AFIT/GOR/ENS/94M-01

AD-A278 574

APPLICATION OF SEQUENTIAL QUADRATIC
PROGRAMMING TO LARGE-SCALE
STRUCTURAL DESIGN PROBLEMS

THESIS

Mark Aaron Abramson
Captain, USAF

AFIT/GOR/ENS/94M-01

Approved for public release; distribution unlimited

Best
Available
Copy

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Master's Thesis

4. TITLE AND SUBTITLE

APPLICATION OF SEQUENTIAL QUADRATIC PROGRAMMING TO LARGE-SCALE STRUCTURAL DESIGN PROBLEMS

5. FUNDING NUMBERS

6. AUTHOR(S)

Mark Aaron Abramson, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/94M-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Vipperla Venkayya
WL/FIBRA
Wright-Patterson AFB OH 45433-0000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Large-scale structural optimization problems are often difficult to solve with reasonable efficiency and accuracy. Such problems are often characterized by constraint functions which are not explicitly defined. Constraint and gradient functions are usually expensive to evaluate. An optimization approach which uses the NLPQL sequential quadratic programming algorithm of Schittkowski, integrated with the Automated Structural Optimization System (ASTROS) is tested. The traditional solution approach involves the formulation and solution of an explicitly defined approximate problem during each iteration. This approach is replaced by a simpler approach in which the approximate problem is eliminated. In the simpler approach, each finite element analysis is follwed by one iteration of the optimizer. To compensate for the cost of additional analyses incurred by the elimination of the approximate problem, a much more restrictive active set strategy is used. The approach is applied to three large structures problems, including one with constraints from multiple disciplines. Results and algorithm performance comparisons are given. Although not much computational efficiency is gained, the alternative approach gives accurate solutions. The largest of the three problems, which had 1527 design variables and 6124 constraints was solved with ASTROS for the first time using a direct method. The resulting design represents the lowest weight feasible design recorded to date.

14. SUBJECT TERMS

Optimization; Structural Optimization; Nonlinear Programming; Sequential Quadratic Programming, Active Set Strategies

15. NUMBER OF PAGES

84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/GOR/ENS/94M-01

APPLICATION OF SEQUENTIAL QUADRATIC PROGRAMMING TO LARGE-SCALE STRUCTURAL DESIGN PROBLEMS

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degrees of

Master of Science in Operations Research,

Master of Science in Applied Mathematics

Mark Aaron Abramson, B.S.

Captain, USAF

March, 1994

THESIS APPROVAL

STUDENT: Mark A. Abramson, Captain, USAF CLASS: GOR-94M

THESIS TITLE: Application of Sequential Quadratic Programming to Large-Scale Structural Design Problems

DEFENSE DATE: 24 February 1994

COMMITTEE:

Name/Title/Department	Signature
Advisor: JAMES W. CHRISSIS Associate Professor of Operations Research Department of Operational Sciences	
Reader: ROBERT A. CANFIELD, Capt, USAF Assistant Professor of Aeronautical Engineering Department of Aeronautics and Astronautics	R. A. Canfield
Reader: LEE J. LEHMKUHL, Major, USAF Assistant Professor of Operations Research Department of Operational Sciences	Lee J. Lehmkuhl

Acknowledgements

I express my sincere appreciation to all those who helped make this research effort such a success. In particular, my thesis committee was extremely supportive and positive, even when I was not. My advisor, Dr. James Chrissis, provided excellent instruction and insight into the vital optimization issues. His guidance and encouragement led me to learn more in a short time than I ever expected. My readers, Major Lee Lehmkuhl and Captain Robert Canfield, lent their particular expertise to the project as technical editors, keeping the document thorough and precise. I owe a special thanks to Captain Canfield, whose help in area of structures and Automated Structural Optimization System (ASTROS) were invaluable to the success of this project.

In addition to my committee, several individuals at Wright Laboratories deserve sincere thanks for their efforts on my behalf. My research sponsor, Dr. Vipperla Venkayya willingly made available his expertise, real-world perspective, and an extra office with computer. Also at Wright Labs, Vicki Tischler, Steve Pitrof, and Ray Kolonay were tremendously helpful and supportive in helping me learn the basics of ASTROS, editing data files, and giving me valuable debugging suggestions. Also deserving of thanks is Bob Wolfe, the Convex system administrator, for his efforts and understanding while I used up most of his system memory for almost four weeks.

Special thanks also go to my good friends, Curtis Martin, who assisted in implementing a Unix shell to run my software, and Craig Willits, who lent his superior skills as an extra editor.

Finally and foremost, I thank my dear wife, Shelly, whose enduring love and sacrifice were, without question, the difference in making this project a success.

Thank you all for making this brutal experience so marvelous.

Mark Aaron Abramson

Table of Contents

List of Figures

List of Tables

AFIT/GOR/ENS/94M-01

Abstract

Large-scale structural optimization problems are often difficult to solve with reasonable efficiency and accuracy. Such problems are often characterized by constraint functions which are not explicitly defined. Constraint and gradient functions are usually expensive to evaluate. An optimization approach which uses the NLPQL sequential quadratic programming algorithm of Schittkowski, integrated with the Automated Structural Optimization System (ASTROS) is tested. The traditional solution approach involves the formulation and solution of an explicitly defined approximate problem during each iteration. This approach is replaced by a simpler approach in which the approximate problem is eliminated. In the simpler approach, each finite element analysis is follwed by one iteration of the optimizer. To compensate for the cost of additional analyses incurred by the elimination of the approximate problem, a much more restrictive active set strategy is used. The approach is applied to three large structures problems, including one with constraints from multiple disciplines. Results and algorithm performance comparisons are given. Although not much computational efficiency is gained, the alternative approach gives accurate solutions. The largest of the three problems, which had 1527 design variables and 6124 constraints was solved with ASTROS for the first time using a direct method. The resulting design represents the lowest weight feasible design recorded to date.

APPLICATION OF SEQUENTIAL QUADRATIC PROGRAMMING TO LARGE-SCALE STRUCTURAL DESIGN PROBLEMS

I. Introduction

1.1 Structural Optimization

Engineering optimization problems are characterized by the optimization of some design criterion subject to various design constraints. If the problem involves optimizing the design of a structure (i.e., a system consisting of spars, trusses, beams, etc.), then it is often referred to as a ***structural optimization*** problem.

In most structural design problems, the goal is to find the design vector $x \in \mathcal{R}^n$ which minimizes the value of an objective function f (typically weight of the structure), such that certain behavioral and performance requirements or constraints are met [7:79]. This is expressed mathematically in terms of the following general nonlinear programming (NLP) model:

$$\text{(NLP1)} \qquad \min f(x)$$

subject to

$$g_j(x) = G_j(x) - \bar{G}_j \geq 0, \quad j = 1, 2, \ldots, m$$

$$x_l \leq x \leq x_u,$$

where m is the number of constraints, n is the number of design variables, $\bar{G}$ is a vector of constraint limits, and x_l, x_u are design variable lower and upper bound vectors, respectively. It is assumed that the functions f and $g_j (j = 1, \ldots, m)$ are twice continuously differentiable, and that design points exist which satisfy the Karush-Kuhn-Tucker (KKT) necessary conditions for optimality. Also, since constraint functions in structural optimization problems are often highly nonlinear, an optimal solution can never be guaranteed to be global.

Structural optimization problems can be quite large — on the order of hundreds of design variables and thousands of constraints — and, therefore, difficult to solve. Large problems are often found in the design of aerospace structures, such as airframes and satellites. One computer program which is used in the design of aerospace structures is ASTROS (Automated Structural Optimization System) [20]. ASTROS is unique in that it "combines mathematical optimization algorithms with traditional structural analysis disciplines such as static forces, normal modes, static aeroelasticity, and dynamic aeroelasticity (flutter), all in a finite element context, to perform automated preliminary design of an aircraft structure" [20:i]. This integration of many different engineering disciplines is often called an *integrated* or *multidisciplinary environment.* Figure 1.1 shows the complex interrelationships which exist among the different disciplines within ASTROS.

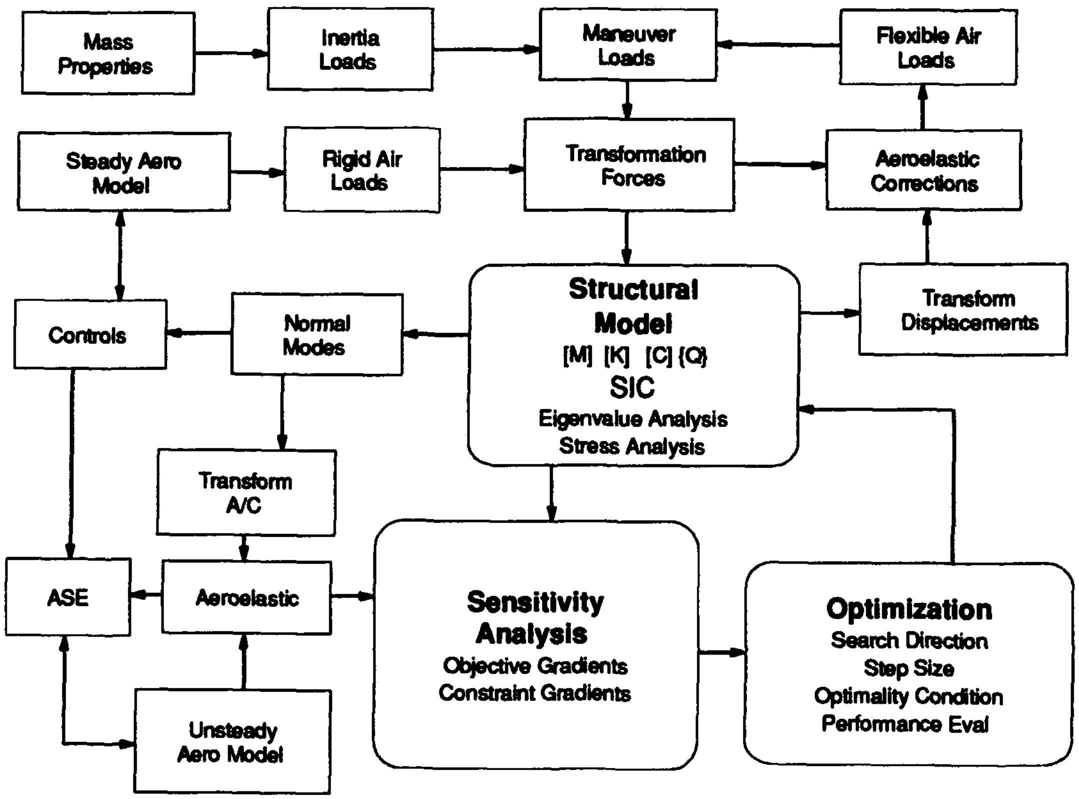

Figure 1.1 ASTROS Interdisciplinary Design

This thesis addresses a modification to the ASTROS software, that of applying a sequential quadratic programming (SQP) algorithm to the structural analysis process in an effort to increase the efficiency of ASTROS in solving very large problems. For this research, airframes are the specific type of structure considered (although other structures are discussed). Airframe design may require additional constraints and variables so that the aircraft can fly and perform its mission; even so, problems and techniques discussed in this document are easily generalized to many other structures.

Most aircraft structures consist of special types of components such as spars, ribs, stiffeners, skins, and so on. Descriptions of these items can be found in many mechanical engineering design textbooks (e.g., see [39]). Examples of typical structures are shown as "wire models" in Figures 1.2 and 1.3. The following design aspects are assumed known:

- Geometry of the structure
- Structural concept (includes numbers of spars, ribs, stiffeners, etc.)
- Materials used to build the structure
- Flight Conditions (includes maneuver requirements, such as takeoff, pullout, roll, and landing, as well as inertial and aerodynamic loads).

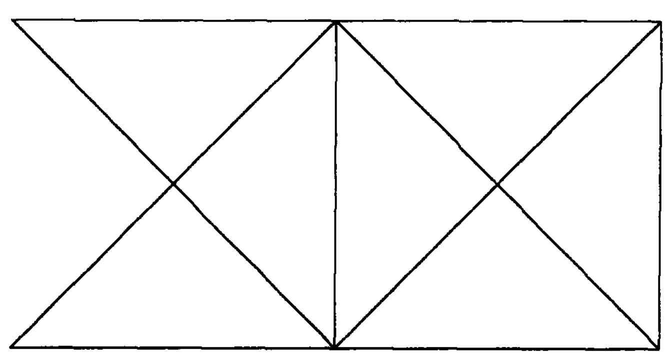

Figure 1.2 Example of a structure: 10-bar truss

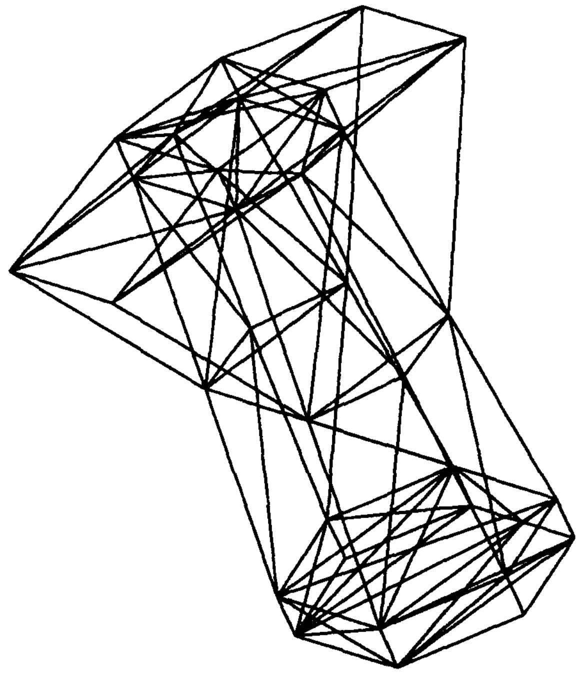

Figure 1.3 Example of a structure: ACOSS Satellite

Design Variables. The vector of design variables, x, may represent any of the following aspects of the structure:

- Thickness of skin, spars, and ribs
- Areas of spar caps, rib caps, and posts
- Composite fiber orientation.

Constraints. In order to function properly, a structure must function within certain design limitations. Aircraft structures are typically constrained by:

- Strength
- Stiffness (deflections)
- Natural frequency
- Flutter speed
- Divergence speed

- Lift-curve slope
- Control surface effectiveness.

For a description of these concepts, see [1] or [39].

Optimization Problem. A more specific representation of the structural optimization problem (NLP1) can now be given:

$$\min f(x)$$

subject to

$$\begin{array}{rl} \sigma_j \leq \bar{\sigma}_j & \text{allowable stresses} \\ u_j \leq \bar{u}_j & \text{allowable deflections} \\ \omega_j \geq \bar{\omega}_j & \text{minimum fundamental frequency} \\ V_f \geq \bar{V}_f & \text{flutter speed} \\ V_{div} \geq \bar{V}_{div} & \text{divergence speed} \\ \frac{dC_L}{d\alpha} \geq \bar{C} & \text{lift-curve slope} \\ x_l \leq x \leq x_u & \end{array}$$

Structural optimization problems differ from traditional mathematical programming problems in that the constraint functions may not be explicitly defined [17]. At any new design point, finite element analysis (FEA) is used to construct a new set of constraint functions having the same form but different parameters as previous designs. Also, for most structural optimization approaches, sensitivity analysis is also required at each new design point. This requires calculation of gradients for each constraint. The computational cost of evaluating the constraints and gradients at each new design can be great if the problem is large. Therefore, for a solution technique to be most efficient, it must use as few function and gradient evaluations and FEA iterations as possible.

1.2 Approach

To improve computational efficiency, this research focuses on improvements in two areas:

1. An optimizer with better convergence properties, and

2. An efficient active set strategy.

An improved optimizer can speed convergence, thus reducing the number of iterations. An active set strategy can reduce the problem dimension and, therefore, the number of gradient calculations. Each is briefly discussed below.

1.2.1 Improving the Optimizer. One class of algorithms which has received a great deal of attention for its robustness and superior convergence properties is known as *sequential quadratic programming* (SQP). These methods are based on solving a linearly constrained quadratic subproblem (based on Taylor series approximations of the objective and constraint functions) during each iteration, the solution of which yields the search direction to the next design point. Many implementations have been shown to converge globally (from any initial design point) and superlinearly in a sufficiently small neighborhood of the optimum [2:803-804]. For these reasons, a SQP approach was used.

1.2.2 Active Set Strategies. An active set strategy is typically implemented as part of an optimization algorithm to make large problems more manageable. At any given feasible design point, each constraint can be either binding or nonbinding (i.e., for each j, either $g_j = 0$ or $g_j < 0$, respectively). The set of binding constraints, J, is referred to as the *active set.* If $g_j < 0$, then its Lagrange multiplier, denoted v_j, vanishes. This means that the gradient of each constraint not in the active set need not be calculated. Active set strategies exploit this advantage by only holding *active* a handful of constraints during each iteration, and eventually converging to the optimal active set (i.e., the actual active set at the optimal design point). These strategies are used within algorithms to reduce problem size at each iteration, thereby reducing computational cost.

For small problems, the savings is negligible, but large optimization problems cannot be solved efficiently without an active set strategy [40]. For example, a problem with 20 variables and 1000 constraints (assume no inconsistencies), would have no more than 20 active constraints at any time. Without an active set strategy, 20000 (20×1000) derivatives per iteration would need to be computed. An active set strategy would only require, at most, 400 (20×20) per iteration, a savings of 98 percent ($1 - 400/20000$). While this example is somewhat extreme, it shows the potential savings.

1.3 Purpose of Research

The purpose of this research is to adapt an SQP method with active set strategy that, when combined with FEA in an integrated environment, can improve algorithm performance for large problems as compared to currently used methods. More specifically, an SQP algorithm is combined in a loop with ASTROS in a new way in an effort to solve larger problems more efficiently. The approach is applied to three structures problems and performance is compared to that of ASTROS in terms of computer processing (CPU) time and number of iterations required.

1.4 Overview

The next chapter describes pertinent background information found in the literature. Chapter III describes the SQP algorithm used and how it was integrated within ASTROS to solve structures problems. Chapter IV gives comparative results of this implementation against the method currently used in ASTROS applied to the structures problems described in Appendix A. Finally, Chapter V gives conclusions and recommendations for areas of further research.

II. Literature Review

Methods for solving nonlinear structural optimization problems fall into one of two classes: *optimality criteria* (OC) and *mathematical programming* (MP) methods. MP methods generate a sequence of design points in the primal space converging to an optimal, feasible solution. OC methods make use of the dual problem and Kuhn-Tucker necessary conditions and iteratively converge to a design point that satisfies them.

OC methods can often solve large-scale problems with better computational efficiency, mainly because the dual problem has much smaller dimension; the dual variables or Lagrange multipliers are zero for inactive primal constraints. However, MP methods are more robust and reliable in terms of the classes and types of problems they can solve. One popular MP method that has shown promise, SQP, has more favorable convergence properties than other MP methods [2:803-804].

2.1 Optimality Criteria Methods

OC methods, also referred to as *indirect methods* [17:82], are often applied to large-scale problems. These methods have two main components [7:80]:

1. A set of necessary conditions which hold at optimality, and
2. An iterative redesign scheme in which successive designs converge to the set of necessary conditions.

To obtain the set of necessary conditions, the Lagrangian of (NLP1) is formed:

$$L(x) = f(x) + \sum_{j=1}^{m} v_j g_j(x),$$

where $v = (v_1, \ldots, v_m)$ denotes the vector of Lagrange multipliers.

Minimization of L yields the optimality conditions,

$$\frac{\partial L}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^{m} v_j \frac{\partial g_j}{\partial x_i} = 0, \qquad i = 1, \ldots, n \tag{2.1}$$

$$g_j \geq 0, \qquad j = 1, \ldots, m \tag{2.2}$$

$$v_j g_j = 0, \quad j = 1, \ldots, m \tag{2.3}$$

$$v_j \geq 0, \quad j = 1, \ldots, m \tag{2.4}$$

$$x_l \leq x \leq x_u. \tag{2.5}$$

Equation (2.1) reduces to

$$\sum_{j=1}^{m} e_{ij} v_j = 1, \qquad i = 1, \ldots, n, \tag{2.6}$$

where

$$e_{ij} = \frac{-\partial g_j / \partial x_i}{\partial f / \partial x_i}. \tag{2.7}$$

Special modifications are made if the denominator of Equation (2.7) is zero.

Finally, since the relationship of Equation (2.6) holds at any Kuhn-Tucker point, an iterative scheme can be used to converge to a solution. Many such schemes have been developed. One common approach, attributed to Khot, Berke, and Venkayya [16] is to multiply both sides of Equation (2.6) by x_i^2 and then take the square root resulting in the recurrence formula

$$x_i^{(k+1)} = x_i^{(k)} \left(\sum_{j=1}^{m} e_{ij}^{(k)} v_j^{(k)} \right)^{\frac{1}{2}}, \tag{2.8}$$

where $v_k = (v_1^{(k)}, \ldots, v_m^{(k)})$ are estimates of v. Note that as $x_k = (x_1^{(k)}, \ldots, x_n^{(k)})$ approaches a Kuhn-Tucker point, the iterates defined by Equation (2.8) get closer and closer together due to the term inside the parentheses approaching unity [44]. A more thorough treatment of OC methods, including some specific implementations of various recurrence formulas for x_k and v_k can be found in [15:319-328]. When Venkayya generalizes this method for multiple types of constraints, he uses a compound scaling algorithm to track violated, active, and inactive constraints ([6], [44]).

While OC methods have been successfully used to solve large problems ([6], [7]), they have many limitations. One main drawback is that they typically work well for specific types of constraints, but have difficulty computing Lagrange multipliers in problems with multiple constraints [7:79]. They also become increasingly inefficient as the number of active constraints approaches the number of design variables. This is because OC methods reduce the dimension by exploiting the dual problem; if the dual problem is nearly as large, little efficiency can be gained. OC methods typically converge quickly in the initial stages, but the step-size becomes more difficult to determine as the design approaches optimal [7:79]. In addition, convergence to the optimum is also not guaranteed [17:83]. Venkayya suggests that a hybrid method of OC for the first few iterations, and then an MP method for accuracy, could yield even better results [7:79].

2.2 Mathematical Programming Methods

Mathematical programming (or direct) methods are based on the following iterative scheme:

$$x_{k+1} = x_k + \alpha_k s_k, \quad k = 0, 1, \ldots,$$

where k is the iteration number, x_k is the design vector at the kth iteration ($k = 0$ is the initial design), s_k is a search direction vector, and α_k is the step size in the search direction. Computation of s_k and α_k varies among the many different MP methods. Discussion of specific MP methods, such as Frank-Wolfe, feasible directions, generalized reduced gradient, and gradient projection methods, can be found in most standard optimization or nonlinear programming textbooks (e.g., see [12] or [28]).

MP methods are generally more robust than OC methods. Many are *globally convergent* (i.e., they converge from any initial starting design point) and, unlike the OC methods, their convergence does not depend on the types of constraint functions used. However, many MP methods lack the efficiency and convergence rate necessary to be useful for solving large-scale problems.

2.3 Active Constraint Set Strategies

One approach for improving the efficiency of MP algorithms is to incorporate an active constraint set strategy. Indeed, large-scale problems cannot be solved without them ([40], [41]). This is because methods without such strategies must evaluate all constraints at every iteration; for very large problems this becomes intractable. Active set strategies deal only with a subset of the constraints at any time. This reduces the number of constraint gradient evaluations performed at each iteration, thus greatly diminishing computational cost.

2.3.1 General Active Set Algorithm. An active set strategy generally consists of the following steps (adapted from [23:354]):

1. Input initial design point and working constraint set.
2. Perform termination test (including test of KKT conditions). If point is not optimal, either continue with same working set or go to 7.
3. Compute a feasible search vector s_k.
4. Compute a step length α_k along s_k, such that $f(x_k) + \alpha_k s_k < f(x_k)$. If α_k violates a constraint, continue; otherwise, go to 6.
5. Add a violated constraint to the working set and reduce α_k to the maximum possible value that keeps feasibility.
6. Set $x_{k+1} = x_k + \alpha_k s_k$.
7. Change the working set (if necessary) by deleting a constraint, update all quantities (including $k = k + 1$), and go to step 2.

While the implementation of an active set algorithm can be very flexible, the key decision is determining how constraints are added and deleted from the working set (other details are typically defined by the type of algorithm using the strategy).

2.3.2 Add and Drop Rules. The constraints to be added or deleted are determined by specific add and drop rules, and it is these rules that make each strategy different [4:430]. Some strategies add more than one constraint at a time, while others add or drop only when necessary to continue toward optimality [9:221].

Lenard classifies active set strategies by defining two add and two drop rules as follows (adapted from [18]):

1. AWN (Add-when-necessary): Constraint is added to the working set only when necessary for feasibility.

2. AWP (Add-when-possible): Constraint is added whenever the line search is blocked by it.

3. DWN (Drop-when-necessary): Constraint is dropped only when necessary for continued improvement in the objective function.

4. DWP (Drop-when-possible): Constraint is dropped whenever reasonable improvement can be gained.

Rules 2 and 3 yield the *most constrained strategy*, while rules 1 and 4 give the *least constrained strategy* [18:86]. The least constrained methods usually perform faster and more efficiently since fewer gradients are evaluated at each iteration, but they tend to have problems with constraints cycling in and out of the working set ([9:221], [18:82], [22:270]). Clearly, the AWP and DWP rules suggest the possibility of adding or dropping several constraints at a time.

Although, in theory, these rules define the differences between active set strategies, algorithm performance depends chiefly on the drop rule [9:229-230]. In Lenard's study of strategies used for linearly constrained nonlinear programs (NLPs), the AWN-DWN method was discarded because the results were too close to the AWP-DWN strategy [18:86]. Das, Cliff, and Kelley considered three strategies in their research, all of which used an AWP approach [9:230]. Panier asserts that active set strategies used for linearly constrained NLPs differ only in the choice of descent direction and the constraint drop rule [22:270].

2.3.3 Cycling. The use of active set strategies which drop more than one constraint at a time can often induce a phenomenon called cycling or zigzagging. This occurs when constraints "cycle" in and out of the working set. Examples of this can be found in [8]. Without cycling, an algorithm is usually much faster if it can drop many constraints simultaneously; however, cycling slows the progress of the algorithm, thereby negating the benefits gained by the less restrictive strategy. Several approaches exist to alleviate this problem. For example, Das, Cliff, and Kelley use Zoutendijk's rule (see [47]), which keeps all previously dropped constraints in the working set until the algorithm converges to a stationary point [9].

2.3.4 Examples of Strategies. Four of the most common active set strategies are briefly described in this section.

Sargent and Murtagh use an AWP-DWP "worst-violator" strategy in which multiple constraints are dropped whenever a constraint is added or the optimum is obtained with respect to the current working set. When either occurs, the constraint with the most negative multiplier (called the *worst violator*) is deleted and the multipliers are recomputed. This is repeated until the working set contains only constraints with nonnegative multipliers [31].

Fletcher uses an AWP-DWN (most constrained) strategy in which the worst violator is dropped only when no more improvement can be made with the current working set. Only one constraint is dropped at a time [11]. This method is also employed in the gradient projection methods described in [28].

Rosen's strategy is also an AWP-DWN strategy that drops constraints one at a time. However, when a constraint is added, Rosen chooses as the constraint to be dropped the violator (constraint with a negative multiplier) which would produce the largest additional decrease of the objective function (provided the decrease is greater than the decrease without dropping) ([29], [30]).

The *ϵ-active* method is one which appears in the literature extensively (e.g., see [2], [27], or [33]). In this method, a constraint is considered active whenever it is infeasible or within ϵ of its limit. That is, the active set J is defined at the point x by

$$J = \{j \in I : g_j(x) \leq \epsilon \quad or \quad v_j > 0\},$$

where $I = \{1, \ldots, m\}$ and ϵ is some small user-specified tolerance value. In this scheme, the entire active set is determined anew prior to each iteration.

Structural design problems are frequently solved using an ϵ-active method because the constraint functions are implicit and can change between iterations. The natural consequence of this is that constraints held in the active set during one iteration may be inactive or infeasible during the next without any attempt to add or drop. Therefore, recomputing the active set before every iteration becomes the only reasonable strategy for structural optimization problems. An advantage of this method is that it avoids the problem of cycling [22].

2.4 Sequential Quadratic Programming

Among the large class of MP techniques (also referred to as direct methods), SQP methods have become very popular. They converge to a solution by solving a sequence of linearly constrained quadratic programming subproblems to determine each successive search direction. This problem is described mathematically ([7:79], [23:365-367]) as

$$\min s^T \nabla f + \tfrac{1}{2} s^T \mathbf{W}(x, v) s$$

subject to

$$g_j(x) + s^T \nabla g_j(x) \leq 0, \qquad j = 1, \ldots, m,$$

where v is a vector of Lagrange multipliers, and $\mathbf{W}$ is a positive definite matrix approximating the Hessian of the Lagrangian function, $\nabla^2 L$, at the current design point. If $\nabla^2 L$ is positive definite at s^*, then s^* is a local minimum [37:189].

Although many different SQP algorithms exist, they generally consist of the following steps (adapted from [23:369]):

General SQP Algorithm

1. Initialize (including starting point).
2. Solve the quadratic subproblem to determine the search vector, s_k.
3. Minimize a merit function along s_k to determine the step size, α_k.
4. Set $x_{k+1} = x_k + \alpha_k s_k$.
5. Perform the termination test; if criteria not met, go to step 2.

A merit function is used in place of the objective function to account for infeasible points along the line of search. It is constructed to reward optimality and penalize infeasibility simultaneously [23:367]. Examples of merit functions can be found in Powell [25], Schittkowski ([33],[34],[35]), or Reklaitis, Ravindran, and Ragsdell [28:218-241].

SQP methods have two advantages over many other MP methods. The first is global convergence. For most implementations, SQP methods have been proven to converge from any inital starting design [2:803-804]. The second feature is local superlinear convergence. In a neighborhood of the optimal solution x^*, the equation

$$\|x_{k+1} - x^*\| \leq \gamma_k \|x_k - x^*\| \tag{2.9}$$

holds, where $\gamma_k \in \mathcal{R}$ converges to zero [32:11–12]. A higher convergence rate means that fewer iterations are required to arrive at a solution. This is especially vital when the cost of testing a design point is computationally expensive.

2.5 SQP Methods with Active Set Strategies

One area that has drawn significant attention in recent years is the implementation of active set strategies with SQP. Comparative studies show these algorithms to perform favorably on more moderately-sized problems ([3], [7], [35], [40], [41]). With an efficient strategy, an SQP method should be able to handle much larger problems, including very large structures problems. Two prominent algorithms are the PLBA (Pshenichny-Lim-Belegundu-Arora) algorithm [2], based on Pshenichy's original work [27] and Schittkowski's NLPQL algorithm ([33], [34], [35]), based on earlier work done by Wilson [45], Han ([13],[14]), and Powell ([24], [25]). Each of these is briefly described in this section.

2.5.1 PBLA Algorithm. Pshenichny was the first to use SQP with an active set strategy [27]. Arora and two of his students, Lim and Belegundu, have improved it in two key ways. The first improvement is the use of second-order information. Pshenichny, uses $\mathbf{W} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. In the PBLA algorithm, $\mathbf{W}$ is a positive definite approximation to $\nabla^2 L$. This accelerates the convergence of the algorithm [3:1588].

The second improvement is a change of a descent function parameter. To determine the step size at each iteration, Pshenichny minimizes a descent function of the form $\theta(x) = f(x) + rV(x)$, where r is a scalar and V is a penalty function which measures infeasibility of x. The PBLA algorithm uses a different condition on r (necessary for proof of global convergence of the algorithm) that alleviates a small step size problem brought on by large values of r which often occur with Pshenichny's algorithm [3:1588].

2.5.2 NLPQL Algorithm. Han's SQP algorithm (based on the ideas of Wilson [45]) is based on a nondifferentiable exact penalty function ([13],[14]). Powell improved Han's method with a better method for updating $\mathbf{W}$ at each iteration. In the NLPQL algorithm, Schittkowski replaces the exact penalty function of Han and Powell with a differentiable augmented Lagrangian function [33:85-88]. Numerical results have shown that NLPQL is more efficient than the algorithms of Han and Powell in solving a variety of problems with up to 100 variables [35].

2.6 Comparative Studies

2.6.1 Structural Optimization Algorithms. Several relevant studies comparing algorithms for structural optimization appear in the literature. Perhaps the most comprehensive study to date is that of Schittkowski, Zillober, and Zotemantel. They compare the performance of eleven algorithms within the MBB-LAGRANGE (see [46]) structural optimization system on 79 test problems having up to 144 design variables and 1020 constraints [32]. Belegundu and Arora compare a variety of MP methods on structural design problems, including some moderately large problems. Both theoretical and numerical aspects are considered [3]. Thanedar, et al. discuss differences between various SQP algorithms and compare their performance using 17 structural design problems having up to 96 design variables and 1051 constraints; however, NLPQL is not included in the study [40]. Schittkowski compares computational performance of his algorithm to other MP codes on smaller problems, but against only one other SQP method (Powell's – see [24]), which does not use an active set strategy [35]. Canfield, Grandhi, and Venkayya compare the performance of several MP algorithms, along with an OC method, on moderately sized problems and large problems with up to 100 design variables in the ASTROS environment; however, once again, NLPQL was not tested [7]. In a comparison which included larger problems (200 to 1527 design variables and 722 to 6124 constraints), Canfield and Venkayya compare a generalized OC method to Vanderplaats' ADS algorithm within AS-

TROS. Unfortunately, the MP method ran out of memory before solving the largest of these problems [6]. Testing of some important algorithmic characteristics has also been accomplished, both for PLBA [41] and NLPQL [33]. Tseng and Arora, in particular, study how performance changes as specific parameters within the PLBA algorithm are altered [41]. However, to date, comparative studies on large-scale structural problems with hundreds of variables are scarce.

2.6.2 Active Set Strategies. Current literature on active set strategies is focused primarily on linearly-constrained quadratic programming (QP) (even NLP studies use primarily QPs). However, most studies are generally applicable to SQP, since the subproblem is a QP.

There are three particularly relevant studies of active set strategies. Lenard published the first specific comparative study on active set strategies for nonlinear problems. Her results show that, for the projection method she used, the least constrained strategies were superior for all but one of the test problems used. For one 30 variable problem, a 40 to 80 percent savings in computer time is observed [18].

Das, Cliff, and Kelley compare the strategies of Sargent, Fletcher, and Rosen to a proposed strategy. The proposed method is based on establishing rules for when more than one constraint can be safely dropped without cycling occurring. (Theoretical development is limited to the case where there are three or fewer active constraints.) Their results show that for a few simple quadratic problems, the least constrained strategies generally outperforms the most constrained [9].

Dax studied active set strategies used on linear least squares problems constrained only by simple variable bounds. In this case, the results show that dropping one constraint at a time is more efficient than dropping many [10].

2.7 Summary

A search of the literature has revealed that efficiently solving large (thousands of design variables and constraints) structural design problems is extremely difficult. OC methods can only be used effectively for certain specific classes of problems and do not always converge to a solution. To date, the more robust MP methods have not been as successful because they are computationally inefficient; however, they possess the best convergence properties (i.e., they generally converge for any initial starting design point and do not depend on any significant constraint characteristics). A more efficient MP algorithm could provide structural engineers a way to routinely solve very large problems without sacrificing robustness.

Of the MP techniques, SQP algorithms have the fastest convergence rates, and converge from any initial starting design. Studies show that algorithms, such as the well-known PBLA and NLPQL codes, which use SQP with an active set strategy, generally perform better on small and moderately-sized stuctures problems than other MP methods. Thus, the main thrust of this thesis becomes the application of an SQP algorithm with active set strategy to very large structures problems in an effort to improve algorithm performance.

III. Approach

3.1 Overview

This chapter describes the details of the research conducted. Schittkowski's SQP code, NLPQL, was chosen over Arora's PBLA code because of its special "reverse communication" logic which makes it much easier to apply universally to finite element codes (this is discussed further in Section 3.6.1). A driver program was needed to interact with the ASTROS system. In this chapter, the current and proposed optimization loops are contrasted, the NLPQL algorithm is described in greater detail, and an outline is given of how NLPQL and ASTROS were used in tandem.

3.2 ASTROS Optimization Loop

ASTROS (Automated Structural Optimization System) is a computer program used in multidisciplinary design of aerospace structures [20]. It combines nonlinear optimization techniques with finite element analysis (FEA) to arrive at an optimal, or at least a much improved, preliminary design. Figure 3.1 represents a traditional design approach which is used in ASTROS and many other structural optimization codes.

In this procedure, an initial FEA is performed, and constraints are deleted (or flagged as inactive) according to specific criteria chosen by the user. These criteria keep as active the most infeasible and binding (or close to binding) constraints while deleting those which are easily satisfied. More details are given in 3.6.2, as well as in the ASTROS user's and programmer's manuals ([20], [21]). ASTROS then calculates gradients for the "active" constraints and uses *approximation concepts* (see [38]) to construct an approximate problem formulated by linearizing the constraints in the *reciprocal* design space. The optimizer is then called to solve the simplified problem using Vanderplaats' ADS algorithm ([42], [43]). The key point is that the approximate problem is optimized at very little cost, because, it has fewer constraints and the functions are explicitly defined. After the approximate problem is solved, an FEA is performed to see if the full design meets the convergence criteria for the entire structure. Generally, fewer FEAs are required because all of the optimization iterations are done with the approximate problem at low cost. For most structural

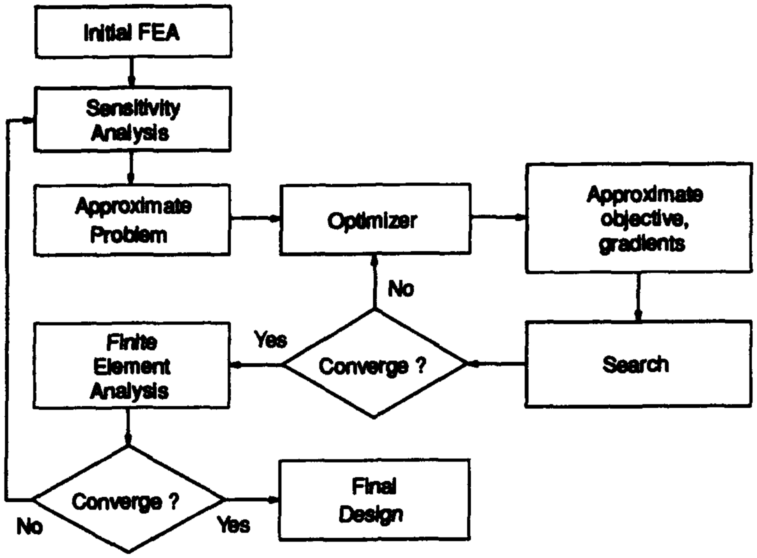

Figure 3.1 Typical Optimization Procedure

problems, this is very efficient because FEA is computationally expensive, particularly in problems which have constraints from several disciplines, such as aeroelasticity and flutter analysis. However, for large problems, solving the approximate problem can also become very expensive because the active set must contain a fairly generous number of constraints for which gradients are expensive to evaluate.

3.3 Alternative Optimization Loop

Figure 3.2 shows an alternative approach in which there is no approximate problem and an entire FEA is performed during each iteration of the optimization procedure. While this approach may seem less efficient due to a greater number of FEAs required, in practice, there could be significant savings for very large problems if a tight active set strategy is used in conjunction with a fast optimizer. This would potentially yield fewer gradient calculations per iteration, and the tradeoff could decrease computer CPU time. In ASTROS or any method similar to Figure 3.1, the constraint deletion procedure must necessarily be generous in flagging active constraints because optimization of an approximate problem may otherwise yield an infeasible next design. In some instances, the constraint deletion routine flags nearly all the constraints as active, making the approximate problem large.

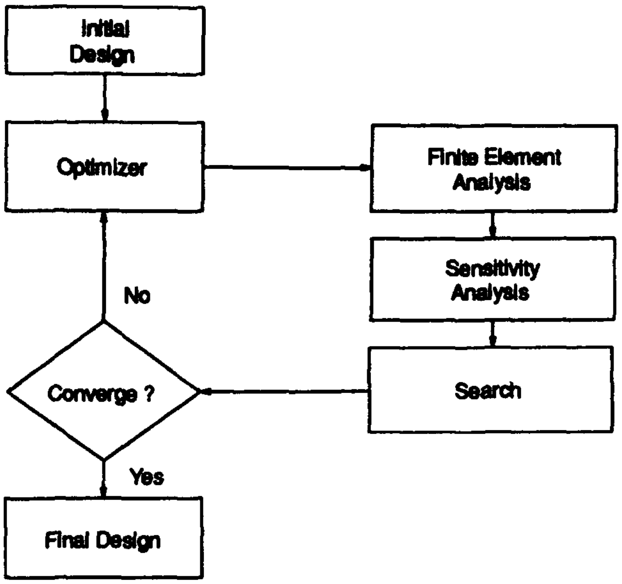

Figure 3.2 Alternative Optimization Procedure

3.4 Design of Investigation

NLPQL was implemented as the optimizer to solve structures problems using the approach described in Figure 3.2. This was done in a two-phase approach using ASTROS to conduct the structural analysis (FEA) and sensitivity analysis portions of the loop. The code was verified by solving the classic ten-bar truss problem (for a description, see [7] or [38]).

In testing the ASTROS-NLPQL implementation, there were two objectives. The first goal was to obtain an accurate solution. The second was to compare algorithm performance of the method against the traditional approach.

The ASTROS-NLPQL approach was applied to three structures problems provided by the Flight Dynamics Directorate of Wright Aeronautical Laboratories. The test problems chosen were:

1. 200 member plane truss (Tr200),
2. Intermediate complexity wing (ICW), and
3. High-altitude long endurance aircraft (HALE).

The Tr200 problem, which has 200 design variables and 2500 constraints, is a classic large problem with known solutions. The ICW problem, which has 350 design variables and 750 constraints, was chosen because it was the largest available problem with constraints from multiple disciplines. The HALE problem, with 1527 design variables and 6124 constraints, was the largest problem available. It has been solved, but only by OC methods [6:1041]. More detailed descriptions of these problems are given in Appendix A. Comparisons against ASTROS solutions were made, where possible, with respect to computer CPU time and number of iterations required.

3.5 The NLPQL Algorithm

In Chapter 2, a general SQP algorithm framework was presented. The second and third steps of the algorithm, namely solving the quadratic subproblem and determining the step size, typically vary among the different SQP approaches. NLPQL is a unique and very efficient implementation of SQP. The details of this algorithm are now presented.

3.5.1 The Quadratic Subproblem. Before explaining the details of the subproblem, it is necessary to redefine the original optimization problem (NLP1), as

(NLP2) $$\min f(x)$$

subject to

$$g_j(x) = 0, \qquad j = 1, \ldots, m_e$$
$$g_j(x) \geq 0, \qquad j = m_e + 1, \ldots, m$$
$$x_l \leq x \leq x_u.$$

Linearization of the constraints and quadratic approximation of the Lagrangian give the QP subproblem whose solution yields the next search direction s_k:

$$\min \; s^T \nabla f + \tfrac{1}{2} s^T \mathbf{W} s$$

subject to

$$g_j(x_k) + s^T \nabla g_j(x_k) = 0, \quad j = 1, \ldots, m_e, \tag{3.1}$$
$$g_j(x_k) + s^T \nabla g_j(x_k) \geq 0, \quad j = m_e + 1, \ldots, m$$
$$x_l - x_k \leq s \leq x_u - x_k,$$

where $x_k = (x_1^{(k)}, \ldots, x_n^{(k)})$, and **W** is updated during each iteration using a quasi-Newton method. Schittkowski uses BFGS updating [32:16], which, under relatively weak conditions, preserves the symmetric positive definiteness of **W** provided it is initially chosen symmetric positive definite [19:193-194]. This is easily satisfied by choosing $\mathbf{W} = \mathbf{I}$ as the initial estimate.

Schittkowski provides two improvements which make the algorithm more efficient. The first is the addition of an ϵ-active set strategy which reduces the number of unnecessary gradient calculations. The set of constraints is partitioned into the active set J and its complement K, defined respectively by

$$\begin{aligned} J_k &= \{1,\ldots,m_e\} \cup \{j \mid m_e < j \leq m,\ g_j(x_k) \leq \epsilon \text{ or } v_j{}^{(k)} > 0\}, \\ K_k &= \{1,\ldots,m\} \cap \{j \mid j \notin J_k\}, \end{aligned}$$

where $v_k = (v_1{}^{(k)}, \ldots, v_m{}^{(k)})$ is the vector of (approximate) Lagrange multipliers, obtained iteratively as explained in the next section (initially, $v_0 = 0$). As a second improvement, an additional variable δ is added to the subproblem in Equation (3.1) to alleviate possible inconsistent subproblems (constraint qualification does not hold) when the original problem (NLP2) has a solution. The QP subproblem given in Equation (3.1) is rewritten (after both modifications) as

$$\min \ s^T \nabla f + \tfrac{1}{2} s^T \mathbf{W} s + \tfrac{1}{2}\rho_k \delta^2$$

subject to

$$\begin{aligned} &(1-\delta) g_j(x_k) + s^T \nabla g_j(x_k) \left\{ \begin{array}{c} = \\ \geq \end{array} \right\} 0, \quad j \in J_k \qquad (3.2) \\ &g_j(x_k) + s^T \nabla g_j(x_{k(j)}) \geq 0, \quad j \in K_k \\ &x_l - x_k \leq s \leq x_u - x_k \\ &0 \leq \delta \leq 1, \end{aligned}$$

where the indices $k(j)$ correspond to gradients computed during previous iterations (i.e., previous design variable values). The penalty parameter ρ_k is added to reduce the influence of δ on the solution of Equation (3.2) [35:489-491]. If the subproblem is consistent, δ vanishes, giving back the original subproblem. By "perturbing" an inconsistent system slightly, the constraint qualification will hold for some value of δ, where $0 < \delta < 1$, and the variable is held as small as possible by including its square in the objective function.

The subproblem can be solved by any QP code. NLPQL calls QLD, a modified version of Powell's convex QP solver ZQPCVX (see [26]), which solves the unconstrained QP and then successively adds violated constraints until a minimum is reached. NLPQL assigns $\delta = 0$ unless QLD returns with an error message due to consistency problems.

3.5.2 Line Search Procedure. The subproblem yields a solution s_k (the search direction) with subproblem multipliers u_k. The next step is to choose a step size α_k to give the new iterate x_{k+1} and new Lagrange multipliers v_{k+1}. The new iterates are obtained for suitable α_k by the equations

$$x_{k+1} = x_k + \alpha_k s_k$$
$$v_{k+1} = v_k + \alpha_k(u_k - v_k).$$

In order to determine α_k such that the next iterate is feasible and represents a significant improvement, a merit or penalty function

$$\phi_k(\alpha) = \psi_{r_k}\left(\begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha \begin{pmatrix} s_k \\ u_k - v_k \end{pmatrix}\right)$$

is minimized, where $r_k = (r_1^{(k)}, \ldots, r_m^{(k)})$ is the vector of penalty parameters. NLPQL gives the option of using an L_1-exact penalty function

$$\psi_r(x) = f(x) + \sum_{j=1}^{m_e} r_j \mid g_j(x) \mid + \sum_{j=m_e+1}^{m} r_j \mid \min(0,\ g_j(x)) \mid$$

or the more efficient L_2 augmented Lagrangian function proposed by Schittkowski [33, 34]

$$\begin{aligned} \psi_r(x,v) = f(x) \quad & - \quad \sum_{j=1}^{m_e}(v_j g_j(x) - 0.5 r_j g_j^2(x)) \\ & - \quad \sum_{j=m_e+1}^{m'} \begin{cases} (v_j g_j(x) - 0.5 r_j g_j^2(x)) & \text{if } g_j(x) \leq v_j/r_j \\ 0.5 v_j^2/r_j & \text{otherwise,} \end{cases} \end{aligned} \tag{3.3}$$

where $m' = m + 2n$ is used to include the design variable bounds as constraints. Schittkowski asserts that the augmented Lagrangian function is superior to the L_1-penalty function because the latter requires known upper bounds on the Lagrange multipliers to guarantee global convergence. Local superlinear convergence can also be affected [33:84]. Schittkowski shows that Equation (3.3) can improve convergence and prevent cycling if penalty parameters r_k are chosen properly [34:201].

The line search begins with $\alpha = 1$, and α is reduced on subsequent iterations until a stopping condition

$$\psi_k(\alpha) \leq \psi_k(0) + \mu\alpha\psi_k'(0)$$

is satisfied, where $\psi_k'(0) < 0$ must hold [32:11]. To guarantee this convergence, the penalty parameters (for this algorithm, each constraint has a different parameter) are defined [34:201] by

$$\begin{aligned} r_j^{(k+1)} &= \max\left(\sigma_j^{(k)} r_j^{(k)},\ \frac{2m(u_j^{(k)} - v_j^{(k)})^2}{(1-\delta_k)s_k^T \mathbf{W} s_k}\right), \quad j = 1, \ldots, m, \\ \text{where} \quad \sigma_j^{(k)} &= \min\left(1,\ \frac{k}{\sqrt{r_j^{(k)}}}\right). \end{aligned}$$

3.6 Integration of NLPQL and ASTROS

3.6.1 Main Loop Implementation. ASTROS and NLPQL were integrated in the manner described by Figure 3.2. A special driver program was written to take advantage of NLPQL's "reverse communication" option, in which the algorithm exits to the driver each time function or gradient evaluations are required. Since ASTROS cannot be used as a callable subroutine, the driver program simply exits to the operating system at each iteration. Because of this, special measures were taken to store the NLPQL data between iterations.

The driver program has several functions. It uses the ASTROS memory manager to dynamically allocate array space, and it accesses the current function and gradient information from the database created by ASTROS at each iteration. It then calls NLPQL with this data and gets back a new design point to test. The new point may or may not be the next current design, since NLPQL's line search procedure may require intermediate function calls. Using this data, the driver then rewrites the ASTROS input file in the proper format. A Unix shell was written to control the ASTROS-NLPQL loop. The source code is provided in Appendix C.

ASTROS is a very complex collection of FORTRAN subroutines controlled by a long sequence of commands written in the MAPOL computer language. In order to prevent ASTROS from using its own optimizer during each iteration, changes to the standard MAPOL sequence were necessary. This was accomplished by inserting the changes into the ASTROS input file as described in the ASTROS User's Manual [20]. During each iteration, a call to "EXIT", an ASTROS subroutine, was inserted into the MAPOL sequence prior to the call to the optimizer. To save computational time, the point at which the exit call was made was determined by NLPQL. If NLPQL needed function evaluations only, the call was made prior to ASTROS sensitivity analysis; otherwise, it was made immediately prior to the optimizer call.

3.6.2 Integrating Active Constraint Flags. Since ASTROS normally optimizes approximate problems, its method of tracking constraints is quite different from NLPQL. In the traditional optimization loop, the active set strategy must be very generous in holding constraints active. Since the error in linearly approximating the constraints can be unpredictable, a restrictive strategy can yield highly infeasible iterates. The only way to prevent this is to hold active any constraints whose continued feasibility would be in doubt. Since, in practice, there is no scientific means of guessing which constraints to hold active, a generous strategy is typically used.

The constraint deletion procedure in ASTROS forms its working set according to the values of two parameters, EPS and NRFAC. EPS is similar to the ϵ used in NLPQL's active set strategy, except that, in ASTROS, the constraints are normalized and of opposite sign:

$$G_j(x)/\bar{G}_j - 1 \leq 0.$$

NRFAC is a minimum number of active constraints expressed as a factor of the number of design variables. For example, if there are 30 constraints and 10 design variables, and

NRFAC = 0.5, then ASTROS would keep, at a minimum, 5 constraints in the active set at all times.

For the three structures problems tested, NRFAC was set to zero in the alternative loop in an effort to mimic the NLPQL strategy, and EPS was set at 10^{-2} (better accuracy is generally not needed for structures problems). In the ASTROS approach, each problem used "realistic" values for NRFAC and EPS (generally NRFAC = 1.0, EPS = 10^{-1}).

3.7 Convergence Criteria

ASTROS and NLPQL have different convergence criteria. The ASTROS criteria, which will be denoted "Criteria (C1)" are given by

$$\|f_k - f_{k-1}\| < .005 \mid f_0 \mid \quad \text{and} \quad MCV < 0.01,$$

where $f_k, k = 0, 1, \ldots$ denotes the value of the objective function at the kth iteration and MCV denotes the maximum constraint violation. Its bound was chosen "realistically", but tighter bounds are sometimes needed in practice, particularly with respect to flutter and frequency constraints. The criteria for NLPQL, denoted by "Criteria (C2)", are given by

$$KTO = \mid s^T \nabla f \mid + \sum_{i=1}^{m} \mid u_j g_j(x) \mid \leq \epsilon \quad \text{and} \quad SCV \leq \sqrt{\epsilon},$$

where KTO is a measure of the Kuhn-Tucker optimality conditions, SCV denotes the sum of constraint violations, and ϵ is the error tolerance, chosen to be $\epsilon = 10^{-2}$. Although this degree of accuracy is rarely needed in practice, it was used to demonstrate the accuracy of NLPQL.

So as to compare ASTROS and ASTROS-NLPQL properly but still demonstrate accuracy, it was necessary to run ASTROS-NLPQL to completion using Criteria (C2), but then manually compare them afterward; i.e., ASTROS-NLPQL was compared against ASTROS up to the point at which each satisfied Criteria (C1).

3.8 Summary

The test problems were run using both the traditional ASTROS loop and the alternative ASTROS-NLPQL implementation. Results of this study are reported in the next chapter.

IV. Results

4.1 Overview

This chapter presents the results of this study, including solutions to the test problems and algorithm performance comparisons. Integrity of the driver program and algorithm implementation was first verified by solving the classic ten-bar truss problem (drawing shown in Figure 1.2) and comparing the results with known solutions. The test problems were solved (or attempted) using both the standard (ASTROS) and alternative (ASTROS-NLPQL) approaches in the manner described in the previous chapter.

Each problem is discussed individually, and comparative plots of weight versus CPU time and number of iterations are provided. Since gradient evaluations are generally much more computationally expensive than function evaluations, an iteration for the ASTROS-NLPQL approach is defined to be a gradient evaluation; that is, each iteration of the ASTROS-NLPQL loop consists of up to five (set by the driver program, but rarely exceeds three) function evaluations and one gradient evaluation. A plot of maximum constraint violation (MCV) versus number of iterations is also included for each problem. A feasible design is defined as one whose MCV is less than 10^{-2}. Additional supporting data for each problem is provided in Appendix B.

For each of the tables in this chapter, the following notation is used:

Tr200	=	200-member plane truss
ICW	=	Intermediate complexity wing
HALE	=	High-altitude, long-endurance aircraft
F	=	Objective function value (weight in lbs)
MCV	=	Maximum constraint violation
CPU	=	CPU time required (sec)
NITER	=	Number of iterations required
NFUNC	=	Number of function evaluations required
NGRAD	=	Number of gradient evaluations required
NCG	=	Number of individual gradients computed

In addition to these conventions, ASTROS-NLPQL (C1) refers to results when Criteria (C1) is satisfied for the first time, while ASTROS-NLPQL (C2) refers to final results within the stricter Criteria (C2).

4.2 200 Member Plane Truss

The results of the Tr200 problem are summarized in Table 4.1. Iteration histories comparing the two approaches are given in Figures 4.1–4.3. As shown in Table 4.1,

Table 4.1 Tr200 Optimization Results

	ASTROS	ASTROS-NLPQL (C1)	ASTROS-NLPQL (C2)
F	30000.7	29951.1	28772.3
MCV	0.0	0.004404	0.000001
CPU	879.7	1785.9	13738.9
NFUNC	13	31	196
NGRAD	13	18	150
NCG	2711	670	2076

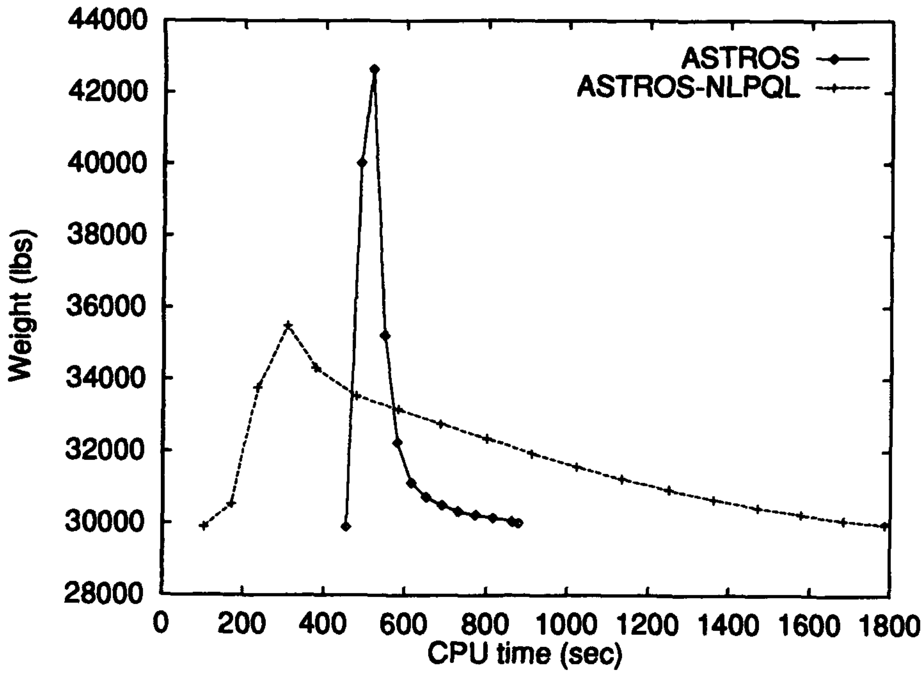

Figure 4.1 Tr200: Weight vs. CPU time

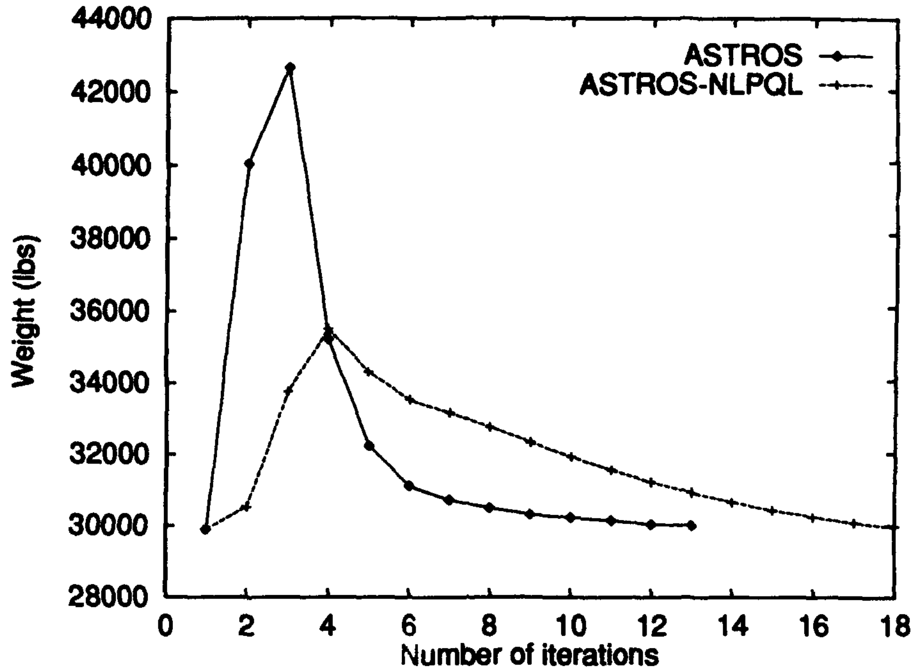

Figure 4.2 Tr200: Weight vs. Number of Iterations

ASTROS-NLPQL successfully converged to an optimal design of 28772 pounds. ASTROS required fewer iterations and function evaluations, while ASTROS-NLPQL computed fewer individual gradients. ASTROS finished in less time, but ASTROS-NLPQL yielded a lower feasible weight.

ASTROS-NLPQL was very accurate, but the improvement was computationally costly. This was expected, since most optimization algorithms exhibit this type of behavior. However, the improvement is not insignificant. Table 4.2 shows most of the improvement occurring early, but still a four percent improvement between Criteria (C1) and (C2).

Table 4.2 Tr200: ASTROS-NLPQL Iteration Analysis

Event	NGRAD	CPU	F	Improvement
First feasible design	6	478.7	33521	———
(C1) reached	18	1785.9	29951	3570 (11%)
(C2) reached	150	13738.9	28772	1179 (4%)

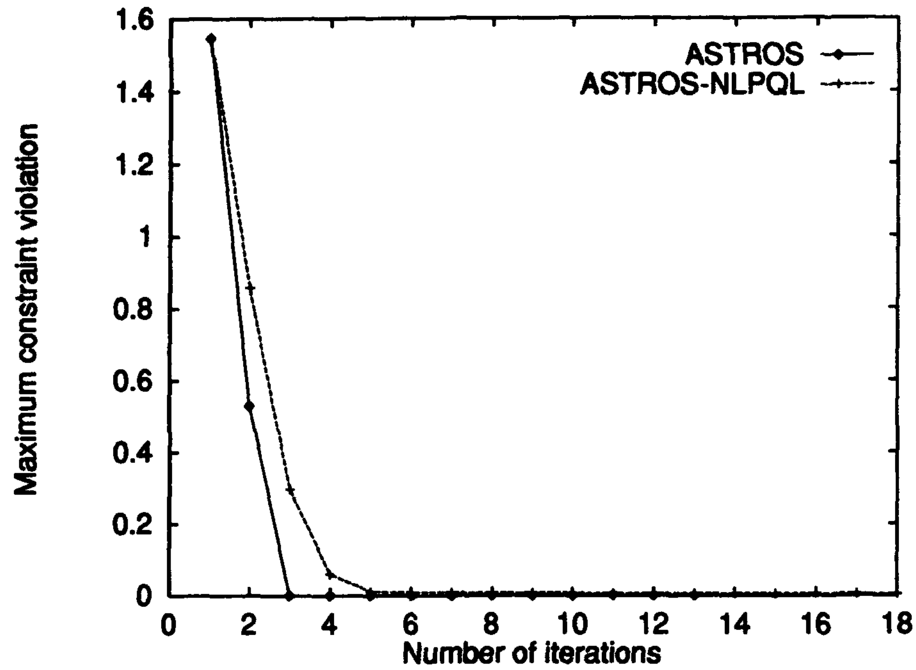

Figure 4.3 Tr200: MCV vs. Number of Iterations

4.3 Intermediate Complexity Wing

The ICW problem was actually solved twice, with and without scaling of the objective function. Results of both versions are summarized in Table 4.3, and iteration histories are given in Figures 4.4–4.6. As seen in Table 4.3, ASTROS-NLPQL successfully converged to optimal weights of 41.55 and 41.59 pounds for the scaled and unscaled versions, respectively. Once again, ASTROS required fewer iterations and function evaluations, while ASTROS-NLPQL computed fewer individual gradients. ASTROS finished in less time, but ASTROS-NLPQL yielded a lower feasible weight.

Unlike the Tr200 problem, the first feasible design of this structure occurred concurrently with the satisfaction of Criteria (C1). That is, the change in weight between iterations had already satisfied the convergence tolerance limit. At that point, the scaled version was much faster than the unscaled version due to the fewer gradient calculations required.

Table 4.3 ICW Optimization Results

		with scaling	
	ASTROS	ASTROS-NLPQL (C1)	ASTROS-NLPQL (Last)*
F	42.483	41.369	41.551
MCV	0.0002466	0.009025	0.004560
CPU	4434.1	9829.4	15935.7
NFUNC	8	59	95
NGRAD	8	31	48
NCG	2801	1040	1801
		without scaling	
F	42.483	41.591	41.594
MCV	0.0002466	0.004599	0.003267
CPU	4434.1	24251.6	27319.3
NFUNC	8	56	65
NGRAD	8	34	38
NCG	2801	2348	2505

*Criteria (C2) not reached. Last feasible design is given.

In both cases, ASTROS-NLPQL ended prematurely with a message that it had exceeded the maximum allowable number of line search iterations. With scaling, a feasible design satisfying Criteria (C1) had been reached. In order to solve the unscaled problem with ASTROS-NLPQL, NLPQL's line search type parameter was changed to force NLPQL to use the nondifferentiable L_1-penalty function given by Equation 3.5.2. This strategy was recommended by Schittkowski in the opening comments of his source code [36]. Theoretically, this slows convergence, but it allowed the program to continue (Appropriate logic was subsequently added to the driver program). ASTROS-NLPQL then proceeded to a feasible design (satisfying Criteria (C1)), but once again ended with the same error message. However, at that point, there was no substantive change in weight (at least less than 5×10^{-4} pounds) between the last two iterations.

The early termination of NLPQL is believed to have occurred for one of two reasons. The first possibility is that, at that point, the penalty parameters had values that made convergence to the line search stopping condition too slow to be accomplished within the allowable number of function calls. The second possibility is a precision problem between ASTROS and NLPQL. ASTROS stores its design variable data in single precision form, while NLPQL stores in double precision. When NLPQL rewrites the ASTROS input

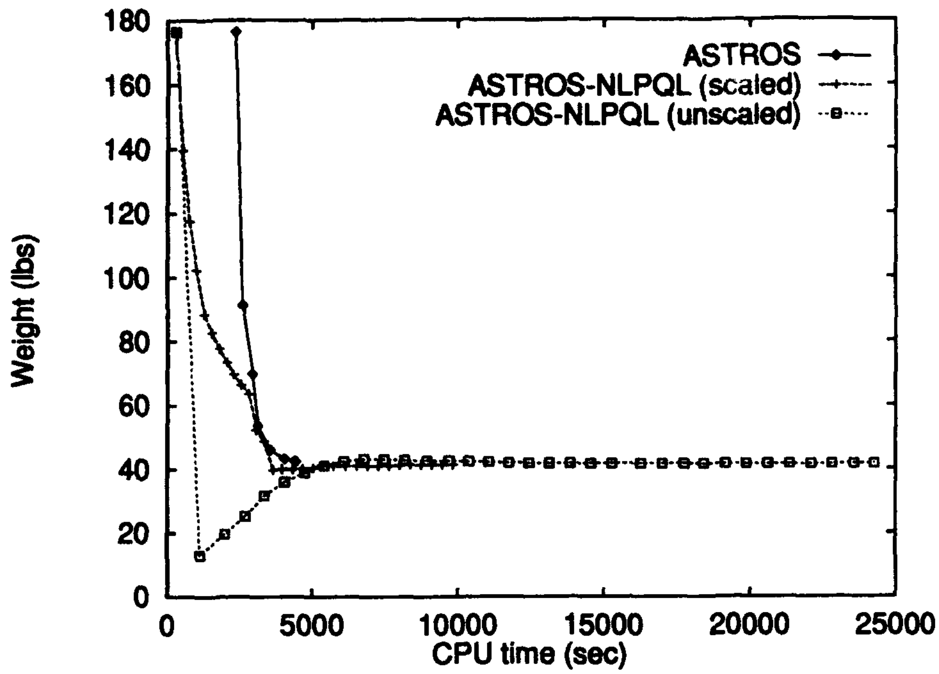

Figure 4.4 ICW: Weight vs. CPU time

file with a new set of design variables, not only does it write in single precision, but it must also write each record in a set of eight-character fields to remain compatible with ASTROS. Currently, this is not more accurate than 10^{-5}. Certainly, a combination of these two reasons is possible, if not likely. That is, by the time the stopping criterion would have been satisfied, there was already insufficient precision in the process. Full integration into ASTROS may remedy this problem. This was not considered originally because such accuracy is almost never needed in structures problems (it is unrealistic from the manufacturer's viewpoint).

4.4 High-Altitude, Long-Endurance Aircraft

ASTROS could not solve the HALE problem within its available memory. However, HALE was successfully solved using the ASTROS-NLPQL approach. Results are given in Table 4.4. Iteration history is given in Figures 4.7, 4.8, and 4.9. Although no comparison data for ASTROS is given, the final feasible weight of 1601.4 pounds recorded by ASTROS-NLPQL is 49.2 pounds lower than the previous lowest feasible weight of 1650.6 pounds

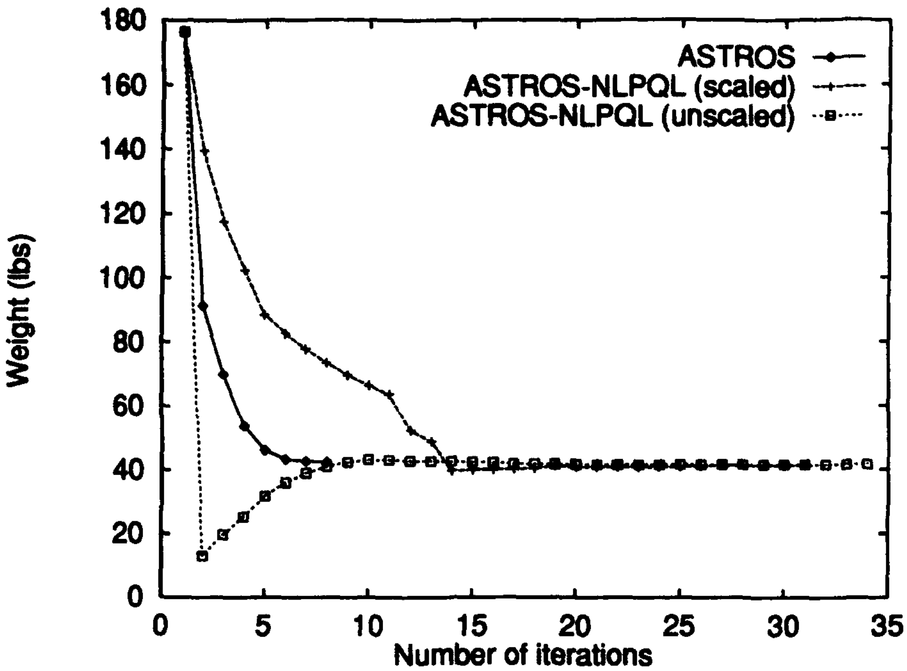

Figure 4.5 ICW: Weight vs. Number of Iterations

recorded by Canfield and Venkayya using an OC approach ([5], [6]). An even lower weight could have been obtained, had computer resources been available to continue running the program. During the run of the HALE problem, measurements were periodically taken with respect to CPU time required for one function evaluation and one gradient evaluation. Each iteration averaged between 7000–11000 seconds of CPU time. Of that time one function evaluation required approximately 25–30 seconds. Because there were

Table 4.4 HALE Optimization Results

ASTROS-NLPQL:	(C1)	(lf)*	(Last)**
F	1731.2	1633.0	1518.9 (1601.4)
MCV	0.002037	0.008631	0.054296 (0.0)
CPU (hrs)	60.6	143.8	198.3
NFUNC	66	113	148
NGRAD	42	72	91
NCG	2085	4487	5942

*Criteria (C2) not reached. Last feasible (lf) design given.

**Last design: infeasible (Equivalent feasible design).

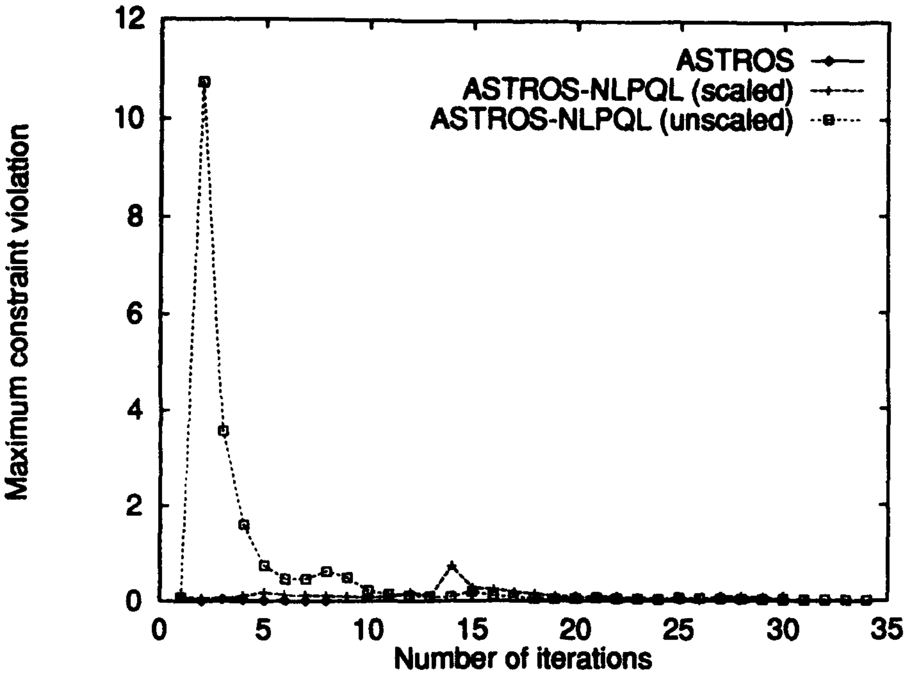

Figure 4.6 ICW: MCV vs. Number of Iterations

no flutter constraints, this was actually less expensive than the 35–45 seconds of CPU time per function evaluation required by the ICW problem.

4.5 Summary

ASTROS-NLPQL was successful in solving the three structures problems. Performance of the two approaches has been compared and some analytic observations made. The next chapter gives conclusions and recommendations for further research.

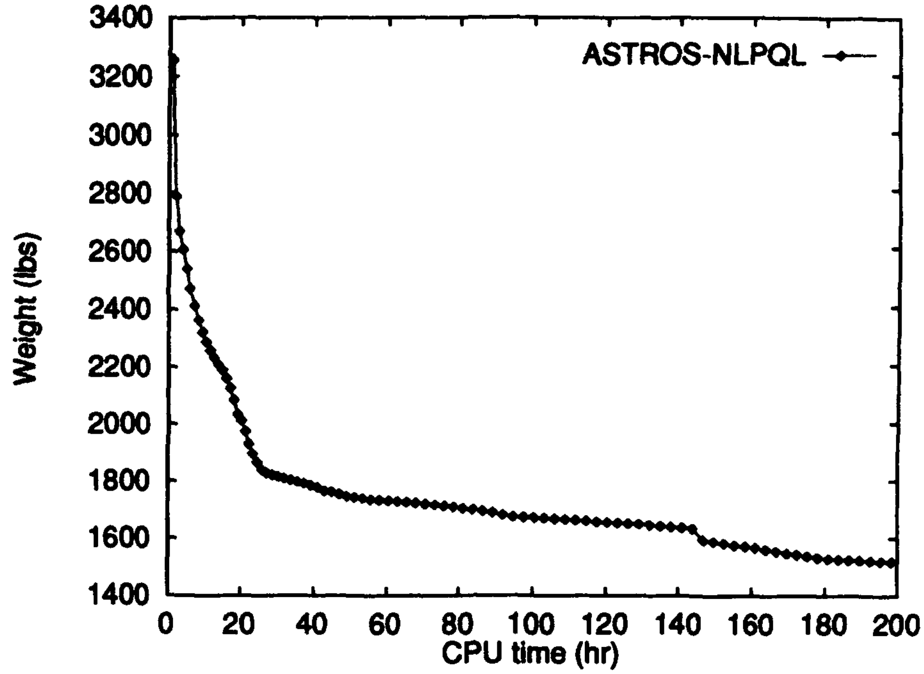

Figure 4.7 HALE: Weight vs. CPU time

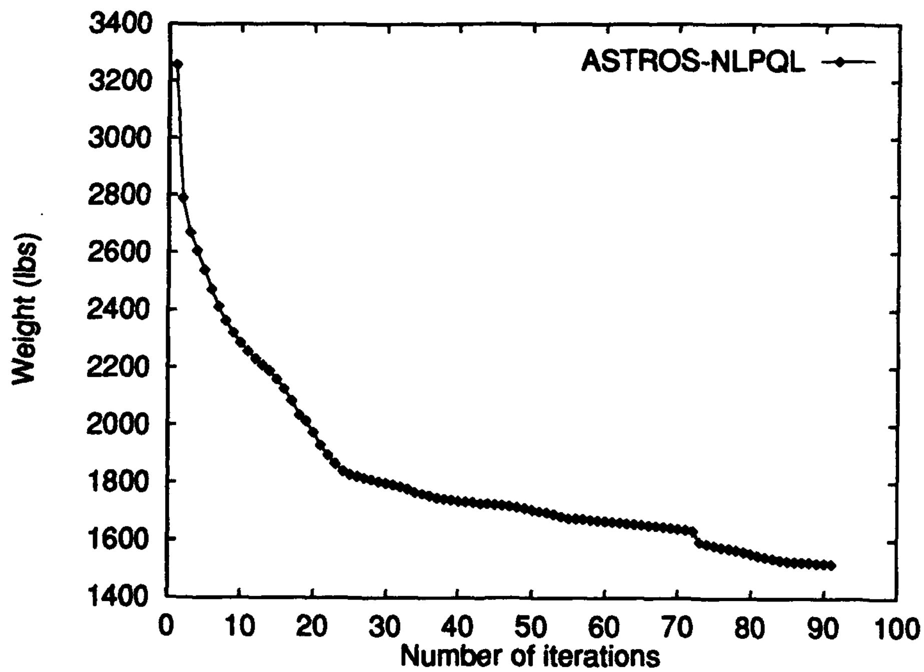

Figure 4.8 HALE: Weight vs. Number of Iterations

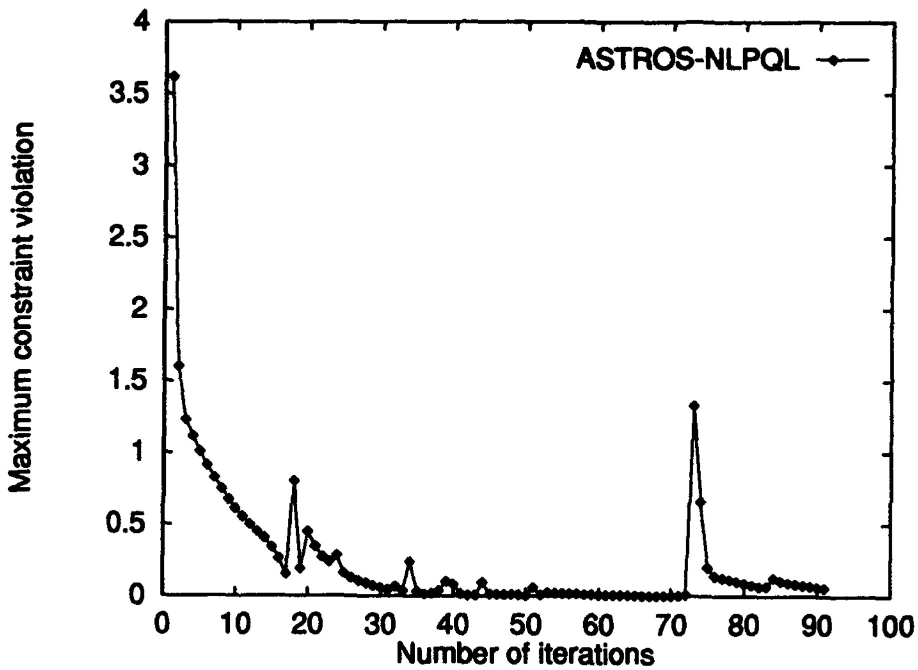

Figure 4.9 HALE: MCV vs. Number of Iterations

V. Conclusions and Recommendations

5.1 Summary

The purpose of this thesis was to adapt an SQP method with active set strategy to efficiently solve large-scale structural optimization problems within a multidisciplinary environment. The literature suggests that SQP methods generally converge faster than other MP methods, and an efficient active set strategy can reduce the problem size, speeding up convergence even more.

Schittkowski's NLPQL algorithm was chosen as the optimizer primarily because it has special logic which could be adapted for structures problems, which often do not have explicitly defined constraint functions. This feature made NLPQL less cumbersome to implement than other available codes. It has also been shown to perform very well on smaller problems relative to other MP methods.

NLPQL was integrated into ASTROS with a different loop structure from what is currently used. In the normal ASTROS approach, ASTROS computes constraint values and gradients (for the retained constraints) and optimizes an approximate problem based on first-order approximations with respect to the reciprocal design variables. Since the approximate constraints are explicitly defined, it is normally computationally inexpensive to optimize. Convergence is achieved when the optimal solution to the approximate problem is feasible and the successive improvement in weight becomes small. The ASTROS-NLPQL implementation eliminated the approximate problem and used a much tighter tolerance for holding constraints in the active set. The hypothesis was that the computational cost of more iterations caused by the elimination of the approximate problem could be offset by the reduction in the number of gradients computed. In the current ASTROS approach many constraints are held active in an attempt to avoid highly infeasible designs caused by optimizing an underconstrained approximate problem. This implementation represents the first time an SQP method has been employed as the optimizer within the ASTROS environment.

ASTROS-NLPQL was tested on three large-scale structures problems, one with constraints from multiple disciplines. ASTROS-NLPQL successfully solved each within Criteria (C1). The Tr200 problem was solved within Criteria (C2). The ICW problem would probably be solvable to within (C2) if fully integrated into ASTROS. The HALE problem was stopped early, having exceeded available computer resources, but otherwise would probably have converged to within Criteria (C2). The results showed, as expected, that ASTROS-NLPQL generally required more iterations, while ASTROS computed more individual gradients. ASTROS required less CPU time.

The largest structure tested was the HALE problem. It was solved successfully for the first time by a direct method within the ASTROS environment. The resulting design is the lowest feasible design weight of this structure ever computed by ASTROS (specifically the structure described in Appendix A).

5.2 Conclusions and Observations

This research has demonstrated that NLPQL can be used to solve large-scale structural optimization problems. It has further shown that a direct method can be used to solve larger problems than those previously solved by ASTROS. In addition, some conclusions and observations are offered.

5.2.1 Convergence and Efficiency. The traditional ASTROS approach for solving large problems proved much more efficient than ASTROS-NLPQL. The more restrictive constraint retention tolerance could not offset the cost of the additional iterations. Based on the results, if the memory size within ASTROS could be made large enough to run the HALE problem, it is likely that ASTROS could solve HALE faster than ASTROS-NLPQL.

Although SQP methods are generally among the fastest MP algorithms, the benefit of local superlinear convergence of the NLPQL algorithm was found to be overrated for these large structures problems. In reality, either the neighborhood in which this occurs is very small, or the sequence of constants, γ_k, approaching zero which defines superlinear convergence does so very slowly (see Equation 2.9). Since large problems often have very flat regions near minima, convergence can be slow. This appears to be what happened in the Tr200 problem.

Although the HALE problem was successfully solved, ASTROS-NLPQL took an unreasonable amount of time to arrive at a solution. This was caused primarily by limited computer resources. For much of the time, it required between fifty and eighty percent of the Convex system memory. In solving very large problems, SQP was discovered to have a potential drawback: the benefit of superlinear convergence could be negated by the cost of storing and working with second-order information. As the number of design variables or constraints increases, the approximate Hessian matrix grows an order of magnitude faster. For small or moderately-sized problems, this is not as noticeable.

5.2.2 Feasible Designs. Another important observation was that it was easier for ASTROS to maintain a feasible design. Since ASTROS-NLPQL minimizes a merit function in determining a step size, the step size is chosen based on improvement in the merit function. This is an indirect way of measuring feasibility; in other words, NLPQL mathematically performs a "tradeoff" during each iteration between feasibility and optimality based on computed values of the merit function. Because of this, there is no way to directly control or affect whether an iterate is feasible or not. The results of the HALE problem demonstrated the tradeoff. In fact, the only way to affect convergence at all is by scaling the objective function before running it (see [35:492] for details). This can dramatically change convergence behavior, but, as observed in the ICW problem, this does not necessarily mean that feasibility will be attained faster.

In contrast, ASTROS can affect feasibility by controlling the number of constraints retained for the approximate problem. This was, in fact, observed during the study of the Tr200 problem. If a great number of constraints are held active and passed to the approximate problem, the approximate problem becomes overconstrained. Optimization then yields a design at the next iteration which has a higher probability of being feasible, but has less improvement. On the other hand, if few constraints are held active, the optimizer solves an underconstrained approximate problem. The resulting design may improve the weight greatly, but can frequently be highly infeasible.

5.2.3 Precision. ASTROS-NLPQL has been shown to be capable of accurately solving large structures problems. Criteria (C2) was deliberately set unrealistically tight

($\epsilon = 10^{-2}$). Although this criteria was achieved by only one of three problems, there is no evidence to show that it would not have occurred, given a more precise implementation and enough CPU time available.

Such precision is rarely needed in practice. Usually, the optimization process is allowed to continue only as long as the percent improvement is judged to be worth the computational cost of additional iterations. This may seem to make the benefit of accuracy less significant; however, accuracy makes other types of research possible.

For example, most large structures are not optimized using all independent variables; otherwise the problems would simply be too large to solve. Instead, variables are linked together to shrink the problem size. For, say, an aircraft wing, one common linking scheme is to design the top and bottom surfaces symmetrically; i.e., each design variable representing a component size on the top surface is forced to have the same value as the corresponding variable on the bottom surface. This has the effect of imposing additional constraints on the problem, because it limits the freedom to design all variables independently. With the ability to solve larger problems to a greater accuracy, researchers and designers can determine the cost of linking variables, simply by solving problems with and without linking and measuring the difference. Judgements can then be made as to whether it is worth the additional cost to link variables.

5.3 Recommendations for Future Research

In addition to the proposed study of design variable linking costs just described, there are several other avenues for further investigation and research. These are briefly discussed.

Improvements within NLPQL. The NLPQL software was structured so that its subroutines would be easily replaceable with other available codes. In particular, NLPQL allows the use of a different line search procedure or quadratic programming solver.

The results of the HALE problem showed that the cost of one gradient evaluation can be enormous compared to that of a function evaluation. This is often the case in problems with stress and displacement constraints only (i.e., no aerodynamic constraints). Flutter and frequency constraints require more computational effort, but gradient evaluations still dominate the total computational cost. With this in mind, a more efficient line search technique within NLPQL could potentially yield a significant savings. By searching along the line at a few more points, a more improved weight can be achieved at each iteration. The tradeoff would be a few more function evaluations per iteration, but fewer iterations or gradient evaluations.

A different quadratic programming solver may also significantly improve computational performance for large problems. One of the possible improvements may be focused on dealing with the large matrices. The current QP solver requires the Cholesky decomposition of $\nabla_2 L$.

Extensions to ASTROS-NLPQL Integration. Direct integration of NLPQL into ASTROS would provide additional insight into the performance of the NLPQL optimizer. It involves creating a new driver for NLPQL and modifying the MAPOL sequence (main program) in ASTROS to call the NLPQL subroutine. This can be done within either the traditional or alternative loop structure. Traditional loop integration would provide insight into how NLPQL performs as an optimizer of the approximate problem. For that matter, other optimizers, such as PBLA, could be used as the optimizer and compared against NLPQL and ADS. The savings could be become significant for large problems with many binding constraints at the optimum. This would make the approximate problem larger and more costly to solve. Alternative loop integration would provide a much more precise estimate of CPU time required.

Sequential Linear Programming. Although SQP methods have faster convergence, sequential linear programming (SLP) in the alternative loop structure could potentially improve efficiency. SLP methods solve linear programs to compute search directions rather than quadratic subproblems. SLP algorithms do not have to store second-order information, which becomes expensive for large problems. Also, linear subproblems are typically cheaper to solve than quadratic problems. Schittkowski also adds that SLP algorithms do not inherit roundoff error in the approximate Hessian matrix often seen in large problems. He asserts that this is usually brought on by inexact numerical derivatives [32:40]. ASTROS computes analytical derivatives, but if finite difference estimates, which are based on previous function evaluations, are available at much less cost, and SLP methods do not have the roundoff error that SQP methods would, a computational savings could be realized. One drawback, however, is that the slower convergence of SLP means more iterations.

ASTROS Constraint Retention. Perhaps the greatest inefficiency in ASTROS is the method by which constraints are retained as active. As discussed earlier, ASTROS employs a generous strategy so that each successive design produced by the optimizer is as close to feasible as possible. The current defaults for the strategy parameters are NRFAC = 3.0 and EPS = 0.1. Such conservatism is not needed. The effect of changing the parameters should be studied in greater detail. A more scientific approach, particularly with some solid theoretical development, could lead to a heuristic for choosing the "best" strategy parameters.

Hybrid Methods. One of the newest areas of study is hybrid methods, in which more than one algorithm is used so that each takes advantage of its strength. One proposed approach would use OC methods to get near an optimum quickly, and then an SQP method to tighten the accuracy as quickly as possible.

Parallelization. Finally, the development of parallel computer architectures has led to a vast amount of research in algorithm development. Such research is focused on exploiting the features of the parallel architecture to increase the speed and efficiency of algorithms. Parallelization of optimization algorithms such as NLPQL can increase efficiency.

Appendix A. *Description of Test Problems*

This chapter gives a brief description of each test problem solved. A finite element "wire model" and table of design conditions is provided for each.

A.1 200 Member Plane Truss

The first structural problem solved was a 72-node plane truss consisting of 200 steel elements subject to five loading conditions. A diagram of this structure is given in Figure A.1, and a more detailed description is given in Table A.1. This problem has 200 design variables and 2500 stress and displacement limit constraints [6:1040].

A.2 Intermediate Complexity Wing

Figure A.2 shows an intermediate complexity wing with 158 elements and 234 degrees of freedom. In this problem, composite cover skins are made of graphic epoxy (properties given in Table A.2). Constraints include stress limits on all membrane elements and wing tip transverse displacements limits for two independent loading conditions. Also imposed was a flutter speed limit of 925 knots (corresponds to 0.8 Mach at sea level). The resulting problem has 350 design variables and 722 constraints [6:1040–1041].

A.3 High-Altitude Long Endurance Aircraft

Figure A.3 shows a finite element model of the right wing of a high-altitude long endurance (HALE) aircraft consisting of a truss substructure and metallic cover skins. This 270-ft span aircraft is designed to patrol for several days at 65,000 feet at a speed of 150-200 knots. Three static loads were applied to an aluminum version of this aircraft, and stress and wing-tip deflection limits were imposed. The resulting problem has 1527 design variables and 6124 constraints [6:1041].

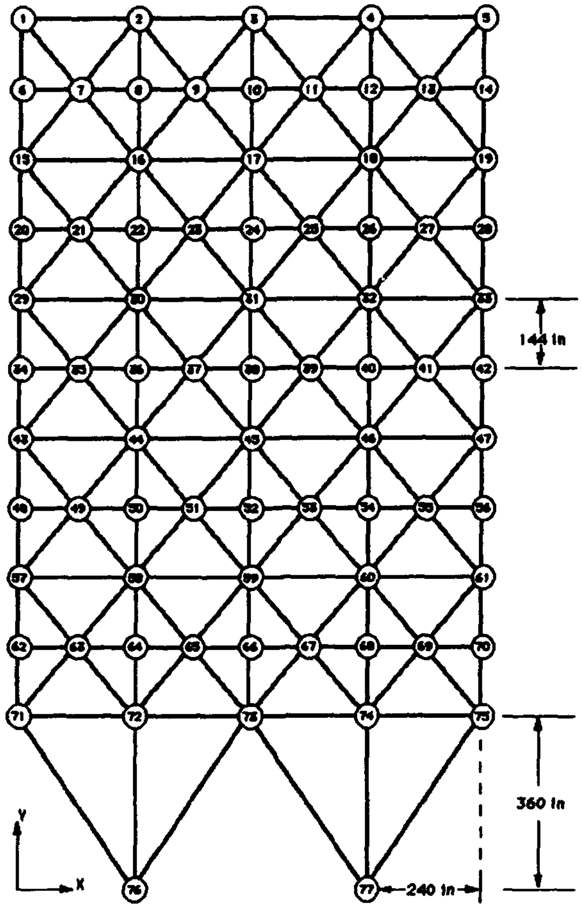

Figure A.1 200 Member Plane Truss Model

No. of Nodes	No. of Elements		No. of DoF's	
88	39	Rods	485	Constrained
	55	Shear Panels	217	Unconstrained
	62	Quadrilateral Membranes	702	Total
	2	Triangular Membranes		
	158	Total		

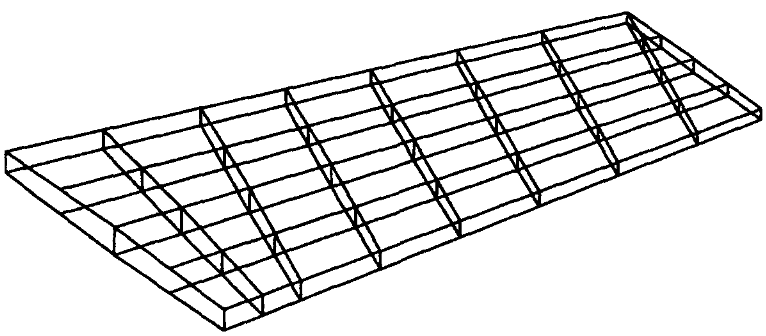

Figure A.2 Intermediate Complexity Wing Model

Table A.1 200 Member Plane Truss Design Conditions

Material, steel	
Modulus of elasticity	$E = 30 \times 10^6$ psi
Weight density	0.283 lb/in.^{3}
Stress limits	30,000 psi
Lower limit on rod areas	0.1 in.^{2}
Displacements on all nodes (horizontal, vertical directions)	0.5 in.
Number of loading conditions	5
Loading condition 1	1000 lb acting in $+X$ direction at nodes 1, 6, 15, 20, 29, 34, 43, 48, 57, 62, 71
Loading condition 2	1000 lb acting in $-X$ direction at nodes 5, 14, 19, 20, 28, 33, 42, 47, 56, 61, 70, 75
Loading condition 3	10,000 lb acting in $-Y$ direction at nodes 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20, 22, 24, ..., 71, 72, 73, 74, 75
Loading condition 4	Loading conditions 1 and 2 together
Loading condition 5	Loading conditions 2 and 3 together

Table A.2 Intermediate Complexity Wing Design Conditions

Isotropic material, aluminum	
Modulus of elasticity	$E = 30 \times 10^6$ psi
Poisson's ratio	0.30
Weight density	0.1 lb/in.^{3}
Tensile stress limit	67,000 psi
Comprehensive stress limit	57,000 psi
Shear stress limit	39,000 psi
Lower limit on thickness (shear panels)	0.02 in.
Lower limit on rod areas	0.02 in.^{2}
Orthotropic material, graphite epoxy	
Modulus of elasticity	$E_1 = 30 \times 10^6$ psi
	$E_2 = 1.6 \times 10^6$ psi
	$G_{12} = 0.65 \times 10^6$ psi
Poisson's ratio	0.25
Weight density	0.055 lb/in.^{3}
Stress limits	115,000 psi
Lower limit on plies	0.00525 in.
Behavior constraints	
Limit on transverse tip displacements	10.0 in.
Flutter speed limit	925 knots

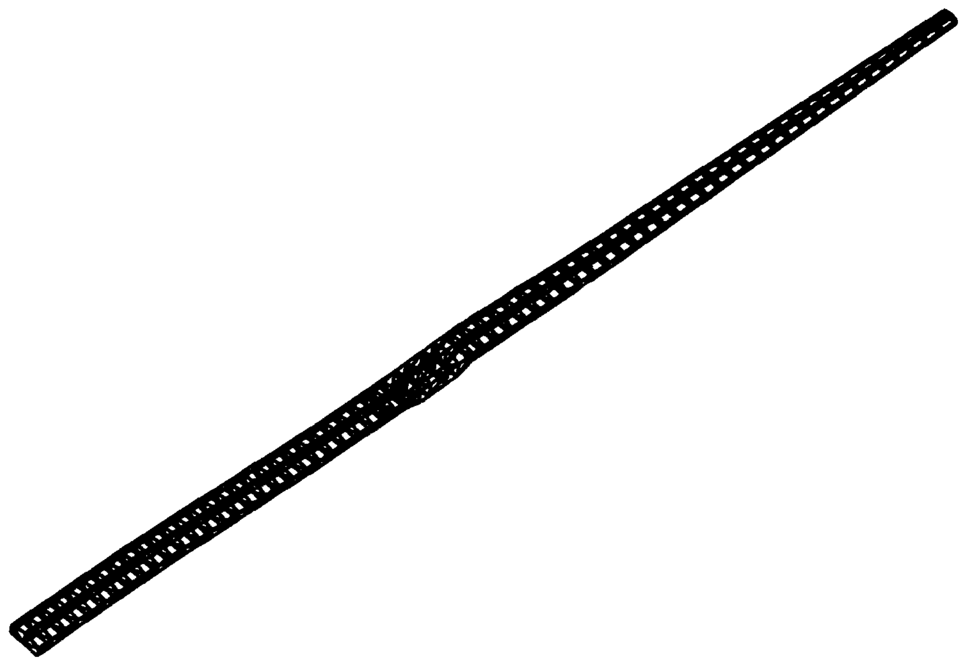

Figure A.3 High-Altitude Long Endurance Aircraft

Table A.3 High-Altitude Long Endurance Aircraft Wing Design Conditions

Material, aluminum	
Modulus of elasticity	$E = 10.5 \times 10^6$ psi
Poisson's ratio	0.30
Weight density	0.1 lb/in.^{3}
Stress limits	60,000 psi
Lower limit on thickness (shear panels)	0.021 in.
Lower limit on rod areas	0.10 in.^{2}
Behavior constraints	
Limit on transverse tip displacements	200.0 in.
Number of loading conditions	4

Appendix B. *Test Problem Iteration Summaries*

In support of the results presented in Chapter IV, the underlying data for each problem is given here in tabular form. Column headings common to all the tables are defined as follows.

Tr200	=	200 bar truss structure
ICW	=	Intermediate complexity wing
HALE	=	High-altitude long endurance aircraft
I	=	Iteration number
F	=	Objective function value
MCV	=	Maximum constraint violation
CPU	=	Cumulative CPU time elapsed (sec)
NFUNC	=	Cumulative number of function evaluations
NGRAD	=	Cumulative number of gradient evaluations
NACT	=	Number of active (or retained) constraints

Table B.1: Tr200: ASTROS Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	29890.2	1.5454	454.2	1	1	311
2	40018.6	0.5290	487.9	2	2	200
3	42639.3	0.0	516.0	3	3	200
4	35213.7	0.0	547.8	4	4	200
5	32239.1	0.0	580.4	5	5	200
6	31119.2	0.0	615.0	6	6	200
7	30719.2	0.0	651.3	7	7	200
8	30496.2	0.0	689.7	8	8	200
9	30318.7	0.0	730.0	9	9	200
10	30231.1	0.0	772.0	10	10	200
11	30150.4	0.0	815.8	11	11	200
12	30043.7	0.0	863.4	12	12	200
13	30000.7	0.0	879.7	13	13	200

Table B.2: Tr200: ASTROS-NLPQL Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	29890.192	1.545400	102.473	1	1	289
2	30525.377	0.857279	168.954	2	2	166
3	33752.177	0.296509	233.911	3	3	93
4	35488.013	0.059716	307.959	4	4	24
5	34303.776	0.010453	377.166	5	5	7
6	33521.057	0.008921	478.030	7	6	7
7	33159.736	0.008279	581.213	9	7	7

Table B.2: (continued)

I	F	MCV	CPU	NFUNC	NGRAD	NACT
8	32760.821	0.007644	686.238	11	8	7
9	32343.548	0.007119	800.252	13	9	7
10	31936.528	0.006684	912.195	15	10	7
11	31563.039	0.006308	1022.490	17	11	7
12	31224.840	0.005986	1133.907	19	12	7
13	30923.061	0.005704	1250.573	21	13	7
14	30657.191	0.005449	1361.607	23	14	7
15	30428.247	0.005206	1470.286	25	15	7
16	30237.079	0.004953	1577.472	27	16	7
17	30079.620	0.004685	1682.959	29	17	7
18	29951.053	0.004404	1785.907	31	18	7
19	29846.510	0.004113	1888.131	33	19	7
20	29761.280	0.003818	1990.627	35	20	7
21	29690.816	0.003525	2092.868	37	21	7
22	29631.442	0.003242	2195.229	39	22	7
23	29580.318	0.002973	2297.602	41	23	7
24	29535.368	0.002720	2400.525	43	24	7
25	29495.231	0.002484	2505.591	45	25	7
26	29458.864	0.002265	2610.131	47	26	7
27	29426.320	0.002062	2716.454	49	27	9
28	29398.057	0.001903	2821.656	51	28	9
29	29375.501	0.001745	2927.300	53	29	9
30	29338.735	0.001524	3032.521	55	30	9
31	29284.345	0.001237	3137.684	57	31	9
32	29146.282	0.018594	3217.081	58	32	11
33	29142.519	0.017483	3295.421	59	33	13
34	29106.163	0.016581	3375.475	60	34	13
35	29093.176	0.014795	3482.718	62	35	11
36	29069.331	0.007582	3562.921	63	36	9
37	29048.457	0.010525	3643.943	64	37	9
38	29026.147	0.009212	3724.073	65	38	9
39	29021.481	0.007769	3830.775	67	39	9
40	28995.211	0.008209	3913.592	68	40	11
41	28988.451	0.006608	4021.815	70	41	11
42	28993.303	0.001065	4102.937	71	42	11
43	28953.122	0.007346	4184.648	72	43	11
44	28945.258	0.004248	4266.306	73	44	11
45	28928.561	0.005640	4347.392	74	45	11
46	28920.623	0.002250	4428.083	75	46	11
47	28907.235	0.002708	4508.825	76	47	11
48	28907.971	0.000950	4589.292	77	48	11
49	28907.141	0.000828	4671.132	78	49	11
50	28900.665	0.001618	4752.638	79	50	11
51	28900.119	0.000470	4834.610	80	51	11
52	28898.253	0.000320	4916.639	81	52	11
53	28896.492	0.000459	4998.455	82	53	11
54	28895.703	0.000238	5080.888	83	54	11
55	28895.329	0.000358	5162.561	84	55	11
56	28895.521	0.000049	5244.081	85	56	11
57	28895.015	0.000075	5326.125	86	57	11
58	28894.817	0.000015	5408.354	87	58	11
59	28894.363	0.000033	5490.592	88	59	11
60	28894.104	0.000045	5573.042	89	60	11

Table B.2: (continued)

I	F	MCV	CPU	NFUNC	NGRAD	NACT
61	28893.751	0.000036	5654.948	90	61	11
62	28893.303	0.000030	5737.385	91	62	11
63	28892.801	0.000037	5820.282	92	63	11
64	28892.225	0.000053	5902.438	93	64	11
65	28891.357	0.000066	5985.119	94	65	11
66	28890.085	0.000086	6067.467	95	66	11
67	28888.748	0.000114	6149.648	96	67	11
68	28887.699	0.000146	6231.844	97	68	11
69	28886.304	0.000214	6313.221	98	69	11
70	28883.815	0.000465	6395.043	99	70	11
71	28879.964	0.001040	6476.848	100	71	11
72	28873.883	0.003451	6560.159	101	72	11
73	28872.142	0.003218	6669.891	103	73	11
74	28869.355	0.003190	6780.511	105	74	11
75	28866.438	0.003150	6891.008	107	75	11
76	28863.532	0.003087	6999.996	109	76	11
77	28860.710	0.003001	7110.051	111	77	11
78	28857.999	0.002893	7219.847	113	78	11
79	28855.198	0.002765	7330.704	115	79	11
80	28852.472	0.002632	7443.620	117	80	11
81	28849.830	0.002496	7554.780	119	81	11
82	28849.580	0.002471	7671.021	122	82	11
83	28847.054	0.002333	7787.203	124	83	11
84	28844.548	0.002192	7909.731	126	84	11
85	28841.297	0.002035	8025.926	128	85	11
86	28837.556	0.001872	8138.138	130	86	11
87	28833.125	0.001715	8250.422	132	87	11
88	28827.901	0.001721	8364.785	134	88	11
89	28818.383	0.003378	8450.053	135	89	11
90	28817.883	0.000362	8535.869	136	90	11
91	28817.356	0.000189	8622.311	137	91	11
92	28816.517	0.000280	8708.749	138	92	11
93	28815.937	0.000232	8795.591	139	93	11
94	28815.878	0.000084	8882.240	140	94	11
95	28814.853	0.000159	8969.209	141	95	11
96	28814.278	0.000099	9055.440	142	96	11
97	28812.966	0.000217	9143.501	143	97	11
98	28812.303	0.000159	9232.085	144	98	11
99	28811.969	0.000063	9320.134	145	99	11
100	28811.311	0.000072	9407.561	146	100	11
101	28810.742	0.000048	9495.425	147	101	11
102	28809.872	0.000061	9582.294	148	102	11
103	28809.187	0.000067	9669.087	149	103	11
104	28808.651	0.000084	9755.971	150	104	11
105	28808.238	0.000073	9843.407	151	105	11
106	28807.822	0.000041	9929.012	152	106	11
107	28807.156	0.000076	10014.556	153	107	11
108	28806.932	0.000055	10100.100	154	108	11
109	28806.629	0.000035	10186.868	155	109	11
110	28806.109	0.000046	10273.688	156	110	11
111	28805.499	0.000046	10361.221	157	111	11
112	28805.016	0.000052	10447.509	158	112	11
113	28804.774	0.000047	10534.728	159	113	11

Table B.2: (continued)

I	F	MCV	CPU	NFUNC	NGRAD	NACT
114	28804.650	0.000028	10621.516	160	114	11
115	28804.519	0.000016	10708.163	161	115	11
116	28804.413	0.000012	10794.295	162	116	11
117	28804.370	0.000009	10880.664	163	117	11
118	28804.359	0.000005	10967.497	164	118	11
119	28804.343	0.000002	11053.979	165	119	11
120	28804.323	0.000001	11141.017	166	120	11
121	28804.276	0.000004	11227.554	167	121	11
122	28804.218	0.000002	11314.499	168	122	11
123	28804.136	0.000003	11401.379	169	123	11
124	28804.043	0.000006	11488.469	170	124	11
125	28803.950	0.000009	11575.026	171	125	11
126	28803.834	0.000012	11661.426	172	126	11
127	28803.640	0.000016	11747.438	173	127	11
128	28803.276	0.000022	11834.108	174	128	11
129	28802.665	0.000022	11920.758	175	129	11
130	28801.807	0.000021	12007.399	176	130	11
131	28800.894	0.000048	12093.698	177	131	11
132	28799.979	0.000085	12180.109	178	132	11
133	28798.585	0.000125	12266.813	179	133	11
134	28795.986	0.000213	12353.771	180	134	11
135	28791.728	0.000309	12440.543	181	135	11
136	28786.395	0.000299	12527.532	182	136	11
137	28781.455	0.000289	12615.335	183	137	11
138	28777.596	0.000465	12702.218	184	138	11
139	28775.589	0.000447	12788.570	185	139	11
140	28774.501	0.000228	12875.671	186	140	11
141	28773.277	0.000336	12962.383	187	141	11
142	28773.236	0.000143	13049.531	188	142	11
143	28773.107	0.000078	13136.056	189	143	11
144	28772.447	0.000078	13222.859	190	144	11
145	28772.390	0.000027	13309.557	191	145	11
146	28772.323	0.000016	13395.903	192	146	11
147	28772.357	0.000002	13482.182	193	147	11
148	28772.338	0.000002	13568.188	194	148	11
149	28772.315	0.000001	13653.641	195	149	11
150	28772.306	0.000001	13738.863	196	150	11

Table B.3: ICW: ASTROS Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	176.326	0.0749721	2356.0	1	1	351
2	91.3046	0.0035798	2595.8	2	2	350
3	69.8715	0.0373328	2940.7	3	3	350
4	53.6372	0.0246809	3131.5	4	4	350
5	46.0499	0.0041803	3556.7	5	5	350
6	43.3006	0.0102377	4049.0	6	6	350
7	42.6345	0.0021453	4393.3	7	7	350
8	42.4831	0.0002466	4434.1	8	8	350

Table B.4: ICW (scaled): ASTROS-NLPQL Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	176.326	0.074972	291.639	1	1	1
2	139.595	0.058222	529.350	3	2	2
3	117.385	0.047941	774.726	5	3	3
4	102.214	0.113125	1022.676	7	4	5
5	88.472	0.166341	1277.569	9	5	16
6	82.652	0.131279	1535.698	11	6	14
7	77.689	0.116895	1789.719	13	7	13
8	73.406	0.104445	2043.860	15	8	15
9	69.717	0.092003	2300.579	17	9	16
10	66.439	0.082243	2556.200	19	10	18
11	63.579	0.074359	2812.407	21	11	19
12	52.314	0.177490	3078.169	23	12	22
13	48.737	0.104046	3347.011	25	13	23
14	39.633	0.726435	3654.681	26	14	96
15	40.002	0.269707	3957.493	27	15	69
16	39.993	0.242442	4321.719	29	16	63
17	40.128	0.191806	4675.822	31	17	51
18	40.298	0.145579	5032.965	33	18	50
19	41.185	0.097423	5340.059	34	19	48
20	40.882	0.088096	5718.729	36	20	49
21	40.820	0.079782	6143.683	39	21	45
22	40.827	0.077850	6522.221	41	22	41
23	40.839	0.057393	6887.639	43	23	41
24	40.859	0.037710	7258.842	45	24	38
25	40.964	0.028279	7632.488	47	25	37
26	41.141	0.021487	7997.387	49	26	39
27	41.219	0.082091	8362.117	51	27	45
28	41.229	0.087521	8725.266	53	28	43
29	41.258	0.064623	9089.373	55	29	40
30	41.347	0.010361	9462.399	57	30	39
31	41.369	0.009025	9829.388	59	31	39
32	41.392	0.011864	10191.055	61	32	41
33	41.415	0.006352	10554.456	63	33	42
34	41.450	0.008863	10919.897	65	34	45
35	41.482	0.011377	11285.005	67	35	45
36	41.488	0.022204	11659.239	69	36	45
37	41.497	0.017484	12028.468	71	37	46
38	41.509	0.012004	12379.808	73	38	44
39	41.519	0.008640	12743.577	75	39	45
40	41.524	0.003274	13094.158	77	40	45
41	41.529	0.001810	13440.160	79	41	46
42	41.534	0.002202	13786.011	81	42	46
43	41.535	0.003663	14175.080	84	43	46
44	41.538	0.002330	14518.517	86	44	44
45	41.541	0.003340	14910.611	89	45	45
46	41.544	0.004227	15254.171	91	46	45
47	41.548	0.004364	15595.949	93	47	45
48	41.551	0.004560	15935.681	95	48	46

Table B.5: ICW (unscaled): ASTROS-NLPQL Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	176.326	0.074972	301.649	1	1	1
2	12.977	10.734125	1130.581	2	2	435
3	19.620	3.565411	1991.375	3	3	283
4	25.236	1.598893	2687.145	4	4	200
5	31.770	0.739623	3368.517	5	5	126
6	35.983	0.462907	4062.834	6	6	77
7	38.932	0.458608	4748.065	7	7	60
8	40.987	0.620461	5418.115	8	8	55
9	42.344	0.496422	6092.010	9	9	39
10	43.130	0.213566	6778.817	10	10	37
11	42.949	0.140882	7462.205	11	11	43
12	42.750	0.117465	8182.694	12	12	41
13	42.600	0.081905	8931.231	14	13	42
14	42.508	0.087616	9700.129	16	14	42
15	42.316	0.191833	10371.999	17	15	43
16	42.124	0.110829	11067.904	18	16	46
17	41.862	0.082706	11738.614	19	17	49
18	41.727	0.046709	12447.945	21	18	44
19	41.698	0.031755	13141.380	23	19	45
20	41.680	0.049737	13860.339	25	20	42
21	41.659	0.063721	14574.190	26	21	47
22	41.621	0.035001	15296.253	28	22	45
23	41.621	0.034832	16261.783	34	23	45
24	41.597	0.021561	17005.426	36	24	45
25	41.588	0.055925	17741.150	38	25	45
26	41.587	0.042885	18429.418	40	26	43
27	41.581	0.038715	19158.195	42	27	45
28	41.578	0.033942	19874.486	44	28	44
29	41.579	0.021147	20616.873	46	29	41
30	41.582	0.016146	21361.938	48	30	40
31	41.583	0.017772	22104.818	50	31	40
32	41.586	0.017194	22808.852	52	32	40
33	41.590	0.013049	23560.695	54	33	39
34	41.591	0.004599	24251.572	56	34	39
35	41.591	0.008654	25046.492	59	35	39
36	41.593	0.003695	25800.477	61	36	40
37	41.594	0.003518	26542.926	63	37	39
38	41.594	0.003267	27319.346	65	38	39

Table B.6: HALE: ASTROS-NLPQL Iteration History

I	F	MCV	CPU	NFUNC	NGRAD	NACT
1	3255.981	3.616151	3779.254	1	1	68
2	2790.074	1.600892	7200.729	2	2	59
3	2669.316	1.228016	10735.062	4	3	55
4	2606.707	1.111547	14315.552	6	4	53
5	2539.292	1.005891	18039.311	8	5	48
6	2470.838	0.910062	21939.859	10	6	42
7	2411.810	0.823141	25927.295	12	7	39
8	2362.244	0.744329	29902.691	14	8	39
9	2320.490	0.672897	33935.895	16	9	39
10	2285.157	0.608174	37853.961	18	10	36

Table B.6: (continued)

I	F	MCV	CPU	NFUNC	NGRAD	NACT
11	2255.130	0.549547	41742.414	20	11	34
12	2229.462	0.496456	45655.289	22	12	30
13	2207.304	0.448398	49555.547	24	13	31
14	2187.975	0.404908	53514.414	26	14	31
15	2158.506	0.338180	57380.086	28	15	30
16	2125.306	0.258464	61246.934	30	16	29
17	2083.680	0.150541	64937.727	32	17	31
18	2033.832	0.800140	68700.352	33	18	64
19	2012.234	0.188608	72349.703	34	19	45
20	1973.680	0.444624	76139.008	35	20	47
21	1929.514	0.343728	79969.609	36	21	35
22	1894.753	0.270401	84036.680	37	22	49
23	1865.296	0.236770	88372.750	38	23	60
24	1839.557	0.281212	92952.438	39	24	59
25	1828.110	0.158032	97931.211	41	25	46
26	1821.233	0.125253	103259.703	43	26	43
27	1815.713	0.105952	109091.898	45	27	41
28	1809.732	0.088458	115067.109	47	28	41
29	1803.067	0.070089	121450.156	49	29	41
30	1797.227	0.055588	127815.711	51	30	44
31	1791.532	0.042748	134343.281	53	31	44
32	1785.466	0.064621	141007.016	55	32	46
33	1777.983	0.036153	147787.031	57	33	49
34	1765.082	0.238837	154785.969	58	34	71
35	1761.364	0.031342	161700.312	59	35	68
36	1754.417	0.011728	168758.234	60	36	68
37	1746.482	0.017811	176345.828	61	37	64
38	1743.126	0.035493	184128.609	62	38	69
39	1739.437	0.099594	192326.031	63	39	69
40	1735.731	0.078152	200541.484	64	40	75
41	1733.671	0.013927	209195.984	65	41	81
42	1731.204	0.002037	218109.844	66	42	72
43	1728.450	0.004013	227074.234	67	43	77
44	1725.550	0.092404	236058.562	68	44	82
45	1722.779	0.011033	245225.266	69	45	81
46	1720.082	0.009193	254460.172	70	46	82
47	1717.360	0.007304	263861.000	71	47	81
48	1713.349	0.008837	273281.062	72	48	84
49	1709.689	0.005778	282910.344	73	49	87
50	1704.703	0.004082	292367.781	74	50	80
51	1700.296	0.060179	301912.594	75	51	81
52	1695.990	0.005372	311680.656	76	52	77
53	1690.252	0.023970	321593.750	77	53	82
54	1681.699	0.017465	331599.875	78	54	79
55	1675.253	0.016274	341521.688	79	55	79
56	1674.543	0.014665	351506.156	81	56	79
57	1672.870	0.013507	361706.969	83	57	77
58	1670.712	0.012039	371887.156	85	58	77
59	1668.187	0.010740	382106.375	87	59	78
60	1665.894	0.009528	392419.000	89	60	76
61	1663.717	0.008403	402795.875	91	61	79
62	1661.001	0.007409	413175.750	93	62	80
63	1658.039	0.006232	423503.188	95	63	82

Table B.6: (continued)

I	F	MCV	CPU	NFUNC	NGRAD	NACT
64	1655.723	0.005512	433818.688	97	64	83
65	1653.465	0.002592	444165.875	99	65	81
66	1651.183	0.002366	454676.562	101	66	81
67	1648.746	0.002142	465131.219	103	67	79
68	1646.039	0.002042	475678.469	105	68	79
69	1643.017	0.001935	486301.781	107	69	78
70	1639.923	0.002157	496884.812	109	70	79
71	1636.712	0.002710	507416.969	111	71	79
72	1633.146	0.008631	517663.562	113	72	83
73	1593.084	1.325360	528171.562	114	73	126
74	1587.990	0.655727	538380.312	115	74	110
75	1582.243	0.198173	548607.500	116	75	87
76	1575.541	0.137358	558951.625	118	76	80
77	1571.832	0.124146	569481.812	120	77	78
78	1566.539	0.112205	579700.750	122	78	73
79	1560.427	0.100104	589924.875	124	79	72
80	1554.297	0.086229	600195.062	126	80	66
81	1547.530	0.073722	610376.125	128	81	64
82	1543.036	0.065304	620529.438	130	82	64
83	1537.696	0.065919	630827.312	132	83	68
84	1531.592	0.125743	641085.688	134	84	67
85	1527.855	0.102366	651383.000	136	85	71
86	1526.088	0.091200	661599.875	138	86	69
87	1524.513	0.082344	671954.500	140	87	70
88	1523.005	0.074261	682487.000	142	88	71
89	1521.581	0.066853	692982.562	144	89	71
90	1520.211	0.060287	703369.188	146	90	74
91	1518.914	0.054296	713754.875	148	91	74

Appendix C. *Source Code*

Source code for this research consists of a Unix shell, *sqpast.csh*, to control the optimization loop and the driver for Shittkowski's NLPQL code, em sqpast.f. It was compiled on a Convex Unix-based system at Wright Aeronautical Laboratories. The source code for the NLPQL subroutine is omitted at the request of its author.

C.1 *Optimization Loop Control*

```
#!/bin/csh
set zero = "0"
set ext = ".inc"
set rext = ".rst"
set pfile = "sqpast.dat"
set pext = ".inp"
set lext = ".log"
set oext = ".out"
set text = ".tim"

set probname = $argv[1]
if (-e $probname$lext) rm $probname$lext
if (-e $probname$oext) rm $probname$oext
if (-e $probname$rext) rm $probname$rext
if (-e $probname$text) rm $probname$text
if (-e $pfile) rm $pfile
if (-e sqpast.ed) rm sqpast.ed
cp $probname$zero$ext $probname$ext

echo " "
echo "Working problem:  $probname"
echo "    2    0  $probname" > $pfile
echo "0.0" > sqpast.clk

set control = (`head -1 sqpast.dat`)
set c1 = $control[1]
set c2 = $control[2]

while ($c1 < '3' || $c2 < '0')
   if ($c2 == '-1') then
      echo "REPLACE 1254" > sqpast.ed
   else
      echo "REPLACE 1791, 1793" > sqpast.ed
   endif
```

```
    astros $probname$pext
    sqpast.exe
    set control = ('head -1 $pfile')
    set c1 = $control[1]
    set c2 = $control[2]
    echo "ifail = $c2"
end
#rm $probname$rext $pfile sqpast.ed sqpast.clk
```

C.2 NLPQL Driver

```
C****************************** SQPAST.F *************************
C  This program is a driver for Schittkowski's NLPQL sequential
C  quadratic programming algorithm.  It was created to interface
C  with the ASTROS (Automated STRuctural Optimization System)
C  software to solve structural optimization problems.  In order to
C  run, it must be compiled with astros.a, the ASTROS library (which
C  contains the database and dynamic memory manager) and the NLPQL
C  object code, nlpqld.o and qld.o.  ASTROS and this program are run
C  sequentially within a Unix loop shell structure, sqpast.csh.
C  This is necessary for data file compatibility.
C
C     AUTHOR:  Capt Mark A. Abramson, AFIT/ENS, GOR-94M, 31-JAN-94
C*****************************************************************
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER M, ME, N, MMAX, NMAX, MNN2, MAXFUN, MAXIT, IPRINT, MODE
      INTEGER IOUT, IFAIL, LWA, LKWA, LACTIV, ISTAT, MSIZE, RSIZE
      INTEGER IX, IDX, IXU, IXL, IDF, IG, IDG, IU, IC, ID, IWA
      INTEGER IKWA, IACTIV, ICON, ITEMP, IACT, INFO(20), NACT
      DOUBLE PRECISION EPS, ACC, SCBOU, F, CV, MAXCV
      REAL CPU, CPUSQP, CPUAST, CPUOLD, CPU1, CPU2
      CHARACTER*12 INCNAME, LOGNAME, RSTNAME, TIMNAME
      CHARACTER*8 PROB, PROB1, ETYPE, LABEL, CTEMP1, CTEMP2
      CHARACTER*8 CSLIST(4), CSTYPE(4)
      LOGICAL MOVE, FSCALE
C
C Dynamic memory allocation: All data arrays stored in CORE
C
      DOUBLE PRECISION DCORE(1)
      REAL CORE(1)
      INTEGER ICORE(1)
      LOGICAL LCORE(1)
C
      EXTERNAL DBBD
      EXTERNAL X[illegible]D
```

```
      EQUIVALENCE (CORE,ICORE,DCORE,LCORE)
      COMMON/MEMLEN/LENGTH
      COMMON/CMACHE/EPS
      DATA CSLIST/'BCID','SCNUM','CTYPE','PNUM'/
      DATA CSTYPE/'ASC','ASC','ASC','ASC'/
C
C Initialize and assign appropriate problem and file names
C
      CALL MMINIT(LENGTH)
      CALL MMBASE(CORE)
      OPEN(1,FILE='sqpast.dat', STATUS='OLD', FORM='FORMATTED')
      READ(1,'(2I5,2X,A8)') MODE, IFAIL, PROB
      CLOSE(1)
      MOVE = .TRUE.
      IF (IFAIL .EQ. -1) MOVE = .FALSE.
      CALL TIMING(IFAIL, CPUAST, CPUOLD)
      L = INDEX(PROB,' ') - 1
      IF (L .LE. 0) L = 8
      INCNAME = PROB(1:L)//'.inc'
      LOGNAME = PROB(1:L)//'.log'
      RSTNAME = PROB(1:L)//'.rst'
      TIMNAME   = PROB(1:L)//'.tim'
C
C Open and sort constraint database, get required memory
C
      PROB1 = PROB
      CALL DBINIT(PROB1,'GORGAM','OLD','R/W',' ')
      CALL DBEXIS('&SORTNEW', IDBEX, IDBTYP)
      IF (IDBEX .EQ. 1 .AND. IDBTYP .EQ. 1) CALL DBDEST('&SORTNEW')
C
      CALL RESORT('CONST', 4, CSTYPE, CSLIST, CORE)
      CALL DBOPEN('CONST',INFO, 'RO','NOFLUSH',ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(1)
      M = INFO(3)
      MMAX = M + 1
      ME = 0
C
      CALL MMGETB('CON', 'RSP', MMAX + M, 'NLPQL', ICON, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(2)
      IACT = ICON + MMAX
C
C Retrieve ASTROS constraint data (values, and flag if active)
C
      CALL RECOND('CONST', 'CTYPE', 'GT', 5)
      CALL REENDC
```

```
      CALL REPROJ('CONST', 1, 'CVAL')
      CALL REGB('CONST', CORE(ICON), NAERO, ISTAT)
C
      CALL RECOND('CONST', 'CTYPE', 'LE', 5)
      CALL REENDC
      CALL REPROJ('CONST', 1, 'CVAL')
      CALL REGB('CONST', CORE(ICON+NAERO), NSTAT, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(3)
C
      CALL RECOND('CONST', 'CTYPE', 'GT', 5)
      CALL REENDC
      CALL REPROJ('CONST', 1, 'ACTVFLAG')
      CALL REGB('CONST', ICORE(IACT), NAERO, ISTAT)
C
      CALL RECOND('CONST', 'CTYPE', 'LE', 5)
      CALL REENDC
      CALL REPROJ('CONST', 1, 'ACTVFLAG')
      CALL REGB('CONST', ICORE(IACT+NAERO), NSTAT, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(4)
      CALL DBCLOS('CONST')
C
      CALL MMGETB('G', 'RDP', MMAX, 'NLPQL', IG, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(5)

C
C Convert constraint values to real*8 and active flag to 0-1.
C
      NACT = 0
      MAXCV = 0.0D0
      DO 10 I = 1, M
         DCORE(IG + I - 1) = -1.0D0*DBLE(CORE(ICON + I - 1))
         IF (ICORE(IACT + I - 1) .NE. 1) ICORE(IACT + I - 1) = 0
         IF (ICORE(IACT + I - 1) .EQ. 1) NACT = NACT + 1
         CV = DCORE(IG + I - 1)
         IF ((CV.LT.0.0D0).AND.(DABS(CV).GT.MAXCV)) MAXCV=DABS(CV)
   10 CONTINUE
C
C Open the design variable database, get required memory
C
      CALL DBOPEN('GLBDES',INFO, 'RO','NOFLUSH',ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(6)
      N = INFO(3)
      NMAX = N + 1
      IF (NMAX .LT. 2) NMAX = 2
C
```

```
      CALL MMGETB('X', 'RSP', 10*N + NMAX, 'NLPQL', IX, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(7)
      IXL = IX + N
      IXU = IXL + N
      IDVID = IXU + N
      IEID = IDVID + N
      IETYPE = IEID + N
      ILNUM = IETYPE + 2*N
      ILBL = ILNUM + N
      IDF = ILBL + 2*N
C
C Retrieve ASTROS design variable data (values, bounds, etc.)
C
      CALL REPROJ('GLBDES', 1, 'VALUE')
      CALL REGB('GLBDES', CORE(IX), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(8)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'VMIN')
      CALL REGB('GLBDES', CORE(IXL), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(9)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'VMAX')
      CALL REGB('GLBDES', CORE(IXU), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(10)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'DVID')
      CALL REGB('GLBDES', ICORE(IDVID), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(11)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'EID')
      CALL REGB('GLBDES', ICORE(IEID), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(12)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'ETYPE')
      CALL REGB('GLBDES', ICORE(IETYPE), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(13)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'LAYRNUM')
      CALL REGB('GLBDES', ICORE(ILNUM), N, IST
```

```
      IF (ISTAT .NE. 0) CALL ERR(14)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'LABEL')
      CALL REGB('GLBDES', ICORE(ILBL), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(15)
      CALL RECLRC('GLBDES')
C
      CALL REPROJ('GLBDES', 1, 'DOBJ')
      CALL REGB('GLBDES', ICORE(IDF), N, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(16)
      CALL RECLRC('GLBDES')
C
C Convert variable values, bounds, and objective gradient to real*8
C
      CALL MMGETB('DX', 'RDP', 3*N+NMAX, 'NLPQL', IDX, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(17)
      IDXL = IDX + N
      IDXU = IDX + N*2
      IDDF = IDX + N*3
      DO 20 I = 1, N
         DCORE(IDX  + I - 1) = DBLE(CORE(IX  + I - 1))
         DCORE(IDXL + I - 1) = DBLE(CORE(IXL + I - 1))
         DCORE(IDXU + I - 1) = DBLE(CORE(IXU + I - 1))
         DCORE(IDDF + I - 1) = DBLE(CORE(IDF + I - 1))
   20 CONTINUE
      CALL DBCLOS('GLBDES')
C
C Calculate current objective function value
C
      F = 0.0D0
      DO 30 I = 1, N
         F = F + DCORE(IDX + I - 1)*DCORE(IDDF + I - 1)
   30 CONTINUE
C
C Define NLPQL parameters and memory sizes
C
      EPS = 1.0D-12
      ACC    = 1.0D-2
      MAXIT  = 200
      MAXFUN = 5
      IPRINT = 2
      IOUT   = 3
C
      MNN2 = M + N + N + 2
```

```
      LWA = MMAX*N+4*MMAX+4*M+18*N+55 + 3*NMAX*NMAX/2+10*N+2*M+10
      LKWA = M + N + N + 19
      LACTIV = 2*MMAX + 15
      MSIZE = MNN2 + NMAX*NMAX + NMAX + LWA
      RSIZE = 8*(2 + MSIZE + MMAX*NMAX) + 4*LKWA + 2*LACTIV + 1
C
C Allocate and initialize exact memory required by NLPQL
C
      CALL MMGETB('U', 'RDP', MSIZE, 'NLPQL', IU, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(18)
      IC = IU + MNN2
      ID = IC + NMAX*NMAX
      IWA = ID + NMAX
C
      CALL MMGETB('TEMP', 'RDP', N, 'NLPQL', ITEMP, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(21)
      CALL MMGETB('DG', 'RDP', MMAX*NMAX, 'NLPQL', IDG, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(22)
C
      CALL MMGETB('KWA', 'RSP', LKWA, 'NLPQL', IKWA, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(19)
      CALL MMGETB('ACTIVE', 'RSP', LACTIV, 'NLPQL', IACTIV, ISTAT)
      IF (ISTAT .NE. 0) CALL ERR(20)
C
C Read data (including constraint gradient data) from restart file
C
      OPEN(99, FILE=RSTNAME, STATUS='OLD', FORM='UNFORMATTED',
     +          RECL=RSIZE, ERR=998)
      READ(99) DCORE(IDDF + NMAX), DCORE(IG + MMAX),
     +        (DCORE(IU  + I - 1), I = 1, MSIZE),
     +        (DCORE(IDG + I - 1), I = 1, MMAX*NMAX),
     +        (CORE(IKWA + I - 1), I = 1, LKWA),
     +        (LCORE(IACTIV + I - 1), I = 1, LACTIV)
      CLOSE(99)
      SCBOU = -1
      GOTO 999
  998 DO 60 I = 1, LKWA
         ICORE(IKWA + I - 1) = 0
   60 CONTINUE
      DO 61 I = 1, MMAX*NMAX
         DCORE(IDG + I - 1) = 0.0D0
   61 CONTINUE
      SCBOU = 1.0D+3
C
C Retrieve matrix of active constraint gradients
```

```
C
  999 IF ((IFAIL .NE. -1) .AND. (NACT .GT. 0)) THEN
         CALL DBOPEN('AMAT',INFO, 'RO','NOFLUSH',ISTAT)
         IF (ISTAT .NE. 0) CALL ERR(23)
         DO 50 I = 1, M
            IF (ICORE(IACT+I-1) .EQ. 1) THEN
               CALL MXUNP('AMAT',DCORE(ITEMP),1,N)
               DO 40 J = 1, N
                  DCORE(IDG+MMAX*(J-1)+(I-1)) = -DCORE(ITEMP+J-1)
   40          CONTINUE
            ENDIF
   50    CONTINUE
         CALL DBCLOS('AMAT')
      ENDIF
C
C Convert active constraint flags to logical for NLPQL1
C
      DO 66 I = 1, M
         LCORE(IACTIV+I-1) = .FALSE.
         IF (ICORE(IACT+I-1).EQ.1) LCORE(IACTIV+I-1) = .TRUE.
   66 CONTINUE
C
C Call the NLPQL optimizer, getting new design point
C
      OPEN(IOUT, FILE=LOGNAME, STATUS='UNKNOWN', ACCESS='APPEND',
     +           FORM='FORMATTED')
      CALL XXCPU(CPU1)
      CALL NLPQL1(M, ME, MMAX, N, NMAX, MNN2, DCORE(IDX), F,
     +   DCORE(IG), DCORE(IDDF), DCORE(IDG), DCORE(IU), DCORE(IDXL),
     +   DCORE(IDXU), DCORE(IC), DCORE(ID), ACC, SCBOU, MAXFUN,
     +   MAXIT, IPRINT, MODE, IOUT, IFAIL, DCORE(IWA), LWA,
     +   ICORE(IKWA), LKWA, LCORE(IACTIV), LACTIV, .TRUE., .TRUE.)
      CALL XXCPU(CPU2)
      CLOSE(IOUT)
      CPUSQP = CPU2 - CPU1
C
C Convert new design variables back to single precision
C
      DO 70 I = 1, N
         CORE(IX + I - 1) = SNGL(DCORE(IDX + I - 1))
   70 CONTINUE
C
C Create new ASTROS .inc file to be included in ASTROS input file
C
      OPEN(9, FILE=INCNAME, STATUS='UNKNOWN', FORM='FORMATTED')
```

```
      DO 90 I = 1, N
         II = I - 1
         CALL DBMDHC(ICORE(IETYPE+2*II),ETYPE,8)
         CALL DBMDHC(ICORE(ILBL+2*II),LABEL,8)
         WRITE(9,8) ICORE(IDVID + II), ICORE(IEID + II), ETYPE,
     +               CORE(IXL + II), CORE(IXU + II), CORE(IX + II),
     +               ICORE(ILNUM + II), LABEL
    8 FORMAT('DESELM  ',2I8,A8,F8.6,F8.2,F8.5,I8,A8)
   90 CONTINUE
      CLOSE(9)
C
C Save pertinent data to files and terminate
C
      MODE = 13
      OPEN(1,FILE='sqpast.dat', STATUS='OLD', FORM='FORMATTED')
      WRITE(1,'(2I5,2X,A8)') MODE, IFAIL, PROB
      CLOSE(1)
C
      OPEN(99, FILE=RSTNAME, STATUS='UNKNOWN', FORM='UNFORMATTED',
     +         RECL=RSIZE)
      WRITE(99) DCORE(IDDF + NMAX), DCORE(IG + MMAX),
     +         (DCORE(IU  + I - 1), I = 1, MSIZE),
     +         (DCORE(IDG + I - 1), I = 1, MMAX*NMAX),
     +         (CORE(IKWA + I - 1), I = 1, LKWA),
     +         (LCORE(IACTIV + I - 1), I = 1, LACTIV)
      CLOSE(99)
C
      CPU = CPUOLD + CPUAST + CPUSQP
      OPEN(21, FILE='sqpast.clk', STATUS='OLD', FORM='FORMATTED')
      WRITE(21,'(F11.3)') CPU
      CLOSE(21)
C
      IF (MOVE) THEN
        OPEN(11, FILE=TIMNAME, STATUS='UNKNOWN', ACCESS='APPEND',
     +          FORM='FORMATTED')
        FSCALE = LCORE(IACTIV + 2*MMAX + 6)
        IF (IFAIL .NE. 0 .AND. FSCALE) F = F/DCORE(IWA + MMAX)
        WRITE(11,'(F15.3, 2X, F15.6, 2X, F11.3, 3I5)')
     +     F, MAXCV, CPU, ICORE(IKWA), ICORE(IKWA+1), NACT
      ENDIF
C
      STOP
      END
C
C
```

```
C  **************************** SUBROUTINES ************************
C
C  This subroutine computes previous ASTROS CPU time.
C
      SUBROUTINE TIMING(IFAIL, AST, OLD)
      REAL AST, SS(2)
      INTEGER MODE, IFAIL, HH(2), MM(2)
      CHARACTER*26 TEMP, TFLAG

      OPEN(21, FILE='sqpast.clk', STATUS='OLD', FORM='FORMATTED')
      READ(21,'(F11.3)') OLD
      CLOSE(21)
C
      OPEN(22, FILE='fort.98', STATUS='UNKNOWN', FORM='FORMATTED')
      TFLAG = '*** BEGIN ASTROS ***      '
      IF (IFAIL .EQ. -1) TFLAG = 'BEGIN OPTIMIZATION        '
      IF (IFAIL .EQ. -2) TFLAG = '    *** MAKDFV      BEGIN'
      DO 100 I = 1, 1000
         READ(22,11) TEMP
         IF (TEMP .EQ. TFLAG) GOTO 101
  100 CONTINUE
  101 BACKSPACE 22
      READ(22,10) HH(1),MM(1),SS(1)
C
      DO 200 I = 1, 1000
         READ(22,11) TEMP
         IF (TEMP .EQ. '    *** EXIT         BEGIN') GOTO 201
  200 CONTINUE
  201 BACKSPACE 22
      READ(22,10) HH(2),MM(2),SS(2)
      CLOSE(22)
      AST = (3600*HH(2)+60*MM(2)+SS(2))-(3600*HH(1)+60*MM(1)+SS(1))
C
   10 FORMAT(T11, 2(I2,1X),F4.1)
   11 FORMAT(T22, A26)
      RETURN
      END
C
C This subroutine tracks database access errors
C
      SUBROUTINE ERR(K)
      INTEGER K, NCORE
      CALL MMSTAT(NCORE)
      PRINT *, 'DB ACCESS ERROR # ', K, ' -- ',
     +         NCORE, 'RSP WORDS LEFT'
```

```
      STOP
      END
C
C These subroutines, used for explicitly defined functions, are
C empty for structural applications.
C
      SUBROUTINE NLFUNC(M,ME,MMAX,N,F,G,X,ACTIVE)
      RETURN
      END
C
      SUBROUTINE NLGRAD(M,ME,MMAX,N,F,G,DF,DG,X,ACTIVE,WA)
      RETURN
      END
```

Bibliography

1. Arnold, Edward. *Aerodynamics for Engineering Students* (3rd Edition). London: E.L. Houghton and N.B. Carruthers, 1986.
2. Belegundu, Ashok D. and Jasbir S. Arora. "A Recursive Quadratic Programming Method with Active Set Strategy for Optimal Design," *International Journal for Numerical Methods in Engineering*, *20*:803–816 (1984).
3. Belegundu, Ashok D. and Jasbir S. Arora. "A Study of Mathematical Programming Methods for Structural Optimization," *International Journal for Numerical Methods in Engineering*, *21*:1583–1624 (1985).
4. Best, M.J. and N. Chakravarti. "Active set algorithms for Isotonic Regression: A Unifying Framework," *Mathematical Programming*, *47*(3):425–439 (August 1990).
5. Canfield, R. A., 1988. unpublished computer output for the HALE problem solved by optimality criteria methods.
6. Canfield, R.A. and V.B. Venkayya. "Implementation of Generalized Optimality Criteria in a Multidisciplinary Environment," *Journal of Aircraft*, *27*(12):1037–1042 (December 1990).
7. Canfield, R.A., R.V. Grandhi and V.B. Venkayya. "Optimum Design of Structures with Multiple Constraints," *AIAA Journal*, *26*(1):78–85 (January 1988).
8. Chamberlain, R.M. "Some Examples of Cycling in Variable Metric Methods for Constrained Optimization," *Mathematical Programming*, *16*:378–384 (1979).
9. Das, Alok, Eugene M. Cliff, and Henry J. Kelley. "An Active-Constraint Logic for Non-linear Programming," *Optimal Control Applications and Methods*, *5*:221–234 (1984).
10. Dax, Achiya. "On Computational Aspects of Bounded Linear Least Squares Problems," *ACM Transactions on Mathematical Software*, *17*:64–73 (March 1991).
11. Fletcher, R. "A General Quadratic Programming Algorithm," *Journal of the Institute of Mathematics and Applications*, *7*:76–91 (1971).
12. Gill, Philip E., Walter Murray, and Margaret H. Wright. *Practical Optimization*. London: Academic Press, Inc., 1981.
13. Han, S.P. "Superlinearly Convergent Variable Metric Methods for General Nonlinear Programming," *Mathematical Programming*, *11*:263–282 (1976).
14. Han, S.P. "A Globally Convergent Method for Nonlinear Programming," *Journal of Optimization Theory and Applications*, *22*:297–309 (1977).
15. Haug, Edward J. and Jasbir S. Arora. *Applied Optimal Design*. New York: John Wiley and Sons, Inc., 1979.
16. Khot, N.S., L. Berke, and V.B. Venkayya. "Comparison of Optimality Criteria Algorithms for Minimum Weight Design of Structures." *AIAA/ASME/SAE 19th Structures, Structural Dynamics, and Materials Conference*. 37–46. April 1978.
17. Kirsch, Uri. *Optimum Structural Design*. New York: McGraw-Hill, Inc., 1981.

18. Lenard, Melanie L. "A Computational study of active set strategies in Nonlinear Programming with Linear Constraints," *Mathematical Programming*, *16*:81–97 (November 1979).

19. McCormick, Garth P. *Nonlinear Programming: Theory, Algorithms, and Applications*. New York: John Wiley and Sons, Inc., 1983.

20. Neill, D.J. and D.L. Herendeen. *ASTROS Enhancements, Volume I–ASTROS User's Manual*. Technical Report WL-TR-93-3025, Wright Laboratory, March 1993.

21. Neill, D.J., D.L. Herendeen, and R.L. Hoesly. *ASTROS Enhancements, Volume II–ASTROS Programmer's Manual*. Technical Report WL-TR-93-3038, Wright Laboratory, March 1993.

22. Panier, Eliane R. "An Active Set Method for Solving Linearly Constrained Nonsmooth Optimization Problems," *Mathematical Programming*, *37*:269–292 (1987).

23. Papalambros, Panos Y. and Douglass J. Wilde. *Principles of Optimal Design*. Cambridge: Cambridge University Press, 1988.

24. Powell, M.J.D. "The Convergence of Variable Metric Methods for Nonlinearly Constrained Optimization Calculations." *Nonlinear Programming, 3* edited by O.L. Mangasarian and R.R. Meyer and S.M. Robinson, New York: Academic Press, 1978.

25. Powell, M.J.D. "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations." *Lecture Notes in Mathematics, 630* edited by G.A. Watson, Berlin: Springer-Verlag, 1978.

26. Powell, M.J.D. *ZQPCVX, A FORTRAN Subroutine for Convex Programming*. Technical Report DAMTP/1983/NA17, University of Cambridge, 1993.

27. Pshenichny, B.N. "Algorithms for General Mathematical Programming Problems," *Kibernetika*, (5):120–125 (September-October 1970). (In Russian—translated in *Cybernetics*, May 1973).

28. Reklaitis, G.V., A. Ravindran, and K.M. Ragsdell. *Engineering Optimization: Methods and Applications*. New York: John Wiley and Sons, Inc., 1983.

29. Rosen, J.B. "The Gradient Projection Method for Nonlinear Programming. Part I: Linear Constraints," *SIAM Journal of Applied Mathematics*, *8*(1):181–217 (1960).

30. Rosen, J.B. "The Gradient Projection Method for Nonlinear Programming. Part I: Nonlinear Constraints," *SIAM Journal of Applied Mathematics*, *9*(4):514–532 (1961).

31. Sargent, R.W.H. "Projection Methods for Non-linear Programming Problems," *Mathematical Programming*, *4*:245–268 (1973).

32. Schittkowski, K., C. Zillober, and R. Zotemantel. *Numerical Comparison of Nonlinear Programming Algorithms for Structural Optimization*. Technical Report 453, Mathematisches Institut, Universität Bayreuth, 1993.

33. Schittkowski, Klaus. "The Nonlinear Programming Method of Wilson, Han, and Powell with an Augmented Lagrangian Type Line Search Function," *Numerische Mathematik*, *38*:83–114 (1981).

34. Schittkowski, Klaus. "On the Convergence of a Sequential Quadratic Programming Method with an Augmented Lagrangian Line Search Function," *Mathematische Operationsforschung und Statistik, Series Optimization*, *14*:197–216 (1983).

35. Schittkowski, Klaus. "NLPQL: A FORTRAN Subroutine Solving Constrained Nonlinear Programming Problems," *Annals of Operations Research*, *5*:485–500 (1985/6).

36. Schittkowski, Klaus, 1991. unpublished NLPQL algorithm FORTRAN source code.

37. Schmidt, J. William and Robert P. Davis. *Foundations of Analysis in Operations Research*. Orlando: Academic Press, Inc., 1981.

38. Schmit, L.A. and H. Miura. *Approximation Concepts for Efficient Structural Synthesis*. Technical Report CR-2552, NASA, March 1976.

39. Shigley, Joseph Edward and Larry D. Mitchell. *Mechanical Engineering Design* (4th Edition). New York: McGraw-Hill, Inc., 1983.

40. Thanedar, P.B., J.S. Arora, C.H. Tseng, O.K. Lim, and G.J. Park. "Performance of Some SQP Algorithms on Structural Design Problems," *International Journal for Numerical Methods in Engineering*, *23*:2187–2203 (1986).

41. Tseng, C.H. and J.S. Arora. "On Implementation of Computational Algorithms for Optimal Design 1: Preliminary Investigation," *International Journal for Numerical Methods in Engineering*, *26*:1365–1382 (1988).

42. Vanderplaats, Garret N. "An Efficient Feasible Directions Algorithm for Design Synthesis," *AIAA Journal*, *22*(11):1633–1640 (November 1984).

43. Vanderplaatz, G.N. *ADS - A FORTRAN Program for Automated Design of Synthesis*. Technical Report CR-172460, NASA, October 1984.

44. Venkayya, V.B. "Generalized Optimality Criteria Method." *Structural Optimzation: Status and Promise* edited by Manohar P. Kamat, chapter 8, 151–182, AIAA, 1993.

45. Wilson, R.B. *A Simplical Algorithm for Concave Programming*. PhD dissertation, Graduate School of Business Administration, Harvard University, 1963.

46. Zotemantel, R. "MBB-LAGRANGE: A Computer Aided Structural Design System." *Software Systems for Structural Optimization* edited by H. Hörnlein and K. Schittkowski, Basel: Birkhäuser, 1993.

47. Zoutendijk, G. *Method of Feasible Directions*. Amsterdam: Elsevier, 1960.

Vita

Capt Mark A. Abramson was born on 31 January 1963 in Miami, Florida. He graduated from Hayfield High School in Alexandria, Virginia in 1981 and then attended Brigham Young University in Provo, Utah with a four-year Air Force ROTC scholarship. After taking a two-year leave of absence to serve his church in Switzerland, he returned to BYU and completed his studies, graduating with a Bachelor of Science degree in Computational Mathematics in April 1987.

Upon graduation, Captain Abramson received a reserve commission in the USAF and served his first tour at Edwards AFB, California. Assigned to the 31st Test and Evaluation Squadron, he was responsible for evaluating navigation performance and reliability of the AGM-129 Advanced Cruise Missile during both Initial and Follow-on Operational Test and Evaluation. While there, he also provided analysis support for the B-52 Global Positioning System/Conventional Weapons Integration test program, a system which later performed admirably during Operation Desert Storm. In May 1992, he entered the Graduate School of Engineering, Air Force Institute of Technology as a student of both Operations Research and Applied Mathematics. In January 1994, he was awarded a regular commission in the Air Force. Upon graduation, Captain Abramson is assigned to the Air Force Logistics Management Agency, Maxwell AFB, Alabama.

Captain Abramson is a member of the Tau Beta Pi and Omega Rho national honor societies, as well as the Society for Industrial and Applied Mathematics and the Military Operations Research Society.

Captain Abramson and his wife, the former Michelle Marie Inman of Concord, California, have three children: Tommy, Elizabeth, and Daniel (and another soon to come).

Permanent address: 5617 Marble Arch Way
Alexandria, VA 22310