.4f], [%0.4f %0.4f], [%0.4f %0.4f],
[%0.4f %0.4f], [%0.4f %0.4f]', irom.X));
    vprint(sprintf('XR[%d, %d]: ',size(XR)))
    vprint(sprintf('[%0.4f, %0.4f, %0.4f, %0.4f, %0.4f, %0.4f, %0.4f,
%0.4f]', XR));
    vprint(sprintf('\n[%0.4f %0.4f], [%0.4f %0.4f], [%0.4f %0.4f],
[%0.4f %0.4f], [%0.4f %0.4f]', XR));
    update_irom = 1;
    iromdb(:) = [];
    if(debug)
        vprint(sprintf('\t\t\t055 Updating IROM...'));
    end
end

% Search for existing interpolated ROM in iromdb; if it matches, return
it.
% Otherwise build as usual and add new ROM to iromdb.
if(checkdb(X,iromdb))
    vprint(sprintf('      Found a current IROM for X.\b'));
    R = getdb(X,iromdb);
    I = R.ROM;
    TR = [];
    B0 = [];
    CV = [];
    return;
end

% Select a reference basis from the database - in this case, the
nearest point
vprint('      Selecting reference basis...');
k = RDB(1).ROM.k;
d = norm(X-RDB(1).X,2);
di = 1;
for i=2:length(RDB)
    if(norm(X-RDB(i).X,2) < d)
        d = norm(X-RDB(i).X,2);
        di = i;
    end
end

% Set Reference Data
X0 = RDB(di).X;
B0 = RDB(di).ROM.basis(:,1:Prm.k);
C0 = RDB(di).ROM.C;
GMP1 = eye(size(B0,1)) - B0*B0';

% Project ROM bases into tangent space
vprint('      Projecting ROMs into tangent manifold...');
```

```
for i=length(RDB):-1:1    % Preallocate backwards!  Faster for list,
still have to iterate: can't preallocate all fields simultaneously
    GMP(i).T = zeros(size(B0));
end
for i=1:length(RDB)
    if (i==di)
        continue
    end
    Mgmp = GMP1 * RDB(i).ROM.basis(:,1:Prm.k) * inv(B0' *
RDB(i).ROM.basis(:,1:Prm.k)); %#ok<MINV>
    [Ui,Si,Vit] = svd(Mgmp,0);
    GMP(i).T = Ui*atan(Si)*Vit;
end

% Gather info for RBF interps
vprint('      Gathering RBF interpolation info...');
Xset = XR;
% [nsamp x xdim]
Mset = zeros(size(GMP(1).T,1)*size(GMP(1).T,2), length(RDB));
% [Tn*Tm x nsamp]
Mr = size(GMP(1).T,1);
if(update_irom)
    %         xbset = zeros(length(RDB(1).ROM.xbias), length(RDB));
% [_dim x nsamp]
    %         xnset = zeros(length(RDB(1).ROM.xscale), length(RDB));
% ...
    ybset = zeros(length(RDB(1).ROM.ybias), length(RDB));
    ynset = zeros(length(RDB(1).ROM.yscale), length(RDB));
end
for i=1:length(RDB)
    Mset(:,i) = vectorize(GMP(i).T);
    if(update_irom)
        %             xbset(:,i) = RDB(i).ROM.xbias;
% Note we are storing these as column vectors, as expected for the
system output for RBF interpolation
        %             xnset(:,i) = RDB(i).ROM.xscale;
        ybset(:,i) = RDB(i).ROM.ybias;
        ynset(:,i) = RDB(i).ROM.yscale;
    end
end

% Interpolate matrices elementwise using Radial Basis Functions
vprint('      Building Projected Basis ROMs...');
vprint(sprintf('      Building basis interpolation ROMs (%d
total)...',length(RDB(1).ROM.ybias)));
RBFM = build_rbf(Xset, Mset, Prm);
for i=length(RDB(1).ROM.ybias):-1:1
    RBFyb(i).ROM = [];
end
if(update_irom)
    vprint('      (Update IROM) Updating Y-Bias and Y-Norm ROMs...');
    %         RBFxb = build_erbf(Xset, xbset);
    %         RBFxn = build_erbf(Xset, xnset);
    L = length(RDB(1).ROM.ybias);
    if(usecluster) % ad hoc parallelism for speeding up this update
silliness
        ndiv = usecluster;
        Lrng = [ 1:max(floor(L/ndiv),1):L (L+1) ];
        save one.mat L ndiv Lrng Xset ybset Param update_irom -v7;
        % Sub
        for j=1:length(Lrng)-1
            if(exist(sprintf('%s/irom_one_%d.mat',pwd,j),'file') == 2)
                vprint(sprintf('Sub: irom_one_%d.mat already exists,
skipping...',j));
                continue
            end
            filename = sprintf('buildirom_one_%d.scr',j);
            jobname = sprintf('irom_one-%d\n',j);
            cmd = sprintf('matlab -nodisplay -r "buildirom_one(%d);
quit"\n',j);
            makeparjob(filename, jobname, clust, nodes, wallhours,
cmd);
            cmd = sprintf('qsub -q @raptor.hpc.msstate.edu -d %s %s',
pwd, filename);
            system(cmd);
            vprint(sprintf('Submitted buildirom_one_%d.scr',j));
            pause(0.1);  % Give the queue a breather!
        end
        % Wait
        for i=1:L
            RBFyb(i).ROM = [];
        end
        for j=1:length(Lrng)-1
            vprint(sprintf('Wait: Waiting on
%s/irom_one_%d.mat',pwd,j));
            while(exist(sprintf('%s/irom_one_%d.mat',pwd,j),'file') ==
0)
            end
        end
        % Load
        for k=1:length(Lrng)-1
            vprint(sprintf('Load: Waiting on
%s/irom_one_%d.mat',pwd,k));
            while(exist(sprintf('%s/irom_one_%d.mat',pwd,k),'file') ==
0)
            end
            cmd = sprintf('grep \"COMPLETE\" buildirom_one-%d-
log.txt',k);
            running = 1;
            while(running ~= 0)
                running = system(cmd);
                pause(1);
            end
            load(sprintf('irom_one_%d.mat',k),'RBFyb_tmp');
            for j=1:length(RBFyb_tmp)
                if(~isempty(RBFyb_tmp(j).ROM))
                    RBFyb(j).ROM = RBFyb_tmp(j).ROM;
                end
            end
        end
    elseif(useparfor) % not using cluster parallelism
        nw = Prm.nworkers;
        lyb = length(RDB(1).ROM.ybias);
        parfor i=1:length(RDB(1).ROM.ybias)
            if(nw <= 1)
                vprint(sprintf('      (Update IROM) Building Y-Bias
ROM(%d)/(%d)...',i,lyb));
            end
            RBFyb(i).ROM = build_rbf(Xset, ybset(i,:), Prm);
        end
    else
        for i=1:length(RDB(1).ROM.ybias)
            vprint(sprintf('      (Update IROM) Building Y-Bias
ROM(%d)/(%d)...',i,length(RDB(1).ROM.ybias)));
            RBFyb(i).ROM = build_rbf(Xset, ybset(i,:), Prm);
        end
    end
    vprint('      (Update IROM) Building Y-Norm ROM...');
    RBFyn = build_rbf(Xset, ynset, Prm);
else
    vprint('      Using existing Y-Bias and Y-Norm ROMs...');
    %         RBFxb = irom.ROMxb;
    %         RBFxn = irom.ROMxn;
    for i=1:length(RDB(1).ROM.ybias)
        RBFyb(i).ROM = irom.ROMyb(i).ROM;
    end
    RBFyn = irom.ROMyn;
end

% Evaluate basis interpolation ROM
vprint('      Evaluating interpolated basis...');
TR = matrixize(eval_rom(RBFM, X, Prm), Mr);
%     xbR = eval_rom(RBFxb, X, Prm)';
% Convert these back to row vectors
%     xnR = eval_rom(RBFxn, X, Prm)';
ybR = zeros(length(RDB(1).ROM.ybias),1);
for i=1:length(RDB(1).ROM.ybias)
    ybR(i,1) = eval_rom(RBFyb(i).ROM, X, Prm)';
end
ynR = eval_rom(RBFyn, X, Prm)';

% Project interpolated basis from manifold back into basis space
vprint('      Reprojecting new basis into basis space...');
[UR,SR,VRt] = svd(TR,0);
BR = B0*VRt'*diag(cos(diag(SR))) + UR*diag(sin(diag(SR)));

% Normalize the basis vectors
% TODO - Review this step (necessary?  impact?)
for i=1:min(Prm.k,size(BR,2))
    BR(:,i) = BR(:,i)/norm(BR(:,i),2);
end

% Collect snapshots
vprint('      Collecting Snapshots for interpolation...');
```

```
V = zeros([length(RDB), length(RDB(1).ROM.ybias), size(Prm.P,1)]);
for r=1:length(RDB)
    V(r,:,:) = eval_rom(RDB(r).ROM,Prm.P, Prm);
end

% Elementwise snapshot interpolation
vprint(sprintf('      Interpolating snapshots elementwise (%d x %d = %d
total)...',size(V,2),size(V,3),size(V,2)*size(V,3)))
for k=size(V,3):-1:1
    for j=size(V,2):-1:1
        CRBF(j,k).ROM = [];
    end
end
CV = zeros(size(V,2), size(V,3));
%         for j=1:size(V,2)  % Healing
%             for k=1:size(V,3)
%                 CRBF(j,k).ROM = I.ROMC(j,k).ROM;
%             end
%         end

if(usecluster) % ad hoc parallelism for speeding up this update
nonsense
    ndiv = usecluster;
    L = size(V,2); % Rows of Y
    Lrng=[];
    Lrng = [1:max(floor(L/ndiv),1):L (L+1)];
    save two.mat L Lrng ndiv CRBF irom X Xset V Param update_irom -v7;
    % Sub
    for k=1:size(V,3) % Cols of Y
        for j=1:length(Lrng)-1 % Row-groups
            if(exist(sprintf('%s/irom_two_%d-%d.mat',pwd,k,j),'file')
== 2)
                vprint(sprintf('Sub: irom_two_%d-%d.mat already exists,
skipping...',k,j));
                continue
            end
            filename = sprintf('buildirom_two_%d_%d.scr',k,j);
            jobname = sprintf('irom_two-%d-%d\n',k,j);
            cmd = sprintf('matlab -nodisplay -r "buildirom_two(%d,%d);
quit"\n',j,k);
            makeparjob(filename, jobname, clust, nodes, wallhours,
cmd);
            cmd = sprintf('qsub -q @raptor.hpc.msstate.edu -d %s %s',
pwd, filename);
            system(cmd);
            pause(0.1);
        end
    end
    % Wait
    for k=1:size(V,3)
        for kk=1:length(Lrng)-1
            vprint(sprintf('Wait: Waiting on %s/irom_two_%d-
%d.mat...',pwd,k,kk));
            while(exist(sprintf('%s/irom_two_%d-
%d.mat',pwd,k,kk),'file') == 0)
```

```
            end
        end
    end
    % Load
    for k=1:size(V,3)
        for i=L:-1:1
            CRBF(i,k).ROM = [];
            CV(i,k) = 0;
        end
        for kk=1:length(Lrng)-1
            % Wait for each file in turn
            vprint(sprintf('Load: Waiting on %s/irom_two_%d-
%d.mat...',pwd,k,kk));
            while(exist(sprintf('%s/irom_two_%d-
%d.mat',pwd,k,kk),'file') == 0)
            end
            cmd = sprintf('grep \"COMPLETE\" buildirom_two-%d-%d-
log.txt',k,kk);
            running = 1;
            while(running ~= 0)
                running = system(cmd);
                pause(1);
            end
            load(sprintf('irom_two_%d-
%d.mat',k,kk),'CRBF_tmp','CV_tmp');
            for j=1:length(CRBF_tmp)
                if(~isempty(CRBF_tmp(j).ROM))
                    CRBF(j,k).ROM = CRBF_tmp(j).ROM;
                    CV(j,k) = CV_tmp(j);
                end
            end
        end
    end
else % Not using cluster parallelism
    for k=1:size(V,3) % Cols of Y
        %             vprint(sprintf('snapshot col %d',k));
        IRM = irom;
        if(useparfor)
            if(update_irom || isempty(IRM.ROMC(j,k).ROM))
                vprint(sprintf('      Building Y-Norm ROMs(-
,%d)/(%d,%d)...',j,k,size(V,2),size(V,3)));
            end
            parfor j=1:size(V,2) % Rows of Y
                if(update_irom || isempty(IRM.ROMC(j,k).ROM))
                    CRBF(j,k).ROM = build_rbf(Xset, V(:,j,k)', Prm);
                else
                    CRBF(j,k).ROM = IRM.ROMC(j,k).ROM;
                end
                %                 vprint(sprintf('j=%d',j));
                CV(j,k) = eval_rom(CRBF(j,k).ROM,X,Prm);
            end
        else
            for j=1:size(V,2) % Rows of Y
                if(update_irom || isempty(IRM.ROMC(j,k).ROM))
```

```
                    vprint(sprintf('      Building Y-Norm
ROM(%d,%d)/(%d,%d)...',j,k,size(V,2),size(V,3)));
                    CRBF(j,k).ROM = build_rbf(Xset, V(:,j,k)', Prm);
                else
                    CRBF(j,k).ROM = IRM.ROMC(j,k).ROM;
                end
                %                 vprint(sprintf('j=%d',j));
                CV(j,k) = eval_rom(CRBF(j,k).ROM,X,Prm);
            end
        end
    end
end
%         for j=1:size(V,2)  % Healing
%             for k=1:size(V,3)
%                 I.ROMC(j,k).ROM = CRBF(j,k).ROM;
%             end
%         end
%         irom = I;

% Bias/Normalize Projected Snapshots
%     if(Prm.podoutbias)
%         for s=1:size(CV,2)
%             CV(:,s) = (CV(:,s)-ybR');
%         end
%     end
%     if(Prm.podoutnorm)
%         for s=1:size(CV,2)
%             CV(:,s) = CV(:,s)/ynR;
%         end
%     end

% Project onto basis
CR = zeros(size(Prm.P,1), size(BR,2));
vprint('      Projecting onto new basis...');
for s=1:size(Prm.P,1)
    for b=1:size(BR,2)
        CR(s,b) = dot((CV(:,s)-ybR)/ynR, BR(:,b));
        %             err = 0.000001*rand(size(ybR));
        %             CR(s,b) = dot(((Y(:,s)+err)-ybR)/ynR, BR(:,b));
    end
end

% Build RBF model for parameter interpolation
vprint('      Building new coordinate interpolation ROM...');
P = Prm.P;
xbias = RDB(1).ROM.xbias;       % Bias:
if(Prm.podinbias == 1)
    for i=1:size(P,1)           % bias each parameter row to center on
xbias average
        P(i,:) = P(i,:)-xbias;
    end
end
xscale = RDB(1).ROM.xscale;     % Normalize:
if(Prm.podinnorm == 1)
    for i=1:size(P,1)                       % Scale each parameter
```

```
        P(i,:) = P(i,:)/xscale;
    end
end
RBFM = build_rbf(P,CR', Prm);

% Pack neatly {type,X,basis,C,k,RBF,xbias,xscale,ybias,yscale}
vprint('      Packing new ROM...');
I.type = 'POD';
I.X = P;
I.basis = BR;
I.C = CR;
I.k = Prm.k;
I.RBF = RBFM;
I.xbias = RDB(1).ROM.xbias;       % No need to interpolate input
adjustments, all ROMs use same inputs
I.xscale = RDB(1).ROM.xscale;
I.ybias(1:size(RDB(1).ROM.ybias,1),1) = ybR(:);
I.yscale = ynR;

% Update IROM if needed
if(update_irom)
    vprint('      (Update IROM) Committing update changes...');
    irom.X = XR;
    for j=1:length(RDB(1).ROM.ybias)
        irom.ROMyb(j).ROM = RBFyb(j).ROM;
    end
    irom.ROMyb = RBFyb;
    irom.ROMyn = RBFyn;
    for j=1:size(V,2) % Rows of Y
        for k=1:size(V,3) % Cols of Y
            irom.ROMC(j,k).ROM = CRBF(j,k).ROM;
        end
    end
    vprint('      Saving IROM...');
    save('irom.mat', 'irom', '-v7');
end

% Add interpolated ROM to iromdb
vprint('      Adding IROM to database...');
iromdb(end+1).X = X;
iromdb(end).ROM = I;
% vprint('      Saving IROMDB...');
% save('iromdb.mat', 'iromdb');

end
```

APPENDIX F

MATERIAL PROPERTIES FOR STRUCTURAL ANALYSIS

Table F.1 Material Properties for Aluminum 6061

Property	*Symbol*	*Value*	*Units*
Density	ρ	2700.0	kg/m^3
Elastic Modulus	E	68.9475729 E+9	Pa
Shear Modulus	G	25.993235 E+9	Pa
Poisson's Ratio	ν	0.33	$-$
Compressive Allowable	M_{CS}	344.737865 E+6	Pa
Tensile Allowable	M_{TS}	172.368932 E+6	Pa

Table F.2 Material Properties for Divinycell F40

Property	*Symbol*	*Value*	*Units*
Density	ρ	40	kg/m^3
Elastic Modulus	E_1	68.950 E+3	Pa
	E_2	68.950 E+3	Pa
Shear Modulus	G_{13}	8.5 E+6	Pa
	G_{23}	8.5 E+6	Pa
Poisson's Ratio	ν	0	$-$

APPENDIX G

DERIVATION OF EQUIVALENT SOLID PLATE THICKNESS FOR COMPOSITE SANDWICH PANELS

In order to simplify the buckling analysis for the TWO problem, each aluminum sandwich panel was approximated as a solid plate of equivalent thickness. This procedure was drawn from the literature.[158–160] The derivation for that relation is shown here, based on Bruhn.[161]

For the TWO problem, the chief parameter which must be matched between the composite and solid plates is that of bending stiffness. For a solid plate, this is given as

$$D = \frac{Et^3}{12(1-\nu^2)} \tag{G.1}$$

where E is the modulus of elasticity, t is the plate thickness, and ν is Poisson's ratio. For a composite sandwich plate with equal face sheet thicknesses of arbitrary value

$$D = \frac{E' t_F h^2}{2(1-\nu^2)} + \frac{E' t_F{}^3}{6(1-\nu^2)} \tag{G.2}$$

where E' is the effective modulus of elasticity, t_F is the face thickness, and h is the distance between the face sheet centroids. For a symmetric sandwich plate, $h = t - t_F$, where t_F is the thickness of one face sheet. It is also assumed for this problem that the effective modulus of elasticity equals the normal modulus.

Thus, to compute the equivalent solid plate thickness, t_{eq} for a composite sandwich plate with the same bending thickness, we can equate the two formulae

$$\frac{E t_{eq}{}^3}{12(1-\nu^2)} = \frac{E t_F h^2}{2(1-\nu^2)} + \frac{E t_F{}^3}{6(1-\nu^2)} \tag{G.3}$$

Solving, the equivalent solid plate thickness is given as

$$t_{eq} = \left(6t_F(t - t_F)^2 + 2t_F{}^3\right)^{1/3} \tag{G.4}$$

APPENDIX H

REFERENCE WING PARAMETERS FOR TRANSPORT WING OPTIMIZATION PROBLEM

Table H.1 Reference Wing Parameters

Property	*Symbol*	*Value*	*Units*
Span	b_{ref}	36.576	m
Root Chord	c_{ref}	7.620	m
Aspect Ratio	AR_{ref}	6.8571	$-$
Taper Ratio	$TR_{ref}, \left(\frac{c_t}{c_r}\right)_{ref}$	0.4	$-$
Sweep	p_{ref}	7.62 / 36.576	$-$
Airfoil Thickness Ratio	$\left(\frac{t}{c}\right)_{ref}$	0.12	$-$
Wing Area	S_{ref}	195.096	m^2
Drag Force	D_{ref}	177,928.965	N
Takeoff Gross Mass	$TOGM$	136,078	kg
Range	R	9.260 E+6	m