A Direct Approach to Two-Level Decomposition:
Structural Optimization Using the
Generalized Reduced Gradient

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in Mechanical Engineering

by

Thomas A. Veilleux,

November 1991
Abstract

The goal of this thesis is to develop a straightforward approach to multi-level decomposition for the optimization of large engineering systems. The method's organizational structure is based on the Model Coordination approach outlined by Kirsch (1975). In this approach, the system is partitioned into a set of substructures or subsystems, where the local detailed dimensions of quantities in each subsystem are the design variables in that subsystem, while all other problem dimensions are held fixed. The optimization sub-problems formed from each of the subsystems are solved separately. There is one system level sub-problem in which the system's gross geometric dimensions are the design variables, while all other lower level dimensions are held fixed. The set of sub-problems is repeatedly solved until the system level objective function converges.

Two new developments contribute to the success of this innovative direct approach to structural decomposition. The first new feature is that each of the sub-problems is solved with the Generalized Reduced Gradient optimization algorithm (Abadie & Carpentier, 1969). Because the Generalized Reduced Gradient method develops a sequence of feasible points as intermediate solutions and the value of the objective function must decrease monotonically with that sequence, the iterations can be stopped at any time with a resulting feasible solution.
that will have a superior objective function value. This is particularly advantageous when a practical optimum will be accepted if the theoretical optimum can not be found. The second and more significant development is as follows: by adjusting the maximum number of iterations per sub-problem and the initial step size, progress is limited in each of the sub-problems so that none strongly bias the direction that the overall solution takes. This causes the individual sub-problems to approach the system's optimum solution in a uniform manner. Successful convergence of this approach is demonstrated with two planar truss problems.

The most valuable advantage of this technique over others discussed in the current literature is its ease of use. The analyst need only supply the objective functions, the constraints, their gradients, an initial starting point and an initial step size. This method has potential applications in the optimization of large multi-disciplinary engineering systems. Different optimization algorithms, specialized for a particular problem type or discipline, could be used for each of the various subsystems as is necessary.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>ix</td>
</tr>
<tr>
<td>Chapter 1: Basic Concepts in Optimization</td>
<td></td>
</tr>
<tr>
<td>1.1 Problem Formulation Process for Optimization Studies</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Optimization Algorithms</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Basic Nomenclature in Optimization</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Optimality Conditions</td>
<td>9</td>
</tr>
<tr>
<td>1.5 List of Symbols</td>
<td>18</td>
</tr>
<tr>
<td>Chapter 2: Development of the Generalized Reduced Gradient Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>2.1 Implicit Variable Elimination</td>
<td>27</td>
</tr>
<tr>
<td>2.2 Development of the Search Vector</td>
<td>34</td>
</tr>
<tr>
<td>2.3 Treatment of Bounds</td>
<td>43</td>
</tr>
<tr>
<td>2.4 Basis Update Rules</td>
<td>48</td>
</tr>
<tr>
<td>2.5 Generalized Reduced Gradient Algorithm</td>
<td>49</td>
</tr>
<tr>
<td>2.6 Single-Level Example: Two Bar Truss</td>
<td>56</td>
</tr>
<tr>
<td>Chapter 3: Decomposition</td>
<td>60</td>
</tr>
<tr>
<td>3.1 Introduction to Multi-Level Decomposition:</td>
<td></td>
</tr>
<tr>
<td>Motivation for a Direct Approach</td>
<td>62</td>
</tr>
</tbody>
</table>
3.2 Development of a Direct Method..........................64
3.3 Decomposition Algorithm.....................................79
3.4 Two-Level Decomposition Example: Two Bar Truss...........84

Chapter 4: Conclusions & Recommendations.......................93
4.1 Conclusions.....................................................93
4.2 Recommendations..............................................96

Bibliography........................................................97

Appendices..........................................................99
A  GRGOPT32 Fortran Code........................................100
B  Solution of Single-Level Optimization Problems..............136
B.1 General Usage Notes...........................................137
B.2 Output Format & Error Messages..............................143
B.3 Example Problems..............................................150
B.4 Key Definition Information for Single-Level
  Problems..........................................................161
B.5 Template 1: Single-Level Optimization Problem
  Outline............................................................173
C  Solution of Two-Level Optimization Problems.................182
C.1 General Usage Notes..........................................182
C.2 Output Format.................................................187
C.3 Example Problems..............................................191
C.4 Key Definition Information for Two-Level Problems......195
C.5 Template 2: Two-Level Optimization Problem Outline......205
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kuhn-Tucker Conditions 16</td>
</tr>
<tr>
<td>2</td>
<td>Linearization of a Nonlinear Constraint 37</td>
</tr>
<tr>
<td>3</td>
<td>Reduced Gradient Search Direction &amp; the Curve of Feasible Points 38</td>
</tr>
<tr>
<td>4</td>
<td>Linear Interpolation to Satisfy Variable Bounds 47</td>
</tr>
<tr>
<td>5</td>
<td>Single-Level Example: Two-Bar Truss 57</td>
</tr>
<tr>
<td>6</td>
<td>Two-Level Example: Two-Bar Truss 85</td>
</tr>
<tr>
<td>B-1</td>
<td>Example Test5: Modified Two-Bar Truss 158</td>
</tr>
<tr>
<td>C-1</td>
<td>Example Test5D: Modified Two-Bar Truss 193</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Estimation of Initial Step Size: Starting Point A</td>
<td>87</td>
</tr>
<tr>
<td>2</td>
<td>Solution A for Two-Level Truss Problem</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>Estimation of Initial Step Size: Starting Point B</td>
<td>88</td>
</tr>
<tr>
<td>4</td>
<td>Solution B for Two-Level Truss Problem</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>Solution B: Initial Step Size $\alpha^0 = 300$</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>Solution B: Initial Step Size $\alpha^0 = 1$</td>
<td>90</td>
</tr>
<tr>
<td>B-1</td>
<td>Data Format for Intermediate Status Output</td>
<td>144</td>
</tr>
<tr>
<td>B-2</td>
<td>Data Format for Summary Status Report</td>
<td>145</td>
</tr>
<tr>
<td>B-3</td>
<td>Solution Sequence for Problem Test1</td>
<td>152</td>
</tr>
<tr>
<td>C-1</td>
<td>Main Interactive Menu for Decomposition Problems</td>
<td>188</td>
</tr>
<tr>
<td>C-2</td>
<td>Output Data Format for Decomposition Problems</td>
<td>189</td>
</tr>
<tr>
<td>C-3</td>
<td>Solution A for Problem Test5D</td>
<td>194</td>
</tr>
<tr>
<td>C-4</td>
<td>Solution B for Problem Test5D</td>
<td>194</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to express my sincere gratitude for the assistance of B. C. Fabien. His guidance, motivation and support were instrumental in the successful completion of this thesis. My dearest thanks to my wife, L. Davis, who with much patience and love, put up with me and this obsession.
In this chapter, basic concepts needed to embark on an optimization study are presented. The topics discussed are the process of problem formulation, the elemental operation of optimization algorithms, nomenclature used in expressing the problem in standard format, optimality conditions for constrained and unconstrained problems and symbols used in this thesis.

1.1 Problem Formulation Process for Optimization Studies

Optimization theory is a collection of mathematical results and numerical techniques used for finding and identifying the best candidate from a set of alternatives without having to explicitly enumerate and evaluate all possible alternatives. The optimization process lies at the root of engineering, since the traditional function of the engineer is to design new, better, more efficient, and less expensive systems as well as to devise plans and procedures for the improved operation of existing systems (Reklaitis, Ravindran and Ragsdell, 1983).

Independent of which optimization technique is employed, proper formulation of the engineering optimization problem is key to the success of the optimization study. Reklaitis et. al. (1983) outline this process as described below.
The process of optimization begins with the definition of the engineering system to be optimized. In setting up the optimization problem, as is the case when any physical situation is modeled mathematically, the system must first be isolated from its surroundings. Since interactions always exist between the system and the remainder of the universe, the mathematical model will be an approximation. To fully analyze the given system, all factors that have a strong effect on that system must be considered. The system's boundaries are defined through decisions to include or exclude the various factors that have any effect on that system.

Having selected a system with well defined boundaries, the second step is to choose a criterion on which the system performance or the system design can be evaluated. Typical performance criteria are weight, power consumption, profit and manufacturing cost. Regardless of the criterion used, in the context of optimization, the best always means the candidate system with the maximum or minimum value of the performance criterion.

The third step in formulating a problem for optimization is deciding which variables adequately characterize the system. These variables are called the independent or design variables. It is necessary to distinguish between variables that can change and variables that are fixed by external factors. It is important to include all variables that significantly influence the performance of the system or the
design of the system. The number of independent variables in a problem is often referred to as the problem's dimension.

Once the system's performance criterion and the independent variables have been selected, the fourth step is to develop a mathematical model that describes the relationship between the independent variables and the manner in which those variables influence the performance criterion. Generally, the system model is a collection of equations that define how the system's variables are related and a set of inequalities that constrain those variables to have acceptable values. In the context of structural engineering optimization, the mathematical model consists of expressions to determine the weight of the structure coupled with constraints that limit the stress, load, natural frequency and/or deflection of the various members of the system.

In summary, to set up a problem for an optimization study, the following steps must be completed: define the system to be studied, select a performance criterion, choose a set of design variables and construct a mathematical model of the system. Once this has been accomplished, the next step is to express the problem in a format that can be recognized by an optimization algorithm. In the following section, various optimization algorithms that are currently in use are discussed.
1.2 Optimization Algorithms

Optimization algorithms are numerical techniques used in finding and identifying the best candidate from a set of alternatives without having to explicitly enumerate and evaluate all possible alternatives. Though there are many different algorithms available, they all share some basic features. Starting with some initial point, the common goal is to traverse the design space seeking an improvement in the value of the performance criterion. This process is usually performed iteratively, where in each iteration a search is carried out in a direction likely to provide improvement in the performance criterion. If the new point should prove acceptable, various tests are then performed to check to see if it is an optimal solution. This procedure is repeated until either an optimum solution is found or the maximum number of iterations is reached.

The many algorithms available can be put into different categories. For example, one classification is what types of problems the method can or cannot solve, e.g. linear or nonlinear functions, continuous or discrete variables, large or small problem size. Others have limitations on what types of constraints the problem can have, e.g. linear, nonlinear, equality, inequality or no constraints at all. All algorithms have certain information that the analyst must provide, therefore algorithms can be categorized by the type or complexity
of this information, e.g. objective or constraint functions only (i.e. no derivatives), first derivatives of functions and/or second derivatives of functions. Still others may develop a sequence of feasible points (i.e. points that satisfy all the constraints) throughout the search for optima, while others may produce infeasible points during the solution sequence.

Optimization algorithms are rated on how robust they are, that is, how efficient or successful the particular method is for solving a variety of problem types. There are benchmark problems, whose solutions are known, that are used exclusively to test the robustness of optimization algorithms. These problems have a topology or structure that makes the search for the optimum solution either difficult or impossible.

Another subclass of optimization algorithms are those that are specialized for optimizing an engineering system through decomposition. In this type of algorithm the engineering system is partitioned into subsystems, each solved separately as an individual sub-problem. What makes these decomposition techniques unique is how the solutions of each optimization sub-problem are related to the others and to the system as a whole.

Depending on the type of problem, the nature of the variables, the type of constraints and the number of independent variables, the analyst must decide which type of optimization algorithm is best suited for the application.
However, the optimization algorithm may have been selected in advance, leaving the analyst with no choice but to modify the problem to adapt to the software available.

Two good references on the historical aspects of optimization theory are (Vanderplaats, 1982) and (Schmidt, 1981). These articles offer a useful sense of perspective in the field of optimization and both have extensive bibliographies. A very useful introductory text on this subject is Reklaitis, Ravindran and Ragsdell, (1983). This work derives the first and second order optimality theorems for constrained and unconstrained problems, develops many classical optimization algorithms as well as state of the art techniques (circa 1983) and has numerous examples that are useful in testing code.

The optimization algorithm presented in this thesis is called the Generalized Reduced Gradient. It was selected because its implementation is relatively straight forward and it develops only feasible points as intermediate solutions. A useful implication of this second feature is that the succession of objective function values associated with the sequence of feasible points decreases monotonically. Therefore, the iterations can be stopped at any time, resulting in a feasible point with a objective function value superior to the initial point. This is advantageous when a practical optimum will be accepted if the search for the theoretical optimum has stalled.
1.3 Basic Nomenclature in Optimization

A design optimization problem can be expressed in the following form.

Minimize: \( F(X) \) \hspace{1cm} 1.1
Subject to: \( H_k(X) = 0 \) \hspace{1cm} for \( k = 1,2,\ldots,K \) 1.2
\( A_j \leq G_j(X) \leq B_j \) \hspace{1cm} for \( j = 1,2,\ldots,J \) 1.3
\( X_i^L \leq X_i \leq X_i^U \) \hspace{1cm} for \( i = 1,2,\ldots,N \) 1.4
\( X = (X_1,X_2,\ldots,X_N) \) \hspace{1cm} 1.5

where, \( F(x) \) is the objective function or performance criterion of the design problem. This function may represent the total weight of a structure, the cost incurred by a manufacturing process or the power consumed by a control system. The goal of the optimization study is to minimize this function without violating the constraints or the variable bounds. The \( N \) components of the \( N \)-vector \( X \) are the optimization problem's design variables, also called the independent variables. Examples of design variables are the length of a beam, the width of a weld and the wall thickness of a strut. The \( K \) equality constraints and the \( J \) inequality constraints are denoted by \( H_k \) and \( G_j \), respectively. The \( J \)-vectors \( A \) and \( B \) represent lower and upper limits on the \( J \) inequality constraints, respectively. These constraints could be limitations on deflection of the structure or the stress
in an individual component. In addition to the equality and inequality constraints on $X$, the components of $X$ are often individually bounded between lower and upper limits, denoted here by $N$-vectors $X^l$ and $X^u$, respectively. Variable bounds can prevent a design variable from falling below some minimum gauge or restrain the structural dimensions to some pre-determined limits.

It is often beneficial to convert inequality constraints into equality constraints with the introduction of slack variables. This allows all constraints to be treated in a similar manner by the optimization algorithm. The slack variable represents the slack between the boundary and the inequality expression. This conversion of inequalities to equalities has a disadvantage in that it increases the number of independent variables from $N$ to $N+J$ because there will be one new independent variable for each inequality constraint converted. Therefore the design optimization problem defined by equations (1.1) through (1.5) is converted to the following format.

\[
\begin{align*}
\text{Minimize:} & \quad F(X) \\
\text{Subject to:} & \quad H_k(X) = 0 \quad \text{for } k = 1,2,\ldots,K \\
& \quad G_j(X) - X_{j\text{N}} = 0 \quad \text{for } j = 1,2,\ldots,J \\
& \quad X_i^l \leq X_i \leq X_i^u \quad \text{for } i = 1,2,\ldots,N \\
& \quad A_j = X_{j\text{H}}^l \leq X_{j\text{H}} \leq X_{j\text{H}}^u = B_j \quad \text{for } j = 1,2,\ldots,J \\
& \quad X = (X_1,X_2,\ldots,\ldots,X_{N+J})
\end{align*}
\]
Equations (1.6) through (1.11) are the standard form that will be used in this thesis.

Finally, two more references from standard optimization terminology that are used throughout this thesis are the initial design vector and the optimum solution, $x^{(0)}$ and $x^*$, respectively. Section 1.5 contains a complete list of all symbols and terminology used in this thesis.

1.4 Optimality Conditions

It is important to develop conditions or tests that allow the characterization of points in the design space. Examination of optimality criteria is meaningful because they are necessary to recognize optimum solutions and they provide the motivation for most optimization methods (Reklaitis et. al. 1983).

**Unconstrained Optimality Conditions**

Consider the following unconstrained nonlinear optimization problem.

Minimize: $F(X)$  \hspace{1cm} 1.12

$X_i$ \hspace{1cm} for $i = 1, 2, \ldots, N$  \hspace{1cm} 1.13
Now consider a second order truncated Taylor expansion of $F(X)$ about some point $X'$.

$$F(X) = F(X') + \text{Grad} F(X') (X - X') + \frac{1}{2} (X - X') \text{Grad}^2 F(X') (X - X') \quad 1.14$$

Recall that by its definition, a minimum is a point such that all other nearby points produce a larger objective function value. That is,

$$\Delta F = F(X) - F(X')$$
$$= \text{Grad} F(X') (X - X') + \frac{1}{2} (X - X') \text{Grad}^2 F(X') (X - X') \geq 0 \quad 1.15$$

If equation (1.15) holds for all $X$ in $\mathbb{R}^n$, the point $X'$ is a **global minimum** and it is denoted as $X'$. Since in practice this is usually very difficult to prove without testing all the points in the design space, a more useful definition is needed. If equation (1.15) is true for some **local** region near $X'$ where

$$|X - X'| \leq \delta$$

and

$$\delta > 0$$

then $X'$ is a **local minimum**, denoted $X'$. A definition for
local maximum is obtained by reversing the inequality in equation (1.15). When \( \Delta F \) in equation (1.15) is either positive, negative or zero depending on the choice of nearby points, the point \( X' \) is a \textbf{saddlepoint}.

For the sign of \( \Delta F \) to be known for arbitrary values of \( (X-X') \), \( \text{Grad} \, F(X') \) must be zero, that is, \( X' \) must be a stationary point. Otherwise, the value of \( \Delta F \) could be plus or minus depending on the sign of \( \text{Grad} \, F \) and \( (X-X') \). At a stationary point, equation (1.15) becomes:

\[
\Delta F = F(X) - F(X')
\]
\[= \frac{1}{2}(X - X')^T \text{Grad}^2 F(X') (X - X') \geq 0 \quad 1.16
\]

It is clear that the sign of \( \Delta F \) depends on the nature of the quadratic form

\[Q(X) = (X - X')^T \text{Grad}^2 F(X') (X - X')\]

From linear algebra it is known that:

\( \text{Grad}^2 \, F \) is positive definite, if for all \( (X-X') \), \( Q(X) > 0 \)

\( \text{Grad}^2 \, F \) is positive semidefinite, if for all \( (X-X') \), \( Q(X) \geq 0 \)

\( \text{Grad}^2 \, F \) is negative definite, if for all \( (X-X') \), \( Q(X) < 0 \)

\( \text{Grad}^2 \, F \) is negative semidefinite, if for all \( (X-X') \), \( Q(X) \leq 0 \)

\( \text{Grad}^2 \, F \) is indefinite, if for some \( (X-X') \), \( Q(X) > 0 \) and for some other \( (X-X') \), \( Q(X) < 0 \)
Therefore it follows that the necessary conditions for $X^*$ to be a local minima are:

$$\nabla F(X^*) = 0$$ \hspace{1cm} 1.17a

and

$$\nabla^2 F(X^*) \text{ is positive semi-definite}$$ \hspace{1cm} 1.17b

The sufficient conditions for $X^*$ to be an isolated local minima are:

$$\nabla F(X^*) = 0$$ \hspace{1cm} 1.18a

and

$$\nabla^2 F(X^*) \text{ is positive definite}$$ \hspace{1cm} 1.18b

The conditions required for $X^*$ to be a saddlepoint are that

$$\nabla F(X^*) = 0$$ \hspace{1cm} 1.19a

and

$$\nabla^2 F(X^*) \text{ is indefinite}$$ \hspace{1cm} 1.19b

Typically, the analyst is satisfied to find a local minimum, but when $(X - X^*)^\dagger \nabla^2 F(X^*) (X - X^*) \geq 0$, for all $X$, a local minimum is a global minimum.
Constrained Optimality Conditions

Now consider the general nonlinear optimization problem with constraints.

Minimize: \( F(X) \) 
Subject to: 
\( H_k(X) = 0 \) \hspace{1cm} \text{for } k = 1,2,\ldots,K \hspace{1cm} (1.20) 
\( G_j(X) \geq 0 \) \hspace{1cm} \text{for } j = 1,2,\ldots,J \hspace{1cm} (1.21) 
\( X_i \) \hspace{1cm} \text{for } i = 1,2,\ldots,N \hspace{1cm} (1.22) 

The constrained problem expressed by equations (1.20) through (1.23), can be converted to an unconstrained problem by using the method of Lagrange multipliers to obtain function \( L(X,u,v) \), given by equation (1.24). In this new problem \( u_j \) and \( v_k \) are unspecified constants called Lagrange multipliers.

\[ L(X,u,v) = F(X) - \sum_{j=1}^{J} [u_j G_j(X)] - \sum_{k=1}^{K} [v_k H_k(X)] \hspace{1cm} (1.24) \]

Assuming that functions \( F, G_j \) and \( H_k \) are all differentiable, the necessary and sufficient optimality conditions for the general constrained nonlinear problem are called the Kuhn-Tucker conditions (KTC) (Reklaitis et. al. 1983). The conditions may be expressed in the form of finding a solution to the following system of nonlinear equations.
Grad $L(X,u,v) = 0$  

Grad $F(X) - \sum_{j=1}^{J} [u_j \text{ Grad } G_j(X)] - \sum_{k=1}^{K} [v_k \text{ Grad } H_k(X)] = 0$  

where: $G_j(X) \geq 0$ for $j=1,2,\ldots,J$  
\quad $H_k(X) = 0$ for $k=1,2,\ldots,K$  
\quad $u_j G_j(X) = 0$ for $j=1,2,\ldots,J$  
\quad $u_j \geq 0$ for $j=1,2,\ldots,J$  

The Kuhn-Tucker problem is to find the $N$-vector $X$, the $J$-vector $u$ and the $K$-vector $v$ that satisfy equations (1.26) through (1.30). This point is called a Kuhn-Tucker point or a Lagrangian stationary point. Equation (1.28) is true for all feasible points. Equation (1.29) is true for all active constraints, that is constraints for which

$$G_j(X) = 0$$

For constraints that are inactive, that is constraints for which equation (1.31) fails, $u_j$ must be zero in order for equation (1.29) to hold. This in effect removes inactive constraints from the problem.

Equation (1.26) implies that the gradient of the objective function can be expressed as a linear combination of the constraint surface gradients. Graphically this means that the Grad $F(X)$ vector must lie within the cone formed by the constraint-surface normal vectors (Gallagher, 1973). See
Figure 1. The values of $u_j$ and $v_k$ are scalar multipliers that allow the constraint normals to be combined to equal the objective function gradient. The condition expressed by equation (1.30) assures that by forcing all $u_j$ to be positive, the constraint-surface normals are pointing in the infeasible direction. Note, there is no restriction on the sign if $v_k$, since all feasible points must remain on the equality constraint surfaces. Therefore, the orientation of their constraint normals are not critical. The requirement that the gradient of the objective function must lie within the cone of constraint-surface normals therefore, simply means that there is no component of the descent direction parallel to the constraint surfaces. Since there is no direction parallel to the constraint surfaces that will yield objective function improvement, and the constraints prevent motion in the direction that the objective function gradient indicates, this point is a local optima.

The conditions defined by equations (1.26) through (1.30), combined with the requirement that equations

$$\nabla G_j(X^\dagger) \quad \text{for } j=1,\ldots,J$$  \hspace{1cm} 1.32

and

$$\nabla H_k(X^\dagger) \quad \text{for } k=1,\ldots,K$$  \hspace{1cm} 1.33

are linearly independent, are the Kuhn-Tucker necessary conditions for constrained optimality. Of course, this is
Figure 1a. Design point is not a minimum.

Figure 1b. Design point is a minimum. The Kuhn-Tucker Conditions hold.
difficult to prove in practice because $x^*$ must be known in advance! However, for problems where a practical optimum solution is considered an improvement over present conditions, this is often adequate.

The Kuhn-Tucker sufficiency conditions are equations (1.26) through (1.30), (1.32) and (1.33) combined with the requirements that the equality constraints be linear, the objective function is convex and the inequalities are all concave. Since most problems do not possess these properties, verification of the Kuhn-Tucker sufficiency conditions is very difficult.

In practice, if the algorithm converges to the same solution from numerous starting points, and if that point satisfies the Kuhn-Tucker necessary conditions, the point is assumed to be a global optima. Confidence in this assumption depends on the robustness of the algorithm used, the complexity of the problem and the number of initial starting points used. Again, if a practical solution is the goal of the optimization study, this may be accurate enough.
1.5 List of Symbols

Listed below are the symbols used in this thesis along with a description of each and the symbol's equivalent FORTRAN name, where applicable. Note, N/A is an abbreviation for not applicable. The FORTRAN variables are also listed in solution Templates 1 and 2, which can be found in Appendices B.5 and C.5, respectively.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FORTRAN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>XLOWER</td>
<td>J-vector, lower limits on inequalities, $A(1, 2, \ldots, J) = XLOWER(N+1, N+2, \ldots, N+J)$</td>
</tr>
<tr>
<td>B</td>
<td>XUPPER</td>
<td>J-vector, upper limits on inequalities. $B(1, 2, \ldots, J) = XUPPER(N+1, N+2, \ldots, N+J)$</td>
</tr>
<tr>
<td>C</td>
<td>GRDCON</td>
<td>Matrix of constraints gradients with respect to all of the variables.</td>
</tr>
<tr>
<td>C</td>
<td>JMAT</td>
<td>Matrix of constraints gradients with respect to the basic variables.</td>
</tr>
<tr>
<td>C</td>
<td>CMAT</td>
<td>Matrix of constraints gradients with respect to the non-basic variables.</td>
</tr>
<tr>
<td>CON</td>
<td>CON</td>
<td>Constraint vector constructed from $[H_k, G_j] H_k \ k=1, \ldots, K \ &amp; \ G_j \ j=1, \ldots, J$</td>
</tr>
<tr>
<td>CON</td>
<td>N/A</td>
<td>Basic partition of constraint vector constructed from CON.</td>
</tr>
<tr>
<td>CON</td>
<td>N/A</td>
<td>Nonbasic partition of constraint vector constructed from CON.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>CON'</td>
<td>Linearized approximation of the constraint vector constructed from CON.</td>
<td></td>
</tr>
<tr>
<td>CYCLE</td>
<td>Maximum number of times a problem is restarted at the optimum solution with reduced tolerances.</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>Search vector.</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>Basic search vector.</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>Nonbasic search vector.</td>
<td></td>
</tr>
<tr>
<td>F(X)</td>
<td>Objective function.</td>
<td></td>
</tr>
<tr>
<td>FACTOR</td>
<td>Factor by which tolerances are divided after successful completion of the algorithm.</td>
<td></td>
</tr>
<tr>
<td>GAMMA</td>
<td>Factor used to reduce step size $\alpha$, when Newton's iterations fail to produce a feasible point.</td>
<td></td>
</tr>
<tr>
<td>Grad F</td>
<td>Gradient of $F(X)$ with respect to $X$.</td>
<td></td>
</tr>
<tr>
<td>Grad$^2$ F</td>
<td>Hessian of $F(X)$ with respect to $X$.</td>
<td></td>
</tr>
<tr>
<td>Grad F</td>
<td>Gradient of $F$ with respect to basic variables.</td>
<td></td>
</tr>
<tr>
<td>Grad F</td>
<td>Gradient of $F$ with respect to nonbasic variables.</td>
<td></td>
</tr>
<tr>
<td>Grad F'</td>
<td>Reduced gradient vector.</td>
<td></td>
</tr>
<tr>
<td>Gj</td>
<td>$j^{th}$ inequality constraint.</td>
<td></td>
</tr>
<tr>
<td>Hk</td>
<td>$k^{th}$ equality constraint.</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Number of inequality constraints.</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Number of equality constraints.</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>MAXIT</td>
<td>Maximum number of algorithm iterations allowed.</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Number of original design variables.</td>
<td></td>
</tr>
<tr>
<td>NEWTIT</td>
<td>Maximum number of Newton's iterations allowed.</td>
<td></td>
</tr>
<tr>
<td>TOL1</td>
<td>Tolerance used for testing norms of reduced gradient and the search vector.</td>
<td></td>
</tr>
<tr>
<td>TOL2</td>
<td>Tolerance used for testing feasibility of constraints, length K+J.</td>
<td></td>
</tr>
<tr>
<td>TOL3</td>
<td>Tolerance used for testing convergence of Newton's iterations, proximity to variable bounds and uniqueness of consecutive solutions, length N+J.</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>(N+J)-vector of design variables.</td>
<td></td>
</tr>
<tr>
<td>XBAS</td>
<td>Basic partition of X.</td>
<td></td>
</tr>
<tr>
<td>XNBAS</td>
<td>Nonbasic partition of X.</td>
<td></td>
</tr>
<tr>
<td>XLOWER</td>
<td>(N+J)-vector of lower bounds on X.</td>
<td></td>
</tr>
<tr>
<td>XLOWER</td>
<td>(N+J)-vector of upper bounds on X.</td>
<td></td>
</tr>
<tr>
<td>X₀</td>
<td>Initial feasible design variables.</td>
<td></td>
</tr>
<tr>
<td>X(i)</td>
<td>X vector from the i&lt;sup&gt;th&lt;/sup&gt; iteration.</td>
<td></td>
</tr>
<tr>
<td>X*</td>
<td>Optimum design variables.</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>Vector of nearest relative distances the components of X are from their bounds.</td>
<td></td>
</tr>
<tr>
<td>α₀</td>
<td>Initial step size.</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>Current step size.</td>
<td></td>
</tr>
</tbody>
</table>
Listed below are symbols used in the context of decomposition problems. Substitute the subsystem number for $s$. The FORTRAN variables used in two-level decomposition problems are also listed in solution Template 2, which can be found in Appendix C.5.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FORTRAN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYCLE$_s$</td>
<td>CYCLES</td>
<td>Maximum number of times a problem is restarted at the optimum solution with reduced tolerances.</td>
</tr>
<tr>
<td>FACTOR</td>
<td>FACTOR</td>
<td>Factor by which tolerances are divided after successful completion of the algorithm. (Same as for single level)</td>
</tr>
<tr>
<td>$F_s$</td>
<td>Fs</td>
<td>Objective function associated with subsystem $s$.</td>
</tr>
<tr>
<td>$F_{sys}$</td>
<td>FS</td>
<td>System level objective function.</td>
</tr>
<tr>
<td>$H_{sys}$</td>
<td>HCONS</td>
<td>All system level equality constraints.</td>
</tr>
<tr>
<td>$H_{s,k}$</td>
<td>HCONS</td>
<td>Equality constraint $k$ of subsystem $s$.</td>
</tr>
<tr>
<td>$G_{sys}$</td>
<td>GCONS</td>
<td>All system level inequality constraints.</td>
</tr>
<tr>
<td>$G_{s,j}$</td>
<td>GCONS</td>
<td>Inequality constraint $j$ of subsystem $s$.</td>
</tr>
<tr>
<td>N/A</td>
<td>JS</td>
<td>Total number of system level inequality constraints ($J_{sys}+J'_{sys}$).</td>
</tr>
<tr>
<td>$J_s$</td>
<td>Js</td>
<td>Number of inequality constraints in subsystem $s$.</td>
</tr>
<tr>
<td>$J_{sys}$</td>
<td>N/A</td>
<td>Number of inequality constraints exclusive to system level sub-problem.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>$J'_{sys}$</td>
<td>N/A Total number of inequality constraints that originate in the subsystems.</td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>KS Total number of system level equality constraints ($K_{sys}+K'_{sys}$).</td>
<td></td>
</tr>
<tr>
<td>$K_s$</td>
<td>Ks Number of equality constraints in subsystem $s$.</td>
<td></td>
</tr>
<tr>
<td>$K_{sys}$</td>
<td>N/A Number of equality constraints exclusive to system level sub-problem.</td>
<td></td>
</tr>
<tr>
<td>$K'_{sys}$</td>
<td>N/A Total number of equality constraints that originate in the subsystems.</td>
<td></td>
</tr>
<tr>
<td>MAXIT$_s$</td>
<td>MAXITs Maximum number of algorithm iterations allowed in sub-problem $s$.</td>
<td></td>
</tr>
<tr>
<td>NEWTIT</td>
<td>NEWTIT Maximum number of Newton's iterations allowed. (Same as for single level)</td>
<td></td>
</tr>
<tr>
<td>$N_s$</td>
<td>Ns Number of design variables in subsystem $s$.</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>N/A Total number of subsystems in a decomposition problem.</td>
<td></td>
</tr>
<tr>
<td>TOL$_s$</td>
<td>TOLs1 Tolerance used for testing norms of reduced gradient and the search vector within subsystem $s$.</td>
<td></td>
</tr>
<tr>
<td>TOL$_s$</td>
<td>TOLs2 Tolerance for testing feasibility of constraints in subsystem $s$, length $K_s+J_s$.</td>
<td></td>
</tr>
</tbody>
</table>
TOL_{s3}  TOL_{s3}  Tolerance used for testing convergence of Newton's iterations, proximity to variable bounds and uniqueness of consecutive solutions in subsystem s, length N_{s}+J_{s}.

TOL_{4}  TOL_{4}  Tolerance used to test the convergence of the system level objective function.

X  N/A  Vector of all subsystem level design variables \( X = (X_1, X_2, \ldots, X_s) \).

X_{Ls}  XLOWS  Vector of variable lower bounds for subsystem s.

X_{Us}  XUPs  Vector of variable upper bounds for subsystem s.

X_{s}  X  Sub-vector of X, representing the design variables in subsystem s.

XCOM(s,i)  The \( i^{th} \) component of the design variable vector \( X_s \)

\[
XCOM = \begin{bmatrix}
X_{1,1} & X_{1,2} & \cdots & X_{1,N_{1}+J_{1}} \\
X_{2,1} & X_{2,2} & \cdots & X_{2,N_{2}+J_{2}} \\
\vdots & \vdots & \ddots & \vdots \\
X_{s,1} & X_{s,2} & \cdots & X_{s,N_{s}+J_{s}} 
\end{bmatrix}
\]

Y  Y  System level design variables.
Chapter 2: Development of the Generalized Reduced Gradient Algorithm

The Generalized Reduced Gradient optimization method was selected because its implementation is relatively straightforward, it develops only feasible points as intermediate solutions and it is one of the most widely used nonlinear optimization methods in engineering (Reklaitis et al., 1983). A useful implication of this second feature is that the succession of objective function values associated with the sequence of feasible points decreases monotonically. Therefore, the iterations can be stopped at any time, resulting in a feasible point with an objective function value superior to the initial point. This is advantageous when a practical optimum will be accepted if the search for the theoretical optimum has stalled. As will be shown in Chapter 3, this feature is key to the success of the decomposition method presented.

In this chapter the Generalized Reduced Gradient method proposed by Abadie and Carpentier (1969) will be presented. The formulation follows the construction of the Generalized Reduced Gradient algorithm given by Reklaitis, Ravindran and Ragsdell in Engineering Optimization: Methods and Applications (1983, pp. 377-402). The sequence and some of the terminology have been altered for convenience, but the theory is the same. The key features of this algorithm are summarized below and
explained in more detail in the sections that follow.

1) This optimization method uses implicit variable elimination to remove the constraints and to reduce the problem's dimension. This is accomplished by first partitioning the vector of design variables into independent and dependent variables. Then, by using linearizations of the constraints to solve for the dependent or basic variables in terms of the independent or nonbasic variables, the new reduced objective function depends only on the independent variables.

2) The nonbasic search vector is constructed from the reduced gradient, the gradient of the objective function in the reduced space. The basic search vector is then modified with Newton's method to produce feasible points.

3) To account for variable bounds, the variables are sorted according to the nearest relative distances from their bounds to define the basis. Components of the search vector are set to zero if they point beyond a particular bound into the infeasible region. Newton's iterations are modified to ensure the resulting point is within the bounds.
4) The basic/nonbasic variable partition is updated regularly to improve convergence of the algorithm. This is accomplished with a procedure that methodically changes which variables are in basis whenever progress has stalled.

One key aspect of the generalized reduced gradient algorithm developed here is that the iterative optimization process must start with a feasible point. The initial vector of design variables must satisfy all of the constraints and no variable bounds may be violated. Though it may require considerable effort to generate even a single feasible point for some problems, this requirement has clear the advantage stated earlier.

This optimization algorithm is both demonstrated and verified with an example involving minimizing the weight of a two bar truss.
2.1 Implicit Variable Elimination

Consider the following nonlinear equality constrained optimization problem expressed in standard form. Note the inequality constraints have been converted to equality constraints through the introduction of slack variables.

Minimize:  \( F(X) \)  
Subject to:  \( H_k(X) = 0 \) for \( k = 1,2,\ldots,K \)  
\( G_j(X) - X_{j^\#} = 0 \) for \( j = 1,2,\ldots,J \)  
\( X^I_i \leq X_i \leq X^U_i \) for \( i = 1,2,\ldots,N \)  
\( A_j = X^L_j \leq X_j \leq X^U_j = B_j \) for \( j = N+1,\ldots,N+J \)  
\( X = (X^I_1,X^I_2,\ldots,X^I_N) \)

If any of the equality constraints are simple enough so that it is possible to solve for any one variable, it is prudent to do so. This subsequently eliminates that variable from the problem and thereby reduces both the problem's dimension and the number of constraints. If however, the equality constraints are too algebraically complex to explicitly solve for any variables, then the problem's dimension can be reduced through implicit variable elimination.

First assemble a constraint vector \( CON \), constructed from the \( K \) original equality constraints and the \( J \) converted inequality constraints. This combined constraint vector is
defined as follows

\[ \text{CON} = [ H_1, H_2, \ldots, H_K, G_1, G_2, \ldots, G_J ]^T \]  

Construct a linear approximation of the constraints, (equation 2.8), at \( X^{(1)} \), a point that satisfies the \( K+J \) equality constraints.

\[ \text{CON}'(X,X^{(1)}) \approx \text{CON}_k(X^{1}) + \text{Grad CON}_k(X^{(1)})(X - X^{(1)}) \]

for \( k = 1, 2, \ldots, K+J \)  

Note that since point \( X^{(1)} \) is feasible, it therefore satisfies the constraints, and the constraint term at \( X^{(1)} \) vanishes.

\[ \text{CON}_k(X^{1}) = 0 \quad \text{for} \ k = 1, 2, \ldots, K+J \]  

If this set of linearized equations (2.8) is used to predict the location of another feasible point, that is, a point where

\[ \text{CON}'_k(X,X^{(1)}) = 0 \quad k = 1, 2, \ldots, K+J \]  

then the new point \( X \) must satisfy the following set of equations, denoted equation (2.11). It is important to note

\[ \text{Grad CON}_k(X^{(1)})(X - X^{(1)}) = 0 \quad \text{for} \ k = 1, 2, \ldots, K+J \]
that although the new point will satisfy the linearizations of the constraints, it may not be feasible. That is, it may not satisfy the original constraints or it may violate the variable bounds. At this time, ignore the variable bounds as they will be dealt with when the search vector is constructed and during the Newton's iterations.

Equation (2.11) is a set of $K+J$ equations with $N+J$ unknowns. This system of equations usually will have more unknowns than equations ($K+J < N+J$), since the number of constraints ($K+J$) is generally less than the number of unknowns ($N+J$) in the problem. Consequently, this system has no unique solution. However, $K+J$ of the variables can be solved for in terms of the remaining $N-K$ variables. Select the first $K+J$ of the original $N+J$ variables and label them $X$ or basic. The remaining $N-K$ variables are then labeled $\bar{X}$ or nonbasic. Corresponding to this partition of $X$, partition the row vectors of $\text{Grad CON}_k$ into $\text{Grad CON}_1$ and $\text{Grad CON}_2$, then collect these row vectors into matrix $C$ of order $K+J$ by $K+J$ and matrix $\bar{C}$ of order $K+J$ by $N-K$, respectively.

\[
X = [X, \bar{X}]^T
\]

\[
C = \begin{bmatrix}
\text{Grad CON}_1 \\
\text{Grad CON}_2 \\
\vdots \\
\vdots \\
\text{Grad CON}_{K+J}
\end{bmatrix}
\]

\[(K+J \times K+J)\]
The system of equations defined by (2.11) can be rewritten with the matrices $C$ and $C$ in the following manner.

\[
C \begin{bmatrix}
\text{Grad } \text{CON}_1 \\
\text{Grad } \text{CON}_2 \\
\vdots \\
\text{Grad } \text{CON}_{K+J}
\end{bmatrix}
= \begin{bmatrix}
\text{Grad } \text{CON}_1 \\
\text{Grad } \text{CON}_2 \\
\vdots \\
\text{Grad } \text{CON}_{K+J}
\end{bmatrix}
\quad (K+J \times N-K)
\]

If square matrix $C$ is invertible, this set of equations (2.15) can be solved for the $K+J$ $\bar{x}$ variables in terms of the $N-K$ $\bar{x}$ variables.

\[
C \left( \bar{x} - \bar{x}^{(1)} \right) + C \left( \bar{x} - \bar{x}^{(1)} \right) = 0
\]

\[
\bar{x} = \bar{x}^{(1)} - C^{-1} C \left( \bar{x} - \bar{x}^{(1)} \right)
\]

Given any set of nonbasic $\bar{x}$ variables, matrix equation (2.16) will yield a first order approximation of the basic variables $\bar{x}$ that satisfy the linear approximations to the original constraints in the vicinity of the feasible point $\bar{x}^{(1)}$. This linearization has essentially allowed implicitly solving the constraints for $K+J$ variables even though it was impossible to do so explicitly with the original constraints. This result can be used to reduce the problem's dimension from the $N+J$ original variables to the $N-K$ nonbasic variables $\bar{x}$. Therefore substituting equation (2.16) into the objective
function \( F(X) = F(X, X) \), thereby eliminates the basic variables \( X \), reducing the original objective function \( F \) to a function \( F' \), that depends on only the \( N-K \) nonbasic variables \( \bar{X} \).

\[
F'(X, \bar{X}) = F(X^{(1)} - C^{-1} \ C \ (\bar{X} - X^{(1)}), \bar{X}) \tag{2.17}
\]

If equation (2.17) is used as the new objective function and the variable bounds on \( X \) are ignored at this time, the problem becomes one of unconstrained optimization in the nonbasic variable \( \bar{X} \). The first order condition for a local minimum of the unconstrained function \( F' \) is that the gradient of \( F' \) with respect to \( \bar{X} \) be zero. Since \( F' \) is a function of \( \bar{X} \) and \( X \), and \( \bar{X} \) is also a function of \( X \),

\[
F'(\bar{X}) = F'(X(\bar{X}), \bar{X}) \tag{2.18}
\]

the chain rule must be used to find \( d \ F'/d \ \bar{X} \).

\[
d \ F'/d \ \bar{X} = d \ F/d \ \bar{X} + (d \ F/d \ X) \cdot (d \ X/d \ \bar{X}) \tag{2.19}
\]

From equation (2.16), an expression for \( d \ X/d \ \bar{X} \) can be derived.

\[
d \ X/d \ \bar{X} = \ -C^{-1} \ C \tag{2.20}
\]

Making the following substitutions into equation (2.19)
an expression for Grad \( F' \), called the reduced gradient vector, can be obtained.

\[
\text{Grad} \ F' = \text{Grad} \ F - \text{Grad} \ L (c' - c)
\]

The first-order necessary conditions are thus

\[
\text{Grad} \ F' = 0
\]

or

\[
\text{Grad} \ F - \text{Grad} \ L (c' - c) = 0
\]

It now will be demonstrated that reduced gradient optimality criterion for the equality constrained problem is equivalent to the Lagrangian optimality criterion. Therefore, the solutions to the reduced gradient equation are the Lagrangian stationary points. Recall that the Lagrangian necessary conditions (Eq. 1.26) with \( \text{CON} \) substituted for \( H \) are

\[
\text{Grad} \ F(x^\dagger) - [v^\dagger]^T \text{Grad} \ \text{CON}(x^\dagger) = 0
\]

where \( \text{Grad} \ \text{CON} \) is the matrix of constraint gradients. Now rewrite this the system of equations substituting the defin-
iterations of basic and nonbasic variables $X$ and $\underline{X}$ along with the $C(X^t)$ and $\underline{C}(X^t)$ partitions of the $\text{Grad CON}(X^t)$ matrix.

\[
\text{Grad } F(X^t) - [v^t]^T C(X^t) = 0 \quad 2.28 \\
\text{Grad } \underline{F}(X^t) - [v^t]^T \underline{C}(X^t) = 0 \quad 2.29
\]

Solving for $v^t$ in the first equation (2.28) yields

\[
[v^t]^T = \{\text{Grad } F(X^t)\} C^{-1}(X^t) \quad 2.30
\]

which when substituted into the second equation (2.29) generates

\[
\text{Grad } \underline{F}(X^t) - \{\text{Grad } F(X^t)\} (C^{-1}(X^t) \underline{C}(X^t)) = 0 \quad 2.31
\]

an expression identical to equation (2.26). It is useful to note that the Lagrange multipliers are determined automatically when evaluating the reduced gradient.

In summary, by using linear approximations to the constraints two things have been accomplished:

1. A set of linear equations have been obtained that provide an estimate of values of $X$, the basic variables, corresponding to small perturbations of $X$, the nonbasic variables, about the point $X^{(1)}$ yielding a feasible point.

2. A first-order necessary condition has been derived for
the equality constrained problem that implicitly accounts for the equality constraints.

2.2 Development of the Search Vector

As a consequence of the linearizations of the previous section, the problem can be treated as an unconstrained optimization problem. This however, is limited to small excursions about the linearization point. In order to move through the design space towards the optimum solution, a search vector that leads to an improvement in the objective function is needed. The search vector \( d \), is partitioned with the basic and nonbasic variables as follows

\[
\mathbf{d} = (\mathbf{d}_b, \mathbf{d}_n)^T
\] 2.32

An obvious choice for the nonbasic search direction is that of steepest descent, the negative of the reduced gradient.

\[
\mathbf{d}_n = (-\text{Grad} F')^T
\] 2.33

The basic search vector is implicitly determined by the constraint linearizations (Eq. 2.17) to eliminate the \( X \) variables.
The current point at iteration $t$ is referred to as $x^{(t)}$ and the new point generated at this iteration is $x^{(t+1)}$. If the step size in the direction of the search vector is controlled by a scalar parameter $\alpha$, then $x^{(t+1)}$ is determined by the following expression.

$$x^{(t+1)} = x^{(t)} + \alpha \mathbf{d} \quad 2.35$$

By using a first-order Taylor series expansion of equation (2.17), it can be shown that regardless of the nonlinearity, the direction $\mathbf{d}$ is always a decent direction.

$$F(x^{(t+1)}) - F(x^{(t)}) \approx F'(x^{(t+1)}) - F'(x^{(t)})$$

$$\approx \text{Grad } F'(x^{(t)})(x^{(t+1)} - x^{(t)}) \quad 2.37$$

$$\approx \alpha \text{ Grad } F'(x^{(t)}) \mathbf{d} \quad 2.38$$

So if $\mathbf{d}$ is selected as using equation (2.33), it follows that

$$F(x^{(t+1)}) - F(x^{(t)}) \approx \alpha [\text{Grad } F'(x^{(t)})][ - \text{Grad } F'(x^{(t)})]^T \quad 2.39$$

$$\approx -\alpha |\text{Grad } F'(x^{(t)})|^2 \quad 2.40$$

The right-hand side of equation (2.40) must be less than zero for all positive values of $\alpha$. Therefore, for all values
of a greater than zero and small enough so that the linearizations of the constraints are sufficiently accurate, it follows that $F(X^{(t+1)})$ must always be less than $F(X^{(t)})$.

$$F(X^{(t+1)}) - F(X^{(t)}) < 0$$  \hspace{1cm} 2.41

$$F(X^{(t+1)}) \approx F(X^{(t)}) - \alpha |\text{Grad } F'(X^{(t)})|^2$$  \hspace{1cm} 2.42

According to this argument, it should be clear that the search vector is always a descent direction.

It has been shown that the search vector $d$ is a descent direction, but will points generated in that direction with equation (2.35) be feasible? In general, almost all the points on the tangent line or the tangent plane formed by the linearizing the constraints will be infeasible if the constraints are concave. In Figure 2, the only feasible point on the tangent plane is the linearization point itself! This means that although $d$ is a descent direction, it leads to infeasible points. Equations (2.36) through (2.42) actually prove that only the $d$ partition of vector $d$ is a descent direction in the space of the nonbasic variables. Since the composite search vector $d$ has undesirable properties, the problem must be with the implicit determination of the basic search vector $d$. 
A graphical example best illustrates the short-comings of the basic search vector $d$ and how to remedy this problem. Consider the problem shown in Figure 3, with a single spherical constraint surface.

$$H_1(x_1, x_2, x_3) = (x_1)^2 + (x_2)^2 + (x_3)^2 - 1 = 0$$

The linearization of this constraint at $x^{(0)}$ yields a plane tangent to the surface at $x^{(0)}$. Suppose $x_1$ and $x_2$ are selected as the nonbasic variables, leaving $x_3$ as the basic variable.
Furthermore, suppose that the unspecified objective function in this problem produces the search vector $d$ as shown in the figure. The nonbasic search vector $\underline{d}$, is simply the projection of total search vector $d$ on the $X_1$-$X_2$ plane, while the nonbasic search vector $d$, is the projection of $d$ on the $X_3$ axis. The points designated as $V^{(1)}$, $V^{(2)}$ and $V^{(3)}$ are new trial points obtained by moving in the $d$ direction, from $X^{(0)}$ for various values of $\alpha$. These trial points can be partitioned into their basic and nonbasic components.
\[ v^{(i)} = (v^{(i)}, v^{(i)}) \]

Though the trial points satisfy the constraint linearizations, it is clear that none satisfy the original constraint equation, that is,

\[ H(v^{(1)}) \neq 0 \]
\[ H(v^{(2)}) \neq 0 \]
\[ H(v^{(3)}) \neq 0 \]

\( v^{(1)} \), \( v^{(2)} \) and \( v^{(3)} \) are not feasible. While the nonbasic components of trial point \( v^{(i)} \) are determined with

\[ v^{(i)} = x^{(t)} + \alpha d \]

the approximations of the basic components generated with

\[ v^{(i)} = x^{(t)} + \alpha d = -C^{-1} \sum (v^{(i)} - x^{(0)}) \]

are not accurate enough to produce values of \( v^{(i)} \) that satisfy the original equality constraint equation exactly.

\[ H(v^{(i)}, x^{(i)}) \neq 0 \]

In Figure 3, only a step size of length zero would produce a trial point that satisfies the original constraint!
This implies that although the linearizations are good enough to determine local descent directions, they are not good enough to actually calculate new feasible points. By performing a search for the minimum objective function value, along the fixed line defined by the d vector, there is little chance of finding a feasible minimum. However, if \( d \) is projected onto the constraint surface, and the minimization is performed along the resulting curve, the search will be performed in a descent direction and will result in feasible points. This implicitly defined curve is determined by the set of \( a \) and \( V_\alpha \) values that satisfy equation (2.51).

\[
H(V, (x^{(0)} + \alpha d)) = 0 \quad 2.51
\]

This is the curve labeled Feasible Projection of \( d \) in Fig. 3, that passes through points \( x^{(0)}, x^{(1)} \) and \( V^{(2)} \). Suppose that

\[
x = (0.1, 0.1)^T \quad 2.52
\]

and

\[
d = (1,1)^T \quad 2.53
\]

then the implicitly defined curve is given by the set of \( a \) and \( V=x_3 \) values that satisfy

\[
(0.1 + a)^2 + (0.1 + a)^2 + (x_3)^2 - 1 = 0 \quad 2.54
\]
In the general case of multiple constraints, for every trial value of step size $\alpha$ selected, the constraint set will have to be solved for the basic variables $V$, that cause the resulting point to be feasible.

This can be accomplished by using a numerical method that, for a given step size $\alpha$, can solve the constraints for the values of $V$ that satisfy the constraints. Newton's method is an efficient iterative technique for solving the set of $K+J$ constraints for the $K+J$ basic variables $V$. Recall that Newton's method uses a truncated Taylor's expansion about the current approximation to predict the next approximation to the solution. Using a first order Taylor's series expansion about $V[i]$ to get

$$CON(V) = CON(V[i]) + Grad CON(V[i]) (V[i] - V[i]) = 0 \quad 2.55$$

Recall from equation (2.13)

$$Grad CON(V[i]) = C(V[i]) \quad 2.56$$

and since

$$V[i] = (V[i], \bar{V}[i]) = (V[i], (\bar{X}[t] + \alpha\bar{d})) \quad 2.57$$

therefore

$$CON(V[i]) = CON(V[i], (\bar{X}[t] + \alpha\bar{d})) \quad 2.58$$

and

$$Grad CON(V[i]) = C(V[i], (\bar{X}[t] + \alpha\bar{d})) \quad 2.59$$
Note that during the Newton's iterations the nonbasic variables \( v^{(i)} = x^{(t)} + \alpha d \) are fixed when step size \( \alpha \) is selected. Therefore the gradient term in the Taylor series is with respect to the basic variables \( v \) only. An explanation of the notation used in equations (2.58) and (2.59) is required, the superscript \( t \) refers to the \( t^{th} \) iteration of the optimization algorithm, while the superscript \( i \) refers to the \( i^{th} \) Newton iteration. Finally, solving for \( v^{(iti)} \) yields the Newton iteration formula

\[
v^{(iti)} = v^{(i)} - [C^{-1}(v^{(i)}, (x^{(t)} + \alpha d))] \cdot [\text{CON}(v^{(i)}, (x^{(t)} + \alpha d))]
\]

or

\[
v^{(iti)} = v^{(i)} - [C^{-1}(v^{(i)})] \cdot [\text{CON}(v^{(i)})] \tag{2.60}
\]

For a given step size \( \alpha \), the Newton's iterations are allowed either to converge to a point or to iterate a maximum number of times. In the first case, the trial point must be checked for feasibility and for objective function improvement over \( x^{(t)} \). If the trial point is not feasible, if there is no objective function improvement or if the maximum number of Newton's iteration was reached, the step size was probably too large and should be reduced (usually halved) and the Newton's iteration restarted. For example if Newton's method is started with the point \( v^{(i)} \) shown in Figure 3, it is impossible to satisfy the constraint because there is no value of \( v^{(i)} \) corresponding to the nonbasic variables \( v^{(i)} \) that will
intersect the constraint. The step size needs to be reduced to some value that will yield a better starting point like $V^{(1)}$ or $V^{(2)}$.

If Newton's method converges to a feasible point that offers objective function improvement over $X^{(t)}$, then the step size can be enlarged and Newton's methods restarted. Regardless, in practice it is more common to accept the current point and continue. Of course convergence to the optimum solution is quicker if exact searches on the step size $\alpha$ are performed. However, since the computational effort required in this algorithm is concentrated in the Newton's iterations, exact searches on $\alpha$ are seldom worth the effort (Reklaitis et. al. 1983). The algorithm developed in this thesis accepts the first feasible point that shows objective function improvement.

Up to this time the test for feasibility of points has only involved their satisfaction of the constraint equations. In the next section variable bounds will be considered in defining the feasibility of points and in modifying the search vector when necessary.

### 2.3 Treatment of Variable Bounds

Upper and lower variable bounds can be handled either explicitly by adding an inequality constraint for each
variable or implicitly by dealing with them at the appropriate times throughout the algorithm. The second choice is clearly preferred in that it prevents the problem from getting bigger, which means a smaller matrix $C$ to invert. In order to implicitly incorporate variables bounds into this algorithm, three modifications must be made to the method developed thus far.

1. Basic Variable Selection

Logic must be added to the basis selection process to prevent variables that are at or near their bounds from entering the basis, ie becoming a basic variable. This check is required to ensure that the basic variables have some free adjustment at all times. This is accomplished by sorting the variables according to the distance to their nearest bound (Reklaitis et. al. 1983). To prevent a variable's relative size from distorting the sorting process, the nearest distance should be divided by the variable's range (the difference between $X^u$ and $X^l$). This allows all variables to be compared in a normalized manner. Let vector $Z_i^{(t)}$ represent the distance to the nearest bound of the $i^{th}$ component of $X$, divided by the difference between the upper bound and the lower bound.

$$Z_i^{(t)} = \min[(X_i^u - X_i^{(t)}), (X_i^{(t)} - X_i^l)] / (X_i^u - X_i^l) \quad 2.61$$

for $i = 1, 2, \ldots, N + J$
The components of vector $Z^{(t)}$ can than be sorted from largest to smallest and the variables corresponding to the largest $K+J$ components can be selected as the basic variables. This selection process may have to been overruled if the matrix $C$ associated with this basis is singular. In that case, one or more of the first $K+J$ variables may be rejected before a non-singular matrix $C$ can be found. This can be accomplished by swapping the variable associated with the column of $C$ that had a zero on the main diagonal with the next best variable in the sorted vector $Z^{(t)}$. For example, the first time this occurs, switch the bad variable with the $(K+J+1)^{th}$ member of the $Z^{(t)}$ vector. The second time this occurs on the same iteration, swap the $(K+J+2)^{th}$ variable and so on until the $N-K$ nonbasic variables are exhausted.

2. Modification of the Search Vector

The nonbasic search vector needs to be modified to prevent the violation of nonbasic variable bounds. This is necessary in the event that a nonbasic variable is at or near its bounds and movement in the direction of the nonbasic search vector would lead to violation of the bound. This can be done by defining $d$ as follows

$$d_i = \begin{cases} 
0 & \text{if } x_i = x_i^0 \text{ and } (\text{Grad } F')_i < 0 \\
0 & \text{if } x_i = x_i^l \text{ and } (\text{Grad } F')_i > 0 \\
- (\text{Grad } F')_i & \text{Otherwise}
\end{cases}$$
This modification also ensures that if

\[ \mathbf{d}_i = 0 \text{ for all } i = 1, \ldots, N-K \]

then the Kuhn-Tucker conditions are satisfied at that point. This can be verified by intuitively by recalling that at a Kuhn-Tucker point there is no feasible descent direction. (Figure 1) If all of the components of \( \mathbf{d} \) are zero either because movement would violate a variable bound or because \((\text{Grad } F')_i\) itself is zero, then there is no descent direction whatsoever! The point, therefore, is a local optima.

3. Prevention of Bound Violation in the Newton's Iterations

Steps must be added to the Newton's iterations to ensure that the variable bounds are not exceeded either during the search on \( \alpha \) or during the Newton's iterations. This check can be incorporated by checking \( v^{(i)} \), the result of the Newton's iterations, for bound violations. If \( v^{(i)} \) violates a variable bound, then linear interpolation can be used to predict a new value of \( \alpha \) that will yield a feasible point. The new point will lie on the line segment between points \( x^{(t)} \) and \( v^{(1)} \). In this way, Newton's iterations can be restarted here with a better chance of converging to a point that does not violate that bound. Figure 4 depicts bound \( x_1^L \) violated by \( v^{(i)} \), the result of the Newton's iterations started at \( v^{(1)} \). Note that \( \alpha \) is the step size that produced \( v^{(1)} \). By using linear
interpolation, an estimate for $\alpha^*$, the step size that is associated with the new feasible starting point $v^{(l')}$, can be found with the following expressions.

$$
\frac{\alpha |d|}{X_2^{(t)} - v_2^{(i)}} = \frac{\alpha^* |d|}{X_2^{(t)} - X_2^L}
$$

$$
\alpha^* = \alpha \left[ \frac{X_2^{(t)} - X_2^L}{X_2^{(t)} - v_2^{(i)}} \right]
$$

$$
v^{(l')} = x^{(t)} + \alpha^* d
$$

Figure 4  Linear Interpolation to Satisfy Variable Bounds.
2.4 Basis Changes to Improve Convergence

Basis changes that alter the partition assist convergence especially in the final stages of the iterations when little change would normally occur to the sorted order of vector Z defined with equation (2.61) (Reklaitis et. al. 1983). There are other situations when basis changes are necessary, for example when matrix C is singular. In addition to the basis changes that result from sorting the Z vector, the basis is changed heuristically in three other situations: 1) when the sorted sequence of the Z vector fails to change for two consecutive iterations; 2) when the matrix C is singular; and 3) when the Newton's iterations have been restarted a maximum number of times, each failing the converge to a feasible point with objective function improvement.

When either situation 1) or 3) above occurs, the result is to switch the qth basic variable with the pth nonbasic variable. The goal is to try some, but not all, of the permutations of the basic variable combinations. Counters and logical decisions are used to index through the various basis combinations. The candidate nonbasic variable to enter the basis is controlled by the value of integer p. The value of p increases from 1 to N-K, thereby considering all of the nonbasic variables. The nonbasic variable can be rejected if it is near its bounds. The basic variable that is leaving the basis is determined by the value of integer q. Each time
the \((N-K)^{th}\) nonbasic variable has been tried and failed, the value if \(q\) is decremented. The value of \(q\) starts with the \((K+J)^{th}\) basic variable and counts backwards from \(K+J, K+J-1, K+J-2\) down to 1. When the first basic variable is reached the process terminates with an error stating that the basis combinations have been exhausted with no progress. A total of \((K+J)\times(N-K)\) basis permutations are attempted. For every basis combination attempted, the iteration is restarted. When situation 2) above occurs \((C^{-1}\) does not exist), the basis variable associated with the column of \(C\) that had a zero on the main diagonal is switched with the \(p^{th}\) nonbasic variable. The same rule as above applies in that the nonbasic variable can be rejected if it is near its bounds. The value of \(p\) is controlled in the same manner as well. If all of the nonbasic variables attempted result in failure, the process terminates with an error stating that a linearly independent basis could not be found.

These simple heuristics have proved adequate for the problems solved thus far with this algorithm.

2.5 Generalized Reduced Gradient Algorithm

Now that the basic features have been presented, the Generalized Reduced Gradient algorithm can be assembled. The actual FORTRAN program, called GRGOPT32, follows this pseudo-
code outline very closely. The source code for GRGOPT32 is located in Appendix A. The information provided to GRGOPT32 must be organized in a format outlined by Template 1. (Appendix B.5) An example problem is demonstrated in the next section.

Step 0.0 Given the following input parameters, convergence tolerances, initial feasible point, functions and subroutines:

Parameters:

- **N**: Number of independent variables.
- **K**: Number of equality constraints.
- **J**: Number of inequality constraints.
- **a^0**: Initial step size.
- **GAMMA**: Step size reduction parameter (use 0.5).
- **TOL1**: Convergence tolerance for reduced gradient & search vector norms.
- **TOL2**: Tolerance vector for feasibility of constraints (Length K+J).
- **TOL3**: Convergence tolerance vector for Newton iterations and tolerance for vector component deltas (Length N+J).
- **X^{(0)}**: Initial feasible point (Length N+J).
\(x^u\): Vector of upper bounds of \(X\) (Length \(N+J\)).

\(x^l\): Vector of lower bounds of \(X\) (Length \(N+J\)).

MAXIT: Maximum number of iterations.

NEWTIT: Maximum number of Newton iterations allowed.

**Functions:**

- \(F\): Objective function.

**Subroutines:**

- Grad \(F\): Gradient of objective function.
- \(H\): Equality constraints.
- Grad \(H\): Gradient of equality constraints.
- \(G\): Inequality constraints.
- Grad \(G\): Gradient of inequality constraints.

0.1 Declare variables, vectors, arrays, logical values and common block dimensions.

0.2 Test: Is problem size is too large for present array dimensions?

Yes: Stop, with warning about problem dimensions.

No: Continue.

0.3 Output initial feasible point & objective function value.

0.4 Initialize counters and flags.
0.5 Define the initial variable subscripts (start with sequential order). This is used to track the variable's addresses.

0.6 Determine variable's ranges $\text{RANGE} = x^u - x^l$.

0.7 Equate $X$ & $X^{(0)}$.

0.8 Define the function gradients with respect to the slack variables. Since these are constant throughout the solution sequence, they need only be defined once.

Step 1.0 Basis definition & determination of the reduced gradient.

1.1 Start of main iterations loop.

1.2 Sort the variables according to the normalized distance from their nearest bound (vector $Z$, Eq. 2.61). Select the first $K+J$ variables as the basis.

1.3 Basis update heuristics (only if required & terminate with warning when all combinations have been tried)

1.4 Construct the matrix $C$ (Eq. 2.13)

1.5 Test: Does $C^{-1}$ exist?

   No: Goto Step 1.3 to find a new basis.

   Yes: Continue.

1.6 Determine $X$, $x$, Grad $F$, Grad $F$, Grad $F$ and $C$ for the current basis definition.
1.7 Determine the reduced gradient Grad F' (Eq. 2.24) 
1.8 Test: Check for optimum solution 

\[ |\text{Grad } F'| < \text{TOL1} \]

Yes: Exit, an optimum solution has been found. 
No: Continue. 

Step 2.0 Construction of search vector. 

2.1 Determine the nonbasic search vector \( \mathbf{d} \) (Eq. 2.33). 
2.2 Modify \( \mathbf{d} \) to accommodate the variable bounds (Eq. 2.62) if necessary. 
2.3 Test: Check for optimum solution (Eq. 2.63) 

\[ |\mathbf{d}| < \text{TOL1} \]

Yes: Exit, an optimum solution has been found. 
No: Continue. 

2.4 Determine initial basic search vector \( \mathbf{d} \) (Eq 2.34). 

Step 3.0 Find a feasible point \( \mathbf{V} \), that satisfies the constraints, doesn't violate any variable bounds and shows objective function improvement relative to \( \mathbf{X}^{(t)} \). 

3.1 Let \( \alpha = \alpha^0 \) 
3.2 Determine \( \mathbf{V}^{(i)} = \mathbf{X}^{(t)} + \alpha \mathbf{d} \) (Eq. 2.35). 

Note, the number of times Newton's iterations can be restarted per basis is limited to
NEWTIT. When step size $\alpha$ has been reduced $\text{NEWTIT}$ times, Goto Step 1.3 to change basis. Otherwise continue.

3.3 Test: Check for variable bound violation?

Yes: Use linear interpolation to find $\alpha'$ with (Eq. 2.64), Goto Step 3.2

No: Continue.

3.4 Test: Is $V^{(i)}$ feasible?

$\text{CON}_i(V^{(i)}) \leq TOL2_i$ for $i = 1, \ldots, K+J$

Yes: Goto Step 3.7

No: Continue.

3.5 Perform a Newton iteration on basic partition of $V$.

\[ V^{(i+1)} = V^{(i)} - C^{-1}(V^{(i)} \cdot \text{CON}(V^{(i)})) \quad \text{(Eq. 2.60)} \]

3.6 Test: Check for convergence

\[ |V^{(i+1)}_m - V^{(i)}_m| \leq TOL3_m \text{ for all } m = 1, \ldots, N+J \]

YES: Test: Is $V^{(i+1)}$ feasible?

$\text{CON}_i(V^{(i+1)}) \leq TOL2$

for $i = 1, \ldots, K+J$

Yes: Goto Step 3.7

No: Reduce step size & restart

$\alpha = \alpha(\text{GAMMA}),$

Goto Step 3.2

No: Continue Newtons Iterations,

Goto Step 3.5
3.7 Test: Are any variable bounds violated?
   Yes: Use linear interpolation to find $\alpha$,
   (Eq. 2.64) Goto Step 3.2
   No: Continue.

3.8 Test: Check objective function improvement at $V^{(i+1)}$?
   Is $F(X^{(i)}) \leq F(V^{(i+1)})$
   Yes: Reduce step size & restart
       $\alpha = \alpha(\text{GAMMA})$, Goto Step 3.2
   No: Accept $V^{(i+1)}$ & Continue.

3.9 Print status report & let $X^{(i+1)} = V^{(i+1)}$

4.0 Test: Check for limit on maximum number of iterations
   Is $t \leq \text{MAXIT}$?
   Yes: Goto Step 1.0
   No: Terminate with warning.

With the development of the Generalized Reduced Gradient method's mathematical rationale and the formation of the algorithm now complete, the next step is to discuss how to solve an optimization problem with this method. The next section demonstrates the optimization of a simple two bar truss structure through the minimization of its weight.
2.6 Single-Level Example: Two Bar Truss

Consider the task of finding the minimum weight of the simple two-bar truss depicted in Figure 5 (Kirsch, 1975). This structure is subjected to a single 100 kN vertical load at the common joint C. The design variables are $X_1$, the vertical coordinate of the common joint and the cross sectional areas of the two struts, $X_1$ and $X_2$. The constraints on the system are a 100,000 kN/m² upper limit on the axial stress in the bars, an interval between 1 meter and 3 meters for the vertical coordinate and the bars' cross sectional areas must be positive. The performance criterion is the minimization of the volume of the truss members, since the volume is directly proportional to the weight. The objective function is simply the sum of each bar's cross sectional area times its respective length.

When design vector $X$ is defined as follows

\[ X_1 = \text{Cross sectional area of bar 1} \]
\[ X_2 = \text{Cross sectional area of bar 2} \]
\[ X_3 = \text{Vertical coordinate of joint C} \]

the objective function becomes

\[ F(X) = X_1 \left(16 + (X_3)^2\right)^{\frac{1}{2}} + X_2 \left(1 + (X_3)^2\right)^{\frac{1}{2}} \]
Expressions for the axial loads in the truss members, $F_1$ and $F_2$, are derived from the equations of equilibrium. The equilibrium condition in the $X$ (horizontal) direction yields an expression for $F_2$ in terms of $F_1$.

$$\Sigma_x = 0 = -4 \frac{F_1}{(16 + (X_3)^2)^{\frac{1}{2}}} + \frac{F_2}{(1 + (X_3)^2)^{\frac{1}{2}}}$$

$$F_1 = \frac{4}{3}F_2 \frac{(16 + (X_3)^2)^{\frac{1}{2}}}{(1 + (X_3)^2)^{\frac{1}{2}}}$$  \hspace{1cm} 2.66

Combining equation (2.66) with the equilibrium condition in the $Y$ (vertical) direction yields an expression for $F_2$.

$$\Sigma_y = 0 = -100 + X_3 \frac{F_1}{(16 + (X_3)^2)^{\frac{1}{2}}} + X_3 \frac{F_2}{(1 + (X_3)^2)^{\frac{1}{2}}}$$
Now substituting (2.67) into equation (2.66) produces an expression for $F_1$.

$$F_1 = 20 \left(16 + (X_3^2)^\frac{1}{3}\right) / X_3$$

Since stress $\delta = \text{Force/Area}$, equations (2.67) and (2.68) can be used to derive expressions for the axial stresses.

$$\delta_1 = G_1(X) = 20 \left(16 + (X_3^2)^\frac{1}{3}\right) / (X_3 X_1) \leq 100000.$$  
$$\delta_2 = G_2(X) = 80 \left(1 + (X_3^2)^\frac{1}{3}\right) / (X_3 X_2) \leq 100000.$$  

With the introduction of two slack variables $X_4$ and $X_5$, the problem now can be expressed in standard format.

Minimize:  
\[ F(X) = X_1 \left(16 + (X_3^2)^\frac{1}{3}\right) + X_2 \left(1 + (X_3^2)^\frac{1}{3}\right) \]  
Subject to:
\[ G_1(X) = 20 \left[\left(16 + (X_3^2)^\frac{1}{3}\right) / (X_3 X_1) - X_4\right] = 0 \]  
\[ G_2(X) = 80 \left[\left(1 + (X_3^2)^\frac{1}{3}\right) / (X_3 X_2) - X_5\right] = 0 \]  
\[ 0 \leq X_1 \leq 0.003 \text{ m}^2 \]  
\[ 0 \leq X_2 \leq 0.003 \text{ m}^2 \]  
\[ 1 \text{m} \leq X_3 \leq 3 \text{ m} \]  
\[ 0 \leq X_4 \leq 100000. \text{ kN/m}^2 \]  
\[ 0 \leq X_5 \leq 100000. \text{ kN/m}^2 \]
An initial feasible vector $x^{(0)}$, the optimum solution found $x^*$ and the objective function values at those points are listed below.

$$x^{(0)} = (0.001, 0.001, 3.0, 33333.33, 84327.4)^\top$$
$$F(x^{(0)}) = 0.0081623 \text{ m}^3$$

$$x^* = (0.0004472, 0.0008944, 2.0, 100000., 100000.)^\top$$
$$F(x^*) = 0.00400 \text{ m}^3$$

This agrees with the optimum solution found by Kirsch (1975). The problem's key definition parameters are listed in Appendix B.4. Appendix B contains detailed solutions to four other problems used in testing and verifying the GRGOPT32 FORTRAN program. Appendix B can be regarded as a complete users manual for solving single-level optimization problems with GRGOPT32, as it contains information on setting up the problem, advice on selection of key parameters, explanations of the output format and error messages.

This concludes the chapter of single-level optimization with Generalized Reduced Gradient. The next chapter is an introduction to an optimization technique called decomposition, where a problem is partitioned into subsystems or sub-problems with each solved separately. The GRGOPT32 program is employed as the work horse of the proposed method.
Chapter 3: Decomposition

This chapter begins with a discussion of basic concepts in multi-level decomposition of large optimization problems. Recent trends in decomposition are then described, revealing the motivation for a more direct approach. In the following section, the Model Coordination strategy proposed by Kirsch (1975), is outlined. In this method, the problem is partitioned into a set of sub-problems, each with its own design variables. Within each sub-problem, only the corresponding design variables are allowed to change, while all other design variables are held fixed. The set of sub-problems is repeatedly solved until the system level objective function converges. Since this strategy is more an organizational structure for decomposition of a structural system than an actual computational tool, a methodology is needed to find the optimum solution.

In the next section, a new decomposition methodology is presented which utilizes the Generalized Reduced Gradient program from Chapter 2 to solve each of the decomposed system's sub-problems. As stated in Chapter 2, the GRG algorithm develops a sequence of feasible points as intermediate solutions and the value of the objective function must decrease monotonically with that sequence. Therefore, the iterations can be stopped at any time, resulting with a feasible solution that will have an objective function value
superior to the initial value. This is particularly advantageous when a practical optimum will be accepted if the theoretical optimum can not be found. The new and significant feature employed by this decomposition method is as follows: by adjusting the maximum number of iterations per sub-problem and the initial step size, progress is limited in each of the sub-problems so that none strongly bias the direction that the overall solution takes. This causes the individual sub-problems to approach the system's optimum solution in a uniform manner.

Finally, the new method is demonstrated and verified by solving the single-level two bar truss problem from Chapter 2 as a two-level decomposed system.
3.1 Introduction to Multi-Level Decomposition:

Motivation for a Direct Approach

Optimum structural design by means of mathematical programming has been the subject of many studies in the past 25 years. Since most of the available algorithms function most efficiently when solving problems with a reasonable amount of variables and constraints, considerable effort has been directed to the development of new approaches to deal with large problems. One technique involves decomposition of the original problem into smaller and hopefully more manageable sub-problems (Kirsch & Moses, 1979). The basic philosophy behind this approach is that it is more efficient to solve several small problems many times than it is to solve one large problem once. For example, inverting a matrix of dimension $N$ requires $N^3$ arithmetic operations. It is more efficient to invert five $20 \times 20$ matrices requiring a total of $4 \times 10^4$ operations, than one $100 \times 100$ matrix requiring $10^6$ operations!

The concept of decomposing a design problem into sub-problems is appealing for several reasons. The original system can be partitioned into subsystems or substructures and each optimized separately. The partitioning can be implemented along intuitive lines based on the nature of the particular system or by the disciplines involved (Sobieski, 1982). Note that the subsystems need not be of the same
discipline. Suppose the engineering system to be optimized is a spacecraft and the goal is to minimize the total weight of the vehicle. Some of the subsystems may be structural, another could be a thermal system, while yet another could be the control system for the vehicle. In this manner, a different specialized computational tool could be employed for each of the disciplines with the common goal of minimizing weight of each subsystem (Kirsch & Moses, 1979). This work breakdown is analogous to the organizational structure of many aerospace engineering companies, where engineers tend to be specialized analysts in one particular area only (Sobieski, 1982).

The man-power required to find an optimal solution to a design optimization problem tends to be concentrated in initially setting up the problem. It would be desirable then if the decomposed problem could be set up with a level of difficulty comparable (or easier) to that of the original problem. Current decomposition methods found in the literature have levels of complexity that seem prohibitive. For example, methods proposed by Sobieski (1985), Bhatt and Ragsdell (1988) and Haftka (1984) all require second order derivatives of the objective functions and the constraints and employ penalty functions, meaning infeasible intermediate solutions will result. The desire for ease of problem formulation combined with the requirement for no infeasible intermediate solutions were the chief motivations for the
method developed in this thesis. By using the Generalized Reduced Gradient algorithm to solve the individual sub-problems, only feasible points are generated throughout the solution sequence, therefore all sub-problems yield feasible solutions. Since all intermediate solutions are feasible, should the iterations be terminated after a small number of cycles, a feasible and perhaps practical optimum solution will result (Kirsch & Moses, 1979). This is advantageous especially if a practical optimum is an improvement over the present situation. The decomposition approach developed in the next section is only slightly more difficult to set up than the single-level method it strives to imitate.

3.2 Development of a Direct Method

The technique presented in this thesis is based on the Model Coordination method outlined by Kirsch (1975 & 1979). Consider a single-level optimization problem organized in such a way that allows for the assembled system to be broken down or decomposed into S individual components or subsystems. Partition the design variable vector into Y, the system level gross geometry, and X, the detailed dimensions of quantities in the S subsystems. Further partition vector X into X_1, X_2,...,X_s,...,X_S, where each X_s is the design variable vector for the s^{th} subsystem.

In the Model Coordination approach, the system is
decomposed into two levels according to this partition. On the first level are \( S \) individual structural components with one optimization sub-problem for each of the \( S \) components or subsystems. Within the \( s^{th} \) sub-problem on this level, the design variables \( X_s \) are the detailed dimensions of quantities in the \( s^{th} \) structural component. All other dimensions are fixed within each sub-problem e.g. the detailed dimensions of the other subsystems and the gross geometry of the whole system. The sub-problems on the first level each have their own performance criterion, constraints and variable bounds. Therefore, since there are a total of \( S \) components on the first level, there are \( S \) sub-problems as well. These \( S \) sub-problems are defined with equations (3.1) through (3.6).

\[
\begin{align*}
\text{Minimize:} & \quad F_s(X_s, Y) \\
\text{Subject to:} & \quad H_{s,k}(X_s, Y) = 0 \quad \text{for } k = 1, 2, \ldots, K_s \\
& \quad G_{s,j}(X_s, Y) - X_{s,j}^{vl} = 0 \quad \text{for } j = 1, 2, \ldots, J_s \\
& \quad X_{s,i}^{vl} \leq X_{s,i} \leq X_{s,i}^{vr} \quad \text{for } i = 1, 2, \ldots, N_s + J_s \\
& \quad X_{s,j}^{vl} = A_{s,j} \quad \text{for } j = 1, 2, \ldots, J_s \\
& \quad X_{s,j}^{vr} = B_{s,j} \quad \text{for } j = 1, 2, \ldots, J_s \\
& \quad X_s = (X_{s,1}, X_{s,2}, \ldots, X_{s,N_s+J_s}) \\
\end{align*}
\]

One sub-problem for each \( s = 1, \ldots, S \).

Note that vectors \( A \) and \( B \) are the lower and upper bounds, respectively, for the original inequality constraints.

It should be emphasized that though the sub-problems on
the first level depend on the system level variables \( Y \), those values are held constant. Another point to note is that it is physically possible that a design variable may be shared by two or more of the system's components. In that case, the design variable will then appear in two or more first level sub-problems.

The second or system level consists of the whole assembled system with the gross geometry of the problem as the design variable vector, \( Y \). At the system level, all of the subsystem's detailed dimensions (\( X = X_s \) for \( s = 1, \ldots, S \)) are held fixed. Recall, the vector \( X \) is a short hand representation for all of the first level design variables. The performance criterion \( F(X,Y) \), on the system level is the sum of all the individual subsystem's performance criteria. The system's constraints are any subsystem level constraints that involve system level design variables plus any exclusively system level constraints. The system level sub-problem takes the form described in equations (3.9) through (3.18).

Equations (3.9) through (3.13) represent a system level optimization problem that ignores the constraints from the first level entirely. Therefore, equations (3.10) and (3.11) are the \( K_{sys} \) equality and \( J_{sys} \) inequality constraints that are exclusively system level constraints, e.g. a system level buckling constraint or an limit on the natural frequency of the assembled structure. Since all of the subsystem level
constraints must be satisfied on the system level, those
carried by the system level design variables and their
corresponding slack variables must be added to the
system level problem. (Eqs. 3.14 & 3.15) The maximum number
of equality and inequality constraints, $K'_{sys}$ and $J'_{sys}$,
respectively, that the subsystems contribute to the system
level problem is determined by the following expressions.

\[
K'_{sys} = \sum_{s=1}^{S} K_s \\
J'_{sys} = \sum_{s=1}^{S} J_s
\] (3.7) (3.8)

The system level sub-problem is described below.

Minimize: \[ F(X,Y) = \sum_{s=1}^{S} F_s(X_s,Y) \] (3.9)
Subject to: \[ H_k(X,Y) = 0 \quad \text{for } k = 1,2,\ldots,K_{sys} \] (3.10)
\[ G_j(X,Y) - \sum_{s=1}^{S} Y_j^{s}_{sys} = 0 \quad \text{for } j = 1,2,\ldots,J_{sys} \] (3.11)
\[ Y_l^i \leq Y_i \leq Y_u^i \quad \text{for } i = 1,2,\ldots,N_{sys} \] (3.12)
\[ A_j \leq Y_j^{s}_{sys} \leq B_j \quad \text{for } j = 1,2,\ldots,J_{sys} \] (3.13)
\[ H_{s,k}(X_s,Y) = 0 \quad \text{for } k=1,\ldots,K_s \text{ & } s=1..S \] (3.14)
\[ G_{s,j}(X_s,Y) - X_j^{s}_{sys} = 0 \quad \text{for } j=1,\ldots,J_s \text{ & } s=1..S \] (3.15)
\[ A_s,j \leq X_j^{s}_{sys} \leq B_s,j \quad \text{for } j=1,2,\ldots,J_s \text{ & } s=1..S \] (3.16)
\[ Y = (Y_1, Y_2, \ldots, Y_{N_{sys}+J_{sys}+J'_{sys}}) \] (3.17)

There will be one new slack variable for each of the $J'_{sys}$
inequalities added to the system level problem. These slack
variables are added sequentially to the vector $Y$ as follows
Because the slack variables must change to satisfy the inequality constraints on both the system and the subsystem levels, they are not held fixed on either level. Therefore, to meet this requirement, all of the $J_{sys} + J'_{sys}$ slack variables on the system level must be allowed to change. In addition, within each first level sub-problem, the corresponding $J_s$ slack variables must be free to change while all others are held fixed.

Now that the first and second level sub-problems have been defined, they must be individually solved in such a manner as to cause the initial design vector $[x^{(0)}, y^{(0)}]$ to converge to the optimum solution $[x^*, y^*]$. A methodology is needed for solving the $S+1$ sub-problems ($S$ subsystems and one system level problem) so that the optimum solution of the decomposed system at least approximates the optimum solution of the single-level problem. Unfortunately, proponents of the Model Coordination method have limited their published research to formulation of the problem's organizational structure and stopped short of actually defining an algorithm to find the optimum solution (Kirsch, 1975, and Kirsch & Moses, 1979).

The three different decomposition solution techniques
attempted are described below. Since the first two approaches failed to converge to the optimum solution, those methods will be explained only briefly. In contrast, the last technique, which actually did converge to the same optimum solution as the single-level equivalent problem, will be described in detail.

All three techniques share a common computational structure. Recall the solution strategy of single-level problems, where the GRGOPT32 algorithm can be restarted with the previous cycle's optimum solution $X^t$ becoming the next cycle's initial feasible starting point $X^{(0)}$. In all three approaches, the solution sequence allows for the $S+1$ sub-problems to be individually restarted in cycles as before. In addition, these $S+1$ individual sub-problems are embedded in another loop, called the main solution loop. For any given pass through the main solution loop, each of the $S+1$ sub-problems is solved at least once. A pass through this loop is analogous to a cycle through an individual sub-problem in the same way that the total system (the collection of all $S+1$ sub-problems) is analogous to an individual sub-problem. Within the main solution loop, there is freedom to arrange the actual sequence of solving the sub-problems such that a particular sub-problem is first, last or in any order. The number of cycles per individual sub-problem and the maximum number of passes through the main solution loop are controlled. The three methods are described below.
Decomposition Method 1:

In this approach, when each of the sub-problems is submitted to GRGOPT32, it is allowed to iterate until it converges to its individual optimum solution. The individual sub-problems are set up in the same manner as the single-level problems from Chapter 2. The objective function and constraints gradients are constructed from partial derivatives exactly as for the single-level problems.

This method failed because it is highly dependent on the location of the initial starting point relative to the optimum solution. Consider a particularly simple sub-problem with an objective function that depends on only one active variable $X_{l,1}$, equation (3.19)

$$F(X_0) = kX_{l,1} \quad 3.19$$

where $k$ is some positive constant. An active variable is one that is not held fixed in a particular sub-problem. In contrast, inactive variables are the remaining variables that are held fixed in a sub-problem. Since the GRG optimization algorithm requires the objective function to decrease from one iteration to the next, if the active variable $X_{l,1}$ ever drops below $X_{l,1}'$, the unknown optimum value of the system, that sub-problem can never converge to the optimum solution. This is because the movement needed to return the variable to the system's optimum value would require the objective function's
value to increase. In other words, the search vector on the subsystem level would indicate that the descent direction is negative, because a decrease in $X_{i,j}$ produces a decrease in $F(X_{i,j})$. This change would produce a decrease in the local objective function only. In contrast, the system as a whole, would benefit from an increase in $X_{i,j}$. It is likely that if this sub-problem is now sub-optimal, that other sub-problems will be prevented from attaining their optimum solutions as well.

A physical example of this situation is as follows: structural component Z, which plays a significant part in the load path of a structural system is made too weak (compared to the optimum solution) at some intermediate solution because its sub-problem is presented a low set of loads. Since the loads are low, the local dimensions can be reduced and therefore the weight as well, until the stress or deflection constraints become active. In order for the system to accommodate this weak link, the load must be carried inefficiently through some remote portion of the structural system. This would make the system heavier than the optimum value, but reduction of the system's weight is prevented because the weight of component Z would have to increase before the other subsystems' weights could decrease. Since the objective function value of any sub-problem (i.e. the weight) can not increase in the optimization algorithm, progress towards the system's optimum solution is completely
halted. Therefore, at least in very simple decomposition problems, this method will fail to converge to the optimum solution whenever one of its sub-problems starts off below or drops below the unknown optimum value.

At the time, the remedy for this undesirable situation was thought to be the replacement of the partial derivatives with total derivatives for the construction of the various gradients. The reasoning was that, using the sophisticated derivative information to form the search vectors, would prevent passing beyond the optimum solution. The implementation of the total derivatives and why they failed are described in the next approach.

Decomposition Method 2:

This approach is similar to the first except that total derivatives are used, when available, wherever partial derivatives were used previously. Recall the chain rule for finding total derivatives on a function $F(X,Y)$, where both $X$ and $Y$ are vectors.

$$\text{Grad}_Y F(X,Y) = \frac{dF}{dY} = \frac{dF}{dX} \frac{dX}{dY} + (\frac{dF}{dX})^T (\frac{dX}{dY})$$

Now since variables $X$ and $Y$ are problem variables, the quantities $\frac{dF}{dY}$ and $\frac{dF}{dX}$ should be known or easily obtainable numerically. The difficulties stem from the need for $\frac{dX}{dY}$, the rate of change of the inactive variables with
respect to the active variables. The matrix \[ \frac{d X}{d Y} \] is called the design variable or parameter sensitivity matrix.

A search of the available literature revealed that design sensitivities are used in two areas, post-optimality studies and decomposition. Post-optimality studies are performed to investigate the behavior of the optimum solution when a problem parameter, that was previously held constant, is changed. The parameters that were previously held constant, could easily be the inactive variables in a decomposition problem. The trouble with obtaining the design sensitivities is twofold: first, their determination is very computationally intensive, and second, they can only be obtained at the optimum solution.

Three different methods for finding design sensitivities were attempted. The first two numerical approaches failed because the process for generating initial feasible starting points for the GRGOPT32 program could not be automated (Vanderplaats & Yoshida, 1985 & Beltracchi & Gabriele, 1988). Though these numerical methods are not too cumbersome to set up, a major disadvantage is that a new optimization problem must be solved for each inactive parameter and each time the total derivatives are required!

The third method for generating design sensitivities is based on the use of second order derivative information of the objective function and the constraints (Li & Azarm, 1990). This method offers a closed form solution for the design
sensitivities, provided second order partial derivative information is available. The main limitation of this technique is that it could be used only at an optimum solution and only in the system level sub-problem. The GRGOPT32 algorithm was modified to use partial derivatives to construct the reduced gradient until a Kuhn-Tucker point was found, at which time the sensitivities could be determined and the total derivatives obtained. Though this technique was made to produce accurate total derivative information, since it was only available after an optimum solution was found and only on the system sub-problem level, the damage was already done. At that point, either the system itself had a variable go to far beyond the optimum value or one of the other sub-problems had a variable drop below its optimum value.

The remedy for this problem is to provide all sub-problems with total derivative information at all times. The trouble with approach is that since the design sensitivity information required to construct the total derivatives is only available at the optimum solution, the total derivatives can not be made available at all times.

This paradox prompted the idea of severely limiting how much progress can be made in each sub-problem. This would allow all the variables to approach the optimum in a more uniform manner. This concept evolved into the successful decomposition technique described next.
Decomposition Method 3:

This approach is similar to the first method in that it also uses partial derivatives. What makes this new method different and unique, is instead of allowing each of the sub-problems to be solved very accurately within each pass through the main solution loop, three key parameters are adjusted to severely limit progress. This prevents any one sub-problem from being solved too accurately and causing one or more of its active variables to drop below the theoretical optimum. The three key parameters are the maximum number of iterations allowed per cycle, the initial step size and the initial starting point. The philosophy behind this approach is to force each of the S+1 sub-problems' set of active variables to approach the optimum solution in a more uniform fashion. No set of active variables is allowed to change too radically on any particular restart, thereby allowing the other variables, in turn, to pick up the slack. This forces all sets of active variables to approach the optimum solution together and prevents any one sub-problem from strongly biasing the direction that the overall solution takes.

During the development of this method, it was found that within each pass through the main solution loop, it is necessary to restrict the number of restarts per sub-problem to one and to limit the maximum number of iterations allowed per cycle to three. This limits the progress within each of sub-problems so that none are allowed to be solved too
accurately. In addition, tests revealed that the sub-problem's tolerances should not be reduced unless an optimum solution is found on that particular cycle. With these strict limitations defined, it was found that the method's ability to converge to the optimum solution is most significantly controlled by the selection of the initial step size $a^0$ and the initial feasible starting point $x^{(0)}$. For any given initial starting point, the relative success of the method is almost solely governed by the initial step size. The influence of the initial step size is so crucial, in fact, that the entire decomposition process can be reduced to an optimization problem in this one parameter!

Selection of these critical variables is fundamental to the success of the optimization process. It assumed that any number of initial feasible starting points can be generated, therefore discussion will focus on choosing the initial step size $a^0$. An interactive computer program was written to take a value of $a^0$ from the analyst and to allow the main solution loop to be repeated until the system level objective function converged. The analyst's goal is to minimize the system level objective function through the prudent selection of $a^0$. The analyst is prompted repeatedly for a new value of $a^0$ until a minimum has been found.

The analyst also has the option of selecting an interval of initial step sizes over which to perform a search for the minimum objective function value. The program returns the
best initial step size in the interval based on an approximation by a quadratic interpolating polynomial. The interpolation is accomplished by dividing the interval into ten sub-intervals and a routine simply searches for a function value that is less than the function values both to the left of the point being considered and to the right. When this situation is found, a quadratic polynomial is fit to the three points and the approximation for the minimum is returned. If the objective function at either of the intervals' endpoints is lower then that value is returned.

For the two problems tested, values for $a^0$ required to find the optimum solution were $5 \times 10^{-5}$ and 300. It is recommended to try values from ranging $1 \times 10^{-6}$ to $1 \times 10^{16}$. Finding an optimum value of $a^0$ over this huge interval may seem like an enormous task, however, if approached methodically it is demonstrated to be possible.

Of course this approach offers no relief when the original starting point is bad. The challenge is to identify a bad initial starting point. Based on how the solution progresses for a number of different initial starting points, bad initial starting points usually reveal themselves. The most common indication is when a variable seems to hit an artificial bound, that is, a variable's progress through the design space slows to a stop at some value that may be away from its upper or lower bounds. This trend may indicate that a variable has exceeded its optimum value and would like to
reverse its direction of motion but is not allowed. These artificial bounds will differ slightly for each starting point used. The best way to find a good initial starting point is to try as many points as possible that are distributed evenly over the design space. Starting points with at least some components located on their variable bounds seem to fair best.

This optimization on $\alpha^0$ is repeated for each of the initial feasible starting points until a practical optimum solution is found.

In the next section, a decomposition algorithm that utilizes this new method is constructed. In the last section, this decomposition technique is demonstrated with the two-level version of the single-level example truss problem previously solved in Chapter 2. It is shown that the new decomposition method successfully converges to the problem's known optimum solution.
3.3 Decomposition Algorithm

Now that the basic features of this method have been presented in the preceding section, an outline of the decomposition algorithm can be constructed. This algorithm employs a repetitive solution loop strategy with the endpoint of each iteration becoming the starting point for the next. Within each pass through the main solution loop, every sub-problem is solved to a degree of accuracy controlled by the following parameters: the initial step size, the maximum number of iterations allowed in a cycle, the maximum number of cycle restarts for each sub-problem and the tolerances. This procedure is repeated until the process converges or until the maximum number of passes through the main solution loop is reached.

The solution procedure for decomposition problems differs from that for single-level optimization problems in that the process is interactive. As before, the analyst prepares a file which specifies an initial feasible starting point and all of the other parameters, functions and subroutines that define the problem. The solution sequence is interactive in that the initial step size $\alpha^0$, must be provided by the user at run time. For each value of $\alpha^0$ input, the loops are repeated until convergence is achieved. The analyst then makes a prediction of the next value of $\alpha^0$ based on the minimum system level objective function that resulted from the last
approximation. This process continues until a practical optimum solution is found.

It should be noted that the sub-problems are solved with the same subroutine that was used to solve single-level optimization problems, namely GRGOPT32. The source code for GRGOPT32 is located in Appendix A. The format for setting up the problem is defined with Template 2. (Appendix C.5) Details for formulating a two-level optimization problem, discussion of the data output and an example problem are addressed in the Appendix C.

An outline of this decomposition algorithm follows.

Step 0.0 Sub-problem definition information:

For each of the S first level sub-problems as well as for the one second level sub-problems, given the following input parameters, convergence tolerances, initial feasible point, functions and subroutines:

Parameters: (for \( s = 1,2,\ldots,S \))

\( N_s \): Number of independent variables.

\( K_s \): Number of equality constraints.

\( J_s \): Number of inequality constraints.

GAMMA: Step size reduction parameter:

Use 0.5

\( TOL_{s1} \): Convergence tolerance for reduced gradient & search vector norms.
TOL_s2: Tolerance vector for feasibility of constraints (Length $K_s + J_s$).

TOL_s3: Convergence tolerance vector for Newton iterations and tolerance for vector component deltas (Length $N_s + J_s$).

TOL4: Convergence tolerance for the system level objective function.

X^{(0)}: Initial feasible point (Length $N_s + J_s$).

X^U_s: Vector of upper bounds of X (Length $N_s + J_s$).

X^L_s: Vector of lower bounds of X (Length $N_s + J_s$).

MAXIT_s: Maximum number of iterations per cycle. (Use $\text{MAXIT}_s = 3$)

NEWTIT: Maximum number of Newton iterations allowed.

CYCLE_s: Maximum number of cycle restarts (Use $\text{CYCLE}_s = 1$)

Functions: (for $s = 1, 2, \ldots, S$)

$F_s$: Objective function.

Subroutines: (for $s = 1, 2, \ldots, S$)

Grad $F_s$: Gradient of objective function.

$H_s$: Equality constraints.

Grad $H_s$: Gradient of equality constraints.
G_s: Inequality constraints.
Grad G_s: Gradient of inequality constraints.

Functions: (System level)
F_sys: Objective function.

Subroutines: (System level)
Grad F_sys: Gradient of objective function.
H_sys: Equality constraints.
Grad H_sys: Gradient of equality constraints.
G_sys: Inequality constraints.
Grad G_sys: Gradient of inequality constraints.

Step 1.0 Interactive input: User is prompted for....
\( \alpha^0 \) (Initial step size)
\( \alpha^0_{\text{Left}} \) & \( \alpha^0_{\text{Right}} \) (Interval over which to search)
&/or TOL4 (Convergence Tolerance)

Step 2.0 Start solution loop.

2.1 For each of the S+1 sub-problems,
do Steps 2.2 to 2.4.

2.2 Solve the sub-problem with the given information.
Note: \( X_s^0 \) comes from the common block.

2.3 Test for optimum solution.
Was Optimum solution found?
Yes: Reduce tolerances & continue.
No: Continue.

2.4 Update common block with new \( X_s^t \).
Step 3.0 All sub-problems have been solved, therefore
Test for convergence of system level objective function.

Has system level objective function converged?

\[ \left| \frac{F^{(i)}_{sys} - F^{(i+1)}_{sys}}{F^{(i+1)}_{sys}} \right| \leq TOL4 \] ?

Yes: Candidate optimum has been found.
Output \( F^{(i+1)}_{sys} \), \( x^* \) & \( y^* \) (\&/\( a^0_{best} \))
Goto Step 1.0

No: Test main solution loop repeat limit.
Has limit been reached?
No: Print status
Output \( F^{(i)}_{sys} \), \( XCOM^{(i)} \) & \( Y^{(i)} \)
Goto Step 2.1
Yes: Goto Step 1.0

With the development of the algorithm now complete, the next step is to discuss how to solve a decomposed optimization problem with this method. The next section addresses setting up and solving a two-level optimization problem. This is the same problem that was solved as a single-level problem in Chapter 2.
This decomposition methodology is most clearly explained with an example problem. For purposes of comparison, the example selected is one that was solved previously as a single-level problem in Chapter 2. Figure 6 shows the truss to be optimized in this problem.

The system is decomposed into two levels. The first level consists of two sub-problems, one for each of the structural members. On this level the design variables are the cross sectional areas of each member and the only constraints are the axial stresses in the members, one per subsystem. The performance criterion for each subsystem is its volume. The gross geometry of the system is held fixed at this solution level. The system level problem has only the vertical dimension as a design variable, while the cross sectional areas of the members are held fixed. The system's performance criterion is the sum of the two members' volumes and the constraints are the two stress constraints from the first level problems because system level design variable \( Y_1 \) appears in both.

In summary, the system has been decomposed into three sub-problems; one is a system level problem and the other two correspond to the two subsystems i.e. the truss members.
Mathematically, the two subsystem level sub-problems are defined as follows:

Minimize: \( F_1(X,Y) = X_{1,1} \left( 16 + (Y_1)^2 \right)^{\frac{1}{2}} \) \( \quad \) 3.20
Subject to: \( G_{1,1}(X,Y) = 20 \left[ \left( 16 + (Y_1)^2 \right)^{\frac{1}{2}} / (Y_1 X_{1,1}) \right] - X_{1,2} = 0 \) \( \quad \) 3.21
\( 0 \leq X_{1,1} \leq 0.003 \) m\(^2\) \( \quad \) 3.22
\( 0 \leq X_{1,2} \leq 100000 \) kN/m\(^2\) \( \quad \) 3.23

Minimize: \( F_2(X,Y) = X_{2,1} \left( 1 + (Y_1)^2 \right)^{\frac{1}{2}} \) \( \quad \) 3.24
Subject to: \( G_{2,1}(X,Y) = 80 \left[ \left( 1 + (Y_1)^2 \right)^{\frac{1}{2}} / (Y_1 X_{2,1}) \right] - X_{2,2} = 0 \) \( \quad \) 3.25
\( 0 \leq X_{2,1} \leq 0.003 \) m\(^2\) \( \quad \) 3.26
\( 0 \leq X_{2,2} \leq 100000 \) kN/m\(^2\) \( \quad \) 3.27
The system level problem can now be expressed as follows.

\[
\text{Minimize: } F_{\text{sys}}(X,Y) = X_{1,1} \left(16 + (Y_1)^2\right)^{\frac{1}{2}} + X_{2,1} \left(1 + (Y_1)^2\right)^{\frac{1}{2}}
\]

Subject to: \[
G_1(X,Y) = 20 \left[\frac{(16 + (Y_1)^2)}{X_{1,1}}\right] - Y_2 = 0
\]
\[
G_2(X,Y) = 80 \left[\frac{(1 + (Y_1)^2)}{X_{2,1}}\right] - Y_3 = 0
\]
\[
lm \leq Y_1 \leq 3 \text{ m}
\]
\[
0 \leq Y_2 \leq 100000 \text{ kN/m}^2
\]
\[
0 \leq Y_3 \leq 100000 \text{ kN/m}^2
\]

Recall that from equation (3.18), the slack variables in the system level sub-problem are related to the slack variables in the first level sub-problems.

\[
Y_2 = X_{1,2}
\]
\[
Y_3 = X_{2,2}
\]

Two initial feasible \(X^{(0)}\) vectors were selected and the best value for the initial step size was found for each. The first design variable vector attempted was the initial feasible point used in the single-level example from Chapter 2. Table 1 chronicles the process of finding the best estimate for the \(\alpha^0\). Because the goal of the study was to find the theoretical optimum solution, the estimation process was continued until the minimum \(F_{\text{sys}}\) converged. Table 2 shows initial starting point A and the minimum objective function found when the best estimate for \(\alpha^0\) was used.
### Table 1: Estimation of Initial Step Size, Starting Point A.

<table>
<thead>
<tr>
<th>Initial Step Size ($F_{sys} \times 10^{13}$)</th>
<th>Convergence Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$10^{-7}$</td>
</tr>
<tr>
<td>10</td>
<td>&quot;</td>
</tr>
<tr>
<td>100</td>
<td>&quot;</td>
</tr>
<tr>
<td>500</td>
<td>&quot;</td>
</tr>
<tr>
<td>1000</td>
<td>&quot;</td>
</tr>
<tr>
<td>200</td>
<td>&quot;</td>
</tr>
<tr>
<td>300</td>
<td>&quot;</td>
</tr>
<tr>
<td>400</td>
<td>&quot;</td>
</tr>
<tr>
<td>500</td>
<td>&quot;</td>
</tr>
<tr>
<td>700</td>
<td>&quot;</td>
</tr>
<tr>
<td>(500, 700)</td>
<td>4.00100</td>
</tr>
<tr>
<td>(490, 530)</td>
<td>4.00095802</td>
</tr>
<tr>
<td>(490, 530)</td>
<td>$10^{-15}$</td>
</tr>
</tbody>
</table>

**Notes on Data Format:**

1. $\alpha_{\text{Guess}}$  
   $F_{sys}(\alpha_{\text{Guess}})$ (Simple guess)

2. $(\alpha_{\text{Left}}, \alpha_{\text{Right}}) \alpha_{\text{Best}}$  
   $F_{sys}(\alpha_{\text{Best}})$ (Quadratic interpolation)

3. Best $\alpha$  
   (This $\alpha$ produced minimum $F_{sys}$)

---

Table 2: Solution A for Two-Level Example.
The next solution sequence uses a different starting point. Table 3 shows the estimation process for the initial step size, while Table 4 shows initial starting point B and the minimum objective function found when the best estimate for $a^0$ was used.

<table>
<thead>
<tr>
<th>Initial Step Size</th>
<th>$F_{st}$ x $10^{13}$</th>
<th>Convergence Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.318</td>
<td>10^{-7}</td>
</tr>
<tr>
<td>10</td>
<td>4.219</td>
<td>&quot;</td>
</tr>
<tr>
<td>100</td>
<td>9.25</td>
<td>&quot;</td>
</tr>
<tr>
<td>200</td>
<td>4.038</td>
<td>&quot;</td>
</tr>
<tr>
<td>300</td>
<td>4.000649</td>
<td>&quot;</td>
</tr>
<tr>
<td>400</td>
<td>5.00</td>
<td>&quot;</td>
</tr>
<tr>
<td>350</td>
<td>4.029</td>
<td>&quot;</td>
</tr>
<tr>
<td>250</td>
<td>4.00877</td>
<td>&quot;</td>
</tr>
<tr>
<td>275</td>
<td>4.00125</td>
<td>&quot;</td>
</tr>
<tr>
<td>(275, 300)</td>
<td>289.6527</td>
<td>&quot;</td>
</tr>
<tr>
<td>(275, 300)</td>
<td>289.6527</td>
<td>4.0000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10^{-16}</td>
</tr>
</tbody>
</table>

Best $a^0$ (This $a^0$ produced minimum $F_{st}$)

Table 3 Estimation of Initial Step Size, Starting Point B.

PASS ==> 0
ALPHA0 ==> 289.65270000000000
TOL4 ==> 1.0000000000000E-016

XCOM(1,i)= 2.499999941206E-003 13333.33300781300000
XCOM(2,i)= 2.499999941206E-003 33730.96093750000000
Y(i)= 3.00000000000000

OBJECTIVE FUNCTION VALUE ==> 2.0405693694318E-002

PASS ==> 18

XCOM(1,i)= 4.4721358621263E-004 100000.00000000000000
XCOM(2,i)= 8.9442718635625E-004 100000.00000000000000
Y(i)= 2.000000005191770

OBJECTIVE FUNCTION VALUE ==> 4.0000000000000E-003

Table 4 Solution B for Two-Level Example.
The results shown in Table 4 agree very well with the known optimum solution to this problem (Kirsch, 1975). This solution also matches the findings from the single-level example problem solved in Chapter 2. The problem's key definition parameters are listed in Appendix C.5.

It should be noted that the search for the optimum value of $\alpha^0$ was slightly biased because the optimum solution was known in advance for this problem. This advance knowledge allowed the analyst to solve the problem very precisely in order to match the single-level solution. In practice, where the optimum solution is seldom known in advance, the process is usually halted when the value of $F_{sys}$ fails to improve by more than a few percent with each guess.

If finding a practical optimum had been the goal of this particular optimization study, the solution sequence may have been halted after the fourth or fifth estimate of $\alpha^0$. The fourth and fifth estimates of $\alpha^0$ for initial starting points A and B, respectively, returned reasonable optimum solutions. It is useful to compare the optimum solution to one of these reasonable solutions. For example, starting point B and an estimate of $\alpha^0 = 300$ produced the results listed in Table 5. The number in parentheses after each entry in the final solution is the percent deviation from the optimum solution. The relatively small percent errors indicate that this solution is a fairly good approximation and therefore a practical solution to the problem. In contrast, Table 6
lists the solution resulting from the same starting point and
the step size estimate $a^0 = 1$. Though the system level
objective function deviates from the optimum solution by only
8%, the percent deviations for variables $X_{1,1}$ and $Y_1$ are 25% and
49%, respectively. This shows that this problem's objective
function is relatively insensitive to large changes in some
of the variables near the optimum solution.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS</td>
<td>ALPHAO</td>
<td>TOL4</td>
</tr>
<tr>
<td>0</td>
<td>300.00000000000000</td>
<td>1.0000000000000E-007</td>
</tr>
</tbody>
</table>

Table 5 Solution B: Initial Step Size $a^0 = 300$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS</td>
<td>ALPHAO</td>
<td>TOL4</td>
</tr>
<tr>
<td>0</td>
<td>1.00000000000000</td>
<td>1.0000000000000E-007</td>
</tr>
</tbody>
</table>

Table 6 Solution B: Initial Step Size $a^0 = 1$
In the introduction to this chapter, the basic advantage of decomposition was described as follows: it is more efficient to solve several small problems many times than it is to solve one large problem once. However, it is usually difficult to demonstrate the mathematical efficiency of the decomposed solution with a small example problem (Sobieski 1985). Because the computational effort required by most optimization methods tends to be concentrated in matrix inversion, and the work required for this operation is proportional to the matrix dimension cubed, larger problems will generally be solved more efficiently with decomposition. The number of mathematical operations required to find an optimum solution is a good measure of efficiency of a given method.

The number of matrix inverses required to find the optimum solution of the single-level problem from Chapter 2 was 277. Since in that example, the matrix that was inverted was of dimension 2, the total number of mathematical operations was \(277(2)^3 = 2216\). Every estimate of \(\alpha^0\) in a two-level solution, required inverting an average of 136 matrices of dimension 1 and 46 matrices of dimension 2. The average of the total number of mathematical operations required is therefore \(136(1)^3 + 46(2)^3 = 504\). Though this is only 22.7% of the arithmetic operations for the single-level solution, 41 estimates of \(\alpha^0\) were required for starting point A and 25 estimates were required for starting point B. Recall that
each time the quadratic interpolation option is utilized it can require up to 13 estimates of $a_0$ to find the best initial step size in the interval. Therefore, to make a fair comparison of methods, all the $a_0$ estimates required to find the optimum should be accounted for. Hence, the total number of mathematical operations required becomes 20664 and 12600, for starting points A and B, respectively. This exceeds the effort required for the single-level solution by factors of 9.3 and 5.7 for A and B, respectively.

Two factors contribute to this inefficiency: the small problem size and the degree of precision to which the solutions were taken. Larger problems should demonstrate the efficiency of the method for the reasons stated earlier. The degree of precision used here might be considered unrealistic for optimization studies where practical optima will suffice.

Appendix C contains a detailed solution to another problem that was used in testing and verifying this two-level optimization method. Appendix C can be regarded as a complete users manual for solving two-level optimization problems with GRGOPT32 as it contains information on setting up the problem, advice on selection of key parameters, explanations of the output format and how to use the interactive menus.

This concludes the chapter of two-level decomposition with the GRG method. The next chapter lists the conclusions drawn from this thesis as well as recommendations for further study in this area.
Chapter 4: Conclusions & Recommendations

This chapter contains a summary of what has been accomplished in this thesis as well as a list of topics for further work with this decomposition method.

4.1 Conclusions

A method has been developed to decompose a single-level optimization problem into a two-level problem. The computational organization of the decomposition technique is based on the Model Coordination approach proposed by Kirsch (1975 & 1979). In this method, the problem is partitioned into a set of sub-problems, each with its own design variables. Within each sub-problem, only the corresponding design variables are allowed to change, while all other design variables are held fixed. The set of sub-problems is repeatedly solved until the system level objective function converges.

Two new developments have contributed to the success of this innovative direct approach to structural decomposition. The first new feature is that each of the sub-problems is solved with the Generalized Reduced Gradient optimization algorithm (Abadie & Carpentier, 1969). Because the Generalized Reduced Gradient method develops a sequence of feasible points as intermediate solutions and the value of
the objective function must decrease monotonically with that sequence, the iterations can be stopped at any time with a resulting feasible solution that will have a superior objective function value. This is particularly advantageous when a practical optimum will be accepted if the theoretical optimum can not be found. The second and more significant development is as follows: by adjusting the maximum number of iterations per sub-problem and the initial step size, progress is limited in each of the sub-problems so that none strongly bias the direction that the overall solution takes. This causes the individual sub-problems to approach the system's optimum solution in a uniform manner.

The technique is demonstrated with a two bar planar truss example problem. The solutions to the single-level optimization problem and two-level decomposed optimization problem are essentially identical, thereby validating the proposed method.

Since the technique requires no more than first order derivative information, it is much simpler to use than most of the decomposition methods in the current literature. By use of several examples, Appendices B and C describe the procedure for setting up and solving single-level and two-level optimization problems, respectively.

This method has potential applications in the optimization of large multi-disciplinary engineering systems. Different optimization algorithms, specialized for a
particular problem type or discipline, could be used on the various subsystems as is necessary. The computational speed of this method could be increased by using parallel processing, that is solving the various sub-problems simultaneously. This however, poses a data base management obstacle. Suppose one sub-problem needs access to variables that are changing in another sub-problem? The FORTRAN77 common block currently used would be inadequate for tracking the data that is changing in other subroutines running concurrently. Systolic memory could be used to manage the rapidly changing data base by making current values of the design data available to all sub-problems running simultaneously.
4.2 Recommendations

The author has seven recommendations for further work on this topic:

1) The search for the optimum initial step size should be automated for expediency.

2) Problem definition in FORTRAN should be made easier.

3) This technique should be applied to larger problems, that is, more subsystems, more design variables per subsystem, more load cases and more constraints.

4) The method should be tested on problems that have design variables that appear in more than one subsystem.

5) This technique should be tested on problems with more than two levels.

6) This method should be applied to a problem with system level natural frequency constraints.

7) The method should be applied to a multi-disciplinary system using parallel processing and systolic memory to keep track of the design variables common to many of the sub-problems.
Bibliography


Appendices
Appendix A: GRGOPT32 Fortran Code

For a list of FORTRAN variables and the equivalent symbolic names used in the thesis, refer to section 1.5.

SUBROUTINE GRGOPT32(N,K,J,ALPHA0,GAMMA,TOL1,TOL2,TOL3,X0,XUPPER, &XLOWER,MAXIT,NEWTIT,F,GRADF,HCON,GRADH,GCON,GRADG,PRINT,SYs,X,OK)

C
C PROGRAM NAME: GRGOPT32.FOR =======> DOUBLE PRECISION <=========
C BY: TOM VEILLEUX
C DATE CREATED: 09-18-91
C MODIFIED: 09-28-91
C 10-02-91 STATUS PRINT OUT BY-PASS ADDED
C 10-16-91 DECOMP COMMON BLOCK ADDED
C
C PURPOSE:
C
C THIS PROGRAM EMPLOYS THE GENERALIZED REDUCED GRADIENT
C METHOD TO FIND THE OPTIMAL SOLUTION TO A CONSTRAINED
C NONLINEAR OPTIMIZATION PROBLEM. THE ALGORITHM CAN HANDLE
C A NONLINEAR OBJECTIVE FUNCTION, NONLINEAR EQUALITY AND
C INEQUALITY CONSTRAINTS AND BOUNDED VARIABLES. INEQUALITY
C CONSTRAINTS ARE 'CONVERTED' TO EQUALITIES WITH THE ADDITION
C OF SLACK VARIABLES. THIS INCREASES THE DIMENSION OF THE
C PROBLEM FROM N TO N+J. VARIABLE BOUNDS ARE DEALT WITH
C IMPlicitLY BY VARIOUS CHECKS THROUGHOUT THE ALGORITHM. THE
C DESIGN VARIABLES ARE PARTITIONED INTO K+J BASIC (DEPENDENT)
C VARIABLES AND N-K NONBASIC (INDEPENDENT) VARIABLES. BY
C SOLVING FOR THE NONBASIC VARIABLES IN TERMS OF THE BASIC
C VARIABLES, THIS PARTITION EFFECTIVELY REDUCES THE DIMENSION
C OF THE PROBLEM FROM N+J TO N-K.
C
C REFERENCE: ENGINEERING OPTIMIZATION: METHODS AND APPLICATIONS
C G.V. REKLAITIS, A. RAVINDRAN & K.M. RAGSDELL
C JOHN WILEY & SONS, 1983
C PAGES 377 THRU 402
C
C STANDARD PROBLEM FORMAT: PROBLEM DIMENSION N
C MINIMIZE F(X) FOR X AN N-VECTOR
C SUBJECT TO Hk(X) = 0 FOR k = 1, 2, ........ K
C Aj < Gj(X) < Bj FOR j = 1, 2, ........ J
C XLOWERi < Xi < XUPPERi FOR i = 1, 2, ........ N
C
C CONVERTED PROBLEM FORMAT: PROBLEM DIMENSION N + J
MINIMIZE $F(X)$ FOR $X$ AN $(N+J)$-VECTOR

SUBJECT TO $H_k(X) = 0$ FOR $k = 1, 2, \ldots, K$

$C_j(X) - X_{N+j} = 0$ FOR $j = 1, 2, \ldots, J$

$X_{LOWER_i} < X_i < X_{UPPER_i}$ FOR $i = 1, 2, \ldots, N$

$A_j < X_{N+j} < B_j$ FOR $j = 1, 2, \ldots, J$

USAGE:

CALL GRGOPT32(N,K,J,ALPHA0,GAMMA,TOLL,TOL2,TOL3,X0,XUPPER, &XLOWER,MAXIT,NEWIT,F,GRADF,HCON,GRADH,GCON,GRADG,PRINT,SYS,X,OK)

THE USER MUST PROVIDE THE FOLLOWING INPUT VARIABLES, FUNCTIONS & SUBROUTINES.

USER DEFINED INPUT ARGUMENTS:

NOTE: ALL REAL VARIABLES ARE REAL*8 (DOUBLE PRECISION)

N: NUMBER OF INDEPENDENT VARIABLES
K: NUMBER OFEquality CONSTRAINTS
J: NUMBER OF INEQUALITY CONSTRAINTS
ALPHA0: INITIAL STEP SIZE
GAMMA: STEP SIZE REDUCTION PARAMETER (USE 0.5)
TOLL: CONVERGENCE TOLERANCE FOR REDUCED GRADIENT & SEARCH VECTOR NORMS
TOL2: TOLERANCE VECTOR FOR FEASIBILITY OF CONSTRAINTS
LENGTH K+J
TOL3: CONVERGENCE TOLERANCE VECTOR FOR NEWTON ITERATIONS
AND TOLERANCE FOR VECTOR DELTAS (LENGTH N+J)
X0: INITIAL FEASIBLE POINT
XUPPER: VECTOR OF UPPER BOUNDS OF X
XLOWER: VECTOR OF LOWER BOUNDS OF X
MAXIT: MAXIMUM NUMBER OF ITERATIONS, A TERMINATION LIMIT
NEWIT: MAXIMUM NUMBER OF NEWTON ITERATIONS ALLOWED
PRINT: LOGICAL VARIABLE THAT GOVERNS PRINTING OF DATA
PRINT = TRUE ALL DATA IS PRINTED
PRINT = FALSE NO DATA IS PRINTED
SYS: LOGICAL VARIABLE FOR IDENTIFYING SYSTEM LEVEL DECOMPOSITION PROBLEMS.
SYS = TRUE FOR SYSTEM LEVEL DECOMPOSITION PROBLEMS
ALLOWS INCLUSION IN THE BASIS OF NONBASIC VARIABLES WHEN THEY ARE NEAR THEIR BOUNDS.
SYS = FALSE FOR ALL OTHER PROBLEMS, PREVENTS NONBASIC VARIABLES THAT ARE NEAR THEIR BOUNDS FROM ENTERING THE BASIS.

=> USER DEFINED FUNCTIONS: (MUST BE DECLARED EXTERNAL)
F(X): OBJECTIVE FUNCTION

=> USER DEFINED SUBROUTINES: (MUST BE DECLARED EXTERNAL)

GRADF(X,GRDF) GRADIENT OF OBJECTIVE FUNCTION
HCON(X,H) EQUALITY CONSTRAINTS
GRADH(X,GRDH) GRADIENT OF EQUALITY CONSTRAINTS
GOON(X,G) INEQUALITY CONSTRAINTS
GRADG(X,GRDG) GRADIENT OF INEQUALITY CONSTRAINTS

NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY

=> ARGUMENTS: (USED IN SUB PROGRAMS)

X: INDEPENDENT VARIABLES VECTOR, LENGTH N+J (UNSORTED)
GRDF: GRADIENT OF THE OBJECTIVE FUNCTION, LENGTH N+J
H: EQUALITY CONSTRAINTS, LENGTH K
GRDH: GRADIENT OF THE EQUALITY CONSTRAINTS, ORDER K BY N+J
G: INEQUALITY CONSTRAINTS, LENGTH J
GRDG: GRADIENT OF THE INEQUALITY CONSTRAINTS, ORDER J BY N+J

NOTE: * ALL ARRAYS AND VECTORS MUST BE DIMENSIONED TO THE SAME SIZE IN THE MAIN CALLING AND IN EACH SUBROUTINE. USE MX=10

* ALL REAL VARIABLES MUST BE DECLARED REAL*8
DOUBLE PRECISION

VALUES RETURNED BY GRGOPT:

X: OPTIMUM SOLUTION (IF IT WAS DETERMINED SUCCESSFULLY...)
OKAY: LOGICAL VARIABLE THAT DENOTES SUCCESS OF GRG IN FINDING THE OPTIMUM SOLUTION
OKAY = .TRUE. IF THE OPTIMUM SOLUTION WAS DETERMINED
OKAY = .FALSE. IF THE OPTIMUM SOLUTION WAS NOT DETERMINED

ARGUMENTS:
NC: NUMBER OF INDEPENDENT VARIABLES (USED IN COMMON BLOCK)
KC: NUMBER OF EQUALITY CONSTRAINTS (USED IN COMMON BLOCK)
JC: NUMBER OF INEQUALITY CONSTRAINTS (USED IN COMMON BLOCK)
NPJC: N+J FOR DIMENSIONING ARRAYS AND VECTORS
KPJC: K+J FOR DIMENSIONING ARRAYS & VECTORS (SIZE OF BASIS)
NMKC: N-K FOR DIMENSIONING ARRAYS & VECTORS (SIZE OF NON-BASIS)

X: INDEPENDENT VARIABLES VECTOR, LENGTH N+J (UNSORTED)
XBAS: PARTITION OF X, THE BASIC VARIABLES, LENGTH K+J
XNBAS: PARTITION OF X, THE NONBASIC VARIABLES, LENGTH N-K

RANGE: VECTOR OF RANGE OF X (ie XUPPER(i) - XLOWER(i))
UPREND: VECTOR OF DISTANCE X FROM UPPER BOUND (XUPPER(i) - X(i))
LWREND: VECTOR OF DISTANCE X FROM LOWER BOUND (X(i) - XLOWER(i))
EDGE: VECTOR OF RELATIVE POSITIONS OF THE COMPONENTS OF X, TO THEIR NEAREST BOUND MIN[UPREND(i)/RANGE(i) OR LWREND(i)/RANGE(i)]

IPARTN:VECTOR THAT TRACKS THE SUBSCRIPTS OF THE N+J VARIABLES.
THE FIRST K+J ARE THE BASIS AND THE K+J+1 THRU N+J ARE THE NONBASIC.

CHANGE: LOGICAL FLAG RETURNED BY THE BSORT SUBROUTINE INDICATING WHETHER THE PARTITION SEQUENCE HAS CHANGED SINCE THE LAST ITERATION.

NOCHNG: COUNTER THAT TALLIES NUMBER OF CONSECUTIVE TIME BSORT RETURNS A PARTITION SEQUENCE THAT DID NOT CHANGE.

REPEAT: COUNTER THAT TALLIES THE NUMBER OF TIMES THE ALPHA REDUCTION LIMIT WAS REACHED AND THE BASIS HEURISTIC'S USED.

SINGLR: COUNTER THAT TALLIES THE NUMBER OF TIMES THE CURRENT BASIS YIELDED A SINGULAR BASIS.

ISWITCH: INTEGER THAT THE GAUSSIAN ELIMINATION SUBROUTINE RETURNS WHEN THE JMAT MATRIX IS SINGULAR. ISWITCH IS THE ROW OR COLUMN NUMBER THAT CONTAINS A ZERO ON THE MAIN DIAGONAL OF JMAT.

GRDF: GRADIENT OF THE OBJECTIVE FUNCTION, LENGTH N+J
GRDFB: PARTITION OF GRDF IN THE BASIC VARIABLE SPACE, LENGTH K+J
GRDFNB: PARTITION OF GRDF IN THE NONBASIC VARIABLE SPACE LENGTH N-K
REDGRD: REDUCED GRADIENT, GRADIENT IN THE REDUCED SPACE OF THE NON-BASIC VARIABLES, LENGTH N-K

H: EQUALITY CONSTRAINTS, LENGTH K
GRDH: GRADIENT OF THE EQUALITY CONSTRAINTS, ORDER K BY N+J

G: INEQUALITY CONSTRAINTS, LENGTH J
GRDG: GRADIENT OF THE INEQUALITY CONSTRAINTS, ORDER J BY N+J
CON: TOTAL CONSTRAINT VECTOR, FIRST K COMPONENTS ARE THE
EQUALITY CONSTRAINTS (H), THE NEXT J COMPONENTS ARE THE
INEQUALITY CONSTRAINTS (G). LENGTH K+J

FSBLE: A LOGICAL VARIABLE THAT INDICATES IF A POINT SATISFIES
THE K+J EQUALITY CONSTRAINTS.

GRDCON: GRADIENT OF ALL CONSTRAINTS, FIRST K ROWS ARE THE GRADIENT
OF THE K EQUALITY CONSTRAINTS (H), THE NEXT J ROWS ARE THE
GRADIENT OF THE J INEQUALITY CONSTRAINTS (G).
ORDER K+J BY N+J

JMAT: MATRIX WHOSE ROWS ARE THE GRADIENTS (WRT BASIC VARIABLES)
OF THE K EQUALITY CONSTRAINTS & J 'CONVERTED' INEQUALITY
CONSTRAINTS. ORDER IS K+J BY K+J

JINV: INVERSE OF MATRIX JMAT. THE ORDER IS K+J BY K+J.

QMAT: MATRIX WHOSE ROWS ARE THE GRADIENTS (WRT NONBASIC
VARIABLES) OF K EQUALITY CONSTRAINTS & J 'CONVERTED'
INEQUALITY CONSTRAINTS. ORDER IS K+J BY N-K

D: SEARCH VECTOR, LENGTH N+J (SORTED INTO IPARTN ORDER)

DBAS: PARTITION OF D IN THE BASIC VARIABLE SPACE, LENGTH K+J

DNBAS: PARTITION OF D IN THE NONBASIC VARIABLE SPACE, LENGTH N-K

V: VECTOR USED IN NEWTONS ITERATIONS, LENGTH N+J (UNSORTED)

V1: UPDATED VERSION OF V, LENGTH N+J (UNSORTED)

CONVRT: MATRIX THAT CONVERTS NON-BASIS VECTORS TO BASIS VECTORS
JINV*QMAT WHEN RIGHT MULTIPLIED AND CONVERTS BASIS VECTORS
TO NON-BASIS VECTORS WHEN LEFT MULTIPLIED. ORDER K+J X N-K

CRRCT: VECTOR USED TO UPDATE THE BASIS VECTORS IN THE NEWTON
ITERATIONS [JINV(V)]*[CON(V)]

TTLINV: ACCUMULATIVE TALLY OF NUMBER OF J INVERSES REQUIRED

==> SUBROUTINES USED WITHIN CRGOPT:

NOTE: SEE ACTUAL SUBROUTINES FOR COMPLETE ARGUMENT LISTS &
DESCRIPTIONS

BSORT(EDGE, ISEQ, LENGTH, FLAG)

THIS PROGRAM EMPLOYED A BUBBLE SEARCH STRATEGY TO SORT
VECTOR A FORM LARGEST TO SMALLEST COMPONENT.

GRDCON(X, GRADH, GRADG, GRDCON)
THIS PROGRAM FORMS A MATRIX FROM THE GRADIENTS OF THE EQUALITY AND INEQUALITY CONSTRAINTS EVALUATED AT X.

CNSTRT(X,HCON,GCON,TOL2,CON,FSBLE)

THIS PROGRAM FORMS A VECTOR FROM THE GRADIENTS OF THE KPJ EQUALITY AND INEQUALITY CONSTRAINTS EVALUATED AT X.

THIS SUBROUTINE RETURNS A LOGICAL FLAG CONCERNING FEASIBILITY OF THE POINT

GSSINV(A,MA,NA,AINV,TOL,ISWITCH,FLAG1)

THIS SUB-PROGRAM IS USED TO DETERMINE THE INVERSE OF A MATRIX USING GAUSSIAN ELIMINATION. THE ALGORITHM RETURNS A DIAGNOSTIC FLAG THAT IS USED TO INDICATE THE EXISTENCE OF THE INVERSE.

MATMUL(A,MA,NA,B,MB,NB,C)

THIS SUB-PROGRAM IS USED TO PERFORM BASIC MATRIX MULTIPLICATION. MATRICES A & B ARE INPUTS, AND MATRIX C IS OUTPUT. (C = A * B)

VMATMUL(A,LA,B,MB,NB,C)

THIS SUB-PROGRAM IS USED TO PERFORM BASIC MATRIX MULTIPLICATION. VECTOR A & MATRIX B ARE INPUTS, AND VECTOR C IS OUTPUT. ([C] = [A] [B])

================================BEGIN MAIN PROGRAM=================================

C SUBPROGRAMS:
REAL*8 F
     EXTERNAL F,GRADF,HCON,GRADH,GCON,GRADG

C PARAMETERS: USED TO DIMENSION DUMMY VECTORS & ARRAYS
PARAMETER (MX=10)

C COMMON BLOCK VARIABLES:
     COMMON NC,KC,JC,NPJC,NMKC,KPJC
     COMMON /DECOMP/Y,XCOM

C VARIABLES:
     INTEGER IPARTN(MX),N,K,J,NPJ,KPJ,NMK,NC,JC,NPJC,KPJC,NMKC,NJ,
&MAXIT,NEWTIT,IT,ISWITCH,RESTRT,INEWT,TTLINV,NOCHNG,ITEMP,DEDEND(3),
&REPEAT,SINGLR,NMKPJ
     REAL*8 X0(MX),XUPPER(MX),XLOWER(MX),EDGE(MX),CON(MX),D(MX),
&VXEMAX(MX),CORRECT(MX),DBAS(MX),DNBAS(MX),V(MX),V1(MX),X(MX),
&XBAS(MX),XNBASE(MX),GRDF(MX),GRDFB(MX),GRDFNB(MX),REDGRD(MX),
&CONVRT(MX,MX),CMAT(MX,MX),GRDCON(MX,MX),JIND(MX,MX),JINV(MX,MX),
&SUPREND(MX),LWREND(MX),RANGE(MX),TOL2(MX),TOL3(MX),Y(MX),
&XCOM(MX,MX),TOL1,ALPHA0,ALPHA,GAMMA,TEMP,XUP,XLOW,MAG
C
LOGICAL FLAG,FSBLE,OK,CHANGE,PRINT,SYS
C
C DEFINE COMMON BLOCK PARAMETERS:
C
NPJ=N+J
NMK=N-K
KPJ=K+J
NC=N
KC=K
JC=J
NPJC=NPJ
NMKC=NMK
KPJC=KPJ
NJ=NPJ
NMKJPJ=NMK*KPJ
C
C TEST IF PROBLEM SIZE IS TOO LARGE FOR PRESENT ARRAY DIMENSIONS
IF (NPJ.GT.MX) THEN
C
C WRITE WARNING TO MODIFY ARRAY DIMENSIONING PARAMETER AND STOP.
C
C WRITE TO SCREEN
WRITE(*,*)
WRITE(*,*)'**************************************************************************************'
WRITE(*,*)'************************ WARNING ***********************************************'
WRITE(*,*)' PROBLEM IS LARGER THAN GROPT28 DEFAULT DIMENSIONS. '
WRITE(*,*)' MODIFY "MX" ARRAY DIMENSIONING PARAMETER IN GROPT32. '
WRITE(*,*)' MX MUST BE GREATER THAN N+J TO AVOID ARRAY OVERWRITES'
WRITE(*,*)'**************************************************************************************'
WRITE(*,*)'************************ WARNING ***********************************************'
WRITE(*,*)'**************************************************************************************'
WRITE(*,*)'**********
C
C WRITE TO FILE
WRITE(1,*)
WRITE(1,*)'**************************************************************************************'
WRITE(1,*)'************************ WARNING ***********************************************'
WRITE(1,*)' PROBLEM IS LARGER THAN GROPT28 DEFAULT DIMENSIONS. '
WRITE(1,*)' MODIFY "MX" ARRAY DIMENSIONING PARAMETER IN GROPT32. '
WRITE(1,*)' MX MUST BE GREATER THAN N+J TO AVOID ARRAY OVERWRITES'
WRITE(1,*)'**************************************************************************************'
WRITE(1,*)'************************ WARNING ***********************************************'
WRITE(1,*)'**************************************************************************************'
C
STOP
RETURN
C
ENDIF
C
C BY PASS PRINT OUT?
IF (PRINT) THEN

C OUTPUT INITIAL FEASIBLE POINT & PROBLEM HEADING
C
C WRITE TO THE SCREEN
WRITE(*,*)'================================================================='
&='INITIAL "FEASIBLE" VECTOR X0'
WRITE(*,5)(I,XO(I),I=1,NPJ)
5 FORMAT(I5,E14.7)
WRITE(*,*)' OBJECTIVE FUNCTION VALUE AT X0 ',F(XO)
WRITE(*,*)'================================================================='

C WRITE TO FILE
WRITE(1,*)'================================================================='
&='INITIAL "FEASIBLE" VECTOR X0 '
WRITE(1,5)(I,XO(I),I=1,NPJ)
WRITE(1,*)' OBJECTIVE FUNCTION VALUE AT X0 ',F(XO)
WRITE(1,*)'================================================================='

ENDIF

C INITIALIZE COUNTERS AND FLAGS
DEDEND(1)=KPJ
DEDEND(2)=0
REPEAT=0
NOCHNG=0
TTLINV=0

C DEFINE THE INITIAL VARIABLE SUBSCRIPTS (START WITH SEQUENTIAL ORDER)
DO 10,I=1,NPJ
IPARTN(I)=I
10 CONTINUE

C DETERMINE VARIABLE RANGE    RANGE(I) = XUPPER(I) - XLOWER(I)
DO 15,I=1,NPJ
  RANGE(I)=ABS(XUPPER(I)-XLOWER(I))
15 CONTINUE

C EQUATE X(i) AND XO(i)
DO 20,I=1,NPJ


\[ X(I) = X_0(I) \]

20 CONTINUE

C

C DEFINE THE FUNCTION GRADIENTS WITH RESPECT TO THE SLACK VARIABLES
C SINCE THESE ARE CONSTANT THROUGHOUT THE SOLUTION SEQUENCE, THEY NEED
C ONLY BE DEFINED ONCE.
C
C GRADIENTS OF OBJECTIVE FUNCTION WITH RESPECT TO SLACK VARIABLES
DO 25, I = N + 1, NPJ
   GRDF(I) = 0.0
25 CONTINUE

C

C GRADIENTS OF EQUALITY CONSTRAINTS WITH RESPECT TO SLACK VARIABLES
DO 30, I = 1, K
   DO 30, M = N + 1, NPJ
      GRDCON(I, M) = 0.0
30 CONTINUE

C

C GRADIENTS OF INEQUALITY CONSTRAINTS WITH RESPECT TO SLACK VARIABLES
DO 35, I = 1, J
   DO 35, M = 1, J
      IF (M.EQ.I) THEN
         GRDCON(I + K, M + N) = -1.0
      ELSE
         GRDCON(I + K, M + N) = 0.0
      ENDIF
35 CONTINUE

C

C START GRG ALGORITHM LOOP

C

C TERMINATE WHEN:
C   MAGNITUDE OF REDUCED GRADIENT < TOLL ........ (OK=T)
C OR MAGNITUDE OF NON-BASIC SEARCH VECTOR < TOLL ........ (OK=T)
C OR UNABLE TO FIND A LINEARLY INDEPENDENT BASIS ........ (OK=F)
C OR UNABLE TO FIND ANY NONBASIC VARIABLES ELIGIBLE
C TO ENTER THE BASIS .................................... (OK=F)
C OR ALPHA SEARCH HAS BEEN RESTARTED NEWTIT TIMES
C & THE VARIOUS BASIS HAVE BEEN TRIED ............... (OK=F)
C OR CONVERGENCE TO SAME X CONSECUTIVELY ............ (OK=F)
C OR MAXIT IS ENCOUNTERED ................................. (OK=F)

DO 1000, IT = 1, MAXIT

C

C STEP 1 PARTITION THE VECTOR X INTO BASIC AND NONBASIC VARIABLES.
C
C RULES:
C   DON'T ALLOW ANY VARIABLES INTO THE BASIS THAT ARE
C   NEAR THEIR BOUNDS. (BASIS VARIABLES NEED MAXIMUM
C   LATITUDE FOR CHANGE)
C   
C   MATRIX JMAT MUST HAVE A NONZERO DETERMINANT, MEANING
C   THE GRADIENTS OF THE CONSTRAINTS WRT TO THE BASIS
C   MUST BE LINEARLY INDEPENDENT. SHUFFLING MAY BE REQUIRED.
C
C DETERMINE THE NORMALIZED DISTANCE TO THE NEAREST VARIABLE BOUND FOR
C EACH OF THE COMPONENTS OF X. CALL THIS VECTOR 'EDGE'.
C
DO 40,I=1,NPJ
  UPRBND(I)=DABS(XUPPER(I)-X(I))
  XUP=UPRBND(I)/RANGE(I)
  LWRBND(I)=DABS(X(I)-XLOWER(I))
  XLOW=LWRBND(I)/RANGE(I)
C
  IF (XUP.LT.XLOW)THEN
   EDGE(I)=XUP
  ELSE
   EDGE(I)=XLOW
  ENDIF
40 CONTINUE
C
C ORDER THE INDEPENDENT VARIABLES ACCORDING TO THE RELATIVE DISTANCE
C FROM THEIR NEAREST BOUND. SORT EDGE(I) BY DECREASING MAGNITUDE USING
C BUBBLE SORT STRATEGY. THE FIRST K+J COMPONENTS OF IPARTN WILL BE THE
C BASIS VARIABLE SUBSCRIPTS PROVIDED MATRIX J HAS A NONZERO DETERMINANT.
C (JINV MUST EXIST)
C
  SINGLR=0
  REPEAT=0
  DEDEND(3)=0
45 CALL BSORT(EDGE,IPARTN,NPJ,CHANGE)
C
C TEST FOR PARTITION CHANGE: IF IPARTN HAS CHANGED CHANGE=.TRUE
C IF IPARTN HAS NOT CHANGED CHANGE=.FALSE
C
C FOR TWO CONSECUTIVE ITERATIONS WITH NO IPARTN SEQUENCE CHANGE,
C TRY ALTERING BASIS 'HEURISTICALLY' TO ASSIST CONVERGENCE.
C
C THE REASON FOR EMPLOYING THIS HEURISTIC BASIS CHANGE IS TO ATTEMPT TO
C FIND A BASIS SPACE WHERE PROGRESS (IE OBJECTIVE FUNCTION REDUCTION)
C CAN BE MADE. IT ALSO HAS BEEN SHOWN THAT FREQUENT BASIS CHANGES
C ASSISTS CONVERGENCE OF THIS ALGORITHM.
C
C THE CONCEPT BEHIND THIS HEURISTIC IS TO SEQUENTIALLY TRY SOME, BUT
C NOT ALL, PERMUTATIONS OF THE BASIS. THE (KPJ)th BASIC VARIABLE IS
C SWITCHED WITH THE FIRST NONBASIC VARIABLE AND THEN THE SECOND AND SO
C ON UNTIL THE NMKth NONBASIC VARIABLE HAS BEEN TRIED. NEXT THE
C (KPJ-1)th BASIC VARIABLE IS SWITCHED WITH THE NMK NONBASIC VARIABLES.
C THIS CONTINUES UNTIL ALL THE BASIC VARIABLES HAVE BEEN TRIED AND THEN
C IT IS RESTARTED WITH THE KPJth BASIC VARIABLE.
C
  IF (IT.EQ.1)THEN
    CHANGE=.TRUE.
  ENDIF
C
  IF (.NOT.(CHANGE))THEN
C CHANGE=.FALSE. PARTITION DID NOT CHANGE, INCREMENT NOCHNG
C & CHECK NOCHNG LIMIT
   NOCHNG=NOCHNG+1
   IF (NOCHNG.LE.0) THEN
      GOTO 85
   ENDIF
C
C THE NUMBER OF ITERATIONS WITH NO PARTITION CHANGE HAS BEEN EXCEEDED
C ALTER BASIS USING HEURISTICS
   NOCHNG=0
   ELSE
C
C CHANGE=.TRUE. PARTITION HAS CHANGED DURING BSORT, IGNORE HEURISTICS
C & CONT.
   NOCHNG=0
   GOTO 85
   ENDIF
C
60   DEDESD(2)=DEDESD(2)+1
   IF (DEDESD(2).LE.NMK) THEN
C
C IF THIS IS THE SYSTEM LEVEL, BY-PASS THIS RULE
C
   IF (.NOT.(SYS)) THEN
C
C NOTE: PREVENT VARIABLES THAT ARE VERY NEAR THEIR BOUNDS FROM ENTERING
C BASIS
   INB=IPARTN(KPJ+DEDESD(2))
   IF ((LWRBND(INB).LE.TOL3(INB)).OR.(UPRBND(INB).LE.TOL3(INB))) THEN
C
C THIS VARIABLE IS AT ITS BOUNDS, FIND ANOTHER CANDIDATE BASIS VARIABLE
C
   GOTO 60
   ENDIF
   ENDIF
   ITEMP=IPARTN(DEDESD(1))
   IPARTN(DEDESD(1))=IPARTN(KPJ+DEDESD(2))
   IPARTN(KPJ+DEDESD(2))=ITEMP
   GOTO 85
   ELSE
   DEDESD(1)=DEDESD(1)-1
   DEDESD(2)=0
   ENDIF
C
   IF (DEDESD(1).EQ.0) THEN
      DEDESD(1)=KPJ
      DEDESD(2)=0
      DEDESD(3)=DEDESD(3)+1
   IF (DEDESD(3).EQ.2) THEN
C
C ALL OF THE NON-BASIC CANDIDATE VARIABLES HAVE BEEN REJECTED FOR ONE OF
C THE FOLLOWING REASONS:
1) NON-BASIC VARIABLE WAS WITHIN TOL3 OF ITS BOUNDS <= [MOST LIKELY]
2) MATRIX JMAT FORMED WITH THIS BASIS WAS SINGULAR
3) BASIS YIELDED NO CONVERGENCE IN NEWTON'S ITERATIONS
4) NEWTON'S ITERATIONS CONVERGED BUT WITH NO OBJECTIVE FUNCTION IMPROVEMENT

SINCE THE BASIS CANNOT BE CHANGED, NO NEW VARIABLES CAN BE ADDED, THE SEARCH HAS STAGNATED.....RETURN WITH WARNING OF FAILURE.

BY PASS PRINT OUT?
IF (PRINT) THEN

WRITE TO SCREEN
WRITE(*,*)
WRITE(*,*)'********************************************
WRITE(*,*)' WARNING ********************************************
WRITE(*,*)' ALL THE NON-BASIC VARIABLES HAVE BEEN TRIED AND NONE'
WRITE(*,*)' ARE ELIGIBLE TO BE ADDED TO THE BASIS'
WRITE(*,*)'********************************************
WRITE(*,*)' WARNING ********************************************
WRITE(*,*)'********************************************
WRITE(*,*)'********************************************
WRITE(*,*)'********************************************
WRITE(*,*)

WRITE TO FILE
WRITE(1,*),
WRITE(1,*)'********************************************
WRITE(1,*)' WARNING ********************************************
WRITE(1,*)'********************************************
WRITE(1,*)'********************************************
WRITE(1,*)'********************************************
WRITE(1,*)' ALL THE NON-BASIC VARIABLES HAVE BEEN TRIED AND NONE'
WRITE(1,*)' ARE ELIGIBLE TO BE ADDED TO THE BASIS'
WRITE(1,*)'********************************************
WRITE(1,*)' WARNING ********************************************
WRITE(1,*)'********************************************
WRITE(1,*)'********************************************
WRITE(1,*)'********************************************

ENDIF
OK=.FALSE.
RETURN
ENDIF
GOTO 45
ENDIF
GOTO 60

CONTINUE

DETERMINE OBJECTIVE FUNCTION GRADIENT & CONSTRAINT GRADIENTS AT X
CALL GRADF(X,GRDF)
CALL GRADCN(X,GRADH,GRADG,GRDCON)

FIND A BASIS THAT YIELDS A NONSINGULAR JMAT:
C USE THE FIRST K+J COMPONENTS OF IPARTN AS THE BASIS VARIABLE
C SUBSCRIPTS
C
90     DO 100, I=1, KPJ
       DO 100, M=1, KPJ
          JMAT(I, M)=GRDCON(I, IPARTN(M))
100    CONTINUE
C
C FIND INVERSE OF JMAT: USE GAUSSIAN ELIMINATION SUBROUTINE
TTLINV=TTLINV+1
       CALL GSSINV(JMAT, KPJ, KPJ, JINV, TOL3, ISWTCH, FLAG)
C
C CHECK FOR EXISTENCE OF INVERSE VIA LOGICAL FLAG
C IF FLAG=FALSE THEN BASIS MUST BE MODIFIED
C       ELSE CONTINUE
C
        IF (.NOT.(FLAG))THEN
C BASIS UPDATE NEEDED ===> SWITCH IPARTN(ISWTCH) AND
C (IPARTN(KPJ+DEDEDEND(2)) SINGLR IS INCREMENTED WITH EACH FAILED ATTEMPT
C TO FIND A LINEARLY INDEPENDENT BASIS. IN THIS WAY ALL OF THE
C NON-BASIC VARIABLES ARE TRIED SEQUENTIALLY.
C RECALL ISWTCH IS THE ROW (COLUMN) OF JMAT THAT HAD A ZERO ON THE MAIN
C DIAGONAL. THE ROW CONTAINING THAT ZERO MAIN DIAGONAL COMPONENT WAS
C NOT LINEARLY INDEPENDENT. SINCE ROWS CAN NOT BE SWITCHED, SUBSTITUTE
C THE COLUMN CONTAINING THAT ZERO MAIN DIAGONAL COMPONENT WITH THE
C IPARTN(KPJ+DEDEDEND(2)) COLUMN OF GRDCON. LET DEDEDEND(1)=ISWTCH
C
110    SINGLR=SINGLR+1
       IF (SINGLR.GT.(NMKKPJ))THEN
C
C ALL THE VARIABLES HAVE BEEN SEARCHED AND A LINEARLY INDEPENDENT BASIS
C COULD NOT BE FOUND......TERMINATE
C
C BY PASS PRINT OUT?
C       IF (PRINT)THEN
C WRITE TO THE SCREEN
WRITE(*,*)' 
WRITE(*,*)' ITERATION NUMBER ', IT
WRITE(*,*)' OBJECTIVE FUNCTION AT X ', F(X)
WRITE(*,*)' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ', TTLINV
WRITE(*,*)' '--------------------------------------------'
&--------------------------------------------'
WRITE(*,*)' I IPARTN X'
WRITE(*,571)(I, IPARTN(I), X(I), I=1, NJ)
WRITE(*,*)' '--------------------------------------------'
&--------------------------------------------'
WRITE(*,*)' 
WRITE(*,*)' UNABLE TO FIND A LINEARLY INDEPENDENT BASIS'
WRITE(*,*)'******************************************************'
WRITE(*,*)
WRITE(*,*)'*
WRITE(*,*)'WRITE STATUS PARAMETERS TO FILE:
WRITE(1,*),' ITERATION NUMBER ',IT
WRITE(1,*),' OBJECTIVE FUNCTION AT X ',F(X)
WRITE(1,*),' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV
WRITE(1,*)'-------------------------------------------------------
&-----------------------I
WRITE(1,*)' I IPARTN X' 
WRITE(1,571)(I,IPARTN(I),X(I),I=1,NJ)
WRITE(1,*)'-------------------------------------------------------
&-----------------------I
WRITE(*,*)'******************************************************'
WRITE(1,*),'******************
WRITE(*,*)'******************
WRITE(1,*),'******************
WRITE(1,*),'******************
WRITE(*,*)'******************************************************'
WRITE(*,*)
WRITE(*,*)'*
WRITE(*,*)'ENDIF
C
OK=.FALSE.
RETURN
ENDIF
C
C ALTER BASIS AND RE-EVALUATE JMAT IN ATTEMPT TO FIND A LINEARLY
C INDEPENDENT BASIS. USE BASIS CHANGING HEURISTICS LOOP.
DEDEND(I)=ISWTCH
GOTO 60
C
ENDIF
C
C DEFINE BASIS ARRAYS: (BASED ON CURRENT PARTITION)
C USE THE 1st THRU (K+J)th COMPONENTS OF IPARTN AS BASIC VARIABLE
C SUBSCRIPTS
DO 115 I=1,KPJ
   IB=IPARTN(I)
   XBAS(I)=X(IB)
   GRDFB(I)=GRDF(IB)
115 CONTINUE
C
C DEFINE NONBASIS ARRAYS & VECTORS: (BASED ON CURRENT PARTITION)
C USE THE K+J+1 THRU N+J COMPONENTS OF IPARTN AS NON-BASIS VARIABLE
C SUBSCRIPTS
DO 120 I=1,NMK
INB=IPARTN(I+KPJ)
XNBAS(I)=X(INB)
GRDFNB(I)=GRDF(INB)

120 CONTINUE
C
DO 140,I=1,KPJ
   DO 140,M=1,NMK
      CMAT(I,M)=GRDCON(I,IPARTN(M+KPJ))
140 CONTINUE
C
C DETERMINE THE REDUCED GRADIENT
CALL MATMUL(JINV,KPJ,KPJ,CMAT,KPJ,NMK,CONVRT)
CALL VMATMUL(GRDFB,KPJ,CONVRT,KPJ,NMK,VECNB)
DO 150,I=1,NMK
   REDGRD(I)=GRDFNB(I)-VECNB(I)
150 CONTINUE
C
C STEP 2 CHECK FOR OPTIMUM SOLUTION: MAGNITUDE OF REDGRD < TOLL
C
CALL VMATMUL(REDGRD,NMK,REDGRD,NMK,1,MAG)
IF (DSQRT(MAG).LE.TOLL)THEN
C REDUCED GRADIENT HAS MAGNITUDE LESS THAN OR EQUAL TO TOLL
C TERMINATE WITH STATUS DATA DUMP
C
C BY PASS PRINT OUT?
   IF (PRINT)THEN
C WRITE TO THE SCREEN
   WRITE(*,*)
   WRITE(*,*)' ITERATION NUMBER ',IT
   WRITE(*,*)' OBJECTIVE FUNCTION AT X ',F(X)
   WRITE(*,*)' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV
   WRITE(*,*)'----------------------------------------'
   WRITE(*,*)' I IPARTN X GRDF REDGRD'
   WRITE(*,573)(I,IPARTN(I),X(I),GRDF(I),REDGRD(I),I=1,NJ)
   WRITE(*,*)'----------------------------------------'
   WRITE(*,*)
   WRITE(*,*)' AN OPTIMUM SOLUTION HAS BEEN FOUND. THE REDUCED <==
   WRITE(*,*)
   WRITE(*,*)' GRADIENT HAS MAGNITUDE LESS THAN OR EQUAL TO TOLL <==
   WRITE(*,*)
   WRITE(*,*)'----------------------------------------'
   WRITE(*,*)
   WRITE(*,*)'----------------------------------------'
   WRITE(*,*)
C WRITE STATUS PARAMETERS TO FILE:
   WRITE(1,*)
   WRITE(1,*)' ITERATION NUMBER ',IT
   WRITE(1,*)' OBJECTIVE FUNCTION AT X ',F(X)
   WRITE(1,*)' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV
WRITE(1,*)'----------------------------------------'
&----------------------------------------
WRITE(1,*)' I IPARTN X GRDF REDGRD'
WRITE(1,573)(I,IPARTN(I),X(I),GRDF(I),REDGRD(I),I=1,NJ)
WRITE(1,*)'----------------------------------------'
&----------------------------------------
WRITE(1,*)' AN OPTIMUM SOLUTION HAS BEEN FOUND. THE REDUCED <=
&----------------------------------------
WRITE(1,*)' GRADIENT HAS MAGNITUDE LESS THAN OR EQUAL TO TOL1 <=
&----------------------------------------
WRITE(1,*)' ' 
WRITE(1,*)'----------------------------------------'
&----------------------------------------
C
C ENDIF
C
OK=.TRUE.
RETURN
END
C
C DEFINE SEARCH VECTOR:
C CHECK THE BOUNDS ON EACH OF THE N-K INDEPENDENT (NON-BASIC) VARIABLES
C TO SEE IF THEY WILL BE VIOLATED WHILE SEARCHING IN THE MINUS REDUCED
C GRADIENT DIRECTION
DO 160,I=1,NMK
INB=IPARTN(I+KPJ)
IF (((UPRBND(INB).LE.TOL3(INB)).AND.(REDGRD(I).LT.0.0))THEN
DNBAS(I)=0.0
ELSEIF (((LWRBND(INB).LE.TOL3(INB)).AND.(REDGRD(I).GT.0.0))THEN
DNBAS(I)=0.0
ELSE
DNBAS(I)=(-1.0)*REDGRD(I)
ENDIF
D(INB)=DNBAS(I)
160 CONTINUE
C
CALL MATMUL(CONVRT,KPJ,NMK,DNBAS,NMK,1,DBAS)
C
DO 170,I=1,KPJ
DBAS(I)=(-1.0)*DBAS(I)
D(IPARTN(I))=DBAS(I)
170 CONTINUE
C
C CHECK THE BOUNDS ON EACH OF THE K+J DEPENDENT (BASIC) VARIABLES TO
C SEE IF THEY WILL BE VIOLATED WHILE SEARCHING IN THE DBAS DIRECTION
DO 180,I=1,KPJ
IB=IPARTN(I)
IF (((UPRBND(IB).LE.TOL3(IB)).AND.(DBAS(IB).GT.0.0))THEN
DBAS(IB)=0.0
ELSEIF (((LWRBND(IB).LE.TOL3(IB)).AND.(DBAS(IB).LT.0.0))THEN
DBAS(IB)=0.0
ENDIF
D(IB)=DBAS(I)

180  CONTINUE
C
C CHECK MAGNITUDE OF NON-BASIC SEARCH VECTOR:
C MAGNITUDE OF [ DNBAS ] < TOL1 ?
C
C NOTE: WHEN DNBAS HAS A NEARZERO NORM, A KUHN-TUCKER POINT HAS BEEN
C REACHED, NO DESCENT DIRECT CAN BE FOUND.
C
CALL VMAATMUL(DNBAS,NMK,DNBAS,NMK,1,MAG)
C IF (DSQRT(MAG).LE.TOLL)THEN
C THE NONBASIC SEARCH VECTOR HAS MAGNITUDE LESS THAN OR EQUAL TO TOLL
C TERMINATE WITH STATUS DATA DUMP
C
C BY PASS PRINT OUT?
  IF (PRINT)THEN
C
C WRITE TO THE SCREEN
WRITE(*,*)' ' 
WRITE(*,*)' ITERATION NUMBER ',IT 
WRITE(*,*)' OBJECTIVE FUNCTION AT X ',F(X) 
WRITE(*,*)' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV 
WRITE(*,*)' '-----------------------------------------------
&-----------------------&
WRITE(*,*)' I IPARTN  X  D  GRDF
& REDGRD 
WRITE(*,574)(I,IPARTN(I),X(I),D(I),GRDF(I),REDGRD(I),I=1,NJ)
WRITE(*,*)' '-----------------------------------------------
&-----------------------&
WRITE(*,*)' AN OPTIMUM SOLUTION HAS BEEN FOUND. THE SEARCH <===
&-----------------------&
WRITE(*,*)' VECTOR HAS MAGNITUDE LESS THAN OR EQUAL TO TOLL <===
&-----------------------&
WRITE(*,*)' '-----------------------------------------------
&-----------------------&

C WRITE STATUS PARAMETERS TO FILE:
WRITE(1,*)' ' 
WRITE(1,*)' ITERATION NUMBER ',IT 
WRITE(1,*)' OBJECTIVE FUNCTION AT X ',F(X) 
WRITE(1,*)' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV 
WRITE(1,*)' '-----------------------------------------------
&-----------------------&
WRITE(1,574)(I,IPARTN(I),X(I),D(I),GRDF(I),REDGRD(I),I=1,NJ)
WRITE(1,*)' '-----------------------------------------------
&-----------------------&
&=================================='
WRITE(1,'')'
WRITE(1,'') AN OPTIMUM SOLUTION HAS BEEN FOUND. THE SEARCH <===
&=================================='
WRITE(1,'') VECTOR HAS MAGNITUDE LESS THAN OR EQUAL TO TOL1 <===
&=================================='
WRITE(1,'')'
WRITE(1,'')==============================================
&=================================='

C

ENDIF
C

OK=.TRUE.
RETURN
ENDIF

C

STEP 3 alpha search loop <========
C

Determine new trial point using search vector d & current
step size alpha. Use Newton's iterations to converge on a
feasible point. Check for improvement in the objective
function at the trial point. Check for violation of variable
bounds before & after Newton's iteration.
C

Start search for point v1:
Exit search loop if all three of the following are true:
V1 satisfies all constraints
Objective function is reduced [ ie F(V1) < F(X) ]
V1 violates no variable bounds
C

Number of restarts for alpha search loop is limited to Newtit
C

alpha=alphao
DO 550, RESTART=1, NEWTIT
C

Determine new trial point (step size is alpha)
C
200 DO 220, I=1, KPJ
   V(IPARTN(I))=XBAS(I)+(ALPHA*DBAS(I))
220 CONTINUE
C

Note: once alpha is determined, the nonbasis part of v does not change
throughout the newton iterations
C

DO 240, I=1, NMK
   INB=IPARTN(I+KPJ)
   V(INB)=XBAS(I)+(ALPHA*DBAS(I))
   V1(INB)=V(INB)
240 CONTINUE
C

Check to see if there are any variable bounds violated at v
C
If a bound is found to be violated, use linear interpolation to
C BRING IT BACK. ONLY FIX THE FIRST VARIABLE BOUND FOUND TO BE VIOLATED.
C RESTART WITH THE REDUCED ALPHA (STEP SIZE).
C
DO 250, I=1, NPJ
   IB = IPARTN(I)
   IF (V(IB).LT.LOWER(IB)) THEN
      ALPHA = ALPHA*(LWRND(IB))/DABS(X(IB)-V(IB))
      ALPHA = ALPHA*0.9
   C AT LEAST ONE VARIABLE LOWER BOUND HAS BEEN VIOLATED
   C RESTART STEP 3 WITH NEW ALPHA
   C
   GOTO 550
ELSEIF (V(IB).GT.UPPER(IB)) THEN
   ALPHA = ALPHA*(UPRNDBD(IB))/DABS(X(IB)-V(IB))
   ALPHA = ALPHA*0.9
C AT LEAST ONE VARIABLE UPPER BOUND HAS BEEN VIOLATED
C RESTART STEP 3 WITH NEW ALPHA
C
   GOTO 550
ENDIF
250 CONTINUE
C NO VARIABLE BOUNDS WERE FOUND TO BE VIOLATED,
C CONTINUE WITH NEWTON'S ITERATIONS
C
C PERFORM NEWTON'S ITERATIONS <===================================
C NOTE: MAXIMUM NUMBER OF ITERATIONS ALLOWED IS NEWTIT
C
260 DO 400, INEWT=1, NEWTIT
C
C CHECK THE FEASIBILITY IF POINT V (ie DOES V SATISFY THE CONSTRAINTS?)
C
   CALL CNSTRNT(V, HCON, GCON, TOL2, CON, FSBLE)
C
THIS SUBROUTINE RETURNS A LOGICAL FLAG CONCERNING FEASIBILITY
C OF THE POINT:
C   IF POINT IS FEASIBLE =====> FSBLE=.TRUE.
C   IF POINT IS NOT FEASIBLE => FSBLE=.FALSE.
C
   IF (FSBLE) THEN
C POINT IS FEASIBLE: EQUATE THE BASIC PART OF V AND V1
C
EXiT NEWTON LOOP
C
CHECK FOR OBJECTIVE FUNCTION IMPROVEMENT AT V1
C
DO 270, M=1, KPJ
   V1(IPARTN(M)) = V(IPARTN(M))
270 CONTINUE
GOTO 500
ENDIF
C POINT IS NOT FEASIBLE THEREFORE......
C UPDATE THE BASIC PARTITION OF V USING NEWTON'S ITERATIONS.
C THE GRADIENTS AT V ARE REQUIRED TO CONSTRUCT JMAT AT POINT V.
C V1 = V - JINV [CON]
C
C CONSTRUCT COMBINED CONSTRAINT GRADIENT MATRIX AT V
C
CALL GRADCN(V,GRADH,GRADG,GRDCON)
C
C DEFINE JMAT ACCORDING TO THE CURRENT BASIS DEFINITION
C
DO 300,I=1,KPJ
   DO 300,M=1,KPJ
      JMAT(I,M)=GRDCON(I,IPARTN(M))
300 CONTINUE
C
C FIND INVERSE OF JMAT: USE GAUSSIAN ELIMINATION SUBROUTINE
C
TTLINV=TTLINV+1
   CALL GSSINV(JMAT,KPJ,KPJ,JINV,TOL1,ISWTCH,FLAG)
C
C CHECK FOR EXISTENCE OF INVERSE VIA LOGICAL FLAG
C IF FLAG=FALSE THEN JMAT IS SINGULAR & ALPHA (STEP SIZE) MUST BE
C CHANGED BECAUSE BASIS COMPOSITION CAN'T BE ALTERED
C AT THIS TIME.
C ELSE CONTINUE
C
   IF (.NOT.(FLAG))THEN
      C ALPHA UPDATE NEEDED, RESTART STEP 3 WITH SMALLER ALPHA
C      ALPHA=gamma*ALPHA
      GOTO 550
   ENDIF
C
C DETERMINE JINV [CON] AND UPDATE THE BASIC PARTITION OF VECTOR V1
C
   CALL MATMUL(JINV,KPJ,KPJ,CON,KPJ,1,CORRCT)
C
   DO 320,I=1,KPJ
      V1(IPARTN(I))=V(IPARTN(I))-CORRCT(I)
320 CONTINUE
C
C CHECK FOR CONVERGENCE OF NEWTON'S ITERATIONS
C
   DO 360,I=1,KPJ
      IB=IPARTN(I)
      IF (DABS(V1(IB)-V(IB)).GT.TOL3(IB))THEN
C
C NO CONVERGENCE: INCREMENT V
C
      CONTINUE WITH NEWTON'S ITERATIONS
C
   ENDIF 550
C
    DO 340,M=1,KPJ
      V(IPARTN(M))=V1(IPARTN(M))
 340     CONTINUE
      GOTO 400
  ENDIF
360     CONTINUE
C
C CONVERGENCE HAS BEEN ACHIEVED: EXIT NEWTON LOOP AND TEST
C FOR FEASIBILITY
C
      GOTO 420
400   CONTINUE
C
C END OF NEWTONS ITERATION LOOP
C
C NEWTIT EXCEEDED: REDUCE ALPHA AND RESTART STEP 3
C
      ALPHA=ALPHA*GAMMA
      GOTO 550
C
420   CONTINUE
C
C NEWTON'S ITERATIONS HAVE CONVERGED
C IF V1 IS FEASIBLE: THEN CHECK FOR OBJECTIVE FUNCTION IMPROVEMENT
C ELSE REDUCE ALPHA (STEP SIZE) AND RESTART STEP 3
C
C CALL CONSTRAINT SUBROUTINE
C
      CALL CNSTRT(V1,HCON,GCON,TOL2,CON,FSBLE)
C
THIS SUBROUTINE RETURNS A LOGICAL FLAG CONCERNING FEASIBILITY
OF THE POINT:
C IF POINT IS FEASIBLE ======> FSBLE=.TRUE.
C IF POINT IS NOT FEASIBLE ====> FSBLE=.FALSE.
C
      IF (.NOT.(FSBLE))THEN
C
C V1 WAS NOT FEASIBLE, REDUCE ALPHA AND RESTART STEP 3 (ALPHA SEARCH)
C
      ALPHA=ALPHA*GAMMA
      GOTO 550
      ENDIF
C
500   CONTINUE
C
C POINT IS FEASIBLE, CHECK FOR OBJECTIVE FUNCTION IMPROVEMENT
C
      IF (F(X).LE.F(V1))THEN
C
C NO IMPROVEMENT, REDUCE ALPHA AND RESTART STEP 3 (ALPHA SEARCH)
C

\[ \text{ALPHA} = \text{ALPHA} \times \text{GAMMA} \]

\[ \text{GOTO 550} \]

ENDIF

C

C YES, THERE IS IMPROVEMENT BUT... ARE ANY VARIABLE BOUNDS VIOLATED??
C IF A BOUND IS FOUND TO BE VIOLATED, USE LINEAR INTERPOLATION TO BRING
C IT BACK. ONLY FIX THE FIRST VARIABLE BOUND FOUND TO BE VIOLATED.
C RESTART STEP 3 WITH THE REDUCED ALPHA (STEP SIZE).
C
\[ \text{DO 520, I=1,NPJ} \]
\[ \text{IB=IPARTN(I)} \]
\[ \text{IF (V1(IB).LT.XLOWER(IB)) THEN} \]
\[ \text{IF (LWREND(IB).EQ.0.0) THEN} \]
\[ \text{ALPHA} = \text{ALPHA} \times 0.9 \]
\[ \text{ELSE} \]
\[ \text{ALPHA} = \text{ALPHA} \times (LWREND(IB)/DABS(X(IB)-V1(IB))) \]
\[ \text{ALPHA} = \text{ALPHA} \times 0.9 \]
ENDIF

C

C AT LEAST ONE VARIABLE LOWER BOUND HAS BEEN VIOLATED
C RESTART STEP 3 (ALPHA SEARCH) WITH NEW ALPHA
C
\[ \text{GOTO 550} \]

ELSEIF (V1(IB).GT.XUPPER(IB)) THEN
\[ \text{IF (UPREND(IB).EQ.0.0) THEN} \]
\[ \text{ALPHA} = \text{ALPHA} \times 0.9 \]
\[ \text{ELSE} \]
\[ \text{ALPHA} = \text{ALPHA} \times (UPREND(IB)/DABS(X(IB)-V1(IB))) \]
\[ \text{ALPHA} = \text{ALPHA} \times 0.9 \]
ENDIF

C

C AT LEAST ONE VARIABLE UPPER BOUND HAS BEEN VIOLATED
C RESTART STEP 3 (ALPHA SEARCH) WITH NEW ALPHA
C
\[ \text{GOTO 550} \]

ENDIF

520 CONTINUE

C

C NO VARIABLE BOUNDS WERE FOUND TO BE VIOLATED
C EXIT ALPHA SEARCH LOOP & CONTINUE
C
\[ \text{GOTO 570} \]

C

550 CONTINUE

C

C ALPHA SEARCH LOOP HAS ITERATED NEWTIT TIMES WITHOUT SUCCESSFUL
C CONVERGENCE. THIS MEANS NO REASONABLE STEP SIZE IN THIS PARTICULAR
C ITERATION'S SEARCH DIRECTION PRODUCES A FEASIBLE POINT WITH A
C REDUCTION IN OBJECTIVE FUNCTION. TRY ALTERING THE BASIS
C 'HEURISTICALLY' AND RESTARTING FROM THE TOP.
C
C
C THE REASON FOR EMPLOYING THIS HEURISTIC BASIS CHANGE IS TO ATTEMPT
TO FIND A BASIS SPACE WHERE PROGRESS (IE OBJECTIVE FUNCTION
REDUCTION) CAN BE MADE. ALTERATION LOGIC IS IDENTICAL TO PREVIOUSLY
DESCRIBED TECHNIQUE.

NOTE: THE LIMIT FOR THE NUMBER OF BASIS ALTERATIONS
WITHIN ONE PARTICULAR ITERATION IS (N-K)*(K+J)

REPEAT=REPEAT+1
IF (REPEAT.LE.(NMKJP))THEN
   GOTO 60
ELSE

ALPHA SEARCH RESTART LIMIT HAS BEEN ATTAINED NEWTIM TIMES WITHOUT
SUCCESSFUL CONVERGENCE. THE BASIS HAS BEEN ALTERED (N-K)*(K+J) TIMES
AND STILL NO PROGRESS. TERMINATE WITH WARNING

BY PASS PRINT OUT?
   IF (PRINT)THEN

WRITE TO SCREEN
   WRITE(*,*)
   WRITE(*,*)'******************************************************'
   WRITE(*,*)'****************** WARNING ******************'
   WRITE(*,*)'NO PROGRESS IN THIS CYCLE..... BASIS HAS BEEN ALTERED'
   WRITE(*,*)' (N-K)*(K+J) TIMES & STILL NO PROGRESS.'
   WRITE(*,*)'
   WRITE(*,*)'WITHIN EACH TRIAL BASIS, STEP SIZE ALPHA HAS BEEN'
   WRITE(*,*)'REDUCED TO ITS LIMIT WITHOUT SUCCESSFUL CONVERGENCE.'
   WRITE(*,*)'ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV
   WRITE(*,*)'WARNING ********************'
   WRITE(*,*)'******************************************************'
   WRITE(*,*)'

WRITE TO FILE
   WRITE(1,*,'
   WRITE(1,*)'******************************************************'
   WRITE(1,*)'*********** WARNING ***********'
   WRITE(1,*)'NO PROGRESS IN THIS CYCLE.....BASIS HAS BEEN ALTERED'
   WRITE(1,*)'(N-K)*(K+J) TIMES & STILL NO PROGRESS.'
   WRITE(1,*)
   WRITE(1,*)'WITHIN EACH TRIAL BASIS, STEP SIZE ALPHA HAS BEEN'
   WRITE(1,*)'REDUCED TO ITS LIMIT WITHOUT SUCCESSFUL CONVERGENCE.'
   WRITE(1,*)'ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ',TTLINV
   WRITE(1,*)'WARNING ********************'
   WRITE(1,*)'******************************************************'
   WRITE(1,*)

C

ENDIF
C
OK=.FALSE.
RETURN
ENDIF

C
570 CONTINUE
C
STEP 3 HAS CONVERGED TO V1 WHICH IS A FEASIBLE POINT THAT DOESN'T
VIOLATE ANY VARIABLE BOUNDS AND THE OBJECTIVE FUNCTION IS REDUCED
COMARED TO F(X)
C
C BY PASS PRINT OUT?
IF (PRINT) THEN
C OUTPUT STATUS PARAMETERS TO THE SCREEN:
WRITE(*,*) ''
WRITE(*,*) ' ITERATION NUMBER ', IT
WRITE(*,*) ' FINAL STEP SIZE = ', ALPHA
WRITE(*,*) ' OBJECTIVE FUNCTION AT X ', F(X)
WRITE(*,*) ' OBJECTIVE FUNCTION AT V1 ', F(V1)
WRITE(*,*) ' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ', TTLINV
WRITE(*,*) ''
WRITE(*,575)(I,IPARTN(I),X(I),D(I),V1(I),GRDF(I),REDGRD(I),I=1,NJ)
WRITE(*,*) ''
C
C WRITE STATUS PARAMETERS TO FILE:
WRITE(1,*) ''
WRITE(1,*) ' ITERATION NUMBER ', IT
WRITE(1,*) ' FINAL STEP SIZE = ', ALPHA
WRITE(1,*) ' OBJECTIVE FUNCTION AT X ', F(X)
WRITE(1,*) ' OBJECTIVE FUNCTION AT V1 ', F(V1)
WRITE(1,*) ' ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: ', TTLINV
WRITE(1,*) ''
WRITE(1,575)(I,IPARTN(I),X(I),D(I),V1(I),GRDF(I),REDGRD(I),I=1,NJ)
WRITE(1,*) ''
C
ENDIF

C
571 FORMAT(I3,I6,1E14.7)
572 FORMAT(I3,I6,2E14.7)
573 FORMAT(I3,I6,3E14.7)
574 FORMAT(I3,I6,4E14.7)
575 FORMAT(I3,I6,5E14.7)
C CHECK TO SEE IF THIS IS THE SAME POINT TO WHICH NEWTON'S ITERATIONS
C CONVERGED TO LAST ITERATION, ie Vl(i) = X(i) ? FOR i = 1 to N+J
DO 590,I=1,NPJ
   IF (DABS(Vl(I)-X(I)).GT.TOL3(I))THEN
C C ONE COMPONENT WAS FOUND TO BE DIFFERENT, THEREFORE X & Vl ARE
C DIFFERENT, EXIT LOOP.
C
   GOTO 595
ENDIF
590 CONTINUE
C C NEWTON'S ITERATIONS HAVE CONVERGED TO THE SAME POINT IT STARTED WITH.
C TERMINATE WITH WARNING!
C
C BY PASS PRINT OUT?
   IF (PRINT)THEN
C C WRITE TO THE SCREEN
      WRITE(*,*)'*
      WRITE(*,*)'*******************************************
      WRITE(*,*)'******* WARNING **********************'*
      WRITE(*,*)'*******************************************
      WRITE(*,*)'NO PROGRESS IN THIS CYCLE.......'
      WRITE(*,*)'ALGORITHM HAS CONVERGED TO THE SAME POINT TWICE !'
      WRITE(*,*)'*******************************************
      WRITE(*,*)'*******************************************
      WRITE(1,*)'NO PROGRESS IN THIS CYCLE.......'
      WRITE(1,*)'ALGORITHM HAS CONVERGED TO THE SAME POINT TWICE !'
      WRITE(1,*)'*******************************************
      WRITE(1,*)'*******************************************
      WRITE(1,*)'
C C WRITE TO FILE
      WRITE(1,*)'*
      WRITE(1,*)'*******************************************
      WRITE(1,*)'******* WARNING **********************'
      WRITE(1,*)'*******************************************
      WRITE(1,*)'NO PROGRESS IN THIS CYCLE.......'
      WRITE(1,*)'ALGORITHM HAS CONVERGED TO THE SAME POINT TWICE !'
      WRITE(1,*)'*******************************************
      WRITE(1,*)'*******************************************
      WRITE(1,*)'
C ENDIF
C
   OK=.FALSE.
   RETURN
C C VECTOR Vl IS NOT EQUAL TO VECTOR X
C UPDATE VECTOR X & RETURN TO STEP 1
C
595 DO 600,1=1,NPJ
X(I)=V1(I)
600 CONTINUE
C
1000 CONTINUE
C
C MAXIMUM NUMBER OF ALGORITHM ITERATIONS (MAXIT) EXCEEDED
C
C BY PASS PRINT OUT?
IF (PRINT) THEN
C
C WRITE TO THE SCREEN
WRITE(*,*)'
WRITE(*,*)'************** WARNING **************'
WRITE(*,*)'MAX NUMBER OF ITERATIONS EXCEEDED WITHOUT CONVERGENCE'
WRITE(*,*)'************** WARNING **************'
WRITE(*,*)'************** MAX IT **************'
WRITE(*,*)'
C
C WRITE TO FILE
WRITE(1,*)'
WRITE(1,*)'************** WARNING **************'
WRITE(1,*)'MAX NUMBER OF ITERATIONS EXCEEDED WITHOUT CONVERGENCE'
WRITE(1,*)'************** WARNING **************'
WRITE(1,*)'************** MAX IT **************'
WRITE(1,*)'
C
ENDIF
C
OK=.FALSE.
RETURN
END
C
C=======================================================================
C THE FOLLOWING ARE A COLLECTION OF SUBPROGRAMS USED FOR MATRIX & VECTOR
C MANIPULATION.
C=======================================================================
C
SUBROUTINE BSORT(A,ISEQ,LENGTH,FLAG)
C
SUBPROGRAM NAME: BSORT.FOR
C
BY: TOM VEILLEUX
C
DATE: 3-28-91
C
PURPOSE: THIS PROGRAM EMPLOYS A BUBBLE SEARCH STRATEGY TO SORT
C VECTOR A FORM LARGEST TO SMALLEST COMPONENT. VECTORS
A and ISEQ are both inputs and outputs. At the start
of the algorithm the ILOCAL vector represents the original
order of the components. Vector ISEQ returns reflecting
how the original sequence was shuffled in the sorting
process. LENGTH is the length of the vectors.

Arguments:

A: Input = vector to be sorted

Output = sorted vector

ISEQ: Input = original sequence of vector A's components

Output = final arrangement of vector A's components

LENGTH: Input = length of vector to be sorted

FLAG: Output = logical flag to indicate changes in the variable
part'n

C=------------------------------------------------------------------------

PARAMETER (MX=10)
INTEGER ISEQ(MX), ILOCAL(MX), LENGTH, ITEMP
REAL*8 A(MX), TEMP
LOGICAL FLAG

C DEFINE LOCAL PARTITION
DO 5, I=1, LENGTH
  ILOCAL(I)=I
5 CONTINUE

DO 10, M=LENGTH, 2, -1
  DO 10, I=1, M-1
    IF (A(I).LT.A(I+1)) THEN
      TEMP=A(I)
      A(I)=A(I+1)
      A(I+1)=TEMP
      C UPDATE ISEQ
      ITEMP=ILocal(I)
      ILOCAL(I)=ILocal(I+1)
      ILOCAL(I+1)=ITEMP
    ENDIF
10 CONTINUE

FLAG=.TRUE.

C IF ILOCAL DIFFERS FROM ISEQ, EQUATE ISEQ(i) & ILOCAL(i) (FLAG=.TRUE.)
C IF ILOCAL DID NOT CHANGE THEN RETURN THE ORIGINAL INTEGER SEQUENCE
C AND FLAG=.FALSE.
DO 20, I = 1, LENGTH
  IF (ISEQ(I).NE.ILOCAL(I)) THEN
    FLAG = .TRUE.
    DO 15, II = 1, LENGTH
      ISEQ(II) = ILOCAL(II)
    15    CONTINUE
    RETURN
  ELSE
    FLAG = .FALSE.
  ENDIF
C
20    CONTINUE
C
RETURN
END

SUBROUTINE GRADCN(X, GRADH, GRADG, GRDCON)
C
SUBPROGRAM NAME: GRADCN FOR
BY:        TOM VEILLEUX
DATE:      5-6-91

PURPOSE: THIS PROGRAM FORMS A MATRIX FROM THE GRADIENTS OF THE
EQUALITY AND INEQUALITY CONSTRAINTS EVALUATED AT X.

ARGUMENTS:
X:        POINT AT WHICH THE GRADIENT MATRIX IS TO BE DETERMINED
(INPUT)
GRADH:    SUBROUTINE THAT DETERMINES THE GRADIENTS OF THE K EQUALITY
CONSTRAINTS (INPUT)
GRADG:    SUBROUTINE THAT DETERMINES THE GRADIENTS OF THE J INEQUALITY
CONSTRAINTS (INPUT)
GRDCON:   COMBINED GRADIENTS MATRIX OF THE K EQUALITY AND THE J
INEQUALITY CONSTRAINTS, EVALUATED AT THE POINT X (OUTPUT)

PARAMETER (MX=10)
INTEGER NC, KC, JC, NPJC, NMKC, KPJC, IT
REAL*8 X(MX), GRDH(MX, MX), GRDG(MX, MX), GRDCON(MX, MX)
EXTERNAL GRADH, GRADG
COMMON NC, KC, JC, NPJC, NMKC, KPJC

CALL GRADH(X, GRDH)
CALL GRADG(X, GRDG)

CONSTRUCT THE COMBINED GRADIENT MATRIX AT X
DO 30,I=1,KPJC
  IF (I.LE.KC) THEN
    DO 10,M=1,NC
      GRDCON(I,M)=GRDH(I,M)
    10 CONTINUE
  ELSE
    DO 20,M=1,NC
      GRDCON(I,M)=GRDG(I-KC,M)
    20 CONTINUE
  ENDIF
30 CONTINUE

C=====================================================================
C SUBROUTINE CNSTRT(X,HCON,GCON,TOL2,CON,FSBLE)
C=====================================================================
C SUBPROGRAM NAME: CNSTRT.FOR
C BY: TOM VEILLEUX
C DATE: 5-5-91
C PURPOSE: THIS PROGRAM FORMS A VECTOR FROM THE OF THE KPJ
C EQUALITY AND INEQUALITY CONSTRAINTS EVALUATED AT X.
C THIS SUBROUTINE RETURNS A LOGICAL FLAG CONCERNING
C FEASIBILITY OF THE POINT
C ARGUMENTS:
C X: POINT AT WHICH THE CONSTRAINT VECTOR IS TO BE DETERMINED
C (INPUT)
C HCON: SUBROUTINE THAT DETERMINES THE VALUE OF EACH OF THE K
C EQUALITY CONSTRAINTS (INPUT)
C GCON: SUBROUTINE THAT DETERMINES THE VALUE OF EACH OF THE K
C INEQUALITY CONSTRAINTS (INPUT)
C TOL2: TOLERANCE USED FOR FEASIBILITY TEST (INPUT)
C CON: COMBINED VECTOR OF THE K EQUALITY AND THE J INEQUALITY
C CONSTRAINTS, EVALUATED AT THE POINT X (OUTPUT)
C FSBLE: LOGICAL FLAG INDICATING FEASIBILITY OF THE POINT X
C IF (DABS(CON(I)).LE.TOL2) FOR ALL I = 1 TO KPJ
C THEN  FSBLE = .TRUE.
C ELSE  FSBLE = .FALSE.
C=====================================================================
PARAMETER (MX=10)
INTEGER NC,KC,JC,NPJC,NMKC,KPJC
REAL*8 X(MX),H(MX),G(MX),CON(MX),TOL2(MX)
SUBROUTINE GSSINV(A,MA,NA,AINV,TOL,ISWTCH,FLAG)

SUBPROGRAM NAME: GSSINV
BY:        TOM VEILLEUX
DATE:      3-21-91

PURPOSE:   THIS SUBPROGRAM IS USED TO DETERMINE THE INVERSE
OF A MATRIX USING GAUSSIAN ELIMINATION. THE
ALGORITHM RETURNS A DIAGNOSTIC FLAG THAT IS
USED TO INDICATE THE EXISTENCE OF THE INVERSE. THE
INVERSE DOES NOT EXIST WHEN EITHER A IS NOT SQUARE
OR ITS RANK IS N-1 OR LESS. MATRIX A IS THE INPUT
AND AINV IS THE OUTPUT.

ARGUMENTS:
A:         MATRIX A (INPUT)
MA & NA:    ORDER OF MATRIX A (INPUT)
AINV:       INVERSE OF MATRIX A (OUTPUT)
TOL:        TOLERANCE USED IN TESTING FOR MAIN DIAGONAL ZEROS
            APPROXIMATELY 1D-5 TO 1D-8
ISWTCH:     NUMBER OF THE LINEARLY DEPENDENT ROW (COLUMN)
RETURNED WHEN ROWS ARE SWITCHED IN FINDING THE INVERSE

FLAG1: DENOTES SUCCESS OF MATRIX INVERSE OPERATION (OUTPUT)

FLAG1 = 'FALSE' OPERATION UNSUCCESSFUL OR UNDEFINED

FLAG1 = 'TRUE' FOR NORMAL SUCCESSFUL COMPLETION

PARAMETER (MX=10)
INTEGER MA,NA,ISEQ(MX),ISWTCH,ITEMP
REAL*8 A(MX,MX),AINV(MX,MX),TEMP,B(MX,MX),TOL
LOGICAL FLAG1

DEFINE INITIAL SEQUENCE VECTOR

DO 5,I=1,MA
    ISEQ(I)=I
5    CONTINUE

TEST FOR SQUARE MATRIX

IF(NA.NE.MA)THEN
    INVERSE DOES NOT EXIST, RETURN TO MAIN PROGRAM

    FLAG1=.FALSE.
    ISWTCH=0
    RETURN
ENDIF

INITIALLY DEFINE MATRIX AINV AS THE IDENTITY MATRIX
COPY MATRIX A INTO MATRIX B

DO 10,I=1,NA
    DO 10,M=1,NA
        IF (I.EQ.M)THEN
            AINV(I,M)=1.0
        ELSE
            AINV(I,M)=0.0
        ENDIF
        B(I,M)=A(I,M)
10    CONTINUE

START GAUSSIAN ELIMINATION FROM TOP DOWN

DO 90,I=1,NA
    IF (DABS(B(I,I)).GT.TOL)THEN
        NORMALIZE THE ROW RELATIVE TO THE MAIN DIAGONAL COMPONENT BY DIVIDING THE ROW BY THE MAIN DIAGONAL COMPONENT
        TEMP=B(I,I)
DO 20, M=1, NA
   B(I, M)=B(I, M)/TEMP
   AINV(I, M)=AINV(I, M)/TEMP
20    CONTINUE
ELSE
C THERE IS A ZERO ON MAIN DIAGONAL
C IF THIS OCCURS IN THE BOTTOM ROW (I = NA) RETURN TO MAIN PROGRAM
C INVERSE DOES NOT EXIST
C
   IF (I.EQ.NA) THEN
      FLAG1=.FALSE.
      ISWTCH=ISEQ(MA)
      RETURN
   ENDIF
C
C SEARCH FOR A ROW WITH A NON-ZERO Ith COMPONENT TO SWITCH WITH
C
   DO 50, II=I+1, NA
      IF (DABS(B(II, I)).GT.TOL) THEN
         C A SUITABLE RCM HAS BEEN FOUND, SWITCH ROWS I & II AND EXIT SEARCH LOOP
         C
         SWITC ISEQ
            ITEMP=ISEQ(I)
            ISEQ(I)=ISEQ(II)
            ISEQ(II)=ITEMP
         C
         DO 30, JJ=1, NA
         C SWITCH B
            TEMP=B(I, JJ)
            B(I, JJ)=B(II, JJ)
            B(II, JJ)=TEMP
         C
         SWITCH AINV
            TEMP=AINV(I, JJ)
            AINV(I, JJ)=AINV(II, JJ)
            AINV(II, JJ)=TEMP
         30    CONTINUE
C
C NORMALIZE THE NEW ROW
C
            TEMP=B(I, I)
   DO 40, M=1, NA
      B(I, M)=B(I, M)/TEMP
      AINV(I, M)=AINV(I, M)/TEMP
40    CONTINUE
   GOTO 60
ENDIF
50    CONTINUE
C
C NO SUITABLE ROW WAS FOUND TO SWITCH WITH, THEREFORE RETURN TO MAIN
C PROGRAM, INVERSE DOES NOT EXIST
FLAG1=.FALSE.
ISWTCH=ISEQ(MA)
RETURN
ENDIF

C ELIMINATE THE Ith VARIABLE FROM THE REMAINING ROWS BY SUBTRACTING
C THE Ith ROW TIMES THE Ith COMPONENT OF EACH OF THE REMAINING ROWS
C
DO 90,II=(I+1),NA
   TEMP=B(II,I)
   DO 90,JJ=1,NA
      B(II,JJ)=B(II,JJ)-(B(I,JJ)*TEMP)
      AINV(II,JJ)=AINV(II,JJ)-(AINV(I,JJ)*TEMP)
90 CONTINUE

C START GAUSSIAN ELIMINATION FROM BOTTOM UP TO THE TOP
C THE MATRIX IS NOW IN UPPER TRIANGULAR FORM
C
C ELIMINATE THE Ith VARIABLE FROM THE REMAINING ROWS BY SUBTRACTING
C THE Ith ROW TIMES THE Ith COMPONENT OF EACH OF THE REMAINING ROWS
C THIS IS BACKWARDS SUBSTITUTION
C
DO 100,I=NA,1,-1
   DO 100,II=I-1,1,-1
      TEMP=B(II,I)
      DO 100,JJ=1,NA
         B(II,JJ)=B(II,JJ)-(B(I,JJ)*TEMP)
         AINV(II,JJ)=AINV(II,JJ)-(AINV(I,JJ)*TEMP)
100 CONTINUE

C INVERSE WAS SUCCESSFULLY DETERMINED, RETURN TO MAIN PROGRAM
C
FLAG1=.TRUE.
ISWTCH=ISEQ(MA)
RETURN
END

C SUBROUTINE MATMUL(A,MA,NA,B,MB,NB,C)
C SUBPROGRAM NAME: MATMUL
C BY: TOM VEILLEUX
C DATE: 2-10-91
C PURPOSE: THIS SUBPROGRAM IS USED TO PERFORM BASIC MATRIX
C MULTIPLICATION. MATRICES A & B ARE INPUTS, AND
C MATRIX C IS OUTPUT. (C = A * B)
C ARGUMENTS:
C MA & NA: ORDER OF MATRIX A (INPUT)
C MB & NB: ORDER OF MATRIX B (INPUT)
C MC & NC: ORDER OF MATRIX C (OUTPUT)
C A: MATRIX A (INPUT)
C B: MATRIX B (INPUT)
C C: MATRIX C (OUTPUT)

C================================BEGIN SUB-PROGRAM=================================
C
INTEGER MA, NA, MB, NB, MC, NC
PARAMETER (MX=10)
REAL*8 A(MX,MX), B(MX,MX), C(MX,MX)

MC=MA
NC=NB
DO 10, I=1, MC
   DO 10, JJ=1, NC
      C(I, JJ)=0.0
   10 CONTINUE

C RETURN
END

C=================================END OF SUB-PROGRAM=================================

C

SUBROUTINE VMATMUL(A, LA, B, MB, NB, C)

SUBPROGRAM NAME: MATMUL
BY: TOM VEILLEUX
DATE: 4-11-91

PURPOSE: THIS SUBPROGRAM IS USED TO PERFORM BASIC MATRIX
         MULTIPLICATION. VECTOR A & MATRIX B ARE INPUTS, &
         VECTOR C IS OUTPUT. ([C] = [A] [B])

ARGUMENTS:

LA: LENGTH OF VECTOR A (INPUT)
MB & NB: ORDER OF MATRIX B (INPUT)
MC & NC: ORDER OF MATRIX C (OUTPUT)
A: VECTOR A (INPUT)
B: MATRIX B (INPUT)
C: VECTOR C (OUTPUT)

C================================BEGIN SUB-PROGRAM=================================
C
INTEGER LA, MB, NB, LC
PARAMETER (MX=10)
REAL*8 A(MX), B(MX,MX), C(MX)

LC=NB
DO 10, JJ=1, LC
   C(JJ)=0.0
   DO 10, KK=1, LA
      C(JJ)=C(JJ)+(A(KK)*B(KK, JJ))
10 CONTINUE
C C STATUS OUTPUT SUBROUTINE: C C THIS ROUTINE OUTPUTS XSTAR AND ITS BOUNDS. C X COMPONENTS THAT WITHIN TOL3 OF THEIR RESPECTIVE C BOUNDS ARE FLAGED AS SUCH. C SUBROUTINE STATUS(X,XLOWER,XUPPER,TOL3,F,NPJ,L,BOUNDS) C PARAMETER (MX=10) C EXTERNAL F C C VARIABLES: INTEGER NPJ,L REAL*8 F,X(MX),XUPPER(MX),XLOWER(MX),TOL3(MX) LOGICAL BOUNDS(MX) C C DETERMINE IF THE COMPONENTS OF VECTORS ARE NEAR THEIR BOUNDS C DO 10,M=1,NPJ IF (((X(M)-XLOWER(M)).LE.TOL3(M)).OR. &((XUPPER(M)-X(M)).LE.TOL3(M)))THEN BOUNDS(M)=.TRUE. ELSE BOUNDS(M)=.FALSE. ENDIF 10 CONTINUE C C WRITE TO SCREEN C WRITE(*,*)'======================================================================' WRITE(*,*)' ' WRITE(*,11) L ELSE WRITE(*,*)' AN OPTIMUM SOLUTION HAS BEEN FOUND: SYSTEM LEVEL' ENDIF 11 FORMAT(AN OPTIMUM SOLUTION HAS BEEN FOUND: SUB-SYSTEM ',I5) WRITE(*,*)' ' WRITE(*,*)' WITH IN TOL3' WRITE(*,*)' I LOWER Bound X UPPER Bound OF B &OUNDS?' WRITE(*,12) (M,XLOWER(M),X(M),XUPPER(M),BOUNDS(M),M=1,NPJ) 12 FORMAT(I4,3E14.7,1L9) WRITE(*,*)' ' WRITE(*,*)' THE MINIMUM OBJECTIVE FUNCTION VALUE AT X:',F(X)
WRITE(*,*)'========================================================================
C WRITE TO FILE
C
WRITE(1,*)'========================================================================
&========================================================================
WRITE(1,*)' '' IF (L.NE.0) THEN
WRITE(1,11)L
ELSE
WRITE(1,*)' AN OPTIMUM SOLUTION HAS BEEN FOUND: SYSTEM LEVEL'
ENDIF
WRITE(1,*)' '' WRITE(1,*)' WITH
&IN TOL3'
WRITE(1,*)' I LOWER BOUND X UPPER BOUND OF B &BOUNDS?'
WRITE(1,*)' '' WRITE(1,12)(M,XLOWER(M),X(M),XUPPER(M),BOUNDS(M),M=1,NPJ)
WRITE(1,*)' '' WRITE(1,*)' THE MINIMUM OBJECTIVE FUNCTION VALUE AT X:',F(X)
WRITE(1,*)'========================================================================
&========================================================================
C RETURN
END
C========================================================================
Appendix B: Solution of Single-Level Optimization Problems.

This appendix contains all of the information that a potential user needs to set up and solve a single-level optimization problem using the GRGOPT32 program. The first section starts with some basic information about the code, followed by detailed information on defining the problem's variables, functions and subroutines. Next, the recommended cyclical solution technique and the parameters that are used to control it are described. The following section contains an explanation of how to run GRGOPT32, including a description of the output data format, contents of the output file and error messages. The next section sets up and solves four single-level optimization problems. These problems were selected because the optimum solutions are known. In this thesis, they serve the purpose of verifying the code and exemplifying the process of converting the problem from standard format into the FORTRAN sub-program used by GRGOPT32. Each example problem is listed in the standard format used in this thesis, followed by the initial feasible starting point, the optimum solution and the pertinent set-up information for each problem. A template to use as a guide in setting up single-level problems in FORTRAN is included in the last section of this appendix.
B.1 General Usage Notes

The FORTRAN program GRGOPT32.FOR was written on an IBM compatible personal computer. The computer was a CompuAdd 386SX-16MHz with a math co-processor. The compiler and object linker used were Microsoft-FORTRAN version 3.31 and version 3.04, respectively. The full-length program, GRGOPT32.FOR is listed in Appendix A.

The optimization problem's input data must be in a format that the GRGOPT32 program can understand. For example, since the code was written in double precision, all real variables, function names, vector and arrays must be declared using the REAL*8 format. A template has been created that shows the user how to arrange the problem's dimensions, functions, constraints and so forth in the correct form. Template 1, used to set up single-level optimization problems, can be found section B.5 of this appendix.

The first step is to express the problem in the standard format used in the thesis. This format is described with equations (1.6) through (1.11).

The second step involves checking if MX, the parameter used to dimension arrays and vectors, is large enough for the problem at hand. Parameter MX should be greater than or equal to N+J, and MX should be the same in all subroutines and functions, including GRGOPT32. FORTRAN has a peculiar quirk where arrays that are passed to sub-programs must be
dimensioned identically or memory addresses will be overwritten and loss of data will result.

Next, the problem's key dimensions are defined: \( N \) the number of independent variables; \( K \), the number of equality constraints; \( J \), the number of inequality constraints; \( \text{ALPHA}_0 \), the initial step size (use \( \alpha^0 = 1 \) unless no progress is made from the \( X^{(0)} \), then use \( \alpha^0 = 10^6 \)); and \( \text{GAMMA} \), the step size reduction factor, usually 0.5.

The tolerances \( \text{TOL}_1 \), \( \text{TOL}_2 \) and \( \text{TOL}_3 \) are defined next. Good initial values for the tolerances are:

\[
\begin{align*}
\text{TOL}_1 & \approx 10^{-4} \\
\text{TOL}_{2_k} & = (\text{No analytical expression available}) \\
\text{for } k = 1, \ldots, K \\
\text{TOL}_{2_kj} & \approx 10^{-3} \times X_{k+j}^U \quad \text{for } j = 1, \ldots, J \\
\text{TOL}_{3_i} & \approx 10^{-4} \times X_i^U \quad \text{for } i = 1, \ldots, N+J 
\end{align*}
\]

This allows the convergence tolerance for the inequality constraints to be ten times bigger than the convergence tolerance for its corresponding slack variable. The convergence tolerance for an individual variable is approximately \( 10^{-4} \) times its upper bound. The convergence tolerances for the equality constraints should be on par with the convergence tolerances used for the inequality constraints. In general, optimization problems are usually first solved with a set of crude tolerances. Assuming that
some progress was made from the original \( x^{(0)} \), the tolerances are then reduced and \( x^* \), the optimum solution returned becomes the new \( x^{(0)} \) for the next cycle. The process of restarting the algorithm at the optimum solution with reduced tolerances can be repeated until convergence occurs. Typically, the tolerances are reduced by a factor of ten or one hundred between cycles. This reduction factor can be adjusted depending on the type of problem and the terminal convergence rate. By defining CYCLE and FACTOR, the user can define how many times the cycles will be repeated and by what factor the tolerances will be divided within each cycle, respectively.

The limits for the maximum number of iterations for the algorithm, MAXIT, and for the Newton's iterations, NEWTIT, are then defined. A good number for MAXIT is 100. Since the step size \( \alpha \), is reduced by GAMMA a total of NEWTIT times, not accounting for linear interpolations, the resulting reduced \( \alpha \) will be proportional to \( (\text{GAMMA})^{\text{NEWTIT}} \). If \( \text{GAMMA} = 0.5 \) and NEWTIT = 50, \( \alpha \) will be given by

\[
\alpha = \alpha^0 (\text{GAMMA})^{\text{NEWTIT}} = \alpha^0 (0.5)^{50} = \alpha^0 (8.88 \times 10^{-16})
\]

Therefore, to prevent \( \alpha \) from getting too small, NEWTIT should be about 20 to 40.

Beginners should set logical variable PRINT to .TRUE. to allow GRGOPT32 to return the maximum amount of information to the user.
Logical variable SYS is used to identify system level sub-problems in multi-level decomposition problems. In the solution of single-level problems, set SYS to false. There is an exception to this statement for very small problems. This exception is discussed in the section on errors. When SYS is false, nonbasic variables that are within TOL3 of their bounds are excluded from the basis. When SYS is true, those nonbasic variables very near their bounds are allowed to enter the basis.

The initial feasible starting point $x^{(0)}$ or $X_0$ is then defined. This vector must satisfy all the constraints within the accuracy of the tolerances and no variable bounds may be violated. A useful method for generating feasible starting points is to make sound engineering approximations for the first N variables. Next, adjust the variables so they satisfy the equality constraints. Finally, evaluate the inequality constraints and solve for the slack variables. If the slack variables are within their bounds, the point is feasible. If one or more slack variables are not within their bounds, then change the variables that reduce the violations. It is usually apparent which variables need to be changed in order to decrease stress or deflection. Start again with the equality constraints and repeat this process until the point is feasible.

The next step involves definition of the upper and lower variable bounds, $x^u = \text{XUPPER}$, and $x^l = \text{XLOWER}$, respectively.
Recall from the standard problem format that the original inequality constraint bounds $A$ and $B$ become the variable bounds for the slack variables as follows.

\[ x^L_{m,j} = A_j \quad \text{for } j = 1,2,\ldots,J \]
\[ x^U_{m,j} = B_j \quad \text{for } j = 1,2,\ldots,J \]

The user then has the option of defining the heading that will appear at the top of the output file. It may be useful to put a title or other information in that header.

The user must next define the objective function. This is a FORTRAN function not a FORTRAN subroutine. This function simply has the vector $X$ as the input argument and returns the function value through the function name, $F$.

The gradient of the objective function is defined as a FORTRAN subroutine named $GRADF(X,GRDF)$. It has the vector $X$ as input and the vector $\text{Grad } F(X) = GRDF$ as the output. The gradients of the objective function with respect to the slack variables are all zero. Since they are constant throughout the solution, they defined with the GRGOPT32 program.

The equality constraints are defined with a FORTRAN subroutine named $HCON(X,H)$. This subroutine takes vector $X$ as the input and returns vector $H$, the values of the equality constraints evaluated at $X$.

The gradients of the equality constraints ($\text{Grad } H$), are defined in a FORTRAN subroutine named $GRADH(X,GRDH)$. 
Subroutine GRADH takes vector X as input and returns the first K rows of matrix \( C = [\text{GRDH}, \text{GRDG}] \). The components of this matrix are defined as follows

\[
[C_{k,n}] = [\text{GRDH}_{k,n}] = d(H_k(X))/dX_n \quad \text{for } k = 1, \ldots, K
\]
\[
\text{for } n = 1, \ldots, N
\]

Again, since the gradients of the equality constraints with respect to the slack variables are all zero, and they are constant, they are defined within the GRGOPT32 program.

The inequality constraints are defined with a FORTRAN subroutine named GCON(X,G). This subroutine takes vector X as the input and returns vector G, the values of the inequality constraints evaluated at X. The slack variables are dealt with automatically by the loop at the bottom of this subroutine. See Template 1.

The gradients of the inequality constraints (Grad G), are defined in a FORTRAN subroutine named GRADG(X,GRDG). Subroutine GRADG takes vector X as input and returns the bottom J rows of matrix \( C = [\text{GRDH}, \text{GRDG}] \). The components of this matrix are defined as follows

\[
[C_{j,n}] = [\text{GRDG}_{j,n}] = d(G_j(X))/dX_n \quad \text{for } j = 1, \ldots, J
\]
\[
\text{for } n = 1, \ldots, N
\]

In the case of the inequality constraints, the gradients with respect to the slack variables are not all zero, but since they are constant, they are defined within the GRGOPT32 program.
B.2 Output Format & Error Messages

When all the variables, functions and subroutines are defined, the file is ready to be compiled and linked with GRGOPT32. The DOS commands required for the Microsoft FORTRAN (Version 3) compiler/linker are:

```
FOR1 filename
PAS2 filename
LINK filename GRGOPT32
```

If all goes well, a lot of data scrolls by on the monitor and an optimum solution is found. A copy of all the data that scrolls by on the monitor is dumped to a file called GRG.FOR. This file name can be changed easily by altering the FILE='GRG.OUT' option in the OPEN statement or by using the DOS copy command after the program is executed. The user can then examine the output file to check the entire optimization process iteration by iteration because GRGOPT32 dumps a status report after every iteration. The output file will contain the heading selected by the user, followed by the current tolerances, the initial feasible vector and the objective function at that point. What follows is the information that is reported after each iteration. This data appears in the format listed in Table B-1.
ITERATION NUMBER 1

FINAL STEP SIZE = 5.0000000000000E-001 (\(\alpha\))

OBJECTIVE FUNCTION AT \(X\) 5.0000000000000 \(F(X^{(t)})\)

OBJECTIVE FUNCTION AT \(V_1\) 4.99999999998790 \(F(V)\)

ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: 8

<table>
<thead>
<tr>
<th>I</th>
<th>IPARTN</th>
<th>X</th>
<th>D</th>
<th>V1</th>
<th>GRADF</th>
<th>REDGRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2.000E+01</td>
<td>2.000E+01</td>
<td>3.000E+01</td>
<td>4.000E+01</td>
<td>-2.000E+01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4.000E+01</td>
<td>-1.000E+01</td>
<td>3.316E+01</td>
<td>-8.000E+01</td>
<td>0.000E+00</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5.000E+01</td>
<td>-2.000E+01</td>
<td>4.000E+01</td>
<td>1.000E+02</td>
<td>0.000E+00</td>
</tr>
</tbody>
</table>

Table B-1 Data Format for Intermediate Status Output.

The information is useful for detecting a trend in the convergence or errors in the problem formulation. The status report starts off with the iteration number and the step size that was used to produce \(V_1\) from \(X^{(t)}\). The objective function values are listed next. The difference between the two indicates the success of this particular iteration. The accumulative number of matrix inverses is useful as a measure of the amount of work required by the algorithm to churn out the solution. The column of integers on the extreme left are the indices of the \(N+J\) variables. The second column represents the partition of the basic and nonbasic variables. The first \(K+J\) rows of this column are the current basis definition for this iteration. The third through sixth columns are the vector \(X^{(t)}\), the current search vector \(d\), the point that resulted from the Newton's iterations \(V\) and the vector Grad \(F\), respectively. The seventh column is the reduced gradient Grad \(F'\). Recall that it is a vector in the reduced space of the nonbasic variables, therefore its length
If an optimum solution was found on a particular iteration, the contents of Table B-1 would be followed by a message that either the reduced gradient (Grad F') or the nonbasic search vector (\( \mathbf{d} \)) has magnitude less than or equal to TOL1. The logical variable OKAY would be returned true and the vector X is the optimum solution. When OKAY is true, a final summary status report is printed in the format demonstrated in Table B-2.

---

AN OPTIMUM SOLUTION HAS BEEN FOUND: SYSTEM LEVEL

WITHIN TOL3

<table>
<thead>
<tr>
<th>I</th>
<th>LOWER BOUND</th>
<th>X</th>
<th>UPPER BOUND</th>
<th>OF BOUNDS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.0000000E+00</td>
<td>.2500000E+01</td>
<td>.1000000E+02</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>.0000000E+00</td>
<td>.3708099E+01</td>
<td>.1000000E+02</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>.0000000E+00</td>
<td>.4500000E+01</td>
<td>.1000000E+02</td>
<td>F</td>
</tr>
</tbody>
</table>

THE MINIMUM OBJECTIVE FUNCTION VALUE AT X: 4.499999999632

Table B-2 Data Format for Summary Status Report.

The first column contains the index of the N+J design variables. The second and fourth columns are the lower and upper bounds, respectively. The third column is the optimum solution. The fifth column contains a logical variable that indicates whether a particular design variable is within TOL3 of its bounds. The last entry in Table B-2 is the objective function value at the optimum solution.
It should be noted that the logical variable PRINT is used to control the amount of data GRGOPT32 returns while it is solving a problem. When PRINT is set true, all of the status data listed in Table B-1, plus any error messages will be printed to the screen and to the output file. If PRINT is set false, the only information returned will be the final status report listed in Table B-2. GRGOPT32 runs much quicker when PRINT is set false, because so much effort is required to write to the screen and to file. Therefore, if the speed of the algorithm is the main concern, set PRINT to false.

If the optimum solution was not found, an error message would appear and logical variable OKAY is returned false and vector X is the last approximation of the optimum solution. In the current version of GRGOPT, when this occurs, a message stating that GRGOPT failed to find the optimum solution is printed. There are six possible reasons that OKAY was returned false. The error messages are explained below along with action that can be taken to correct these errors.

1) MX < N+J Problem is larger than default dimensions.

This error appears when the default vector/array dimension parameter, MX is too small for the problem. This error can be corrected by increasing parameter MX in GRGOPT32.FOR and in all sub-programs to some value greater than N+J. Note, parameter MX must be set to the same value in all sub-programs or data overwrite will result.
2) Unable to find a linearly independent basis.

This error results from GRGOPT32 being unable to find K+J linearly independent rows of the CON (GRDCON) matrix. The user should first check the constraint subroutines for typographical errors. If this fails to produce the culprit, two constraints are redundant and one needs to be removed.

3) No progress in this cycle, basis permutation exhausted.

This error message will appear when the Newton's iterations fail to find a feasible point that offers objective function improvement after all of the basis permutations have been tried. This point can be thought of as the end of a dead end search path for this particular set of tolerances. Though the Kuhn-Tucker conditions do not hold at this point, it could be very near to an optimum solution, but the tolerances are too small. If the last point in the solution sequence is undesirable, the user could try a new starting point with bigger initial tolerances. It is common for this error to occur towards the tail end of the repeating cycles when the tolerances are very very small and an optimum solution was found on a preceding cycle. However, since the Generalized Reduced Gradient algorithm uses only feasible points throughout its solution sequence, this point is feasible and may be an improvement over the starting point. When a practical optimum solution is sought, this is a good candidate.
4) No progress this cycle, algorithm converged to same point twice.

This message occurs when the point that the Newton's iterations selected \( V \) is within TOL3 of the current value of \( x^{(t)} \). This occurs when TOL3 is set much too large or when the point \( x^{(t)} \) is located near to the bounds and only a very small step size \( \alpha \) is used. Though this step caused the objective function to improve, the two points are too close together to be called unique. This can be prevented sometimes by decreasing the value of NEWTIT. This message is similar to the last in that this point is a dead end. The search has flattened out against the constraints and/or the variable bounds. If the current solution is undesirable, probable solutions are decreasing the initial tolerances or increasing the rate at which they are decreased by increasing FACTOR. A different initial starting point may help also.

5) Maximum number of iterations exceeded.

This message is delivered when the iteration limit MAXIT is reached. It can be avoided by increasing MAXIT.

6) None of the nonbasic variables are eligible to enter basis.

This error occurs when logical variable SYS has been set false and all of the nonbasic variables are with TOL3 of their respective bounds. When SYS is set false, these nonbasic variables are prevented from entering the basis to ensure the
basic variables maximum latitude for change. This situation is more likely to occur with small problems with very few nonbasic variables. If this message is issued, the user is recommended to set parameter SYS to true, even though this is not a system level decomposition problem.

With the FORTRAN problem format outlined in this section, combined with the problem template from section B.5, the user has all the information to use GRGOPT32 to perform an optimization study. The next section describes the problems used to test and demonstrate GRGOPT32.
B.3 Example Problems

In this section three example optimization problems are presented to demonstrate the problem formulation procedure and to verify the performance of the GRGOPT32 program. These problems were selected because the optimum solutions are known and they illustrate some key features of problem formulation. Next, a practical application of the GRGOPT32 program is presented. The two-bar truss structure example from Chapter 2 is modified to include an additional design variable. This problem was selected because it has an organizational structure that lends itself to a decomposition solution. This problem is solved as two-level decomposed optimization problem later in Appendix C. Finally, each problem's key definition parameters are included (section B.4) to be used as guidelines. These key parameters include the problems' dimensions, tolerances, variable bounds, initial starting points, objective function and gradient, constraints and constraint gradients.

Problem Test 1

This problem is used by Reklaitis et. al. (1883) to demonstrate the GRG algorithm. It was selected because not only is the optimum solution given but intermediate calculations are provided for which to test the code as it was developed. The variable bounds, originally absent, were
added such that they would not interfere in the optimum solution.

Minimize: \( F(X) = 4X_1 - X_2^2 + X_3^2 - 12 \)

Subject to: \( H_1(X) = 20 - X_1^2 - X_2^2 = 0 \)
\( H_2(X) = X_1 + X_3 - 7 = 0 \)
\( 0 \leq X_1, X_2, X_3 \leq 10 \)

Initial feasible design variables: \( X^{(0)} = (2, 4, 5)^T \)
\( F(X^{(0)}) = 5 \)

Optimum solution: \( X^* = (2.5, 3.71, 4.5)^T \)
\( F(X^*) = 4.5 \)

For illustrative purposes, the solution sequence is presented in Table B-3. Notice that the reduced gradient has zero magnitude at the optimum solution. The known optimum solution was determined in only six iterations. The problem's key definition parameters are listed in section B.4 of this appendix.
Table B-3 Solution Sequence for Problem Test 1. (Continued)
ITERATION NUMBER

FINAL STEP SIZE = \(5.0000000000000E-001\)

OBJECTIVE FUNCTION AT X: \(4.63284868740360\)

OBJECTIVE FUNCTION AT V1: \(4.63284866182200\)

ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: 58

<table>
<thead>
<tr>
<th>I</th>
<th>IPARTN</th>
<th>X</th>
<th>D</th>
<th>V1</th>
<th>GRADF</th>
<th>REDGRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.2757E+01</td>
<td>-.1030E+01</td>
<td>.2242E+01</td>
<td>.4000E+01</td>
<td>-.1030E+01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>.3520E+01</td>
<td>.8075E+00</td>
<td>.3869E+01</td>
<td>-.7041E+01</td>
<td>.0000E+00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>.4242E+01</td>
<td>.1030E+01</td>
<td>.4757E+01</td>
<td>.8484E+01</td>
<td>.0000E+00</td>
</tr>
</tbody>
</table>

ITERATION NUMBER

FINAL STEP SIZE = \(2.5000000000000E-001\)

OBJECTIVE FUNCTION AT X: \(4.63284866182200\)

OBJECTIVE FUNCTION AT V1: \(4.49999999963220\)

ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: 66

<table>
<thead>
<tr>
<th>I</th>
<th>IPARTN</th>
<th>X</th>
<th>D</th>
<th>V1</th>
<th>GRADF</th>
<th>REDGRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>.2242E+01</td>
<td>.1030E+01</td>
<td>.2500E+01</td>
<td>.4000E+01</td>
<td>-.1030E+01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>.3869E+01</td>
<td>-.5974E+00</td>
<td>.3708E+01</td>
<td>-.7738E+01</td>
<td>.0000E+00</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>.4757E+01</td>
<td>-.1030E+01</td>
<td>.4500E+01</td>
<td>.9515E+01</td>
<td>.0000E+00</td>
</tr>
</tbody>
</table>

ITERATION NUMBER

OBJECTIVE FUNCTION AT X: \(4.49999999963220\)

ACCUMULATIVE NUMBER OF J INVERSES REQUIRED: 67

<table>
<thead>
<tr>
<th>I</th>
<th>IPARTN</th>
<th>X</th>
<th>GRDF</th>
<th>REDGRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.2500E+01</td>
<td>.4000E+01</td>
<td>.0000E+00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>.3708E+01</td>
<td>-.7416E+01</td>
<td>.0000E+00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>.4500E+01</td>
<td>.9000E+01</td>
<td>.0000E+00</td>
</tr>
</tbody>
</table>

AN OPTIMUM SOLUTION HAS BEEN FOUND. THE REDUCED GRADIENT HAS MAGNITUDE LESS THAN OR EQUAL TO TOL1.

AN OPTIMUM SOLUTION HAS BEEN FOUND: SYSTEM LEVEL WITHIN TOL3

<table>
<thead>
<tr>
<th>I</th>
<th>LOWER BOUND</th>
<th>X</th>
<th>UPPER BOUND</th>
<th>OF BOUNDS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.00000000E+00</td>
<td>.25000000E+01</td>
<td>.10000000E+02</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>.00000000E+00</td>
<td>.3708099E+01</td>
<td>.10000000E+02</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>.00000000E+00</td>
<td>.45000000E+01</td>
<td>.10000000E+02</td>
<td>F</td>
</tr>
</tbody>
</table>

THE MINIMUM OBJECTIVE FUNCTION VALUE AT X: \(4.49999999963220\)

Table B-3 Solution Sequence for Problem Test1. (Continued)
Problem Test2

This linear problem was selected to test the ability of GRGOPT32 to handle inequality constraints and active variable bounds. It was originally a maximization problem (Reklaitis et. al., 1983) but was converted to a minimization problem by multiplying the objective function by negative one. To demonstrate the conversion of inequality constraints to equality constraints with the introduction of slack variables, the original problem is followed by its equivalent expressed in the standard problem format.

Original Problem:
Maximize: \( F(X) = 10X_1 + 6X_2 + 4X_3 \)
Subject to: 
\[ G_1(X) = X_1 + X_2 + X_3 \leq 100 \]
\[ G_2(X) = 10X_1 + 4X_2 + 5X_3 \leq 600 \]
\[ G_3(X) = 2X_1 + 2X_2 + 6X_3 \leq 300 \]
\( 0 \leq X_1, X_2, X_3 \leq 200 \)

Standard Problem Format:
Minimize: \( F(X) = -10X_1 - 6X_2 - 4X_3 \)
Subject to: 
\[ G_1(X) = X_1 + X_2 + X_3 - X_4 = 0 \]
\[ G_2(X) = 10X_1 + 4X_2 + 5X_3 - X_5 = 0 \]
\[ G_3(X) = 2X_1 + 2X_2 + 6X_3 - X_6 = 0 \]
\( 0 \leq X_1, X_2, X_3 \leq 200 \)
\( 0 \leq X_4 \leq 100 \quad 0 \leq X_5 \leq 600 \quad 0 \leq X_6 \leq 300 \)
Initial feasible design variables:
\[
X^{[0]} = (10, 10, 10, 30, 190, 100)^7
\]
\[
F(X^{[0]}) = -200
\]

Optimum solution:
\[
X^* = (33.33, 66.66, 4.631E-6, 100, 600, 200)^7
\]
\[
F(X^*) = -733.333
\]

Note that component \(X_3\) of the optimum solution vector is zero for all practical purposes. The GRGOPT32 routine matched the optimum solution in 2 cycles with 30 iterations on the first and 4 iterations on the second. Since the second cycle improved the optimum objective function value by only 1.6E-5 percent, it can be safely said that the problem was solved in one cycle. The problem's key definition parameters are listed in section B.4.

Problem Test3

Problem Test3 was selected to test the robustness of the GRGOPT32 algorithm. It is a highly nonlinear non-convex problem for which four local minima are known. Seven different starting points were tried, three were selected randomly and the remaining four were the known local minima. For the three randomly selected points and three of the local optima, GRGOPT32 converged to the local minima with the smallest objective function value. For one starting point
however, no progress was made. This demonstrates that the algorithm's success in converging to the optimum solution is related to how close the starting point is to a local minima and how isolated that local optima may be. In general, a large number of starting points, distributed in all portions of the design space, are necessary to confirm that an optimum solution is a global optima. The problem's key definition parameters are listed in section B.4.

Minimize: \( F(X) = (X_1 - 1)^2 + (X_1 - X_2)^2 + (X_2 - X_3)^3 + (X_3 - X_4)^4 \\
\quad + (X_4 - X_5)^4 \)

Subject to: \( H_1(X) = X_1^2 + X_2^2 + X_3^3 - (2 + 3(2)^1) = 0 \)
\( H_2(X) = X_2 - X_3^2 + X_4 + (2 - 2(2)^1) = 0 \)
\( H_3(X) = X_2X_3 - 2 = 0 \)
\(-10 \leq X_i \leq 10 \text{ for } i = 1, \ldots, 5 \)

Initial feasible design variables:
\( X^{(0)} = (-3, -3, 0.6237174219, 4.217450546, -0.66666666)^T \)
\( F(X^{(0)}) = 704.25 \)

Optimum solution:
\( X^* = (1.11663, 1.22044, 1.53779, 1.97277, 1.79110)^T \)
\( F(X^*) = 0.029307 \)

This is the local minima with smallest objective function value.
**Problem Test4**

This is the same problem that was used as a single-level demonstration example in Chapter 2. The problem's key definition parameters are listed in section B.4. The GRGOPT32 program found the optimum solution in two cycles, with 17 iterations in the first cycle and 15 iterations in the second cycle. It should be noted that all of the effort in the second cycle decreased the value of objective function by only 0.024%. Hence, for all practical purposes, the second cycle was unnecessary.

**Problem Test5**

This example is a modified version of problem Test4. The horizontal coordinate of the common joint has been added as an design variable (Figure B-1). The vertical coordinate is referred to as $X_3$, while the horizontal coordinate is $X_4$. The vector of design variables is now defined as follows

\[ X_1 = \text{Area of bar 1} \]
\[ X_2 = \text{Area of bar 2} \]
\[ X_3 = \text{Vertical coordinate of joint C} \]
\[ X_4 = \text{Horizontal coordinate of joint C} \]
\[ X_5 = \text{Slack variable associated with constraint } G_1 \]
\[ X_6 = \text{Slack variable associated with constraint } G_2 \]
The new variable $x_4$ is bounded between support points A and B, i.e. 0 and 5 meters. Expressions for the objective function and the constraints can be derived from the conditions of equilibrium with the same methodology as was used for the single-level example problem in Chapter 2. Problem Test5 expressed in standard problem format is described with equations (B.1) through (B.9).
Minimize: \( F(X) = X_1((X_4)^2 + (X_3)^2)^{\frac{1}{2}} + X_2((5 - X_4)^2 + (X_3)^2)^{\frac{1}{2}} \) \hspace{1cm} B.1

Subject to:

\[ G_1(X) = 20(5 - X_4)[((X_4)^2 + (X_3)^2)^{\frac{1}{2}} / (X_3 X_1)] - X_5 = 0 \hspace{1cm} B.2 \]
\[ G_2(X) = 20 X_4 [((5 - X_4)^2 + (X_3)^2)^{\frac{1}{2}} / (X_3 X_2)] - X_6 = 0 \hspace{1cm} B.3 \]
\[ 0 \leq X_1 \leq 0.003 \text{ m}^2 \hspace{1cm} B.4 \]
\[ 0 \leq X_2 \leq 0.003 \text{ m}^2 \hspace{1cm} B.5 \]
\[ 1 \leq X_3 \leq 3 \text{ m} \hspace{1cm} B.6 \]
\[ 0 \leq X_4 \leq 5 \text{ m} \hspace{1cm} B.7 \]
\[ 0 \leq X_5 \leq 100000 \text{ kN/m}^2 \hspace{1cm} B.8 \]
\[ 0 \leq X_6 \leq 100000 \text{ kN/m}^2 \hspace{1cm} B.9 \]

The same initial feasible \( X \) vector as before can be used giving the same initial objective function value.

\[ X^{(0)} = (0.001, 0.001, 3.0, 4.0, 33333.33, 84327.4)^T \]
\[ F(X^{(0)}) = 0.0081623 \text{ m}^3 \]

The problem's key definition parameters are listed in section B.4. GRGOPT32 found the optimum solution in two cycles, with 25 iterations in the first cycle and 9 iterations in the second cycle. The second cycle improved the optimum objective function value by 0.10%. The optimum solution and the objective function at that point are listed below.

\[ X^{(f)} = (6.88E-10, 0.001, 1.0, 5.0, 100000, 100000)^T \]
\[ F(X^{(f)}) = 0.00100 \text{ m}^3 \]
This solution is interesting in that five of the six variables are at or very near to their bounds. This solution corresponds to the physical situation where bar 2 is aligned vertically with the applied load and therefore carries all the load. As a result, the cross sectional area of bar 1 is nearly zero. Joint C is located in the upper right corner of its spatial envelope, directly below point B. This location minimizes the length of bar 2. This is actually not a truss anymore, but a mechanism of sorts, with a single member aligned with the applied load. Even a small side load resulting from the misalignment of the applied load, would cause large deflections as bar 2 freely rotates about point B. If this situation is undesirable, perhaps adding a constraint that limits side deflection would prevent this occurrence. The symmetric solution is obtained when joint C is located in the upper left corner of its spatial envelope, directly below point A and bar 1 carries all of the load. Convergence to either extreme corner is governed by the initial starting point selected. In any case, this example shows that the analyst must use common sense and sound engineering judgement when setting up the problem and interpreting the results of the optimization study.
B.4 Key Definition Information for Single-Level Problems

Problem Test1

C PROBLEM VARIABLES:
N=3
K=2
J=0
NPJ=N+J
KPJ=K+J
NMK=N-K
ALPHA0=1.0
GAMMA=0.5
TOL1=1.0D-4
TOL2(1)=1.0D-5
TOL2(2)=1.0D-5
TOL3(1)=1.0D-6
TOL3(2)=1.0D-6
TOL3(3)=1.0D-6
MAXIT=100
NEWTIT=30
CYCLE=1
FACTOR=100.0
SYS=.FALSE.
PRINT=.TRUE.

C
C INITIAL FEASIBLE POINTS
C
C INITIAL POINT 1
X0(1)=2.0
X0(2)=4.0
X0(3)=5.0

C
C UPPER BOUNDS OF VARIABLES
XUPPER(1)=10.0
XUPPER(2)=10.0
XUPPER(3)=10.0

C
C LOWER BOUNDS OF VARIABLES
XLOWER(1)=0.0
XLOWER(2)=0.0
XLOWER(3)=0.0

C
C DEFINE OBJECTIVE HERE
C
F=4.0*X(1)-(X(2)*X(2))+(X(3)*X(3))-12.0

C
C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C
GRDF(1)=4.0
GRDF(2) = -2.0*X(2)
GRDF(3) = 2.0*X(3)

C DEFINE EQUALITY CONSTRAINTS HERE
C
H(1) = 20.0 - (X(1)*X(1)) - (X(2)*X(2))
H(2) = X(1) + X(3) - 7.0

C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE
C
GRDH(1,1) = -2.0*X(1)
GRDH(1,2) = -2.0*X(2)
GRDH(1,3) = 0.0
GRDH(2,1) = 1.0
GRDH(2,2) = 0.0
GRDH(2,3) = 1.0

C DEFINE INEQUALITY CONSTRAINTS HERE
C
C THIS PROBLEM HAS NONE
C
C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
C THIS PROBLEM HAS NONE
Problem Test2

C PROBLEM VARIABLES:
N = 3
K = 0
J = 3
NPJ = N + J
KPJ = K + J
NMK = N - K
ALPHA0 = 1.0
GAMMA = 0.5
TOL1 = 1.0D-2
TOL2(1) = 1.0D-3
TOL2(2) = 1.0D-3
TOL2(3) = 1.0D-3
TOL3(1) = 1.0D-4
TOL3(2) = 1.0D-4
TOL3(3) = 1.0D-4
TOL3(4) = 1.0D-4
TOL3(5) = 1.0D-4
TOL3(6) = 1.0D-4
MAXIT = 100
NEWTIT = 30
CYCLE = 2
FACTOR = 100.0
SYS = .FALSE.
PRINT = .TRUE.

C
C INITIAL FEASIBLE POINTS
C
C INITIAL POINT 1
X0(1) = 10.
X0(2) = 10.
X0(3) = 10.
X0(4) = 30.
X0(5) = 190.
X0(6) = 100

C
C UPPER BOUNDS OF VARIABLES
XUPPER(1) = 200.0
XUPPER(2) = 200.0
XUPPER(3) = 200.0
XUPPER(4) = 100.0
XUPPER(5) = 600.0
XUPPER(6) = 300.0

C
C LOWER BOUNDS OF VARIABLES
XLOWER(1) = 0.0
XLOWER(2) = 0.0
XLOWER(3) = 0.0
XLOWER(4) = 0.0
XLOWER(5) = 0.0
XLOWER(6)=0.0

C DEFINE OBJECTIVE FUNCTION HERE
C F=-1.0*(10.0*X(1)+6.0*X(2)+4.0*X(3))
C
C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C GRDF(1)=-10.0
GRDF(2)=-6.0
GRDF(3)=-4.0

C DEFINE EQUALITY CONSTRAINTS HERE
C THIS PROBLEM HAS NONE
C
C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE
C THIS PROBLEM HAS NONE
C
C DEFINE INEQUALITY CONSTRAINTS HERE
C G(1)=X(1)+X(2)+X(3)
G(2)=10.0*X(1)+4.0*X(2)+5.0*X(3)
G(3)=2.0*X(1)+2.0*X(2)+6.0*X(3)

C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
GRDG(1,1)=1.0
GRDG(1,2)=1.0
GRDG(1,3)=1.0
C
GRDG(2,1)=10.0
GRDG(2,2)=4.0
GRDG(2,3)=5.0
C
GRDG(3,1)=2.0
GRDG(3,2)=2.0
GRDG(3,3)=6.0
Problem Test3

C PROBLEM VARIABLES:
N=5
K=3
J=0
NPJ=N+J
KPJ=K+J
NMK=N-K
ALPHA0=1.0
GAMMA=0.5
TOL1=1.0D-3
TOL2(1)=1.0D-4
TOL2(2)=1.0D-4
TOL2(3)=1.0D-4
TOL3(1)=1.0D-5
TOL3(2)=1.0D-5
TOL3(3)=1.0D-5
TOL3(4)=1.0D-5
TOL3(5)=1.0E-5
MAXIT=100
NEWTIT=30
CYCLE=2
FACTOR=100.0
SYS=.FALSE.
PRINT=.TRUE.

C INITIAL FEASIBLE POINTS
C
C POINT 1 (CONVERGES TO X-STAR-1) F0 = 4.198
C
X0(1)=1.0
X0(2)=2.0597671
X0(3)=1.0
X0(4)=0.8284271
X0(5)=2.0

C POINT 2 (CONVERGES TO X-STAR-1) F0 = 3.341
C
X0(1)=2.0
X0(2)=1.8007333
X0(3)=1.0
X0(4)=0.027693824
X0(5)=1.0

C POINT 3 (CONVERGES TO X-STAR-1) F0 = 704.25
X0(1)=-3.0
X0(2)=-3.0
X0(3)=0.6237174219
X0(4)=4.217450546
X0(5)=-2.0/3.0

C
C POINT 4, X-STAR-1 (CONVERGES TO X-STAR-1) F0 = 0.0293109
C
X0(1)=1.11663
C
X0(2)=1.22044
X0(3)=1.53779
X0(4)=1.97277
X0(5)=1.79110
C
C POINT 5, X-STAR-2 (CONVERGES TO X-STAR-1) F0 = 607.034
C
X0(1)=-2.79087
X0(2)=-3.00414
X0(3)=0.205371
X0(4)=3.87474
X0(5)=-0.716623
C
C POINT 6, X-STAR-3 (CONVERGES TO X-STAR-3) F0 = 27.87
C
X0(1)=-1.27305
X0(2)=2.41035
X0(3)=1.19486
X0(4)=-0.154239
X0(5)=-1.57103
C
C POINT 7, X-STAR-4 (CONVERGES TO X-STAR-4, ALPHAO=10^-6) F0 = 44.021
C
X0(1)=-0.703393
X0(2)=2.63570
X0(3)=-0.0963618
X0(4)=-1.79799
X0(5)=-2.84336
C
C UPPER BOUNDS OF VARIABLES
XUPPER(1)=10.0
XUPPER(2)=10.0
XUPPER(3)=10.0
XUPPER(4)=10.0
XUPPER(5)=10.0
C
C LOWER BOUNDS OF VARIABLES
XLOWER(1)=-10.0
XLOWER(2)=-10.0
XLOWER(3)=-10.0
XLOWER(4)=-10.0
XLOWER(5)=-10.0
C
C DEFINE OBJECTIVE HERE
C
F=((X(1)-1.0)**2)+((X(1)-X(2))**2)+((X(2)-X(3))**3)+
&((X(3)-X(4))**4)+((X(4)-X(5))**4)
C
C DEFINE OBJECTIVE FUNCTION GRADIENTS HERE
C
GRDF(1)=2.0*(X(1)-1.0)+2.0*(X(1)-X(2))
GRDF(2)=-2.0*(X(1)-X(2))+3.0*((X(2)-X(3))**2)
GRDF(3)=-3.0*((X(2)-X(3))**2)+4.0*((X(3)-X(4))**3)
GRDF(4)=-4.0*((X(3)-X(4))**3)+4.0*((X(4)-X(5))**3)
GRDF(5) = -4.0*((X(4)-X(5))*3)

C DEFINE EQUALITY CONSTRAINTS HERE

C  H(1) = X(1) + (X(2)*X(2)) + (X(3)*3) - 2.0 - (3.0*DSQRT(2))
C  H(2) = X(2) - (X(3)*X(3)) + X(4) + 2.0 - (2.0*DSQRT(2))
C  H(3) = X(1)*X(5) - 2.0

C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE

C  GRDH(1,1) = 1.0
C  GRDH(1,2) = 2.0*X(2)
C  GRDH(1,3) = 3.0*X(3)*X(3)
C  GRDH(1,4) = 0.0
C  GRDH(1,5) = 0.0

C  GRDH(2,1) = 0.0
C  GRDH(2,2) = 1.0
C  GRDH(2,3) = -2.0*X(3)
C  GRDH(2,4) = 1.0
C  GRDH(2,5) = 0.0

C  GRDH(3,1) = X(5)
C  GRDH(3,2) = 0.0
C  GRDH(3,3) = 0.0
C  GRDH(3,4) = 0.0
C  GRDH(3,5) = X(1)

C DEFINE INEQUALITY CONSTRAINTS HERE

C  THIS PROBLEM HAS NO INEQUALITY CONSTRAINTS

C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE

C  THIS PROBLEM HAS NO INEQUALITY CONSTRAINTS
C PROBLEM VARIABLES:
N=3
K=0
J=2
NPJ=N+J
KPJ=K+J
NMK=N-K
ALPHA0=10.0
GAMMA=0.5
TOL1=1.0D-4
TOL2(1)=100.0
TOL2(2)=100.0
TOL3(1)=1.0D-8
TOL3(2)=1.0D-8
TOL3(3)=1.0D-5
TOL3(4)=10.0
TOL3(5)=10.0
MAXIT=100
NEWTIT=30
CYCLE=2
FACTOR=100.0
SYS=.FALSE.
PRINT=.TRUE.

C INITIAL FEASIBLE POINTS
C
C POINT 1
   X0(1)=0.001
   X0(2)=0.001
   X0(3)=3.0
   X0(4)=33333.333
   X0(5)=84327.404

C UPPER BOUNDS OF VARIABLES
   XUPPER(1)=0.003
   XUPPER(2)=0.003
   XUPPER(3)=3.0
   XUPPER(4)=100000.0
   XUPPER(5)=100000.0

C LOWER BOUNDS OF VARIABLES
   XLOWER(1)=0.0
   XLOWER(2)=0.0
   XLOWER(3)=1.0
   XLOWER(4)=0.0
   XLOWER(5)=0.0

C DEFINE OBJECTIVE FUNCTION HERE
F=(X(1)*DSQRT(16.0+(X(3)*X(3))))+(X(2)*DSQRT(1.0+}
&((X(3)*X(3))
C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C
GRDF(1)=DSQRT(16.0+(X(3)*X(3)))
GRDF(2)=DSQRT(1.0+(X(3)*X(3)))

GRDF(3)=((X(1)*X(3))/(DSQRT(16.0+(X(3)*X(3))))+(X(2)*X(3)/
 & (DSQRT(1.0+(X(3)*X(3))))
C DEFINE EQUALITY CONSTRAINTS HERE
C
THIS PROBLEM HAS NONE
C
C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE
C
THIS PROBLEM HAS NONE
C
C DEFINE INEQUALITY CONSTRAINTS HERE
C
G(1)=20.0*(DSQRT(16.0+(X(3)**2)))/(X(1)*X(3))
G(2)=80.0*(DSQRT(1.0+(X(3)**2)))/(X(2)*X(3))
C
C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
GRDG(1,1)=-20.0*(DSQRT(16.0+(X(3)**2)))/
 & (X(1)*X(1)*X(3))
GRDG(1,2)=0.0
GRDG(1,3)=(20.0/X(1))*((1.0/DSQRT(16.0+(X(3)**2)))-
 & (DSQRT(16.0+(X(3)**2)))/(X(3)**2))
C
GRDG(2,1)=0.0
GRDG(2,2)=-80.0*(DSQRT(1.0+(X(3)**2)))/
 & (X(2)*X(2)*X(3))
GRDG(2,3)=(80.0/X(2))*((1.0/DSQRT(1.0+(X(3)**2)))-
 & (DSQRT(1.0+(X(3)**2)))/(X(3)**2))
Problem Test5

C PROBLEM VARIABLES:
N=4
K=0
J=2
NPJ=N+J
KPJ=K+J
NMK=N-K
ALPHA0=1.0D+6
GAMMA=0.5
TOL1=1.0D-4
TOL2(1)=100.0
TOL2(2)=100.0
TOL3(1)=1.0D-8
TOL3(2)=1.0D-8
TOL3(3)=1.0D-5
TOL3(4)=1.0D-5
TOL3(5)=10.0
TOL3(6)=10.0
MAXIT=100
NEWTIT=30
CYCLE=2
FACTOR=100.0
SYS=.FALSE.
PRINT=.TRUE.

C
C INITIAL FEASIBLE POINTS
C
C POINT 1
X0(1)=0.001
X0(2)=0.001
X0(3)=3.0
X0(4)=4.0
X0(5)=33333.333
X0(6)=84327.404

C
C POINT 2 CENTER POINT (SYMMETRIC SOLUTION)
C
X0(1)=707.106781D-6
X0(2)=707.106781D-6
X0(3)=2.5
X0(4)=2.5
X0(5)=1.0D+5
X0(6)=1.0D+5

C
C POINT 3 REFLECTION OF FIRST X0
C
X0(1)=0.001
X0(2)=0.001
X0(3)=3.0
X0(4)=1.0
X0(6)=33333.333
X0(5)=84327.404
C C UPPER BOUNDS OF VARIABLES
  XUPPER(1)=0.003
  XUPPER(2)=0.003
  XUPPER(3)=3.0
  XUPPER(4)=5.0
  XUPPER(5)=100000.0
  XUPPER(6)=100000.0
C
C C LOWER BOUNDS OF VARIABLES
  XLOWER(1)=0.0
  XLOWER(2)=0.0
  XLOWER(3)=1.0
  XLOWER(4)=0.0
  XLOWER(5)=0.0
  XLOWER(6)=0.0
C
C C DEFINE OBJECTIVE FUNCTION HERE
C
  F=(X(1)*DSQRT(X(3)**2+X(4)**2)) +
     &((X(2)*DSQRT((5-X(4))**2+X(3)**2))
C
C C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C
  GRDF(1)=DSQRT(X(4)**2+X(3)**2)
  GRDF(2)=DSQRT((5-X(4))**2+X(3)**2)
  GRDF(3)=(X(1)*X(3))/((DSQRT(X(4)**2+X(3)**2)) +
           &((X(2)*X(3))/((DSQRT((5-X(4))**2+X(3)**2)) +
            &((X(2)*(5.0-X(4))/((DSQRT((5-X(4))**2+X(3)**2))))
C
C C DEFINE EQUALITY CONSTRAINTS HERE
C
  C THIS PROBLEM HAS NONE
C
C C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE
C
  C THIS PROBLEM HAS NONE
C
C C DEFINE INEQUALITY CONSTRAINTS HERE
C
  G(1)=20.0*(5.0-X(4))*(DSQRT(X(4)**2+X(3)**2)) /
          &(X(1)*X(3))
  G(2)=20.0*X(4)*(DSQRT((5-X(4))**2+X(3)**2)) /
          &(X(2)*X(3))
C
C C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
  GRDG(1,1)=-20.0*(5.0-X(4))*(DSQRT(X(4)**2+X(3)**2)) /
           &(X(1)*X(1)*X(3))
  GRDG(1,2)=0.0
  GRDG(1,3)=(20.0*(5.0-X(4))/X(1))*
\begin{align*}
&((1.0/\text{DSQRT}(X(4)**2+X(3)**2)) - \\
&\text{DSQRT}(X(4)**2+X(3)**2))/X(3)**2) \\
\text{GRDG}(1, 4) &= 20.0*(5.0-X(4))*X(4)/(X(1)*X(3)* \\
&\text{DSQRT}(X(4)**2+X(3)**2))-20.0*(\text{DSQRT}(X(4)**2+X(3)**2))/ \\
&(X(1)*X(3)) \\
\text{GRDG}(2, 1) &= 0.0 \\
\text{GRDG}(2, 2) &= -20.0*X(4)*(\text{DSQRT}((5-X(4))**2+X(3)**2))/ \\
&(X(2)*X(2)*X(3)) \\
\text{GRDG}(2, 3) &= (20.0*X(4)/X(2))*((1.0/\text{DSQRT}((5*X(4))**2+ \\
&X(3)**2)) - (\text{DSQRT}((5-X(4))**2+X(3)**2))/X3) \\
\text{GRDG}(2, 4) &= -20.0*X(4)*(5.0-X(4))/(X(2)*X(3)* \\
&\text{DSQRT}((5-X(4))**2+X(3)**2))+ \\
&20.0*(\text{DSQRT}((5-X(4))**2+X(3)**2))/(X(2)*X(3)) 
\end{align*}
B.5 Template 1: Single-Level Optimization Problem Outline.

C PROGRAM NAME: TEMPL1.FOR [USED WITH SUBROUTINE GRGOPT32]
C BY: TOM VEILLEUX
C DATE: 10-16-91
C
C ==> TEMPLATE FOR SINGLE LEVEL OPTIMIZATION SOLUTIONS <==
C
C PURPOSE: THIS PROGRAM EMPLOYS THE GENERALIZED REDUCED
C GRADIENT METHOD TO FIND THE OPTIMAL SOLUTION TO A
C CONSTRAINED NONLINEAR OPTIMIZATION PROBLEM. THE
C ALGORITHM CAN HANDLE A NONLINEAR OBJECTIVE
C FUNCTION, NONLINEAR EQUALITY AND INEQUALITY
C CONSTRAINTS AND BOUNDED VARIABLES. INEQUALITY
C CONSTRAINTS ARE 'CONVERTED' TO EQUALITIES WITH
C THE ADDITION OF SLACK VARIABLES. THIS INCREASES
C THE DIMENSION OF THE PROBLEM FROM N TO N+J.
C VARIABLE BOUNDS ARE DEALT WITH IMPLICITLY BY
C VARIOUS CHECKS THROUGHOUT THE ALGORITHM. THE
C DESIGN VARIABLES ARE PARTITIONED INTO K+J BASIC
C (DEPENDENT) VARIABLES AND N-K NONBASIC
C (INDEPENDENT) VARIABLES. BY SOLVING FOR THE
C NON-BASIC VARIABLES IN TERMS OF THE BASIC
C VARIABLES. THIS PARTITION EFFECTIVELY REDUCES THE
C DIMENSION OF THE PROBLEM FROM N+J TO N-K.
C
C REFERENCE: ENGINEERING OPTIMIZATION:
C METHODS AND APPLICATIONS
C G.V. REKLAITIS, A. RAVINDRAN & K.M. RAGSDELL
C JOHN WILEY & SONS, 1983
C PAGES 377 - 402
C
C-----------------------------------------------------------
C STANDARD PROBLEM FORMAT: PROBLEM DIMENSION N
C
C MINIMIZE F(X) FOR X AND N-VECTOR
C
SUBJECT TO Hk(X) = 0 FOR k = 1, 2, .... K
AJ < Gj(X) < Bj FOR j = 1, 2, .... J
XLOWERI < Xi < XUPPERI FOR i = 1, 2, .... N
C-----------------------------------------------------------
C CONVERTED PROBLEM FORMAT: PROBLEM DIMENSION N + J
C
C MINIMIZE F(X) FOR X AN (N+J)-VECTOR
C
SUBJECT TO Hk(X) = 0 FOR k = 1, 2, .... K
Gj(X) - XN+j = 0 FOR j = 1, 2, .... J
XLOWERI < Xi < XUPPERI FOR i = 1, 2, .... N
Aj < XN+j < Bj FOR j = 1, 2, ..., J

USAGE:

CALL GRGOPT32(N,K,J,ALPHE0,GAMMA,TOL1,TOL2,TOL3,X0,
&XUPPER,XLOWER,MAXIT,NEWTIT,F,GRADF,HCON,GRADH,GCON,
&GRADG,PRINT,SYS,X,OKAY)

THE USER MUST PROVIDE THE FOLLOWING INPUT VARIABLES,
FUNCTIONS & SUBROUTINES.

=> USER DEFINED INPUT ARGUMENTS:
N: NUMBER OF INDEPENDENT VARIABLES
K: NUMBER OF EQUALITY CONSTRAINTS
J: NUMBER OF INEQUALITY CONSTRAINTS
ALPHA0: INITIAL STEP SIZE
GAMMA: STEP SIZE REDUCTION PARAMETER (USE 0.5)
TOL1: CONVERGENCE TOLERANCE FOR REDUCED GRADIENT
TOL2: TOLERANCE FOR FEASIBILITY OF CONSTRAINTS
TOL3: CONVERGENCE TOLERANCE FOR NEWTON ITERATIONS
X0: INITIAL FEASIBLE POINT
XUPPER: VECTOR OF UPPER BOUNDS OF X
XLOWER: VECTOR OF LOWER BOUNDS OF X
MAXIT: MAX NUMBER OF ITERATIONS, A TERMINATION LIMIT
NEWTIT: MAX NUMBER OF NEWTON ITERATIONS: USE 20
PRINT: LOGICAL VARIABLE THAT CONTROLS THE DATA OUTPUT
TRUE = YES
FALSE = NO INTERMEDIATE PRINT-OUT
SYS: LOGICAL VARIABLE USED FOR IDENTIFYING SYSTEM
LEVEL PROBLEMS IN DECOMPOSITION
TRUE: FOR SYSTEM LEVEL SUB-PROBLEMS
IN DECOMP ONLY
FALSE: FOR ALL OTHER

=> USER DEFINED FUNCTIONS: (MUST BE DECLARED EXTERNAL)
F(X): OBJECTIVE FUNCTION

=> USER DEFINED SUBROUTINES: MUST BE DECLARED EXTERNAL
GRADF(X,GRDF): GRADIENT OF OBJECTIVE FUNCTION
HCON(X,H): EQUALITY CONSTRAINTS
GRADH(X,GRDH): GRADIENT OF EQUALITY CONSTRAINTS
GCON(X,G): INEQUALITY CONSTRAINTS
GRADG(X,GRDG) GRADIENT OF INEQUALITY CONSTRAINTS

NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY

=> ARGUMENTS: (USED IN SUB PROGRAMS)

X: INDEPENDENT VARIABLES VECTOR, LENGTH N+J

GRDF: GRADIENT OF THE OBJECTIVE FUNCTION, LENGTH N+J

H: EQUALITY CONSTRAINTS, LENGTH K

GRDH: GRADIENT OF THE EQUALITY CONSTRAINTS, ORDER K BY N+J

G: INEQUALITY CONSTRAINTS, LENGTH J

GRDG: GRADIENT OF THE INEQUALITY CONSTRAINTS, ORDER J BY N+J

NOTE: ALL ARRAYS & VECTORS MUST BE DIMENSIONED TO THE SAME SIZE IN THE MAIN CALLING AND IN EACH SUBROUTINE...USE MX=10

USER DEFINED VARIABLES THAT ARE NOT PASSED TO GRGOPT32

CYCLE: DETERMINES HOW MANY TIMES GRGOPT32 WILL BE CALLED. IN EACH SOLUTION CYCLE, THE OPTIMAL SOLUTION WILL BE SET EQUAL TO THE INITIAL STARTING POINT FOR THE NEXT CYCLE AND GRGOPT32 IS CALLED AGAIN WITH REDUCED TOLERANCES.

FACTOR: THE FACTOR BY WHICH THE TOLERANCES ARE DIVIDED BY IN BETWEEN CYCLES.

VALUES RETURNED BY GRGOPT:

X: OPTIMUM SOLUTION (IF DETERMINED SUCCESSFULLY...)

OKAY: LOGICAL VARIABLE THAT DENOTES SUCCESS OF GRGOPT32 IN FINDING THE OPTIMUM.

OKAY = .TRUE. IF OPTIMUM SOLUTION WAS DETERMINED

OKAY = .FALSE. IF OPTIMUM SOLUTION WAS NOT DETERMINED

BEGIN MAIN PROGRAM

SUBPROGRAMS:

REAL*8 F
EXTERNAL F,GRADF,HCON,GRADH,GCON,GRDG
C INPUT PARAMETERS:
   PARAMETER (MX=10)

C VARIABLES:
   INTEGER MAXIT, NEWTIT, N, K, J, NPJ, KPJ, NMK, LABEL, CYCLE
   REAL*8 X0(MX), XUPPER(MX), XLOWER(MX), X(MX), ALPHA0, GAMMA, TOL1, TOL2(MX), TOL3(MX), FACTOR
   LOGICAL OKAY, BOUNDS(MX), PRINT, SYS

C BEGIN USER DEFINED SECTION

C OUTPUT FILE: A COPY OF ALL DATA PRINTED TO THE SCREEN IS
C ALSO SENT TO THE FILE GRG.OUT

C OPEN (1, FILE='GRG.OUT', STATUS='NEW')

C PROBLEM VARIABLES:
   N=
   K=
   J=
   NPJ=N+J
   KPJ=K+J
   NMK=N-K
   ALPHA0=1.0
   GAMMA=0.5
   TOL1=1.0D-4
   TOL2(1)=
   TOL2(2)=
   TOL3(1)=
   TOL3(2)=
   TOL3(3)=
   TOL3(4)=
   TOL3(5)=
   MAXIT=100
   NEWTIT=30
   CYCLE=
   FACTOR=
   SYS=.FALSE.
   PRINT=.TRUE.

C INITIAL FEASIBLE POINTS

C INITIAL POINT 1
   X0(1)=
   X0(2)=
   X0(3)=
   X0(4)=
   X0(5)=

C INITIAL POINT 2
   X0(1)=
C X0(2)=
C X0(3)=
C X0(4)=
C X0(5)=
C
C UPPER BOUNDS OF VARIABLES
XUPPER(1)=
XUPPER(2)=
XUPPER(3)=
XUPPER(4)=
XUPPER(5)=
C
C LOWER BOUNDS OF VARIABLES
XLOWER(1)=0.0
XLOWER(2)=0.0
XLOWER(3)=0.0
XLOWER(4)=0.0
XLOWER(5)=0.0
C
C WRITE PROBLEM HEADER TO FILE
C
WRITE(1,*)'=========================================',
&'========================================='
WRITE(1,*)' TITLE:
WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'=========================================',
&'========================================='
C
C CALL GRGOPT
C
DO 100,IT=1,CYCLE
C
C WRITE CYCLE NUMBER SUB-HEADING THE SCREEN
C
WRITE(*,*)'=========================================',
&'========================================='
WRITE(*,*)'=========================================',
&'========================================='
WRITE(*,1000)IT
WRITE(*,*)'=========================================',
&'========================================='
C
C WRITE CYCLE NUMBER SUB-HEADING THE FILE
C
WRITE(1,1000)IT
1000 FORMAT(' ============================================================== CYCLE = ',I3,' ============================================================== ',
&' ============================================================== ')
WRITE(1,*)'=========================================', 
&'=================================='
C WRITE TOLERANCES TO FILE
C WRITE(1,1001)TOL1
1001 FORMAT(' TOLERANCE 1 =',5X,E14.7)
WRITE(1,*)'
WRITE(1,1002)(M,TOL2(M),M=1,KPJ)
1002 FORMAT(' TOLERANCE 2 =',15,E14.7)
WRITE(1,*)'
WRITE(1,1003)(M,TOL3(M),M=1,NPJ)
1003 FORMAT(' TOLERANCE 3 =',15,E14.7)
C CALL GRGOPT32(N,K,J,ALPHA0,GAMMA,TOL1,TOL2,TOL3,X0, &XUPPER,XLOWER,MAXIT,NEWIT,F,GRADF,HCON,GRADH,GCON, &GRADG,PRINT,SYS,X,OKAY)
C CHECK FOR EXISTENCE OF SOLUTION
C IF (OKAY)THEN
C OUTPUT RESULTS TO THE SCREEN & TO FILE
C LABEL=0
CALL STATUS(X,XLOWER,XUPPER,TOL3,F,NPJ,LABEL,BOUNDS)
C ELSE
C WRITE TO THE SCREEN
C WRITE(*,*)'+++++++++++++++++++++++++++++++++++++++++++++++++++++', &'++++++++++++++++++++++++++++++++++++++++++++++++++++'
WRITE(*,*)'GRG FAILED TO FIND THE OPTIMUM SOLUTION'
WRITE(*,*)'
WRITE(*,*)'+++++++++++++++++++++++++++++++++++++++++++++++++++++', &'++++++++++++++++++++++++++++++++++++++++++++++++++++'
C WRITE TO FILE
C WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'
WRITE(1,*)'+++++++++++++++++++++++++++++++++++++++++++++++++++++', &'++++++++++++++++++++++++++++++++++++++++++++++++++++'
WRITE(1,*)'
WRITE(1,*)'GRG FAILED TO FIND THE OPTIMAL SOLUTION'
WRITE(1,*)'
WRITE(1,*)'+++++++++++++++++++++++++++++++++++++++++++++++++++++', &'++++++++++++++++++++++++++++++++++++++++++++++++++++'
CENDIF
C
TOLL1 = TOLL1 / FACTOR
C
DO 100, M = 1, NPJ
X0(M) = X(M)
TOL2(M) = TOL2(M) / FACTOR
TOL3(M) = TOL3(M) / FACTOR
100 CONTINUE
STOP
END
C========================================
END OF MAIN PROGRAM=====================
C
C USER DEFINED FUNCTION: OBJECTIVE FUNCTION
C
FUNCTION F(X)
PARAMETER (MX=10)
COMMON N, K, J, NPJ, NMK, KPJ
REAL*8 F, X(MX)
C
C DEFINE OBJECTIVE FUNCTION HERE
C
F =
C
RETURN
END
C========================================
C
C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION
C
SUBROUTINE GRADF(X, GRDF)
PARAMETER (MX=10)
COMMON N, K, J, NPJ, NMK, KPJ
REAL*8 X(MX), GRDF(MX)
C
C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C
GRDF(1) =
GRDF(2) =
GRDF(3) =
C
GRADIENTS OF OBJECTIVE FUNCTION WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN VERSIONS GROPT14+
C
RETURN
END
C========================================
C
C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
C
SUBROUTINE HCON(X, H)
PARAMETER (MX=10)
COMMON N,K,J,MPJ,NMK,PKJ
REAL*8 X(MX),H(MX)

C DEFINE EQUALITY CONSTRAINTS HERE
C
H(1)=
H(2)=
C
RETURN
END

C===========================================================
C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
C
SUBROUTINE GRADH(X,GRDH)
PARAMETER (MX=10)
COMMON N,K,J,MPJ,NMK,PKJ
REAL*8 X(MX),GRDH(MX,MX)

C DEFINE EQUALITY CONSTRAINT GRADIENTS HERE
C
GRDH(1,1)=
GRDH(1,2)=
C
GRDH(2,1)=
GRDH(2,2)=

C GRADIENTS OF EQUALITY CONSTRAINTS WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN VERSIONS GROPT14+
C
RETURN
END

C===========================================================
C USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
C
SUBROUTINE GCON(X,G)
PARAMETER (MX=10)
COMMON N,K,J,MPJ,NMK,PKJ
REAL*8 X(MX),G(MX)

C DEFINE INEQUALITY CONSTRAINT HERE
C NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY BELOW
C
G(1)=
G(2)=
C
C ADDING SLACK VARIABLES !!! =========== DO NOT ALTER =========== !!!
C
DO 10,I=1,J
   G(I)=G(I)-X(I+N)
10 CONTINUE
RETURN
END

C===========================================================
C
C USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY
C CONSTRAINTS
C
SUBROUTINE GRADG(X,GRDG)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
REAL*8 X(MX),GRDG(MX,MX)

C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
GRDG(1,1)=
GRDG(1,2)=
GRDG(1,3)=

C
GRDG(2,1)=
GRDG(2,2)=
GRDG(2,3)=

C GRADIENTS OF INEQUALITY CONSTRAINTS WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN VERSIONS GRGOPT14+
C
RETURN
END
Appendix C: Two-Level Decomposition Solutions

This appendix contains all of the information that a potential user needs to set up and solve a simple two-level optimization problem. The first section starts with some basic information about the code, followed by detailed information regarding the definition of variables, functions and subroutines for each of the sub-problems. The next section contains an explanation of how to run the program, including a description of the output data format. In the last section, the single-level problem TEST5, from Appendix B is decomposed into a two-level optimization problem. This problem was selected so its solution could be compared to the single-level optimum solution. This problem is used to verify the decomposition technique and demonstrate the process of setting up a two-level decomposition problem. The pertinent definition information for the problem is provided in section C.4 of this appendix.

C.1 General Usage Notes

Unlike the solution sequence to single-level problems, the solution sequence for two-level decomposition problems is interactive. For a given set of problem definition information, the decomposed optimization problem has been
reduced to a single parameter search for the initial step size that minimizes the system level objective function, $F_{sys}$.

The first step in setting up the problem is to partition the problem into $S$ subsystems. Then each subsystem's design variables, objective function and constraints are identified. Next, each subsystem is expressed as an optimization sub-problem in the format given by equations (3.1) through (3.6). Formulation of the system level sub-problem follows with the identification of the system level design variables, e.g. gross geometric dimensions of the problem, constraints and objective function. Note that only those constraints that actually involve system level design variables need be included in this sub-problem. This system level sub-problem is then expressed in the format given by equations (3.9) through (3.18).

A template has been created that demonstrates the proper arrangement of each of the sub-problem's dimensions, functions and constraints. This blueprint for two-level optimization problems, called Template 2, can be found in section C.5 of this appendix. Since the sub-problems are solved with the same subroutine that was used to solve single-level optimization problems, namely GRGOPT32, the inputs are in the same format as before with each of the $S+1$ sub-problems having its own set of definition parameters, functions and subroutines. Appendix B should serve as a reference for questions regarding input/output arguments for GRGOPT32.
Since the solution strategy of this technique involves solving each of the S+1 sub-problems once within each pass through the main solution loop, care must be taken to correctly define each sub-problem's definition parameters. Template 2 has three clearly labelled input zones for defining each sub-problem's set of definition parameters. The parameters defined in **Input Zone 1** include the sub-problem's dimensions: $N_s$, $K_s$ and $J_s$, the maximum number of iterations allowed, $\text{MAXIT}_s=3$; the limit on how many cycles per sub-problem solution, $\text{CYCLE}_s=1$; the parameter that determines how many times each sub-problem will be restarted; the upper and lower variable bounds, $\text{XUP}_s$ and $\text{XLOW}_s$; the initial tolerances, $\text{TOL1}_s$, $\text{TOL2}_s$ and $\text{TOL3}_s$; the initial feasible starting point $X^{(0)}_s$; the tolerance reduction parameter, $\text{FACTOR}=100$; the output suppression flag, $\text{PRINT}$; the output heading; the step size reduction parameter, $\text{GAMMA}=0.5$; and the limit on the maximum number of Newton's iterations allowed, $\text{NEWTIT}=25$. Though some of the names have been changed, these problem definition parameters are used in the manner as described in Appendix B. The advice on their selection for single-level optimization problems remains unchanged here.

Two new features have been added for the two-level solution method. It was necessary to add a tolerance to test for convergence of the system level objective function $F_{sys}$ after each pass through the solution loop. This convergence tolerance, $\text{TOL4}$ should be set to about $10^{-7}$ initially. It can
be changed during the solution sequence through the interactive menu. The other new feature was required to allow access to the inactive variables within any sub-problem. This is accomplished with the addition of a FORTRAN common block called /DECOMP/. Within this common block are the vector $Y$, the system level design variables and the array $XCOM$. The $s^{th}$ row vector of $XCOM$ corresponds to $X_s$, the local design variables of the $s^{th}$ first level sub-problems. (Eq C.1)

$$XCOM = \begin{bmatrix}
X_{1,1}, X_{1,2}, \ldots, X_{1,N1+1} \\
\vdots \\
X_{s,1}, X_{s,2}, \ldots, X_{s,Ns+Js} \\
\vdots \\
X_{S,1}, X_{S,2}, \ldots, X_{S,Ns+JS}
\end{bmatrix} \tag{C.1}$$

Since the data in $XCOM$ is the most current information available, it is used to define the new initial feasible point for each sub-problem. After each sub-problem is solved, the optimum solution vector, $X^*_s$ replaces the appropriate row in $XCOM$. In this fashion, slack variables used in more than one sub-problem will always satisfy the corresponding constraint. In Input Zone 1, the initial feasible vector must be converted to conform with the data structure defined by equation (3.18). The user must also update the slack variables in Input Zone 2. All other common block updates are handled automatically before and after each sub-problem solution.

Users should set logical variable PRINT to false thereby
forcing GRGOPT32 to return the minimum amount of information to the user. This will allow the process to converge in an reasonable amount of time.

Logical variable SYS is used to identify the system level sub-problem. The variable should be set false before each first level sub-problem and true before each system level sub-problem. When SYS is false, nonbasic variables that are within TOL3 of their bounds are excluded from the basis. When SYS is true, those nonbasic variables very near their bounds are allowed to enter the basis. Tests have found that this arrangement allows more freedom for variables to leave their bounds on the system level.

In Input Zone 3, the user must define the objective function, its gradient, the constraints and their gradients for each of the sub-problems. These functions and subroutines are defined exactly as for single level problems with one exception. Since there are now S+1 sets of these functions and subroutines, a labelling scheme that uniquely defines each sub-problem's set is necessary. The labelling scheme used is simply: Fi, GRADFi, HCONi, GRADHi, GCONi and GRADGi. There will be one such set of FORTRAN sub-programs for each value of i between 1 and S, the total number of subsystems. The system level function and subroutines are labelled FS, GRADFS, HCONS, GRADHS, GCONS and GRADGS.

The last problem formulation issue to cover is how to alter the solution loop to accommodate an arbitrary number of
subsystems. Firstly, all new variables added to the problem should be declared as integers or a double precision real numbers. The variable declaration statements in Template 2 are grouped in such a way as to make it clear what new variables will need to be declared for each new subsystem added. The portion of the solution loop that must be copied S times is also clearly marked. Every occurrence of i is simply replaced by the subsystem number to be added.

C.2 Output Format

When all the variables, functions and subroutines are defined, the file is ready to be compiled and linked with GRGOPT32. The DOS commands required for the Microsoft FORTRAN (Version 3) compiler/linker are:

```
FOR1 filename
PAS2 filename
LINK filename GRGOPT32
filename
```

At this time, an interactive session has been initiated and the user should see the menu reproduced in Table C-1 below.
QUIT..........................INPUT 0
INPUT ALPHA0....................INPUT 1
INPUT AN INTERVAL OF ALPHA'
OVER WHICH TO SEARCH* ............INPUT 2
CHANGE OVERALL CONVERGENCE TOL4..INPUT 3

*QUADRATIC INTERPOLATION FINDS
"BEST" ALPHA IN RANGE GIVEN

Table C-1 Main Interactive Menu for Decomposition Problems.

If the user wants to exit the program, an entry of 0 will suffice and the session is over. If a decision is made to enter a value of $\alpha^0$, typing 1 causes the program to prompt the user for a new value of $\alpha^0$. When PRINT is set false, a single status report will be output after each pass through the solution loop. Table C-2 contains the format for this status report. The first status report for pass zero, gives the initial step size, the convergence tolerance, the initial design vector and the system level objective function at this point. The subsequent status reports offer all of that information less the convergence tolerance and the initial step size, as they are constant.

These status reports are output after each pass until either the system level objective function converges or the
maximum number of passes is attained. In either case, the program returns to the main interactive menu.

<table>
<thead>
<tr>
<th>PASS</th>
<th>ALPHA0</th>
<th>TOL4</th>
<th>XCOM(1,i)</th>
<th>XCOM(2,i)</th>
<th>Y(i)</th>
<th>OBJECTIVE FUNCTION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>289.65780000000000</td>
<td>1.0000000000000E-016</td>
<td>2.4999999441206E-003</td>
<td>2.4999999441206E-003</td>
<td>3.0000000000000</td>
<td>2.0405693694318E-002</td>
</tr>
<tr>
<td>1</td>
<td>13333.3330781300000</td>
<td>33730.9609375000000</td>
<td>17888.5435837740000</td>
<td>43461.9001450740000</td>
<td>2.00000008889270</td>
<td>1.5782071475955E-002</td>
</tr>
</tbody>
</table>

Table C-2 Output Data Format for Decomposition Problems.

With no previous information available about the particular problem at hand, the user should try the following values of $\alpha^0$: 1, 10, 100, 1000, 0.1, 0.01, 0.001 and so forth. The value of the system level objective function resulting from each step size guess should serve as a guide for the next approximation.

The user can continue to select values for $\alpha^0$, or a range of step sizes can be specified over which to perform a search. This option returns the best initial step size in the interval based on an approximation by a quadratic interpolating
polynomial. This is accomplished by dividing the interval into ten sub-intervals and the routine simply searches for a function value that is less than the function values both to the left of the point being considered and to the right. When this situation is found, a quadratic polynomial is fit to the three points and the approximation for the minimum is returned. If the objective function value at either of the interval's endpoints is lower then that value is returned.

The user also has the option of changing the convergence tolerance. Initially, TOL4 should be set to about $10^{-7}$. When close to the best approximation for $a^0$, TOL4 can be reduced to $10^{-15}$ or even zero.

When dissatisfied with the resulting objective function values at this initial starting point, the user should quit the program, change the starting point and start over. This process should be repeated until a practical or acceptable optimum is found.

As before, a copy of all data that scrolls by on the monitor is dumped to a file called GRG.FOR. If the logical flag PRINT is set to true, the intermediate status data within each pass as well as any error messages will be output in the format described in section B.2 of Appendix B. Again, this is not recommended unless debugging the code is the goal.
C.3 Example Problems

**Problem Test4D**

This is the same problem that was used as a two-level demonstration example in Chapter 3. The problem's key definition parameters are listed in section C.4.

**Problem Test5D**

Problem Test5D is the two-level equivalent of the single-level problem Test5. This problem is also a modified version of the two-level example problem from Chapter 3. The system level sub-problem is modified with the addition of the horizontal coordinate of the common joint C (Figure C-1).

Mathematically, the two sub-problems for Test5D are now defined as follows:

Minimize:  \( F_i(x, y) = x_{1,1} \left( (y_i)^2 + (y_i')^2 \right) \)

Subject to:

\[ G_{1,1}(x, y) = 20(5-y_2)[((y_i)^2+(y_i')^2)^{\frac{1}{2}}/(y_i \times x_{1,1})] - x_{1,2} = 0 \]  
\[ 0 \leq x_{1,1} \leq 0.003 \text{ m}^2 \]  
\[ 0 \leq x_{1,2} \leq 100000 \text{ kN/m}^2 \]
Minimize: \( F_2(X,Y) = \frac{X_2}{I - ((5-Y_1)^2 + (Y_1)^2)^{\frac{1}{2}}} \) \( C.5 \)

Subject to:
\[
G_{2,1}(X,Y) = 20Y_2[(5-Y_1)^2 + (Y_1)^2)^{\frac{1}{2}} / (Y_1 X_{2,1})] \quad X_{2,2} = 0 \quad C.6
\]
\[
0 \leq X_{2,1} \leq 0.003 \text{ m}^2 \quad C.7
\]
\[
0 \leq X_{2,2} \leq 100000 \text{ kN/m}^2 \quad C.8
\]

The system level problem can now be expressed as

Minimize: \( F_{sys}(X,Y) = \frac{X_{I,1}}{I - ((Y_2)^2 + (Y_1)^2)^{\frac{1}{2}}} + X_{4,1} (5-Y_2)^2 + (Y_1)^2)^{\frac{1}{2}} \) \( C.9 \)

Subject to:
\[
G_{1}(X,Y) = 20(5-Y_2)[((Y_2)^2 + (Y_1)^2)^{\frac{1}{2}} / (Y_1 X_{1,1})] \quad Y_3 = 0 \quad C.10
\]
\[
G_1(X,Y) = 20Y_2[(5-Y_1)^2 + (Y_1)^2)^{\frac{1}{2}} / (Y_1 X_{2,1})] \quad Y_4 = 0 \quad C.11
\]
\[
1 \leq Y_1 \leq 3 \text{ m} \quad C.12
\]
\[
0 \leq Y_2 \leq 5 \text{ m} \quad C.13
\]
\[
0 \leq Y_3 \leq 100000 \text{ kN/m}^2 \quad C.14
\]
\[
0 \leq Y_4 \leq 100000 \text{ kN/m}^2 \quad C.15
\]

Recall that from equation (3.18), the slack variables in the system level sub-problem are related to the slack variables in the first level sub-problems.

\[
Y_3 = X_{1,2} \quad C.16
\]
\[
Y_4 = X_{2,2} \quad C.17
\]
Two initial feasible $X^{(0)}$ vectors were required for this problem because the first attempt did not lead to an acceptable solution. The unsuccessful design variable vector and its minimum solution are listed in Table C-3. Comparison of the minimum objective function found with the solution to problem TEST5 reveals the inadequacy of this starting point. It is 88% larger than the true optimum.

The second starting point selected proved to be successful, with a value of the initial step size of $5 \times 10^{-5}$. Note that at the optimum solution, five of the six variables are at their bounds. Table C-4 lists the starting point and the minimum solution found. This solution matches the optimum
solution to the single-level problem, TEST5 from Appendix B. This problem's key definition parameters are listed in section C.4.

<table>
<thead>
<tr>
<th>PASS</th>
<th>ALPHA0</th>
<th>TOL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>140.0000000000000000</td>
<td>1.0000000000000000E-007</td>
</tr>
<tr>
<td></td>
<td>XCOM(1,i) = 3.00000000260770E-003</td>
<td>6479.0000000000000000</td>
</tr>
<tr>
<td></td>
<td>XCOM(2,i) = 3.00000000260770E-003</td>
<td>30410.0000000000000000</td>
</tr>
<tr>
<td></td>
<td>Y(i) = 3.0000000000000000</td>
<td>4.5000000000000000</td>
</tr>
</tbody>
</table>

OBJECTIVE FUNCTION VALUE ==> 2.5349124755379E-002

<table>
<thead>
<tr>
<th>PASS</th>
<th>OBJECTIVE FUNCTION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1.8858694040940E-003</td>
</tr>
</tbody>
</table>

Table C-3 Solution A for Problem Test5D.

<table>
<thead>
<tr>
<th>PASS</th>
<th>ALPHA0</th>
<th>TOL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.0000000000000000E-005</td>
<td>1.0000000000000000E-016</td>
</tr>
<tr>
<td></td>
<td>XCOM(1,i) = 3.00000000260770E-003</td>
<td>.0000000000000000</td>
</tr>
<tr>
<td></td>
<td>XCOM(2,i) = 3.00000000260770E-003</td>
<td>33333.3320312500000000</td>
</tr>
<tr>
<td></td>
<td>Y(i) = 3.0000000000000000</td>
<td>5.0000000000000000</td>
</tr>
</tbody>
</table>

OBJECTIVE FUNCTION VALUE ==> 2.6492855914821E-002

<table>
<thead>
<tr>
<th>PASS</th>
<th>OBJECTIVE FUNCTION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>1.0000000000000000E-003</td>
</tr>
</tbody>
</table>

Table C-4 Solution B for Problem Test5D.
C.4 Key Definition Information for Two-Level Problems

Problem Test4D (Two-Level Example Problem Ch. 3)

C**************BEGIN USER INPUT ZONE 1**************
C
C OUTPUT FILE: A COPY OF ALL DATA PRINTED TO THE SCREEN IS
C ALSO SENT TO FILE NAMED GRG.OUT
C
OPEN (1, FILE='TEST4D.OUT', STATUS='NEW')
C
C DEFINE PARAMETERS & BOUNDS FOR THE INDIVIDUAL SUB-PROBLEMS
C
C SYSTEM LEVEL
   NS=1
   KS=0
   JS=2
   CYCLES=1
   MAXITS=3
C
C UPPER BOUNDS SYS
   XUPS(1)=3.0
   XUPS(2)=100000.0
   XUPS(3)=100000.0
C
C LOWER BOUNDS SYS
   XLOWS(1)=1.0
   XLOWS(2)=0.0
   XLOWS(3)=0.0
C
C SUB-SYSTEM 1
   N1=1
   K1=0
   J1=1
   CYCLE1=1
   MAXIT1=3
C
C UPPER BOUNDS SUB 1
   XUP1(1)=0.003
   XUP1(2)=100000.0
C
C LOWER BOUNDS SUB 1
   XLOW1(1)=0.0
   XLOW1(2)=0.0
C
C SUB-SYSTEM 2
   N2=1
   K2=0
   J2=1
   CYCLE2=1
   MAXIT2=3
C
C UPPER BOUNDS SUB 2
   XUP2(1)=0.003
   XUP2(2)=100000.0
C
C LOWER BOUNDS SUB 2
XLOW2(1)=0.0
XLOW2(2)=0.0

FACTOR=100.0
PRINT=.FALSE.

C CONVERGENCE TOLERANCE FOR SYSTEM LEVEL OBJECTIVE FUNCTION
C USED TO CONTROL THE PRECISION OF THE WHOLE PROCESS
C
TOL4=1.0D-7

C WRITE PROBLEM HEADER TO FILE
WRITE(l,*)'-----------------------------------------'
WRITE(l,*)'------------------------'
WRITE(l,*)'
WRITE(l,*)'DECOMPOSITION PROBLEM TEST4D:'
WRITE(l,*)'
WRITE(l,*)'URI KIRSCH: JRNL OF THE STRCTRL DIVISION',
WRITE(l,*)' APRIL 1975'
WRITE(l,*)'
WRITE(l,*)' VERIFICATION OF TEST4'
WRITE(l,*)'
WRITE(l,*)'-----------------------------------------'

C DEFINE INITIAL COMMON VALUES: INITIAL 'FEASIBLE' POINTS
C
C STARTING POINT 1 (ORIGINAL)
XCOM(1,1)=0.001
XCOM(1,2)=33333.333
XCOM(2,1)=0.001
XCOM(2,2)=84327.404
Y(1)=3.0

C STARTING POINT 2
XCOM(1,1)=0.0025
XCOM(1,2)=13333.333333
XCOM(2,1)=0.0025
XCOM(2,2)=33730.96172
Y(1)=3.0

C INITIAL POINT 3 (SINGLE LEVEL OPTIMUM SOLUTION)
XCOM(1,1)=4.472D-4
XCOM(1,2)=1.0D+5
XCOM(2,1)=8.944D-4
XCOM(2,2)=1.0D+5
Y(1)=2.0

Y(2)=XCOM(1,2)
Y(3)=XCOM(2,2)

C SUB-SYSTEM #1 INITIAL TOLERANCES
C SUB-SYSTEM #2 INITIAL TOLERANCES
C
TOL12=1.0D-5
TOL22(1)=100.0
TOL32(1)=1.0D-8
TOL32(2)=10.0

C SYSTEM LEVEL INITIAL TOLERANCES
C
TOL1S=1.0D-5
TOL2S(1)=100.0
TOL2S(2)=100.0
TOL3S(1)=1.0D-4
TOL3S(2)=10.0
TOL3S(3)=10.0

C****************END USER INPUT ZONE 1**************************
C
C****************BEGIN USER INPUT ZONE 2**************************
C
C UPDATE COMMON BOCK SLACK VARIABLES
C
XCOM(1,2)=Y(2)
XCOM(2,2)=Y(3)

C****************END USER INPUT ZONE 2**************************
C
C****************BEGIN USER INPUT ZONE 3**************************
C
C USER DEFINED FUNCTION: SYSTEM LEVEL OBJECTIVE FUNCTION
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=4.0
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
FS=(X1*SUB1)+(X2*SUB2)

C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION
C SYSTEM LEVEL
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=4.0
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
C THIS PROBLEM HAS NONE
C
C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
C THIS PROBLEM HAS NONE
C
C USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=4.0
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
G(1)=20.0*(5.0-Y2)*SUB1/(X1*Y1)
G(2)=20.0*Y2*SUB2/(X2*Y1)
C
C USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY
C CONSTRAINTS SYSTEM LEVEL
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=4.0
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
GRDG(1,1)=(20.0*(5.0-Y2)/X1)*((1.0/SUB1)-(SUB1/(Y1**2)))
&GRDG(2,1)=(20.0*Y2/X2)*((1.0/SUB2)-(SUB2/(Y1**2)))
C
C USER DEFINED FUNCTION: OBJECTIVE FUNCTION FOR SUBSYSTEM 1
C
Y2=4.0
F1=X(1)*DSQRT((Y(1)**2)+(Y2**2))
C
C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION
C FOR SUBSYSTEM 1
C
Y2=4.0
GRDF(1)=DSQRT((Y(1)**2)+(Y2**2))
FOR SUBSYSTEM 1

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS FOR SUBSYSTEM 1

Y2 = 4.0
SUB = DSQRT((Y(1)**2) + (Y2**2))
G(1) = 20.0*(5.0-Y2)*SUB/(X(1)*Y(1))

USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY CONSTRAINTS FOR SUBSYSTEM 1

Y2 = 4.0
SUB = DSQRT((Y(1)**2) + (Y2**2))
GRDG(1,1) = -20.0*(5.0-Y2)*SUB/(X(1)*X(1)*Y(1))

USER DEFINED FUNCTION: OBJECTIVE FUNCTION FOR SUBSYSTEM 2

Y2 = 4.0
F2 = X(1)*(DSQRT(((5.0-Y2)**2) + (Y(1)**2)))

USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION FOR SUBSYSTEM 2

Y2 = 4.0
GRDF(1) = DSQRT(((5.0-Y2)**2) + (Y(1)**2))

USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS FOR SUBSYSTEM 2

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS FOR SUBSYSTEM 2

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS FOR SUBSYSTEM 2

Y2 = 4.0
SUB = DSQRT(((5.0-Y2)**2) + (Y(1)**2))
G(1) = 20.0*Y2*SUB/(X(1)*Y(1))

USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY CONSTRAINTS FOR SUBSYSTEM 2

Y2 = 4.0
SUB = DSQRT((Y(1)**2) + ((5.0-Y2)**2))
GRDG(1,1) = -20.0*Y2*SUB/(X(1)*X(1)*Y(1))

C*****************END USER INPUT ZONE 3******************
Problem Test5D

C*******************BEGIN USER INPUT ZONE 1*******************

C OUTPUT FILE: A COPY OF ALL DATA PRINTED TO THE SCREEN IS
C ALSO SENT TO FILE NAMED GRG.OUT
C
C OPEN (1, FILE='TEST5D.OUT', STATUS='NEW')
C
C DEFINE PARAMETERS & BOUNDS FOR THE INDIVIDUAL SUB-PROBLEMS
C
C SYSTEM LEVEL
  NS=2
  KS=0
  JS=2
  CYCLES=1
  MAXITS=3

C UPPER BOUNDS SYS
  XUPS(1)=3.0
  XUPS(2)=5.0
  XUPS(3)=100000.0
  XUPS(4)=100000.0

C LOWER BOUNDS SYS
  XLOWS(1)=1.0
  XLOWS(2)=0.0
  XLOWS(3)=0.0
  XLOWS(4)=0.0

C SUB-SYSTEM 1
  N1=1
  K1=0
  J1=1
  CYCLE1=1
  MAXIT1=3

C UPPER BOUNDS SUB 1
  XUP1(1)=0.003
  XUP1(2)=100000.0

C LOWER BOUNDS SUB 1
  XLOW1(1)=0.0
  XLOW1(2)=0.0

C SUB-SYSTEM 2
  N2=1
  K2=0
  J2=1
  CYCLE2=1
  MAXIT2=3

C UPPER BOUNDS SUB 2
  XUP2(1)=0.003
  XUP2(2)=100000.0

C LOWER BOUNDS SUB 2
  XLOW2(1)=0.0
XLOW2(2)=0.0
C FACTOR=100.0
PRINT=.FALSE.
C
C CONVERGENCE TOLERANCE FOR SYSTEM LEVEL OBJECTIVE FUNCTION
C USED TO CONTROL THE PRECISION OF THE WHOLE PROCESS
C
TOL4=1.0D-7
C
C WRITE PROBLEM HEADER TO FILE
C
WRITE(1,*)'----------------------------------------------------------',
&'-----------------------------------------------',
WRITE(1,*)' DECOMPOSITION PROBLEM TEST5D: ' 
WRITE(1,*)' ' 
WRITE(1,*)' URI KIRSCH: JRNL OF THE STRCTRL DIVISION',
&' APRIL 1975'
WRITE(1,*)' ' 
WRITE(1,*)' MODIFIED INCLUDES 2 SYSTEM LEVEL GEOMETRY'
&,' DIMENSIONS'
WRITE(1,*)' ' 
WRITE(1,*)' VERIFICATION OF TEST5' 
WRITE(1,*)' ' 
WRITE(1,*)' ' 
WRITE(1,*)'----------------------------------------------------------',
&'-----------------------------------------------'
C
C DEFINE INITIAL COMMON VALUES: INITIAL 'FEASIBLE' POINTS
C
C STARTING POINT 1
XCOM(1,1)=0.003 
XCOM(1,2)=0.0 
XCOM(2,1)=0.003 
XCOM(2,2)=33333.333333333333 
Y(1)=3.0 
Y(2)=5.0 
Y(3)=XCOM(1,2) 
Y(4)=XCOM(2,2)
C
C STARTING POINT 2
C XCOM(1,1)=
C XCOM(1,2)=
C XCOM(2,1)=
C XCOM(2,2)=
C Y(1)=
C Y(2)=
C
C SUB-SYSTEM #1 INITIAL TOLERANCES
C
TOL11=1.0D-5 
TOL21(1)=100.0
TOL31(1)=1.0D-8  
TOL31(2)=10.0  
C  
C SUB-SYSTEM #2 INITIAL TOLERANCES  
C  
TOL12=1.0D-5  
TOL22(1)=100.0  
TOL32(1)=1.0D-8  
TOL32(2)=10.0  
C  
C SYSTEM LEVEL INITIAL TOLERANCES  
C  
TOL1S=1.0D-5  
TOL2S(1)=100.0  
TOL2S(2)=100.0  
TOL3S(1)=1.0D-4  
TOL3S(2)=1.0D-4  
TOL3S(3)=10.0  
TOL3S(4)=10.0  
C  
C*******************END USER INPUT ZONE 1********************  
C*******************BEGIN USER INPUT ZONE 2*******************  
C  
C UPDATE COMMON BLOCK SLACK VARIABLES  
C  
XCOM(1,2)=Y(3)  
XCOM(2,2)=Y(4)  
C  
C*******************END USER INPUT ZONE 2*******************  
C*******************BEGIN USER INPUT ZONE 3*******************  
C  
C USER DEFINED FUNCTION: SYSTEM LEVEL OBJECTIVE FUNCTION  
C  
X1=XCOM(1,1)  
X2=XCOM(2,1)  
Y1=X(1)  
Y2=X(2)  
SUB1=DSQRT((Y1**2)+(Y2**2))  
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))  
FS=(X1*SUB1)+(X2*SUB2)  
C  
C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION  
C SYSTEM LEVEL  
C  
X1=XCOM(1,1)  
X2=XCOM(2,1)  
Y1=X(1)  
Y2=X(2)  
SUB1=DSQRT((Y1**2)+(Y2**2))  
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))  
GRDF(1)=(X1*Y1/SUB1)+(X2*Y1/SUB2)  
GRDF(2)=(X1*Y2/SUB1)-(X2*(5.0-Y2)/SUB2)  
C
C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
C THIS PROBLEM HAS NONE
C
C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
C THIS PROBLEM HAS NONE
C
C USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=X(2)
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
G(1)=20.0*(5.0-Y2)*SUB1/(X1*Y1)
G(2)=20.0*Y2*SUB2/(X2*Y1)

C USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY CONSTRAINTS
C SYSTEM LEVEL
C
X1=XCOM(1,1)
X2=XCOM(2,1)
Y1=X(1)
Y2=X(2)
SUB1=DSQRT((Y1**2)+(Y2**2))
SUB2=DSQRT((Y1**2)+((5.0-Y2)**2))
GRDG(1,1)=(20.0*(5.0-Y2)/X1)*((1.0/SUB1)-
&(SUB1/(Y1**2)))
GRDG(1,2)=(20.0/(X1*Y1))*((5.0-Y2)*Y2/SUB1)-SUB1
GRDG(2,1)=(20.0*Y2/X2)*((1.0/SUB2)-(SUB2/(Y1**2)))
GRDG(2,2)=(20.0/(X2*Y1))*((SUB2-((5.0-Y2)*Y2/SUB2))

C USER DEFINED FUNCTION: OBJECTIVE FUNCTION FOR SUBSYSTEM 1
C
F1=X(1)*(DSQRT((Y(1)**2)+(Y(2)**2)))

C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION FOR SUBSYSTEM 1
C
GRDF(1)=DSQRT((Y(1)**2)+(Y(2)**2))

C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
C FOR SUBSYSTEM 1
C
C THIS PROBLEM HAS NONE
C
C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
C FOR SUBSYSTEM 1
FOR SUBSYSTEM 1

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
FOR SUBSYSTEM 1

SUB=DSQRT((Y(1)**2)+(Y(2)**2))
G(1)=20.0*(5.0-Y(2))*SUB/(X(1)*Y(1))

USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY
CONSTRAINTS FOR SUBSYSTEM 1

SUB=DSQRT((Y(1)**2)+(Y(2)**2))
GRDG(1,1)=-20.0*(5.0-Y(2))*SUB/(X(1)*X(1)*Y(1))

USER DEFINED FUNCTION: OBJECTIVE FUNCTION FOR SUBSYSTEM 2

F2=X(1)*(DSQRT(((5.0-Y(2))**2)+(Y(1)**2)))

USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION
FOR SUBSYSTEM 2

GRDF(1)=DSQRT(((5.0-Y(2))**2)+(Y(1)**2))

USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
FOR SUBSYSTEM 2

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
FOR SUBSYSTEM 2

THIS PROBLEM HAS NONE

USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
FOR SUBSYSTEM 2

SUB=DSQRT(((5.0-Y(2))**2)+(Y(1)**2))
G(1)=20.0*Y(2)*SUB/(X(1)*Y(1))

USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY
CONSTRAINTS FOR SUBSYSTEM 2

SUB=DSQRT((Y(1)**2)+((5.0-Y(2))**2))
GRDG(1,1)=-20.0*Y(2)*SUB/(X(1)*X(1)*Y(1))

C****************************************END USER INPUT ZONE 3****************************************
C.5 Template 2: Two-Level Optimization Problem Outline.

C PROGRAM: TEMPLT2.FOR [USE WITH GRGOPT32]
C BY: TOM VEILLEUX
C
C DATE CREATED: 10-16-91
C DATE MODIFIED: 10-24-91 I/O REFINED
C
C TEMPLATE FOR TWO-LEVEL DECOMPOSITION PROBLEMS
C
C PURPOSE:
C
THIS PROGRAM EMPLOYS THE GENERALIZED REDUCED GRADIENT
METHOD TO SOLVE A MULTI-LEVEL DECOMPOSED OPTIMIZATION
PROBLEM. SUBROUTINE GRGOPT32 IS EMPLOYED ON BOTH THE
SUB-SYSTEM LEVEL AND THE SYSTEM LEVEL. THE INPUT
REQUIREMENTS FOR GRGOPT32 ARE DESCRIBED BELOW. THE
USER IS PROMPTED FOR THE VALUE OF ALPHAO, THE INITIAL
STEP SIZE, OR A RANGE OF ALPHAOs OVER WHICH TO PERFORM
A SEARCH. THE SUB-SYSTEMS ARE SOLVED FIRST, FOLLOWED
BY THE OPTIMIZATION OF THE SYSTEM. THESE SUB-PROBLEMS
HAVE THEIR MAXIMUM NUMBER OF ITERATIONS LIMITED TO 2
TO 5. THE TOLERANCES ARE REDUCED ONLY WHEN AN OPTIMUM
SOLUTION HAS BEEN FOUND. THIS CYCLE IS REPEATED UNTIL
THE CYCLE CONVERGES TO ONE VALUE OR UNTIL A MAXIMUM
NUMBER OF RESTARTS HAS OCCURRED.

C GRGOPT32:
C
THIS PROGRAM EMPLOYS THE GENERALIZED REDUCED GRADIENT
METHOD TO FIND THE OPTIMAL SOLUTION TO A CONSTRAINED
NONLINEAR OPTIMIZATION PROBLEM. THE ALGORITHM CAN
HANDLE A NONLINEAR OBJECTIVE FUNCTION, NONLINEAR
EQUALITY AND INEQUALITY CONSTRAINTS AND BOUNDED
VARIABLES. INEQUALITY CONSTRAINTS ARE CONVERTED TO
EQUALITIES BY ADDING SLACK VARIABLES. THIS INCREASES
THE DIMENSION OF THE PROBLEM FROM N TO N+J. VARIABLE
BOUNDS ARE DEALT WITH IMPLICITLY BY VARIOUS CHECKS
THROUGHOUT THE ALGORITHM. THE DESIGN VARIABLES ARE
PARTITIONED INTO K+J BASIC (DEPENDENT) VARIABLES AND
N-K NONBASIC (INDEPENDENT) VARIABLES. BY SOLVING FOR
THE NONBASIC VARIABLES IN TERMS OF THE BASIC
VARIABLES, THIS PARTITION EFFECTIVELY REDUCES THE
DIMENSION OF THE PROBLEM FROM N+J TO N-K.

C REFERENCES: ENGINEERING OPTIMIZATION:
C METHODS & APPLICATIONS
C STANDARD PROBLEM FORMAT: PROBLEM DIMENSION N
C
C MINIMIZE     F(X)     FOR X AN N-VECTOR
C
C SUBJECT TO  Hk(X) = 0     FOR k = 1, 2, ..., K
C           Aj < Gj(X) < Bj     FOR j = 1, 2, ..., J
C           XLOWERi < Xi < XUPPERi FOR i = 1, 2, ..., N
C
C CONVERSTED PROBLEM FORMAT: PROBLEM DIMENSION N + J
C
C MINIMIZE     F(X)     FOR X AN (N+J)-VECTOR
C
C SUBJECT TO  Hk(X) = 0     FOR k = 1, 2, ..., K
C           Gj(X) - XN+j = 0     FOR j = 1, 2, ..., J
C           XLLOWERi < Xi < XUPPERi FOR i = 1, 2, ..., N
C           Aj < XN+j < Bj     FOR j = 1, 2, ..., J
C
C USAGE:
C
CALL RGOPT32(N,K,J,ALPHA0,GAMMA,TOL1,TOL2,TOL3,X0,
&XUPPER,XLOWER,MAXIT,NEWIT,F,GRADF,HCON,GRADH,GCON,
&GRADG,PRINT,SYS,X,OKAY)
C
THE USER MUST PROVIDE THE FOLLOWING INPUT VARIABLES,
FUNCTIONS & SUBROUTINES.
C
C => USER DEFINED INPUT ARGUMENTS:
C
N:       NUMBER OF INDEPENDENT VARIABLES
K:       NUMBER OF EQUALITY CONSTRAINTS
J:       NUMBER OF INEQUALITY CONSTRAINTS
ALPHA0:  INITIAL STEP SIZE
GAMMA:   STEP SIZE REDUCTION PARAMETER (USE 0.5)
TOL1:    CONVERGENCE TOLERANCE FOR REDUCED GRADIENT
TOL2:    TOLERANCE FOR FEASIBILITY OF CONSTRAINTS
TOL3:    CONVERGENCE TOLERANCE FOR NEWTON ITERATIONS
X0:      INITIAL FEASIBLE POINT
XUPPER:  VECTOR OF UPPER BOUNDS OF X
XLOWER:  VECTOR OF LOWER BOUNDS OF X
MAXIT:   MAX NUMBER OF ITERATIONS, A TERMINATION LIMIT
NEWIT:   MAX NUMBER OF NEWTON ITERATIONS ALLOWED: USE 3
PRINT:   STATUS PRINT-OUT BY-PASS  TRUE: PRINT
          FALSE: BY-PASS
SYS:     LOGICAL USED IN IDENTIFYING SYSTEM LEVEL
DECOMPOSITION
PROBLEMS.

TRUE: SYSTEM LEVEL DECOMPOSITION PROBLEM
NON-BASIC VARIABLES NEAR THEIR
BOUNDS ARE ALLOWED TO ENTER THE
BASIS.

FALSE: NON-BASIC VARIABLES NEAR THEIR
BOUNDS ARE EXCLUDED FROM ENTERING
THE BASIS.

=> USER DEFINED FUNCTIONS: (MUST BE DECLARED EXTERNAL)
F(X): OBJECTIVE FUNCTION

=> USER DEFINED SUBROUTINES: MUST BE DECLARED EXTERNAL
GRADF(X,GRDF) GRADIENT OF OBJECTIVE FUNCTION
HCON(X,H) EQUALITY CONSTRAINTS
GRADH(X,GRDH) GRADIENT OF EQUALITY CONSTRAINTS
GCON(X,G) INEQUALITY CONSTRAINTS
GRADG(X,GRDG) GRADIENT OF INEQUALITY CONSTRAINTS

NOTE: SLACK VARIABLES ARE HANDLED
AUTOMATICALLY

=> ARGUMENTS: (USED IN SUB PROGRAMS)
X: INDEPENDENT VARIABLES VECTOR, LENGTH N+J
GRDF: GRADIENT OF THE OBJECTIVE FUNCTION, LENGTH N+J
H: EQUALITY CONSTRAINTS, LENGTH K
GRDH: GRADIENT OF THE EQUALITY CONSTRAINTS
ORDER K BY N+J
G: INEQUALITY CONSTRAINTS, LENGTH J
GRDG: GRADIENT OF THE INEQUALITY CONSTRAINTS,
ORDER J BY N+J

NOTE: ALL ARRAYS AND VECTORS MUST BE DIMENSIONED TO THE
SAME SIZE IN THE MAIN CALLING AND IN EACH SUBROUTINE
USE MX=10

=> VALUES RETURNED BY GRGOPT32:
X: OPTIMUM SOLUTION (IF DETERMINED SUCCESSFULLY...)

C LENGTH N+J
C OKAY: LOGICAL VARIABLE THAT DENOTES SUCCESS OF GRGOPT32
C IN FINDING THE OPTIMUM
C OKAY = .TRUE. IF OPTIMUM SOLUTION WAS DETERMINED
C OKAY = .FALSE. IF OPTIMUM SOLUTION WAS NOT
C DETERMINED
C
C=================================================================================
C
C => TEMPLATE FOR DECOMPOSITION PROBLEM
C
C => KEY VARIABLES: (DECLARED COMMON IN GRGOPT32)
C
C Y: SYSTEM LEVEL DESIGN VARIABLES
C
C XCOM: MATRIX OF LOCAL SUB-SYSTEM LEVEL DESIGN VARIABLES. THE ROWS CORRESPOND TO THE LOCAL
C DESIGN VARIABLES OF THE VARIOUS SUBSYSTEMS. THE iTH ROW BEING ASSOCIATED WITH THE iTH
C SUB-SYSTEM.
C
C-----------------------------------------------------------------------------
C SYSTEM LEVEL FUNCTIONS & SUBROUTINES END WITH SUFFIX 'S'
C SINCE THEY ARE PASSED TO SUBROUTINE GRGSYS32, ALL MUST BE
C DECLARED EXTERNAL
C
C FS(X): SYSTEM LEVEL OBJECTIVE FUNCTION (SUM OF ALL SUB-SYSTEM OBJECTIVE FUNCTIONS)
C
C GRADFS(X,GRDF) GRADIENT OF SYSTEM LEVEL OBJECTIVE FUNCTION WITH RESPECT TO Y
C
C HCONS(X,H) SYSTEM LEVEL EQUALITY CONSTRAINTS
C
C GRADHS(X,GRDH) GRADIENT OF SYSTEM LEVEL EQUALITY CONSTRAINTS WITH RESPECT TO Y
C
C GCONS(X,G) SYSTEM LEVEL INEQUALITY CONSTRAINTS
C
C GRADGS(X,GRDG) GRADIENT OF SYSTEM LEVEL INEQUALITY CONSTRAINTS WITH RESPECT TO Y
C
C NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY
C
C-----------------------------------------------------------------------------
C
C SUB-SYSTEM LEVEL FUNCTIONS & SUBROUTINES END WITH SUFFIX '#', THE SUB-SYSTEM NUMBER. REPEAT FOR EACH
C SUB-SYSTEMS AS REQUIRED. SINCE THEY ARE PASSED TO PASSED
C TO SUBROUTINE GRGOPT32, ALL MUST BE DECLARED EXTERNAL.
C
C F#(X): SUB-SYSTEM '#' OBJECTIVE FUNCTION
C GRADF#(X,GRDF) GRADIENT OF SUB-SYSTEM '#' OBJECTIVE FUNCTION WITH RESPECT TO THE LOCAL SUB-SYSTEM DESIGN VARIABLES VECTOR: XCOM(#,i)
C HCON#(X,H) SUB-SYSTEM '#' EQUALITY CONSTRAINTS
C GRADH#(X,GRDH) GRADIENT OF SUB-SYSTEM '#' EQUALITY CONSTRAINTS WITH RESPECT TO THE LOCAL SUB-SYSTEM DESIGN VARIABLES VECTOR: XCOM(#,i)
C GCON#(X,G) SUB-SYSTEM '#' INEQUALITY CONSTRAINTS
C GRADG#(X,GRDG) GRADIENT OF SUB-SYSTEM '#' INEQUALITY CONSTRAINTS WITH RESPECT TO THE LOCAL SUB-SYSTEM DESIGN VARIABLES VECTOR: XCOM(#,i)
C NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY
C
C BEGIN MAIN PROGRAM
C SUBPROGRAMS:
REAL*8 FS,F1,F2
C EXTERNAL FS,GRADFS,HCONS,GRADHS,GCONS,GRADG1,
&F1,GRADF1,HCON1,GRADH1,GCON1,GRADG1,
&F2,GRADF2,HCON2,GRADH2,GCON2,GRADG2
C INPUT PARAMETERS:
PARAMETER (MX=10)
C VARIABLES:
INTEGER MAXIT,NEWTIT,N,K,JPJ,NMK,LABEL,ANSWER,
&WHERE,IQ,CYCLE,ISLACK,
&ISYS,NS,KS,JS,CYCLES,MAXITS,
&ISUB1,N1,K1,J1,CYCLE1,MALT1,
&ISUB2,N2,K2,J2,CYCLE2,MALT2
C REAL*8 X0(MX),X(MX),ALPHA0,GAMMA,Y(MX),XCOM(MX,MX),
&FACTOR,S1,S2,S3,STEP,XRIGHT,XLEFT,SUM1,SUM2,SUM3,
&FUN1,FUN2,FUN3,FRIGHT,FLEFT,FTEMP,FPLAST,TOL4,
&XUPS(MX),XLOWS(MX),TOL11,TOL21,MX),TOL31(MX),
&XUP1(MX),XLOW1(MX),TOL12,TOL22(MX),TOL32(MX)
C REAL TEMP
C LOGICAL OKAY,OK,BOUNDS(MX),PRINT,SYS
C COMMON N,K,JPJ,NMK,JPJ
COMMON /DECOMP/Y,XCOM

ANSWER=0
WHERE=20

C**********************BEGIN USER INPUT ZONE 1**************

C OUTPUT FILE: A COPY OF ALL DATA PRINTED TO THE SCREEN IS
C ALSO SENT TO FILE NAMED GRG.OUT

C OPEN (1, FILE='GRG.OUT', STATUS='NEW')

C DEFINE PARAMETERS & BOUNDS FOR THE INDIVIDUAL SUB-PROBLEMS

C SYSTEM LEVEL
NS=
KS=
JS=
CYCLES=1
MAXITS=3

C UPPER BOUNDS SYS
XUPS(1)=
XUPS(2)=
XUPS(3)=
XUPS(4)=

C LOWER BOUNDS SYS
XLOWS(1)=0.0
XLOWS(2)=0.0
XLOWS(3)=0.0
XLOWS(4)=0.0

C SUB-SYSTEM 1
N1=
K1=
J1=
CYCLE1=1
MAXIT1=3

C UPPER BOUNDS SUB 1
XUP1(1)=
XUP1(2)=

C LOWER BOUNDS SUB 1
XLOW1(1)=0.0
XLOW1(2)=0.0

C SUB-SYSTEM 2
N2=
K2=
J2=
CYCLE2=1
MAXIT2=3

C UPPER BOUNDS SUB 2
XUP2(1)=

XUP2(2) =
LOWER BOUNDS SUB 2
XLOW2(1)=0.0
XLOW2(2)=0.0

FACTOR=100.0
PRINT=.FALSE.

CONVERGENCE TOLERANCE FOR SYSTEM LEVEL OBJECTIVE FUNCTION
USED TO CONTROL THE PRECISION OF THE WHOLE PROCESS
TOL4=1.0D-7

WRITE PROBLEM HEADER TO FILE

WRITE(1,*)'-----------------------------------------1
&f------------------------1
WRITE(^,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'
WRITE(l,*)'-----------------------------------------~
I
&'------------------------~

RESTART HERE TO REDEFINE XOi & Y0
10 CONTINUE

DEFINE INITIAL COMMON VALUES: INITIAL 'FEASIBLE' POINTS

STARTING POINT 1
XCOM(1,1)=
XCOM(1,2)=
XCOM(2,1)=
XCOM(2,2)=
Y(1)=
Y(2)=
Y(3)=XCOM(1,2)
Y(4)=XCOM(2,2)

STARTING POINT 2
XCOM(1,1)=
XCOM(1,2)=
XCOM(2,1)=
XCOM(2,2)=
Y(1)=
Y(2)=
SUB-SYSTEM #1 INITIAL TOLERANCES

TOL11=1.0D-5
TOL21(1)=
TOL31(1)=
TOL31(2)=

SUB-SYSTEM #2 INITIAL TOLERANCES

TOL12=1.0D-5
TOL22(1)=
TOL32(1)=
TOL32(2)=

SYSTEM LEVEL INITIAL TOLERANCES

TOL1S=1.0D-5
TOL2S(1)=
TOL2S(2)=
TOL3S(1)=
TOL3S(2)=
TOL3S(3)=
TOL3S(4)=

*****************END USER INPUT ZONE 1*******************

2 FORMAT(' TOLERANCE 1 =',5X,E14.7)
3 FORMAT(' TOLERANCE 2 =',15,E14.7)
4 FORMAT(' TOLERANCE 3 =',15,E14.7)

START ANALYSIS LOOP FOR CYCLE OF SYSTEM & ALL SUB-SYSTEMS

CYCLE=0

IF (ANSWER.EQ.2) THEN
   IF (WHERE.EQ.21) GOTO 2100
   IF (WHERE.EQ.22) GOTO 2200
   IF (WHERE.EQ.23) GOTO 2300
   IF (WHERE.EQ.25) GOTO 2500
   IF (WHERE.EQ.28) GOTO 2800
ENDIF

WRITE INTERACTIVE MENU TO SCREEN

20 WRITE(*,*)'=====================================================================
 WRITE(*,*)'
 WRITE(*,*)' QUIT..........................INPUT 0'
 WRITE(*,*)' INPUT ALPHA0..........................INPUT 1'
 WRITE(*,*)' INPUT AN INTERVAL OF ALPHA'
 WRITE(*,*)' OVER WHICH TO SEARCH* ...........INPUT 2'
WRITE(*,*)' CHANGE OVERALL CONVERGENCE TOL4..INPUT 3'
WRITE(*,*)' '
WRITE(*,*)' *QUADRATIC INTERPOLATION FINDS'
WRITE(*,*)' "BEST" ALPHA IN RANGE GIVEN'
WRITE(*,*)' '
READ(*,*)ANSWER
IF (ANSWER.EQ.0) GOTO 3000
IF (ANSWER.EQ.1) GOTO 30
IF (ANSWER.EQ.2) GOTO 2000
IF (ANSWER.EQ.3) THEN
WRITE(*,*)'========================================='
WRITE(*,*)' '
WRITE(*,*)' TOL4 WAS ', TOL4
WRITE(*,*)' '
WRITE(*,*)' INPUT NEW VALUE FOR TOL4'
WRITE(*,*)' '
READ(*,*)TOL4
ENDIF
C RETURN TO INTERACTIVE MENU
GOTO 20
ENDIF
C ERROR TRAP MESSAGE
C
WRITE(*,*)'
WRITE(*,*)'
WRITE(*,*)' **\nWHAT YOU TYPED IS NOT AN OPTION**
WRITE(*,*)'
WRITE(*,*)' **\n***********\nW R I N G**
WRITE(*,*)' *******k**********************************
WRITE(*,*)'
C ERROR TRAP
C RETURN TO INTERACTIVE MENU
GOTO 20
C
C THIS IS WHERE ALPHAO IS INPUT INTERACTIVELY
C
30 WRITE(*,*)'
WRITE(*,*)' INPUT VALUE FOR ALPHAO'
WRITE(*,*)'
READ(*,*)ALPHAO
C
C SUMMARY OF INITIAL FEASIBLE DATA
C
C WRITE TO SCREEN
90 WRITE(*,*)'=========================================',
&'============================='
WRITE(*,*)' PASS ==> ',CYCLE
WRITE(*,*)' ALPHAO ==> ',ALPHAO
WRITE(*,*)' TOL4 ==> ',TOL4

WRITE(*,*)'XCOM(1,i)=', (XCOM(1,i), I=1, (N1+J1))
WRITE(*,*)'XCOM(2,i)=', (XCOM(2,i), I=1, (N2+J2))
WRITE(*,*)' Y(i)=' , (Y(I), I=1, NS)

WRITE(*,*)' OBJECTIVE FUNCTION VALUE ==> ',FS(Y)

DO 1000, CYCLE=1,25

C WRITE SOLUTION SEQUENCE HEADER TO SCREEN

WRITE(*,*)'-----------------------------------------'

WRITE(*,*)' SOLUTION SEQUENCE NUMBER.. ', CYCLE, ' <+++'

WRITE(*,*)'-----------------------------------------'

C WRITE SOLUTION SEQUENCE HEADER TO FILE

WRITE(1,*)'-----------------------------------------'

WRITE(1,*)' OBJECTIVE FUNCTION VALUE ==> ',FS(Y)
&'------------------------'
WRITE(1,*),' ' 
WRITE(1,*),' SOLUTION SEQUENCE NUMBER....',CYCLE,' <+'
WRITE(1,*),' '
C
ENDIF
C
C INITIALIZE COUNTER FOR SYSTEM LEVEL SLACK VARIABLES
ISLACK=0
C
C DEFINE PROBLEM VARIABLES FOR SUB-SYSTEM #1
C
N=N1 
K=K1 
J=J1 
NPJ=N1+J1 
KPJ=K1+J1 
NMK=N1-K1 
GAMMA=0.5 
MAXIT=MAXIT1 
NEWTIT=25
C
C CALL GRGOPT32 FOR SUB-SYSTEM #1 PROBLEM
C
DO 103,ISUB1=1,CYCLE1 
C
C BY-PASS PRINT OUT?
IF(PRINT)THEN
C
C WRITE SUBTITLE TO SCREEN
WRITE(*,*)' ' 
WRITE(*,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++++++++++++++++++++++++++'
WRITE(*,*)'SUB-SYSTEM #1 SOLUTION # ',ISUB1 
WRITE(*,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++++++++++++++++++++++++++'
C
C WRITE SUBTITLE TO FILE
WRITE(1,*),' ' 
WRITE(1,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++++++++++++++++++++++++++'
WRITE(1,*)'SUB-SYSTEM #1 SOLUTION # ',ISUB1 
WRITE(1,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++++++++++++++++++++++++++'
C
C WRITE TOLERANCES TO OUTPUT FILE
C
WRITE(1,2)TOLL1 
WRITE(1,*),' ' 
WRITE(1,3)(M,TOL21(M),M=1,KPJ) 
WRITE(1,*),' ' 
WRITE(1,4)(M,TOL31(M),M=1,NPJ)
WRITE(1,*),'  
C  
ENDIF  
C INITIAL FEASIBLE POINT  
DO 100,I=1,NPJ  
X0(I)=XCOM(I,I)  
100 CONTINUE  
C  
SYS=.FALSE.  
CALL GRGOPT32(N,K,J,ALPHAO,GAMMA,TOL11,TOL21,TOL31,X0,  
&XUP1,XLOW1,MAXIT,NEWTIT,F1,GRADF1,HCON1,GRADH1,  
&GCON1,GRADG1,PRINT,SYS,X,OKAY)  
C UPDATE COMMON BLOCK SYSTEM VECTORS  
C  
DO 101,I=1,NPJ  
XCOM(I,I)=X(I)  
IF (I.GT.N) THEN  
  ISLACK=ISLACK+1  
  Y(NS+ISLACK)=X(I)  
ENDIF  
101 CONTINUE  
C CHECK FOR EXISTENCE OF SOLUTION  
C  
TEMP=CPCLE  
IF ((OKAY).OR.(AMOD(TEMP,3.0).EQ.0.0)) THEN  
  C REDUCE TOLERANCES  
  TOL11=TOL11/FACTOR  
  DO 102,I=1,NPJ  
    TOL21(I)=TOL21(I)/FACTOR  
    TOL31(I)=TOL31(I)/FACTOR  
  102 CONTINUE  
ENDIF  
103 CONTINUE  
C-----------------------------------------------------------  
C REPEAT FOR EACH SUBSYSTEM  
C  
C REPLACE EACH OCCURANCE OF i WITH THE SUBSYSTEM NUMBER  
C-----------------------------------------------------------  
C  
C DEFINE PROBLEM VARIABLES FOR SUB-SYSTEM #i  
C  
N=Ni  
K=Ki  
J=Ji
NPJ=Ni+Ji
KPJ=Ki+Ji
NMK=Ni-Ki
GAMMA=0.5
MAXIT=MAXITi
NEWTIT=25

CALL GRGOPT32 FOR SUB-SYSTEM #i PROBLEM

DO i03,ISUBi=1,CYCLEi

IF (PRINT) THEN

WRITE SUBTITLE TO SCREEN
WRITE(*,*) ''
WRITE(*,*)''++++++++++++++++++++++++++++++++++++++++++++++++++++++
& ''++++++++++++++++++++++++++++++++++++++++++++++++++++++
WRITE(*,*)''SUB-SYSTEM #i SOLUTION # ',ISUBi
WRITE(*,*)''++++++++++++++++++++++++++++++++++++++++++++++++++++++
& ''++++++++++++++++++++++++++++++++++++++++++++++++++++++

WRITE SUBTITLE TO FILE
WRITE(i,*)' '
WRITE(i,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++
& ''++++++++++++++++++++++++++++++++++++++++++++++++++++++
WRITE(i,*)'SUB-SYSTEM #i SOLUTION # ',ISUBi
WRITE(i,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++
& ''++++++++++++++++++++++++++++++++++++++++++++++++++++++

WRITE TOLERANCES TO OUTPUT FILE

WRITE(i,2)TOLli
WRITE(i,*)' '
WRITE(i,3)(M,TOL2i(M),M=1,KPJ)
WRITE(i,*)' '
WRITE(i,4)(M,TOL3i(M),M=1,NPJ)
WRITE(i,*)' '

ENDIF

INITIAL FEASIBLE POINT
DO i00,I=1,NPJ
   X0(I)=XCOM(i,I)
i00 CONTINUE

SYS=.FALSE.
CALL GRGOPT32(N,K,J,ALPHAO,GAMMA,TOLli,TOL2i,TOL3i,X0, &XUPi,XLOWi,MAXIT,NEWTIT,Fi,GRADFi,HCONi,GRADHi, &GCONi,GRADGi,PRINT,SYS,X,OKAY)
C UPDATE COMMON BLOCK SYSTEM VECTORS
C
DO i01, I=1, NPJ
   XCOM(i, I) = X(I)
   IF (I.GT.Ni) THEN
      ISLACK = ISLACK + 1
      Y(NS+ISLACK) = X(I)
   ENDIF
i01 CONTINUE
C
C CHECK FOR EXISTENCE OF SOLUTION
C
TEMP = CYCLE
IF ((OKAY).OR.(AMOD(TEMP, 3.0).EQ.0.0)) THEN
C
C REDUCE TOLERANCES
C
TOLi = TOLi / FACTOR
DO i02, I=1, NPJ
   TOL2i(I) = TOL2i(I) / FACTOR
   TOL3i(I) = TOL3i(I) / FACTOR
i02 CONTINUE
ENDIF
i03 CONTINUE
C
C DEFINE PROBLEM VARIABLES FOR SYSTEM LEVEL PROBLEM
C
N = NS
K = KS
J = JS
NPJ = NS + JS
KPJ = KS + JS
NMK = NS - KS
GAMMA = 0.5
MAXIT = MAXITS
NEWTIT = 25
C
C CALL GRGSYS32 FOR SYSTEM LEVEL PROBLEM
C
DO 903, ISYS = 1, CYCLES
C
C BY-PASS PRINT?
C
IF (PRINT) THEN
C
C WRITE SUBTITLE TO SCREEN
WRITE(*,*)' '
WRITE(*,*)'++++++++++++++++++++++++++++++++++++++++++++++++++++++'
&'++++++++++++++++++++++++++++++++++++++++++++++++++'
WRITE(*,*)'SYSTEM LEVEL SOLUTION REPEAT #', ISYS
WRITE(*,*),'+++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++'
WRITE(*,*),

C
C WRITE SUBTITLE TO FILE
WRITE(1,*),' SYSTEM LEVEL SOLUTION REPEAT #',ISYS
WRITE(1,*),'+++++++++++++++++++++++++++++++',
&'++++++++++++++++++++++++++++++'
WRITE(1,*),

C
C WRITE TOLERANCES TO OUTPUT FILE

WRITE(1,2)TOL1S
WRITE(1,*),
WRITE(1,3)(M,TOL2S(M),M=1,1KJ)
WRITE(1,*),
WRITE(1,4)(M,TOL3S(M),M=1,1NPJ)
WRITE(1,*),

C
ENDIF
C
C INITIAL FEASIBLE POINT

DO 900,I=1,1NPJ
   X0(I)=Y(I)
900 CONTINUE
C
SYS=.TRUE.
CALL GRGOPT32(N,K,J,ALPHA,ALPHA0,GAMMA,TOL1S,TOL2S,TOL3S,XO,
&XUPS,XLOWS,MAXIT,NEWTiT,FS,GRADFS,HCONS,GRADHS,
&GCONS,GRADGS,PRINT,SYs,X,OKAY)
C
C UPDATE COMMON BLOCK SYSTEM VECTORS

DO 901,I=1,1NPJ
   Y(I)=X(I)
901 CONTINUE
C
C*****************BEGIN INPUT ZONE 2*******************
C
C UPDATE COMMON BLOCK SLACK VARIABLES

XCOM(1,2)=Y(3)
XCOM(2,2)=Y(4)
C
C*****************END INPUT ZONE 2*******************
C
C CHECK FOR EXISTENCE OF SOLUTION

TEMP=CYCLE
IF ((OKAY).OR.(AMOD(TEMP,3.0).EQ.0.0))THEN
C
C REDUCE TOLERANCES
C
TOL1S=TOL1S/FACTOR
DO 902,I=1,NPJ
   TOL2S(I)=TOL2S(I)/FACTOR
   TOL3S(I)=TOL3S(I)/FACTOR
902 CONTINUE
ENDIF
903 CONTINUE
C
C CHECK FOR CONVERGENCE
C
IF ((DABS((FS(Y)-FLAST)/FS(Y))).LE.(TOL4))GOTO 1001
C
C SUMMARY DATA DUMP
C
C WRITE TO SCREEN
WRITE(*,*)'========================================='
&'---------------------------------------------'
WRITE(*,*)' PASS ===> ',CYCLE
WRITE(*,*)' Y(i)='(Y(I),I=1,NS)
WRITE(*,*)' OBJECTIVE FUNCTION VALUE ===> ',FS(Y)
WRITE(*,*)'========================================='
&'---------------------------------------------'
C
C WRITE TO FILE
WRITE(1,*)'========================================='
&'---------------------------------------------'
WRITE(1,*)' PASS ===> ',CYCLE
WRITE(1,*)' Y(i)='(Y(I),I=1,NS)
WRITE(1,*)' OBJECTIVE FUNCTION VALUE ===> ',FS(Y)
WRITE(1,*)'========================================='
&'---------------------------------------------'
C
FLAST=FS(Y)
1000 CONTINUE
C
C REDEFINE INITIAL STARTING POINT
C RETURN TO INTERACTIVE MENU
1001 FTEMP=FS(Y)
   GOTO 10
C
THIS IS THE QUADRATIC INTERPOLATOR FOR ALPHA

2000 WRITE(*,*)' '
   WRITE(*,*)' INPUT THE LEFT AND RIGHT BOUNDS ON ALPHA' WRITE(*,*)''
   READ(*,*)XLEFT,XRIGHT

PURPOSE:

THIS SUBPROGRAM USES QUADRATIC INTERPOLATION TO FIND THE MINIMUM OF A FUNCTION \( f(x) \) ON A SPECIFIED INTERVAL \( (x \text{ on } [\text{XLEFT, XRIGHT}]) \). THE ROUTINE USES THE LAGRANGE INTERPOLATING POLYNOMIAL THAT INTERPOLATES TO THREE POINTS THAT 'BOUND' THE FUNCTION'S MINIMUM. THE FUNCTION AND THE SEARCH INTERVAL ARE THE INPUT AND THE VALUE OF THE INDEPENDENT VARIABLE AT THE FUNCTION'S MINIMUM IS THE OUTPUT. THE MAXIMUM NUMBER OF FUNCTION CALLS REQUIRED WILL BE 13 EACH TIME QUADRINT IS USED.

ARGUMENTS:

- **XLEFT**: LEFT ENDPOINT OF SEARCH INTERVAL (INPUT)
- **XRIGHT**: RIGHT ENDPOINT OF SEARCH INTERVAL (INPUT)

OVERVIEW:

DIVIDE INTERVAL INTO 10 SEGMENTS.

START AT ONE STEP BEFORE XLEFT & FINISH ONE STEP AFTER XRIGHT. THIS ALLOWS A SEARCH FOR MINIMA THAT OCCUR VERY CLOSE TO THE ENDPOINTS AS WELL AS THOSE IN THE INTERIOR OF THE INTERVAL.

STEP ACROSS INTERVAL EVALUATING THE FUNCTION AT 1/10 SUBINTERVALS TO ATTEMPT TO BRACKET THE MINIMUM.

IF THE MINIMUM IS BRACKETED THEN USE QUADRATIC INTERPOLATION TO FIND \( \alpha \).

NOTE: THE NEXT OPTION IS NOT USED:

IF THE MINIMUM IS NOT BRACKETED IN THE SEARCH ACROSS THE INTERVAL THEN THE MINIMUM IS \( \min [ f(XRIGHT) \text{ or } f(XLEFT) ] \) \( \alpha \) IS DEFINED ACCORDINGLY.

\[
\text{STEP} = (\text{XRIGHT} - \text{XLEFT}) / 10.0 \\
S1 = \text{XLEFT} - \text{STEP} \\
\alpha = S1 \\
\text{WHERE} = 21 \\
\text{GOTO} 90 \\
2100 \quad \text{FUN1} = \text{FTEMP}
\]
C
S2=S1+STEP
ALPHA0=S2
WHERE=22
GOTO 90
2200 FUN2=FTEMP
FLEFT=FUN2
C
ALPHA0=XRIGHT
WHERE=23
GOTO 90
2300 FRIGHT=FTEMP
C
C LOOP UNTIL IQ=10
C
IQ=0
2400 S3=S2+STEP
ALPHA0=S3
WHERE=25
GOTO 90
2500 FUN3=FTEMP
C
IF ((FUN1.GE.FUN2).AND.(FUN3.GE.FUN2))THEN
C THE MINIMUM HAS BEEN BRACKETED, EXIT LOOP
C
GOTO 2600
ELSE
C UPDATE SEQUENCE AND CONTINUE SEARCHING
C
FUN1=FUN2
S1=S2
FUN2=FUN3
S2=S3
ENDIF
C
IF (IQ.NE.10)THEN
IQ=IQ+1
GOTO 2400
ENDIF
C
THE FUNCTION'S MINIMUM WAS NOT BRACKETED, THEREFORE THE
C FUNCTION'S MINIMUM OCCURS AT EITHER XRIGHT OR XLEFT
C RETURN TO CALLING PROGRAM WITH ALPHA.
C
IF (FRIGHT.LT.FLEFT) THEN
ALPHA0=XRIGHT
ELSE
ALPHA0=XLEFT
ENDIF
GOTO 2700

THE FUNCTION MINIMUM WAS BRACKETED. USE QUADRATIC INTERPOLATION TO FIND ALPHA. ALPHA IS THE ROOT OF THE DERIVATIVE OF THE LAGRANGE QUADRATIC INTERPOLATING POLYNOMIAL.

SEE BURDEN & FAIRES 'NUMERICAL ANALYSIS' PAGE 91

2600 CONTINUE

SUM1=FUN1*(S2+S3)
SUM2=FUN2*(S1+S3)*(-2.0)
SUM3=FUN3*(S1+S2)
ALPHAO=(SUM1+SUM2+SUM3)/(2*(FUN1-(2.0*FUN2)+FUN3))

IN THE EVENT THAT SEARCH NEAR THE ENDPOINTS YIELDS A MINIMA THAT LIES OUTSIDE THE INTERVAL: ALPHAO = LOWEND

IF((ALPHAO.LE.XLEFT).OR.(ALPHAO.GE.XRIGHT))THEN
  IF(FRIGHT.LT.FLEFT)THEN
    ALPHAO=XRIGHT
  ELSE
    ALPHAO=XLEFT
  ENDIF
ENDIF

FIND FS AT THE BEST ALPHAO

2700 WHERE=28
GOTO 90

2800 WRITE(*,*)' ',WRITE(*,*)' THE BEST ALPHAO WAS ===>',ALPHAO
WRITE(*,*)' THE BEST FS(Y) WAS ===>',FTEMP
WRITE(*,*)' '

RETURN TO MAIN MENU
GOTO 20

3000 STOP
END

C====================================================================END OF MAIN PROGRAM====================================================================

C************************************************************************BEGIN USER INPUT ZONE 3************************************************************************

USER DEFINED FUNCTION: SYSTEM LEVEL OBJECTIVE FUNCTION

FUNCTION FS(X)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),Y(MX),XCOM(MX,MX),FS

DEFINE OBJECTIVE FUNCTION HERE
C USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION
SYSTEM LEVEL

SUBROUTINE GRADFS(X,GRDF)
PARAMETER (MX=iO)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),Y(MX),XCOM(MX,MX),GRDF(MX)

C DEFINE OBJECTIVE FUNCTION GRADIENT HERE

GRDF(1)=
GRDF(2)=

C GRADIENTS OF OBJECTIVE FUNCTION WITH RESPECT TO SLACK
VARIABLES ARE HANDLED AUTOMATICALLY IN GRGOPT14+

RETURN
END

C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
SYSTEM LEVEL

SUBROUTINE HCONS(X,H)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),H(MX),Y(MX),XCOM(MX,MX)

C DEFINE EQUALITY CONSTRAINTS HERE

H(1)=
H(2)=

RETURN
END

C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
SYSTEM LEVEL

SUBROUTINE GRADHS(X,GRDH)
PARAMETER (MX=iO)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),GRDH(MX,MX),Y(MX),XCOM(MX,MX)

C DEFINE EQUALITY CONSTRAINT GRADIENT HERE
C
GRDH(1,1)=
GRDH(1,2)=
C
GRDH(2,1)=
GRDH(2,2)=
C
C GRADIENTS OF EQUALITY CONSTRAINTS WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN GROPT14+
C
RETURN
END
C===========================================================
C
C USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS
C
SUBROUTINE GCONS(X,G)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),G(MX),Y(MX),XCOM(MX,MX)

C DEFINE INEQUALITY CONSTRAINTS HERE
C NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY BELOW
C
G(1)=
G(2)=

C ADDING SLACK VARIABLES !!! ======> DO NOT ALTER <=== !!!
C
DO 10,I=1,J
   G(I)=G(I)-X(N+I)
10 CONTINUE
RETURN
END
C===========================================================
C
C USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY CONSTRAINTS
C
SUBROUTINE GRADGS(X,GRDG)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),GRDG(MX,MX),Y(MX),XCOM(MX,MX)

C DEFINE THE INEQUALITY CONSTRAINTS GRADIENTS HERE
GRDG(1,1) =
GRDG(1,2) =

GRDG(2,1) =
GRDG(2,2) =

GRADIENTS OF INEQUALITY CONSTRAINTS WITH RESPECT TO SLACK VARIABLES ARE HANDLED AUTOMATICALLY IN GRGOPT14+

RETURN
END

c===============================================

NOTE: USER MUST DEFINE THE OBJECTIVE FUNCTION, ITS GRADIENT, THE CONSTRAINTS AND THEIR GRADIENTS FOR EACH SUBSYSTEM.

Fi
GRADF Fi
HCONi
GRADHi
GCONi
GRADGi for i = 1,...,(Number of Subsytems)

C USER DEFINED FUNCTION: OBJECTIVE FUNCTION FOR SUBSYSTEM i

FUNCTION Fi(X)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),Y(MX),XCOM(MX,MX),Fi

Fi =

RETURN
END

c===============================================

USER DEFINED SUBROUTINE: GRADIENT OF OBJECTIVE FUNCTION FOR SUBSYSTEM i

SUBROUTINE GRADF i(X,GRDF)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),GRDF(MX),Y(MX),XCOM(MX,MX)
C DEFINE OBJECTIVE FUNCTION GRADIENT HERE
C
  GRDF(1)=
  GRDF(2)=
C
C GRADIENTS OF OBJECTIVE FUNCTION WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN GRGOPT14+
C
  RETURN
END

C USER DEFINED SUBROUTINE: EQUALITY CONSTRAINTS
C FOR SUBSYSTEM i
C
  SUBROUTINE HCONi(X,H)
  PARAMETER (MX=10)
  COMMON N,K,J,NPJ,NMK,KPJ
  COMMON /DECOMP/Y,XCOM
  REAL*8 X(MX),H(MX),Y(MX),XCOM(MX,MX)
C
  DEFINE EQUALITY CONSTRAINTS HERE
C
    H(1)=
    H(2)=
C
  RETURN
END

C USER DEFINED SUBROUTINE: GRADIENT OF EQUALITY CONSTRAINTS
C FOR SUBSYSTEM i
C
  SUBROUTINE GRADHi(X,GRDH)
  PARAMETER (MX=10)
  COMMON N,K,J,NPJ,NMK,KPJ
  COMMON /DECOMP/Y,XCOM
  REAL*8 X(MX),GRDH(MX,MX),Y(MX),XCOM(MX,MX)
C
  DEFINE GRADIENT HERE
C
    GRDH(1,1)=
    GRDH(1,2)=
    GRDH(2,1)=
    GRDH(2,2)=
C
C GRADIENTS OF EQUALITY CONSTRAINTS WITH RESPECT TO SLACK
C VARIABLES ARE HANDLED AUTOMATICALLY IN GRGOPT14+
C
  RETURN
END
C USER DEFINED SUBROUTINE: INEQUALITY CONSTRAINTS FOR SUBSYSTEM i

SUBROUTINE GCONi(X,G)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),G(MX),Y(MX),XCOM(MX,MX)

C DEFINE INEQUALITY CONSTRAINTS HERE
C NOTE: SLACK VARIABLES ARE HANDLED AUTOMATICALLY BELOW
C
G(1)=
C
C ADDING SLACK VARIABLES !!! ======> DO NOT ALTER ====== !!!
C
DO 10,I=1,J
   G(I)=G(I)-X(N+I)
10 CONTINUE
RETURN
END

C USER DEFINED SUBROUTINE: GRADIENT OF INEQUALITY CONSTRAINTS FOR SUBSYSTEM i

SUBROUTINE GRADGi(X,GRDG)
PARAMETER (MX=10)
COMMON N,K,J,NPJ,NMK,KPJ
COMMON /DECOMP/Y,XCOM
REAL*8 X(MX),GRDG(MX,MX),Y(MX),XCOM(MX,MX)

C DEFINE THE INEQUALITY CONSTRAINT GRADIENTS HERE
C
GRDG(1,1)=
C
C GRADIENTS OF INEQUALITY CONSTRAINTS WITH RESPECT TO SLACK VARIABLES ARE HANDLED AUTOMATICALLY IN GRGOPT14+
C
RETURN
END

C******************************END USER INPUT ZONE 3******************************