An Object-Oriented Analysis and Optimization Control Environment for the Conceptual Design of Aircraft

by

Brett Malone

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

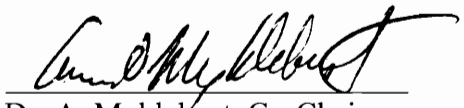
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

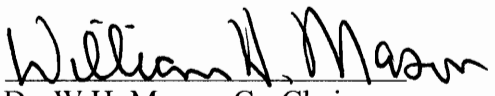
in

Aerospace Engineering

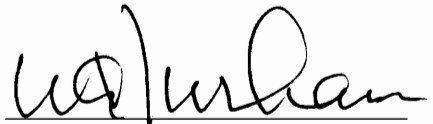
APPROVED:



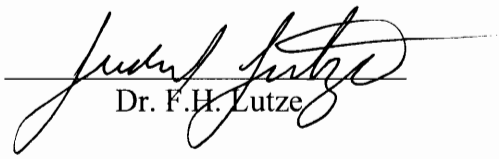
Dr. A. Myklebust, Co-Chairman



Dr. W.H. Mason, Co-Chairman



Dr. W.C. Durham



Dr. F.H. Lutze

Dr. R.T. Haftka

April 23, 1996
Blacksburg, Virginia

AN OBJECT-ORIENTED ANALYSIS AND OPTIMIZATION CONTROL ENVIRONMENT FOR THE CONCEPTUAL DESIGN OF AIRCRAFT

by

Brett Malone

Dr. A. Myklebust, Co-Chairman Dr. W.H. Mason, Co-Chairman

Aerospace Engineering

This dissertation presents a new software environment for performing design activities utilizing numerical optimization algorithms. Specifically, the research focuses on new techniques for interacting with optimization software, unique methods for handling design information, and creative methods for visualizing optimization results.

The application of computer aided optimization algorithms in the engineering design process is hindered by a lack of software methodology and subsequent tools that give the designer adequate control of the process. This process includes setting up the problem, executing the problem with proper key feedback, and extracting the pertinent information from the results to make sound engineering decisions. Elements of data management and visualization are critical to supporting this decision making process.

Numerical optimization can be a powerful tool for the design engineer. If applied properly, vast savings in time and analysis effort can be realized. All too often, however,

optimization is under-utilized because of lack of trust for the methods, or, more commonly, a lack of understanding for why the design arrived at the final result. The optimization approach and supporting software tools developed in this research provide a system that offers the design engineer insight through visualization into the complete history of the optimization problem.

For the designer to understand the results from the optimization process, he must be presented with a *traceable* path of changes to the design and what the influences were at each change. Considerable time is spent disseminating the results of a problem after the optimization algorithms are utilized. Results are scrutinized and histories are examined in great detail to understand why the process arrived at the resulting design.

In addition to presenting comprehensible results of the optimization routines, sophisticated software control of the optimization of complex engineering systems is necessary. The algorithms as well as the actual problem formulation must be accessible and controllable for the design engineer to fully realize the capability of numerical optimization.

The base treatise proposed herein is divided into three distinct areas:

- Software tools for improving process interaction and feedback
- A mathematical strategy for gaining insight from process information
- Illustrative examples of how the developed methods are employed

Acknowledgments

This research was supported through the ACSYNT Institute under the Joint Sponsored Research Agreement #8901 and is gratefully acknowledged. Special thanks to NASA Ames Research Center and all the industry members who contributed to the Institute.

I would like to thank the members of my committee for serving in an excellent teaching capacity early in my graduate career, and in an advisory capacity in the later years of my graduate education. Special thanks go to Dr. Arvid Myklebust for providing all the many opportunities for me as a research associate in the Mechanical Engineering Department. Special thanks also go to Dr. Mason for always keeping the airplane in the picture. I would also like to thank Paul Gelhausen for the support throughout the ACSYNT Institute.

Dr. Scott Woyak has been an inspiration to me both as a university colleague and a business partner. Without his keen software sense, I would still be trying to get my first C++ program to compile. I couldn't have chosen a better person to launch a business with. Many thanks!

To Shahab and Greg, thanks for the support and constant jokenet, movienet and e-mail! Thank you, Eric, for having the passion for aircraft and offering me many a good conversation about flying.

This dissertation is the product of years of hard work and constant support from every individual I have had the pleasure to work with. I thank you all.

There are two people who have devoted their lives to making sure I will succeed in every endeavor I undertake. Words cannot begin to capture the sheer determination and sacrifice that my parents, Matz and Elizabeth Malone have given my life over the past three decades.

To my mother, Elizabeth, you have taught me the true lessons of life and I am forever indebted. You have been behind my education ever since you followed me to kindergarten on the first day.

To my father, Matz, thank you for giving me the ability to separate the good from the bad. My inner philosophy comes from our many years together on the water.

To my Aunt Jean, I hope I'm not too late in finally delivering as the "Young Dr. Malone." Thanks for the support.

Preparing a dissertation means making sacrifices at home as well. Thank you, Liz, for mastering the kitchen and preparing many meals for the family during the late nights.

Finally, to my diamond wife, Edna. You have weathered many a late night alone. Your tireless support has kept me going through the good and the bad. I will always be here for you as you were for me. Thank you.

This dissertation is dedicated to the memory of Oscar Sink,
the best engineer I ever met.

Table of Contents

List of Figures

1. INTRODUCTION

This research is directed at developing a new software environment for performing design activities utilizing numerical optimization algorithms. Specifically, the research focuses on new techniques for interacting with optimization software, unique methods for handling design information, and creative methods for visualizing optimization results.

Numerical optimization can be a powerful tool for the design engineer. If applied properly, vast savings in time and analysis effort can be realized. All too often, however, optimization is under-utilized because of lack of trust for the methods, or, more commonly, a lack of understanding for why the design arrived at the final result.

Optimization research efforts typically focus on the refinement or development of algorithms for searching the design space [1]. Advances in the area of user interfacing and operating shells have supplied better tools for interacting with optimization routines. Little effort is spent, however, in the presentation and analysis of the actual results from the algorithms. For the designer to understand the results from the optimization process, he must be presented with a *traceable* path of changes to the design and what the influences were at each change. Considerable time is spent disseminating the results of a problem after the optimization algorithms are utilized. Results are scrutinized and

histories are examined in great detail to understand why the process arrived at the resulting design.

In addition to presenting comprehensible results of the optimization routines, sophisticated software control of the optimization of complex engineering systems is necessary. The algorithms as well as the actual problem formulation must be accessible and controllable for the design engineer to fully realize the capability of numerical optimization.

It is necessary to provide a flexible architecture for the optimization environment such that access to the key engineering simulation is straightforward. A design environment that allows the simple integration and control of a wide range of analysis modules gives the designer the freedom to combine analysis in new and unique ways. Reuse of existing engineering analysis software to be used in conjunction with other existing systems shrinks the design cycle time and offers the designer the ability to enable the software in ways not originally envisioned.

1.1 Motivation

This section will briefly outline the motivation for undertaking the research presented in this dissertation. The primary motivation for research into optimization implementations was the AirCraft SYNThesis (ACSYNT) [2] project. The ACSYNT Institute [3] was a five-year (extended to six) Joint Sponsored Research Agreement (JSRA) between the

National Aeronautics and Space Administration (NASA), Industry and Virginia Tech. Its goal was to research and develop software for the computer-aided design of aircraft.

1.1.1 AirCraft SYNThesis (ACSYNT)

The ACSYNT program contains analysis modules for conceptual-level design and sizing of aircraft. Figure 1 shows the structure of the program and the available analysis modules. ACSYNT is typically used to perform estimates of takeoff gross weight and mission performance, given a specified aircraft geometry and estimates of propulsion and aerodynamic characteristics.

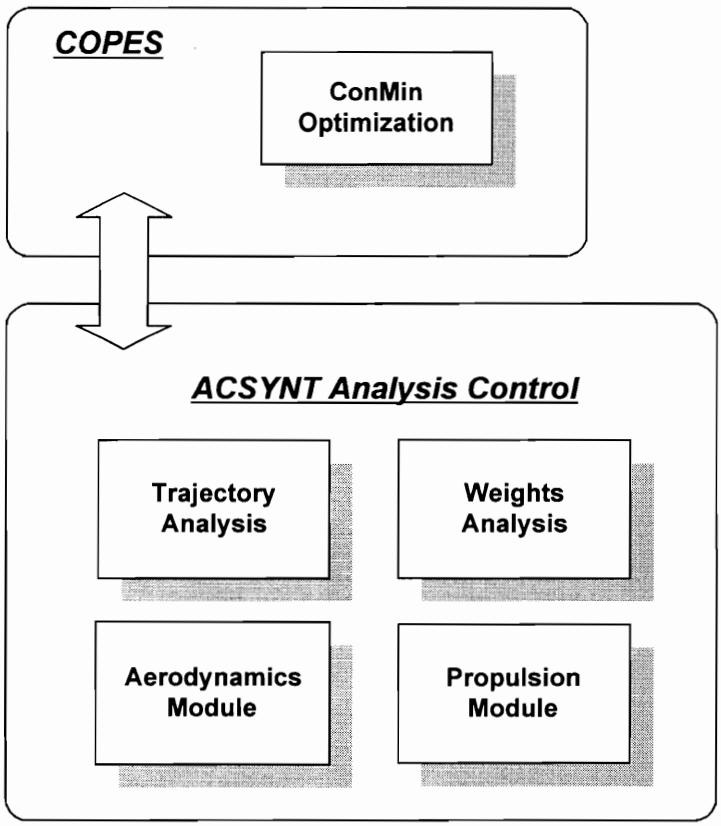


Figure 1 ACSYNT Analysis Software Structure.

1.1.2 Control Program for Engineering Synthesis (COPES)

The optimization capabilities of ACSYNT are controlled by the Control Program for Engineering Synthesis (COPES) [4]. The main functionality of COPES is to provide access to the modules of ACSYNT and to offer various design modes for trade and optimization studies. These modes include functions such as two-variable function plotting, optimization, and sensitivity analysis.

COPES is a FORTRAN program that enhances the functions of any other program by providing optimization capabilities. COPES communicates with the analysis programs by using global data. Global data is disadvantageous because its use results in highly un-maintainable software [5]. Much of the development effort for ACSYNT and COPES has been to update and patch the growing stream of data integrated using global techniques.

CONstrained MINimization (ConMin) [6] is the main optimization algorithm within the COPES program. ConMin is written in FORTRAN and offers basic function minimization through the method of feasible directions [7]. Although still a practical method for numerical optimization, this particular algorithm suffers from numeric instabilities due to scaling, and lacks a robust convergence algorithm.

In addition, the design capabilities within ACSYNT to accommodate project development through graphical analysis of results were outdated. The graphing module of ACSYNT did not allow visualization of optimization results, nor did the file system

allow for the results to be interactively stored and retrieved. Optimized cases could not easily be compared to previous cases. Finally, the ability to perform any type of post-optimization analysis was impossible because the system relied on static, individual output files as its only means of cataloging results.

The research direction, therefore, was to develop a replacement for the COPES/CONMIN program for ACSYNT and to offer enhanced capability for optimization, graphic visualization, and design information handling.

1.2 Proposed Approach

The proposed approach to delivering a replacement for the capabilities of COPES/CONMIN for the ACSYNT program includes the following tasks:

- Creation of a highly interactive environment for utilizing the capabilities of a new optimization algorithm
- Replacement of the CONMIN optimization algorithm with an updated suite of numeric optimization algorithms
- Elimination of the COPES control program in favor of a new integration and control architecture

- Coupling of the main functions of the ACSYNT system to the new optimization environment

1.3 Contributions

The research that was necessary to complete the proposed plan resulted in several contributions in key areas of the system design and optimization communities. Specifically, this work contributes in three main categories:

- Design Information Handling

  This work presents the methodology and a set of software tools for handling multiple optimization problems and presenting information regarding the inter-relationship of these optimization configurations. This method brings the notion of configuration management to the optimization community.

- Multivariate Data Analysis of Optimization Results

  A unique visualization method has been developed to present the user with a concise view of a set of analysis results. This method allows for the visualization of multidimensional results and provides valuable insight into the design space of the each problem.

- Process Feedback

  An original approach has been taken to updating the results of an optimization process as it commences. Through the use of graphical interfaces, an environment has been constructed to give the user unique real-time feedback of the process and the state of all the variables in the problem.

In addition to the above-mentioned contributions, this dissertation presents the reader with four examples of the proposed technology as it is used to solve design problems

1.4 Dissertation Outline

This dissertation is organized in the following manner:

Chapter 2 presents a review of pertinent research conducted in the areas of optimization design environments, visualization, and object-oriented information handling.

Chapter 3 will present an analysis of the system design process. A review of how design activities are conducted using computer modeling techniques is included. Also, an analysis and understanding of the role of numeric optimization in the design process is presented. Finally, the results of this analysis are converted to a basis for a set of software requirements that will be used in the design of the proposed system.

Chapter 4 reviews the fundamentals of the Dynamic Integration System and illustrates the role of Dynamic Integration in the development of the Optimization Workbench.

Chapter 5 presents the Optimization Workbench and outlines the capabilities of the system.

Chapter 6 discusses the mathematical underpinnings of a multivariate approach to the analysis of optimization results.

Chapter 7 and 8 presents several unique design examples, starting with a simple, 2 dimensional algebraic problem, and working through progressive levels of larger system design examples.

Appendix A discusses the software methodology used to develop the Optimization Workbench and the supporting software engineering material. This includes detailed user requirements specifications, a complete set of user scenarios, and a description of the object-oriented design structure

2. INFLUENTIAL RESEARCH

This dissertation focuses on the development of an interactive software environment for delivering optimization capabilities to design engineers. A strategy for identifying the necessary elements to make an innovative, useful environment capable of servicing the evolving needs of the engineering community was key to this research. The following three areas of the published research were investigated to help form this strategy and identify approaches, shortcomings, and implementations of similar efforts.

- Optimization-related engineering design environments
- Visualization approaches for optimization results analysis
- Object-oriented techniques for design information handling

2.1 Optimization-Related Engineering Design Environments

Over the past two decades, significant effort has been focused on the development of flexible software environments to enable integration of user-specific computer models and automated design using numerical optimization algorithms. Vanderplaats has pioneered the concept of an analysis control program through the development of the

COPES [4] program and the follow-on environment, DOC [8]. This work is relevant to the research in this dissertation because it offers the user the ability to integrate his own computer routines and control the execution through a variety of design-related options such as sensitivity analysis, two-variable function space evaluation, and multi-variable optimization.

The ACSYNT [9] program is one example of how COPES is implemented and used to perform engineering design analysis. COPES offers the ACSYNT user the ability to setup and execute optimization problems using a catalog of over 600 design variables, constraints and objective functions.

The ACSYNT program has provided a basis for computer-aided design and optimization of aircraft configurations [10,11,12,13,14,15,16,17,18,19,20,21,22]. ACSYNT is described in detail in chapter 1 of this dissertation. Some of the optimization capabilities of the ACSYNT program are illustrated by Malone [23]. Gallman et al. [24] used the functions of the ACSYNT program to size, validate, and finally optimize a business jet aircraft configuration.

Unfortunately, there are no facilities within COPES for design information management or graphical output. Although COPES can effectively control the user specified analysis program, and perform extensive optimization functions, the program primarily runs as a throughput process. This means that it simply reads a predefined input file, performs the

necessary computations, and writes a formatted output file. The user has no way of interacting with the results, or comparing multiple optimization cases.

Recently, Hwang and Vanderplaats [25] have enhanced the capability of the DOC program by developing a windows-based interactive optimization tool that offers the user problem formulation and results investigation. The user can interactively change the optimization problem and select from a list of user defined analysis variables.

The power of COPES and DOC is that they remain a generic, general purpose control environment that can easily integrate optimization functions with different analysis programs. This attribute is one factor in the longevity of COPES in the engineering design optimization community. The generic interaction feature is an important aspect and has shaped the strategic decisions for the research in this dissertation as well.

Vanderplaats has developed the Design Optimization Tools (DOT) [26] library which works in conjunction with the DOC environment or can be implemented separately. DOT is a suite of optimization algorithms that can be used to perform minimization of constrained, unconstrained and both linear and nonlinear problems. The DOT library will be utilized in this research without the DOC program.

Another early optimization environment that was instrumental in shaping the present research was the "Designer-Augmented Optimization Strategy" developed in 1975 by Michaud and Modrey [27]. Their research focused on human-computer communication

and the effective use of mathematical programming to aid the designer with engineering decisions.

The research of Michaud and Modrey was an early attempt at identifying and offering the user an interactive, visual way of performing optimization. The authors outlined three basic features necessary for implementation of such a system:

> *"1. An operating system allowing interactive communication between the designer himself, his problem, and the optimization algorithm.*
>
> *2. An efficient and flexible optimization algorithm.*
>
> *3. Visual output displays for examination and interrogation of results during and after optimization." [27].*

Similar to the COPES program, a pivotal assumption made in the designer-augmented optimization system was the recognition of the importance of keeping the system as generic as possible. Through a set of four operating modes the user could create the optimization data, execute the optimization algorithm, execute display programs, and edit the input data. Except for the display programs, the system worked in a batch-processing mode on a time-shared terminal system. Execution of the processes occurred on a CDC-6500.

The display of the optimization results was key in the designer-augmented optimization system. The user could create plots of the objective function, constraints and even produce a crude response surface using two independent variables. Additionally, the

authors must be credited for investigating multivariate analysis techniques for results analysis. The concepts of cluster analysis for grouping of optimization solutions were initially introduced into a graphical environment by the authors. Although limited in implementation, the concepts presented by Michaud and Modrey were the direct motivation for the research into advanced multivariate results analysis presented in this dissertation.

As computational models become larger and more complex, the importance of software integration surfaces. The work of Woyak [28,29,30] has laid the ground work for the possibility of the Optimization Workbench. The unique capabilities of the Dynamic Integration System (DIS) allow efficient analysis integration and problem data management. The Dynamic Integration System is an example of a specialized integration architecture focused on solving the problem of integrating multiple engineering analysis programs. The Dynamic Integration System is utilized for the research presented in this dissertation, serving as the connection between the analysis routines and the optimization algorithms. Chapter 4 provides a more complete description of the fundamental concepts and capabilities of the Dynamic Integration System.

As another example of a design environment emphasizing integration aspects, Kroo has developed an engineering design environment for aircraft, the program for aircraft synthesis studies (PASS) [31]. This system is based on a custom integration architecture known as the Quasi-Procedural Method (QPM) [32]. Similar to the assumptions made in

this dissertation, Kroo emphasizes a flexible, generic integration approach coupled with interactive optimization facilities. The PASS system is a specific aircraft design implementation with optimization features built-in. Kroo realized the importance of the user interaction, *"...user interaction with a large multidisciplinary program remains an important issue to be addressed."* [31].

As mentioned above, optimization features are part of the PASS system. Several search algorithms are included in the system, and each has been modified to utilize the features of the QPM integration system. Kroo has shown the importance of customizing the search algorithms to take advantage of integration techniques. For example, the QPM system is an efficient request-driven system that maintains efficiency through executing only the modules necessary when information is requested. This technique proves valuable during the operation of the optimizer such as constraint evaluation. When inactive constraint information is not required by the optimizer, the system does not call the routines to evaluate them. Kroo has brought the issues of efficient engineering software design through integration techniques coupled with optimization algorithms to the attention of the aircraft design community.

Rowell [33] offers a design environment centered around an architecture for multidisciplinary design information management. The Environment for Application Software Integration and Execution (EASIE) system, developed at NASA Langley, illustrates the importance of integration and information management within the design

process. The EASIE system is an executive management application that controls the optimization process. With the recent advanced in optimization techniques within the Multidisciplinary Design Optimization Branch (MDOB) at NASA Langley, the EASIE system is being upgraded to include optimization capabilities along with an X-windows interface. The primary function of EASIE is to aid in the integration of analysis modules through a series of data paths and inter-program communication templates.

One benefit to the EASIE program is that it uses a central database for each analysis method to communicate as well as record configurations as a design evolves. The EASIE program is an effective integration environment, however, it is not interactive, nor does it offer plotting or visualization capabilities.

Design Sheet [34,35] is an interactive environment developed by Rockwell for the conceptual design of aircraft. This tool offers the engineer the ability to perform trade studies and formulate constraints for the design without knowledge of the methods for how to compute the constraints.

Design Synthesis markets and sells the OptDesX [36] program. This program provides the user with exploratory capabilities and good interfacing techniques to the optimization algorithms. Problem information is stored in separate files, and there is no graphical feedback of the process as it executes.

An example of a multidisciplinary framework was constructed for design at the NASA Langley MDO Branch. The Framework for Interdisciplinary Optimization (FIDO) [37,38] program allows multiple processes to execute and perform distributed, multidisciplinary analysis. The user is also presented with a flow diagram of the currently coupled disciplines, and can manipulate the optimization routines to utilize different modeling codes. This system offers the designer optimization capability, however the integration of the analysis programs is limited.

The Notre Dame Optimization (NDOPT) [39] program is an environment for performing optimization and visualization of multidisciplinary processes. This program provides the user with a flowchart of the various disciplines that are contributing to the design problem. Also, NDOPT can perform optimization functions and plotting. Although NDOPT offers interactive optimization, the integration of the analysis methods is limited.

It was the goal of the NDOPT program to demonstrate a feasible multidisciplinary design environment. Simple problems such as 2-bar truss design examples can be solved, however, NDOPT lacks a sophisticated software integration scheme that allows the user to integrate complex programs easily. Finally, there is no interactive control of the optimization results except for simple output files that are saved to the file system. A limited set of plotting tools is available in the NDOPT system, however, no advanced features for visually understanding large amounts of data is available.

Another design environment is the TRANSYS [40] system. This environment presents the user with the capability to perform space vehicle sizing and optimization.

Tong et al. developed the Engineous [41] system for analysis control. Their approach relies on large scale integration through the use input and output data files scheduled through an executive driven by artificial intelligence techniques. Data file scheduling is an example of a procedural, course-grained integration technique. Jennions [42] illustrated the use of the Engineous system for optimization of aircraft engine components.

Hale and Craig [43] illustrate the use of Agent technology to integrate design software. This approach defines "wraps" around existing analysis modules for communication purposes. The current implementation utilizes an interpretive script system to communicate between agents.

Bouchard [44] utilizes a non-procedural approach in the Engineer's Scratch Pad (ESP) system. This approach is well suited to small, analytically defined systems.

Research at Virginia Tech has provided insight into aircraft design using simple, analytic technology models [45,46]. Technology models are simple algebraic expressions of the disciplines in the design process such as aerodynamics, propulsion, or flight performance. Parametric optimization and the use of multiobjective families of optimal solutions are documented in Ref. [47]. The optimization techniques developed using technology

models have provided an understanding of how multiple configurations can be developed to show a more evolutionary approach to design optimization. This work influences the research found in this dissertation by providing a foundation for understanding multiple optimal designs and their interrelationships.

Finally, a complete survey of representative design environments within the field of multidisciplinary aerospace design optimization is presented by Sobieski and Haftka [48]. This reference provides a review of the most active areas in the multidisciplinary design optimization (MDO) field, and contains reference to how human-computer interaction plays a role in the development of optimization-related design environments. The authors state, *"For the human interface, the MDO developers and users seem to have arrived at a consensus that the computer-based MDO methodology is an increasingly useful aid to the creative power of the human mind which is the primary driving force in design."*. This "consensus" supports the need for a flexible optimization environment that allows multiple problem formulation and creative data analysis, and not push-button optimization, thus freeing the engineer to remain the "driving force" in the design process.

2.2 Visualization Approaches for Optimization Results

Design optimization problems usually result in large amounts of data to be interpreted. One area of the published literature that focuses on data analysis and interpretation is

multivariate analysis. Specifically, the techniques that are used to provide visualization of results are investigated within the multivariate analysis community. The methods investigated serve as a foundation on which optimization results visualization techniques can be developed in this dissertation.

Jones [49] has provided a survey of optimization and visualization. He describes a wide variety of applications for visualization of optimization information as well as multivariate data analysis techniques. The ideas found in this dissertation relating to the problem-solving process and the "formulation" of the optimization problem can be traced to Jones. For example, Jones outlines the concept of the different "representation" of the optimization problem. The following items represent the various stages in the optimization modeling process:

- Conceptualization
- Formulation
- Data Collection
- Solution
- Solution Analysis
- Results Presentation

The current approach found in this dissertation focuses on aiding the user with the various stages in the problem life-cycle shown above. Jones surveys visualization techniques for each of the six representations.

Similar to the current research found herein, Jones has identified that *"..algorithms by themselves solve only part of the problem. The formulation must be constructed."* Also, as identified through the use of simplified results presentation, Jones states *"The results of the analysis must be organized or summarized into some representation for the benefit or approval of other parties."*

Finally, Jones provides comments to the research efforts in the area of visualization and optimization. His recommendations for areas of future research include the following:

- Empirical Evidence
- Formulations
- Integration
- New Visualization Techniques

The strategy developed for the present research provide a clear plan for extending the current research and combining the areas of formulation, integration and visualization.

With regard to problem formulation, various authors have proposed the use of graphical or visual languages, to represent a mathematical formulation [50, 51]. Highly interactive

schemes utilizing object-oriented techniques are given in [52]. Reference [53] provides a more complete survey of this area.

The concepts of integration have been outlined in the previous section, therefore, the aspect of visualization as researched for optimization results analysis will be expanded. Specifically, the concepts of scatterplot analysis are interesting for use as a tool for optimization results.

Friedman and Tukey [54] have presented a visualization technique for 3-D scatter plots. This method is based on the elements of maximizing a cluster arrangement using view projection techniques. A unique application of scatterplot diagrams was built by Becker and Cleveland [55]. The method of scatterplot analysis is also presented in Refs. [56, 57, 58]. Additional insight into the application of multivariate analysis techniques using scatter data is given by Keiper and Wickham-Jones [59], developers of the popular Mathematica Program.

The analysis of scatter techniques for data has also been investigated using an approach called "Dynamic Graphics" [60]. Cleveland [61] gives perspective on the use of statistical graphics in support of data structure inspection. The use of the nonlinear mapping technique in this dissertation requires attention to presentation of the mapping to the user. The ideas within Ref. [62] were used to help present the user with a coherent picture of the mapped data. This reference offers the reader ideas for the psychology of graph perception.

Finally, Bell has provided commentary on the future of visualization techniques applied to optimization in reference [63]. Bell's observation that *"Visualization and optimization are complimentary approaches to problem solving, not rivals"* confirms the strategic decision within the current research to bring the two components together for the designer. Furthermore, Bell offers future research recommendations by stating that *"Creative visualization, which involves deriving useful presentation metaphors for problems with no obvious visual representation, should be an important objective for future research."*

2.3 Conceptual Design Information Management Using Object-Oriented Techniques

A major objective of the research proposed herein is to offer a flexible system for handling optimization data and managing multiple problems for rapid recall and comparison. A key aspect to a successful design environment is the ability to manage the vast amounts of data in the form of problem input and results output [64]. A complete survey of the techniques available for managing design information is presented by Fulton and Yeh [65]. The authors outline various methods for organizing design data, including IDEF models [66], Entity Relationship Models (ERM) [67], Systematic Activity Modeling Method (SAMM) [68], Niijssen's Information Analysis Method (NIAM) [69], and Object-Oriented Data Models (OODM) [70]. In support of the concepts delivered in the present research, the authors recognize that *"The OODM seems*

to possess most of the features for the 'ideal' design decision support system for modeling the aerospace vehicle design process."

Object-oriented design methods offers a wide variety of benefits for development of scientific applications [71]. The procedural style of programming, while popular with engineers for solving problems, encounters organization problems such as branching control and model manipulation when applied to large scale implementations [72].

There are many instances of object-oriented techniques applied to the solution of scientific problems. Cellier et al. [73] describe the benefits to using object-oriented techniques for modeling and simulation of physical phenomenon. Ryckbosch developed the "ZAZIE" system to illustrate the benefits of using object-oriented techniques in developing optimization and other numerical methods [74]. Object-oriented methods for aircraft mission and performance analysis was investigated by Yokell and Baker [75]. This approach illustrated the program LOOPS, a modeling and analysis environment comprised of reusable analysis components.

One contributing research focused on utilizing object-oriented techniques to provide optimization functionality to a spacecraft design project [76]. The design and optimization of satellite structural subsystems was investigated using finite element analysis techniques coupled with object-oriented software technology. Specifically, the authors developed a unique class structure to support optimization data organization. Interestingly, a similar analysis of the optimization problem using object-oriented

techniques led to similar class structures as found in the present research. For example, the authors identified that the design variables, objective function, and constraints can be organized into an "OptData" class. The concluding remarks by the authors include the fact that the object-oriented approach helped *"simplify the representation and control of the iterative design process."*

3. PROBLEM ANALYSIS AND REQUIREMENTS DERIVATION

This section introduces a formal analysis of the problem domain for this research. Specifically, the methods for performing engineering design are examined. In support for the development of a new optimization methodology and software environment, the specific activities germane to system design optimization and the use of computer-aided optimization tools are analyzed. An introduction to the current practices of multidisciplinary optimization are given. Also, the application of visualization techniques towards the understanding the results of optimization analysis are presented. This analysis serves to set a foundation for the requirements of the research. Finally, these requirements serve to direct the development of the software tools found in the system.

3.1 Overview

The first part of this chapter provides an introduction and overview of the design process and how the driving factors contribute to the present research.

3.1.1 The Art of Design

The process of engineering design consists of producing information in the form of drawings, analysis, and specifications such that a system can be constructed [77]. This information can be considered "deliverable engineering" and is the product of the iterative design process. There are many phases of the design process. A systems engineering approach to design [78] considers the product in its entire life cycle, from concept through implementation, and finally to retirement.

The elements of a sound design process include methods for understanding the performance of the envisioned system. The goal of the design process is to enhance the performance of the system as much as possible, or to **optimize** the performance. The design engineer is confronted with limitations that he must accept for his system, however. For example, although it might be possible to construct a tunnel across the Atlantic ocean, the cost would be prohibitively expensive. In this case, taking a boat is far more **feasible**.

Systems are designed because the needs of the world change. New technology that is developed as part of basic research in the sciences gives the engineer the opportunity to design better systems. Application of new technologies leads to new systems that are either better in performance, lower in cost, or enhanced in some other performance parameter.

3.1.2 Computer-Aided Engineering

To accomplish the necessary analysis of the system being designed, the engineer must have at his disposal a wide array of tools. The premier tool of the design engineer is the computer. In many situations, an entire system can be designed and tested entirely with a computer. The most notable example of this to date is the Boeing 777. Operating entirely within the domain of the computer is known as a paperless environment. The trend is moving more and more towards this approach to design and manufacturing.

With the advent of powerful computer-aided design tools came the ability to significantly speed the design process. The performance of the system can be estimated in minutes rather than days. Changes to the proposed design can be recorded and tested effortlessly with little or no overhead.

Computer tools are now used at every stage of the design process. The most critical tools used by the design engineer are those that analyze the performance of the system. These methods are most commonly developed as computer models, or simulations, of the actual system. The designer then uses these models to assess a wide variety of design options, quickly evaluating the good and the bad designs.

Typically, these computer-aided tools rely on mathematical models of the actual physical conditions of the system, and the anticipated response of the system to changes in the parameters that define the system. The designer then manipulates these parameters and

observes the changes in the performance. In some instances, the computer models being developed are so large and complex that designers are making computer models of other computer models [79].

3.2 Analysis of the Design Process

The following section provides detail into the process by which systems are designed, including a brief analysis of the dynamics of large and small groups of design engineers, and the information problems associated with various structures.

3.2.1 Design Process Mechanics

The design process is an iterative method for arriving at a system that performs a required set of tasks within acceptable performance boundaries. In a complex system, many performance parameters are considered for evaluation. The system performance is computed using different computer models. Each unique model represents a different aspect of the physical response of the system. The design engineer accumulates a variety of different models that represent various disciplines. These disciplines are usually different branches of a core set of sciences, such as aerodynamics, flight dynamics, or thermodynamics.

3.2.1.1 Group Dynamics

When a small group is engaged in a design activity, the communication between disciplines about the design decisions is usually straightforward. Information is

distributed and processed quickly, and the design can be cycled through many changes rapidly.

However, when the size of the group of engineers becomes large and the computer models being used grow in size and complexity, the design process can grind to a halt. Information is no longer smoothly transferred between computer models, nor are the results shared uniformly among the disciplines. The designer is suddenly consumed in the battle to coordinate the various disciplines and their representative computer models.

3.2.1.2 Discipline Boundaries

As the design of a complex system progresses and becomes more detailed, more emphasis is placed on the design within the boundaries of each discipline domain. When more and more effort is spent within the discipline domain instead of between different disciplines, the various groups within the design team become segregated. The computer models from these segregated groups become harder to coordinate because the lines of communication are blocked by the thick discipline boundaries. The information being passed between disciplines becomes disjoint and sometimes lags behind the current design cycle.

Unfortunately, as the complexity of the system increases, the disciplines must communicate more. Computing the performance for a complex system requires information from all the disciplines. More importantly, however, each discipline must have up-to-date information from the other disciplines. A set of interdependencies

emerges that begin to couple all the disciplines. Soon, each discipline begins to depend on the results from all the other disciplines.

At this point, the designer's job becomes harder because he must not only coordinate the efforts of each discipline, but the information coming from each discipline is sparse and sometimes not even relevant to the current design cycle.

The extreme case of this scenario is where each discipline grows so large that the job of the design coordinator becomes impossible. At this point, each group within the design team begins performing "sub-design" activity. For example, the aerodynamics group within an aircraft design team will perform wing optimization with insufficient knowledge of the internal structural arrangement or the placement of the engine nacelle.

3.2.2 Emergence of Multidisciplinary Optimization

In the midst of the growing boundaries between design disciplines, research began in the late 1980s to formulate a methodology to give the design engineer a way to coordinate and bring control to the design process. The multidisciplinary design optimization approach developed by Sobieski is a method for coordinating large, highly coupled disciplines. The MDO research effort has contributed several key methodologies useful for solving information and optimization problems at several levels of application.

MDO research has offered solutions for decomposing and understanding complex, highly coupled systems. By doing this, the system can be reconfigured in such a way to

minimize the amount of discipline dependency, giving a more straightforward model of the overall system.

3.2.2.1 "Global" Design Sensitivity

The sensitivity of the performance of the system to changes in the defining parameters is critical information for the designer. The sensitivity of multidisciplinary systems is complicated to compute because the disciplines are interdependent. One aspect of the MDO research involves computing the total, or global, sensitivity of the system [80].

The global sensitivity of the system is the response of the system performance to changes in the design parameters. Multidisciplinary optimization techniques provide a methodology to compute global sensitivities by taking into account the influences of all the interdependent disciplines. Without these techniques, erroneous gradient information can be computed for the system and the optimization methods will not change the design in a favorable direction.

3.2.2.2 Identification of a Discipline for the Designer

The MDO efforts have supported the specialized discipline for the design engineer. The designer, however, now requires sophisticated computer modeling for his discipline. Just as developers in various other design disciplines create computer models, or simulations, of their disciplines, multidisciplinary designers are now beginning to model the design process. The research offered in this dissertation begins to address the needs of the design

discipline and provides a set of tools for which the designer can efficiently operate within this discipline.

3.2.3 Visualizing Multidisciplinary Systems: Process Modeling

The design process can be thought of as a group of activities that combine to form either a product or a set of information. Process modeling has been a key focus of the operations research community for many years [81]. Visualization of processes such as a transportation routing system, or a nuclear reactor is key to understanding the information flow and the critical components that must contribute to make the system operate smoothly. Such is the case with the process of multidisciplinary design. The designer must understand the information flowing to and from the contributing disciplines so that the design process can proceed smoothly.

3.2.3.1 Real-Time Systems

Graphical techniques are widely used to model and visualize real-time operating systems with complex processes [82]. For example, computer software exists that monitors and controls the functions of nuclear reactors, steel mills, and automobile manufacturing facilities. Software systems such as Kinisix's Sammi provide the designer with a graphical way of inspecting the results of processes and monitoring the progress of information within that process.

3.2.3.2 Tools for Business Process Modeling

Modeling of information-passing processes can also be found in the business process re-engineering efforts [83]. Software tools such as the TemPRO System [84] offer companies a way of modeling and organizing the processes that they rely on to perform business. Business process modeling consists of examining the flow of information as it is processed within different departments of a company. An example of this modeling is the tracking of an insurance claim through the maze of different departments of a large insurance company. In this case, the different departments represent the various disciplines that operate on the information.

3.2.3.3 Modeling Multidisciplinary Activities

In real-time system simulation, the elements that are passed between components are usually physical entities, such as cars and buses, or fuel and steam. When modeling a design process, the element passed between components is the design information. A multidisciplinary system is first modeled by decomposing the system and representing the information flow between each contributing discipline. Graphical tools have been developed to visualize each component and the flow of information within a multidisciplinary environment. These tools offer the user an understanding of the structure of the process, however, they usually fail to provide an understanding of the dynamics of the information being processed. Furthermore, configuration of the modeling system is cumbersome and a reconfiguration, or re-structuring of the problem can be time-consuming. Finally, these tools still lack the ability to manage multiple problems

flowing through the process, or, the relative differences between two different problem formulations.

3.2.4 Deficiencies: Software Requirements for the Design Process

The previous sections have provided insight into the mechanics of the design process and what kind of software tools are best targeted at solving any problems that are encountered. The following is a list of items that are key to a successful system.

3.2.4.1 Integration of Disciplines

As we have seen, the integration of multiple disciplines is crucial to a successful design. From a software integration standpoint, each model must have access to the information from each computer model. More importantly, the designer must have a coordinated, integrated set of disciplines that he can use to compute the system performance.

3.2.4.2 Information Handling

Once the disciplines are integrated, the designer must interact with the design information in a coordinated manner. Information such as the available parameters that the designer can change for performance enhancement, or the actual system performance variables must be easily accessed and handled as the design is processed. Also, this information must be recorded so that a traceable history of the design is made.

3.2.4.3 Visualization of the Process

As computer modeling becomes more sophisticated, the requirements to visualize more of the actual design process increase. It is well known that visualization of the information flow between complex systems tends to simplify the understanding and provide the user with a concise picture of an otherwise chaotic process [60].

3.2.4.4 Visualization of Results

Finally, the designer is required to process a multitude of information coming from many different computer models. Visualization of the numeric results for the design is a basic feature that must be present for any design environment. An integrated capability to graphically show the design space is crucial to understanding what changes must be made to optimize the design.

3.3 Analysis of Computer-Aided Optimization

This section analyzes the key tool available to the design engineer, the computer-aided optimization algorithm.

3.3.1 The Role of Optimization in the Design Process

The goal of the design process is to optimize the performance of a given system. For some systems, this means minimizing a certain design parameter, weight for example. Other choices might be to maximize a performance parameter, speed, for example.

The act of optimizing a design can take many shapes. The most common approach is to initiate the design, record the performance, then change the design to improve the measured performance accordingly. This process is slow, and is not based on formal feedback information from the design.

3.3.2 Numerical Optimization Algorithms for the Computer

Computational methods for analyzing a given system have provided the designer with the ability to rapidly change the design and recompute the performance almost effortlessly. With this capability, computer algorithms can then be developed to automatically change the design and recompute the performance, leaving the designer to sit back and observe thousands of changes to the system. There are many mathematical methods that have been developed to compute search directions. Some of the more popular methods are outlined below.

- **Gradient-Based Search Direction Approach**

  Computational algorithms to compute the optimal performance of a system rely on the ability to change the system and determine the effects of the change. By computing sensitivities, or gradients, these automatic methods compute the set of best changes to make, or the search direction. The change is made to the design, and the process is repeated, until no more changes can be made to improve the design.

- **Large-Scale Linear Problems**

  Large, linear problems can be solved through a process known as linear programming [85]. These methods are used extensively for routing and scheduling processes through multi-node events. For instance, the airline industry relies heavily on linear

programming optimization techniques to route flights through the minimum number of cities.

- **Other Search Direction Techniques**

  Emerging search direction methods offer approaches that circumvent problems associated calculus-based search methods. One such approach is the genetic algorithm [86]. Instead of using gradient-based information to compute the search direction, the genetic algorithm mimics the natural evolution process of life forms. Included in the process are provisions for mutation and cross-over. The benefits to using a genetic algorithm are that discontinuous design spaces can be traversed, whereas gradient-based, sequential search directions cannot cross discontinuities in design parameters.

3.3.3 Under-utilization of Optimization Algorithms

As outlined above, powerful computational algorithms exist for use by the designer. These methods can automatically search through more design options than humanly possible. Yet, designers are reluctant to employ these techniques within their design process. Why?

3.3.3.1 Getting the Algorithms to the Design

The first problem is exposing the computational algorithms to the computer models of the system being designed. Optimization routines require detailed information from each discipline. The effort necessary to integrate a formal optimization routine with a complex simulation of an engineering system is exceedingly high.

3.3.3.2 Setting up the Problem

Assuming an optimization algorithm can be integrated with a computer model, the proper formulation of the problem must be given. The performance parameters must be selected, and the appropriate design parameters must be selected from each representative discipline. Many times, the designer is not aware of the dependencies of the design parameters to the performance. Sometimes, a design parameter will have no influence on the performance variable selected.

Typically, more than one formulation of the optimization problem is constructed during the design process. As the design effort proceeds, different figures of merit might be selected for investigation. The effects of constraining different parameters are considered, as well as the selection of different design parameters. The most productive strategy is to construct a simple, unconstrained problem initially. Then, as the results of the simple problem are understood, and knowledge about the design space increases, additional design parameters and constraints are added. If the designer continues, the result is a series of optimization formulations that progressively narrow the scope of the design into a well-bounded feasible region.

Although productive, this approach is resource intensive and requires supporting database activity to store and manage each design problem. As more and more optimization problems are executed, the designer is confronted with a configuration management problem for his design data.

3.3.3.3 Understanding the Methods

Robust optimization algorithms are complex. The mathematical formulations necessary to handle a diverse suite of ill-conditioned problems are usually not well understood before a complex problem is tackled. The designer will not entirely trust results from an unknown method.

3.3.3.4 Understanding the Results

Just as it is important to understand the method that automatically changes the design, the results coming from the method must be understood and trusted. The most common concern when a user employs an optimization technique is if that algorithm actually reached an **optimal solution**.

Analysis of optimization results consists of intensive detective work on the part of the designer. The results of an optimization case are seldom looked at with trusting eyes. In fact, very few results are simply accepted without external verification. The practice of analyzing the history of the changes made to the design is common. These **convergence histories** give the designer a view of the combined changes made at each iteration in the automatic process of changing the design. By reviewing convergence histories for a single design, the relative changes from start to finish can be understood.

Users of numerical optimization routines typically try several different algorithms when solving a problem. This technique helps confirm, or deny, the fact that a result is a true optimal point, or simply a **local minimum**.

3.3.4 Optimization and Visualization

"A picture is worth a thousand words," reads the cliché. Engineers communicate in graphs. A table of data is infinitely easier to understand when plotted in 2 or 3 dimensions. Likewise the results of an optimization study are usually best analyzed graphically. Figure 2 illustrates this point by presenting the same convergence history in tabular form and in a 2-D plot. Clearly, the structure of the data and the evolution of the design is better understood by inspecting the plot.

Due to the problems outlined in the previous section, much of the optimization community has relied on graphical visualization techniques to aid in the understanding of optimization results. Many optimization techniques are highly graphical in nature, and the understanding of the results can be made much easier through visualization techniques. Users of these methods rely on graphical techniques such as convergence histories or contour plots to understand what direction the algorithms are taking with the design.

3.3.5 Iterating on Optimal Solutions

As mentioned earlier, the design process is iterative in nature. Likewise, the application of optimization techniques to a design is iterative. The **iterative optimization process** involves application of the optimization algorithms to the computer models, evaluation of the optimal solution, and re-optimizing with a different problem formulation, or different set of models.

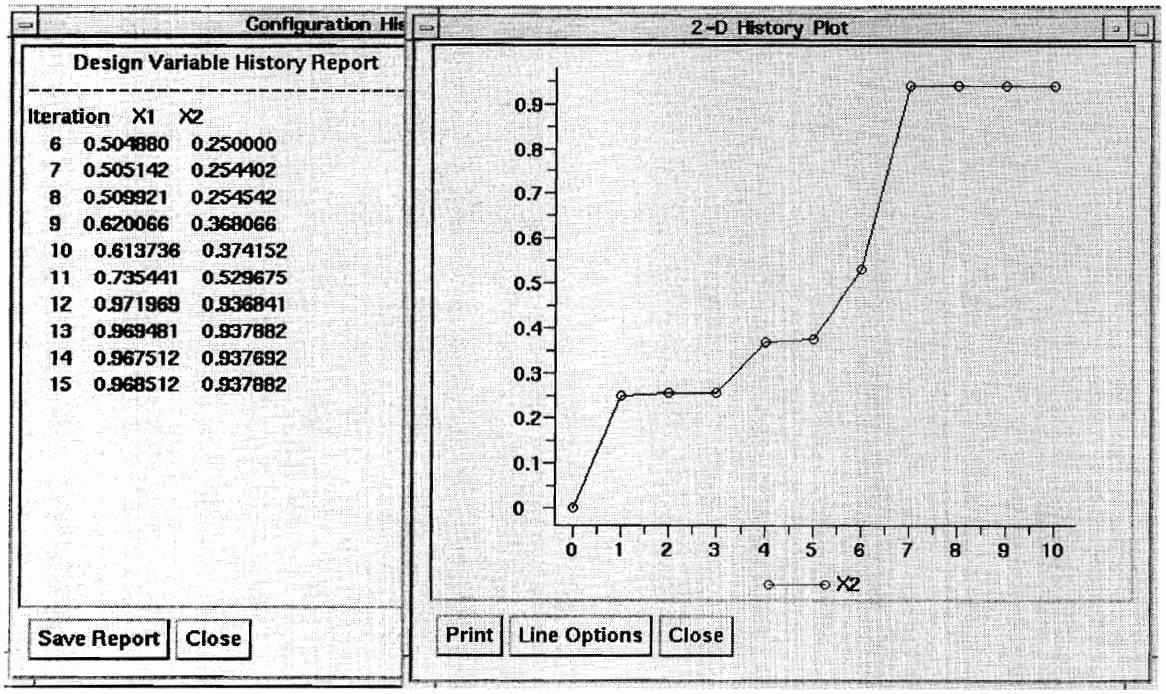


Figure 2 a)Table of convergence history. b) 2-D plot of same convergence history.

3.3.5.1 Designers Create Families of Optimal Solutions

When the computation requirements necessary to analyze a design are cheap, optimal solutions for a variety of different conditions can be constructed. By executing multiple optimization cases, a family of optimal designs is the result. Research has shown that greater insight can be gained from families of optimal solutions as opposed to a single result isolated from the context of any optimal design space.

3.3.6 Deficiencies: Software Requirements for the Computer-Aided Optimization

The previous analysis has provided a foundation for understanding the computer-aided optimization process and what users of these methods require to successfully implement these strategies. The end result of a system delivered to satisfy the above requirements is

to "help the designer become smarter about the design faster" [87]. The following items are critical to delivering a system that provides users of computer-aided optimization a flexible environment for doing optimization.

3.3.6.1 Integration with the Analysis

One of the chief concerns illustrated in the above requirements derivation is the integration and accessibility of the optimization routines with the computer models for the system analysis. Clearly, one of the most important features of a proposed environment will be simple integration with other computer models.

3.3.6.2 Access to Methods

As described above, there are many numerical methods for performing optimization. In fact, one technique used to "debug" the results of an optimization case is to switch methods and attempt the same problem.

A critical requirement for a new optimization environment is to have access to many different methods, and be able to switch them effortlessly for a rapid re-configuration.

3.3.6.3 Problem Formulation/Re-formulation

The formulation of the optimization problem must be simple and straightforward. As illustrated above, if it is complicated for the user to specify the problem, the optimization routines will never be utilized.

3.3.6.4 Feedback from the Process

Much of the requirements analysis presented above focus on understanding the optimization processes and investigating the information coming from those processes. A proposed new environment must give the user a simple, yet powerful set of feedback information such that it is always clear what the process is engaged in.

3.3.6.5 Visualization of Results

The requirements also point to a need for a system that can easily present the design evolution graphically. This feature must be an inspection tool that can bring out the structure in the data such that a designer can understand why the choices were made within the algorithm.

3.3.6.6 Results Analysis

Once the optimization algorithm has completed the changes to the design, and a valid result exists, additional analysis of the results are usually performed. For example, the ability to perturb an optimal design to see the sensitivity of the performance, or assess the change in a constraint. A requirement for the proposed system must include the ability to manipulate the optimal design.

In addition, because we have specified that multiple problem formulations are required to effectively perform optimization, one result verification technique is to present the multiple problem formulations as a family of optimal results and verify critical information such as similarities and differences in the configurations.

3.4 Summary

The previous analysis has formed the foundation for an understanding of the overall design optimization process and the software requirements that best compliment that process. These requirements can be broken down into three main categories:

- Project Control
- Process Visualization
- Results Visualization

4. ENABLING TECHNOLOGY: THE DYNAMIC INTEGRATION SYSTEM

This chapter describes the underlying integration architecture used to facilitate access of the engineering analysis modules to the optimization routines. The base elements in each configuration of each project utilize the unique integration aspects of this system to provide updated values for the model.

4.1 Overview

The architecture utilized to solve the problems of software integration is called the **Dynamic Integration System**. The Dynamic Integration System was developed by Woyak [28] to provide a method to make explicit the functionality and needs of a module. This explicit expression is then used by the Dynamic Integration System to integrate and coordinate modules. This is in contrast to the current technique requiring the programmer to hand integrate each module. By automating the process, it is much easier to deal with large, complex systems.

4.2 Explicit Expression

Two general type of modules that define engineering applications have been identified:

- **analysis codes**. These modules take some input, perform some function on it, and generate some sort of output. Examples include finite-element codes or aerodynamics modules.

- **interaction codes**. These modules interact with the parameter space defined by an application. The modules either specify values for parameters, use values of parameters, or some combination of the two. Examples include graphing and optimization modules.

The purpose of explicitly expressing the role of each module is that the interaction with those modules can be automated. Integration of analyses, execution control, and coordination of input from the user can all be handled by the underlying Dynamic Integration System architecture. These features can also be handled in a generic manner such that new modules can be created without prior knowledge of the integrated application.

4.3 Dynamic Variables and Dependency Hierarchies

The two primary constructs used by the Dynamic Integration System are **Dynamic Variables** and **Dependency Hierarchies**. Dynamic Variables are C++ objects used to model key parameters within an application. Dependency Hierarchies are formed by combining relationships between Dynamic Variables.

Dynamic Variables, as implemented in the Dynamic Integration System, exhibit the following features:

- **C++ objects**. Dynamic Variables are implemented as C++ classes. To create a Dynamic Variable, an instance of a Dynamic Data Type class is instantiated.

- **shareable**. Dynamic Variables are identified by text strings known as Dynamic Identifiers. If multiple Dynamic Variables are created using the same Dynamic Identifier, then those instances will share the same data and states.

- **distributed**. Two separately executing applications may share common Dynamic Variables. This permits analyses to be controlled and interactively used from remote sites. This also permits the rapid construction of applications by linking Dynamic Variables from several separately executing processes to form a single, collaborative design environment.

- **states**. All Dynamic Variables are classified by two states. The first state defines whether the Dynamic Variable is an input or an output of an analysis. A Dynamic Variable may also be either valid or invalid. An invalid state means the analysis connected to the Dynamic Variable must be executed to acquire a valid state. Output Dynamic Variables become invalid when corresponding input Dynamic Variables change value.

- **dependencies**. Each output Dynamic Variable has an associated set of input Dynamic Variables. These input Dynamic Variables are expressed as dependencies of the output Dynamic Variables. This information is used by the Dynamic Integration System to build the dependency hierarchy and to control relationships such as secondary dependencies.

- **automatic validation**. Output Dynamic Variables that are invalid can be validated by the Dynamic Integration System by calling the analysis with which they are associated. This validation process is automated by the Dynamic Integration System and occurs each time an attempt to use an invalid Dynamic Variable is detected. Automated validation reverses the traditional approach that requires the programmer to call appropriate functions to acquire an output of some analysis.

The following three steps are necessary to define an individual analysis:

1. **Identify Parameters**. Identify the inputs and outputs of an analysis and model them as Dynamic Variables.

2. **Specify Dependencies**. Specify dependencies between inputs and outputs of the analysis.

3. **Connect the Analysis**. Connect the analysis to the output Dynamic Variables. This information is used by the Dynamic Integration System to validate Dynamic Variables.

Using the above guidelines, an analysis will be represented by the following structure:

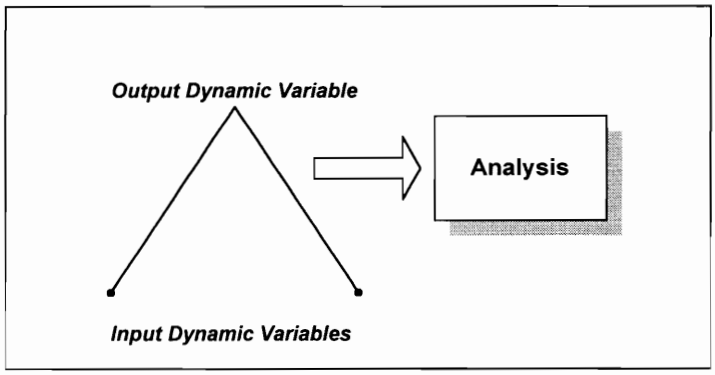


Figure 3 Individual Analysis Structure

As an example, consider an analysis that models a rectangle. The analysis is built around three parameters: area, width, and height. The area parameter is dependent on both the width and height parameters. When required, the analysis is executed to calculate a new value for area. The structure for this simple analysis is shown in Figure 4. It should be noted that while this analysis only models simple single-valued parameters, the Dynamic Integration System can also model much more complex data structures.

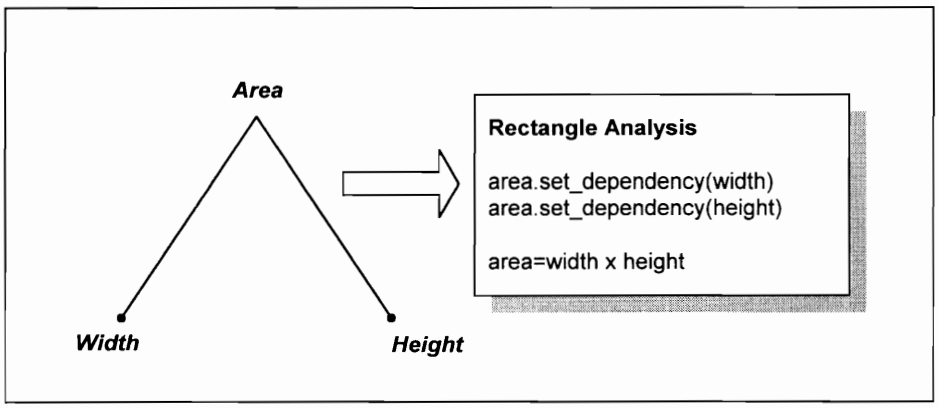


Figure 4 Example Analysis Structure

After the individual analyses have been built, integration is handled by the Dynamic Integration System. Integration occurs by using the same Dynamic Variables as inputs to some analyses and as an output of another. By linking the individual relationships specified by each analysis, the Dynamic Integration System builds a dependency hierarchy.

One advantage of modeling an analysis as a dependency hierarchy is that integration is greatly simplified. Using a hierarchy, the integration process is reduced to grafting new branches onto an existing tree. This is opposed to the current technique of introducing new functions and parameters to a design space and requiring the programmer to manage them. A benefit of Dynamic Integration is that maintenance efforts are drastically reduced because individual modules can be tested, validated, and re-integrated quickly.

Once a dependency hierarchy has been constructed, the coordination and execution of analyses are also simplified. When a particular Dynamic Variable is modified, its effects can be traced by following the path up the dependency hierarchy. This process, know as an **invalidation process**, marks other Dynamic Variables as invalid. The process of acquiring a value for a parameter is also straightforward. Accessing a parameter involves validating all dependencies - the Dynamic Variables below the parameter in the dependency tree. Using knowledge stored within the dependency tree, execution time is minimized to only validating Dynamic Variables that need to be recomputed. This

minimizes individual calls to analyses. The entire validation process is automated by the Dynamic Integration System.

A dependency hierarchy also greatly simplifies the creation of interaction modules, such as graphing modules or optimizers. Modules may interact with the parameter space by connecting to the appropriate Dynamic Variables. The modules can interact with these Dynamic Variables to determine whether they are inputs or outputs (thus controlling whether the user can interactively specify a value for it). The modules can also indirectly interact with the analyses by validating Dynamic Variables rather than directly calling each analysis. By doing so, the interaction modules can safely ignore secondary effects as they are managed by the dependency hierarchy.

Perhaps the most significant benefit of using Dynamic Variables is that interaction modules may specify which Dynamic Variables they are interacting with at run-time. Since Dynamic Variables are identified by text strings, modules can prompt the user for the appropriate identifiers. This has two purposes. First, it allows for the creation of **Generic Interaction Modules** (GIMs). Generic Interaction Modules are modules that interact with Dynamic Variables. Those Dynamic Variables are specified by the user at run-time. An example of this is a graphing module that plots any input versus any output. The second purpose is that it allows the user of an application to interact with any parameter within the application. The choices are not limited to some predefined group. As new analyses are integrated, the parameters they introduce are also exposed to the end

user without modification to the Generic Interaction Modules. This gives newly integrated analyses immediate exposure to powerful design tools.

4.4 Application Structure

The architecture provided by the Dynamic Integration System provides facilities for creating and maintaining a diverse and reusable class of applications. A typical application is composed of several modules. Some of these modules, namely the analysis modules, interact with the application by contributing branches of the dependency hierarchy. Other modules, such as Generic Interaction Modules, contribute to the application by interacting with the dependency hierarchy. These modules instantiate Dynamic Variables (nodes in the dependency hierarchy tree) and interact with those nodes.

4.5 Summary

This section has provided a brief overview of the main integration architecture used by the Optimization Workbench to communicate with the supporting engineering analysis.

5. OPTIMIZATION WORKBENCH ENVIRONMENT

This chapter presents the solution that was developed in response to the set of requirements framed in chapter 3. The system is called the *Optimization Workbench*, and contains an original approach to optimization integration, design data organization, user interfacing and results visualization.

5.1 Functional Overview

In response to the problem as it was framed in chapter 3, the Optimization Workbench contains a wide range of solutions to the requirements of the optimization community. The highlights of the system include the following:

- **Built on the Dynamic Integration System**

  The Optimization Workbench offers flexibility for integrating optimization routines with engineering analysis models. By using the Dynamic Integration System, the Optimization Workbench needs only to interact with Dynamic Variables. Specific knowledge of any one particular analysis module is unnecessary. Any model that can be integrated into the base Dynamic Integration System can have immediate access to all the functions of the Optimization Workbench.

- **Rapid Problem Formulation**

  By constructing a unique user interface to the Dynamic Parameters within the problem configuration, the user can rapidly put together a detailed problem description for the chosen optimization algorithm to process.

- **Process Feedback**

  Once the problem is formulated, the user then can get immediate, animated feedback as the algorithm alters the design. The user can select from a high level of output, or minimal output. Furthermore, the design variables, the constraints, and the merit function are all linked to individual graphical output devices that the user can monitor. If a variable is at its prescribed boundaries, or a constraint is violated, the user knows immediately.

- **Control of the Algorithm**

  The user can change the method used to compute the design changes quickly and easily. As mentioned earlier, the ability to change the algorithm serves to broaden the understanding of the results. In addition, the user can control the detailed aspects of each algorithm. For example, the detailed numerical values for items such as the convergence tolerance, the constraint boundary thickness, or the maximum number of iterations can be accessed and changed by the user.

- **Configuration Management for Optimization**

  The most unique set of functions contained in the Optimization Workbench are the tools to manage and coordinate multiple problem formulations. The user can store multiple optimization formulations, or **configurations**, within one **project**. The underlying software is constructed in such a way to handle multiple problems and their results efficiently.

- **Visualization of Individual Results**

  Another important feature of the Optimization Workbench is the collection of results history visualization tools. All changes to all variables in the configuration are recorded and stored in a configuration history object. When the algorithm is completed, the user has complete control over the results and can prepare a report or a graph showing this information. In addition, through a set of unique data analysis tools, the user can view the entire set of results for as many design variables as he has on one plot.

- **Multivariate Analysis of Results**

  As mentioned above, a unique set of tools has been constructed to reduce the dimensionality of complex, large sets of design data results. This technique is a nonlinear mapping of n-dimensional data into either 2 or 3 dimensional data for visualization.

5.2 Software Design

Appendix A. provides a description of the software development process used to design and construct the system presented here. Appendix A. also provides a description of all the components of the actual system. This section will give an overview of the major components of the final system, and how they work together. The following sections in this chapter will provide a more detailed description of how these major components provide the necessary functionality to the user.

The Optimization Workbench consists of two major subsystems, the **Configuration Controller** subsystem, and the **Optimization Controller** subsystem. A low-level

software infrastructure was constructed to provide basic functionality with no interface. Then, the Motif user interface was constructed on top of the infrastructure to provide the user access to all the functions of both subsystems.

5.2.1 Overview of the Configuration Subsystem

Figure 5 shows the structure of the Configuration Control subsystem. As explained in the Appendix, the object-oriented software technique focuses the design around the identifiable entities of a problem. In this case, the identifiable entity was the formulation for the optimization problem. This formulation is defined as the **configuration**. The Configuration Controller is centered around the configuration.

Individual configurations are managed under a control object called a **Project**. The software was designed with the ability to handle multiple configurations under one project file. In this manner, the designer can prepare an entire library of optimization problems, histories, and graphs all in one project file.

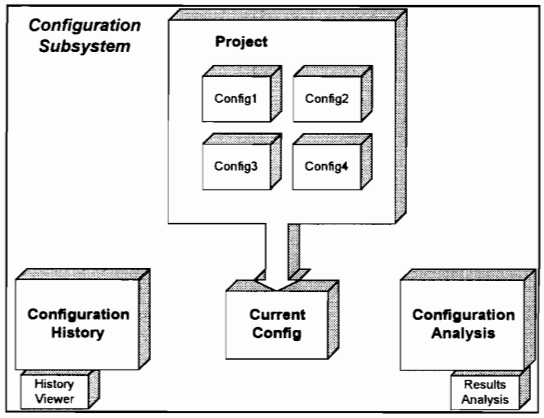


Figure 5 Configuration Control Subsystem; Infrastructure Design.

5.2.1.1 What is a Configuration?

con·fig·u·ra·tion *noun*

1. a. Arrangement of parts or elements.

   b. The form, as of a figure, determined by the arrangement of its parts or elements[88].

The term *configuration* has several meanings in the systems engineering community. In aircraft design terminology, a configuration is the physical layout and arrangement of the major components of the airframe. This term stems from the fact that a designer *configures* an aircraft design.

In a similar way, the developer of an optimization problem *configures* a problem formulation. The configuration is the arrangement of elements such as the design parameters, the constraint functions, and the merit function. Furthermore, the specification of the boundaries of the parameters to be changed, the numerical values of the constraints, or the settings for the optimization algorithm all contribute to the configuration.

A "configuration" is used here as a specific problem formulation for submission to the optimization algorithm. The term configuration was chosen for the optimization interface because the problem formulation for an optimization study is a representation of the engineering model in various states, or arrangements. For instance, wing sweep is a

common design parameter. A problem formulation that uses wing sweep with upper and lower **bounds** represents a collection of all the possible arrangements of sweep.

5.2.1.2 Recording Changes to a Configuration

A configuration can record its **current state** at any time through the functions of the **Configuration History**. The Optimization Controller tells the configuration when it should record a change into its history, for instance at the end of a design iteration, or when the process is restarting with a new set of parameters.

The histories for each configuration are accessed by the user through the functions of the user interface. The user can view the histories either in graphical form, or by constructing a custom report.

5.2.2 Overview of the Optimization Control Subsystem

Figure 6 shows the Optimization Controller infrastructure. The Optimization Controller performs the optimization on the current configuration that is selected by the user. This current configuration is supplied by the Configuration Controller. The Optimization Controller uses the Dynamic Parameters in the configuration to access the analysis. In this manner, the Optimization Controller can have several optimization methods available. The interface to each Optimization Method is standard, thus a variety of methods are available to interact with the analysis. The Optimization Controller can also

interface with different configurations in a standard way. Any method within the Optimization Controller can act on any configuration within the Project.

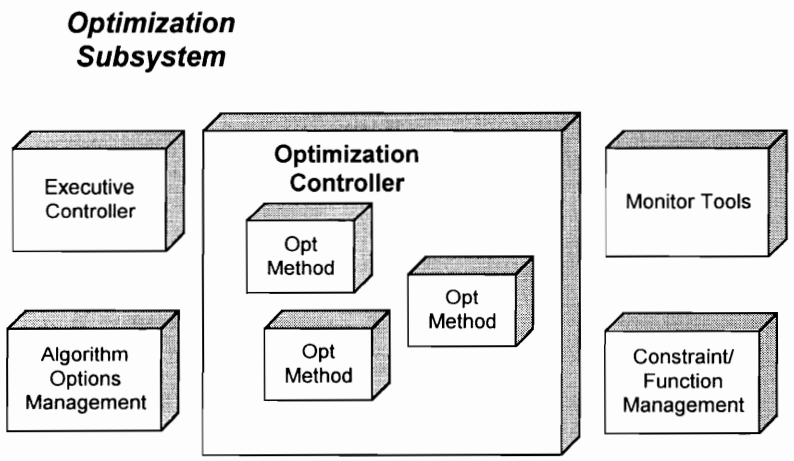


Figure 6 Optimization Control Subsystem; Infrastructure Design.

5.3 Utilization of the Dynamic Integration System™

The Optimization Workbench utilizes the functions of the Dynamic Integration System to provide access to the system analysis. This analysis, or engineering model, is used by the optimization algorithm for performance evaluation of the system. The following section outlines the major features of the Dynamic Integration System that are utilized by the Optimization Workbench.

5.3.1 Links to Analysis are Made Via Dynamic Variables

As mentioned in the previous sections, the Configuration Controller supplies the problem setup, or configuration, to the Optimization Controller subsystem. The base elements for the design parameters, constraints, and merit function in the configuration are **Dynamic**

Parameters. These parameters are variables that the user selects from the reg analysis in the Dynamic Integration System. Once selected and added to the configuration, the Dynamic Variables retain their links back to the analysis methods that can compute new values for them. Figure 7 shows how the Dynamic Variables are added to a configuration, yet retain their links.

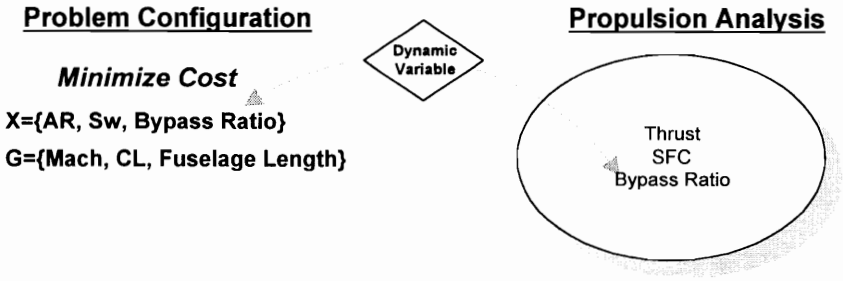


Figure 7 Configuration Retains Links Back to Analysis through Dynamic Variables.

5.3.2 Problems are formulated using Dynamic Variables

As mentioned in chapter 3, a common difficulty in performing multidisciplinary optimization is the access and integration of information from the various different analysis disciplines. Using the Optimization Workbench, a developer of an optimization problem can select variables from different analysis, or disciplines, and add them to a common configuration. This approach offers the designer the ability to extract representations from different, yet coupled components. For example, two design parameters for a problem configuration might be the lift coefficient and the engine bypass ratio. Each variable is linked to a different discipline, however, they can both be selected and added to the problem formulation. Figure 8 illustrates this concept.

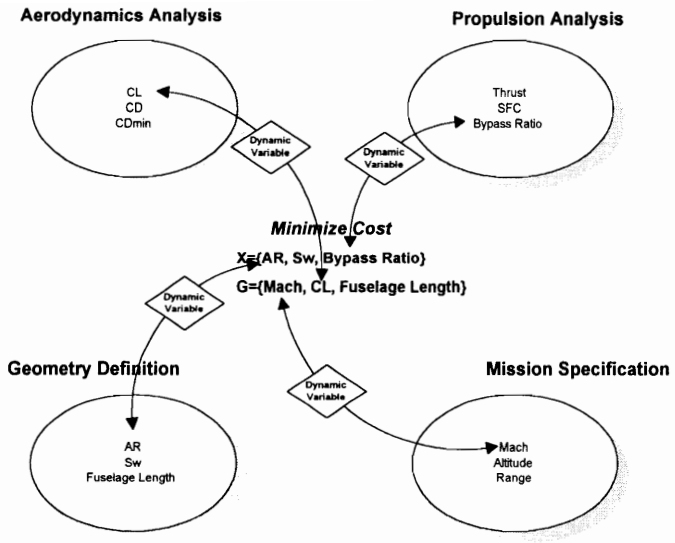


Figure 8 Design Parameters Can Be Selected From Different Disciplines and Added to a Common Configuration.

5.3.3 Independent Software Development:

Using the Dynamic Integration System, independent **analysis components** can be constructed and tested. For example a simple aerodynamics analysis capability would be constructed as an analysis component. This component would be responsible for computing lift and drag coefficients given a set of flight conditions, Mach, angle of attack, etc.

Once these components are built, they can be accessed by any **Generic Interaction Module.** An analysis component is accessed through a Dynamic Variable. Generic Interaction Modules operate solely on Dynamic Variables, hence, a common interface to each analysis component for the interaction modules is the set of Dynamic Variables for that analysis component.

Multiple analysis components can be connected through a larger set, or **hierarchy**, of Dynamic Variables. The integration occurs dynamically and only when variables from different analysis components are selected.

The Optimization Workbench was constructed to take advantage of this capability and offer generic interaction with any analysis component. One advantage of constructing independent analysis components is that analysis can be built without knowledge of how to integrate it with an optimizer. The Optimization Workbench operates on Dynamic Variables that can be registered with any analysis component. This is what makes it truly a generic application capable of servicing with any analysis.

5.4 Process Formulation and Feedback Tools

5.4.1 Functions

The formulation tools are three input box windows in the main optimization popup window. They provide the user with the ability to select input variables and output variables for use in the selected configuration. The function of these interface tools is to allow the user to setup the configuration. The process tools provide the user with feedback of the activities of the Optimization Controller as it operates on a selected configuration.

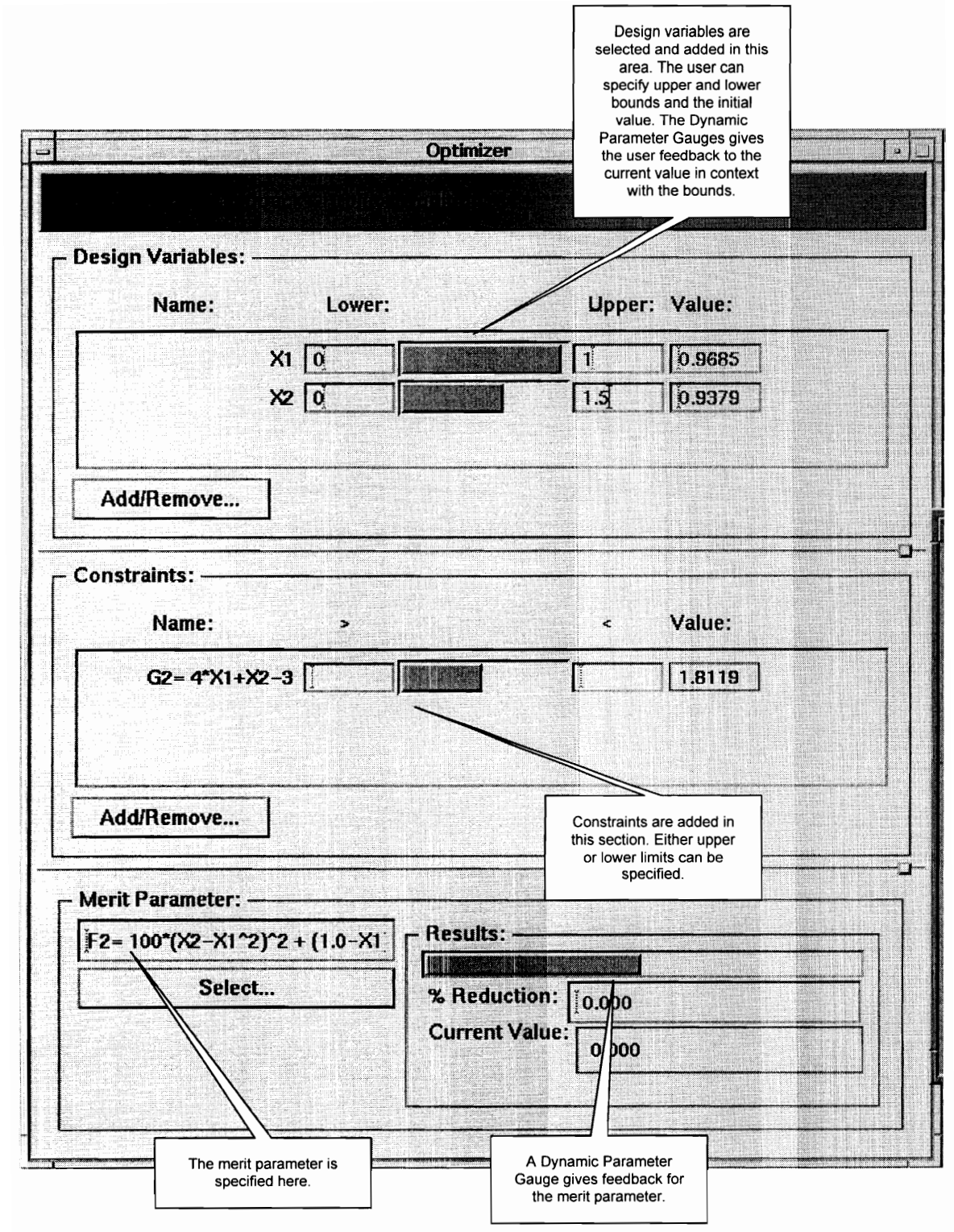


Figure 9 Main Problem Configuration Window With Several Variables Added to the Current Configuration.

5.4.2 Features

The following sections describe the features of the main configuration window.

5.4.2.1 Parameter Addition from Dynamic Registry

The parameters that are presented to the user are from the Dynamic Integration System **registry**. This information is a catalog of all the Dynamic Variables that are registered with the system. Figure 10 shows a typical Registry of information for the available design variables. This registry is accessed by selecting the ADD/REMOVE option for the design variables, constraints, or the merit function.

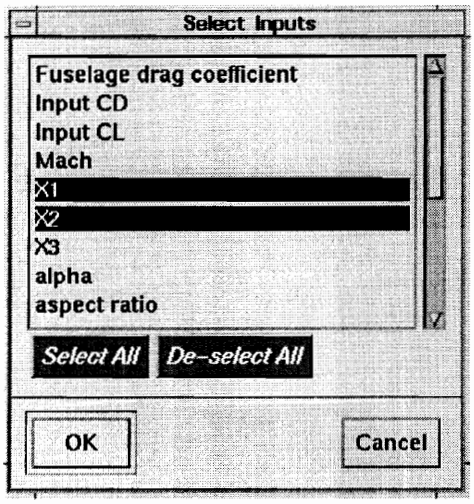


Figure 10 Design Variables are Selected from the Registry of Dynamic Integration System Variables.

5.4.2.2 Dynamic Parameter Gauge: Run-Time Graphical Animation of Design Changes

A bar-gauge format, combined with color cues provides the user quick feedback to the state of each design parameter. This interface element is known as the **Dynamic**

Parameter Gauge. Figure 11 shows the three possible states for the Dynamic Parameter Gauge.

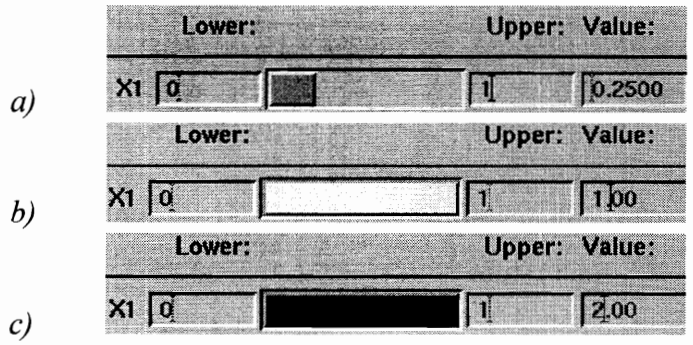


Figure 11 Dynamic Parameter Gauge:
a) Feasible Value (green) b) At upper bound (yellow) c) Beyond Bound (red).

When the gauge is green, the parameter is a feasible value for the design. The yellow bars show variables that are at their respective limits, and a red bar indicates that the parameter is infeasible, or that the limits have been exceeded.

If a configuration contains more than one design parameter, the gauges are stacked, as in Figure 12. Stacking multiple gauges gives a quick scan of which variables are at limits. This arrangement of multiple gauges is well understood and documented in the human factors studies of monitoring large amounts of information. A quick scan of a single gauge shows what state the variable is in. A quick scan of all the variables stacked up shows which variables "stick out" or are at their limits.

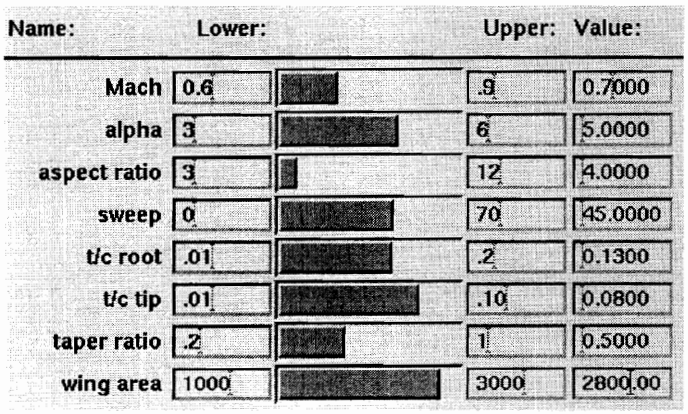


Figure 12 Multiple Dynamic Parameter Gauges are Stacked for Easy Interpretation of States.

Animated activity of the parameter gauge gives the user cues to the most sensitive, or active design variables. At each iteration, all the gauges are updated. When a configuration is being optimized, the user monitors the stack of design parameters. The most graphical activity usually occurs on the highly sensitive design variables. As the user watches the changes made to the design, he can get an indication as to which variables influence the design the most.

5.4.2.3 Merit Function Feedback

The merit function interface tools are located at the bottom of the main Optimization Workbench window. They include a merit gauge similar to the Dynamic Parameter Gauge, and several numeric fields for presenting metric values for the results. The merit gauge graphically shows the merit function as it is minimized/maximized. At each iteration, when the merit function is evaluated, the merit feedback gauge is updated.

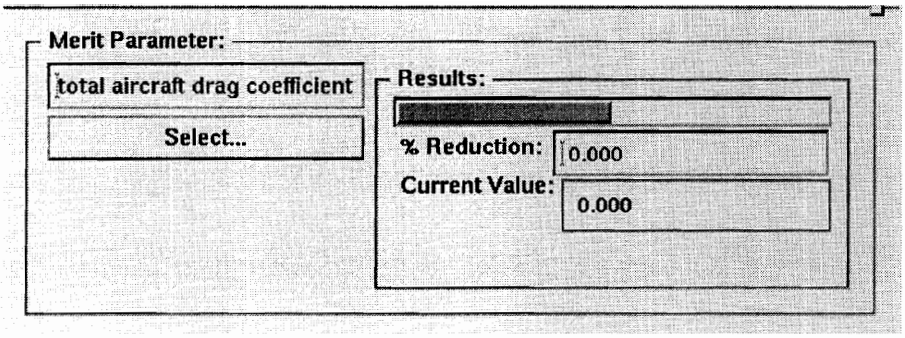


Figure 13 Merit Function Feedback Interface.

5.4.3 Scope

The scope of the formulation tools is defined in the problem formulation and configuration construction phases of the design process. The formulation tools are also used in problem re-formulation and additions of new configurations. These tools are to be used when executing an optimization case on a selected configuration

5.5 Optimization Method Control Tools

5.5.1 Functions

The tools shown in this section help the user set up and deploy an optimization algorithm. The documented methods used in this section are all algorithms from the Design Optimization Tools (DOT) toolkit.

5.5.2 Design Optimization Tools (DOT)™

The Optimization Workbench uses a main library of optimization algorithms. This FORTRAN library is the Design Optimization Tools (DOT) library. DOT is a popular

suite of algorithms capable of performing constrained and unconstrained minimization on both linear and non-linear systems.

5.5.3 Features

5.5.3.1 DOT Setup Window

The DOT Setup window is under the OPTIONS menu of the main Optimization Workbench. This popup window accesses the interface for the options of the DOT methods. Figure 14 shows this window with the method option menu highlighted, displaying the available algorithms. The following main items are currently available on the DOT Setup Window:

- METHOD SELECTION

The method option gives the user a choice of five different numerical algorithms.

1. *BFGS Variable Metric*
2. *Fletcher Reeves Conjugate Gradient*
3. *Modified Feasible Directions*
4. *Sequential Linear Programming*
5. *Sequential Quadratic Programming*

- OUTPUT LEVEL

The output level option specifies the amount and frequency the algorithm provides output to the Monitor window. The options range from 0 to 7.

- MERIT OPTIONS

The user is provided with a choice of options for how the merit parameter will be processed. The user can select from either Minimize, Maximize, or Converge to a target value. If the user specifies that the merit function be converged to a target value, the user must also specify the target value in the accompanying number field.

5.5.4 Scope

The scope of these tools is limited to altering the setup for the DOT library for use as the main optimization method under the Optimization Controller subsystem.

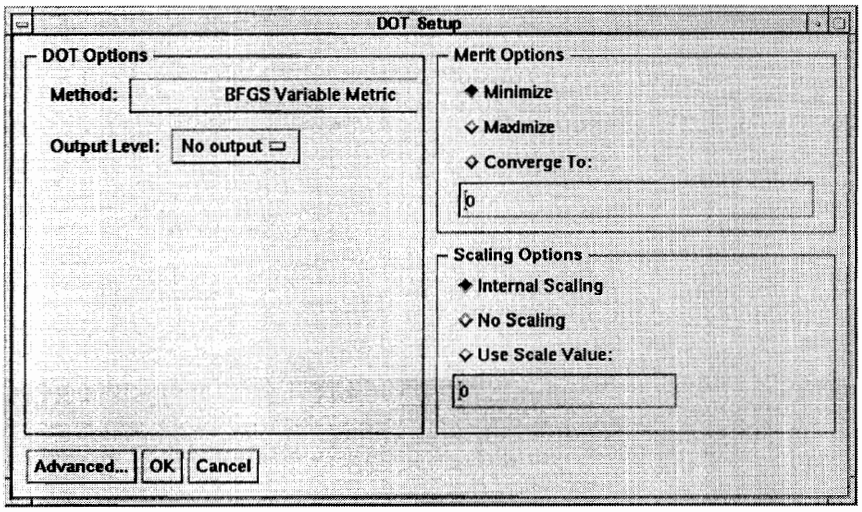


Figure 14 DOT Interactive Setup Window.

5.6 Configuration Management Tools

5.6.1 Functions

The function of the configuration management tools for the Optimization Workbench is to provide the user with the ability to formulate, execute and store multiple problems under one project file.

5.6.2 Features

When an optimization algorithm is used to find a certain minimum/maximum of a model, greater insight into the results is attempted by re-formulating the problem with either a new objective function, or a different set of design variables or constraints. The tools within the Optimization Workbench facilitate rapid reformulation of the problem through the use of multiple configurations.

5.6.2.1 The Project Editor

Multiple configurations are organized under one project. A "configuration" is defined as a specific problem formulation for submission to the optimization algorithm. A configuration consists of a selected set of design variables, constraints and an objective function. By using multiple problem formulations assembled under one project, the user can maintain the history of the design and recall the results for any problem formulation. This information is used later to present the user with a report or graph of the results and for data analysis.

The Optimization Workbench provides the capability to add, delete, or edit configurations within a project. Figure 15 shows the Project Editor, a tool which is used to access the management facilities for a certain project.

- NEW

A user can add a new configuration by selecting the NEW Option under the Project Editor. This will create a new configuration and add it to the current project file.

- MAKE CURRENT

This option takes the selected configuration from the scrolled list and makes it the current configuration in all the supporting windows.

- DELETE

Removes the selected configuration from the project.

- CONFIG HISTORY

Opens the configuration history viewing tools for the selected configuration. Each configuration stores all the results, or histories, of the actions the optimization algorithms performed.

- PROJECT HISTORY

Opens the history viewing tools for the optimal results collected from each configuration in the current project. This tool allows the user to see the optimal solution from each problem setup and plot them in context with each other. In

addition, this module contains features to plot all the optimal sets of design variables on one plot using the nonlinear mapping techniques explained in the next chapter.

- PROJECT INFORMATION

Displays a description of the current project.

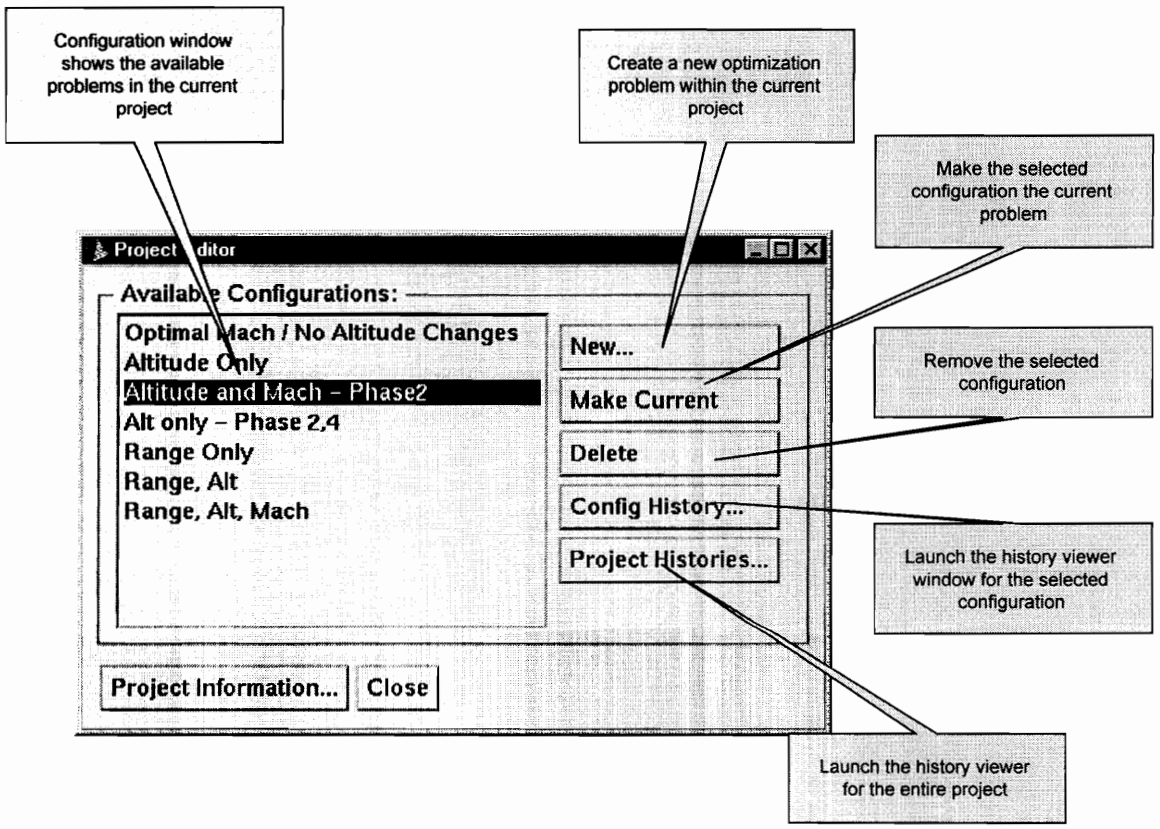


Figure 15 Project Editor for Configuration Control.

5.6.3 Scope

The scope of the Project Editor encompasses all the activities for adding, removing, and editing multiple configurations. The Project Editor also serves as a launching tool for the configuration history tools.

5.7 History Report Tools

5.7.1 Functions

The capabilities shown in this section allow the user to prepare a custom report from the data collected and stored in the configuration management section.

5.7.2 Features

The features found in this section include an interface to a report generation tool. The tool is comprised of two windows, one that offers report options, and a second that appears with the report after the user specifies the options and selects the report to be generated.

5.7.2.1 Preparing a Report

Figure 16 shows the report options window. The user can specify to see any range of iterations, and any combination of the design variables, constraints, or merit function. Once the appropriate options has been selected, the user then selects GO REPORT, and a second popup window appears with the report in a scrolled area. The user has the option

of saving this report to a file, printing the report, or simply closing the window. Figure 17 shows the completed report.

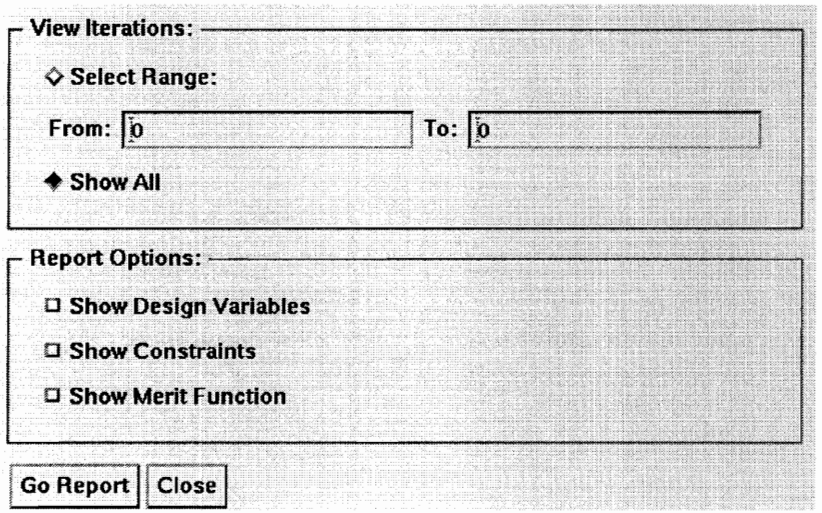


Figure 16 Report Generation Setup Interface.

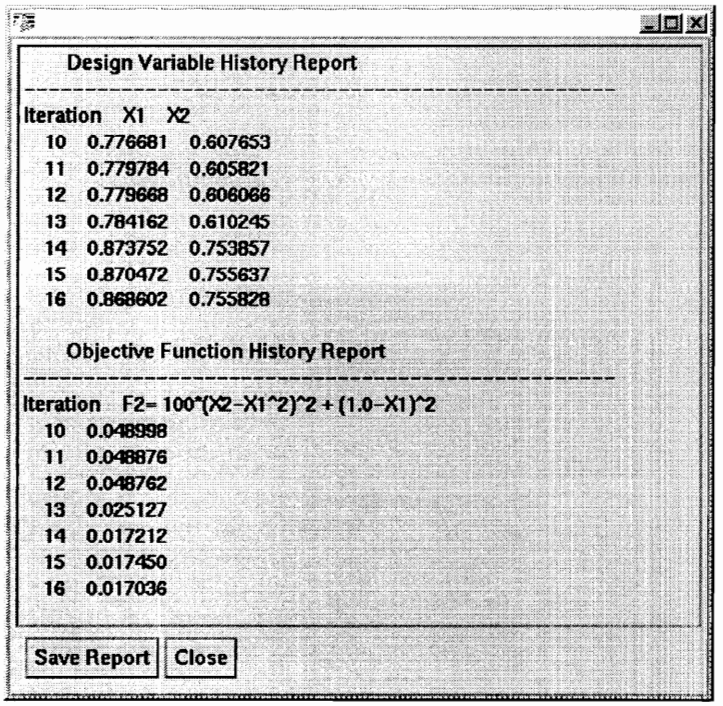


Figure 17 Final Report Appears in a Separate Scrolled Window.

5.7.3 Scope

The scope of this module encompasses all functions to prepare a textual report containing the information in the currently selected configuration. The report that is generated is static and does not supply information to any other module of the program.

5.8 Change Histories Plotting Tools

This section presents the facilities in the Optimization Workbench to prepare 2D and 3D plots of the convergence histories for the currently selected configuration. In addition, the tools for performing nonlinear mapping of all design variables onto one plot are presented.

5.8.1 Functions

The capabilities shown in this section allow the user to prepare a custom plot from the data collected and stored in the configuration history. A standard configuration history plot can be constructed, or a composite plot of all the design variables, mapped to a two dimensional plot can be constructed.

5.8.2 Features

The following sections describe the features of the configuration history plotting interface, and the associated windows.

5.8.2.1 Constructing a Convergence History Plot

A convergence history plot can be made for any design variable. The list of all design parameters, constraints, and the merit function are displayed in a scrolling window in the graph setup interface. The user simply selects any combination of variables from the scrolled list and selects XY PLOT. Figure 18 shows the graph setup window with several variables in the scrolled window ready to be plotted. Figure 19 shows the completed plot..

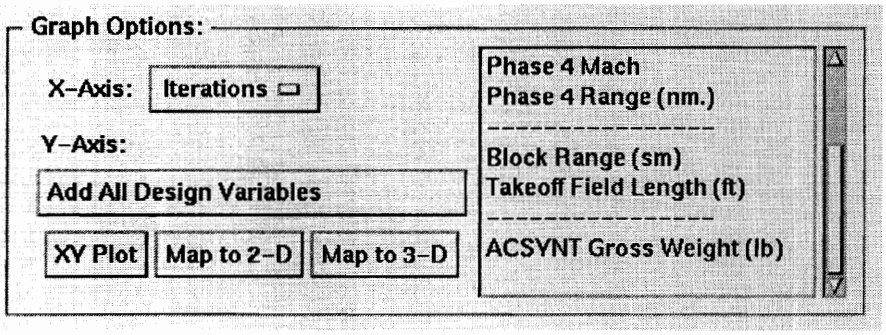


Figure 18 Convergence History Plotting Setup Window.

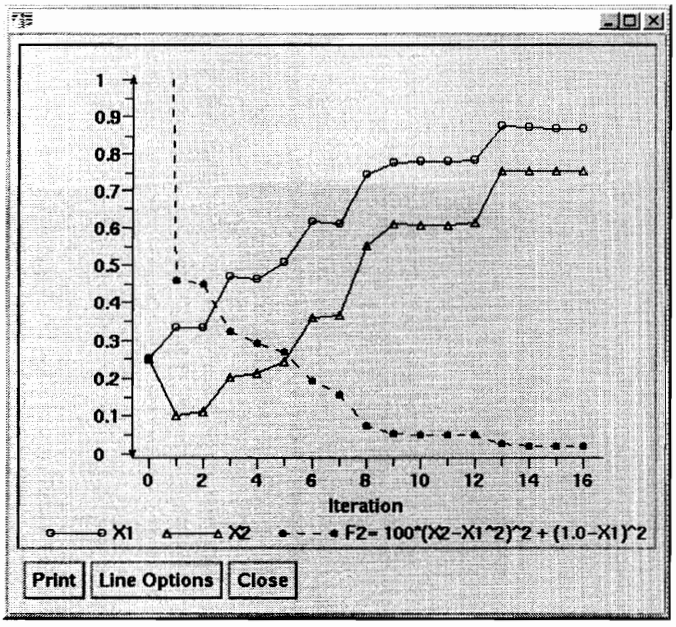


Figure 19 Typical Convergence History Plot.

5.8.2.2 Construction a Convergence Scatterplot Using Nonlinear Mapping

The user can also view the convergence of all the design variables on one plot through the use of the built-in nonlinear mapping algorithms. This tool reduces the dimension of the selected set of data to 2, and then constructs a scatterplot showing each iteration mapped into a non-dimensionalized two-space. These methods are explained in more detail in the next chapter.

To prepare the plot, The user selects the desired design parameters in the current plot, then selects MAP TO 2D. If the mapping is to be done in 3D, the user selects the desired set of variables and then selects MAP TO 3D. The plot will then appear in a separate window. Figure 20 shows an example of a set of 4 design variables reduced to two dimensions for plotting.

5.8.3 Scope

The scope of the convergence history plotting tools remains at viewing and presenting the elements within a single configuration history.

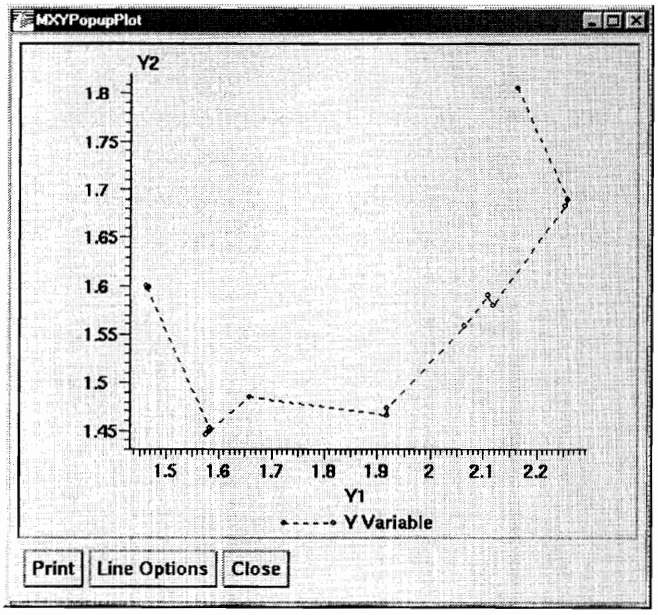


Figure 20 Nonlinear Mapping in Two Dimensions of Convergence History of 4 design variables.

5.9 Project Histories

5.9.1 Functions

The capabilities shown in this section allow the user to prepare a plot from the optimal results collected and stored in the each configuration of the project. These plots can be constructed from the optimal objective functions, or by using a nonlinear mapping technique to plot all the design variables on a scatterplot.

5.9.2 Features

When the PROJECT HISTORIES window is launched from the Project Editor, a new window is opened that allows the user to plot the optimal results from each configuration within

the project. This tool is very useful for presenting the results of multiple optimization cases and illustrating the relative effects of changes in constraints, design variable bounds, optimization methods, or similar parameters.

When the PLOT OPTIMAL OBJECTIVES selection is chosen from the Optimal Histories module, a plot is constructed using the converged value from the histories of each configuration in the project. Figure 22 shows this plot and illustrates how each value on the plot is taken from each configuration in the project.

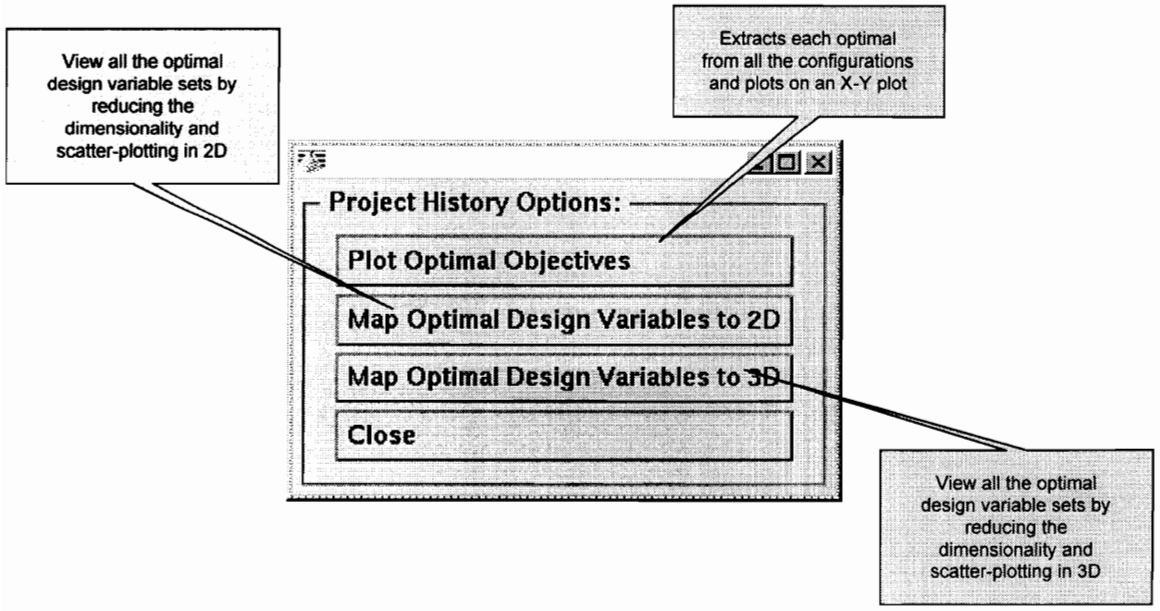


Figure 21 Description of the Optimal History Interface

5.9.3 Scope

The scope of the optimal history plotting tool is limited to accessing and presenting information contained in a project file. The optimal history module can access multiple configurations and provide insight into the data among different configurations.

5.10 Summary

This chapter has provided a description of the capabilities of the Optimization Workbench including application of a new and unique data visualization approach to optimization results. The next chapter will provide a mathematical description of the algorithm used to create these visualization techniques.

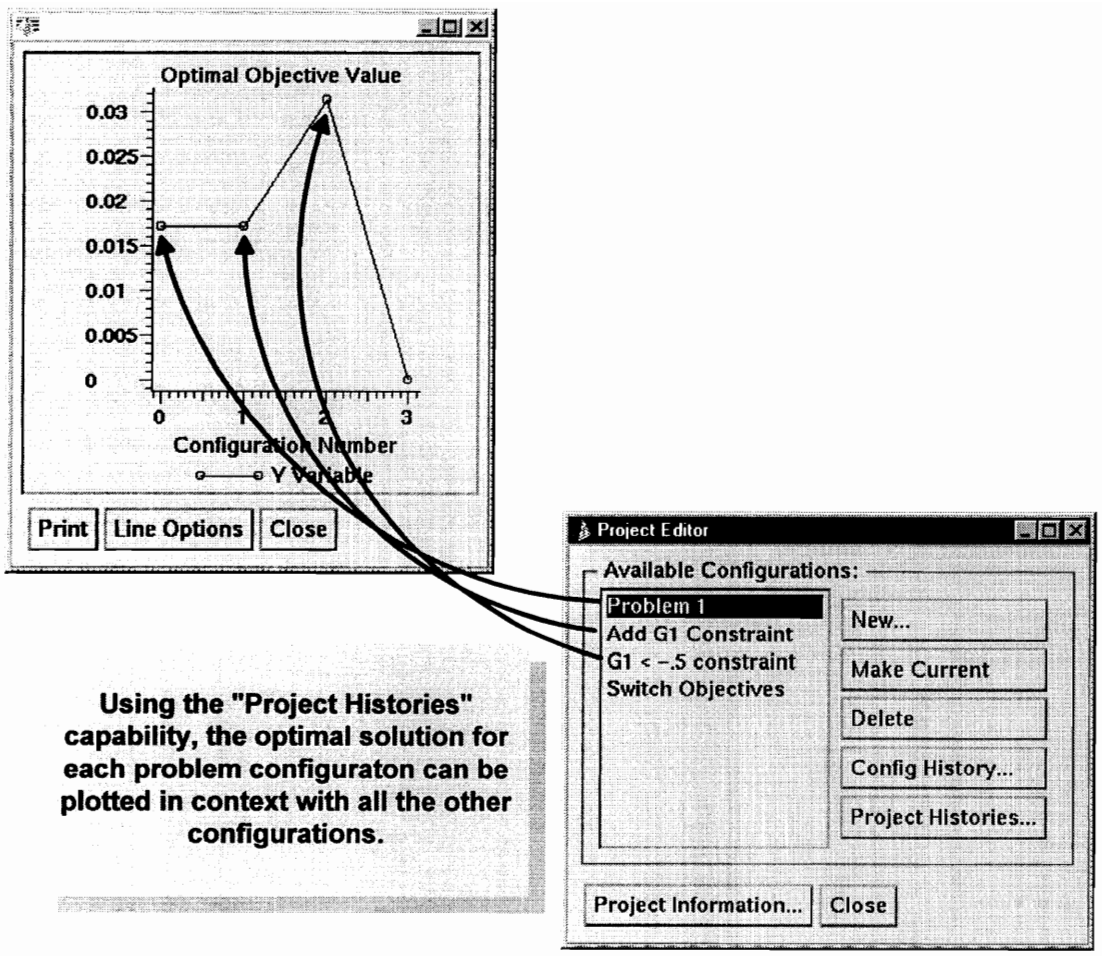


Figure 22 Illustration of Plotting Multiple Optimal Objectives Within a Project.

6. MULTIVARIATE DATA ANALYSIS OF OPTIMIZATION RESULTS

This chapter describes an approach to analyzing multivariate data and presenting a visual tool for optimization results. This tool is helpful for performing results visualization such as exploring multiple optimization cases or viewing a concise history of large amounts of design data. The concepts that drive the design optimization approach are best illustrated visually. This chapter will develop a method that is particularly well suited to inspecting families of optimization results and revealing information that will help the designer arrive at a more robust design.

Configuration analysis can be performed at two levels. First, the results generated for the successful optimization of a single configuration can be analyzed. Secondly, multiple configurations can be analyzed to provide comparison between results from different problem formulations.

6.1 Results Visualization

Michaud and Modrey [27] first showed promise in the area of multivariate data analysis of optimization results. Their application of nonlinear mapping gave the user an

understanding of the results through a cluster analysis technique. This was the first step towards understanding the structure of the optimization problem results.

By utilizing the underlying configuration-based organization of the Optimization Workbench, results visualization of a single problem can be provided to the user. Because all necessary information is derived from the configuration, analysis can be accomplished regardless of the optimization algorithm employed. The entire set of results for each problem is stored in the configuration for analysis and visual inspection. Access to the change history of each problem is provided through utilities in the configuration controller system.

6.2 Results Analysis Using Nonlinear Mapping

The following algorithm is used to reduce the dimensionality of the results such that a composite plot can be made for greater insight into the problem. This algorithm was first explored in 1969 for viability [89]. The algorithm is based on a mapping from L-space to d-space (where $d = 2$ or 3 for visualization requirements).

The history of results for a configuration with L design variables can be described by the vector,

$$\mathbf{X}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iL} \end{bmatrix} \tag{1}$$

Where i is the iteration number within the history of the results and L is the dimension of the original data (in the case of the vector of design variables, L is the number of design variables.)

We now wish to reduce the dimension of each vector of results by mapping each L-dimension vector onto a vector of dimension d. We define the following vector, **Y** as the new, reduced dimension set of data,

$$Y_i = \begin{bmatrix} y_{i1} \\ \vdots \\ y_{id} \end{bmatrix} \tag{2}$$

To perform the mapping, an error function is defined that computes the error between the two computed interpoint distances between each set of vectors.

The interpoint distance between the vectors in the original set of data, or the **X** vectors, can be defined as follows,

$$D_{i,j} = \left\{ \sum_{k=1}^{L} \left[x_{ik} - x_{jk} \right]^2 \right\}^{1/2} \tag{3}$$

Likewise, the interpoint distance between the vectors in the reduced dimension set of data, or the **Y** vectors, can be defined as follows,

$$d_{i,j} = \left\{ \sum_{k=1}^{d} \left[y_{ik} - y_{jk} \right]^2 \right\}^{1/2} \tag{4}$$

Finally, the error function is formed from the difference between the two computed interpoint distance vectors,

$$E = \frac{1}{c}\sum_{i<j}^{N}\frac{\left[D_{ij} - d_{ij}\right]^2}{D_{ij}} \tag{5}$$

and where,

$$c = \sum_{i<j}^{N} D_{ij} \tag{6}$$

With the error function defined, we now set out to select the components of each **Y** vector such that the error is minimized. This is accomplished by performing an optimization problem using the components of each **Y** vector as design variables and the computed error function as the objective function. The algorithm is initiated with a random set of data for the N vectors, Y. Each element of every Y vector then becomes a design variable for the suboptimization problem.

Figure 23 illustrates an example structure of a 3D set of data. This data can be mapped to 2D using the mapping algorithm presented here. In this case, the 3D structure will be retained through the reconstruction of the interpoint distances in 2D. Figure 24 shows the structure of the 2D target data and how the interpoint distances are computed. A composite error function is computed by expressing the differences between each interpoint distance in a summation,

$$E = \frac{1}{\left(D_{12} + D_{13} + D_{14} + D_{23} + D_{24} + D_{34}\right)}\left\{\frac{\left[D_{12} - d_{12}\right]^2}{D_{12}} \quad \cdots \quad \frac{\left[D_{34} - d_{34}\right]^2}{D_{34}}\right\} \tag{7}$$

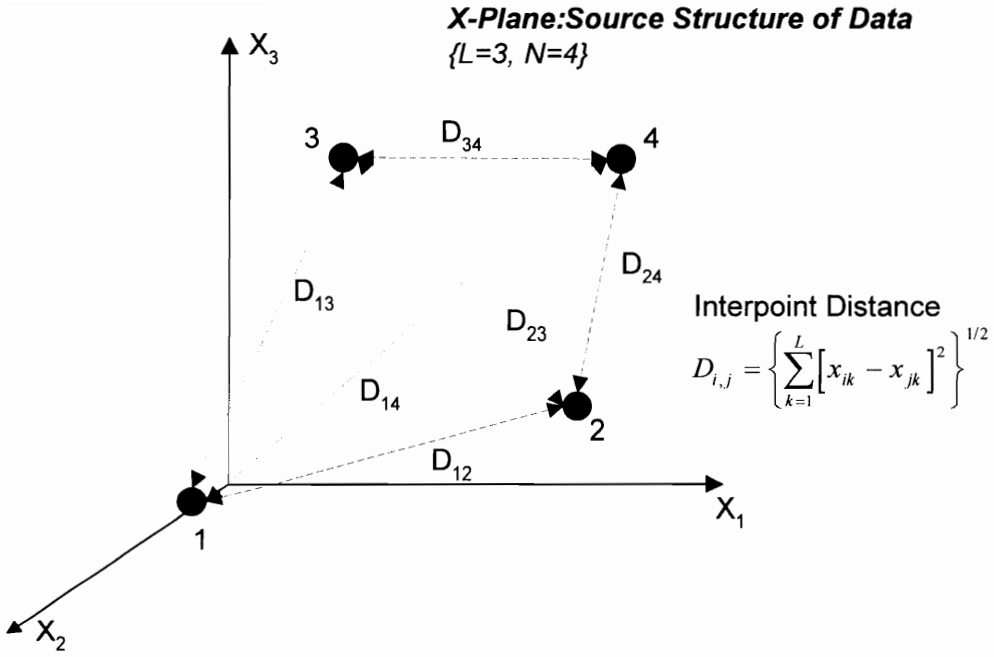


Figure 23 Geometric Interpretation of 3-Dimensional Original Source Data for Mapping.

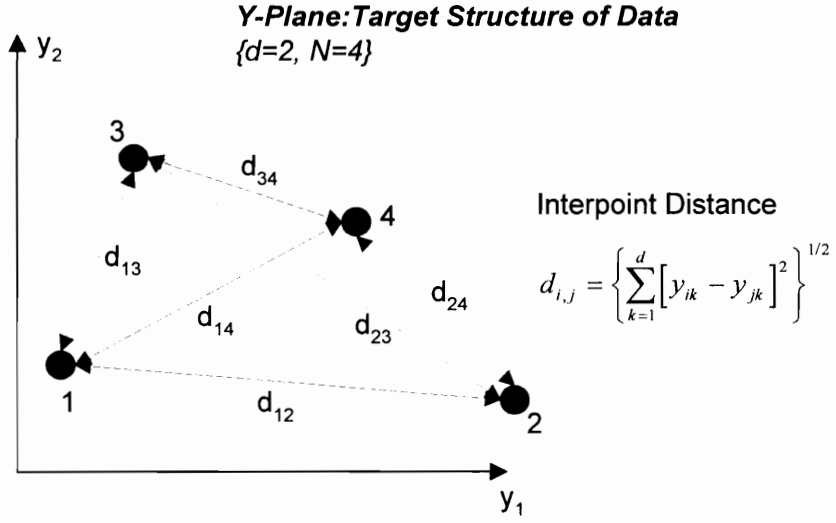


Figure 24 Geometric Interpretation of 2D Target Structure of Mapped Data.

A suboptimization problem is posed using the X,Y coordinates of the data shown in Figure 24 as design variables. Each coordinate of every Y element influences the merit function because the summation includes each interpoint distance. Thus, the optimization problem has enough degrees of freedom to arrive at the minimum error solution. The minimum error solution is the accurate reconstruction of relative interpoint distances.

6.2.1 Algorithm Verification: 2D Linear Mapping

In this first example, a set of nine linear 2D points are mapped to another set of 9, initially random, points. This example illustrates the concept of using this algorithm to map from one set of vectors to another using the interdistance error as the figure of merit. The initial set of linear data is given by,

$$X = \left[\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \cdots \begin{Bmatrix} 8 \\ 8 \end{Bmatrix} \right] \tag{8}$$

The initial set of vectors used as the mapping target is given by,

$$Y = \left[\begin{Bmatrix} Random(0-10) \\ Random(0-10) \end{Bmatrix} \begin{Bmatrix} Random(0-10) \\ Random(0-10) \end{Bmatrix} \cdots \begin{Bmatrix} Random(0-10) \\ Random(0-10) \end{Bmatrix} \right] \tag{9}$$

The resulting optimization problem then utilizes the 18 elements of Y to minimize the error between the measured interpoint distance of the Y points compared to the interpoint distance of the original X points. The unconstrained optimization problem is posed as,

$$Min\ Error(Y) \tag{10}$$

$$Y = \begin{Bmatrix} P1_{y1} & P2_{y1} & P3_{y1} & P4_{y1} & P5_{y1} & P6_{y1} & P7_{y1} & P8_{y1} & P9_{y1} \\ P1_{y2} & P2_{y2} & P3_{y2} & P4_{y2} & P5_{y2} & P6_{y2} & P7_{y2} & P8_{y2} & P9_{y2} \end{Bmatrix}. \quad (11)$$

Figure 25 shows the results of the 2D linear mapping. The linear structure of the original X vector set is immediately visible in the Y vector set. Starting with random values as initial guesses for the Y vector set, the optimization problem converges to the structure shown in Figure 25 in 62 iterations. The final composite interpoint distance error is 3.37e-6.

It is important to note that the orientation of the target data is independent of the Y1 and Y2 axes used to map the data. In other words, since the algorithm only focuses on the relative interpoint distances, and the initial point is a random location, the structure of the data is constructed in an arbitrary orientation. However, the relationship between points is the overall objective, hence the final mapped structure recreates the linear behavior of the original data.

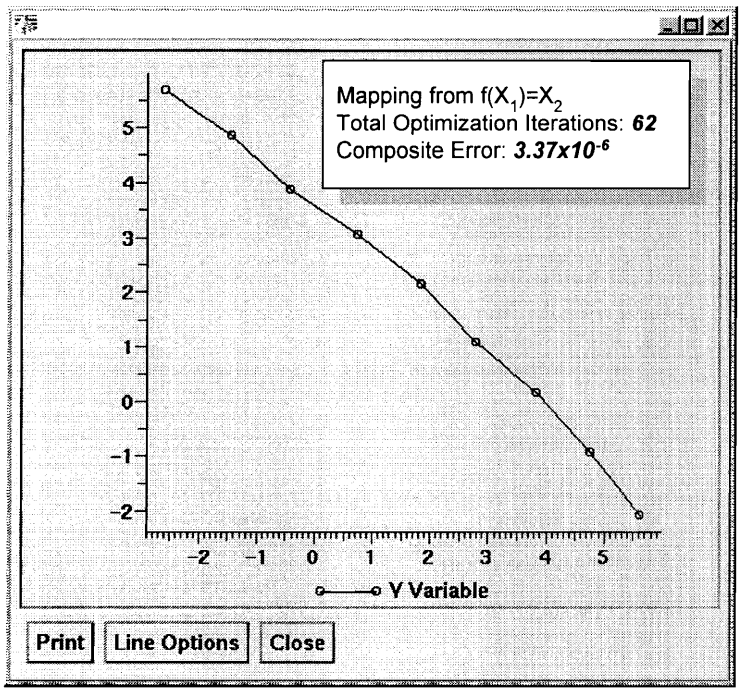


Figure 25 Test Mapping of Nine Linear Data Points using Random Data Points as Starting 2D Vectors.

6.2.2 Algorithm Verification: 3D Linear Mapping of Helix Data

The following example will map an original 3D set of helix data to a separate set of 3D points using the developed mapping technique. This example will illustrate reconstruction of structured data points in three dimensions. The original set of vector is given by,

$$X = \left[\begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} \quad \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \quad \cdots \quad \begin{Bmatrix} x_n \\ y_n \\ z_n \end{Bmatrix} \right] \tag{12}$$

where,

$$\begin{aligned} z_n &= \frac{\sqrt{2}}{2} \cdot n \\ n &= [0 - 30] \end{aligned} \tag{13}$$

and,

$$x_n = \cos(z_n) \tag{14}$$

$$y_n = \sin(z_n). \tag{15}$$

The initial targeted data set was seeded with a range of random numbers. The original source data was computed as shown in the equations above. The optimization problem was then performed to construct the original structure of the data. Figure 26 shows the results of the mapping to the targeted data set. For this example, there were 20 original vectors of data. There were three design variables per target vector, hence the resulting optimization problem to construct the helix structure was a 60 design variable problem.

Again, notice how the algorithm reconstructs the structure of the data independent of the orientation of the axis. Since the algorithm relies solely on the relative interpoint distances and not the absolute position as referenced from a fixed axis, the orientation of the mapped data with respect to the 2D index axis is irrelevant. Thus, in the 3D plot of the mapped helix data, the X, Y, and Z axis has no meaning to the mapped data. The axes simply serve as unit directions for each dimension.

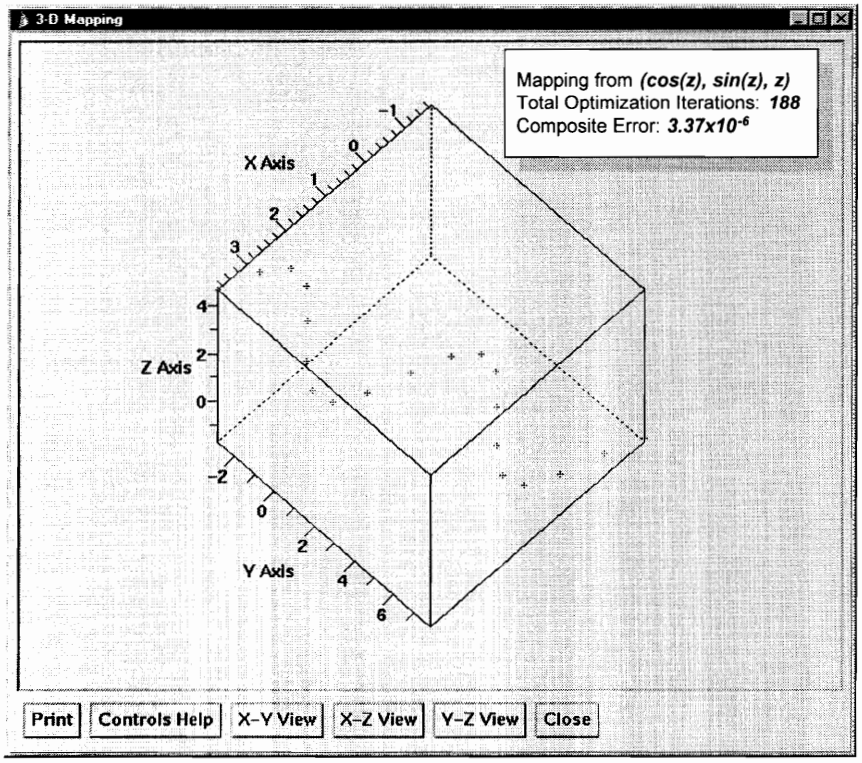


Figure 26 General Nonlinear Mapping: 3D Random Data Mapped to 3D Helix Data for Algorithm Verification.

6.2.3 Practical Applications

The dimensional reduction technique is useful for presenting optimization results. One practical application is a convergence history that can be plotted for multiple design variables. A single plot can be created that represents the convergence of the entire design. Each design variable in the configuration represents a dimension in the mapping problem. To form a plot in 2 dimensions, the values for each design variable at each iteration are mapped to a 2 dimensional vector for the same number of iterations.

Another application is to use the dimensional reduction technique to plot the results of multiple optimization problems. The optimal set of design variables for multiple configurations is plotted in 2 or 3 dimensions. The resulting plot is a concise overview of all the optimal solutions found within a project of configurations.

Figure 27 shows a collection of optimal design variable sets reduced in dimension and plotted in 2D. For this specific example, each design variable set contained 9 original dimensions. Each optimization case had a different set of bounds for the variables used. As seen from the plot, 5 of the 6 cases were located in a cluster, indicating little change in the design. However, the last configuration is radically different, and is shown as an outlier of the cluster of data from the first 5 cases.

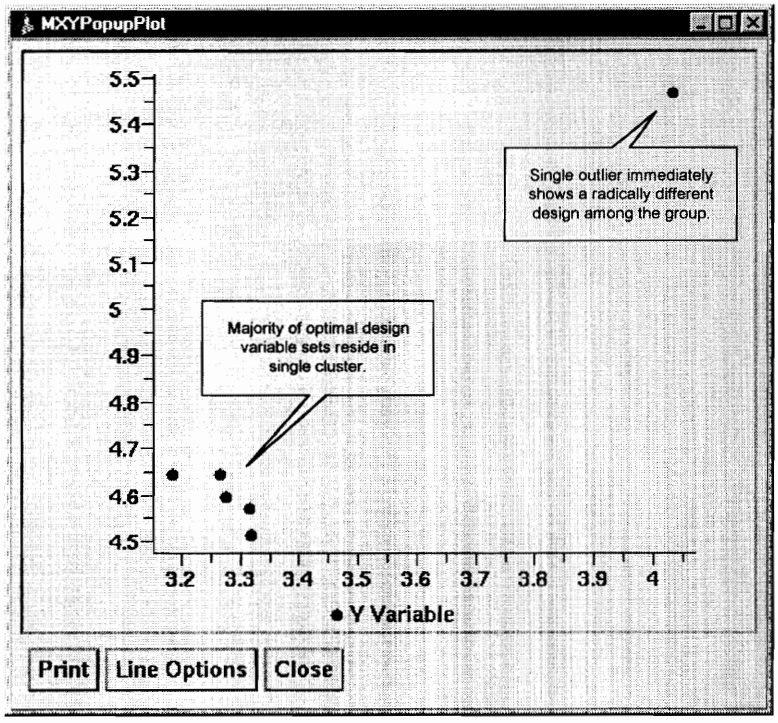


Figure 27 Scatterplot Showing Optimal Design Variable Sets for 6 Different Cases.

6.2.4 Summary

The proposed dimensional mapping for scatterplot analysis provides the user with a tool that can visually illustrate the structure of optimization results data. This tool has proven useful for inspecting the large amounts of data generated during the optimization process by providing a composite view of multiple optimization cases plotted in context with each other. Of particular use, this tool can provide convergence history insight for one problem configuration, or an entire family of problems as stored in a project.

Development of multivariate analysis techniques for optimization configuration analysis is a developing research effort for application in the Optimization Workbench. The configuration control system has been designed and developed with functionality growth considerations. As a result, additional methods to perform similar analysis can be integrated in a straightforward manner.

7. SAMPLE APPLICATIONS

This and the next chapter will illustrate the functionality of the Optimization Workbench by providing four sample problems. This first example is a classic, two-variable algebraic problem. This sample illustrates the concepts of using the project editor and creating multiple problem configurations to help organize the design effort. The second example is a single-discipline problem of optimizing a wing for aerodynamic efficiency. This problem illustrates the concepts of using analysis components, adding constraints, and using modules of the Dynamic Integration System Design Workbench™.

The next chapter will utilize ACSYNT and an additional analysis module to perform detailed aircraft design optimization studies. This example addresses interacting with the core functions of the original ACSYNT program to perform an aircraft sizing study. Finally, the fourth example showcases the optimization of a system of different disciplines integrated through various analysis programs.

7.1 Algebraic Example

The first example was chosen because it is a classic two-variable sample problem for showcasing optimization methods. The Rosenbrock valley function [90] is often used in

the optimization literature because it represents a challenging problem for the numeric algorithms. In the context of configuration management of multiple problems, this function was chosen to illustrate formulating a first problem, executing the problem and investigating the results, then forming a new problem and solving the problem differently.

7.1.1 Rosenbrock Valley Function

The algebraic form for the Rosenbrock Valley function is given by,

$$F(X_1, X_2) = 100(X_2 - X_1^2)^2 + (1 - X_1)^2 \tag{16}$$

Figure 28 shows the contours of the function. This function is highly nonlinear and usually poses a challenge for gradient-based search methods. The optimal point to minimize F is at (1.0,1.0). There are no constraints added to this problem.

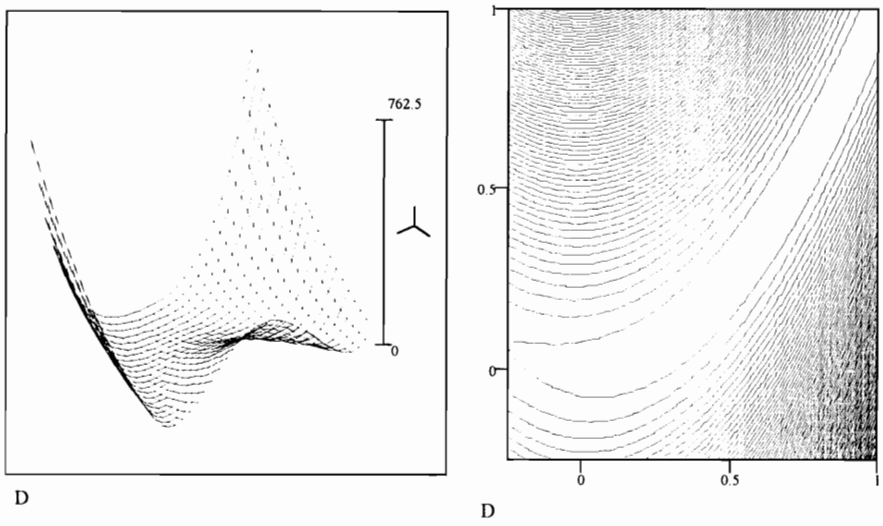


Figure 28 Rosenbrock's Valley Function.

7.1.2 Problem Formulation

Figure 29 shows the problem as it is formulated in the Optimization Workbench for upper and lower bounds of {0, 0} and {1.0, 1.0}. Since there are no constraints, the constraint input area is blank. Two Dynamic Parameter Gauges are activated for the two Dynamic Variables, X1, and X2. We have set the upper and lower bounds for both variables, and the initial values for X1 is 0.20 and the initial value for X2 is 0.40. The analysis module that supplies the Dynamic Variables for X1 and X2 was already constructed and loaded into the Dynamic Integration System for this example.

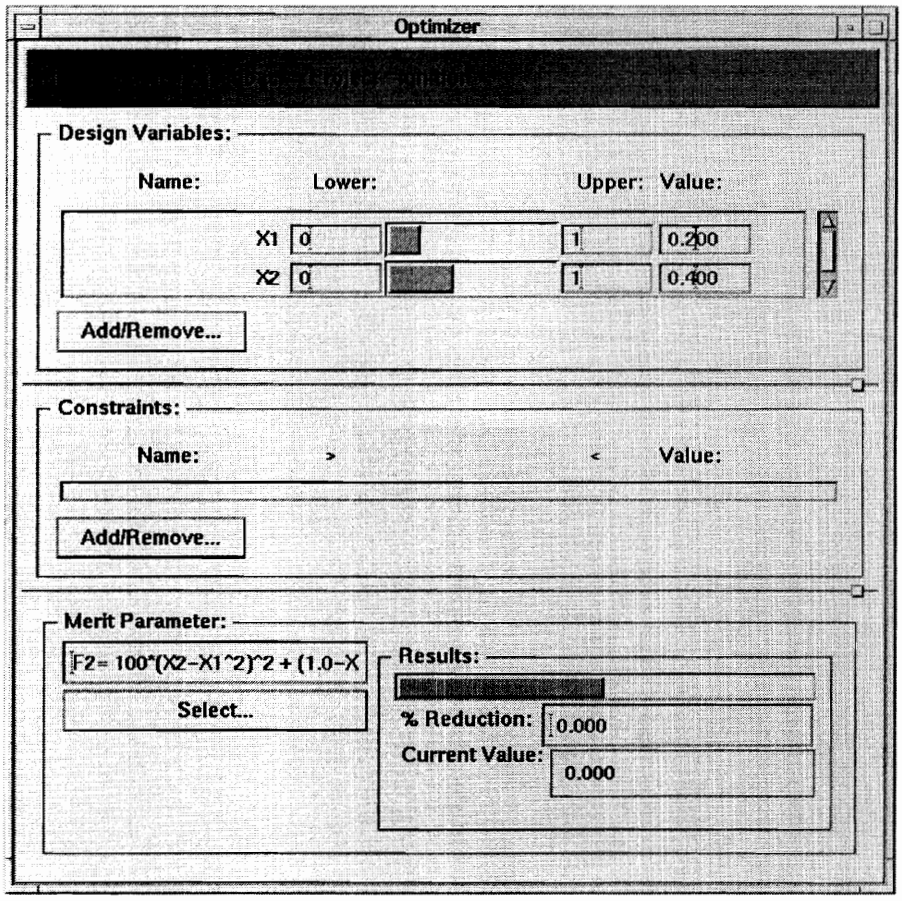


Figure 29 Rosenbrock Valley Function Formulated in the Optimization Workbench.

7.1.3 Case 1. Successful Convergence

The optimization problem was first executed using the initial point {0.20,0.40}. The BFGS unconstrained minimization algorithm was selected for this case. The COLD START option was selected from the Executive Controller and the optimization algorithm began processing the first configuration. After several iterations, the Optimization Workbench completed the problem. Figure 30 shows the convergence history for the two variables, X1 and X2, and the objective function. The correct minimum of {1.0,1.0} was found in 10 iterations.

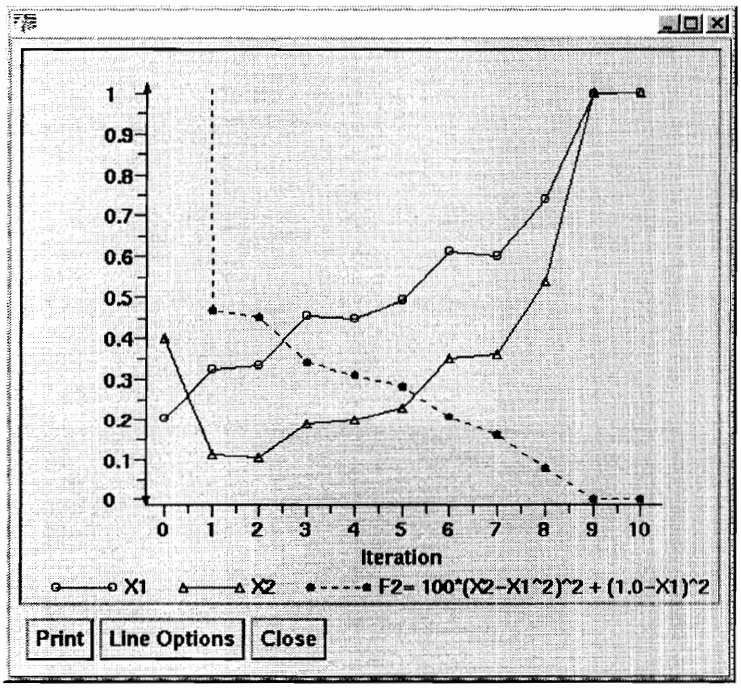


Figure 30 History for Successful Convergence of Rosenbrock Optimum.

7.1.4 Optimality Analysis

For this sample problem, we can check the optimality by checking the Kuhn-Tucker conditions. Since this example is unconstrained, at the optimum design point, the Kuhn-Tucker necessary conditions are the following:

1. The optimal design variable set, X^*, is feasible.
2. The gradient of the objective function equals zero.

The first condition is satisfied because both the design variables are within the limits of the prescribed upper and lower bounds. Since there are no constraints, the bounds of the design variables are the only bounds for feasibility.

The second condition simply states that the optimum point must reside at a stationary point of the objective function. We can check the gradient of the Rosenbrock Valley function analytically.

$$\nabla F(X_1, X_2) = \begin{bmatrix} -400X_1(X_2 - X_1^2) - 2(1 - X_1) \\ 200(X_2 - X_1^2) \end{bmatrix} \tag{17}$$

$$\nabla F(1,1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{18}$$

Since the gradient of the objective function is zero, we have found a minimum point for the Rosenbrock Valley function.

We must now check the second order necessary condition. This is accomplished by verifying that the Hessian of the objective function is positive definite or positive semidefinite at the optimal point. For the Rosenbrock function, the Hessian is given by,

$$H(X_1, X_2) = \begin{bmatrix} -400(X_2 - 3X_1^2) + 2 & -400X_1 \\ -400X_1 & 200 \end{bmatrix}. \tag{19}$$

The Hessian at the optimal point, (1,1) is computed as follows,

$$H(1,1) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}. \tag{20}$$

By computing the principal minors for this matrix, we determine that the Hessian at the optimal point is positive definite and that the second order necessary condition is satisfied.

7.1.5 Case 2. Failure to Converge from Different Initial Point

In an attempt to determine the robustness of an algorithm, it is important to be able to change the starting point for the optimization and still converge to the same optimum. For this example a new problem configuration with different starting points was created so we can compare the results with the original initial point case.

The Rosenbrock problem was restarted with a new initial point of {0.25,0.25}. The new value for the starting point is entered into the value area beside the Dynamic Parameter Gauge. Figure 31 shows the convergence history for this sample starting point. After 12 iterations, the values converged to 0.80 and 0.65, respectively. The convergence criteria for the algorithm was met, however, the problem failed to converge to the correct values of {1.0, 1.0}. The extreme non-linearity of the problem has given the algorithm search direction difficulties, and the algorithm cannot proceed. The value of the objective function was greater than zero, indicating a solution less than optimal compared to the Kuhn-Tucker necessary conditions.

At this point, a restart strategy of displacement of the last solution coupled with an algorithm change was employed. The initial point for the problem was displaced slightly (+10%) from the previous solution to serve as a starting point for a warm start. This was done by changing the current value in the Dynamic Parameter Gauge area.

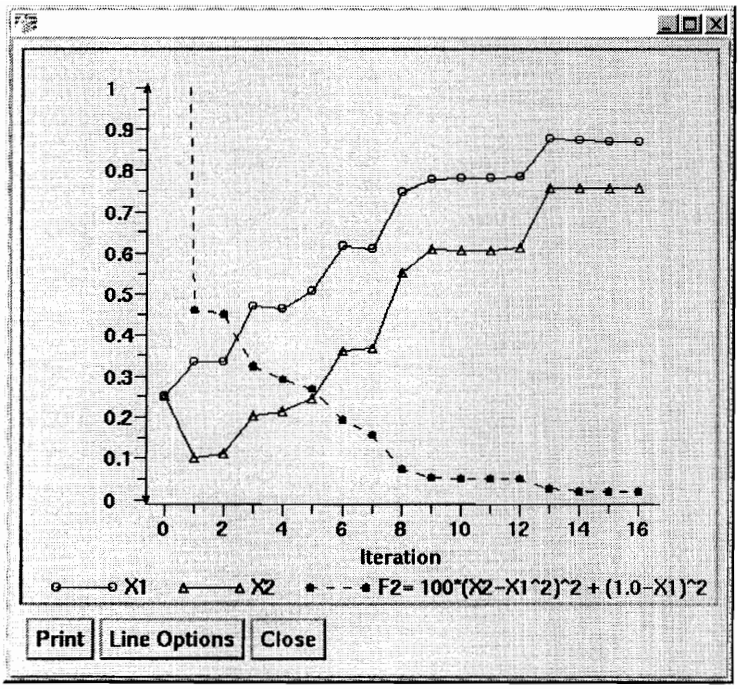


Figure 31 Failure to Converge to Optimum Given an Alternate Starting Point for Rosenbrock Function.

The optimization algorithm was changed from the BFGS method to the Sequential Quadratic Programming method by selecting the DOT SETUP option from the Optimization Workbench main window, and choosing the proper option in the METHOD option menu. The optimization process was then restarted.. Figure 32 shows the restart cases and the final convergence to the correct minimum.

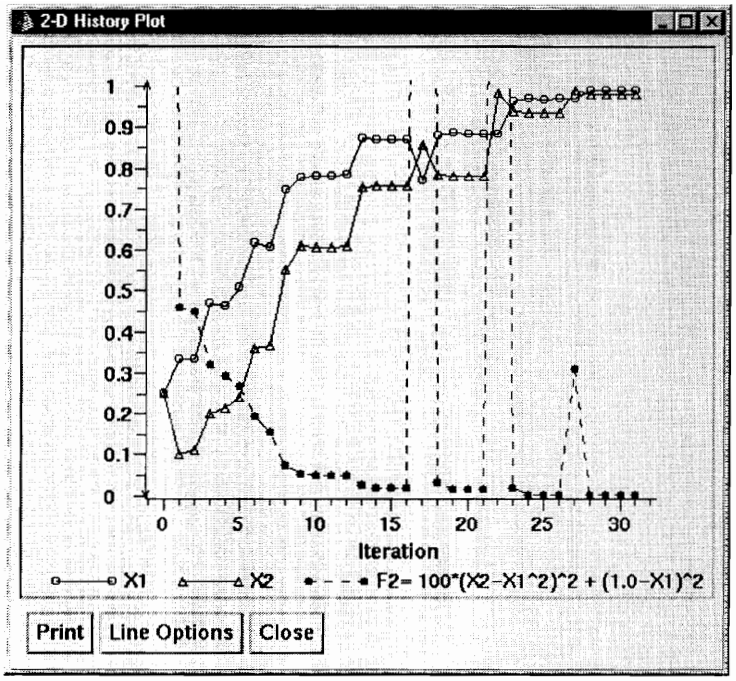


Figure 32 Restart Cases Illustrate Convergence to Minimum.

7.1.6 Mapping Convergence Histories to One Plot

As an example of how the nonlinear mapping tool can be used to gain insight into multi-dimensional problems, both design variables were mapped and plotted on one two-dimensional plot. Since there are only two design variables for this problem, the mapping will simply be a translation of the convergence history data. The mapping strategy will be a one-to-one mapping given the two sets of vectors for the original data and the mapped data. The original set of iteration vectors,

$$\mathbf{X} = \left\{ \frac{X1}{X2} \right\} \quad L = 2 \tag{21}$$

will be mapped to the set of vectors,

$$\mathbf{Y} = \left\{ \frac{Y1}{Y2} \right\} \quad d = 2 \tag{22}$$

Figure 33 shows the structure of the convergence history data, and the local regions where the search algorithms iterated, taking small steps with little changes in the design variables. The figure also shows large movements between the clusters of iterations. This is confirmed by observing the step-function appearance in the convergence history plots for both X1 and X2.

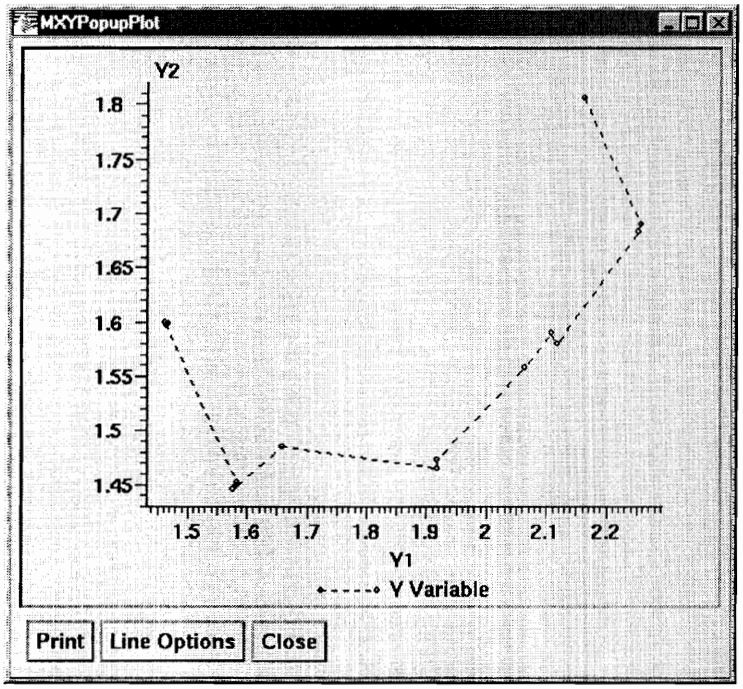


Figure 33 Mapping of Rosenbrock Iteration History.

7.2 Single Discipline: Vortex Lattice Wing Aerodynamics

The second example problem introduces a vortex lattice wing aerodynamics optimization problem with three design variables and one constraint.

7.2.1 Object-Oriented Vortex Lattice Software

An object-oriented aerodynamics algorithm was developed to serve as a continual test analysis for the tools of the Optimization Workbench. This software was developed entirely under the paradigm of the Dynamic Integration System methodology. As a result, the integration of this analysis with the Optimization Workbench is simple and straightforward. In addition, most of the variables used in the aerodynamics analysis are Dynamic Variables, so they are all available for use in the Optimization Workbench. This example problem will use several wing planform variables as the design parameters and attempt to minimize the induced drag while maintaining a lift coefficient constraint.

7.2.2 Problem Formulation

We choose the following wing planform variables to serve as design parameters for the optimization problem:

Aspect Ratio:	$AR = b^2/S$
Taper Ratio:	$TR = C_{Tip}/C_{Root}$
Wing Sweep	Sweep = Quarter Chord Sweep (deg.)

Where the wing area, S, and the root chord, C_{Root}, are held constant, and the span, b, and the tip chord, C_{Tip} are computed from the aspect ratio and taper ratio. The angle of attack and Mach number for the wing also non-zero and held constant.

The constraint formulation is simply a lower bound on the lift coefficient, C_L, set to 0.18. Finally, the objective function is the minimum induced drag coefficient, C_D. The mathematical formulation for the optimization problem can be written as,

$$\begin{gathered} Min\, C_D \\ Subject\ To: \\ 0.18 - CL \le 0 \end{gathered}$$

$$X_L \le \mathbf{X} \le X_U$$

$$X = \begin{Bmatrix} AR \\ TR \\ Sweep \end{Bmatrix}$$

Where appropriate upper and lower bounds are selected for the design variables.

7.2.3 Accessing the Analysis

The vortex lattice analysis is written using the object-oriented strategy. As such, the main object is called a **lift surface**. This object is a representation of the aircraft wing. The lift surface object contains all the methods necessary to compute the lift and drag coefficients on itself. The input variables that are used to define the lift surface are the wing planform geometry parameters. The Dynamic output variables that are computed in the lift surface

object are the lift and drag coefficients. As the geometry of the wing changes, the aerodynamic coefficients are updated with **valid** values. Using this strategy for component development, an entire aircraft design can be constructed using multiple lift surface objects with different specifications for the geometries.

The lift surface aerodynamics analysis is precompiled and integrated with the Dynamic Integration System Design Workbench for this example. In order to activate the analysis and register the Dynamic Variables with the system, however, a lift surface **object** must be created. The ANALYSIS menu from the main Workbench Controller provides the user with the ability to create a lift surface object and initialize the aerodynamic analysis. Figure 34 shows the design workbench and the registered variables after creating the lift surface object.

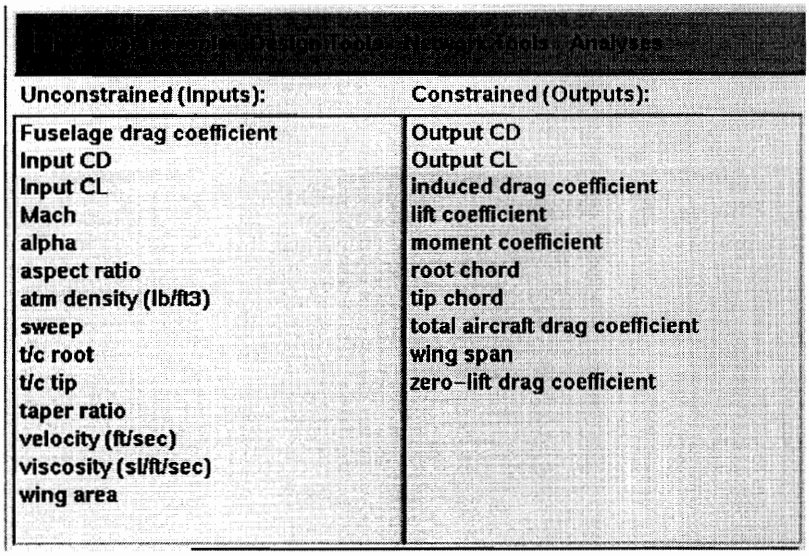


Figure 34 Lift Surface Object is Created in the Main Workbench Controller.

7.2.4 Configuration 1.

Figure 35 shows the problem formulation for this example in the main Optimization Workbench window. The C_L constraint is added and a value for the lower bound is specified. The three design parameters are shown with appropriate upper an lower bounds. For this example, the prescribed bounds for the design parameters are given in Table 1 Wing Variable Bounds:

Table 1 Wing Variable Bounds

Parameter	*Lower Bound*	*Upper Bound*
Aspect Ratio	3	12
Taper Ratio	0	1
Sweep (Deg.)	0	70

The Dynamic Parameter Gauges all indicate that the current values for the parameters are all feasible. Furthermore, the current value for the lift coefficient constraint is feasible at a value of 0.19.

Because we have introduced a constraint to the problem, the optimization method selected for this example is the Modified Method of Feasible Directions. This method is preferred for its ability to maintain feasible designs with non-linear constraints

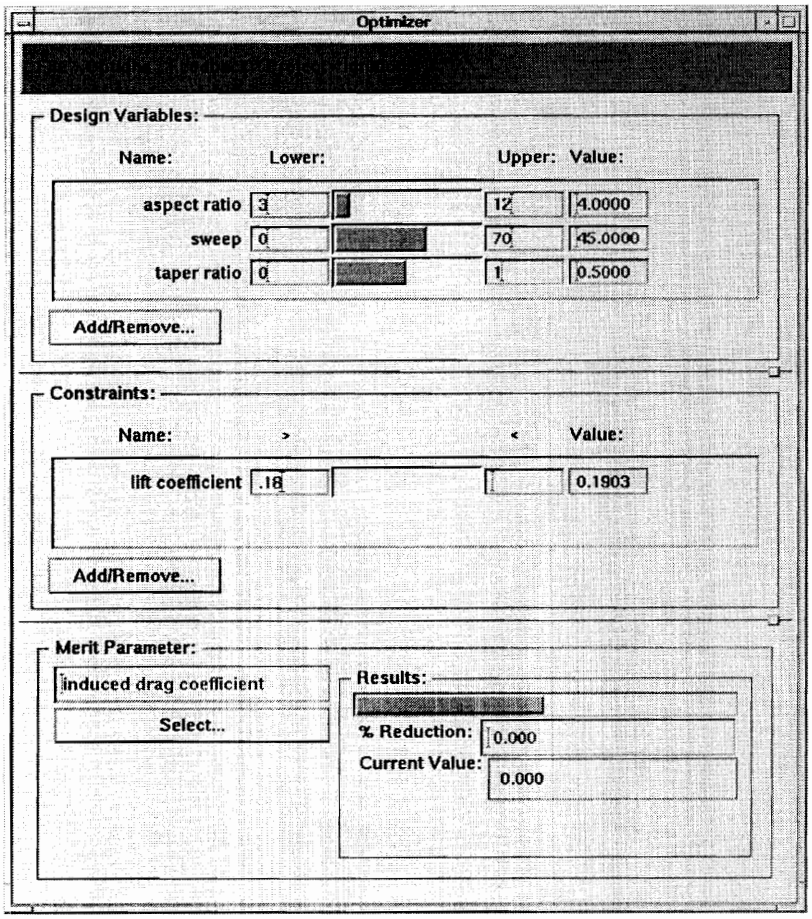


Figure 35 Lift Surface Aerodynamics Problem Formulation.

Using the illustrated starting points for the design and the prescribed methods for performing the algorithm search, the optimization method was initiated through the WARM START function of the Executive Controller.

After watching the Dynamic Parameter Gauges through the entire iteration history, several observations can be made about the sensitivity of the chosen design parameters. First, the optimizer utilized the aspect ratio more than the other two variables. After the first few iterations, the wing sweep came to rest at its upper bound, and the taper ratio

was immediately reduced to zero. The next several iterations, however, showed the aspect ratio converging to a value of 7.8. This was surprising because with only one active constraint it was expected that each variable would be close to either their respective upper or lower bounds. Using the History section of the Project Editor, several plots of the convergence histories were constructed. Figure 36 shows the history of the aspect ratio and taper ratio. As evident in the history, the aspect ratio was increased from a value of 4 to a value of 7.8, well below the upper limit of 12.

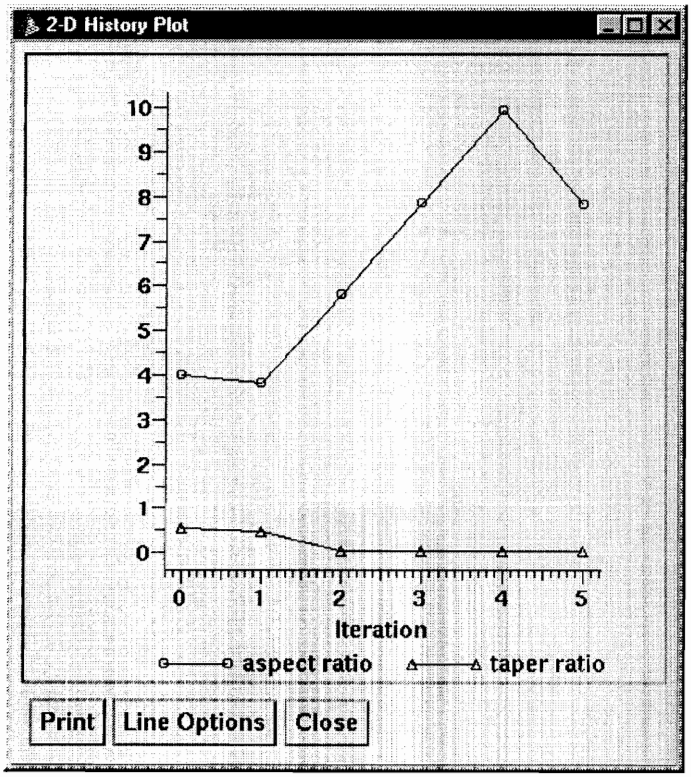


Figure 36 Convergence for the Aspect Ratio and the Taper Ratio.

After reviewing the results for the design parameters, the objective function results were plotted on the convergence history. At this point, it was obvious that the optimizer had

found a set of invalid design parameters. At the combination of design variables that the optimizer had arrived, the induced drag coefficient was computed to be a negative number. Obviously this is an incorrect value, and the tools of the Optimization Workbench and other Dynamic Integration System Generic Interaction Modules were used to explain the problem.

The first investigation involved plotting a parametric range of different aspect ratios and observing the drag coefficient. Figure 37 shows the induced drag plotted as a function of the aspect ratio for the range of the lower bound of 3 to the upper bound of 12. Clearly, the bucket formed near AR=8 provides a stable region for a minimum point. However, it is predicting negative drag, which is still incorrect.

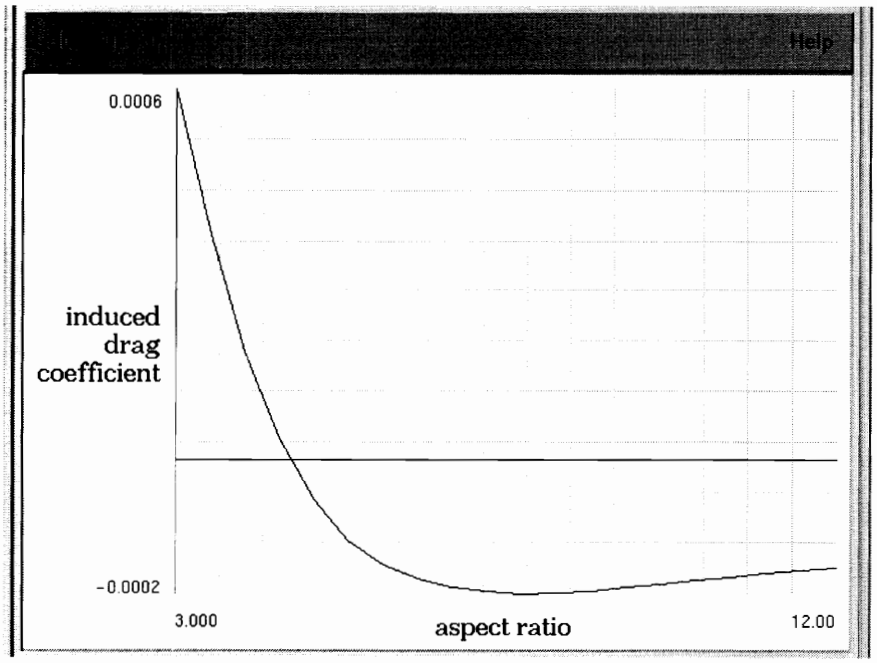


Figure 37 Induced Drag as a Function of the Aspect Ratio. TR=0.0

The next parameter investigated was the taper ratio. Figure 38 shows the induced drag plotted for a range of taper ratios. At this point, it becomes clear that when the taper ratio is a very small value (<0.05) and the aspect ratio is relatively large (>6) the induced drag that is predicted becomes a negative number. In retrospect, this can be correlated with the paneling scheme for the vortex lattice analysis. A taper ratio of 0.0 collapses the tip chord to a point, and causes instability in the paneling of the wing. Nevertheless, the tools of the Optimization Workbench and the Dynamic Integration System made it very easy to search through the design space and determine the parameter that was adversely affecting the design.

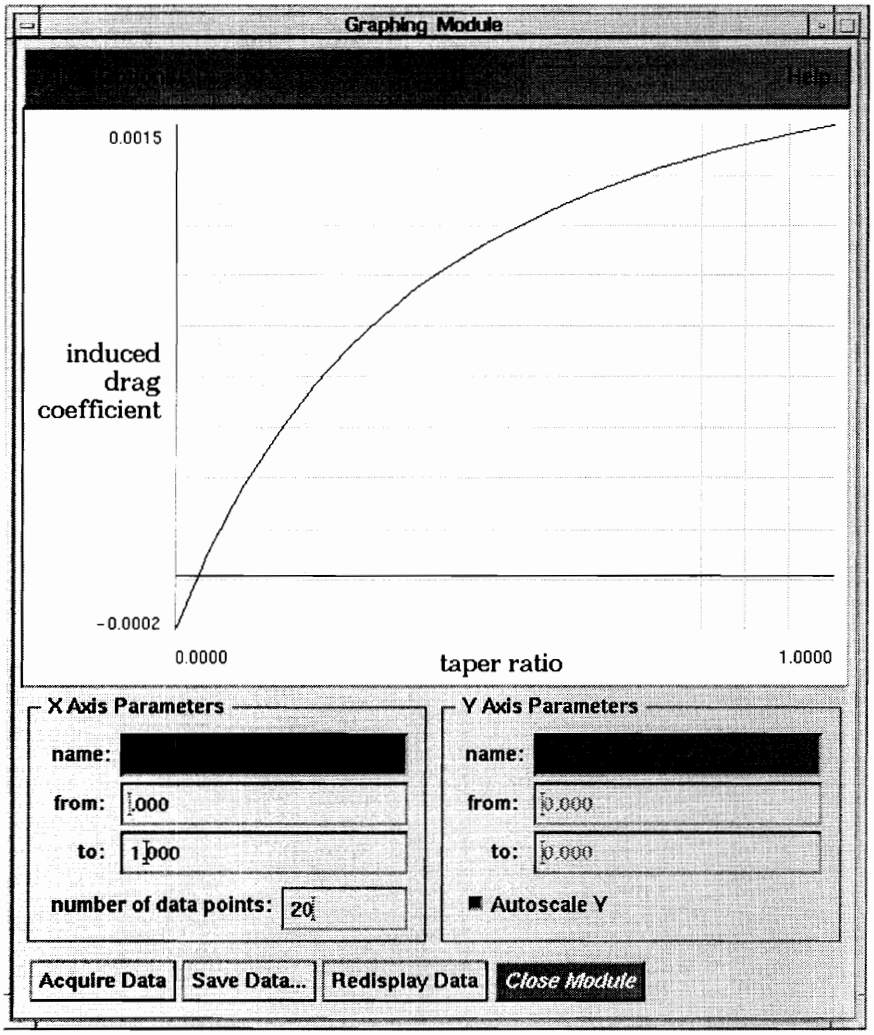


Figure 38 Induced Drag as a Function of the Taper Ratio. AR=7.8

7.2.5 Configuration 2. Changing the Limits of the Design Parameters

To determine a more valid set of parameter bounds, the taper ratio was increased to 0.2 and a plot of the drag was made over a range of aspect ratios. Figure 39 is a plot of these data. The data show that the shape and inflection of the AR curve is radically changed. It was anticipated at this point that the optimizer would select the upper bound for the aspect ratio when the new problem was minimized.

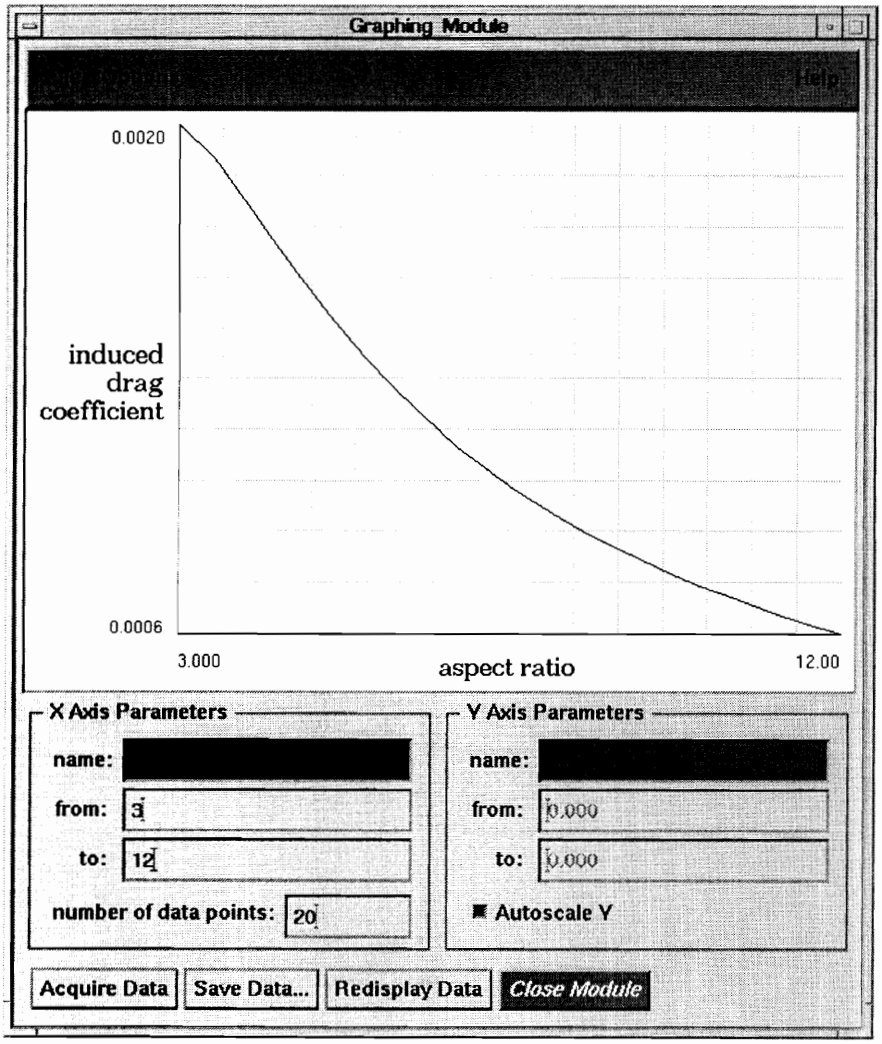


Figure 39 Induced Drag as a Function of the Aspect Ratio. TR=0.2

Again using the tools of the Project Editor, a new configuration was constructed with a new lower bound of 0.20 for the taper ratio. The problem was then restarted with this new problem configuration. As expected, the aspect ratio was increased to its upper bound, and the taper ratio was decreased to 0.20. Figure 40 shows the new convergence history for the aspect ratio. The wing sweep now was used by the optimizer to maintain the lift coefficient constraint. For the initial constraint of 0.18, the sweep was 69.23 deg., just

under the upper bound of 70 deg. As the lift coefficient increased, the wing sweep decreased as the imposed constraint demands more lift from the wing at the fixed angle of attack.

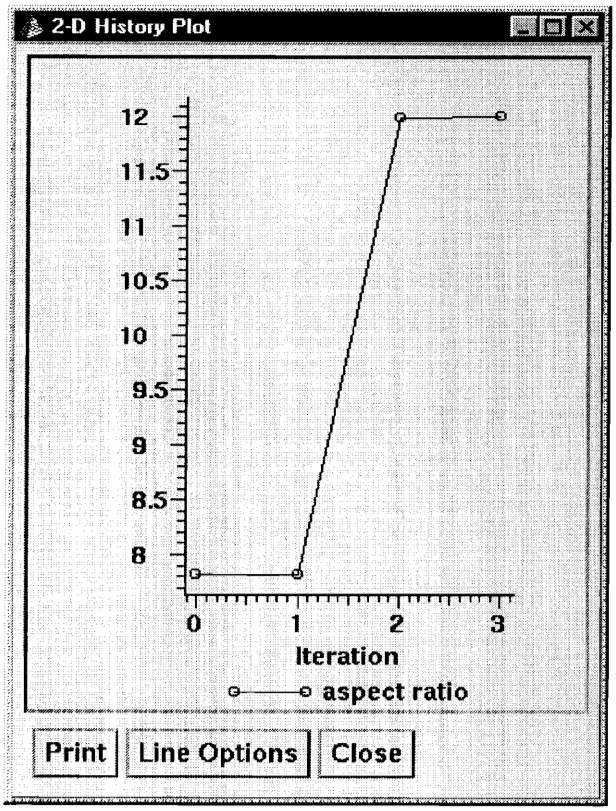


Figure 40 Configuration 2. Convergence History for Aspect Ratio.

Finally, Figure 41 shows the convergence plot for the objective function, C_D. This plot shows the minimum value of the drag being computed as 0.0007, a positive, and far more reasonable number for the drag coefficient.

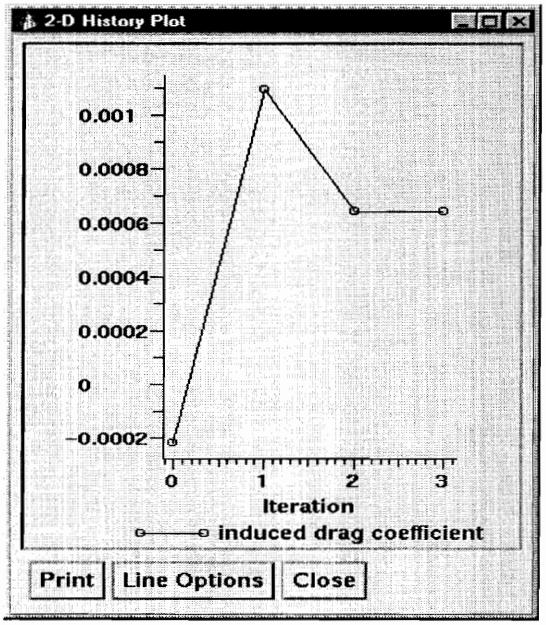


Figure 41 Configuration 2. Convergence History for the Objective Function (C_D).

7.2.6 Post-Optimality Analysis

For this sample problem, we can perform an optimality analysis by checking the Kuhn-Tucker conditions. At the optimum design point, the Kuhn-Tucker necessary conditions state the following:

1. The optimal design variable set, X^*, is feasible.

2. The product of the Lagrangian multiplier, λ, and the constraint value, g, is zero.

$$\lambda \cdot g(X^*) = 0 \tag{23}$$

3. The gradient of the Lagrangian equals zero.

$$\nabla L = \nabla F(X^*) + \sum \lambda \cdot \nabla g(X^*) = 0 \tag{24}$$

For this example, there was only one constraint, the lift coefficient, and it was active at a value of 0.18, keeping the design in the feasible region. Also, all three design variables were within the prescribed limiting bounds. Therefore, the first Kuhn-Tucker condition is satisfied.

We next examine the second condition. Since the value of the lift coefficient constraint is at the prescribed limit, this constraint is active and has a value of 0. Therefore the product of the Lagrange multiplier, λ, and the constraint function, g, is 0. Thus, the second condition is satisfied.

Finally, we examine the gradient of the Lagrangian. At the optimal solution, the gradient vector for F is given by,

$$\nabla F(X^*) = \begin{bmatrix} -1.182e-3 \\ -5.984e-3 \\ 1.321e-3 \end{bmatrix}. \tag{25}$$

The lift coefficient constraint is active, and the gradient of g given by,

$$\nabla g(X^*) = \begin{bmatrix} -1.122e-1 \\ 1.730 \\ 6.610e-2 \end{bmatrix} \tag{26}$$

At the optimal solution, the aspect ratio and the taper ratio are at their upper and lower bound, respectively.

For post-optimality of the solution, the gradient of the Lagrangian was checked. Since there are two side constraints due to the first and third design variable bounds, two additional Lagrange multipliers were introduced into the first and third elements of the Lagrangian. The resulting Lagrangian is then given as,

$$L(X) = F + \lambda_A g_{CI} + \lambda_1 g_{AR} + \lambda_3 g_{TR} \,. \tag{27}$$

The gradient is then formed as in the following,

$$\nabla L(X) = \begin{bmatrix} \dfrac{\partial F}{\partial X_1} + \lambda_A \dfrac{\partial g_{CI}}{\partial X_1} + \lambda_1 \\ \dfrac{\partial F}{\partial X_2} + \lambda_A \dfrac{\partial g_{CI}}{\partial X_2} \\ \dfrac{\partial F}{\partial X_3} + \lambda_A \dfrac{\partial g_{CI}}{\partial X_3} - \lambda_3 \end{bmatrix}. \tag{28}$$

Using the values for the gradients of the objective and constraint at the optimal solution, the three equations were used to solve for the three lagrange multipliers, λ_A, λ_1, and λ_3. The solution yields the following values,

$$\lambda_A = 0.00346 \qquad \lambda_1 = 0.00157 \qquad \lambda_3 = 0.00155. \tag{29}$$

Verification shows that the gradient of the lagrangian is zero.

$$\nabla L(X^*) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{30}$$

8. SAMPLE APPLICATIONS: ACSYNT EXAMPLES

This chapter will utilize the functions of the Optimization Workbench to perform a series of aircraft design optimization studies using ACSYNT. First order assessments of the effects of advanced technology and detailed mission planning are emphasized.

The capabilities of the ACSYNT core analysis system were integrated with the Optimization Workbench by using the Dynamic Integration System. This integration exposed the operating variables of ACSYNT, items such as geometric parameters, weights, and specific performance inputs and outputs, to the optimization facilities in the Optimization Workbench. Once accessible, these functions can be manipulated by the optimization routines to improve flight performance, reduce fuel burn, or meet flight constraints imposed by the FAA.

After a description of the baseline subsonic transport model, the examples in this chapter begin with a wing geometry optimization. The objective will be to minimize weight by reshaping the wing planform. The next set of examples address the design flight profile and attempt to improve performance of the vehicle from a mission operation standpoint.

Specific constraints are added to simulate the effect of an imposed FAA regulation on the departure climb profile in a terminal area.

Finally, the last set of examples will use ACSYNT in conjunction with a newly created model to perform detailed flight segment analysis. This flight segment model utilizes actual measured wind data and computes corrected enroute flight time for a given path between a set of latitudes and longitudes for an origin and target location.

8.1 Baseline Development

Before a complete optimization study can be performed using ACSYNT, a robust baseline model was developed in the ACSYNT geometric design module. Figure 42 shows the aircraft geometric arrangement. A twin-engine, subsonic transport was chosen as the design problem. This type of aircraft is useful because it has wide applicability across a broad spectrum of flight profiles. This aircraft model can also serve as a component in the aircraft fleet economic analysis program utilized in an extended sample application. The key information for the baseline design is shown in Table 2. A basic two segment cruise mission, separated by various climb segments is employed.

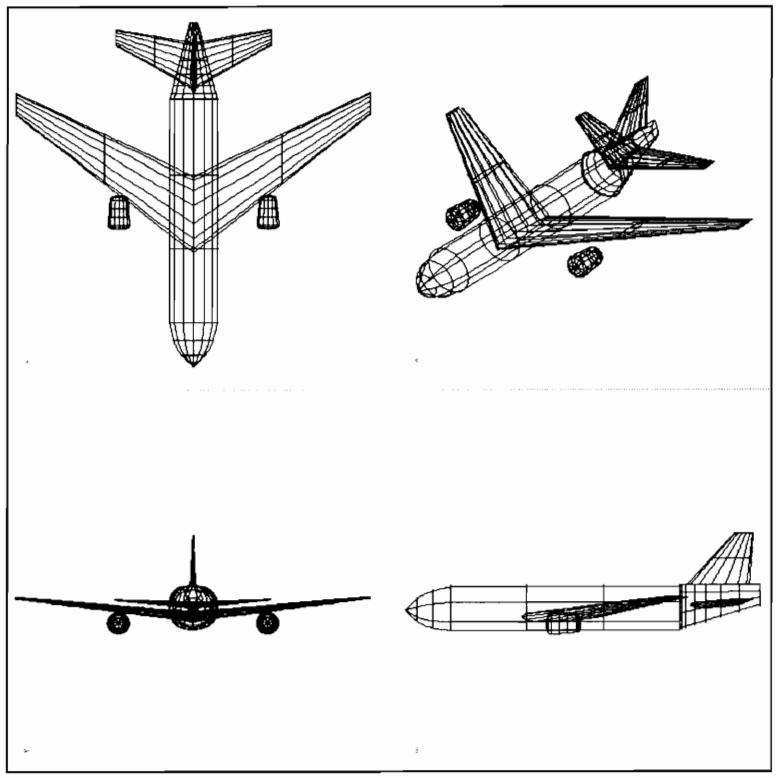


Figure 42 Baseline Aircraft Model Shown in the ACSYNT 3.0 Geometry Modeler.

Table 2 Baseline Design Parameters and Key Performance

Design Parameter	Baseline Value
Wing	
Aspect Ratio	*7.16*
Area	*3541 ft^2*
Sweep	*35 deg*
Span	*159 ft*
Taper Ratio	*0.30*
Engines	
Number	*2*
Thrust Level	*50,000 lb each*
Fuselage	
Overall Length	*155 ft*
Diameter	*19.0 ft*
Weights	
Airframe Structural	*122,850 lb*
Propulsion System	*25,300 lb*
Fixed Equipment	*38,980 lb*
Operating Empty Weight	*194,420 lb*
Fuel	*100,500 lb*
Payload	*49,200 lb*
Total Takeoff Weight	*344,200 lb*

8.2 Wing Geometry / Structural Design Optimization

The first example of using the Optimization Workbench with the ACSYNT program is a simple wing geometry optimization to minimize the structural weight of the aircraft. Several wing parameters were selected to be used as design variables. Figure 43 shows the selection of these variables and their current value in the system.

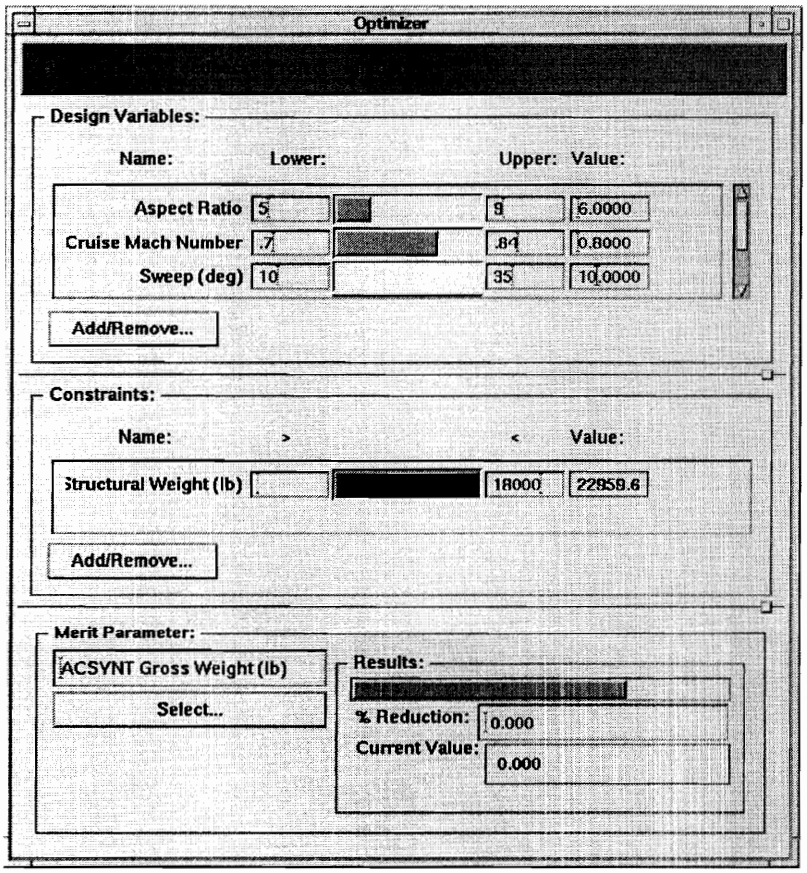


Figure 43 Variables for ACSYNT Sample Activated in the Optimization Workbench.

The objective function for this example is the total takeoff gross weight. After 4 iterations using the Modified Method of Feasible Directions, the minimum weight for the selected

design variables was found. The termination criteria for the algorithm was the satisfaction of the Kuhn-Tucker necessary conditions for the constrained case. Figure 44 shows the convergence history for the aspect ratio.

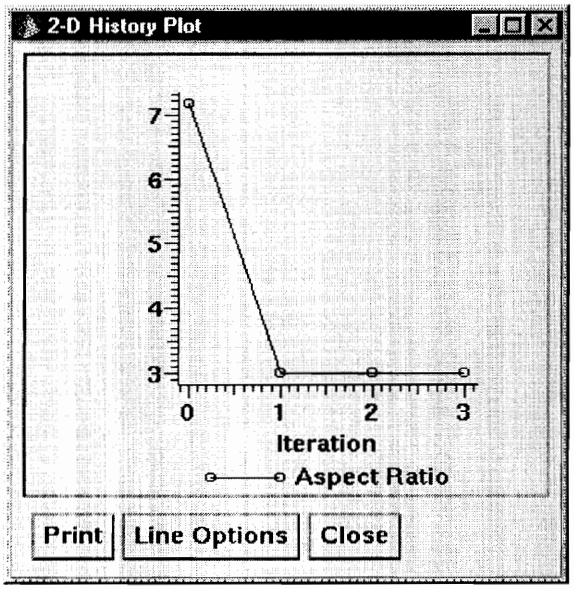


Figure 44 Convergence History of Aspect Ratio for ACSYNT Example.

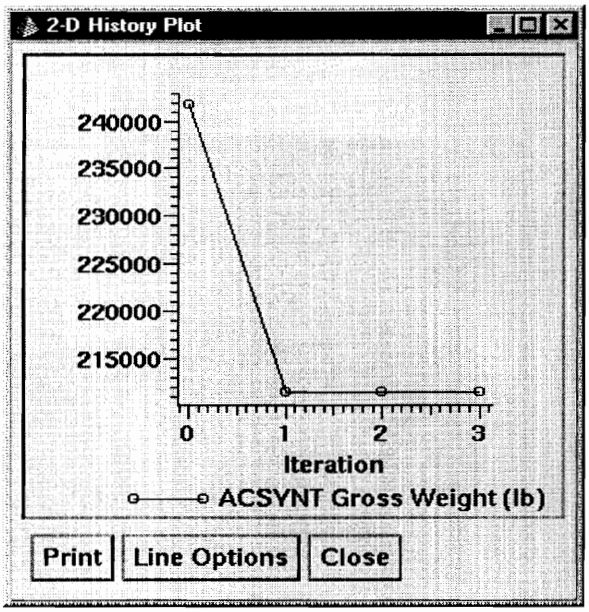


Figure 45 Convergence for Gross Weight.

In order to gain a better understanding of the detail and level of convergence for all the design variables selected, the mapping routine for the 2-D scatter plot was used. Figure 46 shows an overview of the mapped convergence data. There are only two clusters of variables. Closer inspection of the upper-most cluster reveals a tight convergence history near the optimum value.

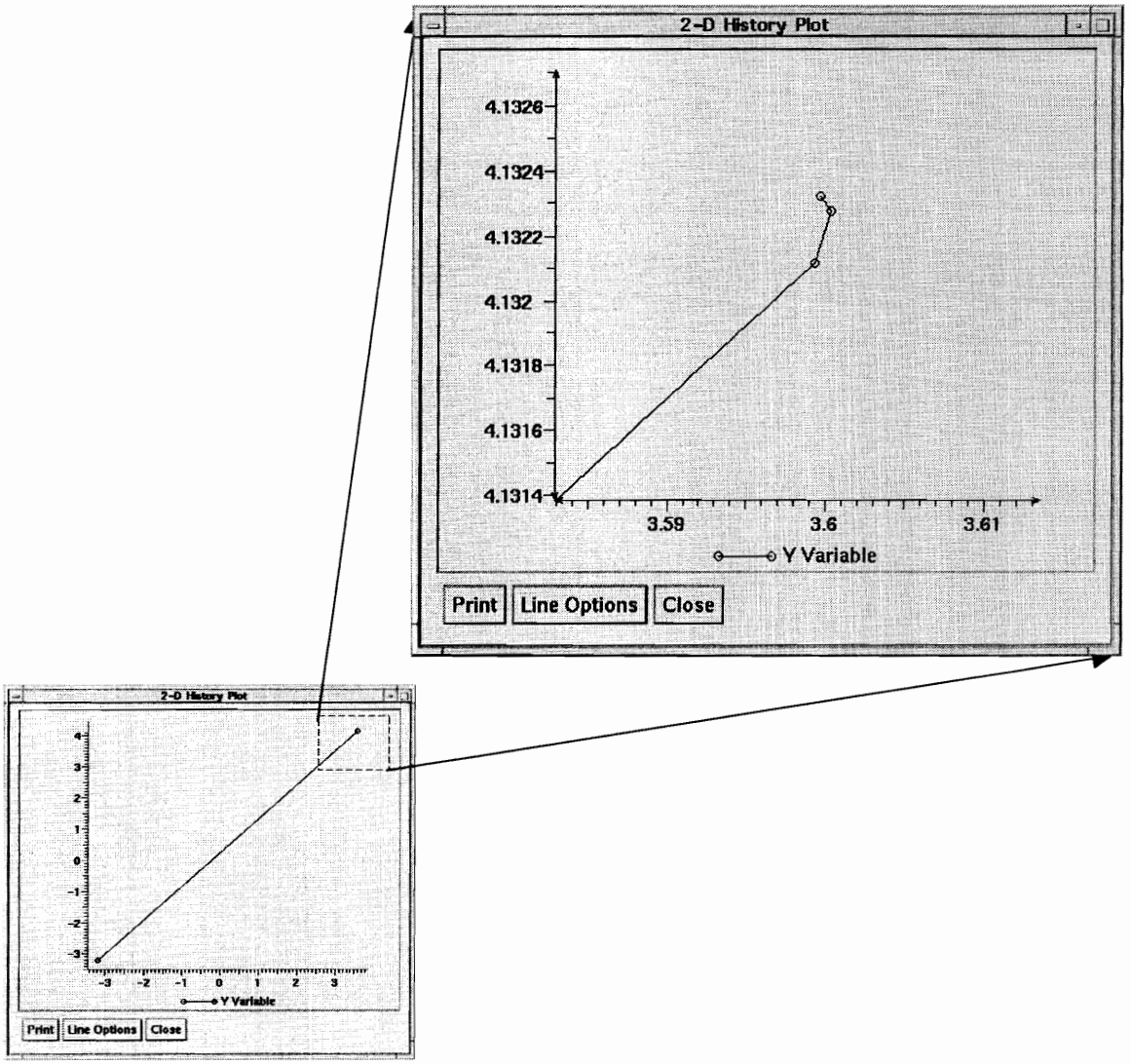


Figure 46 Mapping of Data for AR, TR, Sweep Showing Enlarged Area of Convergence for ACSYNT Example.

8.3 ACSYNT Mission Optimization

The trajectory analysis module of ACSYNT was used to perform detailed flight segment optimization for the baseline aircraft mission. In the following examples, specific problems were solved using the Optimization Workbench. Each example focuses on a different aspect of the mission profile and examines the effect of changing the profile to model influences from the airspace operating environment.

8.3.1 Range Allocation

The purpose of this example is to examine the effect of separation between three target destinations and which profile offers the most fuel efficient delivery to the destinations. The first configuration selects the range for the two cruise phases as the design variables. We choose to minimize the block fuel while keeping a constraint on the total range flown in the profile. Results for this case are shown in Figure 47. As the plot shows, it is best to displace the middle target destination as far away from the first as possible. This can be attributed to the performance of the climb back to the cruise altitude in the second cruise phase. Effectively, the optimization algorithm has minimized the weight of the aircraft during the climb in the 4th and 5th *segments* of the mission. This is accomplished by cruising as long as possible in the first cruise segment.

The convergence history for the gross weight gives insight into the performance of the merit function. The weight increases as the range increases to satisfy the block range constraint. The weight then decreases as the optimal distribution of range is found.

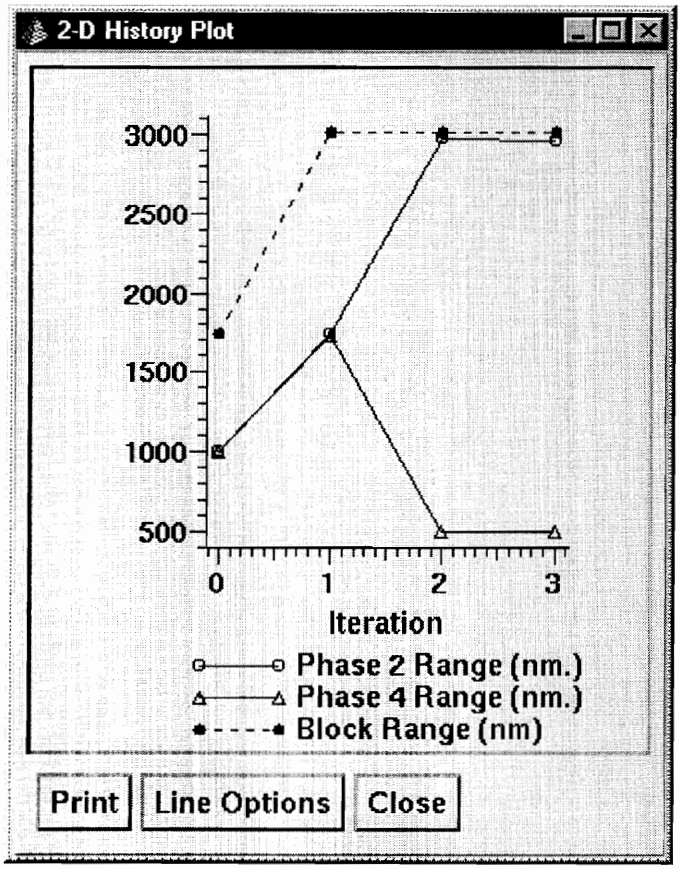


Figure 47 Convergence History for Optimal Range Allocation

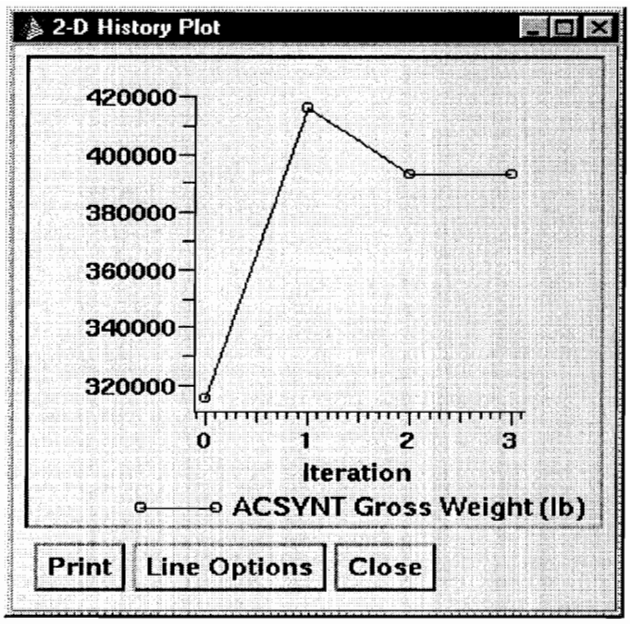


Figure 48 Convergence History for the Range-Allocation Merit Function (Gross Weight).

8.3.2 Minimum Fuel Climb Analysis

The purpose of this analysis is to determine the effect of flying sub-optimal departure paths due to restrictions in airport controlled airspace. The baseline twin-engine transport aircraft was used for this study. The same mission profile was used as in the range allocation study.

Unrestricted Climb Speed: The first problem will use the climb speed below 10,000 ft as a design variable and the fuel burn in the climb phase will be minimized. No restrictions will be placed on the speed. The result for this case is shown in Figure 49.

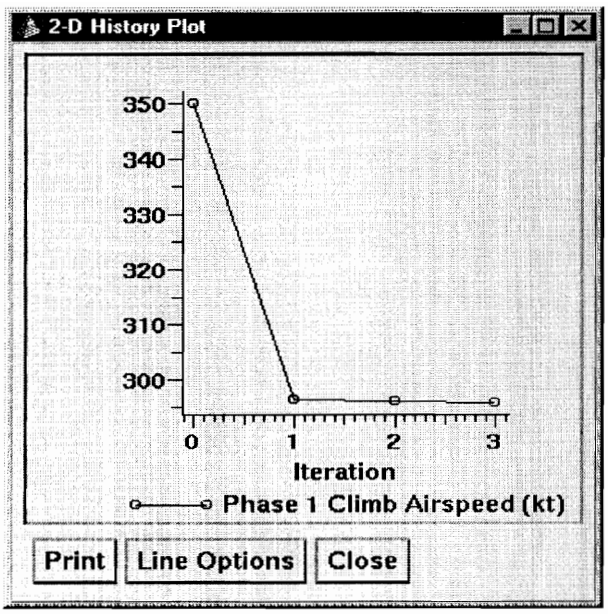


Figure 49 Optimal Speed for Climb to 10,000 ft.

Restricted Climb Profile: After the optimal departure speed is determined, a 250 kt. constraint was applied to the speed below 10,000 ft. We then investigate the resulting weight increase as the configuration is re-optimized.

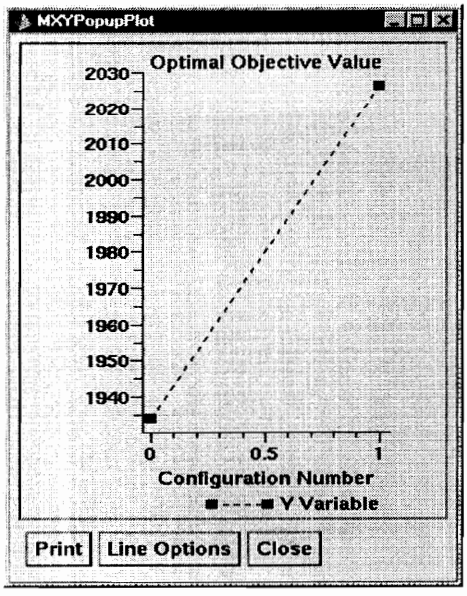


Figure 50 Weight Penalty for Imposing 250 kt. Climb Speed Restriction Below 10,000 ft.

8.3.3 Optimal Cruise Profiles

This sample uses the mission analysis capabilities of ACSYNT to compute optimal cruise configurations for the twin-engine transport baseline model. The first example uses a set of two different fixed altitudes and computes the optimal Mach numbers for both segments. The convergence history for this example can be found in Figure 51. Notice how the Mach number for each phase adjusts to the optimal speed for the phase 2 and phase 4 altitudes. The phase 2 altitude was fixed at 36,000 ft and the phase 4 altitude was set to 10,000 ft. The resulting Mach distribution showed it optimal to cruise faster at the higher altitude. As the altitude and range decrease, the slower Mach number was optimal in the 4th phase. Figure 52 shows the savings in weight for this example. As viewed from the convergence history, a savings of over 60,000 lb. is achieved.

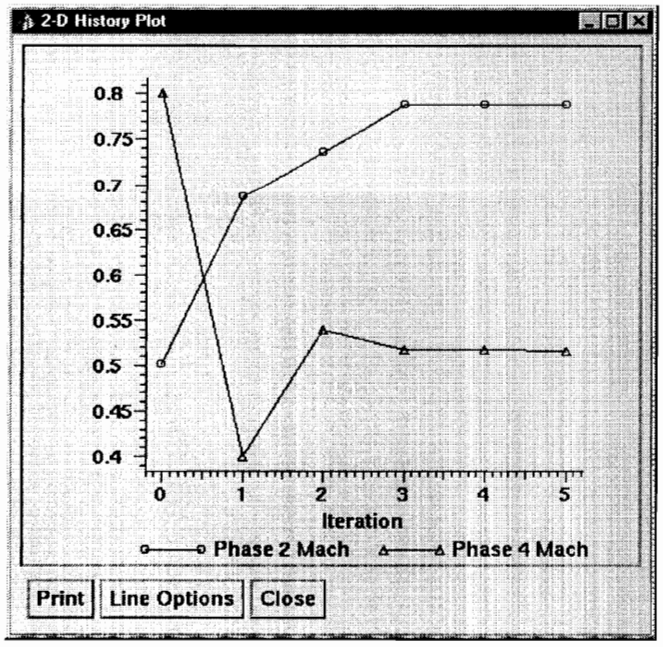


Figure 51 Optimal Mach Number Profile for Fixed Altitude.

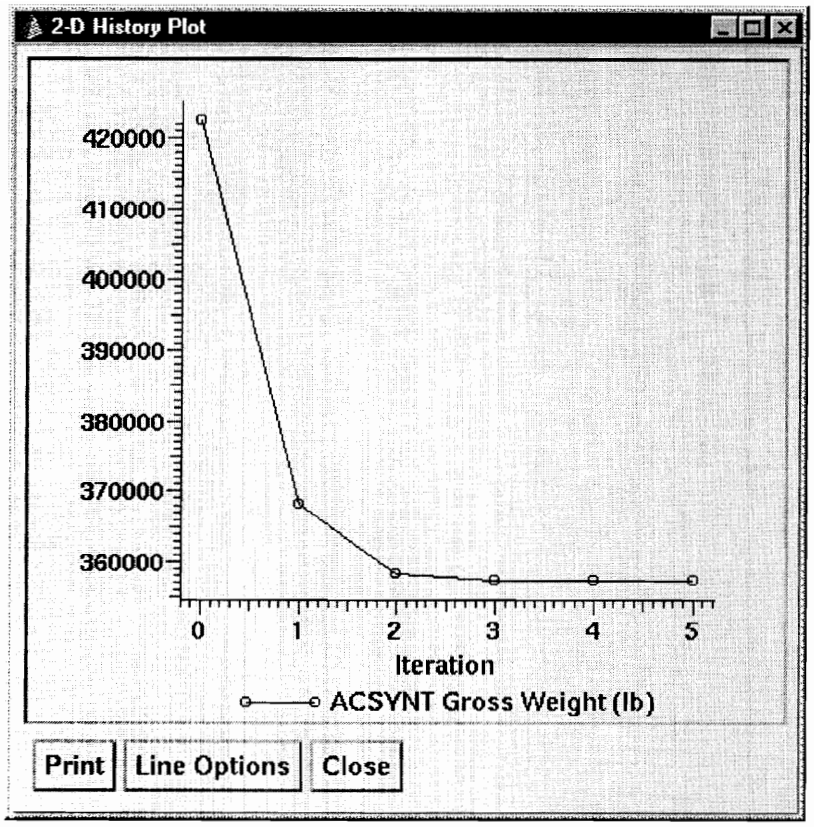


Figure 52 Weight Convergence for Optimal Mach Profile.

Once the Mach numbers were established for the fixed altitude, the phase 2 altitude was added to the optimization problem. The Mach number for phase 2 was also used as a design variable in this problem. Figure 53 shows the convergence history for this example. The first three iterations were showing no movement in the objective with respect to changes in the altitude. This could be attributed to the fact that the Mach number had already been optimized for the 36,000 ft. altitude and further changes from that altitude were minimal. The initial altitude was then reduced to 10,000 ft. and the optimizer was restarted. The altitude was then increased from 10,000 ft. to 29,000 ft. by the optimizer.

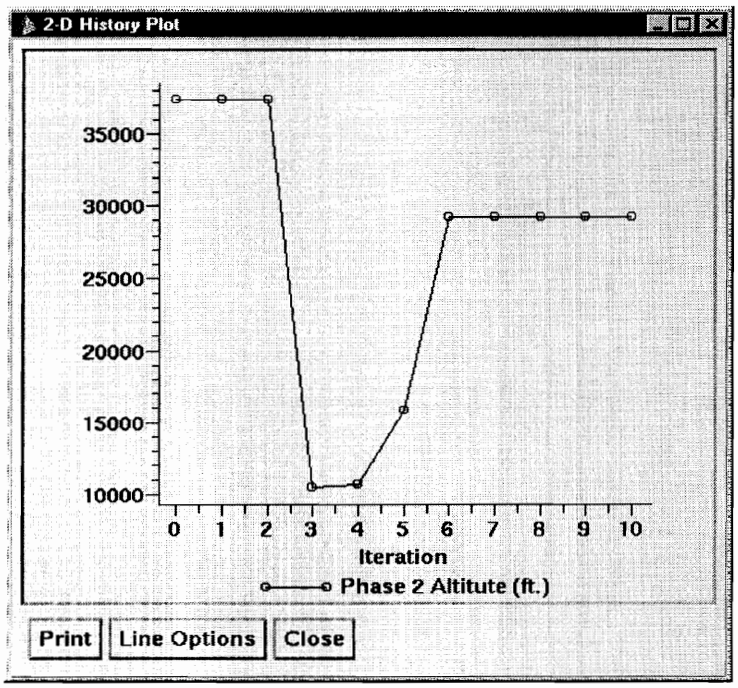


Figure 53 Optimal Altitude Profile for Single Phase (Includes Restart from 10,000 ft.)

Figure 54 illustrates the effect of the altitude changes on the Mach number. It is interesting to note that the Mach number varied slightly in the first three iterations, but then as the altitude was reset to 10,000 ft., slowed considerably.

It appears that both the altitude and Mach number arrived at their optimum for the minimum weight. However, when checking the convergence for the objective function in Figure 55, we notice that the weight did not converge below the original weight, but converged to a different solution. The facilities within the Optimization workbench made it very easy to detect this. In this case further pursuit of the global minimum was not

attempted, though, the facilities within the Workbench would have allowed the user to switch methods, change tolerances, etc. as a way to improve the design.

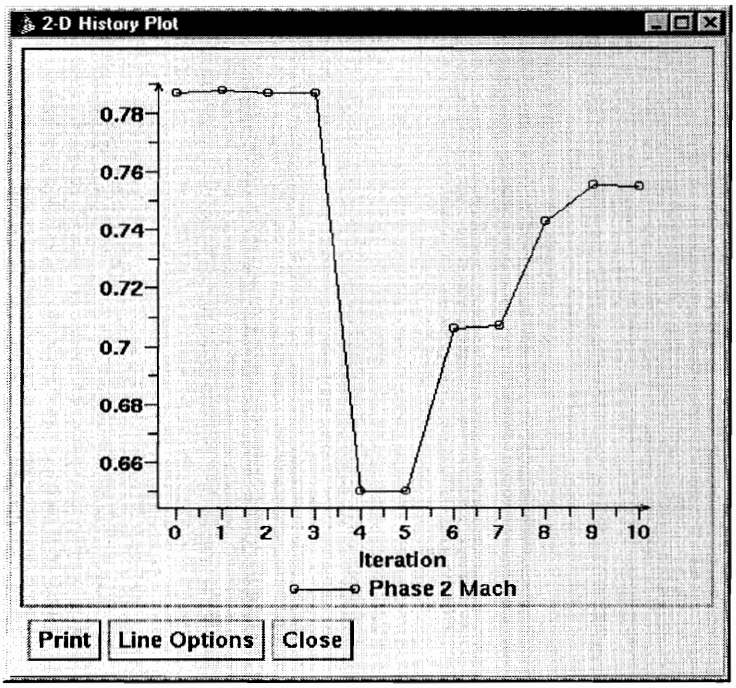


Figure 54 Optimal Mach Profile for First Cruise Segment.

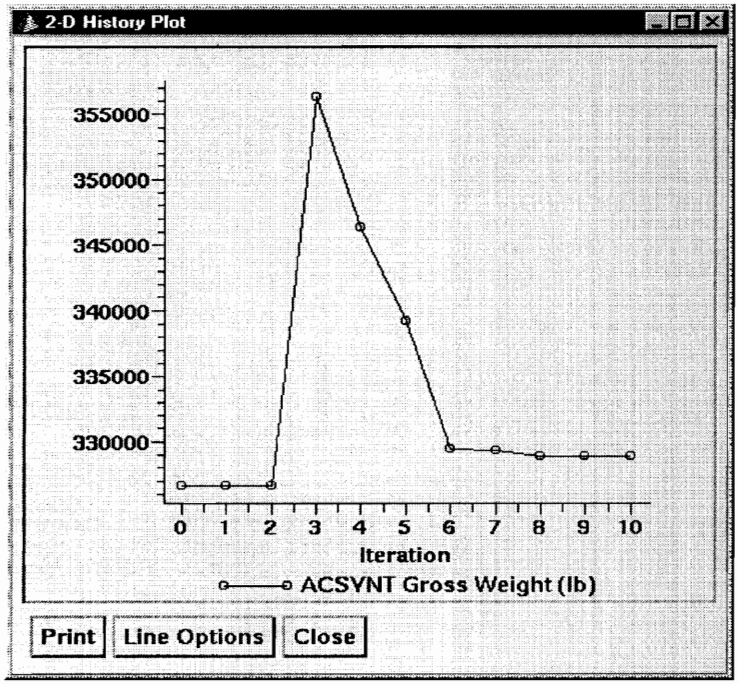


Figure 55 Weight Convergence for Optimal Cruise Profile. (Includes Re-start).

Finally, after multiple configurations were developed within the main project file, a global perspective regarding the resulting optimum was prepared. In this case the PROJECT HISTORIES function was utilized within the project editor. This function allows the user to extract the optimal result from each configuration. A plot is then created to show each optimal result in context with the other elements of the problem. Figure 56 shows the plot for the optimal objectives developed in the ACSYNT mission optimization suite of configurations. There were seven configurations developed. Each had different constraints and design variables applies. The results show that as you increase the number of constraints, as in the last three problems, the objective will be penalized. Adding constraints can be offset by adding degrees of freedom, however. For example, the last three examples progressively reduced the optimal solution. For these

last three configurations, first the range constraint was added, then the Mach and altitude were added. This shows the increasing freedom in the design to overcome the constraint in the system as the weight was reduced by 1000 lb. over the last three problems. The minimum of all the optimal solutions is 310,000 lb. This is achieved by performing an unconstrained problem using altitude only with fixed Mach numbers.

By viewing an evolution of the mission optimization project through histories of multiple configurations, the user becomes more aware of the implications of changing design requirements. Using the optimal history toolkit, the user has the ability to observe the design progress from a low degree of freedom, loosely constrained problem to a highly flexible, tightly constrained problem.

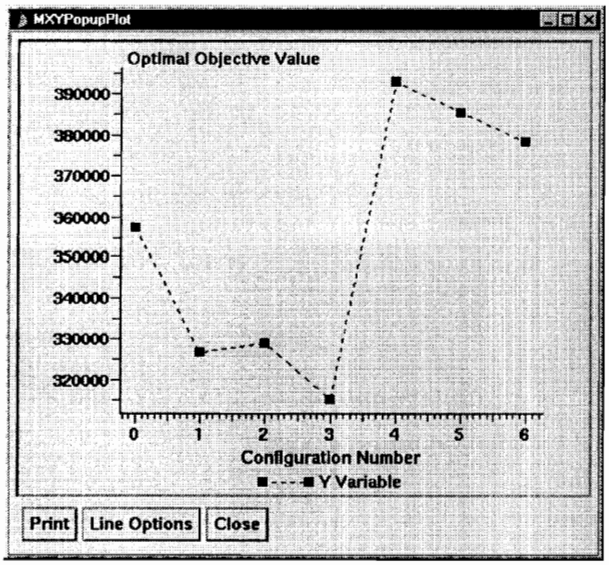


Figure 56 History of Optimal Values for Gross Weight for a Variety of Mission Optimization Problems.

8.4 Flight Segment Optimization Using Winds Aloft Data

The concept of "Free Flight" has been receiving attention recently in the air carrier industry. Increased flexibility within the flight profile can offer the operating airline savings in fuel cost, or conversely, serve to maintain a tight schedule by allowing altitude and speed variations enroute. Delta airlines has outlined a plan that offers their air vehicles increased performance based on detailed flight path optimization [91]. The benefits are encompassed in a suite of results that range from minimum time solutions to minimum fuel burn solutions. These paths utilize speed, altitude and latitude/longitude as variables in the optimization problem.

The following set of examples illustrate using the Optimization Workbench to perform detailed flight path analysis that supports a free flight project study. Measured wind data collected for the United States by NASA Goddard [92] is incorporated into the segment model. The purpose of this analysis is to build a suite of cases that illustrate optimal flight paths between two city pairs given actual weather constraints. Several cases with varying degrees of freedom are performed. The objective is to find both time-optimal solutions with fuel constraints and fuel-optimal solutions with time constraints.

8.4.1 Overview of Flight Segment Model

A flight segment model was constructed to map a flight profile between two cities represented by an origin and target latitude/longitude pair. The path is divided into 10

segments with corresponding latitude and longitude "waypoints." These waypoint coordinates are then used in the optimizer to manipulate the path. The altitude and segment Mach number are controllable for each segment as well, bringing the total degree of freedom to 4 variables for 10 segments, or 40 total design parameters. Figure 57 illustrates the segment coordinate structure.

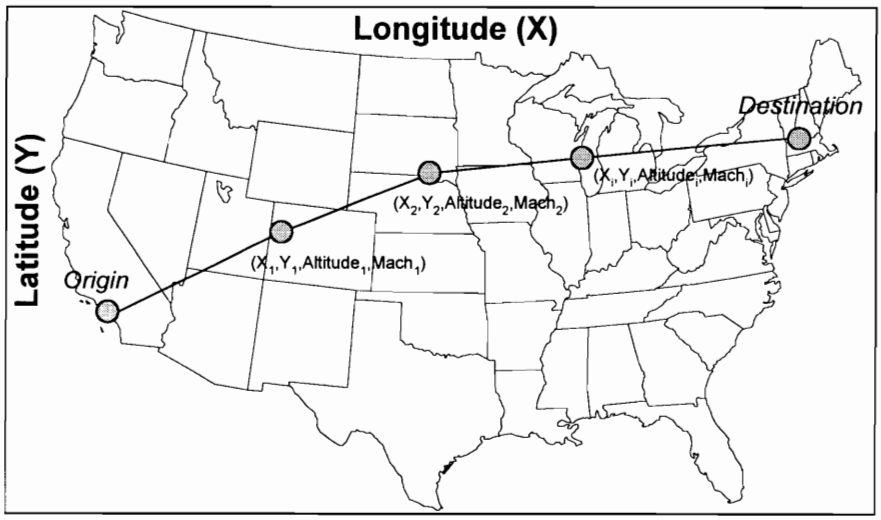


Figure 57 Flight Segment Model Description

A database of measured wind data from the 1995 operating year was incorporated to serve as path constraints. The model computes the equivalent airspeed and resulting segment range for each segment in the model. This equivalent range is then used in ACSYNT along with the altitudes and Mach numbers for each segment to compute fuel burn along the path. Both the flight segment model and the ACSYNT model are integrated with the Optimization Workbench through the Dynamic Integration System. ACSYNT can share the range and speed information from the segment model through the

detailed set of Dynamic Variables for each segment. Total fuel burn and time enroute are accessible for the optimization program. Addition segment output details include segment fuel-burn.

By using the graphical capabilities of the Optimization Workbench, the range of latitudes and longitudes can be observed visually through the Dynamic Gauges. Figure 58 shows the trajectory optimization problem posed in the Optimization Workbench. Each segment coordinate is a design variable in the configuration.

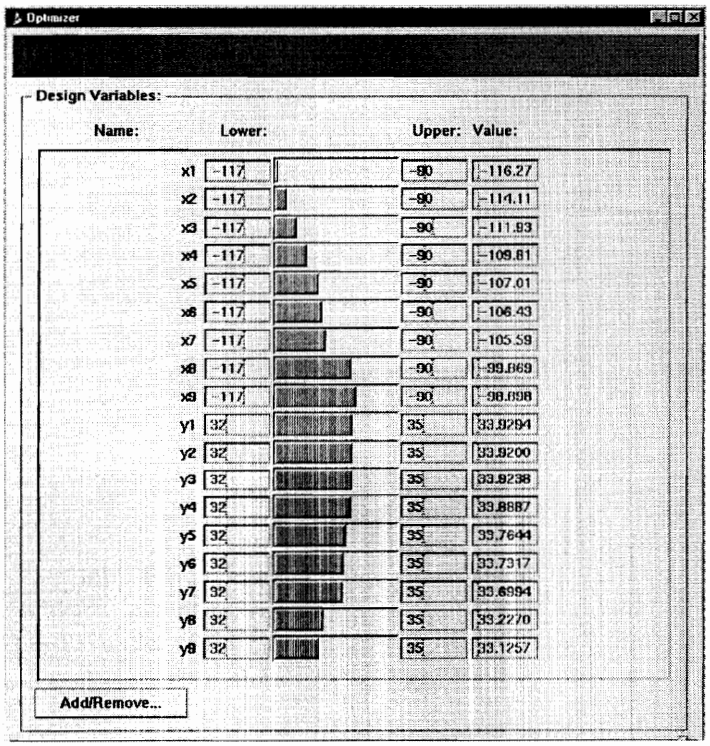


Figure 58 Trajectory Optimization Problem Using the Optimization Workbench.

8.4.2 Min Time: Fixed Speed, Unconstrained

The following analysis investigates the minimization of enroute flight time between two city pairs given starting and ending latitude and longitude. The path is divided into 10 segments. The design variables for this problem are 9 sets of longitude/latitude coordinates corresponding to the segment divisions.

8.4.2.1 Tailwind Case: Los Angeles-Dallas/Fort Worth, December 25, 1995.

Figure 59 shows the initial trajectory for the great circle route between two coordinates representing Los Angeles and Dallas-Fort Worth airports. Actual measured wind data was used for December 25, 1995 as the path constraints.

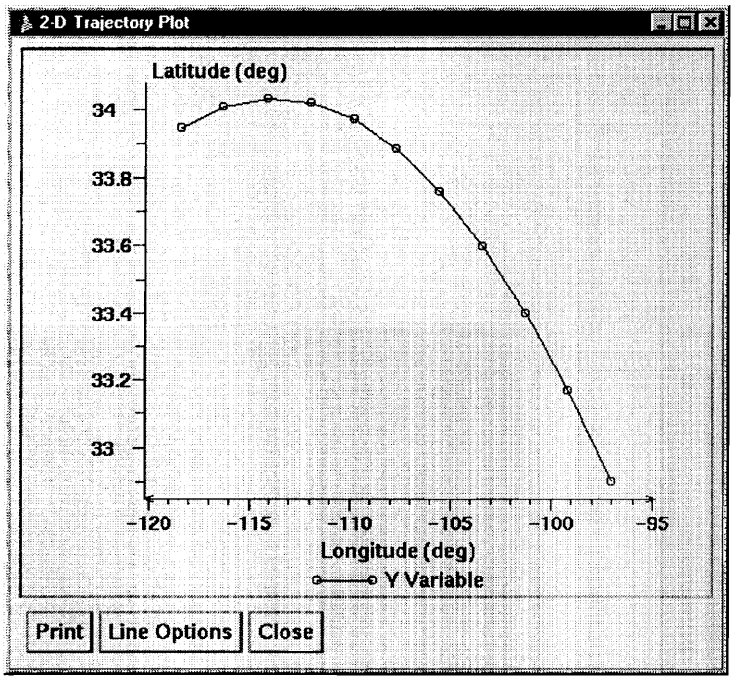


Figure 59 Great Circle Trajectory between Los Angeles and Dallas-Fort Worth

The objective function is the total time spent on the path. The initial value for the great circle path was 2.015 hours. Several of the waypoints were then perturbed to create an extended, sub-optimal path. Figure 60 shows the initial, perturbed trajectory that serves as the starting point for the optimization problem.

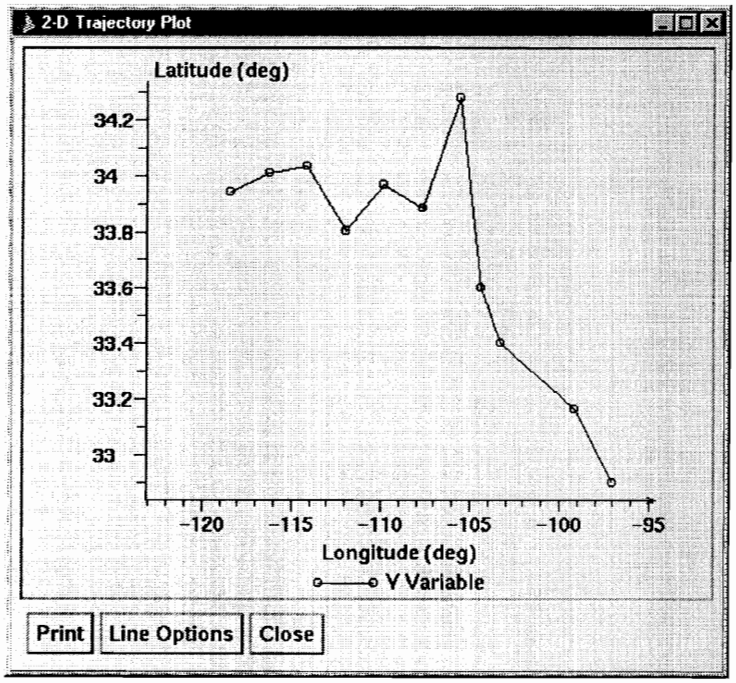


Figure 60 Initial Starting Point: Perturbed Trajectory Between LA and DFW.

The analysis for the trajectory algorithm was integrated into the Dynamic Integration System. A Dynamic Variable was then created for each waypoint longitude and latitude. The Optimization Workbench was opened and each point was added as a design variable. A total of 18 points were used to control the shape of the trajectory.

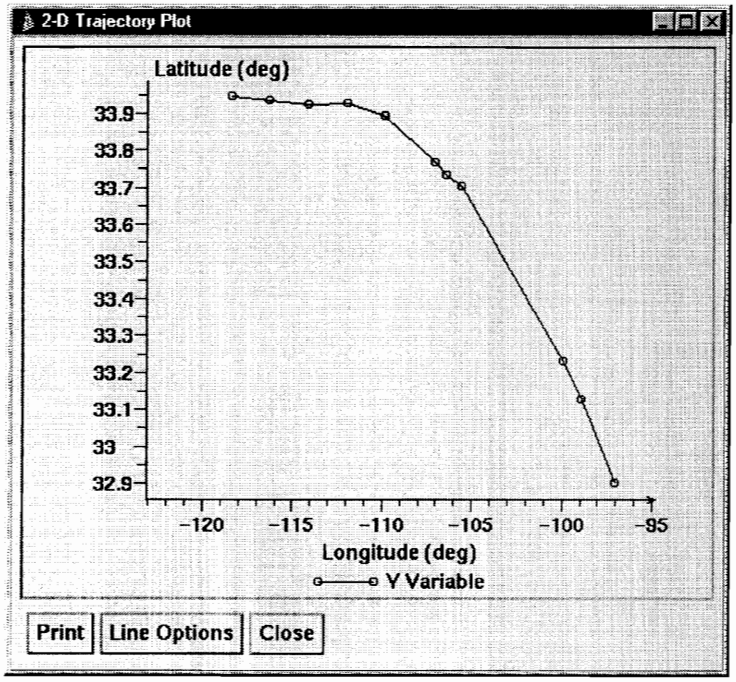


Figure 61 Results for Optimal Trajectory

Figure 61 shows the resulting trajectory after 7 iterations of the optimizer using the BFGS unconstrained optimization algorithm. The convergence history for the time, Figure 62, shows an approximate 30 minute savings in flight time from the initial perturbed path.

Figure 63 shows the convergence of all the latitude waypoints for each segment. Notice how more variations are placed on the first 5 latitudes. This is evident from observing the initial starting path and noticing how far off-path these points were located.

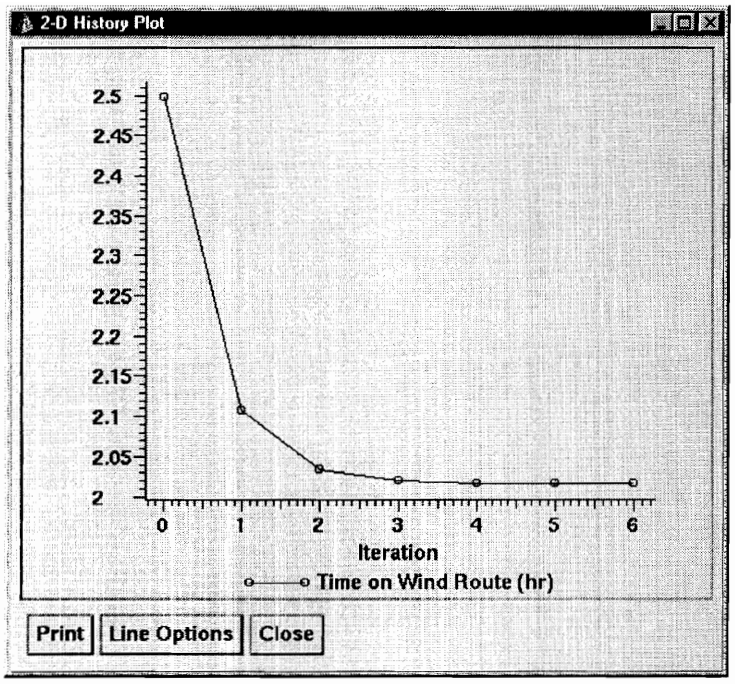


Figure 62 Convergence History for Enroute Cruise Time.

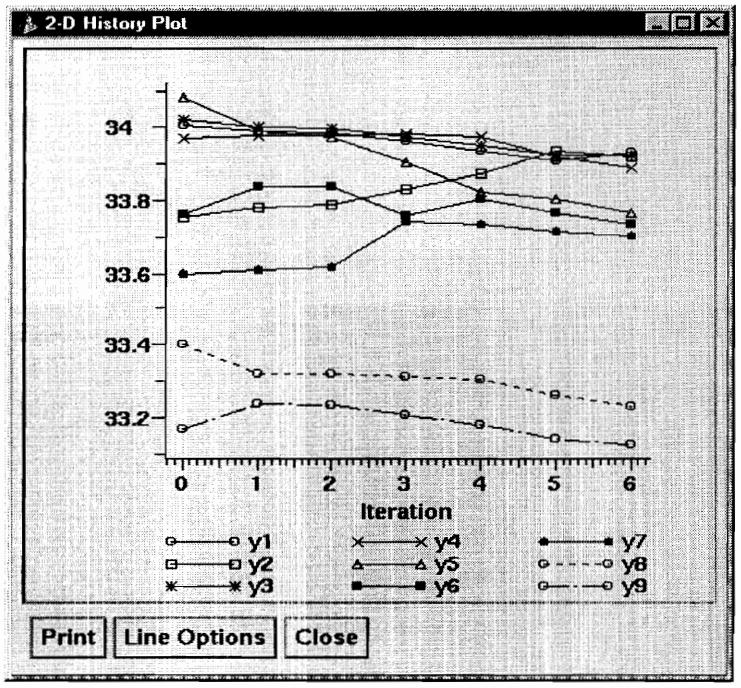


Figure 63 Convergence Histories for All Latitudes.

8.4.2.2 Severe Headwind Case: Boston-Los Angeles, September 22, 1995.

The next example utilizes an extreme case in the winds aloft as measured on Sept. 22, 1995. This example is best illustrated in an east-west trans-continental trajectory between Boston and Los Angeles. Figure 64 shows the shortest path between the two cities by plotting the great circle path. It was determined that a strong headwind was present in the upper altitudes near the 40 and 41 degree latitude and 80 degree longitude. As in the previous example, the latitude and longitude were used as design variables. The starting point for the problem was the initial computed great circle path.

Figure 65 shows the computed path for the optimal time between Boston and Los Angeles. Notice that the path is deviated northward by an additional 1.5 degrees. Also, the path is extended northward past the 90 degree longitude significantly.

This problem was particularly interesting for its convergence behavior. Figure 66 is a plot of the convergence history for the enroute time. This data represents multiple restarts. Finally, the search direction used in the optimization algorithm was changed from the BFGS variable metric to the sequential quadratic programming method and the time decreased radically to its final value. Figure 67 shows the history of the design variable changes for this problem. Again, notice how the latitudes were shifted northward to avoid the excessive headwinds found on the original great circle flight path. The restart and escape from the local minima is also evident from the data.

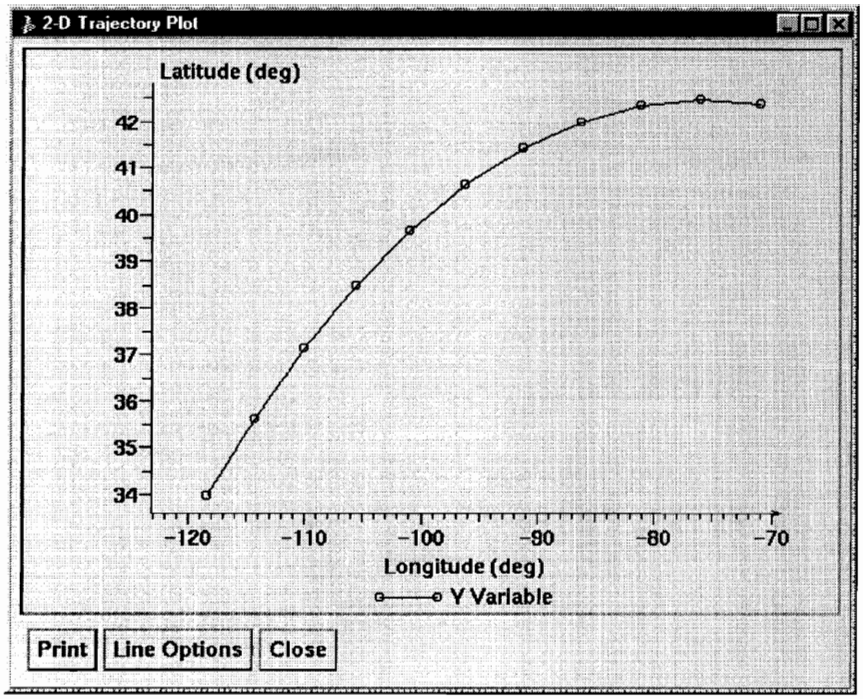


Figure 64 Great Circle Path between Boston and Los Angeles.

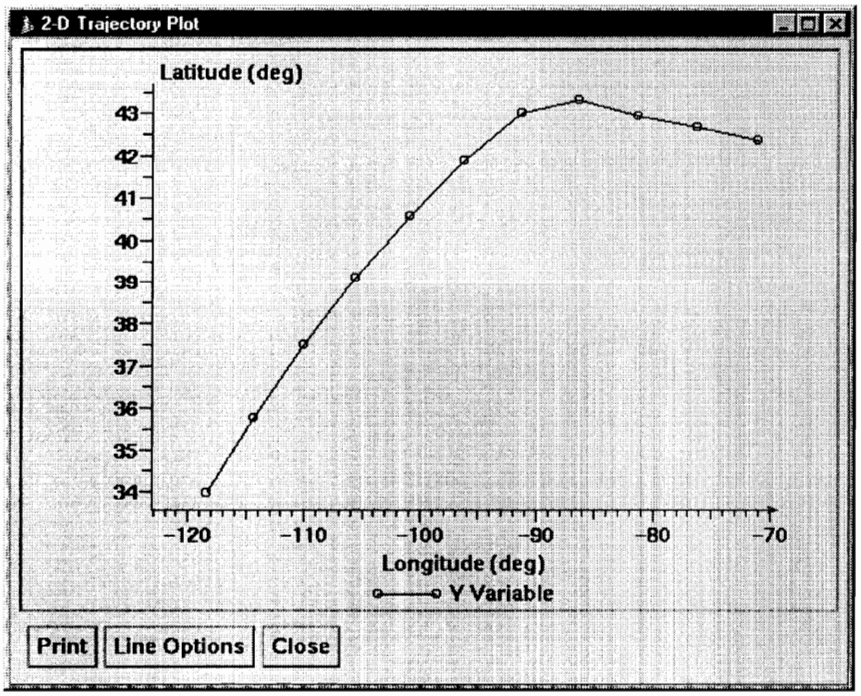


Figure 65 Optimal East-West Path between Boston and Los Angeles for Winds Measured Sept. 22, 1995.

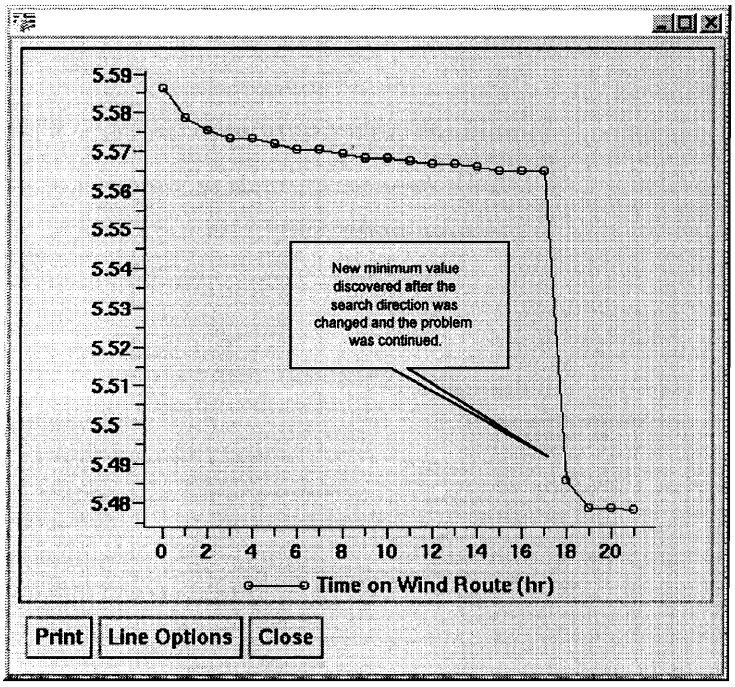


Figure 66 Convergence for Min Time Illustrating Convergence After a Search Direction Method Switch

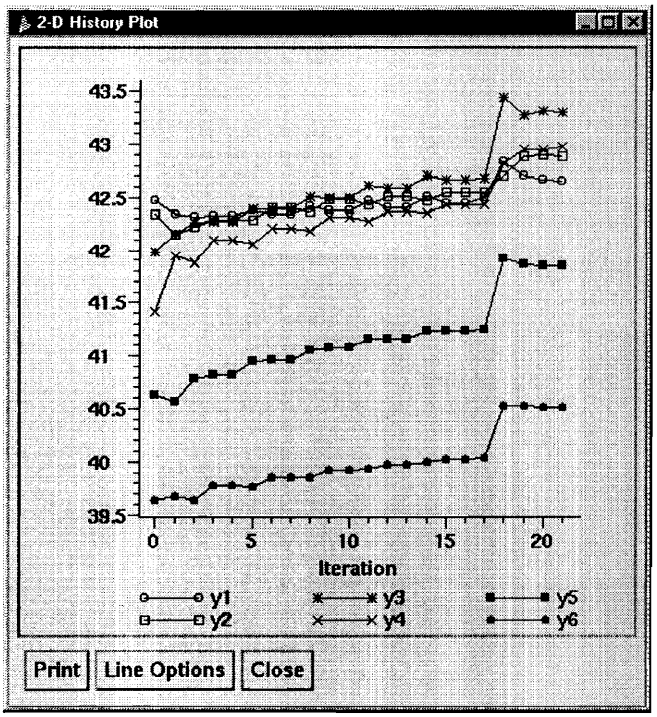


Figure 67 Convergence for Northern-most Latitudes Between Boston and Los Angeles.

8.4.3 Min Time: Direction and Speed Variations, Fuel Constraint

Additional time savings can be realized through speed increases in each segment. These increases in speed are not penalty-free, however. To simulate a realistic operating environment, a fuel constraint was added to the problem. The increase in speed that the optimizer chose is shown in Figure 68. The profile shows an overall Mach increase from 0.80 to 0.81. This accounted for an overall time savings of approximately 5 minutes. The penalty for increasing to this speed, as computed by ACSYNT, is roughly 7500 lb. At approximately $0.10 per pound of fuel, the increase in block time has cost the operating air carrier roughly $750.

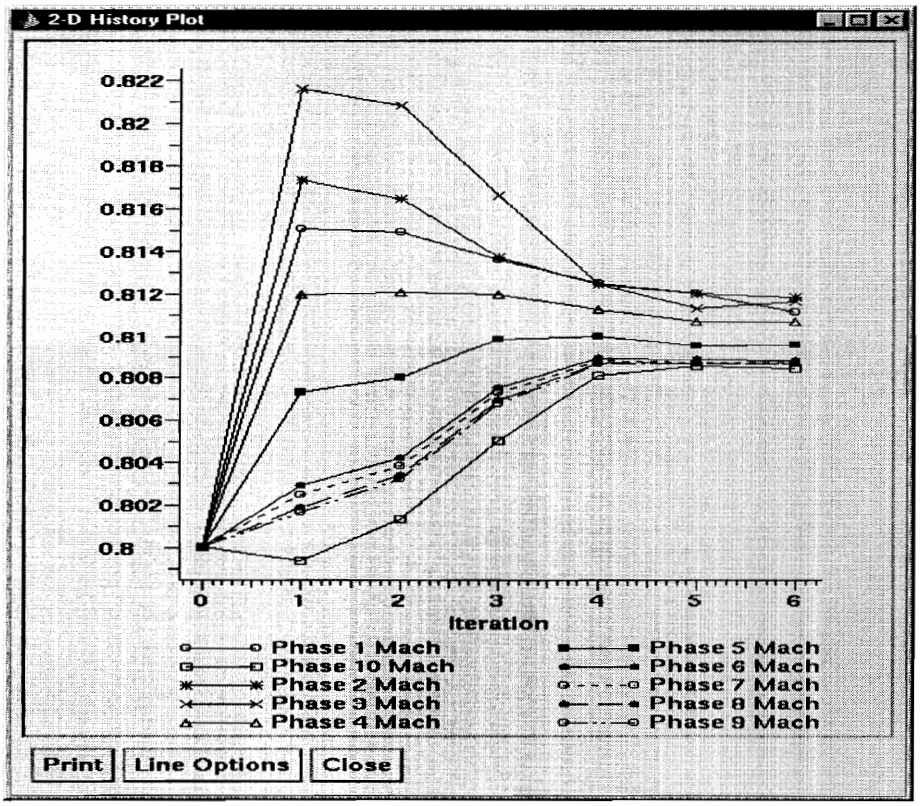


Figure 68 Mach Number Convergence for Min Time/Fuel Constrained Path

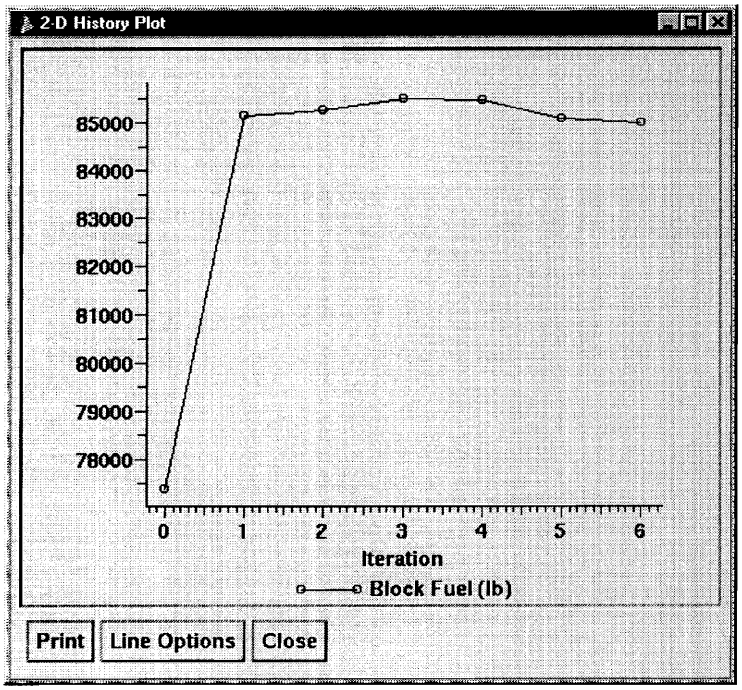


Figure 69 Resulting Fuel Constraint for Min Time Solution Utilizing Mach Variations.

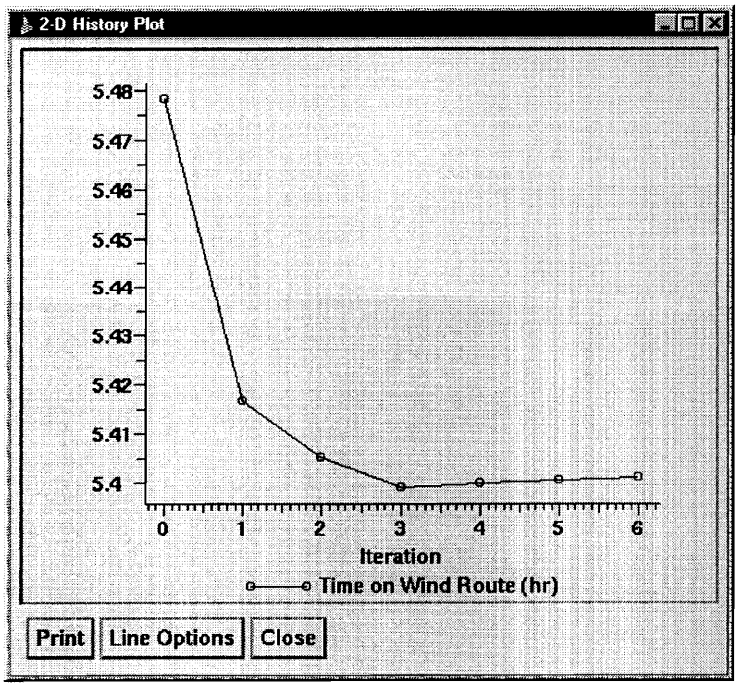


Figure 70 Additional Time Savings Utilizing Segment Mach Numbers.

9. CONCLUSIONS

This research has created a new method for a software environment for performing design activities utilizing numerical optimization algorithms. Specifically, the research focuses on a new technique for interacting with optimization software, a unique method for handling design information, and creative methods for visualizing optimization results.

A *traceable* path of changes to the design and what the influences were at each change has been created. The Optimization Workbench allows the dissemination of the results of a problem after the optimization algorithms are utilized. Results can be scrutinized and histories can be examined in detail to understand why the process arrived at the resulting design.

Software control of the optimization process has been offered. The algorithms as well as the actual problem formulation are accessible and controllable for the design engineer to utilize the capability of numerical optimization.

Integration and control of a wide range of analysis modules has been demonstrated. The use of ACSYNT as an intermediate program in a larger design study such as the

trajectory analysis examples illustrated that the optimization control and integration facilities can bring new capability to existing systems.

9.1 Analysis of Multivariate Data Analysis

The use of the nonlinear mapping algorithm to reduce the dimensionality of the optimization results has proven a useful tool to investigate convergence behavior of a single problem, or the context of different optimization problems. Specifically, the user can obtain an overview of the histories of multiple design variables in one, concise plot. Trends in search directions and meaningful collections of multiple, optimal configurations give the user insight into the design evolution.

9.2 Comments on Visualization Techniques

Graphical feedback from interface tools have been demonstrated to be a useful tool for discerning information quickly and on-the-fly. The use of color for correlation to proximity to boundaries, and the use of movement to correlate search direction trends gives the user insight into the optimization problem as it evolves. These tool makes it easier to "watch" the optimizer in a real-time mode and evaluate the likelihood of mis-directed designs.

9.3 Correlation of Approach

Chapter 1 outlines the approach taken to replace the standard ACSYNT optimization facilities with a new control and optimization environment. Through the course of the many sample problems illustrated in this dissertation, this plan has been proven successful. In review, the highlights of this plan included:

- Creation of a highly interactive environment for utilizing the capabilities of a new optimizer
- Replacement of the CONMIN optimization algorithm with an updated suite of numerical optimization algorithms
- Elimination of the COPES control program in favor of a new integration and control architecture
- Coupling of the main functions of the ACSYNT system to the new optimization environment

Through the use of the Dynamic Integration System, the core functions of ACSYNT have been offered to the user and accessible to the optimization routines. Furthermore, the problems associated with the original ACSYNT/Optimization structure have been alleviated. Some of these original problems include:

- Lack of process feedback from the optimizer

- Poor access to optimization control variables.
- No visualization facilities for results
- No configuration management for multiple problems
- Lack of consistent integration approach

Through the development of the Optimization Workbench and the use of the Dynamic Integration System, these problems have been corrected and the facilities for performing aircraft optimization studies using ACSYNT has been improved.

9.4 Review of Contributions

The research that was necessary to complete the proposed plan resulted in several contributions in key areas of the system design and optimization communities. Specifically, this work contributes in three main categories:

- Design Information Handling
  This work presents the methodology and a set of software tools for handling multiple optimization problems and presenting information regarding the inter-relationship of these optimization configurations. This method brings the notion of configuration management to the optimization community.
- Multivariate Data Analysis of Optimization Results
  A visualization method has been developed to present the user with a concise view of a set of analysis results. This method allows for the visualization of

multidimensional results and provides valuable insight into the design space of the each problem.

- Process Feedback

  An original approach has been taken to updating the results of an optimization process as it commences. Through the use of graphical interfaces, an environment has been constructed to give the user unique real-time feedback of the process and the state of all the variables in the problem.

References

[1] Frank, P. D., Booker, A. J., Caudell, T. P., Healy, M. J., "A Comparison of Optimization and Search Methods for Multidisciplinary Design," AIAA Paper 92-4827-CP, 1992.

[2] ACSYNT Institute, "ACSYNT 3.0: User Guide and Installation Manual," Virginia Tech, 1995.

[3] Myklebust, A., Gelhausen, P., "Putting the ACSYNT on Design," *Aerospace America*, Sept. 1994, pp 26-30.

[4] Vanderplaats, G. N., Madsen, L. E., "COPES - A FORTRAN Control Program for Engineering Synthesis - Research Report for Period Ending 1981," NASA Ames Research Center, March, 1982.

[5] Perry, G., *C++ By Example*, Que Programming Series, Que Publishing, 1992.

[6] Vanderplaats, G. N., "CONMIN - A FORTRAN Program for Constrained Function Minimization," NASA TMX 62-282, NASA Ames Research Center, Moffett Field, CA, August, 1973.

[7] Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design*, McGraw-Hill, 1979.

[8] "DOC Users Manual," VMA Engineering, Colorado Springs, CO, 1994.

[9] Vanderplaats, G. N., "Automated Optimization Techniques for Aircraft Synthesis," AIAA 76-909, AIAA Aircraft Systems and Technology Meeting, Dallas, TX, Sept., 1976.

[10] Wampler, S. G., "Development of a CAD System for Automated Conceptual Design of Supersonic Aircraft," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, May, 1988.

[11] Myklebust, A., Gelhausen, P., "Improving Aircraft Conceptual Design Tools - New Enhancements to ACSYNT," AIAA Paper 93-1228, Monterey CA, August, 1993.

[12] Wampler, S. G., Myklebust, A., Jayaram, S., Gelhausen, P., "Improving Aircraft Design, A PHIGS Interactive Graphics Interface for ACSYNT," AIAA Paper 88-4481, Sept., 1988.

[13] Hasan, S. "An Object-Oriented, PHIGS-based Internal Layout Module for Aircraft Design," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Dec., 1993.

[14] Jayaram, U. "Extracting Dimensional Geometric Parameters from B-Spline Surface Models," Ph.D. Dissertation, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Nov., 1991.

[15] Jones, R. W., "Intersection and Filleting of Non-Uniform B-Spline Surfaces," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Jan., 1991.

[16] Kelly, J. H., "Rule-Based Fuselage and Spine and Cross-Section Methods for Computer-Aided Design of Aircraft Components," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Dec., 1993.

[17] Malan, P., "Evaluation and Upgrade of Inlet and Afterbody Drag Calculation Methods for ACSYNT," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Jan., 1989.

[18] Marcaly, F. W., "Data Reduction and Knot Removal for Non-Uniform B-Spline Surfaces," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, April, 1991.

[19] Myklebust, A., Mahan, J. R., Jayaram, S., Wampler, S. G., Grieshaber, M. M., Taylor, A., and Kolady, K., "A CAD System for Automated Conceptual Aircraft Design," Final Report to NASA Ames Research Center, Grant Number NAG-2-461, July 11, 1988.

[20] Rivera, F., "An Object-Oriented Method for the Definition of Mission Profiles for Aircraft Design," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, Dec., 1993.

[21] Schrock, E. V., "A PHIGS-Based Spreadsheet for Conceptual Design," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, May, 1991.

[22] Squire, D. J., "Afterbody Drag Prediction for Conceptual Aircraft Design," M.S. Thesis, Mechanical Engineering Dept., Virginia Tech, Blacksburg, VA, May, 1992.

[23] Malone, B., "High-Speed Civil Transport Study Using ACSYNT," AIAA 93-4006, Montery, CA, Sept, 1993.

[24] Gallman, J. W., Kaul, R. W., Chandrasekharan, R. M., Hinson, M. L., "Optimization of an Advanced Business Jet," AIAA Paper 94-4303CP, Panama City, FL, Sept. 1994.

[25] Hwang, A., Vanderplaats, G. N., "A Generalized Multidisciplinary Optimal Design Tool for Windows Based Computer Platforms," AIAA Paper 94-4370, Panama City, FL, Sept 7-9, 1994.

[26] "DOT Users Manual," VMA Engineering, Colorado Springs, CO, 1994.

[27] Michaud, G. H., Modrey, J., "A Designer-Augmented Optimization Strategy: Concept and Implementation," ASME Paper 75-DET-99, ASME Design Engineering Technical Conference, Washington, DC, Sept. 17-19, 1975.

[28] Woyak, S. A., "An Object-Oriented Methodology and Supporting Framework for Creating Engineering Software Using Dynamic Integration," Ph.D. Dissertation, Mechanical Engineering, Virginia Polytechnic Institute and State University, 1995.

[29] Woyak, S. A., Myklebust A., "Functionality and Data Integration of Software Modules Through Dynamic Integration," Accepted for Publication in *Integrated Computer-Aided Engineering*.

[30] Woyak, S. A., Myklebust, A., Malone, B., "An Architecture for Creating Engineering Applications: The Dynamic Integration System," Proceedings of the 15th Annual International Computers in Engineering Conference, Boston, MA, Sept. 17-20, 1995.

[31] Kroo, I., "An Interactive System for Aircraft Design and Optimization," AIAA Paper 92-1190, 1992 Aerospace Design Conference, Irvine, CA Feb 3-6, 1992.

[32] Kroo, I., Takai, G., "A Quasi-Procedural, Knowledge-Based System for Aircraft Design," AIAA Paper 88-4428, Atlanta, GA, Sept., 1988.

[33] Rowell, L., Schwing, J. L., "Software Tools for the Integration and Execution of Multidisciplinary Analysis Programs," AIAA 88-4448, AIAA/AHS/ASEE Aircraft Design Systems and Operations Meeting, Atlanta, GA, 1988.

[34] Buckley, M. J., Fertig, K.W., Smith, D. E., "Design Sheet: An Environment for Facilitating Flexible Trade Studies During Conceptual Design," AIAA 92-1191, Aerospace Design Conference, Irvine, CA, Feb. 1992.

[35] Gonda, M., Fertig, K. W., Teeter, R. J., "Aerospace Conceptual Vehicle Design Using an Intelligent Design and Analysis Environment: Design Sheet," AIAA 1992, Aircraft Design Systems Meeting, Hilton Head, SC, 1992.

[36] Parkinson, A., "OptDesX User's Guide," Design Synthesis Inc., 1995.

[37] Townsend, J. C., Weston, R. P., "A Programming Environment for Distributed Complex Computing: An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project," NASA TM-109058, 1993.

[38] Weston, R. P., Townsend, J. C., Eidson, T. M., Gates, R. L., "A Distributed Computing Environment for Multidisciplinary Design," AIAA Paper 94-4372, Panama City, FL, Sept 7-9, 1994.

[39] Renaud, J. E., Batill, S. M., Brockman, J. B., "NDOPT: Multidisciplinary Design Technology Development; A Comparitive Investigation of Integrated Aerospace Vehicle Design Tools," MDO Review Meeting, NASA Langley, Oct., 1995.

[40] Daum, A., Wolf, D. M., "TRANSYS- A Multidisciplinary Software System for Preliminary Design, Analysis, and Evaluation of Space Transportation Systems," AIAA Paper 94-4342 CP, Panama City, FL, Sept., 1994.

[41] Tong, S. S., Powell, D., Goel, S., "Integration of Artificial Intelligence and Numerical Optimization Techniques for the Design of Complex Aerospace Systems," AIAA Paper 92-1189, 1992 Aerospace Design Conference, Irvine, CA, Feb 3-6, 1992.

[42] Jennions, I. K., "Elements of a Modern Turbomachinery Design System," GE Aircraft Engines, 1993.

[43] Hale, M. A., Craig J., "Preliminary Development of Agent Technologies for a Design Integration Framework," AIAA 94-4297, Panama City, FL, Sept 7-9, 1994.

[44] Bouchard, E. E., "Concepts for a Future Aircraft Design Environment," AIAA Paper 92-1188, 1992 Aerospace Design Conference, Irvine, CA, Feb 3-6, 1992.

[45] Malone, B., Mason, W.H., "Multidisciplinary Optimization in Aircraft Design Using Analytic Technology Models," Journal of Aircraft, Vol 32, No. 2, pp. 357-364, March-April 1995.

[46] Mason, W. H., "Analytic Models for Technology Integration in Aircraft Design," AIAA 90-3262, Dayton, OH, Sept, 1990.

[47] Malone, B., Mason, W. H., "Aircraft Concept Optimization Using the Global Sensitivity Approach and Parametric Multiobjective Figures of Merit," Journal of Aircraft, Vol. 33, No. 2, pp. 444-445, March-April, 1996.

[48] Sobieszczanski-Sobieski, J., Haftka, R.T., "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," AIAA Paper 96-0711, 34th Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan. 15-18, 1996.

[49] Jones, C.V., "Visualization and Optimization," ORSA Journal on Computing, Vol. 6, No. 3, pp 221-257, Summer, 1994.

[50] Murphy, F. H., Stohr, E. A., Asthana, A., "Representation Schemes for Linear Programming Models," Management Science, Vol 38, No. 7, 964-991, 1992.

[51] Myers, B. A., "Creating User Interfaces by Demonstration," Academic Press, Boston, MA, 1988.

[52] Piela, P., "ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis," Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, 1989.

[53] Meyers, B. A., "Visual Programming, Programming by Example and Program Visualization: A Taxonomy," SIGCHI Bulletin, Vol. 17, No. 4, 59-66.

[54] Friedman, J. H., Tukey, J. W., "A Projection Pursuit Algorithm for Exploratory Data Analysis," IEEE Trans. Comp., C-23, pp. 881-890, 1974.

[55] Becker, R. A., Cleveland, W. S., "Brushing Scatterplots," American Statistical Association, Technometrics, Vol 29, No.2, May 1987.

[56] Carr, D. B., Littlefield, R. J., Nicholson, W. L., Littlefield, J. S., "Scatterplot Matrix Techniques for Large N," Journal of the American Statistical Association, Vol 82, No. 398, 1987.

[57] Cleveland, W. S., "Robust Locally Weighted Regression and Smoothing Scatterplots," Journal of the American Statistical Association, Vol. 77, pp 541-547, 1982.

[58] Cleveland, W.S., "The Many Faces of the Scatterplot," Journal of the American Statistical Association, Vol. 79, pp 807-822, 1984.

[59] Keiper, J., Wickham-Jones, T. "Designing Tools for Visualization and Optimization," ORSA Journal on Computing, Vol. 6, No. 3, pp. 273-277, Summer, 1994.

[60] Becker, R. A. Cleveland, W. S., "Dynamic Graphics for Data Anlaysis," Statistical Science, Vol 2., No. 4, pp. 355-395, 1987.

[61] Cleveland, W.S., "Research in Statistical Graphics," American Statistical Association, Vol 82, No.398, pp. 419-423, June 1987.

[62] Simpkin, D., Hastie, R., "An Information-Processing Analysis of Graph Perception," American Statistical Association, Vol 82, No.398, pp 454-465, June 1987.

[63] Bell, P., "Visualization and Optimization: The Future Lies Together," ORSA Journal on Computing, Vol. 6, No. 3, pp. 258-260, Summer, 1994.

[64] Jones, K. H., Randall, D. P., Cronin, C. K., "Information Management for a Large Multidisciplinary Project," NASA Langley Research Center, 1993.

[65] Fulton, R. E., Yeh, C., "Managing Engineering Design Information," AIAA 88-4452, 1988.

[66] Parks, C. H., "Tutorial: Reading and Reviewing the Common Schema for Electrical Design and Analysis," Proceedings of the 24th ACM/IEEE Design Automation Conference, pp. 479-483, 1987.

[67] Chen, P. S., "The Entity-Relationship Model: Toward a Unified View of Data," ACM Transactions on Database Systems, Vol. 1, No. 1, pp.9-36, March, 1976.

[68] Southall, J. W., "Development of Integrated Programs for Aerospace Vehicle Design (IPAD) - Information Processing Requirements," NASA Contract NAS1-14700, NASA CR-2981, Boeing Commercial Airplane Company, June, 1977.

[69] Shoval, P., "Essential Information Structure Diagrams and Database Schema Design," Information Systems, Vol. 10, No. 4, pp. 417-423, 1985.

[70] Dittrich, K. R., and Lorie, R. A., "Object-oriented Database Concepts for Engineering Applications," Proceedings of IEEE Computer Aided Technologies, pp. 321-325, 1985.

[71] Buzzi-Ferraris, G., *Scientific C++: Building Numerical Libraries the Object-Oriented Way*, Addison-Wesley,1993, New York, NY.

[72] Prata, S., *C++ Primer Plus*, Waite Group,1991, Mill Valley, CA.

[73] Cellier, F. E, Zeigler, B. P., Cutler, A. H., "Object Oriented Modeling: Tools and Techniques for Capturing Properties of Physical Systems in Computer Code." IFAC Symposium, Swansea, UK, July 15-17, 1991.

[74] Ryckbosch, J., "ZAZIE In the World of Objects: An Object Oriented Experience in Scientific Optmiziation," Proceedings from Software Engineering & Its Applications, Third International Workshop, Tolouse, France, Dec. 3-7, 1990.

[75] Yokell, M. R., Baker, T., "An Object-Oriented Approach to Aircraft Performance Analysis," AIAA Flight Mechanics Conference, Arizona, Jul, 1994.

[76] Kodiyalam, S., Graichen, M., Connell, I. J., Finnigan, P. M., "Object-Oriented Optimization-Based Design of Satellite Structures," AIAA Journal of Spacecraft and Rockets, Vol 31, No. 2, Mar-Apr. 1994.

[77] Aurora, J. S., *Introduction to Optimum Design*, McGraw-Hill, New York, 1989.

[78] Blanchard, B., Fabrycky, W.J., *Systems Engineering and Analysis*, Prentice Hall, 1990.

[79] Kaufman, M., Balabanov, V., Burgee, S., Giunta, A., Grossman, B., Mason, W. H., Watson, L. T., Haftka, R. T., "Variable Complexity Response Surface Approximations for Wing Structural Weight in HSCT Design," AIAA Paper 96-0089, AIAA Aerospace Sciences Meeting, Reno, NV, Jan 15-18, 1996.

[80] Sobieszczanski-Sobieski, J., "On the Sensitivity of Complex, Internally Coupled Systems," AIAA Paper 88-2378, April 1988.

[81] Bell, P. C., O'Keefe, R. M., "Visual Interactive Simulation,- History, Recent Developments, and Major Issues, Simulation, Vol 49, No. 3, pp. 109-116, 1987.

[82] "Sammi - System Overview," Kinisix, Inc., 1995.

[83] Peters, L., "Introducing Dynamic Analysis Into Reengineering; Using Petri Nets for Dynamic Modeling," Enterprise Reengineering, Sept, 1995.

[84] "TemPRO System User's Guide: Release 3.0b," Software Consultants International, Ltd., 1995.

[85] Gill, Murray, Wright, *Practical Optimization,* Academic Press Limited, 1981.

[86] Goldberg, D., "Real-Coded Genetic Algorithms, Virtual Alphabets and Blocking," University of Illinois at Urbana-Champaign, Technical Report No. 90001, Sept, 1990.

[87] Gelhausen, P., *personal communication*, VPI&SU, Aug. 1995.

[88] *The American Heritage® Dictionary of the English Language, Third Edition* copyright © 1992 by Houghton Mifflin Company. Electronic version licensed from InfoSoft International, Inc. All rights reserved.

[89] Sammon, J., "A Nonlinear Mapping for Data Structure Analysis," IEEE Transactions on Computers, Vol. C-18, No. 5, May, 1969.

[90] Rosenbrock, H.H., "An Automatic Method for Finding the Greatest or Least Value of a Function," Computer Journal, Vol 3. pp 175-184, 1960.

[91] Bell, G. F., Curry, R. E., "The Role of Free Flight in Delta's Future," Delta Air Lines, AIAA Transportation TC Workshop, Washington, DC, March, 1996.

[92] Nash, E., Newman, P., NASA/NCEP, 1996 and private communication with David Lee, LMI, March 1996.

APPENDIX: SOFTWARE ANALYSIS AND DESIGN SPECIFICATION

This following is a specification including the analysis, design, and implementation plan for a software system for engineering analysis and optimization control. A high level problem definition is formulated from a general software deficiency identified within the engineering analysis and optimization community. The problem is then framed with a thorough analysis of the user requirements. These requirements are translated into a set of use cases, or interaction scenarios that help define the necessary functionality of the system. A design for the software system is presented using the results of the requirements decomposition. This design includes specification of class structures, subsystems and detailed interaction between the components within the subsystems. An implementation strategy is presented with the details of selected languages, platforms and integration with existing systems. Finally, a validation and verification plan is proposed identifying the critical-test components from the design.

Summary of Design

The proposed software system provides control of both the process and the formulation through a unique object-oriented design structured around the mechanics of both the process and problem formulation. This software design contains two distinct subsystems, the Configuration Control Subsystem and the Optimization Subsystem. The Configuration Controller is a set of tools that work together to offer setup, analysis and storage of multiple problem formulations. The Optimization subsystem offers setup and management of multiple optimization methods. The design of the system in this approach offers a highly reusable set of subsystems that have the flexibility to increase in individual functionality and to be combined with larger systems.

Selected Design Methodology

A consistent methodology for analysis and design must be followed to ensure that not only the right system is built, but that it is built properly. The design methodology used for this project is an adaptation of traditional systems engineering design approaches applied to software development.

Systems Engineering Approach to Software Development

Classical systems engineering philosophy can be easily applied to software development efforts. The key structure for a successful methodology is the integration of user requirements, and the traceability of those requirements through the entire design process.

- **Requirements Analysis**

  Understanding the planned capabilities of the proposed system is to compile and understand the user requirements for the system. A thorough analysis of all available user needs for the proposed system is crucial to a successful design. Requirements are collected and compiled in the form of operating scenarios, required data handling, envisioned capabilities, and anticipated look and feel,

- **Requirements Decomposition**

  Once the requirements for the system are identified, a decomposition of the requirements takes place to categorize and begin looking for key components and patterns. This stage of the design process begins to translate customer requirements into actual technical components.

- **System Design**

  The key components identified in the requirements decomposition are expanded and assigned responsibilities during the system design phase. A consistent methodology

for detailed software design is utilized within this phase. Identification and development of subsystems occurs at this stage of the design process. The interactions between each component as well as between the subsystems are detailed in preparation for the implementation phase.

- **Implementation**

  The implementation phase consists of choosing a language and platform and coding the design details of the system. Integration with existing systems and utilization of predefined libraries are addressed at this stage of the design.

- **System Testing**

  One necessary development step in the design process is the testing of the implementation. A sound strategy for testing the design provides the necessary traceability of the original user requirements to the final product.

Object-Oriented Design

Recent advances in software development techniques have provided new and unique strategies for design of large software systems. One such design technique is the object-oriented approach. Object-oriented design methodologies focus on designing highly reusable software components that are easier to maintain and enhance. A system

designed using an object-oriented approach is more adaptable to changing requirements at either the system level or the user level.

Responsibility Driven Design

One such object-oriented design methodology is the responsibility driven approach. The development of class hierarchies focuses on describing individual responsibilities and collaborations with other classes.

- **Class-Responsibility-Collaboration**

  A responsibility identifies what function the specific class accomplishes, where a collaboration identifies other classes that are necessary to complete the function. By identifying these functions and relationships, powerful subsystems can be developed with well defined tasks and interaction protocols.

- **Use-Case Approach**

  The use-case strategy is a mechanism to collect all the necessary user requirements and translate them into operating scenarios that help shape the design of the system. Use-cases can also be used in the validation and verification stages of the development of the system. By testing the system with use-cases, the original user requirements can be traced through the entire design to the finished product.

Deficiency

A general deficiency exists within the engineering design community for a software system that offers interactive management of numerical optimization processes along with management of the optimization problem formulation. There is a need for flexible access and control of multiple numerical methods along with how they are to be integrated. Furthermore, an adaptable, reusable method for specifying the numerical problem and how the problem is solved is needed. This entails an environment for interactively specifying the problem formulation along with the level of analysis or optimization that is to be applied to the problem. Levels of analysis range from the ability to parametrically inspect the design space to applying complex numerical algorithms for optimization.

Numerical methods are applied to single problem formulations, however, cataloging and management of multiple formulations is necessary to provide a complete project environment in which the user can assess the impact of different problem configurations. There exists a need for a system that maintains multiple problem formulations within one environment.

Visualization of the optimization results is key to understanding the process of design. There exists a need for a highly graphical environment for visualization of the process as well as the results from the process.

Objective

The objective of the systems analysis and design for the proposed software system is to develop an interactive, optimization environment in which existing engineering simulation and modeling software can be integrated and accessed by design-oriented facilities.

Scope of Problem

The need for the proposed system was identified within the engineering analysis and design community for conceptual and preliminary-based efforts. The scope of the problem is contained to integrating and controlling engineering software from this community.

System Analysis

A thorough compilation of a detailed set of user requirements must be derived from the problem definition and description of needs. This information provides the necessary level of detail for the design of the system.

The Computer Aided Optimization Scenario

A definition of the interaction scenario for which the user will be employing the system is necessary. The flowchart shown in Figure A.1 illustrates the general process that the user

will be executing. There are subsections to this chart that have multiple formulations, however, this process shows the necessary detail for understanding the broad requirements outline.

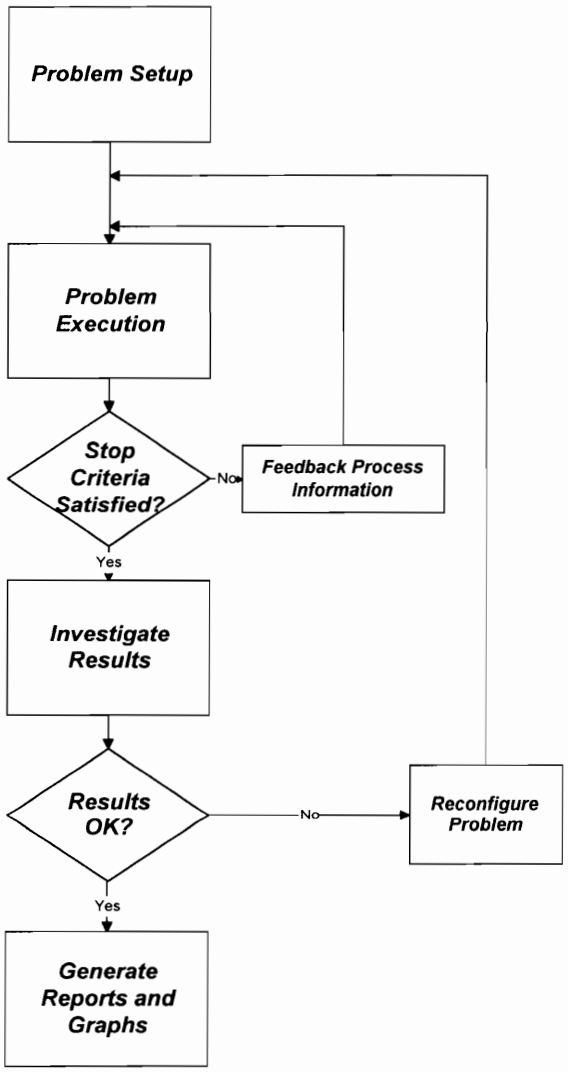


Figure A.1 General Problem Analysis Scenario

User Requirements

Several large categories of user requirements can be identified for the deficiency.

- *Problem Setup*
- *Problem Execution*
- *Execution Process Feedback*
- *Results Investigation*
- *Problem Reconfiguration*
- *Report Generation*

Problem Setup

Components for Use Case

The following sections describe the detailed user requirements for independent interaction cases. These individual cases will be assembled together in different ways to formulate complete use-case scenarios for the system.

Setup Components for the Design Information

User specifies output variable(s) to monitor

Pop-up dialogue that allows user to select output variable(s) from a scrolled list.

Options include:

Formulate multi-objective output.
Apply scaling or weighting to the value of the output(s).
Select based on discipline (scope).
Select multiple variables

User specifies an output variable to constrain

Same as monitor case.

Additional UI needed to input limits.
Auto-set limits based on high/low %
Output UI shows current value compared to limits.

User specifies an output variable to optimize

Same as monitor case, except limited to choose one.
Additional UI needed for target values.
Output UI shows current value compared to starting point/target value.

User specifies an input variable

Pop-up dialogue that allows user to select an input variable from a scrolled list.

Options include:

Apply weighting effects to the inputs. (Weight the influence of the input)
Specify upper and lower bounds
Auto-set bounds for all based on high/low %
Set move limits

Auto-select based on discipline (scope)

(This mode selects all inputs registered with a specified discipline)

Auto-select based on sensitivity

(Chooses multiple input variables based on the relative sensitivity of each input with respect to the selected output variable)

Auto-select based on dependency

(Choose all inputs that the chosen outputs depend on.)

Setup Components for the Optimization Process

Parametric setup for multiple optimization runs

Range of input values for input variables
Number of and details for linear combinations for multiobjective
Range of limits for constraints

User selects optimization features

Method used
Min/max flag
Advanced setup
    Tolerances
    Iteration limits
    Numerical details for each method

Operating Modes: Use-Case Studies

Parametrics (1-D)

User specifies an output variable to monitor
User specifies an input variable
User specifies a range of input values for the input variables
User chooses PLOT or REPORT format
User executes process
User SAVES to a FILE
User PRINTS to printer
User Returns to MAIN

Parametrics (2-D)

User specifies output variable(s) to monitor
User specifies two input variables
User specifies a range of input values for each input variables
User chooses PLOT or REPORT format
User executes process

User SAVES to a FILE
User PRINTS to printer
User Returns to MAIN

Parametrics (n-D)

User specifies output variable(s) to monitor
User specifies multiple input variables
User specifies a range of input values for each input variable
User specifies REPORT format
User executes process
User SAVES to a FILE
User PRINTS to printer
User Returns to MAIN

Sensitivity Analysis

User specifies output variable(s) to monitor
User specifies multiple input variables
User specifies PLOT or REPORT format
User executes process
User SAVES to a FILE
User PRINTS to printer
User Returns to MAIN

Basic Optimization

User specifies output variable(s) to optimize
User selects output variables to constrain
User selects input variables from a selection list
User specifies bounding values for each input variable
User specifies move limits for each input variable
Constraint limits are assigned to each output selected as a constraint
User selects an optimization method to use
User changes detailed optimization method options as needed
User executes process
Data from the process can be stored in a database for comparison to previous designs

Parametric Optimization

User specifies output variable(s) to optimize
User selects output variables to constrain
User selects input variables from a selection list
User specifies bounding values for each input variable
User specifies move limits for each input variable
Constraint limits are assigned to each output selected as a constraint
Detailed parametric information is specified on either:
    Input Variable (Parametric Input)
    Objective function (Parametric Multiobjective)
    Constraint (Parametric Constraint formulation)
User selects an optimization method to use
User changes detailed optimization method options as needed
User executes process
Data from the process can be stored in a database for comparison to previous designs

Results Investigation Requirements

This stage of the optimization process provides the user insight into the current problem through detailed analysis of the results. Specific user requirements can be presented as questions about the design that should be easily answered through the system.

- *How did the design arrive at the optimum?*

    What constraints are affecting the design?
    What part of the design is limiting the results?
    Is the objective function correct?
    Where is the optimum in relation to the surrounding area?
    What was the convergence history?

- *Optimizer Results*

    How many iterations did the optimizer make?
    Select an algorithm

Did the design converge? If not, why?
Where did the optimizer spend the most CPU time?
Is the mathematics of the problem well conditioned?

- *Discipline Influence*

  % Freedom of discipline, or contributing analysis.
  % Total design influence
  % Configuration contribution
  % Requirements contribution

- *Optimal Design Identification Rating*

  Requirements Based (# Active requirements constraints)
  Configuration-Based (# Configuration limits active)
  State-based (# conditions (mission-based) operationally limited)

Reconfiguration

Reconfiguration occurs after a process is executed and continued modification is to take place. Either the process or the configuration can be modified.

Optimization Reconfiguration

Process Modification

Changing Limits/Options

The system must present the user with a set of options for the optimization setup. Such items include the numerical tolerances, iteration limits, etc.

Reformulation

Algorithm Method Selection

The user should be able to easily select or change the current optimization algorithm to be used for the solution of the problem.

Analysis Reconfiguration

Adding/Removing Variables

The user should be presented with an area to add or remove an input variable from a list of all the current input variables. The limits on the input variables must be easily modified.

Changing Analysis Methods

The system must provide a way for the user to add or remove analysis modules with associated input and output variables.

Report Generation

Output in the form of graphs, tables, and report should be accessible for compilation into one basic document that can be saved to a file or printed to a printer.

Table Generation

The system must be able to generate a list of inputs and associated outputs in tabular form. The individual items in the report are specified by the user in a report generation setup interface.

Results Analysis Report

The system must be able to generate output in the form of a report regarding the configuration analysis that is performed for the pre and post optimization cases.

Graph Generation

The system must be able to graphically present results for 1 and 2 variable parametrics and optimization cases.

System Design

In this phase of the system development, the user requirements are translated into a working set of technical entities. Object-oriented techniques provide a clear mechanism for this translation through noun and verb listings. All of the major noun phrases contained in the list of requirements were extracted and analyzed for potential class

Identification of Classes

Parameter

A parameter is a class that contains a dynamic variable along with knowledge about its bounds and current value. Input and Output Parameters are specialized versions of the Parameter.

Configuration

A configuration is a particular set of input and output parameters that comprise an optimization problem.

Project

A Project is a collection of multiple Configurations. This identification was made to allow for configuration management and control.

OptMethod

The OptMethod class controls interaction with one numerical optimization method.

OptController

The OptController is a class that adds, removes and manages multiple OptMethods.

Identification of Subsystems

Two distinct subsystems were identified for the Optimization Workbench. First, the Project class, with its associated configurations and results analysis capabilities, was a clear candidate to be a distinct subsystem. This subsystem is defined as the Configuration Control Subsystem. The other subsystem is defined as the Optimization Subsystem, in which the numerical optimization methods are stored and called upon for processing the configuration.

Identification of Class Design Type

There are four areas of design types that were identified. Each one is representative of a different aspect of the system at the implementation level.

System Classes

The system classes contain all the underlying architecture for the Configuration and Optimization subsystems. Theses classes are interacted with by the user through the User Interface Input and Output classes.

User Interface: Input Classes

This set of classes contain the necessary Motif interfacing to the user for input of configuration, project and optimization options. These classes have access to both the Configuration Controller and the OptController classes for setting internal options and data.

User Interface: Output Classes

The output classes of the user interface provide a set of methods for both the configuration and optimization subsystems to present results and information to the user. A special design type was chosen for this because the needs of the configuration for presenting complex optimization results graphically.

Utility Classes

Finally, utility classes are identified to perform system level duties such as file operations and access to numerical methods.

Class/Responsibility Development

The responsibility-driven design approach requires an analysis of each class for developed responsibilities and collaborations between other classes. Presented below is a generalized set of responsibilities for each class type and subsystem.

System Classes

Configuration Subsystem

It is the responsibility of the configuration subsystem to maintain the current problem formulation along with previous problem formulations and their results.

Optimization Subsystem

It is the responsibility of the optimization subsystem to maintain the suite of numerical algorithms for optimizing the problem configuration.

User Interface: Input Classes

It is the responsibility of the input class of the user interface to accept and process specific input details for each subsystem. These classes interface with each subsystem manager

User Interface: Output Classes

It is the responsibility of the output classes of the user interface to present the results of each subsystem to the user.

Utility Classes

The utility classes have the responsibility of supporting each subsystem with basic system-level capabilities.

Subsystems Definition

The OptWorkbench provides control of both the process and the formulation through two distinct subsystems, the Configuration Control Subsystem and the Optimization Control Subsystem. The Configuration Controller is a subsystem that is composed of a Project, a Configuration Analysis and Configuration History modules. These work together to offer setup, analysis and storage of multiple problem formulations, or configurations. The OptController subsystem offers setup and management of the multiple optimization methods that are available.

Configuration Control Subsystem

The Configuration class is where the actual optimization, or parametric analysis problem is specified. The notion of a configuration consists of the problem setup, including the selected design variables, constraints and objectives along with the current bounds and limits for each. Reconfiguring the optimization problem means editing or creating a new configuration. Multiobjective formulations for the merit function is within the domain of the Configuration Controller.

The Project class is simply a set of configurations. The OptWorkbench is flexible enough to offer control of more than one configuration. This makes running multiple optimization scenarios with different problem setups simple. Performing optimization analysis is an iterative process entailing different problem setups, constraint limits, and formulations for merit functions. The Project class stores each of these problem formulations, or Configurations, and allows the user to create a library of optimization problems.

The Configuration History module performs detailed cataloging for each configuration. A selected configuration has a history of the design changes made to each component of the problem. Detailed information pertaining to convergence, constraint violations and analysis feedback is collected and stored by the configuration History module.

In addition, when the user selects the History for a selected configuration, full access to the design parameter and analysis feedback tools is obtained. This means that the user can walk through the recorded design and view the results through the configuration feedback tools just as if the OptController were operating on it real-time.

The Configuration Analysis module is a supplement to the Project and is used to analyze the post-optimization results. Analysis of these results includes unique design ratings for what type of design was obtained, and what parts of the configuration are influencing the total design. Based on the type of analysis that is specified in the configuration, the design can be over-constrained, under-constrained, requirements-based, configuration-

based, or discipline based. The analysis of the discipline influence of the optimization results gives the designer information about what part of the problem formulation should be modified.

Optimization Control Subsystem

The OptController is the complimentary subsystem to the Configuration Controller. The OptController controls the process, or the actions that are taken on the configuration. The OptController manages the multiple optimization methods that are integrated into the system, and controls the operation of each method. The notion of a 'process' is embodied in the OptController class as well. A process is the 'mode', or the type of analysis (parameter sweeps, optimization, etc.) that is to be performed on the design variables.

The OptMethod is a base class that is used to contain the separate optimization algorithms used by the OptController. Each OptMethod class has standard functions that allow the combination and collaboration of multiple methods simple and straightforward. This is accomplished at the higher level by the OptController.

There are several operating modes for which the OptController is designed. A user might not wish to employ optimization algorithms, rather simply run a parametric sweep of several variables and monitor several outputs to create a table of data for a report. The OptController handles the scheduling and the setup of the parametric sweep.

Parametric optimization setups are handled by the OptController. When a parametric optimization process is specified, a range of input values can be specified for a constraint or design variable limit, and the optimization process is executed for each value in the range. This gives the designer more decision making power through multiple optimization processes. Results of parametric optimization are families of optimal results. This shows disciplinary influence factors being directed through the configuration.

Dynamic Integration System Strategy

The Dynamic Integration System will be utilized within the configuration subsystem to communicate with the integrated modeling and simulation codes underlying the configuration. Specifically, the Parameter class will contain Dynamic Variables that can be linked to any analysis active in the system.

Implementation

The selection of language for the system must support object-oriented features. The language must be standard, portable, and robust. The interfacing techniques must also be programmed in a language that is widely supported on multiple platforms.

Development Environment

For the programming of the system, the C++ language will be used. This will be implemented and compiled under AIX Version 3.2.5 operating system installed on IBM RS6000 PowerStation 350 and PowerServer 980 system hardware. The system will also be tested on HPUX Version 9.0.3 installed on HP 735 hardware and Irix Version 5.2 installed on Silicon Graphics Indy workstations.

Interfacing Language

The Motif version 1.2 system libraries will be used for user interfacing. Library calls to the interface system will be made from the C++ software system.

Integration with Existing Framework

Dynamic Integration

Libraries for the Dynamic Integration System exist on each platform on which the system will be tested. DIS libraries will be linked with the base system to provide the required functionality.

Utilizing Existing Numerical Algorithms

Design Optimization Tools (DOT)

The main optimization algorithm supported in the system will be supplied through linked FORTRAN libraries of DOT.

Analysis Test Suite

The ACSYNT analysis modules will be used as test cases for the modeling and simulation integration and problem formulation. Aircraft design problems will be formulated and tested in the new environment using the ACSYNT analysis capabilities .

Validation and Verification

Once each subsystem and resulting component of the subsystem is developed, testing can take place. Validation of the functions of each class will include correctness of results from algorithms, methods and feedback.

Application of Use-Cases for Testing

Testing will be accomplished by reviewing the detailed use-case scenarios and testing for thorough user requirements traceability by performing walk-throughs of each scenario.

Additional Information

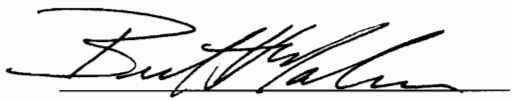
To aid in the software design process, a database for management of the class-responsibility-collaboration (CRC) system was developed on a Macintosh computer using the FileMaker Pro database system. This additional information is available from the author upon request.

VITA

The author was born and raised in West Virginia by his parents, Matz and Elizabeth. The author is married to a wonderful woman, Edna, who lets him pursue his life-long hobby of fishing. At every opportunity, the author can be found jetting across the closest lake at 60 mph in his boat, searching for that citation smallmouth.

In the spring of 1995, the author was involved in the formation of Phoenix Integration, Inc., a software company dedicated to providing integration and analysis software for the engineering community. Mr. Malone is the President of Phoenix Integration.

Upon graduation, the author will don suit and tie and set out to build an empire with Phoenix Integration. After an early retirement at age 48, the author will move to Wyoming and buy thousands of acres of ranch-land.



Brett Malone, 1996