An Interactive Approach for
Solving Multi-Objective Optimization Problems

by

Daniel John Woods


\[A \text{ Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Rice University.}\]
RICE UNIVERSITY

AN INTERACTIVE APPROACH FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

by

DANIEL JOHN WOODS

A Thesis Submitted In Partial Fulfillment Of The Requirements For The Degree

DOCTOR OF PHILOSOPHY

Approved, Thesis Committee:

John E. Dennis Jr.,
Professor of Mathematical Sciences
Chairman

Richard A. Tapia,
Professor of Mathematical Sciences

John Edward Akin,
Professor of Mechanical Engineering

Robert T. Hood,
Assistant Professor of Computer Science

Houston, Texas

May, 1985
AN INTERACTIVE APPROACH FOR
SOLVING MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

DANIEL JOHN WOODS

ABSTRACT

Multi-objective optimization problems are characterized by the need to consider multiple, and possibly conflicting, objectives in the solution process. We present an approach based on the use of interactive computer graphics to obtain qualitative information from a user about approximate solutions. We then use this qualitative information to transform the multi-objective optimization problem into a single-objective optimization problem that we may solve using standard techniques.

Preliminary convergence results for the Nelder-Mead simplex algorithm are presented. Techniques for updating the single-objective problem after each piece of information is obtained from the user are described. These techniques are based on the duality theory for linear and quadratic programming. A software system for the subclass of 1-dimensional curve-fitting problems is also described.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my chairman, John Dennis, to whom I am particularly grateful for his assistance, encouragement and for believing in me and my abilities. I would also like to thank my thesis committee, Richard Tapia for his assistance and advice in many areas, and Bob Hood and Ed Akin for caring about the work contained in this thesis.

This work could not have been possible without the aid of Bill LeFebvre, Mike Caplinger, and Keith Cooper and their various expertises. I am also indebted to Mike Peariman for much of the format of this thesis and for his aid in the theoretical results.

I would especially like to thank Maria Rosa Celis for having given me that which I value most of my years at Rice, her friendship. Lastly, I would like to thank my parents for everything they have given and done for me.

This research supported by NSF grant MCS-81-21884.
TABLE OF CONTENTS

Chapter 1 Introduction .......................................................... 1
  1.1 The Multi-Object Problem .................................................. 1
  1.2 Examples .......................................................................... 2
  1.3 Optimality ........................................................................ 3
  1.4 Our Approach ..................................................................... 7
  1.5 Material to Follow ............................................................. 11

Chapter 2 The Nelder-Mead Simplex Algorithm .......................... 14
  2.1 Introduction ...................................................................... 14
  2.2 Motivation ......................................................................... 15
  2.3 Description ....................................................................... 17
  2.4 Stopping Criteria ............................................................... 24
  2.5 Theoretical Results ............................................................ 27
    2.5.1 Sequences Under Consideration ....................................... 29
    2.5.2 Preliminary Convergence Results ................................. 31

Chapter 3 Mimicking the Users's Choice Criterion ..................... 44
  3.1 Introduction ...................................................................... 44
  3.2 Finding the Weights ........................................................... 45
  3.3 Exploiting Structure in the LP Problem ............................... 48
  3.4 Exploiting Structure in the QP Problem ............................... 54
  3.5 No Feasible Weights .......................................................... 61

Chapter 4 Curve Tailoring .......................................................... 69
  4.1 Introduction ...................................................................... 69
  4.2 What is Required From the User .......................................... 70
  4.3 The System ....................................................................... 73
  4.4 Initializing the Simplex ........................................................ 75

Chapter 5 Summary and Further Research .................................. 80

Bibliography ........................................................................... 82

Appendix A Curve Tailoring System User's Guide ........................ 87

Appendix B Solving an Example Problem .................................... 108

TABLE OF FIGURES

Chapter 2 The Nelder-Mead Simplex Algorithm ......................... 22
  2.1 Iterations of Nelder-Mead Algorithm .................................. 22
  2.2 Operations of Nelder-Mead Algorithm ................................. 23
  2.3 A Problem With Stopping Criterion (NM) ........................... 25
  2.4 Another Problem With Stopping Criterion (NM) ................. 26
  2.5 Counter-Example 1 ............................................................. 30
  2.6 Counter-Example 2 ............................................................. 43

Chapter 3 Mimicking the Users's Choice Criterion ....................... 66
  3.1 L-1 Feasible, L-2 Infeasible ............................................... 66

Chapter 4 Curve Tailoring System User's Guide ......................... 72
  4.1 Typical Curve Tailoring Screen .......................................... 72

Appendix A Curve Tailoring System User's Guide ......................... 92
  A.1 Typical Curve Tailoring Screen ......................................... 92

Appendix B Solving an Example Problem .................................... 111
  B.1 Plot of Initial Point 1 ....................................................... 111
  B.2 Plot of Initial Point 2 ....................................................... 112
  B.3 Plot of Initial Point 3 ....................................................... 113
  B.4 Plot of Nonlinear Least Squares Solution 1 ....................... 114
  B.5 Plot of Nonlinear Least Squares Solution 2 ....................... 115
CHAPTER 1

Introduction

1.1. The Multi-Objective Problem

Multi-objective optimization problems are characterized by the need to consider multiple, and possibly conflicting, objectives in the solution process. We present an approach based on the use of interactive computer graphics to obtain qualitative information from a user about approximate solutions. We then use this qualitative information to transform the multi-objective optimization problem into a single-objective optimization problem that we may solve using standard techniques.

The multi-objective optimization problem can be written as: given a set of criteria \( \{f_1(x), f_2(x), \cdots, f_m(x)\} \), where \( x \in \mathbb{R}^n \) and \( f_i: \mathbb{R}^n \rightarrow \mathbb{R}^1 \), find a solution to

\[
\min_{x \in \mathbb{R}^n} F(x) = (f_1(x), f_2(x), \cdots, f_m(x))^T.
\]  

(1.1)

That is, we would like to minimize \( f_i \) for all \( i \). We refer to the \( f_i(x) \) as the criteria, and the objective is to minimize all of the criteria.

Although we have not yet defined optimality of an \( x^* \in \mathbb{R}^n \) for this problem, we present two example problems in Section 1.2 to aid in the understanding of how diverse the multiple objectives can be. By defining optimality, i.e. a measure of magnitude for \( F(x) \), we may obtain methods for determining a solution \( x^* \). In Section 1.3, we discuss various definitions of optimality and the methods that result. Section 1.4 contains a brief description of our approach for obtaining a solution to multi-objective optimization problems. Section 1.5 introduces the material to be discussed in the remainder of this thesis.

1.2. Examples

In the first example, the \( f_i \)'s are linear for all \( i = 1, 2, \cdots, m \). This problem is due to Boonekamp [1982] and considers the following criteria for an energy model of the Dutch economy:

1. Total costs, capital and operational.
2. \( \text{SO}_2 \) emissions.
3. \( \text{NO}_x \) emissions.
4. Dust.
5. Utilization of Dutch natural gas.
6. Electricity production by nuclear power plants.
8. Loss on Dutch natural gas, i.e. the objective is to maximize profits.
9. Capacity minus production of electricity produced by combined heat-power units, i.e. maximize production.

The variables for this problem are the criteria, e.g., net oil imports. There are
also constraints for this problem, but we have presented only the criteria to show that the objectives may be quite diverse.

The second example is more theoretical and is often nonlinear. Suppose we have a set of 1-dimensional data \((x_i, y_i), i = 1, 2, \ldots, m\), and a model \(y(x, t)\), where \(x \in \mathbb{R}^n\) is a vector of \(n\) parameters, \(t \in \mathbb{R}^1\), and \(y: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^1\). The set of criteria for this problem is defined by

\[
f_i(x) = |y_i - y(x, t)|,
\]
for \(i = 1, 2, \ldots, m\). We call this problem the curve-fitting problem. Often optimality for this problem is defined by a weighted norm such as

\[
\|F(x)\|_W = \|W F(x)\|, 
\]
where \(W\) is a diagonal matrix of weights and \(F\) is given by \((1.1)\). If, for example, the \(l_2\) norm is used, then \((1.2)\) may be written as

\[
\sum_{i=1}^{m} w_i f_i(x)^2.
\]
It is obvious that if there is an \(x^* \in \mathbb{R}^n\) such that \(F(x^*) = (0, 0, \ldots, 0)^T\) in \((1.2)\), then \(x^*\) is optimal. However, it is often the case that no such \(x^*\) exists.

1.3. Optimality

Ignizio [1982] states that there are three classes of methods for solving multi-objective optimization problems. Although he makes this statement in the context of linear programming, the same can be said for nonlinear problems. Ignizio's classifications are:

1. Geometric measure of optimality: In this approach an attempt is made to measure alternatives. For example, a weight is assigned to each criterion and then optimization with respect to the sum of the weighted criteria is performed. Alternatively, the criteria may be ranked by importance and a lexicographic optimization procedure can then be applied to the ordered set of criteria.

2. Efficient solutions: If appropriate weights can not be assigned to conflicting criteria and the criteria can not be ordered in terms of relative importance, then it may be possible to generate the set of nondominated, or Pareto optimal, solutions, see for example Chankong and Haines [1983].

3. Interactive solutions: In this approach, the user provides information interactively which is used to drive the search toward an optimal solution.

In the first classification, the weighting technique transforms the multi-objective problem into the single-objective problem given by

\[
\min_{x \in \mathbb{R}^n} \phi(x, w) = \sum_{i=1}^{m} w_i \mu(f_i(x)) ,
\]
where \(w = (w_1, w_2, \ldots, w_m)^T \in \mathbb{R}^m\) is a vector of weights for the criteria, and \(\mu: \mathbb{R}^1 \rightarrow \mathbb{R}^+\), is some nonnegative function. Thus, the problem is
transformed into a single-objective optimization problem that can be solved using current techniques and available software. The disadvantage of this technique is that weights must be specified. The resultant solution \( x_w^* \) of problem (1.3) will be satisfactory only if appropriate weights can be assigned. Thus, to obtain a satisfactory solution, it may be necessary to iteratively reassign weights and solve each associated single-objective problem.

If the ranking technique of the first classification is used, then it is necessary to solve the unconstrained minimization problem of minimize \( f_1(x) \), then solve the sequence of constrained optimization problems given by: for \( i = 2, 3, \ldots, m \),

\[
\min \ f_1(x) \\
\text{s.t.} \ f_j(x) = f_j^* , \quad \text{for } j = 1, \ldots, i-1 ,
\]

where \( f_j^* \) is the minimum already obtained for the \( j^{th} \) criterion function, and the indices of the functions reflect the order imposed by the ranking. This technique has the disadvantage that an appropriate order must be established, possibly by an iterative process, to obtain a satisfactory solution. Also, the subproblems that must be solved to obtain \( f_j^* \) may be very difficult.

Goal programming is a technique that is related to the above two procedures. In goal programming, a goal \( G_i \) is established for each objective and a negative deviation variable \( \eta_i \geq 0 \) and a positive deviation variable \( \rho_i \geq 0 \) are used to measure the difference between the \( i^{th} \) goal and the value of the \( i^{th} \) criterion. The sum of the positive deviations is then minimized. This may be written as:

\[
\min \sum_{i=1}^{m} \rho_i \\
\text{s.t.} \ f_i(x) + \eta_i - \rho_i = G_i , \quad i = 1, \ldots, m .
\]

The disadvantages of this approach are that the goals must be established, and the resultant problem may be difficult to solve.

Weights may also be added in the goal programming formulation to provide a scaling for the criteria. If weights are added, then the appropriate values for the weights must also be assigned. For linear \( f_i \)'s the resultant problem is a linear programming problem. This is the main area in which goal programming is used.

The second classification, that of efficient solutions, defines the optimal solutions as those for which improvement in one objective results in loss in another. To generate this set of solutions for a nonlinear problem is extremely difficult. In linear problems, the optimal set is the convex hull of an \( m \)-dimensional polyhedron. The disadvantages of this technique are that the solutions may be difficult to obtain, and a satisfactory member of the set of solutions must still be found. For more information regarding these types of methods, see Chankong and Haimes [1983].

In the third classification, a user interactively provides information that is not available at the start of the solution process. For example, the user only
provides information regarding the iterates of the solution process. This information may be that one iterate \( x_i \) provides a better value for the criteria than some other iterate \( x_j \).

Our approach is an interactive solution technique. We obtain information from the user that will enable us to use the weighting technique described briefly in the geometric measure of optimality classification. Our approach assigns an appropriate weight to each criterion by using information that the user provides interactively for the iterates of the solution process. A detailed description of these techniques may be found in Chankong and Haimes [1983], and a survey of interactive techniques may be found in White [1983].

1.4. Our Approach

We assume the user is in an exploratory stage of learning about his problem and the possible solutions. Our approach is based on the assumption that the user wishes to weight each criterion and optimize with respect to a measure of the weighted criteria. We assign appropriate weights to the criteria by asking the user to rank the approximate solutions. The user's rankings enable us to build a partial ordering for the set of trial solutions \( \{x_0, x_1, \ldots, x_k\} \), where \( x_i \) is an iterate of the solution process. For reasons that will become clear, we refer to the first \( n+1 \) elements of the partially ordered set as vertices. We insist that the vertices be ordered, and we denote the vertices by \( v_0, v_1, \ldots, v_n \).

Our method is different from the method of Saaty [1980], Saaty and Vargas [1982], which is presented by Lootsma [1981], [1985], and Kok and Lootsma [1985], in that Saaty's method asks for pairwise ranking information for the set of criteria or dependent variables, whereas we ask for ranking information for a set of approximate solutions or independent variables. In particular, Saaty asks that each criterion be considered pairwise with all other criteria and that a value be chosen from a set of possible values that represent the relative importance of one criterion with respect to another. Both Saaty's method and our method result in weights being assigned to each criterion. Our method may require fewer comparisons to assign appropriate weights. Also, it is easier to rank approximate solutions than to rank a set of criteria.

Our method is closely related to the method of White [1980]. However, White solves the single-objective problem exactly at every iteration. White also provides only one possible criterion function, and makes no allowances for a set of inconsistent rankings. Our work is based on the idea of not solving the single-objective problem until a suitable approximation to the correct problem is identified. Also, we consider the problem of choices by the user that are inconsistent and we perform various operations that enable us to continue in that instance.

To obtain a partial ordering for the set of approximate solutions, we ask the user to rank each unranked point generated by the solution procedure with respect to the best \( n+1 \) points obtained so far. We call the best \( n+1 \) points of
the partially ordered set, vertices, and we ensure that the vertices are totally ordered.

To understand how the ranking procedure is executed, suppose that there are \(k-1\) elements in the partially ordered set and we wish to add a \(k^{\text{th}}\) point \(z_k\). We show the user a representation, preferably a graphical representation, of his problem and the trial solution corresponding to \(z_k\). The user then compares the representation of \(z_k\) with the representations of the vertices of the partially ordered set until he indicates, via an input device, where to insert \(z_k\) into the ordered list of vertices. This process is then repeated with a new trial point furnished by the program.

We ask the user to rank \(z_k\) with respect to only the vertices to keep the number of user comparisons to a reasonable level. Thus, to rank \(z_k\), the user need only compare \(z_k\) with at most \(\log_2(n+1)\) elements of the ordered list of vertices, see Knuth [1973], or Aho, Hopcroft, and Ullman [1974]. We retain \(n+1\) elements as vertices to enable us to execute an algorithm that will choose the next trial point with consideration given to improving the approximate solution. Although we have only \(n+1\) elements of the partially ordered set available to the user, we record all the selections of the user so that we may assign appropriate weights to each criterion in a manner to be discussed later.

There are several reasons for asking that the trial solutions \(z_k\) be improved approximations to the solution. First, we may be unable to mimic a user's choice criterion and are, therefore, unable to transform the multi-objective problem into a single-objective optimization problem. We must then rely upon the technique for generating the trial points to provide a suitable approximate solution for the user. Second, if we try to improve the approximate solutions, then the information we obtain from the user's rankings will be of greater value than the information from the ranking of randomly generated \(z_k\)'s.

A third reason for generating improving approximations to the solution is that we wish to retain the user's interest. If we generate the \(z_k\)'s randomly, then there is little hope of obtaining a satisfactory approximation to the solution. However, if we generate points that exhibit improvement towards a solution, then the user is more likely to have an interest in the next trial points and will continue the process. We also suspect the user will refine his decision process as we improve the solution estimate.

We generate trial points and update the partially ordered set until one of three situations occurs. The first possibility is that the user is satisfied with an approximate solution. In this situation the user may obtain the set of parameters that generated the representation of the solution and halt our procedure. A second possible situation is that we have a good approximation to the user's criterion for choosing the better of a pair of approximate solutions. In this case we may transform the multi-objective problem into a single-objective problem and solve it using standard techniques. We then continue the process with the solution of the single-objective problem as the trial solution \(z_k\). The third situation is that in which we are unable to approximate the user's choice
A method for transforming the multi-objective optimization problem into a single-objective optimization problem is presented in Chapter 3. We assume the user ranks the trial points by a criterion that can be mimicked by using a mathematical formulation. We attempt to find a geometric measure of optimality that will provide the same set of rankings for \{z_0, z_1, \ldots, z_k\} as the user has provided. To do this, we consider a weighted measure of the criteria. If the values of the geometric measure for the elements of the partially ordered set satisfy the partial ordering defined by the user's ranking information, then we say we have mimicked the user's choice criterion for this set. We may then optimize with respect to this mimicking choice criterion which results in a single-objective optimization problem.

In Chapter 4 we describe the software system for the 1-dimensional curve-fitting problem that we have produced to test our ideas. Our system runs on a SUN Workstation and is written in a combination of the FORTRAN and C programming languages. The only machine dependencies of our system are the use of the graphics routines used to represent the user's problem and the trial solutions, and the use of the graphical input interface.

The representations of trial solutions and the user's problem may be thought of as problem dependent since the multiple criteria can range from automobile pollution standards to the amount of calories burned per hour of jogging. It is because of this that our software system considers only the subclass of 1-dimensional curve-fitting problems, i.e., problem (1.2).
CHAPTER 2

The Nelder-Mead Simplex Algorithm

2.1. Introduction

The Nelder-Mead simplex algorithm, see Nelder and Mead [1965], attempts to obtain a local minimizer of a function without the use of derivative information. Although no convergence results have been established for this algorithm, the manner in which the Nelder-Mead simplex algorithm proceeds makes it computationally useful for the minimization of noisy functions, see for example, Glad and Goldstein [1977], and Young [1976]. This algorithm has been used in many areas. In the literature there are examples in physics, Balakrishnan, Gunasekaran, and Gopal [1984], crystallography, Busing and Matsui [1984], biology, Schulze and Rehder [1984], chemistry, Silver [1981], and health care, Sthapit, Ottoway, and Fell [1984]. For a FORTRAN implementation, see Olsson [1974].

The Nelder and Mead version of the simplex algorithm is based on the work of Spendley, Hext and Himsworth [1962] and the work of Box [1955]. The Spendley et al. simplex algorithm uses regular simplices, and Nelder and Mead generalized their procedure to change the shape and size of the simplex. There have been studies into various modifications of the Nelder-Mead simplex...
algorithm, see for example, Parkinson and Hutchinson [1972], and Olsson and Nelson [1976]. There are also variants of the work of Spedle et. al., notably Dambrowski [1970], and Rykov [1980a, 1980b]. Rykov proves convergence of his algorithm to a local minimizer. Rykov's results are discussed briefly in Section 3.5. There has also been work to extend the Nelder-Mead simplex algorithm to non-Euclidean geometries, see Silver [1978, 1981]. However, the Nelder-Mead version of the simplex algorithm is best suited to our needs.

In Section 2.2 we motivate the use of the Nelder-Mead algorithm for our application. Section 2.3 contains a description and an algorithmic statement of the algorithm as presented by Nelder and Mead. In Section 2.4 we discuss various stopping criteria. Preliminary theoretical results for the Nelder-Mead algorithm are presented in Section 2.5.

2.2. Motivation

We use the Nelder-Mead simplex algorithm for several reasons. One reason is the algorithm's robustness with respect to noisy functions. We do not have function values in the quantitative sense, instead, we have ranking information regarding sets of parameters. As will be seen, this ranking information is sufficient to drive the Nelder-Mead simplex algorithm.

A second reason for using the Nelder-Mead algorithm in our particular application, despite its frequently slow convergence, is that we may not need the algorithm to run to completion if we can obtain appropriate weights. With appropriate weights, we can obtain a satisfactory solution of the resulting single-objective optimization problem by applying the library optimizers of MINPACK, see More, Garbow and Hillstrom [1980], which are much faster in practice and theory. This may result in significant savings in obtaining satisfactory values for more than about four parameters, besides the additional information contained in the weights themselves.

A further reason for using the Nelder-Mead algorithm is that we may be unable to generate additional approximations to the solution with the MINPACK subroutines. This can happen if we are unable to copy the user's choice criterion and are, therefore, unable to provide function values to the library routines. It is also possible that we have obtained the minimizer from the library subroutines, which the user determines to be the best so far, but still not satisfactory. The Nelder-Mead simplex algorithm, or one similar to it, will enable us to continue in this case.

Two additional reasons for using the Nelder-Mead algorithm to generate trial points are based on the fact that the algorithm attempts to improve the approximation at each step with a small amount of computational effort. We would like to generate trial points that are improving approximations to the solution so that the user's interest is maintained. If the trial points are generated randomly, then the user may feel that there is little use in continuing the process since there is no trend toward improved approximate solutions. Additionally, we feel that information from ranking the trial points generated
by the Nelder-Mead algorithm will provide more salient information about the user's choice criterion than would ranking of randomly generated points.

One may now suggest generating most of the trial points by using a library optimizer. Experiments with the software system have shown that this leads to the user having a long waiting period for an approximate solution that is not satisfactory. That is, we do not make full use of information the user may provide before solving the single-objective optimization problem. The work involved in finding trial points in this manner, before appropriate weights are known, results in the solution of at least several optimization problems by the library optimizers. The experiments have shown that it is often better to not solve the optimization problem until appropriate weights have been approximated. Additionally, the library routines may not be able to find an improved approximate solution with the current criterion function.

An idea along these lines is to allocate a fixed time to the library optimizer. That is, let the optimizer execute for $p$ seconds and return the best answer after the time period has expired. We still require a method similar to the Nelder-Mead method for the case in which the optimizer can find no better solution in the time allotted. We pursue this idea no further in this thesis.

2.3. Description

The Nelder-Mead simplex algorithm operates by moving an $n$-dimensional simplex through $\mathbb{R}^n$ by accepting one of four operations on the simplex at each iteration. Recall that an $n$-dimensional simplex is defined by $n+1$ points, called vertices, in general position in $\mathbb{R}^n$. General position merely means that no three vertices are colinear. Geometrically, the simplex is defined to be the convex hull of its vertices. For example, in $\mathbb{R}^2$ we have three points that determine a triangle and, in $\mathbb{R}^3$, four points that determine a tetrahedron.

To execute an iteration of the Nelder-Mead simplex algorithm, we need only order information for the vertices. That is, we need to know which vertex is the best approximation to the solution, which vertex is the worst, and which is the second worst. In ordinary function minimization, these would correspond to the vertices with the lowest, the highest, and the second highest function values, respectively. Our notation for the vertices is as follows: a boldface vertex $v_i$ indicates that this vertex has rank $i$ in the current simplex, and a non-boldface $v$ indicates that this vertex has either not been ranked, or else that it is not a current vertex.

Thus, we have at each iteration an ordered list of vertices $<v_0, v_1, \cdots, v_n>$, where $v_0$ is the vertex that is the best approximation to the solution and $v_n$ the worst in the current simplex. The angled brackets $<>$ indicate that the vertices are members of the same simplex. Note that we do not necessarily need to know which vertices are $v_1, v_2, \cdots, v_{n-2}$ to drive the Nelder-Mead simplex algorithm, but in our version of the algorithm, we ask for this additional information in order to reduce the work during subsequent iterations.
The algorithm proceeds by moving the simplex away from the current worst vertex $v_n$. The four operations for moving the simplex are called: reflection, expansion, contraction, and shrinkage. At each iteration, the centroid $\bar{v}$ of the current vertices excluding the worst is calculated. A reflected vertex is computed as

$$v_r = (1+\alpha)\bar{v} - \alpha v_n,$$

where $\alpha > 0$ is the reflection coefficient. We use $\alpha = 1$. If $v_r$ is a better approximation to the solution than $v_{n-1}$ and worse than $v_0$, then $v_r$ replaces $v_n$ as a vertex of the simplex. If the convergence criterion is not met, then it is necessary to rank $v_r$ with respect to the other vertices, and the next iteration begins.

If $v_r$ is a better approximation to the solution than $v_0$, then we want to continue in the direction we are moving, and we compute the expanded vertex defined by

$$v_e = \gamma v_r + (1-\gamma)\bar{v},$$

where $\gamma > 1$ is the expansion coefficient. We use $\gamma = 2$. If $v_e$ is a better approximate solution than $v_0$, but not necessarily better than $v_r$, then the new simplex is defined by replacing $v_n$ with $v_e$, otherwise $v_r$ replaces $v_n$. We use a modified algorithm that accepts $v_e$ only if $v_e$ is a better point than $v_r$. If we have not yet converged, then we can begin the next iteration since we have already obtained the order information for the new vertex.

The operation of contraction occurs when the reflected vertex $v_r$ is a poorer approximation to a solution than $v_{n-1}$. Let $v_l$ be the better approximate solution of the vertices $v_n$ and $v_r$. Then the contracted vertex is given by

$$v_c = \beta v_l + (1-\beta)\bar{v},$$

where $0 < \beta < 1$ is the contraction coefficient. We use $\beta = \frac{1}{2}$. If $v_c$ is better than $v_{n-1}$, then the new simplex is defined by replacing $v_n$ with $v_c$. Again, we check for convergence and rank $v_c$ with respect to the other vertices to begin the next iteration.

If $v_c$ is a poorer approximation to a solution than $v_{n-1}$, then we perform the operation of shrinkage. The shrinkage operation reduces the size of the simplex by moving all but the best vertex halfway towards the best vertex. This is written as

$$v_k = \frac{1}{2}(v_0 + v_k), \quad k = 1, 2, \ldots, n.$$

We must now obtain order information for an entire set $\{v_k\}$ of $n$ new vertices before the next iteration may commence. For most operations, notably reflection, expansion and contraction, the order information for the new vertex is obtained in the decision process. Ordering the vertices after the shrinkage operation requires at most $O(n\log_2 n)$ comparisons, see Aho, Hopcroft and Ullman [1974], or Knuth [1973].

We follow Nelder and Mead in choosing $\alpha = 1, \beta = \frac{1}{2},$ and $\gamma = 2$. As we have noted earlier, there has been research into variants of the Nelder-Mead
algorithm which discuss other choices for \( \alpha, \beta, \) and \( \gamma, \) and which introduce additional parameters into the algorithm. Since no convergence results have been shown for any algorithm that uses only order information, we feel that \((\alpha, \beta, \gamma) = (1, \frac{1}{3}, 2)\) is a perfectly acceptable choice of parameters for a general algorithm and remark that some other sets will be optimal for some specific problems.

Figure 2.1 below, illustrates the Nelder-Mead simplex algorithm on a function defined over \( \mathbb{R}^2. \) The level sets of the function are also displayed to show how the Nelder-Mead simplex algorithm proceeds. In Figure 2.2, we illustrate for the simplex \(<v_0, v_1, v_2> \subset \mathbb{R}^2\) the vertices \(v_r, v_x, v_z, v_s,\) and the shrinkage vertices \(v_z, v_0, v_x, v_s,\) where \(v_r\) is the contraction vertex from \( v_s, \) and \(v_e\) is the contraction vertex from \( v_r.\)

For the moment, we assume we have the initial simplex \( S_0 = <v_0, v_1, \ldots, v_n> \). That is, the vertices are defined as well as the ranking information for the vertices. If two vertices are equally good approximations to a solution, then they should have the same ranking. In function minimization, this occurs when two vertices have the same function value. For reasons that will become clear, we need to make the rankings unique, i.e., no two vertices have the same rank. We arbitrarily choose the order induced by the order in which vertices are obtained. That is, if we have a vertex that is the third best approximate solution and another equally good vertex is obtained later, then the new vertex is given rank four.
We may now write the Nelder-Mead simplex algorithm as follows:

Algorithm A-2.1 Nelder-Mead Simplex Algorithm

Given \( S_0 \) with vertices \(<v_0, v_1, \ldots, v_n>\), set \( \alpha=1, \beta=\frac{1}{2}, \gamma=2 \).

For \( k = 1, 2, \ldots \)

compute \( \overline{v} = \frac{1}{n} \sum_{i=0}^{n-1} v_i \)

compute \( v_r = (1+\alpha)\overline{v} - \alpha v_n \)

\( v_k = v_r \)

if \( v_r \) is better than \( v_{n-1} \) then

if \( v_r \) is better than \( v_0 \) then

compute \( v^* = \gamma v_r + (1-\gamma)\overline{v} \)

if \( v^* \) is better than \( v_0 \) then \( v^* = v_r \)

else set \( v^* = v_n \)

if \( v_r \) is better than \( v^* \) then \( v^* = v_r \)

compute \( v_c = \beta v^* + (1-\beta)\overline{v} \)

if \( v_c \) is better than \( v_{n-1} \) then \( v^* = v_c \)

else \( v_j = \frac{v_0 + v_j}{2} \) for \( j = 1, n-1 \)

\( v_k = \frac{v_0 + v_n}{2} \)

Sort \(<v_0, v_1, \ldots, v_{n-1}, v_n>\) to

obtain \( S_k = <v_0, v_1, \ldots, v_n> \).

2.4. Stopping Criteria

To examine the stopping criterion for the algorithm as stated by Nelder and Mead, we will need to establish some notation. Let the simplex be \(<v_0, v_1, \ldots, v_n>\), and suppose we have function values at each of the vertices, i.e., \( \phi(v_0), \phi(v_1), \ldots, \phi(v_n) \). If we apply the Nelder-Mead simplex algorithm to this function in order to find a minimizer of \( \phi(v) \), then the algorithm is halted when

\[
\frac{1}{n} \sum_{i=0}^{n} (\phi(v_i) - \overline{\phi})^2 < \epsilon,
\]

where

\[
\overline{\phi} = \frac{1}{n} \sum_{i=0}^{n} \phi(v_i),
\]

and \( \epsilon > 0 \) is some preset value. Thus, Nelder and Mead propose stopping the algorithm when the standard error of the function values is less than some preset value.

There are certain cases where the stopping criterion of Nelder and Mead will break down. If all the function values are close enough, then the algorithm will halt. This may occur even when the simplex is very large. In Figure 2.3, for example, the vertices of the initial simplex have the same function value and the algorithm will halt even though one vertex is at a local maximizer of the function.
The simplex need not be large for the algorithm to halt at nonoptimal points. Consider the same problem as defined in Figure 2.3 with the initial simplex given by two arbitrarily close points. Now the simplex is too small and the Nelder-Mead simplex algorithm will halt when the stopping criterion (NM) has $\epsilon > \sum_{i=0}^{1} (\phi(v_i) - \bar{\phi})^2$. This situation is shown in Figure 2.4.

In our application, we have no function values to use in the computation of stopping criterion (NM). Parkinson and Hutchinson [1972] propose other stopping criteria for this algorithm. The most useful of which is related to that of Powell [1964], and is written as

$$\frac{1}{n} \sum_{i=1}^{n} ||v_i^t - v_{i}^{t+1}||^2,$$

(PH)

where $v_i^{t+1}$ is the vertex replacing $v_i^t$ in the algorithm. For all operations except shrinkage, only one term of (PH) is not zero. For shrinkage, each term is the square of the distance that $v_i^t$ moved toward $v_0^t$.

Stopping criterion (PH) is a measure of how far the simplex has moved. This is an interesting idea, but the distance the simplex moves is related to the
size of the simplex. If we use the size of the simplex as a stopping criterion, then our stopping criterion could be the following:

$$\frac{1}{\Delta} \max_{1 \leq i \leq n} \|v_i - v_0\| \leq \epsilon_2,$$

where $\Delta = \max(1, \|v_0\|)$ and $\epsilon_2 > 0$. This measures the relative size of the simplex by considering the length of the longest side incident to $v_0$, and stops when this length becomes smaller than some preset value.

We feel that (SS) is a better stopping criterion than (PH) for several reasons. One reason for our preference is that the value of (PH) for a shrinkage step in the Nelder-Mead algorithm will be greater than the value of (PH) for a contraction step, and we have observed that shrinkage occurs often when the simplex is in a neighborhood of a local minimizer. Upon examination of the properties of the Nelder-Mead algorithm when the various stopping criterion are used, it will be seen that none of these criteria will guarantee convergence of the simplex to a local minimizer or cause the algorithm to terminate in a finite number of iterations.

2.5. Theoretical Results

To establish convergence, we would like to follow the approach due to Wolfe [1969], [1971], and presented in Chapter 14 of Ortega and Rheinboldt [1970]. Ortega and Rheinboldt consider methods of the form

$$v_{k+1} = v_k - \omega_k \alpha_k p_k,$$

for finding critical points of a function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$, where $\alpha \in \mathbb{R}$ is a step length parameter, and $\omega \in \mathbb{R}$ is a relaxation parameter. For our purposes, we may assume $\omega_k = 1$ for all $k$. Additionally, the analysis of Ortega and Rheinboldt requires the method have the property that

$$\phi(v_{k+1}) \leq \phi(v_k), \quad k = 0, 1, \ldots$$

(2.1)

If $\phi$ is bounded below, then we clearly have convergence of the sequence $\{\phi(v_k)\}$.

Of course, this implies nothing about the desired result

$$\lim_{k \to \infty} v_k = v^*,$$

(2.2)

and

$$\nabla \phi(v^*) = 0.$$

(2.3)

The analysis of Wolfe is based on the following propositions:

$$\lim_{k \to \infty} \frac{\nabla \phi(v_k)^T p_k}{\|p_k\|} = 0,$$

(2.4)

and

$$\lim_{k \to \infty} \nabla \phi(v_k) = 0.$$

(2.5)

In Section 2.5.2 we conjecture sufficient conditions for having the iterates of a slightly modified Nelder-Mead simplex algorithm satisfy (2.2), where $v^*$ is a global minimizer.
2.5.1. Sequences Under Consideration

There are three sequences to consider for the Nelder-Mead algorithm. These sequences are: the sequence of best approximate solutions \(\{v^*_0\}\), the sequence of worst approximate solutions \(\{v^*_k\}\), and the sequence of centroids of the simplices, which we denote by \(\{v^*_i\}\). To use the results of Wolfe, we need to establish the non-increase property (2.1) for these sequences. We remark that (2.1) holds for all sequences of iterates \(\{v^*_i\}\) of the Nelder-Mead algorithm except possibly for those obtained through the operation of shrinkage.

It should be obvious that \(\{v^*_0\}\) has property (2.1) since \(v^*_0\) is replaced by \(\hat{v}\) only if \(\phi(\hat{v}) < \phi(v^*_0)\). It is often the case in the Nelder-Mead algorithm that \(v^*_0 = v^*_i\). To show (2.1) for \(\{v^*_k\}\), we need to add a convexity assumption on \(\phi(v)\), i.e.,

\[
\phi(\alpha v_1 + (1-\alpha)v_2) \leq \alpha \phi(v_1) + (1-\alpha) \phi(v_2), \quad \alpha \in [0,1].
\]  

Convexity also provides the inequality

\[
\phi\left(\sum_{i=1}^k \alpha_i v_i\right) \leq \sum_{i=1}^k \alpha_i \phi(v_i),
\]  

when \(\alpha_i \geq 0\) for all \(i\), and \(\sum_{i=1}^k \alpha_i = 1\).

With the additional assumption of convexity of \(\phi\), vertices of any simplex \(S_k\) will have function values less than or equal to \(\phi(v^*_k)\), by (2.7) and (2.6), respectively. If the reflected vertex is accepted, then \(\phi(v^*_i) < \phi(v^*_{i-1}) \leq \phi(v^*_k)\). If the reflected vertex is not accepted, then either the contraction vertex is accepted or the simplex is shrunken and, in all cases, \(\phi(v^*_{i+1}) \leq \phi(v^*_i)\) and (2.1) holds for \(\{v^*_k\}\). This argument also shows that every sequence \(\{v^*_i\}\), \(i = 0, 1, \ldots, n\), has the non-increase property (2.1).

Even with the convexity assumption added, property (2.1) does not necessarily hold for the sequence \(\{v^*_i\}\). This is shown in Figure 2.5 below. The vertices of \(S_k\) are \(v_0\) and \(v_1\) and \(v_i\) is accepted by the Nelder-Mead algorithm. It is obvious by the figure that \(\phi(\frac{v_0 + v_1}{2}) < \phi(\frac{v_0 + v_i}{2})\). So we consider the sequence \(\{v^*_i\}\) no further in the analysis for the Nelder-Mead algorithm.

There is a result that uses the sequence \(\{v^*_i\}\) to obtain convergence of a simplex method. Rykov [1980a], [1980b] uses an adaptation of the algorithm of Spendley et. al. [1982] which makes use of regular simplices. In his method,
Rykov reflects $m_k$ vertices at each iteration, where $m_k$ is chosen so that the centroid $v^*_k$ is moved along the direction that lies closest to $-\nabla \phi(v^*_k)$. It is then necessary to compute $\phi(v^{k+1})$ to test for step acceptance. If the step fails a sufficient decrease condition, then the step is rejected and the simplex is shrunken. If the step is accepted, then it is necessary to compute the function values at the reflected vertices. This enables Rykov to obtain property (2.4) for his steps. Rykov then shows (2.2) and (2.3) under the assumptions that $\phi$ is convex, continuously differentiable, with a Lipschitz condition on the gradient, i.e.,

$$||\nabla \phi(v+h) - \nabla \phi(v)|| \leq L||h||,$$

and all sets of the form $L(v) = \{v: \phi(v) \leq \phi(v)\}$ are bounded.

Rykov uses the function values at the centroid and the vertices of the simplex to obtain an approximation to $\nabla \phi(v^*_k)$. Rykov's algorithm can be considered a gradient-type algorithm. We may not use Rykov's algorithm for our problem since we have no function values at the vertices, only order information. Thus, we have no manner in which to approximate the gradient.

2.5.2. Preliminary Convergence Results

Theorem 2.1, given below, states that our slightly modified version of the Nelder-Mead algorithm will converge to a connected set of limit points under appropriate conditions. We use a relative decrease condition in Theorem 2.1 for the acceptance criterion of $v_r$. We show in Corollary 2.2 that this condition is satisfied when the standard algorithm is implemented on a finite precision machine. We conjecture that Theorem 2.1 remains true when the finite decrease assumption is not used.

**Theorem 2.1**: If the set $L(v^*_n) = \{v: \phi(v) \leq \phi(v^*_n)\}$ is bounded, where $v^*_n$ is the worst vertex of the initial simplex, then a subsequence of the sequence of simplices converges to some $S^* = \langle v^*_1, v^*_2, \ldots, v^*_n \rangle$.

Additionally, let the Nelder-Mead algorithm be modified such that, at the $k^{th}$ iteration, the expansion step $v^*_k$ is accepted only if

$$\phi(v^*_k) < \phi(v^*_k) < \phi(v^*_k),$$

and the reflection step $v^*_k$ is accepted only if

$$\phi(v^*_k) < \min \{ \phi(v^*_k), \phi(v^*_k) (1 - \text{sign}(\phi(v^*_k)) \epsilon)\},$$

for some $\epsilon > 0$ and $\phi(v^*_k) \neq 0$. If $\phi(v^*_k) = 0$, then we accept $v_r$ only if

$$\phi(v_r) < -\epsilon.$$
Proof: We first show that the simplices remain in a compact set. Standard theory for an infinite sequence in a compact set then provides the first result. We then show that for a simplex arbitrarily close to a nondegenerate limit simplex $S^*$, acceptance of any step, e.g. contraction, of the modified Nelder-Mead algorithm results in a contradiction.

Let $D$ be the convex hull of $L(v_0^0)$, i.e.,

$$D = \{z : z = \alpha x + (1-\alpha)y, \ \alpha \in [0,1], \ x,y \in L(v_0^0)\}.$$ 

For the operations of reflection, contraction and expansion, we accept only vertices $v$ such that

$$\phi(v) < \phi(v_0^0) \leq \phi(v_1^1) \leq \phi(v_0^0).$$

Thus, $\langle v_0^0, v_1^1, \cdots, v_n^{k+1} \rangle \subset L(v_0^0) \subset D$. For the operation of shrinkage, the new vertices are obviously elements of $D$ since $D$ is convex. Thus, every simplex of the sequence $\{S_k\}$ is an element of the compact set $D$.

Consider the sequence of best vertices of $S_k$, denoted by $\{v_0^k\}$. By compactness, we have a subsequence of $\{v_0^k\}$ that converges to some point, call it $v_0^*$. Denote this subsequence by $\{v_0^{k_i}\}$. Let $\{S_k\}$ be an associated subsequence of simplices.

Consider the sequence of second best vertices of $\{S_k\}$, denote this sequence by $\{v_1^{k_i}\}$. As before, we have a subsequence of these that converge to some $v_1^*$.

Denote this subsequence by $\{v_1^{k_i}\}$. Note that $v_0^{k_i} \rightarrow v_0^*$.

Repeat this procedure for each ordered vertex $v_2^*, v_3^*, \cdots, v_n^*$. We then obtain a subsequence of the original sequence of simplices, denote this subsequence by $\{S_k\}$, that has the property that

$$v_i^0 \rightarrow v_i^*, \ \text{for} \ i = 0, 1, \cdots, n.$$ 

Thus, the subsequence $\{S_k\}$ converges to $S^* = \langle v_0^*, v_1^*, \cdots, v_n^* \rangle \subset D$, and the first result is verified.

To establish the second proposition, we assume the limit simplex $S^* = \langle v_0^*, v_1^*, \cdots, v_n^* \rangle$ is not a degenerate simplex. That is, at least one vertex is different from $v_0^*$. The centroid of the best $n$ vertices of $S^*$ is defined by

$$v_c^* = \frac{1}{2n} \sum_{i=0}^{n-1} v_i^* + \frac{1}{2} v_n^*.$$ 

and, by strict convexity, we have

$$\phi(v_t^*) < \frac{1}{2n} \sum_{i=0}^{n-1} \phi(v_i^*) + \frac{1}{2} \phi(v_n^*) \leq \phi(v_s^*),$$ 

since $\phi(v_i^*) \leq \phi(v_s^*)$ for $i = 0, 1, \cdots, n-1$. Let

$$\epsilon_c = \frac{\phi(v_s^*) - \phi(v_t^*)}{2},$$

which is greater than zero.

By the continuity of $\phi$, we have that for $\epsilon_c > 0$ there exists a $\delta_c > 0$ such
that for all \( z \) with \( ||z-v_i^*|| < \delta_i \), we have \( |\phi(z) - \phi(v_i^*)| < \epsilon_i \). Let

\[
B_i = \{ z : ||z - v_i^*|| < \delta_i \}.
\]

Thus, for any \( z \in B_i \), we have \( |\phi(z) - \phi(v_i^*)| < \epsilon_i \), which implies \( \phi(z) < \phi(v_i^*) \).

Let \( j \) be the smallest index such that \( \phi(v_j^*) = \phi(v_i^*) \). Note that \( j = 0 \) is possible, but for \( 0 \leq i < j \), we have \( \phi(v_i^*) < \phi(v_j^*) \). For \( 0 \leq i < j \), let \( \epsilon_i = \frac{1}{n} (\phi(v_j^*) - \phi(v_i^*)) \). By continuity, and \( \epsilon_i > 0 \), there exists a \( \delta_i > 0 \) such that for all \( z \) with \( ||z-v_i^*|| < \delta_i \), we have \( |\phi(z) - \phi(v_i^*)| < \epsilon_i \). For \( 0 \leq i < j \), let \( \delta_i = \min \{ \delta_i, \delta_j \} \), and

\[
B_i = \{ z : ||z - v_i^*|| < \delta_i \}.
\]

For \( 0 \leq j \leq i < n \) define

\[
B_i = \{ z : ||z - v_i^*|| < \delta_i \}.
\]

Let \( \epsilon_n = \min \{ \frac{1}{n} \epsilon_i, |\phi(v_n^*)| \} \) for \( \phi(v_n^*) \neq 0 \), and \( \epsilon_n = \min \{ \frac{1}{n} \epsilon_i, \epsilon \} \) for the case \( \phi(v_n^*) = 0 \), where \( \epsilon \) is the positive preset value used to test for an acceptable reflected vertex. By continuity and \( \epsilon_n > 0 \), there exists a \( \delta_n > 0 \) such that for all \( z \) with \( ||z-v_n^*|| < \delta_n \), we have \( |\phi(z) - \phi(v_n^*)| < \epsilon_n \). Let \( \delta_n = \min \{ \delta_n, \delta_j \} \), and

\[
B_n = \{ z : ||z - v_n^*|| < \delta_n \}.
\]

Choose any index \( k \) such that \( S_k \) is an element of the subsequence of simplices that converge to \( S^* \) and \( v_k^* \in B_i \) for \( i = 0, 1, \ldots, n \), i.e., for each \( i \), the \( i^{th} \) best vertex of \( S_k \) is an element of the neighborhood \( B_i \) about \( v_i^* \). By construction of \( B_i \), for \( 0 \leq i < j \), and the fact that \( \{ \phi(v_i^*) \} \) is a non-increasing sequence for all \( i = 0, 1, \ldots, n \), only the \( j \) best vertices of \( S_k \) have function values less than \( \phi(v_j^*) \). We now show that \( S_{k+1} \) will have \( j+1 \) vertices with function values less than \( \phi(v_j^*) \). This implies that \( v_j^* \) is not an accumulation point for the sequence \( \{ v_j^* \} \) since \( \{ \phi(v_j^*) \} \) is a non-increasing sequence with limit point \( \phi(v_j^*) \). This contradiction implies \( S^* \) is a degenerate simplex.

Consider the reflection step for \( S_k \). This step is accepted only if \( \phi(v_i^*) < \phi(v_k^*)(1-\epsilon) \), for \( \phi(v_k^*) \neq 0 \). Because \( v_k^* \in B_n \), we have that

\[
\phi(v_k^*)(1-\epsilon) < \phi(v_k^*) = \phi(v_j^*).
\]

In the case that \( \phi(v_k^*) = 0 \), we accept \( v_i^* \) only if \( \phi(v_i^*) < \epsilon \). By construction of \( B_n \), this provides

\[
\phi(v_k^*) - \epsilon < \phi(v_k^*) = \phi(v_j^*).
\]

Thus, if reflection is accepted at this iteration, then \( \phi(v^*_k) < \phi(v^*_j) \), and we have \( j+1 \) points with function values less than \( \phi(v_j^*) \), a contradiction.

Consider the contraction vertex \( v_i^* \) from \( v_k^* \). We have

\[
v_i^* = \frac{1}{2n} \sum_{i=0}^{n-1} v_i^* + \frac{1}{2} v_k^*.
\]

Subtracting \( v_i^* \) from both sides and using (2.8), we obtain

\[
v_i^* - v_i^* = \frac{1}{2n} \sum_{i=0}^{n-1} (v_i^* - v_i^*) + \frac{1}{2} (v_k^* - v_k^*).
\]

Taking norms and using the triangle inequality provides
\[ \|v^*_t - v^*_s\| \leq \frac{1}{2n} \sum_{i=0}^{n-1} \|v^*_i - v^*_s\| + \frac{1}{2} \|v^*_s - v^*_n\| \leq \frac{1}{2} \epsilon + \frac{1}{2} \epsilon = \epsilon. \]

Thus, if \(v^*_t\) is considered at the \(k^{th}\) iteration, it is accepted and \(\phi(v^*_t) < \phi(v^*_j)\), which provides a contradiction.

Consider the contraction vertex \(v^*_t\) from \(v^*_s\). We may write
\[ v^*_t = \frac{1}{3} v^*_s + \frac{2}{3} v^*_r. \]

Consideration of \(v^*_t\) implies \(\phi(v^*_t) \leq \phi(v^*_s) < \phi(v^*_r)\). Thus,
\[
\phi(v^*_s) < \frac{1}{3} \phi(v^*_s) + \frac{2}{3} \phi(v^*_t) \\
< \frac{1}{3} (\phi(v^*_s) + \epsilon) + \frac{2}{3} \phi(v^*_t) \\
< \frac{1}{3} \phi(v^*_s) + \frac{1}{3} \epsilon + \frac{2}{3} \phi(v^*_r) + \frac{\epsilon}{2} \\
= \frac{1}{3} \phi(v^*_s) + \frac{1}{3} \epsilon + \frac{2}{3} (\phi(v^*_r) + \frac{\epsilon}{2}) \\
\leq \frac{1}{3} \phi(v^*_s) + \frac{1}{3} \epsilon + \frac{2}{3} (\phi(v^*_r) + 2\epsilon + \frac{\epsilon}{2}) \\
= \phi(v^*_s) + 2\epsilon \\
= \phi(v^*_r).
\]

Therefore, if \(v^*_t\) is considered at the \(k^{th}\) iteration, then it is accepted and \(\phi(v^*_t) < \phi(v^*_j)\), a contradiction.

Although \(v^*_t\) may not be accepted by the algorithm without causing a contradiction, it is still possible that \(\phi(v^*_t) < \phi(v^*_0)\), and thus, the expansion vertex \(v^*_t\) is considered at this iteration. We may write
\[ v^*_t = \frac{2}{5} v^*_t + \frac{3}{5} v^*_r, \]
and, by strict convexity, we have
\[
\phi(v^*_t) > \frac{5}{3} \phi(v^*_t) - \frac{2}{3} \phi(v^*_r) \\
= \phi(v^*_t) + \frac{2}{3} (\phi(v^*_r) - \phi(v^*_t)).
\]

If \(\phi(v^*_t) \leq \phi(v^*_r)\) and \(v^*_t\) is accepted, then we have
\[ \phi(v^*_t) < \phi(v^*_t) \leq \phi(v^*_r) < \phi(v^*_j), \]
and we have a contradiction. Otherwise, if \(\phi(v^*_t) > \phi(v^*_r)\), then \(\phi(v^*_t) > \phi(v^*_t)\) and the expansion vertex is not accepted.

We have shown that for \(S_k\) sufficiently close to \(S^*\), we obtain a new vertex that provides a lower function value than \(v^*_j\). Thus, by contradiction, we have \(v^*_j = v^*_k\) for \(j = 1, 2, \ldots, n\).

To show the function values of all limit points are equal, assume there are two limit point \(v^*\) and \(v^{**}\) of the sequence \(\{S_k\}\) that have different function values. Without loss of generality, assume \(\phi(v^*) < \phi(v^{**})\). Let
\[ \epsilon = \frac{\phi(v^{**}) - \phi(v^*)}{3}. \]

By continuity, there exists a \(\delta^*\) such that for \(|z - v^*| < \delta^*\), we have
\[ |\phi(z) - \phi(v^*)| < \epsilon^*. \] For some \( k \), we have

\[ S_k \in B_\epsilon = \{ z : \|z - v^*\| < \epsilon^* \}, \]

and every vertex of \( S_k \) has function value less than \( \phi(v^*) + \epsilon^* \), which is less than \( \phi(v^{**}) \). Since \( \{\phi(v^i)\} \) is a non-increasing sequence for \( i = 0, 1, \ldots, n \), we have that \( v^{**} \) cannot be a limit point of \( \{S_k\} \). Therefore, all limit points of \( \{S_k\} \) have the same function value.

To show the set of limit points is connected, assume the set of limit points is not connected. Let \( V^* \) be the set of limit points of \( \{S_k\} \). Then, since \( V^* \) is not connected, there exists subsets \( T_1 \) and \( T_2 \) of \( V^* \) with the following properties:

\begin{enumerate}
  \item \( T_1 \cap T_2 = \emptyset \),
  \item \( T_1 \cup T_2 = V^* \),
  \item \( T_1 \neq \emptyset \neq T_2 \),
  \item There exists open sets \( O_1 \) and \( O_2 \) such that \( O_1 \cap O_2 = \emptyset \), \( O_1 \subset O_1 \), \( O_2 \subset O_2 \).
\end{enumerate}

It can be shown that there is a finite difference \( \delta_0 \) between elements of the open sets \( O_1 \) and \( O_2 \). Let \( v^* \) be a limit point of \( \{S_k\} \) in \( T_1 \) and \( v^{**} \) be a limit point of \( \{S_k\} \) in \( T_2 \). Additionally, let

\[ \delta_\epsilon = \frac{1}{11} \delta_0. \]

For every limit point \( \Phi^* \) of \( \{S_k\} \), let \( B_\epsilon \) be the corresponding neighborhood defined by

\[ B_\epsilon = \{ z : \|z - v^*\| < \delta_\epsilon \}. \]

We now show that there is some index \( K \) such that for all \( k > K \), we have \( S_k \) is an element of some \( B_\epsilon \). If this is not true, then there is an infinite subsequence of \( \{S_k\} \) which has no elements in any \( B_\epsilon \). By an earlier result of this theorem, we have that this subsequence converges to a degenerate simplex. Thus, for some \( K \) large enough, all elements of this subsequence are in a neighborhood \( B_\epsilon \) about the limit point, i.e., about the degenerate simplex. Thus, the statement regarding the existence of \( K \) is verified.

Suppose \( S_k, k > K \), is in the neighborhood \( B_\epsilon \) for some \( z \in T_1 \). Then, for all \( j > k \), there is no simplex \( S_j \) in the \( B_\epsilon \) neighborhood of \( v^{**} \). To see this, note that \( S_k \in B_\epsilon \) for every \( k > K \) and the step taken in the Nelder-Mead algorithm is bounded by \( 6\delta_\epsilon \). Therefore, the new vertex at the next iteration is at most \( 5\delta_\epsilon \) from \( v^* \). This new vertex may not lie in the \( B_\epsilon \) neighborhood of \( v^{**} \) since the elements of this neighborhood are at least \( 6\delta_\epsilon \) from \( v^* \). Therefore, the new vertex must be a member of \( B_\epsilon \) for some \( \Phi^* \in T_1 \).

Since each simplex \( S_k \) must be in a neighborhood of an element of \( T_1 \), there are no other limit points. That is, \( T_2 \) is empty, and the set of limit points is connected.
COROLLARY 2.2: If the standard condition for acceptance of \( v_r \) is implemented on a finite precision machine, i.e., \( v_r \) is accepted only if \( \phi(v_r) < \phi(v_{n-1}) \), then the conditions of Theorem 2.1 for accepting \( v_r \) hold.

Proof: Let \( -\epsilon \) be the negative number representable on the machine that is closest to zero, and let \( \epsilon \) be the smallest difference between any two different machine representable numbers. Also, let \( f_l(x) \) be the floating point value of \( x \) and define \( \hat{\phi}(v) = f_l(\phi(v)) \).

Consider \( v_1 \) and \( v_2 \) and the condition \( \phi(v_1) < \phi(v_2) \). For \( \hat{\phi}(v_2) \neq 0 \), if \( \hat{\phi}(v_1) < \hat{\phi}(v_2) \), then \( \hat{\phi}(v_1) < \hat{\phi}(v_2) - \frac{\epsilon}{2} \hat{\phi}(v_2) \), and

\[
\hat{\phi}(v_1) < \hat{\phi}(v_2) \left( 1 - \frac{\epsilon}{2 \hat{\phi}(v_2)} \right).
\]

Similarly, for \( \hat{\phi}(v_2) = 0 \), if \( \hat{\phi}(v_1) < \hat{\phi}(v_2) \), then

\[
\hat{\phi}(v_1) < -\epsilon < -\frac{\epsilon}{2 \epsilon}.
\]

and the conditions for accepting \( v_r \) of Theorem 2.1 hold.

We conjecture that the set of limit points defined in Theorem 2.1 is finite, and thus a single point. Also, we believe the limit point to be the minimizer of the function. The analysis is, however, not straightforward and the desired results have not yet been established. As mentioned before, we also conjecture that Theorem 2.1 holds when the finite decrease assumption is not used. However, in view of Corollary 2.2, this result would be of largely academic interest. It is also possible to prove Theorem 2.1 for functions that are convex instead of strictly convex.

To see that some assumptions on the function are necessary to establish convergence to a minimizer, we provide an example where the Nelder-Mead algorithm fails to converge to a local minimizer of a well-behaved function. Displayed in Figure 2.6 is the initial simplex and various level sets of the function. The function is not strictly convex, but the sequence of simplices in this example converges to \( v_0 \). The function is such that shrinkage of the simplex occurs at every iteration. At subsequent iterations, the level sets and current simplex will be scaled down versions of those in Figure 2.6. We remark that the Hessian matrix is singular at \( v_0 \).
CHAPTER 3

Mimicking the User's Choice Criterion

3.1. Introduction

We would like to obtain a set of weights $w_i^* \in \mathbb{R}^m$ such that, for some prescribed criterion function $\phi(x, w)$, we will obtain the same ordering with $\phi(x, w_i^*)$ for the set \{x_0, x_1, \ldots, x_k\} as the user has provided, where $x_1, \ldots, x_k$ are all the points that have been ordered by the user. Then, for a trial point $x_{k+1}$, we may make an educated guess as to the user's ranking for $x_{k+1}$ by finding the elements $x_{i-1}$ and $x_i$ of the partially ordered set such that

$$\phi(x_{i-1}, w_i^*) \leq \phi(x_{k+1}, w_i^*) < \phi(x_i, w_i^*).$$

This mimicking of the user's choice criterion also enables us to transform the multi-objective problem into a single-objective problem, i.e., minimize $\phi(x, w_i^*)$.

In Section 3.2 we derive the linear programming (LP) and quadratic programming (QP) problems, one of which need be solved to obtain $w_i^*$. In Section 3.3 we show how $w_{k-1}^*$ can be updated to $w_k^*$ for the linear programming problem and in Section 3.4 we describe a similar updating technique for the quadratic programming problem. In Section 3.5 we discuss what problems may be encountered in finding $w_i^*$ and how they are dealt with in our approach.
3.2. Finding the Weights

We assume we are given an initial set of weights \( w_0 \in \mathbb{R}^m \), e.g., \( w_0 = (1, 1, \ldots, 1)^T \), and an initial criterion function \( \phi: \mathbb{R}^{n+m} \rightarrow \mathbb{R}^1 \), e.g., weighted least squares. Details regarding the selection of \( w_0 \) and the initial \( \phi \) are discussed at the end of this section. Additionally, we assume the user has provided for some \( p \) parameter pairs \((x_k, z_k)\), \( i = 1, 2, \ldots, p \), that he prefers the approximate solution or, in the case of curve-fitting, the fit, provided by \( x_k \) to the approximate solution, or fit, provided by \( z_k \). We may formulate these preferences as constraints on the weights as

\[
\phi(x_k, w) \leq \phi(z_k, w), \quad i = 1, 2, \ldots, p. \tag{3.1a}
\]

For our purposes, the set of weights that satisfy (3.1a) is too general. Therefore, we introduce the two additional sets of constraints which we call the nonnegativity constraints

\[
w_i \geq 0, \quad i = 1, 2, \ldots, m, \tag{3.1b}
\]

and the normalizing constraint

\[
\sum_{i=1}^{m} w_i = n. \tag{3.1c}
\]

It is clear that if (3.1a) and (3.1b) are consistent, then (3.1a,b) and (3.1c) are consistent. We now define the set of weights that satisfy these constraints.

**Definition 3.1:** We call the weight vector \( w \in \mathbb{R}^m \) \( \phi \)-consistent if \( w \) satisfies equation (3.1a).

**Definition 3.2:** We denote by \( W \) the set of weights that satisfy all of (3.1a,b,c) and we call any \( w \in W \) feasible.

We will discuss the possibility that \( W \) is empty in Section 3.5. In this and the following two sections, we assume \( W \) is nonempty. The set \( W \) will be nonempty as long as the user is consistent in the rankings of the trial solutions with respect to the current criterion function.

As an example of what our constraints may be, let

\[
\phi(x, w) = \sum_{i=1}^{m} w_i \cdot |y_i - y(x, t_i)|^2. \tag{3.2}
\]

This is merely the square of a weighted \( l_1 \) norm of the residuals \( y_i - y(x, t_i) \), \( i = 1, 2, \ldots, m \). Using (3.2) we may write the constraints (3.1a) as

\[
\sum_{i=1}^{m} w_i \cdot |y_i - y(x, t_i)|^2 \leq \sum_{i=1}^{m} w_i \cdot |y_i - y(z, t_i)|^2, \quad i = 1, 2, \ldots, p,
\]

or

\[
\sum_{i=1}^{m} w_i \cdot (y_i - y(x, t_i))^2 - (y_i - y(z, t_i))^2 \geq 0, \quad i = 1, 2, \ldots, p. \tag{3.3}
\]

Since each of the constraints in (3.3) is linear in \( w \), we write (3.3) as

\[
A w \geq 0,
\]

where \( A \in \mathbb{R}^p \times m \) with the \( i \)-th element of \( A \) defined by

\[
(A)_{ij} = (y_i - y(x, t_i))^2 - (y_i - y(z, t_i))^2. \tag{3.4}
\]

We remark that \( W \) is the set of all \( w \in \mathbb{R}^m \) that satisfy the linear constraints.
\[ Aw \geq 0 \]
\[ \sum_{i=1}^{m} w_i = m \]  \hspace{1cm} (3.5)
\[ w_i \geq 0, \quad i = 1, 2, \ldots, m. \]

To select a \( w^*_k \) from \( W \) we need to establish an objective. Our objective will be to change the weights as little as possible at each iteration.

The user may input specific values for the weight of any of the data points. For example, if the user knows that the reliability of the data decreases as \( t \) increases, then he may set the weights accordingly. Since we have no information regarding the relative importance of the criteria, we assume that all the criteria have the same importance and thus assign the value one to all uninitialized weights. We then set any negative weights to zero and renormalize the weight vector to have \( l_1 \) norm \( m \). We call this renormalized weight vector \( w^*_k \). Note that \( w^*_k \) satisfies (3.1b,c) by definition and (3.1a) since the user has made no rankings.

As our objective we ask that \( w^*_k \) be as close to \( w^{*}_{k-1} \) as possible. There are many ways to measure the distance from \( w^*_k \) to \( w^{*}_{k-1} \). We may choose \( w^*_k \) by the \( l_{\infty} \) criterion,

\[ \min_{w \in W} \max_{1 \leq i \leq m} |(w^*_{k-1})_i - w_i|, \]  \hspace{1cm} (3.6)

the \( l_1 \) criterion,

\[ \min_{w \in W} \sum_{i=1}^{m} |(w^*_{k-1})_i - w_i|, \]  \hspace{1cm} (3.7)

or the \( l_2 \) criterion,

\[ \min_{w \in W} \sum_{i=1}^{m} |(w^*_{k-1})_i - w_i|^2, \]  \hspace{1cm} (3.8)

where \( (w^*_{k-1})_i \) is the \( i \)th element of \( w^*_{k-1} \). These result in linear programming problems for (3.6) and (3.7) and a quadratic programming problem for (3.8).

In our system we initialize the objective to the \( l_1 \) criterion as in (3.7). The criterion function that defines the matrix \( A \) of (3.5) is initialized to be the square of the \( l_2 \) norm, so \( A \) is defined as in (3.4). We may also use the \( l_1 \) norm, the \( l_{\infty} \) norm, or a Huber M-estimator, see Huber [1972] to define the matrix \( A \). These additional methods for defining \( A \) are available in our software system at the user's request.

### 3.3. Exploiting Structure in the LP Problem

As noted in the introduction, it may be necessary to find \( w^*_k \) for many values of \( k \). Instead of resolving the LP or QP problem from scratch, we may take advantage of certain properties of these problems. We remark that we are still assuming that \( W \) is nonempty for all \( k \).

Suppose we are choosing \( w^*_k \) by the \( l_1 \) criterion as defined in (3.7) and that we have \( w^*_{k-1} \) available. Thus, we have the solution to the \textit{primal linear program} (P) defined by
\[ \min \sum_{i=1}^{m} |(w_{i-1}^*) - w_i| \]
\[ \text{s.t. } A_{k-1} w \geq 0 \]
\[ \sum_{i=1}^{m} w_i = m \]
\[ w_i \geq 0 \quad i = 1, 2, \ldots, m, \]

where \( A_{k-1} \) is the matrix defined by the ranking information up to and including the \( k-1 \)th user ranking. The user now ranks a new point \( z_k \) by inserting it into the ordered list of vertices at an appropriate position. This introduces one or two new constraints depending upon the insertion location. If \( w_{k-1}^* \) satisfies these additional constraints, then \( w_k^* = w_{k-1}^* \), and we have the updated set of weights.

However, if \( w_{k-1}^* \) does not satisfy all of the additional constraints, then we update \( w_{k-1}^* \) to \( w_k^* \), which is a solution to the updated linear programming problem defined by

\[ \min \sum_{i=1}^{m} |(w_{i-1}^*) - w_i| \]
\[ \text{s.t. } A_k w \geq 0 \]
\[ \sum_{i=1}^{m} w_i = m \]
\[ w_i \geq 0 \quad i = 1, 2, \ldots, m, \]

where \( A_k \) is \( A_{k-1} \) with one or two added constraint rows that result from the \( k \)th user ranking. Theoretically, we need to add only one row to \( A_{k-1} \). This is shown in the following theorem.

**Theorem 3.1:** The matrix \( A_k \) in problem \((P^+)\), obtained from the user’s ranking of the points \( z_0 \) through \( z_k \), may be represented using \( k \) rows.

**Proof:** The proof is by induction. For the zero case we note that no rankings have been made and \( A_0 \) has zero rows since no constraints have been defined.

Assume \( A_{k-1} \in \mathbb{R}^{(k-1) \times m} \). The next point to be ranked, call it \( z_k \), may be inserted at the top of the ordered list, at the bottom of the ordered list, or somewhere between the first and last elements of the ordered list. Let \( \hat{z} \) denote the top element of the list, and \( \underline{z} \) denote the bottom element.

If \( z_k \) is inserted at the top of the list, then we have the single new constraint obtained from \((3.3)\) with \( z_k \) and \( \hat{z} \). Similarly, if \( z_k \) is inserted at the bottom of the ordered list, then we have the single new constraint obtained from \((3.3)\) with \( z_k \) and \( \underline{z} \).

Suppose \( z_k \) is inserted in the list between the adjacent elements \( z_{i-1} \) and \( z_i \). Then we introduce the two new constraints obtained from \((3.3)\) with \( z_k \) and \( z_{i-1} \) and with \( z_k \) and \( z_i \). However, we may delete the constraint row in \( A_{k-1} \) obtained from \((3.3)\) with \( z_i \) and \( z_{i-1} \) since

\[ \phi(z_{i-1}, w) \leq \phi(z_k, w) \leq \phi(z_i, w) \]

implies

\[ \phi(z_{i-1}, w) \leq \phi(z_i, w). \]
Thus, in all cases, $A_k$ can be represented by $k$ rows and $A_k \in \mathbb{R}^{k \times m}$.

We could ignore the similarities between problems (P) and (P+) and solve problem (P+) by employing a linear programming package. Using $w^*_k$ as the initial guess would cause the package to execute a Phase I procedure to obtain a feasible point since $w^*_k$ is nonfeasible. After Phase I obtains a feasible point (we have assumed $W$ nonempty), the package would then execute a Phase II procedure to find $w^*_k$. This may require a great deal of computation since Phase I does not consider the objective function in obtaining feasibility and may provide a poor initial feasible point for Phase II. We remark that Karmarkar's algorithm for linear programming, Karmarkar [1983], also requires an initial feasible point.

An alternative approach is to use the dual simplex algorithm for linear programming as introduced by Lemke [1954] and presented in Murty [1976], or Ignizio [1982]. Computationally, this algorithm uses the primal formulation of the problem but proceeds based on considering variables and values associated with the dual problem. This algorithm requires a basic feasible point for the dual problem to begin execution. We now show how a basic feasible point for the dual of the updated problem (P+) can be constructed from the solution to the primal problem (P).

Suppose that our primal problem (P) is written as

$$
\begin{align*}
\min & \quad c^T x \\
\text{s.t.} & \quad A^T x = \beta \\
& \quad A x \geq b \\
& \quad z_i \geq 0, \quad i = 1, 2, \ldots, m,
\end{align*}
$$

(OD)

where $a, c, x, \beta, b \in \mathbb{R}^m$, $\beta \in \mathbb{R}^1$, $b \in \mathbb{R}^{k-1}$, and $A \in \mathbb{R}^{(k-1) \times m}$, with the $z_i$'s the unknowns. Standard techniques are used to transform problems (3.8) and (3.7) into this form. We refer to problem (OD) as the original primal problem. Its dual, which we call the original dual problem (OD), may be written as

$$
\begin{align*}
\max & \quad \lambda^T \beta + \lambda^T b \\
\text{s.t.} & \quad \lambda^T a + A^T \lambda \leq c \\
& \quad \lambda \text{ unrestricted} \\
& \quad A_i \geq 0, \quad i = 1, 2, \ldots, k-1,
\end{align*}
$$

(OD)

where $\lambda \in \mathbb{R}^1$, and $A \in \mathbb{R}^{k-1}$ are the dual variables.

We have the solution $w^*_k$ of problem (OP) available to us. From $w^*_k$ we may construct the solution $(\lambda^*_k, (A^*_k)^T)^T$ to problem (OD). For details of this construction see Chapter 8 of Ignizio [1982]. For simplicity, we assume the user provides a ranking that produces only one additional inequality constraint. The process for handling more than one additional constraint is a direct extension of the process we describe here. We will refer to problem (OP) with the added inequality constraint as the extended primal problem (EP) and its dual will be called the extended dual problem (ED).

To obtain a set of basic variables for the extended primal, and thus the extended dual, we simply add a slack variable to the new constraint. Our new
equality constraint in (EP) is now of the form

$$a_{+}^T \mathbf{z} + s_1 = \beta_+,$$

(3.9)

where $a_{+} \in \mathbb{R}^m$ is determined by the ranking information and $\beta_+ = 0$. Our basic vector for the extended primal is the basic vector of the solution for (OP) with $s_1$ appended. Computationally, pricing out of (3.9) must be executed to restore the basic vector of (OP), but it is clear that (3.9) is not a linear combination of the other constraints since $s_1$ occurs only in (3.9). The basic vector for the extended dual problem may now be obtained from the basic vector of the extended primal.

To construct a feasible point for problem (ED) we note that (3.9) is an equality constraint in the extended primal. Thus, the extended dual may be written as

$$\begin{align*}
\text{max} & \quad \lambda \beta + A^T b + \lambda_+ \beta_+ \\
\text{s.t.} & \quad \lambda a + A^T \mathbf{A} + \lambda_+ a_+ \leq c \\
& \lambda, \lambda_+ \text{ unrestricted} \\
& \lambda_i \geq 0, \quad i = 1, 2, \cdots, \bar{k} - 1,
\end{align*}$$

(ED)

where $\lambda_+$ is the dual variable associated with (3.9) of (EP) and is unrestricted in sign. We now construct a feasible point for (ED).

**Theorem 3.2:** $(\lambda, A^T, \lambda_+)^T = (\lambda_{k-1}^*, (A_{k-1}^*)^T, 0)^T$ is a feasible point for the extended dual problem (ED).

**Proof:** Since $\lambda_{k-1}^*$ and $A_{k-1}^*$ form a solution to problem (OD) we have that $(A_{k-1}^*)_i \geq 0, \quad i = 1, \cdots, \bar{k} - 1$. Noting that a variable being unrestricted is not a constraint, we have only to satisfy

$$\lambda a + A^T \mathbf{A} + \lambda_+ a_+ \leq c.$$

Substituting our proposed solution yields

$$\lambda_{k-1}^* a + A^T (A_{k-1}^*) + 0 a_+ = \lambda_{k-1}^* a + A^T A_{k-1}^* \geq c,$$

from $(\lambda_{k-1}^*, (A_{k-1}^*)^T)^T$ being a solution to problem (OD). Thus, $(\lambda, A^T, \lambda_+)^T$ is a feasible point for problem (ED).

This basic feasible point enables us to initiate the dual simplex method on problem (ED) and avoid the extra computation needed to obtain a feasible point for the extended primal problem. Also, since we have constructed this basic feasible point from the solution to a very similar problem, we have a good initial point for Phase II on the extended dual problem.

### 3.4. Exploiting Structure in the QP Problem

We now consider the quadratic programming problem (3.8). To simplify our presentation, we manipulate the objective function to have a more general form. The objective function of (3.8) is

$$\sum_{i=1}^{m} (w_{i-1}^*) - w_i)^2.$$
which may be written as
\[(w_{k-1}^*-w)^T(w_{k-1}^*-w),\]
or as
\[w^Tw - 2w_{k-1}^*^Tw_{k-1}^*T - w_{k-1}^*^T w_{k-1}^*.\]  
(3.10)

We now multiply (3.10) by \(\frac{1}{2}\), which does not change our optimizer \(w_k^*\), and we drop the constant term \(-w_{k-1}^*^Tw_{k-1}^*,\) since we have no control over this value.

Our objective is now written as
\[\min_{w \in W} \frac{1}{2} w^Tw - w^Tw_{k-1}^*.\]  
(3.11)

Problem (3.11) has two useful properties. The first is that the objective function is uniformly convex. This is easily seen by noting the Hessian matrix is the identity matrix for all \(w \in \mathbb{R}^m\). For the remainder of this section we assume the \(k^{th}\) ranking produces only one new inequality constraint and remark that the procedure for handling more than one additional constraint is a direct extension of the procedure presented here.

We will refer to problem (3.11) with the added inequality constraint
\[a_i^Tw \leq \beta_i,\]  
(3.12)
as the extended QP problem. The second useful property of problem (3.11) with the constraint (3.12) added, is that we have the solution to a very similar problem.

If \(w_{k-1}^*\) satisfies (3.12), then \(w_k^* = w_{k-1}^*\), otherwise we must solve the extended QP problem. By ignoring the similarities between successive problems,

we may use a library routine to find \(w_k^*\). There exist library routines for the solution of QP problems, notably those in the Harwell library, and QPSOL of the SOL/QPSOL library, see Gill, Murray, Saunders, and Wright [1983]. Similar to Phase II of the simplex method of linear programming, these routines require a feasible point to begin execution. To obtain a feasible point the QPSOL routines execute a Phase I linear programming procedure that ignores the quadratic objective function. Once a feasible point has been obtained, a projected gradient technique is used to find \(w_k^*\), see Gill, Murray and Wright [1981].

The library routines for the solution of the extended QP problem may perform a great deal of computation to produce \(w_k^*\) since \(w_{k-1}^*\), the obvious choice as the initial guess, is nonfeasible. Thus, a Phase I procedure must be performed to obtain a feasible point, which may result in a poor initial feasible point for the QP routine. Therefore, we take an approach similar to that described in Section 3.3 for the extended LP problem.

Two approaches for the extended QP problem come immediately to mind. The first is to form a linear objective function such that \(w_{k-1}^*\) is the solution to the linear programming problem formed by using the linear objective function and the linear constraints of the original QP problem. A linear objective function with this property is easily obtained. Then, once the dual basic feasible solution corresponding to \(w_{k-1}^*\) is formed, the dual simplex method may be employed as described in Section 3.3 on the dual of the linear problem.
formed by using the newly defined linear objective function and the linear constraints of the extended QP problem.

Unfortunately, this technique is only guaranteed to produce a feasible point for the extended QP problem. Thus, it is still necessary to use the QP library routine to solve the extended QP problem. It would be of interest to see if any relationship exists between the feasible point obtained in this manner and the feasible point obtained from Phase I. However, we do not explore that question in this thesis.

The second approach we consider is to form the dual of the extended QP problem and, if possible, to solve this extended dual problem. We may write the dual of the original QP problem as

$$\max_{\lambda, \Lambda} \min_{\lambda \in \mathbb{R}^n} \left\{ \frac{1}{2} w^T w - w^T \Lambda w_{\lambda, \lambda+} + \lambda \left( \sum_{i=1}^{m} w_i - m \right) + \lambda^T A_{\lambda, \lambda+} w + \lambda_+ a^T w \right\}$$

s.t. $\Lambda \geq 0$, $\lambda$ unrestricted

where $a_+$ is determined by the ranking of $\nu_k$.

The original QP problem has solution $w_{\lambda, \lambda+}$. From this solution we may construct a solution to (DQP), i.e., $(\lambda^{*, \lambda+}, (A^{*, \lambda+})^T)^T$. For details of this construction see Chapter 10 of Fletcher [1981]. A feasible point for problem (EDQP) is given by $(\lambda^{*, \lambda+}, (A^{*, \lambda+})^T, 0)^T$. Thus, we may use a quadratic programming procedure to obtain a solution to problem (EDQP). We may then form the associated solution of the extended primal quadratic problem, i.e., $w^{*, \lambda+}$.

If $(\lambda^{*, \lambda+}, (A^{*, \lambda+})^T, 0)^T$ is used as the initial point for problem (EDQP) in the quadratic programming procedure, then a solution to problem (EDQP) should be rapidly obtained. This is due to the fact that we have provided an initial feasible point that is optimal with respect to all but one parameter. Chapter 10 of Fletcher discusses methods that take advantage of the special structure of the QP problem. The simple bounds on the variables $\Lambda$ is the exploitable structure for problem (EDQP).

If the dual problem becomes too large to be solved by the procedure described above, then an active set strategy can be performed on the extended primal problem. Suppose that the active constraints at the solution were
known. Then we could write our problem, in a general form, as

$$\min f(w)$$
$$\text{s.t. } h(w) = 0,$$  \hspace{1cm} (EQ)

where $f: \mathbb{R}^n \to \mathbb{R}^1$ and $h: \mathbb{R}^n \to \mathbb{R}^p$, where $p$ is the number of linearly independent constraints active at the solution. The following discussion is based on the work of Tapia [1977] and Luenberger [1973].

Problem (EQ) has the Lagrangian function given by

$$l(w, \lambda) = f(w) + h(w)^T \lambda,$$

where $\lambda \in \mathbb{R}^m$ is a vector of Lagrange multipliers. For an excellent discussion of multiplier methods, see Tapia [1977]. The dual problem of problem (EQ) is given by

$$\max \ g(\lambda) = \max_{\lambda} \left[ \min_w l(w, \lambda) \right].$$  \hspace{1cm} (DP)

For our problem,

$$\min \ \frac{1}{2} w^T w - w^T w^*_1$$
$$\text{s.t. } h(w) = Hw = 0,$$  \hspace{1cm} (QPE)

which has a uniformly convex objective function and linear constraints. The theory states that, if we can find a solution $\lambda^*$ to problem (DP), then it is the global solution and the associated $w^*$ solves problem (EQ). We now show how $\lambda^*$ is obtained.

**Theorem 3.3:** If $H$ has full row rank, then the solution of the dual of problem (QPE) is given by

$$\lambda^* = (HH^T)^{-1} H w^*_1.$$

**Proof:** Consider the subproblem $\min_w l(w, \lambda)$. We have

$$\nabla_w l(w, \lambda) = w - w^*_1 + H^T \lambda.$$

Setting this to zero yields

$$w^* = w^*_1 - H^T \lambda.$$  \hspace{1cm} (3.13)

This implies $w^*$ is the global solution for the subproblem since $\nabla_w^2 l(w, \lambda) = I$, for all $w \in \mathbb{R}^m$.

Substituting $w^*$ into the dual problem of (QPE) yields

$$g(\lambda) = \frac{1}{2}(w^*_1 - H^T \lambda)^T (w^*_1 - H^T \lambda) - (w^*_1 - H^T \lambda)^T w^*_1 + \lambda^T H(w^*_1 - H^T \lambda),$$

with

$$\nabla g(\lambda) = -(HH^T) \lambda - H w^*_1,$$

and with

$$\nabla^2 g(\lambda) = -(HH^T).$$

Setting $\nabla g(\lambda) = 0$ provides

$$\lambda^* = (HH^T)^{-1} H w^*_1.$$
and $H$ being nonsingular implies $-(HH^T)$ is negative definite. Thus, $\lambda^* = \max_\lambda g(\lambda)$. \hfill \Box

Therefore, all we need do is select a full set of nonredundant constraints that are active at the solution. Active set strategies are discussed in Fletcher [1981] and in Gill, Murray, and Wright [1981]. In the situation that the dual method cannot be employed due to insufficient storage capacity, the active set strategy for the primal problem is reasonable and, if properly implemented, efficient.

Some advantage may be taken in this active set strategy of the fact that we are solving a sequence of related problems. It may be possible to choose the correct set of active constraints in a small number of iterations. For example, we know that the normalizing constraint (3.1c), and the newly acquired constraint (3.9) are active at $w^*_k$. However, since the dual problem formulation is available, we study this approach no further in this thesis.

3.5. No Feasible Weights

There is one major problem with the mimicking of the user's choice criterion as we have presented it here. It is possible that the user may make rankings such that $W = \{\}$. We call the set of constraints that cause $W = \{\}$ inconsistent. We need to be able to detect this situation and to deal with it in some reasonable manner.

In the linear programming problem and the quadratic programming problem, we detect the inconsistency of the constraints of the primal by discovering that the associated dual problem is unbounded. Fortunately, the dual simplex method for the LP problem is able to detect this situation. In fact, the detection of an unbounded dual problem should occur fairly quickly since we have introduced the single variable that will make the solution unbounded and we have optimized with respect to the remaining variables. The detection of unboundedness by the dual method for the QP updating procedure is similar; unboundedness can be detected by the algorithms, and should be detected fairly quickly.

In the active set procedure for the primal extended problem, infeasibility implies that $H$ will always be nonsingular in the dual, or that $w^*_k$ will violate at least one constraint not in the active set, where $w^*_k$ is the vector of trial weights for a specific active set. Thus, if we have an active set and a constraint is violated, then the constraint is added to the active set. This causes $H$ to become singular or at least one more constraint is violated by the new $w^*_k$. If $H$ becomes singular, then it is important to recognize that there are no redundant constraints, and thus no feasible set for the primal. Therefore, the work of Telgen [1979], [1981], Karwan, Lotfi, Telgen, and Zionts [1983], and Thompson, Tonge and Zionts [1966], in which redundant constraints are identified, becomes important for this procedure. Their work is based on the work of Boot [1962], which considers redundant and nonredundant constraints. If the methods for
recognizing redundancies and nonredundancies indicate that the primal has no feasible region, then we can verify it using the Phase I procedure on the primal. This is not necessary if the methods show that H is nonsingular and no constraints are redundant.

There are several possible causes for $W = \emptyset$. We have been assuming that we should be using an $l_2$ norm to define $A$, see (3.4). It is possible that the $l_1$, or $l_\infty$ norm, or a Huber measure will provide an $A$ that produces a nonempty $W$. Another possibility is that the user has been inconsistent in his rankings with respect to the current criterion function.

At this point in our approach there are several options available to us. One option is to ask the user to rerank the points that have produced the inconsistent constraints. A second option is to search for a choice criterion that produces a nonempty $W$. Because of its simplicity, we first ask the user to rerank points whose ordering produced inconsistent constraints. To do this we must first identify these inconsistent constraints. The following theorem is a first step in this identification.

**THEOREM 3.4**: If $w^*_k$ exists and the problem defined by the addition of the constraints for the ranking of $x_k$ has no feasible set, then exactly one of the constraints introduced by the ranking of $x_k$ is not satisfied by $w^*_{k-1}$.

**Proof**: The proof is a direct result of Theorem 3.1.

In our approach we identify which of the new constraints is not satisfied for $w^*_{k-1}$. We then ask the user if we may invert the ordering for the points that produced this constraint. If the user allows this inversion, then we attempt again to obtain a $w^*_k$. Note that $W$ may still be empty, but caused by a different set of constraints. If this occurs, then we repeat the identification-inversion process.

If the user will not allow an inversion of a pair of points, one of which was just entered into the partially ordered set, then we need to identify the set of constraints that are inconsistent with the new constraint. These may be provided by the dual procedure for finding $w^*_k$. If they are not, then a subset of these constraints may be found by solving the linear programming problem given by

$$\begin{align*}
\max & \quad a^T w \\
\text{s.t.} & \quad A_{k-1} w \geq 0 \\
& \quad \sum_{i=1}^{m} w_i = m \\
& \quad w_i \geq 0 \quad i = 1, 2, \ldots, m.
\end{align*}$$

The objective function at the solution of problem (i) will be less than zero since $a^T w \geq 0$ is inconsistent with the constraints. A subset of the constraints that are inconsistent with $a^T w \geq 0$ is given by those inequality constraints that are binding at the solution of problem (i).

Once a subset of the inconsistent constraints has been isolated, we repeat the inversion process. To find $w^*_k$, it may be necessary to solve additional linear
programming problems of the form (1) since we find only a subset of the inconsistent constraints. If the user refuses to allow a sufficient amount of reordering to produce \( W \not= \{ \}, \) then we must consider other options for defining \( w_i^{(1)} \).

One such option is to consider other criterion functions \( \phi \) from a preselected set in order to find one for which \( W \not= \{ \}. \) In the following example, the \( l_1 \) criterion has a nonempty feasible set, but the \( l_2 \) criterion does not.

EXAMPLE 3.1: Suppose we have ranking information which provides the following constraints for the \( l_1 \) criterion:

\[
(100-101)w_1 + (20-19)w_2 \geq 0,
\]

and

\[
(3.5-1.5)w_1 + (0.5-1.5)w_2 \geq 0.
\]

We have written the constraints in the form described by (3.3), except we are using the \( l_1 \) criterion instead of the \( l_2 \) criterion. The form for the \( l_2 \) criterion is

\[
(100^2-101^2)w_1 + (20^2-19^2)w_2 \geq 0,
\]

and

\[
(3.5^2-1.5^2)w_1 + (0.5^2-1.5^2)w_2 \geq 0.
\]

Figure 3.1, given below, shows clearly that the feasible region for the constraints (3.14a,b), with the nonnegativity and normalizing constraints added, is the line segment connecting \((1,1)\) with \((2/3,4/3)\), and that there is no feasible set for constraints (3.15a,b) with the nonnegativity and normalizing constraints.

To find a criterion function that produces a feasible set may be a long process, since for each criterion function \( \phi \), we must define the associated matrix \( A \) and check for nonempty \( W \). The forming of \( A \) requires computing all \( p \times m \) elements of the matrix. The checking for nonempty \( W \) requires a Phase I procedure for each criterion function. Since this may require a great deal of computation, we present the situation to the user and ask the user if an attempt to find such a criterion function should be made. If the user allows the search for \( \phi \), then we consider the criterion functions that are members of our preselected set.

If we are unable to find a criterion function \( \phi \) that produces a nonempty \( W \), or, if the user requests that we not attempt to do so, then we still have various
options left to us. We present these options only briefly, since the second and third are not yet available in our software system. The options that we consider are:

1. Continue with the user providing the ranking information to drive the Nelder-Mead simplex algorithm.

2. Choose $w_i^*$ by a least squares criterion.

3. Use "fuzzy" constraints to define $W$.

In the fuzzy constraint approach we ask the $i^{th}$ constraint be satisfied to some tolerance $\epsilon_i$. By choosing appropriate values for the $\epsilon_i$'s we can force $W$ to be nonempty. We may then choose $w_i^*$ from $W$ as before.

The second alternative approach is to minimize the amount of infeasibility subject to various constraints. Suppose the user designates a set of constraints $A_1$ as those constraints that must be satisfied. If we denote the remaining constraints by $A_2$, then $w_i^*$ is defined by

$$\min \| A_2w \|$$

s.t. $\sum_{i=1}^m w_i = m$

$A_1w \geq 0$

$w_i \geq 0, \quad i = 1, \ldots, m$.

Again, by selecting different norms for $\| A_2w \|$ we will obtain different procedures. It is possible with this method to produce a set of constraints that has no feasible set. This method is related to the goal programming approach, mentioned in Chapter 1. We do not explore this procedure any further in this thesis.

Lastly, we remark that $w_i^*$ is needed only to transform the multi-objective optimization problem into a single-objective optimization problem. If $w_i^*$ does not exist, then we may continue improving the approximate solution via the Nelder-Mead simplex algorithm applied to the user's rankings. This results in a method for the multi-objective problem that is strictly an interactive method.
CHAPTER 4

Curve Tailoring

4.1. Introduction

We have produced a software system for the 1-dimensional curve-fitting problem to test our ideas for multi-objective optimization. We have based the design of our system on the approach for multi-objective optimization problems presented in this thesis. That is, we attempt to find a criterion function \( \phi(x,w) \) and a set of weights \( w^* \) such that we may mimic the user's choice criterion. This enables us to transform the multi-objective optimization problem of making all the residuals small, into the single-objective problem of optimizing with respect to \( \phi(x,w^*) \).

To obtain information regarding the user's choice criterion, we generate approximate solutions \( x_k, k=1, 2, \ldots \), to be ranked by the user. We would like these \( x_k \) to be improved approximations to the solution. As remarked in Chapter 1, the reasons for this are threefold: we wish to maintain the user's interest, we wish to provide points for which the ranking information is salient for the defining of the single objective problem, and because the library optimizers may be unable to provide additional points that exhibit improvement. The Nelder-Mead simplex algorithm, discussed in Chapter 2, attempts to accomplish these objectives and compute the \( x_k \)'s quickly.

Whenever possible, we have used library software routines that are available to us. The three subroutine libraries we rely upon are LINPACK, see Dongarra, Moler, Bunch and Stewart [1979], for certain linear algebraic manipulations, MINPACK, see More', Garbow, and Hillstrom [1980], for obtaining the solution of the single-objective optimization problem, and QPSOL, see Gill, Murray, Saunders, and Wright [1983], for the obtaining of the weight vector \( w^* \). We remark that our system is only a prototype system and many techniques that can be used to speed execution have not been employed. For example, we have not yet implemented the weight updating techniques discussed in Chapter 3.

4.2. What is Required From the User

We have written the user interface so that the user is in control of the procedure for solving the curve-fitting problem. The user decides how trial points are generated, and how and where these trial points are entered into the partially ordered set by making selections from menus. The user makes the ultimate decision of when a proposed solution is satisfactory.

From the user we require the data points, \((t_i,y_i), i=1, 2, \ldots, m,\) and a FORTRAN subroutine or C procedure for the evaluation of the model \( y(x,t) \) for a given \( x \in \mathbb{R}^n \) and \( t \in \mathbb{R}^1 \). We also ask the user to provide an initial guess \( x_0 \) of the solution, but it is not necessary that he do so. We allow the user to define
any or all of the initial weights \( w_i \) but again, it is not necessary that he do so. Our last requirement from the user is that he provide the ranking information for each trial solution.

Displayed in Figure 4.1 is a typical view of the screen during the running of our system. The characters (A) through (H) are labels that are not visible on the screen but are added in Figure 4.1 so that we may refer to regions of the graphics screen. The labeled regions of Figure 4.1 are:

- **(A)** The main menu: This contains menu items: HELP, QUIT, SAVE, USER, AUTOMATIC, REDRAW, RESET, and SCREEN.
- **(B)** The message/prompt region: One line messages and prompts are printed in this region to inform and prompt the user.
- **(C)** The work area: Data and prospective solution curves are displayed here. Also, this area contains multi-lined messages when necessary.
- **(D)** The utilities menu: contains utilities to aid a user in ranking \( z_k \). Menu items are: ZOOM IN, ZOOM OUT, OOPS, DISPLAY, UN-RANK, NO DIFFERENCE, and CHANGE DISPLAY.
- **(E)** Plot icons: each icon (box) is a placeholder for the elements of the ordered list of vertices. Associated with each box is a set of parameters and the plot generated by the model with the set of parameters.
- **(F)** Icon divider: Any icon above this line has been ranked by the user or by an automatic procedure. Any icon below the line, there will be at most one at any time, has not been ranked.
- **(G)** Unranked plot icon: The unranked icon. A solid icon means the corresponding plot is visible in the work area.

![Figure 4.1](image)

**Figure 4.1**

A typical curve tailoring screen with labels (A)-(H) added.
The mouse: a pointing device. If the pointing device is placed in a ranked icon, then the icon becomes solid and the corresponding plot is visible in the work area. The mouse is used to make selections from the menus and to indicate insertion locations for the unranked plots.

See Appendix A for a detailed description of the menu items and what operations they perform.

In our system the user selects an item from a menu by placing the mouse in the box around the item and pushes a mouse button. A user ranks an unranked plot by positioning the mouse at the position in the ordered list of icons where the unranked icon should be placed and pushes a mouse button.

4.3. The System

We remark that this system is not intended to be used to establish the correctness of a specific parametric model as the exact model an underlying physical process, but solely as an aid in fitting a given model to given data. At the conclusion of a session with our system, we provide the set of parameters that generate the best approximate solution and, when possible, a weight for each data point that has some significance in the optimality criterion used to produce the best fit.

The system follows the procedure outlined in Algorithm 4.1 below. We discuss step (1) of Algorithm 4.1 in Section 4.4. The remainder of this section discusses steps (2)-(5).

Algorithm 4.1: Curve Tailoring

Given $x_0 \in \mathbb{R}^n$, $(t_i, y_i)$, $i = 1, 2, \ldots, m$ and $y(x,t)$,

1. Initialize the simplex $S_0$ from $x_0$.

For $k = 1, 2, \ldots$, until 'user satisfaction'

\begin{enumerate}
\item generate a trial point $x_k$
\item rank $x_k$
\item update $w_{k-1} \rightarrow w_k$
\item update $S_k \rightarrow S_{k+1}$, if possible
\end{enumerate}

We refer to the two procedures for the ranking of $x_k$ as the user ranking procedure and the automatic ranking procedure. In the user ranking procedure, the user enters $x_k$ into the ordered list of vertices in an appropriate position. To
do this the user compares the fit of $z_2$ with the fits of the elements of the ordered list. The user then ranks $z_2$ by positioning the mouse and pushing a button.

In the automatic procedure, the trial points are ranked by evaluating $\phi(x, w^*)$. Since we have no user interaction, we solve the single-objective optimization problem and return when we have obtained a $x^*_{k+1}$, i.e., the best possible point for the current criterion function with the current weights. Since we can produce no better estimate with our current criterion, we reenter the user ranking procedure with $x^*_{k+1}$ as $z_2$.

For step (4), the weight selection is initialized to the $l_1$ criterion presented in Chapter 3. Thus, the weights can be updated after each user ranking by the dual simplex algorithm of Section 3.3. However, in our prototype implementation, we have not used the updating procedure for the weights and must resolve the associated problem whenever the weights are desired. If there are no weights that satisfy the constraints provided by the user's rankings, then we proceed as discussed in Section 3.5. If it becomes clear that we are unable to mimic the user's choices, then we halt the updating of the weights.

The simplex $S_k$ is updated to $S_{k+1}$ by using the ranking information obtained in step (3). Simply stated, if a new vertex is added to the simplex $S_k$ and the vertex corresponding to the worst approximation is dropped, then the mapping of the icons to the vertices is updated accordingly. If shrinkage occurs in the Nelder-Mead simplex algorithm, then it is necessary to rank $n$ new vertices. This is indicated to the user by having only one icon above the icon dividing line. The $n$ unranked vertices are then ranked one at a time.

4.4. Initializing the Simplex

There are many ways to form $S_0$ from the single point $x_0$. We would like $S_0$ to have $R(S_0) = n$, i.e., have vertices in general position in $\mathbb{R}^n$. One way of ensuring $R(S_0) = n$ is to take a set of linearly independent vectors $\{z_1, \ldots, z_n\}$ and set

$$x_i = x_0 + z_i, \quad i = 1, \ldots, n.$$  

This may lead to vertices of $S_0$ that are poorer estimates of the solution than $x_0$. The selection process can be adjusted to include a steplength $\alpha_i \neq 0$ such that

$$z_i = x_0 + \alpha_i z_i, \quad i = 1, \ldots, n,$$

where $\alpha_i$ is selected to improve the approximation of the solution. If we choose the directions $z_i$ to be ascent directions for some measure of goodness of fit, then we need consider only $\alpha_i > 0$.

Algorithm 4.2, given below, chooses the $z_i$ and $\alpha_i$ to decrease the initial criterion function $\phi(x, w)$ of the best approximation. Recall that we initialize $\phi$ to be

$$\phi(x, w) = \sum_{i=1}^{n} w_i \left[ y_i - y(x, t_i) \right]^2$$
and we have the initial set of weights \( w_0 \). The index \( B \) in Algorithm 4.2 is the index of the current best approximation, i.e. indexes \( v_0 \) in the ordered list of vertices.

**Algorithm 4.2: Initializing the Simplex**

Given \( x_0 \in \mathbb{R}^n \), \((t_i, y_i), i = 1, 2, \ldots, m\), and \( y(x,t) \)

Set \( B = 0 \), and \( z_0 = 0 \)

For \( k = 1, 2, \ldots, n \)

begin

1. Compute a direction \( p \in \mathbb{R}^n \) such that
   \[
   p^T \nabla_x \phi(x_B, w_k^*) < 0, \quad \text{and} \quad
   x_B - x_0 + p \notin \text{span} \{x_0, z_1, \ldots, z_{k-1}\}.
   \]

2. For some \( 0 < c < 1 \) find an \( \alpha > 0 \) such that
   \[
   \phi(x_B + \alpha p, w_k^*) \leq c \cdot \phi(x_B, w_k^*).
   \]

3. Set \( z_k = x_B - x_0 + \alpha p \) and \( z_k = x_0 + z_k \).

4. Rank \( z_k \) with respect to \( \{x_0, \ldots, z_{k-1}\} \).

5. Update \( w_{k-1}^* \rightarrow w_k^* \).

6. Update \( B \).

end

We remark that step (1) along with \( \alpha > 0 \) ensures that \( z_k \) will not be linearly dependent upon the previous vectors \( \{x_0, \ldots, z_{k-1}\} \) and that \( R(S_0) = n \). Also, because \( p \) is chosen to be a descent direction from the current best approximation, we are attempting to improve the fit as we generate \( S_0 \).

The computation of a step \( p \) with the properties described in step (1) is not a difficult task. In our implementation we take \( p = -\nabla_x \phi(x_B, w_k^*) \), or an approximation to it. If this lies in the space spanned by the previous vectors, then we perturb \( p \) by using a Jacobi rotation such that \( \beta = Jp \) and the properties of step (1) hold for \( \beta \). If \( \nabla_x \phi(x_B, w_k^*) = 0 \), then we ask the user to indicate, via the mouse, which data points are not fit sufficiently well. We then increase the relative weights for these data points. This yields \( \nabla_x \phi(x_B, w_k^*) \neq 0 \).

An \( \alpha \) that produces the desired descent of step (2) may be obtained by using a line search technique, see Dennis and Schnabel [1983], or Gill, Murray and Wright [1981]. Assuming the vertices are being ranked by the user, we use an inexact linesearch since our main interest is in generating \( S_0 \) with as little delay to the user as possible. Also, since we are using \( \phi(x,w) \) as our objective, and it may be a poor approximation to the user's choice criterion, exact minimization along \( p \) is unnecessary.

We let the user perform the rankings of the points. Thus, at each ranking we could update the weights as described in Chapter 3. However, we recompute \( w_k^* \) whenever it is needed by solving the associated problem. The updating of the index \( B \) is now clear, if the user ranks a new point as best, then \( B \) is set to the index of that vertex. Otherwise \( B \) is unchanged.
There exist other algorithms for the initialization of $S_0$. Parkinson and Hutchinson [1972] present alternative procedures in their discussion of simplex algorithms. We have based our procedure on the use of interactive graphics to rank the new points and the desire not to spend a great deal of effort in producing $S_0$.

CHAPTER 5

Summary and Further Research

We have presented an interactive procedure for obtaining a solution to multi-objective optimization problems. Our approach is based on using information obtained from the user to transform the problem into a single-objective optimization problem. This single-objective problem may then be solved using current methods and available software. Our method is similar to the method of White [1980], but we have extended his techniques via the contents of Chapters 2 and 3 of this thesis.

In the course of obtaining information from the user, we also attempt to improve the approximation to the solution. The reasons for this are again: to maintain the user's interest, to produce trial points whose partial ordering provides more salient information regarding the appropriate weights, and because we may be forced to rely upon this method to produce the final solution.

We have also described a software system that tests our approach on a subclass of multi-objective optimization problems. The subclass we consider is the class of curve-fitting problems, i.e., the problem of fitting a parametric curve to a given set of data. Appendix 2 of this thesis is a report of the solution of a
problem with the use of the software system.

There are several aspects of our work that may be extended. The transformation techniques of Chapter 3 can be extended for different geometric measures of optimality. For example, we could attempt to order the objectives by importance and the single-objective problem becomes a lexicographical optimization problem. Also, the set of criterion functions \( \phi(x,w) \) that are considered may be enlarged and the corresponding updating techniques for \( w_i \) would then require consideration.

The software system may also be extended. We would like to consider the general class of multi-objective optimization problems. The system's dependence on the curve-fitting problem is in the representations of the user's problem, his data, and the representations of the trial solutions, the fits. Thus, to consider a larger set of problems we need to represent different criteria and solutions in a manner that is useful to the user. A first step in this direction could be to consider multi-dimensional curve-fitting problems.

Another area where the work presented in this thesis may be extended is to find sufficient conditions for convergence of the Nelder-Mead simplex algorithm. A related extension would be to define an algorithm that possesses better theoretical and practical properties than the Nelder-Mead algorithm, where the new algorithm would also use only the ranking information for the iterates of the solution process.

BIBLIOGRAPHY


To appear in *European Journal of Operational Research*.


POWELL, M.J.D. [1984]. An efficient algorithm for finding the minimum of a


TELGREN, J. [1981]. Redundancy and linear programs, Mathematical Centre Tracts 137.


APPENDIX A

Curve Tailoring System User's Guide

The Curve Tailoring System is a set of menu-driven FORTRAN routines and C procedures for solving the 1-dimensional curve-fitting problem via an approach based on the use of interactive computer graphics. The system executes on a SUN Workstation under version 4.2 of Berkeley Unix and the Suncore Sun Graphics Package up to and including revision E of January 1984. Machine and Suncore dependencies are discussed in Section 4.

1. Overview

The 1-dimensional curve-fitting problem can be stated as follows: given \( m \) pieces of data \((t_i, x_i), i = 1, 2, \ldots, m, \) and a model \( y(x, t) \), where \( x \in \mathbb{R}^n \) is a vector of \( n \) parameters, \( t \in \mathbb{R}^l \), and \( y: \mathbb{R}^{n+l} \rightarrow \mathbb{R} \), find a nonnegative weight vector \( w^* = (w_1^*, w_2^*, \ldots, w_m^*)^T \in \mathbb{R}^m \), and a parameter vector \( z^* = (z_1^*, z_2^*, \ldots, z_l^*)^T \) such that \( z^* \) minimizes the criterion function

\[
\phi(x, w^*) = \sum_{i=1}^{m} w_i^* \mu(x; t_i, y_i),
\]

where \( \mu(x; t_i, y_i) \) is a geometric measure of optimality, or goodness of fit. We provide four types of criterion functions \( \phi(x, w) \), weighted least-squares,

\[
\phi(x, w^*) = \sum_{i=1}^{m} w_i^* |y_i - y(x, t_i)|^2,
\]

weighted \( l_1 \),

\[
\phi(x, w^*) = \sum_{i=1}^{m} w_i^* |y_i - y(x, t_i)|,
\]

weighted \( l_{\infty} \),

\[
\phi(x, w^*) = \max_{1 \leq i \leq m} w_i^* |y_i - y(x, t_i)|,
\]

and a weighted Huber M-criterion which uses \( \mu(x; t_i, y_i) \) defined by

\[
\mu(x; t_i, y_i) = \begin{cases} 
[y_i - y(x, t_i)]^2 & \text{for } |y_i - y(x, t_i)| \leq \gamma \\
2\gamma |y_i - y(x, t_i)| - \gamma^2 & \text{for } |y_i - y(x, t_i)| > \gamma
\end{cases}
\]

where \( \gamma > 0 \) is a constant. The Huber M-criterion is a combination of the least squares criterion and the \( l_1 \) criterion. The Curve Tailoring System uses weighted least squares as the default criterion.

Approximate solutions are generated by the system, then interactively ranked by the user via graphical input and output devices. The system uses the user's ranking information for the approximate solutions to assign a weight to each data point. The user may provide an initial set of weights, but it is not necessary that he do so. Each uninitialized weight is initialized to one by the software. The user is asked to rank each approximate solution as it is generated, with respect to a set of approximate solutions that have already been ranked. The weights are updated after each user ranking to satisfy constraints.
that are formed from the ranking.

At any time during a session with the Curve Tailoring System the user may request that the system compute the optimal set of parameters for the current criterion with the current set of weights. That is, the user may request the solution of

\[ \min_{x \in \mathbb{R}^n} \phi(x, w^*_i). \]  

(A.1)

If the solution \( x^*_i \) is not acceptable to the user, then the user can continue the session by ranking this approximate solution with respect to the best \( n+1 \) elements of the partially ordered set. Note the weights are updated automatically after the user's ranking, so that if \( x^*_i \) is not placed at the top of the list, then the weights are changed. Thus, the criterion is different and a different \( x^*_i \) will be computed as the solution to problem (A.1).

2. Usage

The user is required to provide the 1-dimensional data in a file that has the following form:

\[
\begin{align*}
& t_1 \quad y_1 \\
& t_2 \quad y_2 \\
& \vdots \\
& t_m \quad y_m,
\end{align*}
\]

where \( m > 0 \) is an integer that is the number of data points, and \((t_i, y_i)\) is the \( i^{th} \)
data point with \( t_i, y_i \in \mathbb{R}^1 \). The user is also asked to provide a set of initial points for the system. The number of initial points that the user provides should be an integer in the range \([0, n+1]\), where \( n \) is the number of parameters in the model. These values are inputted to the system after the data and are of the form

\[
\begin{align*}
& t_1 \\
& t_2 \\
& \vdots \\
& t_p \\
& t_{p+1} \\
& \vdots \\
& t_{n+1}
\end{align*}
\]

where \( n \) is the integer number of parameters, \( p \in \{0, 1, \ldots, n+1\} \), is the number of initial points, and \( t_j \in \mathbb{R}^1 \) is the \( j^{th} \) component of the \( i^{th} \) initial point. Additionally, the user must provide a FORTRAN subroutine or C procedure to evaluate the model \( y(x, t) \) for a set of parameters \( x \in \mathbb{R}^m \) and a \( t \in \mathbb{R} \). The FORTRAN subroutine must be of the form

```
SUBROUTINE MODEL (PARAMS, T, RESULT)
REAL*8 PARAMS(1), T, RESULT 
   RETURN
```

and return the value of \( y(x, t) \) as

\[ \text{RESULT} = y(\text{PARAMS}, T) \]

In C, the procedure must have the form

```c
#include <math.h>

double model(double t[], double p[], int n, double T)
{
    double result;
    // Evaluation of the model
    //...
    return result;
}
```
and return the value of $y(x,t)$ in the variable ‘answer’. The user is also asked to perform one additional task, that of ranking prospective solutions. We describe how this task is performed after presenting some preliminary material.

2.1. Preliminaries

Displayed in Figure A.1 is a typical view of the screen during a session with the Curve Tailoring System. The characters (A) through (H) are labels that are not visible on the screen but are added in the figure so that we may refer to regions of the graphics screen. The labelled regions of Figure A.1 are:

(A) The main menu: This is a list of available operations. These operations are: HELP, QUIT, SAVE, USER, Automatic, REDRAW, RESET, and SCREEN.

(B) The message/prompt region: One line messages and prompts are printed in this region to inform and prompt the user.

(C) The work area: Data and prospective solution curves are displayed here. Also, this area contains multi-lined help messages when necessary.

(D) The utilities menu: This menu contains utilities to aid a user in ranking $x_t$. Menu items are: ZOOM IN, ZOOM OUT, OOPS, DISPLAY, UNRANK, NO DIFFERENCE, and CHANGE DISPLAY.

FIGURE A.1
A typical Curve Tailoring screen with labels (A)-(H) added.
(E) Plot icons: Each icon (box) is a placeholder for the elements of the ordered list of vertices. Associated with each box is a set of parameters and the plot generated by the model with the set of parameters.

(F) Icon divider: Any icon above this line has been ranked by the user or by an automatic procedure. Any icon below the line, there will be at most one at any time, corresponds to the approximate solution to be ranked by the user.

(G) Unranked plot icon: The unranked icon. This corresponds to the approximate solution to be added to the ordered list.

(H) The mouse: A pointing device used by the user to select menu items and to perform the ranking operation.

A menu item is selected by placing the mouse in the box around the desired item and then pushing any mouse button. A solid icon below the icon divider, i.e., a solid unranked icon, means the corresponding plot is visible in the work area as a solid curve. The plots corresponding to icons above the icon divider are represented by dotted or flashing curves in the work area. If the mouse is placed in or below a ranked icon, then the icon becomes solid and the corresponding plot is made visible in the work area.

2.2. The Main Menu

At the start of a session with the curve-fitting system, the user sees the main menu and the work area with the data displayed in the work area. The user then chooses an item from the main menu. If HELP is chosen, then a help message is displayed in the work area. The content of the message provides information regarding the current phase of the solution procedure. For example, if a user has just started a session, then a brief explanation of the system is presented. A list of items will appear in a menu structure to the right of the work area. These items are those for which further help is available at the present level. To exit from HELP, click any mouse button in any place outside of the additional help menu.

To exit from the session at any time, select the item QUIT from the main menu. The results of a session may be saved before exiting by selecting SAVE. The information obtained up to the selection of SAVE is written into an output file. To restart the session from the point at which SAVE was selected, it is necessary to provide the output file as the input file for the Curve Tailoring System.

If USER is selected, then the screen is as depicted in Figure A.1. The user is now required to rank each of the approximate solutions, or fits. The ranking procedure is described in the following section. If additional information is desired at this level, then selecting HELP will provide an explanation of the current phase of the system and what is expected of the user during this phase.

The selection of AUTOMATIC results in the system optimizing with respect to the current fitting criterion with the current set of weights. The system obtains a solution \( z' \) for problem (A.1) by using a library optimizer. The system will then present \( z' \) as the approximate solution to the user to be ranked. That is, the system automatically enters USER mode with \( z' \) as the unranked approximate solution after the optimization has terminated.
If REDRAW is selected, then the screen contents that result from the Curve Tailoring System are redrawn. This is useful in the case that another user or the operating system prints a message on the screen. Also, an element of the screen can be erased by the Curve Tailoring System when an element that overwrites it is erased.

Upon selection of RESET, a list of items will appear in a menu structure to the right of the work area. These are the Curve Tailoring System parameters that may be reset by the user at any point during a session. For example, the system initializes the criterion function to be least-squares but the user prefers the weighted $l_1$ criterion. These system parameters are discussed in section 2.8.

If SCREEN is selected, then a copy of the screen is written to a file. After termination of the session with the system, a user may print this file on a laser printer to obtain a hard copy of the saved screen. This is useful for recording the final solution obtained through the use of the system. Alternatively, the screen can be dumped directly to a laser printer. This would result in at least a thirty second pause in the execution of the program. Although this feature is not available, it can be provided very easily.

2.3. Ranking an Approximate Solution

In user mode of the Curve Tailoring System, the unranked plot is visible in the work area as a solid curve. To rank this approximate solution the user compares the unranked plot with the plots corresponding to the $n+1$ best elements of the partially ordered set of approximate solutions. This is executed by moving the mouse up or down thru the list of ordered icons until the location for insertion of the unranked icon is established.

As an example of the ranking procedure, suppose the mouse starts in the bottom icon of the ordered list. The lowest icon is solid and the corresponding plot is visible in the work area as a dotted or flashing curve. If the user decides that the dotted curve is a better approximation to the solution than the solid curve, then the mouse should be positioned between the lowest ranked icon and the icon divider and any mouse button pushed. This indicates to the system that the position to insert the unranked approximate solution into the ordered list, is at the end of the list.

If the user decides that the solid curve is a better approximation to the solution than the dotted curve, then the user should move the mouse upward until the next highest icon of the ordered list becomes solid. As this happens, the lowest icon will become empty and the dotted curve that corresponds to the lowest icon will be replaced in the work area by the dotted curve that corresponds to the next higher icon. The user then compares this dotted curve with the solid curve.

The mouse is moved upward in the list until either the mouse is above the highest icon or until a dotted curve is found that is a better approximation to the solution than the solid curve. If the mouse is above the highest icon and the solid curve is better than the dotted curve, then there are no approximate
solutions that are better than the unranked plot. With the mouse above the highest icon the user inserts the unranked icon at the top of the list by pressing any mouse button. If a better approximate solution is found in the list, then the user inserts the unranked icon below this item by placing the mouse between this icon and the one directly below it and presses any mouse button. The procedure with the mouse starting at the top is similar, the user need only find a ranked curve that is a worse approximate solution than the unranked curve.

2.4. The Utilities

We provide a set of utilities to aid a user in comparing two approximate solutions. These utilities are contained in the menu to the right of the work area. The utility items ZOOM IN and ZOOM OUT let the user rescale the work area to view a region in greater detail, and to increase the visible region, respectively. If ZOOM IN is selected, then the system prompts the user to click any mouse button at a corner of the desired region in the work area. After this is executed, and the mouse is moved from this point, a box is drawn from the mouse to the selected point. The user is then to move the mouse to the diagonally opposite corner of the desired region. The box now contains the desired region. Upon clicking any mouse button, the region displayed in the zoom box will be mapped to the entire work area. Thus, anything outside the zoom box will no longer be visible and the region inside the zoom box will be seen in greater detail.

If ZOOM OUT is selected, then the process is similar except that the work area is mapped into the area inside the zoom box. That is, anything that was contained in the work area will now be in the region contained by the zoom box, and any data and parts of the curves that were outside the work area may now become visible in the work area.

If the item OOPS is selected from the utilities menu, then the most recent action by the user is undone. In the present version of the curve-fitting system, all the actions of the user have not been well-defined. Therefore, this operation has not been implemented.

The utilities menu item DISPLAY, if selected, enables a user to obtain information about any available approximate solution. If DISPLAY is selected, then the user is prompted to indicate the approximate solution of interest by clicking on the corresponding icon. When an icon has been selected, the utility menu is erased and information regarding this approximate solution is displayed to the right of the icons. The displayed information is the set of parameters and the values of the weighted criteria. Also, the corresponding plot is displayed in the work area and, at the bottom of the work area, the residuals are plotted via a bar graph. It is also possible to display the weights via a bar graph as well as display a bar graph of the weighted residuals. To view the information for a different approximate solution, it is only necessary to select the corresponding icon. This is true for all icons, including the unranked icon. A mouse button clicked outside of all icons, with the exception of the selection of HELP or
SCREEN, causes the system to exit from DISPLAY.

Any approximate solution that has been inserted into the ordered list at an incorrect position, can be removed from the list by selecting the utility item UNRANK. Upon selection of UNRANK, the user is prompted to indicate which icon is out of order. The clicking of any mouse button on an icon will remove the associated approximate solution from the ordered list. More than one approximate solution may be unranked at a time. To exit from UNRANK, click a mouse button outside of all ranked icons. The ordered list will always contain at least one element. After exiting from UNRANK, the system presents the unranked approximate solutions to the user to be ranked in the order that they were unranked.

It is possible that a user will be unable to decide where an approximate solution should be inserted into the list. If the user cannot decide which of two approximate solutions is better, then he may indicate this to the system by selecting the utility menu item NO DIFFERENCE. The system then prompts the user to indicate which ranked icon corresponds to the approximate solution that cannot be ordered with respect to the unranked approximate solution. We expect this utility item to be useful far from the solution when both approximate solutions are bad and it is difficult to tell which is worse, and close to the solution when the curves are too close together to tell which is better. The selection of NO DIFFERENCE will insert the unranked plot into the ordered list above or below the indicated icon in the position that corresponds to its ranking with respect to the current criterion and the current weights.

The selection of CHANGE DISPLAY enables a user to reset various parameters that control how the data and curves are represented in the work area. The user may select axes, the number of tick marks, separate bar plots of residuals, weights and weighted residuals, dotted curves or flashing curves for the plots corresponding to the ranked icons, as well as the speed of the flashing. The user can reset the scale to be the original scale as defined by the system and/or reset the scale to be the last zoom scale. The data points may be represented by any of a number of symbols: x's, + signs, circles, boxes, or triangles. Grid lines can be added or removed. Upon selection of CHANGE DISPLAY the user is presented with a menu of the parameters that he may change. Selection of an item from this menu will either change the item or cause another menu to appear. To exit from CHANGE DISPLAY, the mouse must be clicked outside the parameter menu. More than one item may be changed at a time and the changes are reflected in the work area as they are made.

2.5. Obtaining a Copy of the Solution

To obtain a hard copy of the resultant solution at the conclusion of a session, the user need only DISPLAY the desired solution and save a copy of it in a file by selecting SCREEN from the main menu. A copy of the screen contents are then available for printing on a laser printer. We allow multiple
screen dumps to take place, that is, all screen dumps are saved instead of just the latest screen dump. This enables a user to keep a record of the session with the Curve Tailoring System. The default number of screen dumps to be saved is one, i.e., the latest screen dump.

2.6. Curve Tailoring System Parameters

There are only a few Curve Tailoring System parameters that the user may change. These are: the choice criterion function, the method used to obtain the weights, whether mouse or keyboard input is used, and how many screen dumps are saved. The user may change any of the Curve Tailoring System parameters at any time during a session with the system.

The criterion functions are discussed in Section 1 of this user’s guide. If the Huber M-criterion is chosen, then we ask the user to provide a value for $\gamma$. This is accomplished by using the mouse to indicate on the bar graph which residual is the highest value that is less than the threshold value. The system will then set $\gamma$ to be equal to the average of this and the second highest value. Alternatively, the user may enter $\gamma$ via the keyboard.

There are three techniques available to obtain the weights. The weights are defined by using a norm to measure the distance between a desired set of weights and the set of feasible weights. For details see Section 3.1 of this user’s guide. The user may choose any of the three techniques for the defining of the weights. It should be noted that the methods based on the $l_1$ and $l_\infty$ norms may be computationally faster than the method based on the $l_2$ norm.

Although keyboard input for the operations is not available in the current version of the Curve Tailoring System, we intend to provide it in a later version. The menu elements will be numbered and the user selects a menu item by entering a number via the keyboard. Each icon will be assigned a symbol and, to view a plot corresponding to an icon, the user enters the icon’s symbol. Thus, all operations may be entered via the keyboard instead of by the use of the mouse.

The user may also desire to save copies of several screens, either as a set of different solutions or, as a history of the session with the system. We allow the user to save multiple screen dumps. Each screen dump is written into its own file. The name of the file is incremented after each screen dump. If the user saves $k$ screen dumps and resets the number of saved dumps to one, then only one additional screen dump is saved.

3. Algorithmic Details

In this section we briefly discuss several algorithmic topics. The topics of discussion are: how the weights are obtained, how the approximate solutions are generated, and the graphics procedures that are used by the system. For a more detailed description of these topics, see Chapters 2-4 of this thesis.
3.1. Obtaining Weights

In the Curve Tailoring System the weights for the data points are obtained by solving a linearly constrained problem. The constraints are derived from the order information for the approximate solutions. For example, if we have that the parameter set \( x_1 \) is a better approximate solution than the parameter set \( x_2 \), then we would like the weights to satisfy the constraint

\[ \phi(x_1, w) \leq \phi(x_2, w). \]

We form all the constraints that result from the user's rankings and add the additional constraints that the weights be nonnegative and that the sum of the weights be equal to a constant. We choose the constant to be equal to the number of data points.

To obtain an optimal weight from the set of feasible weights, we ask that each weight be as close as possible to a preferred value. The preferred values are merely the weights that were current before the latest user ranking. That is, we ask that the weights change as little as possible after each user ranking. The weights are initialized by a combination of the user and the system. If the user fails to initialize all the weights, then the system sets the uninitialized weights to one. The system then resets any negative weights to zero and renormalizes the weights to have their sum equal to the required constant. Thus, the initial weights satisfy the constraints defined at the start of a session.

If the \( l_1 \) or \( l_\infty \) norm is chosen to determine which set of weights is as close as possible to the preferred set, then the resultant problem that must be solved for the weights is a linear programming problem. The weights can be updated quickly by using the dual simplex algorithm. In the present version of the Curve Tailoring System, the dual simplex algorithm is not used, and the linear programming problem is solved using a different technique.

If the \( l_2 \) norm is chosen to determine which set of weights is as close as possible to the preferred set, then a quadratic programming problem results. The weights can be updated using a technique similar to the dual simplex algorithm for linear programming problems, but the present version of the Curve Tailoring System does not make use of this technique. Instead, a quadratic programming package is used to solve the QP problem.

It may happen that there are no weights that satisfy the constraints imposed by the user and the additional constraints imposed by the system. In this case the user has several options. The system is capable of searching through a prescribed set of criterion functions to determine if any of the criterion functions will produce a feasible set of weights. Also, the system is able to determine which constraints are inconsistent and will prompt the user to see if the orderings that provide these constraints may be reversed. There may be many orderings that require inversion before a feasible set is obtained. If none of the above options is acceptable, then the user may continue to search for a solution without the use of the criterion function and weights. Thus, it is
desired that the approximate solutions that are generated automatically by the system have certain convergence properties.

3.2. Generating Approximate Solutions

The approximate solutions that the user is asked to rank are generated by the Nelder-Mead simplex algorithm. This algorithm moves an n-dimensional simplex thru $\mathbb{R}^n$ based on the relative ranks of the vertices. The movement of the simplex is away from the vertex corresponding to the worst approximate solution. For the execution of this algorithm, only order information of the vertices is required. This is exactly what the user provides by ranking the approximate solutions.

If the user supplies less than $n+1$ initial points, then the simplex is initialized by using a gradient algorithm from the best approximate solution. The function used to define the gradient is that of the current criterion function with the current weights. The search directions used to define the initial simplex are perturbed so that the simplex will span $\mathbb{R}^n$. An inexact linesearch technique is employed in order to obtain sufficient descent along each search direction.

If the user selects AUTOMATIC from the main menu, then the unranked point generated by the Nelder-Mead algorithm is replaced by the solution to the optimization of the criterion function, i.e., $z^\star$. An appropriate vertex of the simplex is dropped after the user has ranked $z^\star$ so that the updated simplex spans $\mathbb{R}^n$.

3.3. Graphics Procedures

There are five basic graphics procedures that the Curve Tailoring System employs. These are: placing text at a position on the screen, drawing lines, drawing filled polygons, mouse input and output, and turning on and off graphics segments. The curves are drawn by using a linear approximation that has as little error as the graphics screen can supply. The icons are drawn by filling a polygon with some pattern. The patterns we use are black, which results in a solid icon, and white, which results in an empty icon. The mouse is tracked over the screen and selections are made by recording the position of the mouse and what symbol is at that position on the screen. Curves, menus, icons, help messages, etc., are made visible and invisible by turning on and off the appropriate graphical segment, respectively.

4. Programming Details

The Curve Tailoring System is dependent upon the graphics procedures and the use of a graphics screen. To be transported to another system, it would be necessary to provide the graphics capabilities that the Curve Tailoring System requires or, rewrite the small amount of the Curve Tailoring System code that uses the machine dependent routines.
The Curve Tailoring System is written in a combination of the FORTRAN and C programming languages. This is due to the fact that the SUN Workstation and Berkeley Unix 4.2 environment are oriented more towards C than FORTRAN. However, there are certain libraries that perform various tasks, such as solving for the weights, or solving problem (A.1), that are available to us in the FORTRAN language. We have made use of these libraries in the course of programming the system.

**APPENDIX B**

**Solving an Example Problem**

We obtained the problem we have used most for the development of our system from Professor Bruce Weisman of the Chemistry Department of Rice University. The problem is to fit the four parameter model given by

\[ y(x,t) = \begin{cases} x_1 e^{-2x^2} + x_2 e^{-x^2} & \text{for } t \geq 0 \\ 0 & \text{for } t < 0 \end{cases} \]

To the sixty-three data points given at the end of this appendix.

After developing the system to the point presented in Appendix A, we asked Professor Weisman to solve his problem via the use of the system. We provided the following three initial points:

\[
\begin{align*}
x_0^1 &= (120, 80, 0.03, 0.001), \\
x_0^2 &= (104.519, 74.7868, 0.003304, 0.000809), \\
x_0^3 &= (112.4556, 78.487, 0.01414, 0.00078).
\end{align*}
\]

The plot corresponding to \( x_0^1 \) is shown in Figure B.1. We have initialized no weights, therefore, each weight is assigned the value one by the system.

At the beginning of the session with the system, Professor Weisman selected USER from the main menu, and adjusted the plot area to that shown...
in Figure B.1. The method for choosing the weights was set, via selection of
RESET from the main menu, to be the QP technique as discussed in Chapter 3,
and the criterion function was set to the $l_2$ criterion. The plot for $x_2^3$ was made
visible on the screen at the selection of USER. Professor Weisman rated this
plot worse than the plot for $x_2^5$. The plot corresponding to $x_2^5$ is displayed in
Figure B.2. The software then presented $x_2^3$ as the next point to be ranked, and
the plot corresponding to this point is displayed in Figure B.3. Professor
Weisman ranked $x_2^3$ between $x_2^1$ and $x_2^5$.

Professor Weisman then selected AUTOMATIC from the main menu. The
weights were obtained via the QP procedure. The weights were identically one,
i.e., no information had been obtained that would cause the weights to be changed from the initial values.

The point $x_1 = (65.786095, 75.32763, 0.011183, 0.000868)$ was obtained as
the solution to the nonlinear least squares problem by LMDIF of the MINPACK
library. The plot corresponding to $x_1$ is shown in Figure B.4. Professor
Weisman rated this plot between those of $x_2^3$ and $x_2^5$. The order of the ranked
points is now, from best to worst, $x_2^1, x_2^3, x_1, x_2^5$.

The weights have been changed at this iteration because the solution to the
nonlinear least squares problem was not rated as the best point. These new
weights are given at the end of this appendix, after the data points. Professor
Weisman then requested the solution to the new nonlinear least squares problem
declared by the updated weights. The solution,

\[ x_2 = (121.3134, 85.17824, 0.026732, 0.0001052), \] is plotted in Figure B.5.

Being satisfied with this solution, Professor Weisman selected DISPLAY
from the utilities menu, and saved a copy of the solution for printing by
selecting SCREEN from the main menu.

By viewing a plot of the data, it is easy to see that the data point
(0, 69.878) contains a great deal of error. The error is due to the necessity of
the measuring apparatus to reach the value of the measurement. That is, there
is a time delay for the machine to warm up, and this data point has been taken
before the machine reached the level of the true value. Thus, the problem may
be solved easily by other means once the erroneous data point is discovered.
However, the exercise of solving the problem via our system shows that our
approach is capable of finding an appropriate set of weights for this problem.

The data and final weights may be found in Tables 1 and 2 following the
figures. Table 1 contains the data for the problem, and Table 2 contains the
weights used in the nonlinear least squares formulation to obtain the optimal
solution.
Figure B.5

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$x_i$</th>
<th>$y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.</td>
<td>1.208</td>
<td>170.</td>
<td>73.073</td>
<td>410.</td>
<td>53.522</td>
</tr>
<tr>
<td>0.</td>
<td>69.878</td>
<td>180.</td>
<td>69.737</td>
<td>435.</td>
<td>55.331</td>
</tr>
<tr>
<td>10.</td>
<td>175.328</td>
<td>185.</td>
<td>70.851</td>
<td>460.</td>
<td>52.573</td>
</tr>
<tr>
<td>20.</td>
<td>154.113</td>
<td>190.</td>
<td>70.843</td>
<td>485.</td>
<td>50.519</td>
</tr>
<tr>
<td>30.</td>
<td>140.188</td>
<td>200.</td>
<td>67.152</td>
<td>510.</td>
<td>49.422</td>
</tr>
<tr>
<td>35.</td>
<td>134.301</td>
<td>210.</td>
<td>69.289</td>
<td>535.</td>
<td>50.028</td>
</tr>
<tr>
<td>40.</td>
<td>123.165</td>
<td>220.</td>
<td>68.640</td>
<td>560.</td>
<td>44.360</td>
</tr>
<tr>
<td>50.</td>
<td>111.074</td>
<td>230.</td>
<td>67.339</td>
<td>585.</td>
<td>44.779</td>
</tr>
<tr>
<td>60.</td>
<td>106.422</td>
<td>235.</td>
<td>67.368</td>
<td>610.</td>
<td>44.934</td>
</tr>
<tr>
<td>70.</td>
<td>97.580</td>
<td>240.</td>
<td>68.897</td>
<td>635.</td>
<td>42.445</td>
</tr>
<tr>
<td>80.</td>
<td>88.108</td>
<td>250.</td>
<td>66.733</td>
<td>660.</td>
<td>39.288</td>
</tr>
<tr>
<td>85.</td>
<td>91.857</td>
<td>260.</td>
<td>66.047</td>
<td>685.</td>
<td>39.234</td>
</tr>
<tr>
<td>90.</td>
<td>83.823</td>
<td>270.</td>
<td>65.769</td>
<td>710.</td>
<td>41.631</td>
</tr>
<tr>
<td>100.</td>
<td>83.571</td>
<td>280.</td>
<td>63.948</td>
<td>735.</td>
<td>39.124</td>
</tr>
<tr>
<td>110.</td>
<td>82.577</td>
<td>285.</td>
<td>64.275</td>
<td>760.</td>
<td>38.274</td>
</tr>
<tr>
<td>120.</td>
<td>77.743</td>
<td>290.</td>
<td>64.490</td>
<td>785.</td>
<td>36.489</td>
</tr>
<tr>
<td>130.</td>
<td>78.373</td>
<td>300.</td>
<td>64.783</td>
<td>810.</td>
<td>37.380</td>
</tr>
<tr>
<td>135.</td>
<td>79.492</td>
<td>310.</td>
<td>63.552</td>
<td>835.</td>
<td>35.938</td>
</tr>
<tr>
<td>140.</td>
<td>73.648</td>
<td>325.</td>
<td>62.021</td>
<td>860.</td>
<td>34.648</td>
</tr>
<tr>
<td>150.</td>
<td>73.491</td>
<td>360.</td>
<td>59.484</td>
<td>885.</td>
<td>34.731</td>
</tr>
<tr>
<td>160.</td>
<td>73.589</td>
<td>385.</td>
<td>55.785</td>
<td>910.</td>
<td>32.524</td>
</tr>
</tbody>
</table>

Table 1
<table>
<thead>
<tr>
<th>$t_i$</th>
<th>$w_i$</th>
<th>$t_i$</th>
<th>$w_i$</th>
<th>$t_i$</th>
<th>$w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.</td>
<td>1.017812</td>
<td>170.</td>
<td>1.013099</td>
<td>410.</td>
<td>1.015601</td>
</tr>
<tr>
<td>0.</td>
<td>0.</td>
<td>180.</td>
<td>1.011558</td>
<td>425.</td>
<td>1.010327</td>
</tr>
<tr>
<td>10.</td>
<td>1.286518</td>
<td>185.</td>
<td>1.013915</td>
<td>460.</td>
<td>1.017358</td>
</tr>
<tr>
<td>20.</td>
<td>1.124206</td>
<td>190.</td>
<td>1.014853</td>
<td>485.</td>
<td>1.016200</td>
</tr>
<tr>
<td>30.</td>
<td>1.062270</td>
<td>200.</td>
<td>1.012640</td>
<td>510.</td>
<td>1.016188</td>
</tr>
<tr>
<td>35.</td>
<td>1.043590</td>
<td>210.</td>
<td>1.016333</td>
<td>535.</td>
<td>1.018337</td>
</tr>
<tr>
<td>40.</td>
<td>0.9852980</td>
<td>220.</td>
<td>1.016977</td>
<td>560.</td>
<td>1.012280</td>
</tr>
<tr>
<td>50.</td>
<td>0.9575678</td>
<td>230.</td>
<td>1.016843</td>
<td>585.</td>
<td>1.014121</td>
</tr>
<tr>
<td>60.</td>
<td>0.9737329</td>
<td>235.</td>
<td>1.017411</td>
<td>610.</td>
<td>1.015608</td>
</tr>
<tr>
<td>70.</td>
<td>0.9673938</td>
<td>240.</td>
<td>1.019425</td>
<td>635.</td>
<td>1.013557</td>
</tr>
<tr>
<td>80.</td>
<td>0.9567759</td>
<td>250.</td>
<td>1.018289</td>
<td>660.</td>
<td>1.010569</td>
</tr>
<tr>
<td>85.</td>
<td>0.9833943</td>
<td>260.</td>
<td>1.018543</td>
<td>685.</td>
<td>1.011705</td>
</tr>
<tr>
<td>90.</td>
<td>0.9662380</td>
<td>270.</td>
<td>1.019152</td>
<td>710.</td>
<td>1.016122</td>
</tr>
<tr>
<td>100.</td>
<td>0.9842573</td>
<td>280.</td>
<td>1.018181</td>
<td>735.</td>
<td>1.013923</td>
</tr>
<tr>
<td>110.</td>
<td>0.9951890</td>
<td>285.</td>
<td>1.018919</td>
<td>760.</td>
<td>1.013930</td>
</tr>
<tr>
<td>120.</td>
<td>0.9945688</td>
<td>290.</td>
<td>1.019541</td>
<td>785.</td>
<td>1.012655</td>
</tr>
<tr>
<td>130.</td>
<td>1.000208</td>
<td>300.</td>
<td>1.020643</td>
<td>810.</td>
<td>1.014965</td>
</tr>
<tr>
<td>135.</td>
<td>1.000108</td>
<td>310.</td>
<td>1.020151</td>
<td>835.</td>
<td>1.014110</td>
</tr>
<tr>
<td>140.</td>
<td>1.001950</td>
<td>325.</td>
<td>1.020393</td>
<td>860.</td>
<td>1.013440</td>
</tr>
<tr>
<td>150.</td>
<td>1.006818</td>
<td>335.</td>
<td>1.019350</td>
<td>885.</td>
<td>1.014594</td>
</tr>
<tr>
<td>160.</td>
<td>1.010542</td>
<td>355.</td>
<td>1.016751</td>
<td>910.</td>
<td>1.012659</td>
</tr>
</tbody>
</table>

**Table 2**