STUDENTS FACULTY STUDY RESEARCH DEVELOPMENT FUTURE CAREER CREATIVITY COMMUNITY LEADERSHIP TECHNOLOGY FRONTIERS DESIGN ENGINEERING APPLIED SCIENCE GEORGE WASHINGTON UNIVERSITY
A Penalty Function Computer Program for Solution, Sensitivity Analysis, and Optimal Value Bound Calculation in Parametric Nonlinear Programs

by Anthony V. Fiacco
Abolfazl Ghaemi

The George Washington University
School of Engineering and Applied Science
Institute for Management Science and Engineering

Contract DAAG 29-79-C-0062,
US Army Research Office

Contract N00014-75-C-0729,
Office of Naval Research

Grant ENG-7906104
National Science Foundation

This document has been approved for public sale and release; its distribution is unlimited.
This manual is intended to serve as a guide for coding, solving, and conducting sensitivity and optimal value bound analysis for parametric nonlinear programming problems with the computer programming model SENSUMT. The basic sensitivity results and bound calculation techniques are briefly reviewed and the algorithms implementing them are presented. A procedure for coding problems for SENSUMT and detailed illustrations of the computer
20. Abstract (continued)

Output is included. For completeness, the computer listing and a brief description of all the subroutines comprising SENSUMT, many of which are taken intact from a previously developed program SUMT-Version 4, are also provided.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENT</td>
<td>viii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. SENSITIVITY ANALYSIS IN NONLINEAR PROGRAMMING</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Basic Sensitivity Results</td>
<td>3</td>
</tr>
<tr>
<td>2.2 The Algorithm Implementing the Sensitivity Results</td>
<td>8</td>
</tr>
<tr>
<td>3. PARAMETRIC OPTIMAL VALUE BOUNDS</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Parametric Bounds on the Optimal Value Function of Convex Right Hand Side Problems CR(ε)</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Algorithm for Calculation of Bounds on f*(ε) of the Problem CR(ε)</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Parametric Bounds on the Optimal Value Function of the Problem SR(ε)</td>
<td>14</td>
</tr>
<tr>
<td>3.3.1 Separable right hand side perturbation problems</td>
<td>14</td>
</tr>
<tr>
<td>3.3.2 Convex underestimating problem</td>
<td>15</td>
</tr>
<tr>
<td>3.3.3 Convex overestimating problem</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Algorithm for Calculation of Bounds on the (Global) Optimal Value Function of the Problem SR(ε)</td>
<td>17</td>
</tr>
<tr>
<td>4. SENSUMT INPUT SPECIFICATIONS</td>
<td>19</td>
</tr>
<tr>
<td>4.1 User-supplied Subroutines</td>
<td>19</td>
</tr>
<tr>
<td>4.1.1 READIN</td>
<td>21</td>
</tr>
<tr>
<td>4.1.2 RESTNT (IN, VAL)</td>
<td>21</td>
</tr>
<tr>
<td>4.1.3 GRADI (IN)</td>
<td>22</td>
</tr>
<tr>
<td>4.1.4 MATRIX (IN, K)</td>
<td>22</td>
</tr>
<tr>
<td>4.2 User's Information Cards</td>
<td>23</td>
</tr>
</tbody>
</table>
4.2.1 Parameter card ................................. 24
4.2.2 Initial vector card(s) .......................... 24
4.2.3 First option card ............................... 25
4.2.4 Tolerance card ................................. 25
4.2.5 Second option card .............................. 25

5. CODED EXAMPLES AND INPUT/OUTPUT ILLUSTRATIONS ........ 32
5.1 Example 1 ....................................... 32
5.1.1 Computer listing of the code deck .................. 32
5.1.2 Selected pages from the computer output .............. 32

5.2 Example 2 ....................................... 51

6. GENERAL DESCRIPTION AND LISTING OF THE SENSUMT SUBROUTINES .. 59
6.1 BØDY ............................................ 61
6.2 BØIND .......................................... 61
6.3 CHECKER ....................................... 61
6.4 CONVRG ....................................... 65
6.5 DIFP ............................................ 65
6.6 DIFP2 .......................................... 65
6.7 ESTIM .......................................... 70
6.8 EVALU .......................................... 70
6.9 FEAS .......................................... 70
6.10 FINAL .......................................... 75
6.11 GRAD .......................................... 75
6.12 INVERS ........................................ 78
6.13 LMULT ......................................... 80
6.14 MAIN .......................................... 80
6.15 OPT ............................................ 80
6.16 OUTPUT ........................................ 86
6.17 PARDIF ........................................ 86
6.18 PERT ........................................... 86
6.19 PEVALU ........................................ 90
6.20 PRESEN ........................................ 91
6.21 PUNCH ......................................... 92
6.22 REJECT ........................................ 92
6.23 RHOCOM ....................................... 92
6.24 SEÇØND ........................................ 92
6.25 SECØRD ....................................... 96
6.26 SENS ........................................... 96
6.27 SET ............................................ 96
6.28 STORE ......................................... 102
6.29 TCHECK ........................................ 103
6.30 TIMEC ......................................... 103
6.31 TRANS .......................................... 104
6.32 XMØVE ......................................... 104

REFERENCES ......................................... 108
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Data Deck Structure for SENSUMT</td>
<td>31</td>
</tr>
<tr>
<td>2. Computer Listing of the Code for Example 1</td>
<td>33</td>
</tr>
<tr>
<td>3. Annotated Computer Output for Example 1</td>
<td>34</td>
</tr>
<tr>
<td>4. Graph of Bounds on $f^*(\varepsilon_2)$ of Example 1 (Computer Solution).</td>
<td>51</td>
</tr>
<tr>
<td>5. Computer Listing of the Code for the Convex Overestimating Problem of Example 2</td>
<td>54</td>
</tr>
<tr>
<td>6. Parametric Upper Bound on $f^*(\varepsilon_1)$ via Problem CSR(\varepsilon), Example 2 (Computer Solution)</td>
<td>56</td>
</tr>
<tr>
<td>7. Parametric Lower Bound on $f^*(\varepsilon_1)$ via Problem CSR(\varepsilon), Example 2 (Computer Solution)</td>
<td>57</td>
</tr>
<tr>
<td>8. Parametric Bounds on $f^*(\varepsilon)$, Example 2 (Computer Solution)</td>
<td>58</td>
</tr>
<tr>
<td>9. Subroutine BØDY</td>
<td>62</td>
</tr>
<tr>
<td>10. Subroutine BØUND</td>
<td>63</td>
</tr>
<tr>
<td>11. Subroutine CHECKER</td>
<td>66</td>
</tr>
<tr>
<td>12. Subroutine ČØNVRG</td>
<td>67</td>
</tr>
<tr>
<td>13. Subroutine DIFF1</td>
<td>68</td>
</tr>
<tr>
<td>14. Subroutine DIFF2</td>
<td>69</td>
</tr>
<tr>
<td>15. Subroutine ESTIM</td>
<td>71</td>
</tr>
<tr>
<td>16. Subroutine EVALU</td>
<td>73</td>
</tr>
<tr>
<td>17. Subroutine FEAS</td>
<td>76</td>
</tr>
<tr>
<td>18. Subroutine FINAL</td>
<td>77</td>
</tr>
<tr>
<td>19. Subroutine GRAD</td>
<td>79</td>
</tr>
<tr>
<td>Figure</td>
<td>Subroutine</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>20.</td>
<td>INVERS</td>
</tr>
<tr>
<td>21.</td>
<td>LMULT</td>
</tr>
<tr>
<td>22.</td>
<td>MAIN</td>
</tr>
<tr>
<td>23.</td>
<td>OPT</td>
</tr>
<tr>
<td>24.</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>25.</td>
<td>PARDIF</td>
</tr>
<tr>
<td>26.</td>
<td>PERT</td>
</tr>
<tr>
<td>27.</td>
<td>PEVALU</td>
</tr>
<tr>
<td>28.</td>
<td>PRESEN</td>
</tr>
<tr>
<td>29.</td>
<td>PUNCH</td>
</tr>
<tr>
<td>30.</td>
<td>REJECT</td>
</tr>
<tr>
<td>31.</td>
<td>RHOCOM</td>
</tr>
<tr>
<td>32.</td>
<td>SECOND</td>
</tr>
<tr>
<td>33.</td>
<td>SECOXD</td>
</tr>
<tr>
<td>34.</td>
<td>SENS</td>
</tr>
<tr>
<td>35.</td>
<td>SET</td>
</tr>
<tr>
<td>36.</td>
<td>STORE</td>
</tr>
<tr>
<td>37.</td>
<td>TCHECK</td>
</tr>
<tr>
<td>38.</td>
<td>TIMEC</td>
</tr>
<tr>
<td>39.</td>
<td>TRANS</td>
</tr>
<tr>
<td>40.</td>
<td>XMOVE</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Convex nuf and Concave nof of the Single Variable Function T(z) over the Closed Interval a ≤ z ≤ b</td>
<td>20</td>
</tr>
<tr>
<td>2. Parameter Card</td>
<td>24</td>
</tr>
<tr>
<td>3. First Option Card</td>
<td>26</td>
</tr>
<tr>
<td>4. Tolerance Card</td>
<td>28</td>
</tr>
<tr>
<td>5. Second Option Card</td>
<td>29</td>
</tr>
</tbody>
</table>
THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Institute for Management Science and Engineering

Abstract
of
Serial T-434
21 October 1980

A USER'S MANUAL FOR SENSUMT:
A Penalty Function Computer Program for Solution,
Sensitivity Analysis, and Optimal Value Bound
Calculation in Parametric Nonlinear Programs

by
Anthony V. Fiacco
Abolfazl Ghaemi

This manual is intended to serve as a guide for coding, solving,
and conducting sensitivity and optimal value bound analysis for parametric
nonlinear programming problems with the computer programming model
SENSUMT. The basic sensitivity results and bound calculation techniques
are briefly reviewed and the algorithms implementing them are pre-
sented. A procedure for coding problems for SENSUMT and detailed illus-
trations of the computer output is included. For completeness, the
computer listing and a brief description of all the subroutines comprising
SENSUMT, many of which are taken intact from a previously developed
program SUMT-Version 4, are also provided.

Research Supported by
Contract DAAG 29-79-C-0052
US Army Research Office-Durham
Contract N00014-75-C-0729
Office of Naval Research
Grant ENG-7906104
National Science Foundation
ACKNOWLEDGMENT

This latest version of the SENSUMT program is an experimental program that is still evolving, though it has been used routinely at The George Washington University over the past several years to solve intermediate sized (up to about 100 variables and constraints) NLP problems and to conduct solution sensitivity analysis. A major part of it is the product of many theoretical, computational and software developments contributed by others. Several subroutines have been taken essentially intact from existing codes, most notably from SUMT-Version 4, coded by W. C. Mylander, R. L. Holmes, and G. P. McCormick; the first version of SENSUMT coded by R. L. Armacost and W. C. Mylander; and an extended version of SENSUMT coded subsequently by Armacost. The authors wish especially to acknowledge these contributions, along with the first program to implement some of the techniques used by this general approach for the calculation of sensitivity analysis, coded by B. Causey.
1. Introduction

This is a manual intended to facilitate the use of the computer program SENSUMT. This program is capable of solving general parametric nonlinear programs and conducting a sensitivity analysis using the Sequential Unconstrained Minimization Technique [10]. It has recently been modified to include the calculation of piecewise linear parametric bounds on the optimal value function of classes of nonlinear programming problems for which this function is convex or concave. In addition, it can calculate similar bounds on a class of separable non-convex right-hand side programs, once appropriate convex overestimating and underestimating programs are formulated.

SENSUMT is the outgrowth of a routine [3] which first implemented a penalty function sensitivity approach. This was motivated by results given by Fiacco and McCormick [10] for a particular class of perturbations. The algorithmic procedure for this routine was suggested by Fiacco. It was later refined and recoded by Mylander [12] making use of
several subroutines from the SUMT-Version 4 computer program coded by Mylander, Holmes, and McCormick [13]. These routines were subsequently completely integrated with SUMT-Version 4 by Armacost and Mylander [7]. Armacost [1] further revised the program to implement the generalized version of the sensitivity theory developed by Fiacco [9] for general parameter perturbations. This routine, again following a procedure suggested by Fiacco, was recently modified by Ghaemi to conduct piecewise linear parametric bound calculation on the optimal value function of certain classes of nonlinear programming problems subject to large parametric changes [11].

This latter version of the model, designated "SENSUMT," is compiled at the Center for Academic and Administrative Computing of The George Washington University.

The various sections of this manual are arranged as follows. Section 2 reviews the basic sensitivity results and presents the steps of the algorithm implementing sensitivity calculation via a sequential unconstrained minimization algorithm.

Section 3 briefly reviews the procedure for calculating piecewise linear parametric bounds on the optimal value function of certain classes of parametric nonlinear programming problems. It presents the relevant algorithms that have been implemented.

Section 4 gives the details regarding the procedure for coding the problems for solution, sensitivity analysis, and bound calculation with SENSUMT.

Section 5 provides the codes and corresponding computer solution for two illustrative examples that are taken from [11]. For clarity, an annotated computer listing of the input and output for the first example is also provided.

Section 6 gives a description and listing of the various subroutines comprising SENSUMT. This section is included to make the
2. Sensitivity Analysis in Nonlinear Programming

2.1 Basic Sensitivity Results

The parametric mathematical programming problem considered by Fiacco [9] is of the following general form:

$$\begin{align*}
\text{minimize} & \quad f(x, \varepsilon) \\
\text{subject to} & \quad g_i(x, \varepsilon) \geq 0, \quad i=1, \ldots, m, \\
& \quad h_j(x, \varepsilon) = 0, \quad j=1, \ldots, p,
\end{align*}$$

where $x$ is the usual vector of variables and $\varepsilon$ is a $k$-component vector of numbers called "parameters." It is desired to analyze the behavior of a solution vector $x(\varepsilon)$ and the optimal solution value $f^*(\varepsilon) = f(x(\varepsilon), \varepsilon)$ near some given value of $\varepsilon$. Without loss of generality, assume that the parameter vector of interest is $\varepsilon = 0$.

The Lagrangian for Problem $P(\varepsilon)$ is defined as

$$L(x, u, w, \varepsilon) = f(x, \varepsilon) - \sum_{i=1}^{m} u_i g_i(x, \varepsilon) + \sum_{j=1}^{p} w_j h_j(x, \varepsilon).$$  \tag{2.1}

The sensitivity results are based on the following four assumptions:

$A1$ - The functions defining Problem $P(\varepsilon)$ are twice continuously differentiable in $(x, \varepsilon)$ in a neighborhood of $(x^*, 0)$.

$A2$ - The second order sufficient conditions for a local minimum of Problem $P(0)$ hold at $x^*$ with associated Lagrange multipliers $y^*$ and $w^*$. 

 manual self-contained. With the exception of the subroutines BOUND, PERT, and TRANS, and the new version of program MAIN, the material in this section with a few minor modifications has been taken from [1] and [13].
A3 – The gradients \( \nabla_{x} g_i(x^*,0) \), for all \( i \) such that \( g_i(x^*,0) = 0 \), and \( \nabla_{x} h_j(x^*,0) \), \( j=1,...,p \) are linearly independent.

A4 – Strict complementary slackness holds at \( x^* \) when \( \varepsilon = 0 \) [i.e., \( u_i^* > 0 \) for all \( i \) such that \( g_i(x^*,0) = 0 \)].

Under the above assumptions, Fiacco [9] established the following generalization of Theorem 6 in [10].

**Lemma 2.1** [local characterization of a Kuhn-Tucker triple]: If assumptions A1, A2, A3, and A4 hold for Problem \( P(\varepsilon) \) at \( (x^*,0) \), then

(a) \( x^* \) is a local isolated minimizing point of Problem \( P(0) \) and the associated Lagrange multipliers \( u^* \) and \( w^* \) are unique;

(b) for \( \varepsilon \) in a neighborhood of 0, there exists a unique once continuously differentiable vector function \( y(\varepsilon) = (x(\varepsilon),u(\varepsilon),w(\varepsilon))^T \) satisfying the second order sufficient conditions for a local minimum of Problem \( P(\varepsilon) \) such that \( y(0) = (x^*,u^*,w^*)^T = y^* \), and hence, \( x(\varepsilon) \) is a locally unique, local minimum of Problem \( P(\varepsilon) \) with associated unique Lagrange multipliers \( u(\varepsilon) \) and \( w(\varepsilon) \); and

(c) for \( \varepsilon \) near 0, the set of binding inequalities is unchanged, strict complementary slackness continues to hold, and the binding constraint gradients are linearly independent at \( x(\varepsilon) \).

(d) (Armacost and Fiacco [3]), for \( \varepsilon \) near 0, the gradient of the optimal value function is

\[
\nabla_{\varepsilon} f^*(\varepsilon) = \nabla_{\varepsilon} L(y(\varepsilon),\varepsilon),
\]

(e) which also means that, for \( \varepsilon \) near 0, the Hessian of the optimal value function is

\[
\nabla_{\varepsilon}^2 f^*(\varepsilon) = \nabla_{\varepsilon}^2 L(y(\varepsilon),\varepsilon).
\]
The above results provide a characterization of a local solution of Problem $P(e)$ and its associated optimal Lagrange multipliers near $e=0$. They show that the Kuhn-Tucker triple $y(e)$ is unique and well behaved, under the given conditions. Since $y(e)$ is once differentiable, the partial derivatives of the components of $y(e)$ are well defined. This fact and assumption Al also mean that the functions defining Problem $P(e)$ are once continuously differentiable functions of $e$ along the "solution trajectory" $x(e)$ near $e=0$, and the Lagrangian is a once continuously differentiable function of $e$ along the "Kuhn-Tucker point trajectory." The above results constitute the structure for numerous developments and extensions, many of which have been established by Fiacco [9] and Armacost and Fiacco [2 - 6].

The realization of this theorem for the parametric right hand side problem of special interest in the present study is treated in detail by Armacost and Fiacco [4]. The parametric right hand side problem is the following important realization of $P(e)$:

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \geq e_i, \quad i=1, \ldots, m, \quad R(e) \\
& \quad h_j(x) = e_{j+m}, \quad j=1, \ldots, p.
\end{align*}
\]

The Lagrangian for $R(e)$ is

\[
L(x, u, w) \equiv f(x) - \sum_{i=1}^{m} u_i (g_i(x) - e_i) + \sum_{j=1}^{p} w_j (h_j(x) - e_{j+m}).
\]

As is evident from the Lagrangian, the results (d) and (e) of Lemma 2.1 for Problem $R(e)$ simplify respectively to

\[
\begin{align*}
(d') \quad \nabla_{e} f^*(e) &= \begin{bmatrix} u(e)^{-T} \\ -w(e) \end{bmatrix} \\
(e') \quad \nabla_{e}^2 f^*(e) &= \begin{bmatrix} \nabla_{e} u(L) \\ -\nabla_{e} w(e) \end{bmatrix}.
\end{align*}
\]

(2.4)

(2.5)
Flacco [9] has shown that the class of algorithms based on twice continuously differentiable penalty functions (specifically, using the logarithmic-quadratic loss penalty function) can be used to estimate \( y(\varepsilon) \) and its derivatives in a neighborhood of \( \varepsilon = 0 \), for the general problem \( P(\varepsilon) \). Minimization of the penalty function with penalty parameter \( r \) yields a solution of a perturbation of the Kuhn-Tucker system in a neighborhood of \( (\varepsilon, r) = (0, 0) \). Armacost and Flacco [3] define an optimal value penalty function and obtain first- and second-order sensitivity estimates which converge to the corresponding sensitivities for the optimal value function for Problem \( P(\varepsilon) \).

The logarithmic-quadratic penalty function is

\[
W(x, \varepsilon, r) = f(x, \varepsilon) - r \sum_{i=1}^{m} \ln g_i(x, \varepsilon) + \left(\frac{1}{2r}\right) \sum_{j=1}^{p} h_j^2(x, \varepsilon). \tag{2.6}
\]

**Lemma 2.2** (Flacco [9, Theorem 3.1]): If the assumptions A1 through A4 hold, then in a neighborhood of \( (\varepsilon, r) = (0, 0) \) there exists a unique once continuously differentiable vector function \( y(\varepsilon, r) = [x(\varepsilon, r), u(\varepsilon, r), w(\varepsilon, r)]^T \) satisfying

\[
\nabla_x L(x, u, w, \varepsilon) = 0,
\]

\[
u_i g_i(x, \varepsilon) = r, \quad i = 1, \ldots, m, \tag{2.7}
\]

\[
h_j(x, \varepsilon) = w_j, \quad j = 1, \ldots, p,
\]

with \( y(0, 0) = (x^*, u^*, w^*) \), and such that for any \( (\varepsilon, r) \) near \( (0, 0) \) and \( r > 0 \), \( x(\varepsilon, r) \) is a locally unique unconstrained local minimizing point of \( W(x, \varepsilon, r) \), with \( g_i(x(\varepsilon, r), \varepsilon) > 0 \), \( i = 1, \ldots, m \), and \( \nabla_x^2 W(x, \varepsilon, r) \) positive definite.

The relevance of Equations (2.7) is the fact that, under the given conditions, when \( r = 0 \), they are necessary conditions that must hold at a local solution of \( P(0) \) and, with \( r > 0 \), they are necessary conditions for an unconstrained minimum of \( W(x, \varepsilon, r) \). The latter fact can be made obvious by solving for \( u_i \) and \( w_j \) in (2.7) and obtaining
Thus, if \( y(\varepsilon,r) \) is a solution of (2.7), then

\[
\nabla_x W[x(\varepsilon,r),\varepsilon,r] = \nabla_x L[x(\varepsilon,r),u(\varepsilon,r),w(\varepsilon,r), \varepsilon] = 0 .
\]

This explicit connection between the optimality conditions of local solutions of \( P(\varepsilon) \) and unconstrained minima of \( W(x,\varepsilon,r) \) makes it possible to approximate information characterizing a local solution of \( P(\varepsilon) \) by algorithmic calculations associated with utilizing \( W(x,\varepsilon,r) \) to solve \( P(\varepsilon) \). In particular, differentiating (2.9) with respect to \( \varepsilon \) yields

\[

\nabla^2_w \nabla_x \varepsilon + \nabla^2_x \varepsilon = 0 ,
\]

and using the fact that \( \nabla^2_x W \) is positive definite (a conclusion of Lemma 2.2) yields

\[

\nabla_x \varepsilon = -\nabla^2_x w^{-1} \nabla^2_w \varepsilon ,
\]

(2.10)
evaluated, of course, at \( x(\varepsilon,r) \). Given \( \nabla_x \varepsilon(\varepsilon,r) \), the derivatives of the multipliers, \( \nabla_x u_i(\varepsilon,r) \) and \( \nabla_x w_j(\varepsilon,r) \), can then be calculated by differentiating the last two systems of equations of (2.7) at \( x(\varepsilon,r) \) with respect to \( \varepsilon \).

**Lemma 2.3** (Fiacco [9]): For \( \varepsilon \) in a neighborhood of \( \varepsilon = 0 \), it follows that

(a) \( \lim_{r \to 0^+} y(\varepsilon,r) = y(\varepsilon,0) = y(\varepsilon) \), the Kuhn-Tucker triple characterized in conclusion (b) of Lemma 2.1; and

(b) \( \lim_{r \to 0^+} \nabla \varepsilon y(\varepsilon,r) = \nabla \varepsilon y(\varepsilon,0) = \nabla \varepsilon y(\varepsilon) \).
Under the conditions of Lemma 2.1, $x(\epsilon, r)$ is a locally unique minimizing point of $W(x, \epsilon, r)$. Define the "optimal value penalty function" as

$$W^*(\epsilon, r) = W(x(\epsilon, r), \epsilon, r).$$  \hspace{1cm} (2.11)

Armacost and Fiacco [2, Theorem 4 and Corollary 4.1] have obtained further results useful for estimating the first- and second-order sensitivity of the optimal value function $f^*(\epsilon)$ of Problem $P(\epsilon)$.

**Lemma 2.4** (Armacost and Fiacco [3]): If the assumptions A1 through A4 hold for Problem $P(\epsilon)$, then in a neighborhood of $\epsilon = 0$,

(a) $\lim_{r \to 0^+} W^*(\epsilon, r) = f^*(\epsilon)$;

(b) $\nabla_\epsilon W^*(\epsilon, r) = \nabla_\epsilon L(y(\epsilon, \epsilon))$ at $y = y(\epsilon, r)$;

(c) $\lim_{r \to 0^+} \nabla_{\epsilon} W^*(\epsilon, r) = \nabla_{\epsilon} f^*(\epsilon)$;

(d) $\nabla_{\epsilon}^2 W^*(\epsilon, r) = \nabla_{\epsilon}^2 L(y(\epsilon, r), \epsilon)$; and

(e) $\lim_{r \to 0^+} \nabla_{\epsilon}^2 W^*(\epsilon, r) = \nabla_{\epsilon}^2 f^*(\epsilon)$;

where convergence is component by component in all cases.

Lemmas 2.1 through 2.4 enable us to calculate an estimate of $y(\epsilon)$, $\nabla_\epsilon y(\epsilon)$, $\nabla_\epsilon f^*(\epsilon)$, and $\nabla_{\epsilon}^2 f^*(\epsilon)$ when $\epsilon$ is near 0 and $r$ is near 0, once $y(\epsilon, r)$ is available.

In the next section we briefly present the algorithmic implementation of some of the above results.

2.2 The Algorithm Implementing the Sensitivity Results

The penalty function algorithm SUMT estimates the solution of the general mathematical problem $P(\epsilon)$ by estimating the unconstrained minima
of the penalty function $W(x, e, r)$ at successively decreasing values of
the penalty function parameter $r > 0$. Under conditions weaker than
those assumed here, Fiacco and McCormick [10] have shown that as $r$ ap-
proaches zero, the sequence of the unconstrained minima of $W(x, e, r)$
will approach a solution of $P(e)$. Each unconstrained penalty function
minimization is thus a "subproblem" associated with a particular value
of the penalty function parameter $r$.

The successive steps of the algorithm for first order sensitivity
analysis of the Kuhn-Tucker triple with respect to the $j$th parameter
that are implemented in SENSUMT [1] are listed below. Here the notation
$\bar{x}$ or $\bar{x}(\bar{e})$ denotes the estimate of a solution point of $P(\bar{e})$ calculated
by SUMT for a given value of the penalty function parameter $r$, where
$\bar{e}$ denotes the value of the problem parameter at which this sensitivity
is estimated.

**Algorithm 2.1:**

**Step 1.** Compute a representation of $V^2W = V_x^2W(\bar{x}, \bar{e}, r)^{-1}$ by L-U
decomposition using the SUMT subroutines. If $V^2W$ is
not positive definite, terminate the sensitivity analysis.

**Step 2.** Estimate $\partial(V_x^T)/\partial e_j$ using the central differencing
formula

$$\partial(V_x^T)/\partial e_j = \left(\frac{1}{2\Delta}\right)(V_x^T(\bar{x}, \bar{e} + \Delta e_j, r)^T - V_x^T(\bar{x}, \bar{e} - \Delta e_j, r)^T),$$

where $\Delta$ is the differencing interval and $e_j$ is the $j$th
unit vector.

**Step 3.** Calculate

$$\partial\bar{x}(\bar{e})/\partial e_j = -V^2W^{-1} \partial(V_x^T)/\partial e_j.$$

**Step 4.** Estimate $V_e g_1(\bar{x}, \bar{e})$ and $V_e h_1(\bar{x}, \bar{e})$ using

$$\partial g_1(\bar{x}, \bar{e})/\partial e_j = \left(\frac{1}{2\Delta}\right)(g_1(\bar{x}, \bar{e} + \Delta e_j) - g_1(\bar{x}, \bar{e} - \Delta e_j)),$$

and
Step 5. Estimate the components of \( \nabla_v u(\varepsilon) \) for \( i=1,\ldots,m \) using
\[
\frac{\partial u_i(\varepsilon)}{\partial \varepsilon_j} \approx \left(-\frac{1}{\Delta}\right)\left(\frac{u_i(\bar{x},\varepsilon+\Delta\varepsilon_j) - u_i(\bar{x},\varepsilon-\Delta\varepsilon_j)}{2\Delta}\right).
\]

Step 6. Estimate the components of \( \nabla_v w(\varepsilon) \) for \( i=1,\ldots,p \) using
\[
\frac{\partial w_i(\varepsilon)}{\partial \varepsilon_j} \approx \left(\frac{1}{\Delta}\right)\left(\frac{w_i(\bar{x},\varepsilon+\Delta\varepsilon_j) - w_i(\bar{x},\varepsilon-\Delta\varepsilon_j)}{2\Delta}\right).
\]

There are two methods for estimating \( \nabla_v f^*(\varepsilon) \), the first using
\[
\nabla_v f^* = \nabla_v x + \nabla_v f,
\]
with \( \nabla_v x \) obtained from Step 3, and the second method using the gradient of the penalty function \( W \), or equivalently, the Lagrangian taken with respect to the parameters [Lemma 2.1, conclusion (d); Lemma 2.4, conclusion (b)]. Both are incorporated in the computer program but used for different purposes. The former method gives the most accurate estimate of \( \nabla_v f^*(\varepsilon) \) and is summarized in Steps 7 and 8.

Step 7. Estimate the components of \( \nabla_v f(\bar{x},\varepsilon) \) using the central differencing formula
\[
\frac{\partial f(\bar{x},\varepsilon)}{\partial \varepsilon_j} \approx \left(\frac{1}{2\Delta}\right)\left(f(\bar{x},\varepsilon+\Delta\varepsilon_j) - f(\bar{x},\varepsilon-\Delta\varepsilon_j)\right).
\]

Calculate an estimate of the components of \( \nabla_v f^*(\varepsilon) \) using the results of Steps 3 and 7 as
\[
\frac{\partial f^*(\varepsilon)}{\partial \varepsilon_j} \approx \nabla_v f(\bar{x},\varepsilon) \frac{\partial \bar{x}(\varepsilon)}{\partial \varepsilon_j} + \frac{\partial f(\bar{x},\varepsilon)}{\partial \varepsilon_j}.
\]

The second method, using the gradient of the Lagrangian to estimate \( \nabla_v f^*(\varepsilon) \), is computationally less expensive and is used to obtain rough estimates that single out the more crucial parameters for further analysis. This approximation is calculated as follows.
Step 9. Estimate the components of $V \ell f^*(\ell)$ using the results of Steps 4 and 7 as

$$
\frac{\partial f^*(\ell)}{\partial c_j} \approx \frac{\partial f(\ell,\ell)}{\partial c_j} - \sum_{i=1}^{m} u_i(\ell,r) \frac{\partial g_i(x,\ell)}{\partial c_j} + \sum_{i=1}^{p} w_i(\ell,r) \frac{\partial h_i(x,\ell)}{\partial c_j}.
$$

3. Parametric Optimal Value Bounds

In this section we briefly review the procedure for calculating piecewise linear parametric upper and lower bounds on the optimal value function of convex problems with right hand side perturbations of the form $CR(\ell)$. It is important to note that the given technique is not only applicable to problems of this form, but also to any class of problems which have convex or concave optimal value functions. In addition, we show how the above technique can be extended to calculate similar bounds on the optimal value function of separable nonconvex right hand side perturbation problems $SR(\ell)$, once their (computable) over- and underestimating problems are available. For a more detailed treatment, readers should refer to [11].

3.1 Parametric Bounds on the Optimal Value Function of Convex Right Hand Side Problems $CR(\ell)$

Consider the following right hand side parametric programming problem $R(\ell)$, discussed in Section 2:

$$
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \geq \ell_i, \quad i=1,m \\
& \quad h_j(x) = \ell_i, \quad j=m+1,p
\end{align*}
$$

where $x \in \mathbb{R}^n$, $f$, $g$, and $h : \mathbb{R}^n + \ell^1$, and $c^2$ and $c \in E^p$. If $f(x)$ and $-g_i(x)$, $i=1,m$, are convex and $h_j(x)$, $j=m+1,p$ are linear, then
the problem \( R(\varepsilon) \) is convex and will be designated by \( CR(\varepsilon) \). It is well known that \( f^*(\varepsilon) \), the optimal value function of the problem \( CR(\varepsilon) \), is a convex function of \( \varepsilon \).

The convexity of the optimal value function \( f^*(\varepsilon) \) of the problem \( CR(\varepsilon) \) enables one to calculate parametric upper and lower bounds on this function when any of the problem parameters is radically perturbed. This of course requires the solution and corresponding optimal value function sensitivity information for both perturbed and unperturbed problems. Before delving into the calculation procedure of these bounds, we must remind ourselves of the following two basic properties of convex functions.

(i) Any line connecting two points on the graph of a convex function does not underestimate that function between the points.

(ii) Any line tangent to the graph of a convex function does not overestimate that function.

These two properties lend themselves in a natural way to the calculation of parametric bounds on the optimal value function of the problem \( CR(\varepsilon) \) under large perturbations of any of the problem parameters, say \( \varepsilon_i \).

3.2 Algorithm for Calculation of Bounds on \( f^*(\varepsilon) \) of the Problem \( CR(\varepsilon) \)

In the following we list the successive steps required in calculation of bounds on \( f^*(\varepsilon) \), the optimal value function of \( CR(\varepsilon) \), as a function of \( \varepsilon_i \), when \( \varepsilon_i \) is perturbed from \( \bar{\varepsilon}_{i1} \) to \( \bar{\varepsilon}_{i2} \) while the remaining parameters are fixed at their base values.

Algorithm 3.1:

Step 1. Solve the unperturbed problem and obtain \( f^*(\varepsilon_i) \) and \( df^*(\varepsilon_i)/d\varepsilon_i \) at \( \varepsilon_i = \bar{\varepsilon}_{i1} \). Note that
\[
\begin{align*}
\frac{df^*(\varepsilon_1)}{d\varepsilon_1} &= \begin{cases} 
  u_1(\varepsilon_1), & i=1, \ldots, m \\
  -\omega_1(\varepsilon_1), & i=m+1, p.
\end{cases} 
\end{align*}
\]

(2.4)

(2.5)

Step 2. Re-solve the problem and obtain \( f^*(\varepsilon_1) \) and \( df^*(\varepsilon_1)/d\varepsilon_1 \) at \( \varepsilon_1 = \bar{\varepsilon}_{i2} \).

Step 3. Derive the equation of the line passing through the points \( (\varepsilon_1 = \bar{\varepsilon}_{i1}, f^*(\varepsilon_1) = f^*(\bar{\varepsilon}_{i1})) \) and \( (\varepsilon_1 = \bar{\varepsilon}_{i2}, f^*(\varepsilon_1) = f^*(\bar{\varepsilon}_{i2})) \). This line provides a parametric upper bound \( \bar{f}^*(\varepsilon_1) \) for \( f^*(\varepsilon_1) \) as a function of \( \varepsilon_1 \in [\bar{\varepsilon}_{i1}, \bar{\varepsilon}_{i2}] \).

Step 4. Derive the equation of the tangent lines to \( f^*(\varepsilon_1) \) at the above two points with the slopes

\[
\left. \frac{df^*(\varepsilon_1)}{d\varepsilon_1} \right|_{\varepsilon_1 = \bar{\varepsilon}_{i1}} \quad \text{and} \quad \left. \frac{df^*(\varepsilon_1)}{d\varepsilon_1} \right|_{\varepsilon_1 = \bar{\varepsilon}_{i2}}
\]

calculated in Steps 1 and 2, respectively. These two lines provide a parametric lower bound \( \underline{f}^*(\varepsilon_1) \) for \( f^*(\varepsilon_1) \) as a function of \( \varepsilon_1 \in [\bar{\varepsilon}_{i1}, \bar{\varepsilon}_{i2}] \).

The lines obtained in Steps 3 and 4 form a triangle which encloses the optimal value function \( f^*(\varepsilon_1) \) over the given range of \( \varepsilon_1 \).

An estimate of \( f^*(\varepsilon_1) \) can also be made by fitting a convex function that passes through the points given in Step 3 and having the corresponding slopes at these points obtained in Steps 1 and 2.

It is clear that the fundamental requirement in using the above algorithm is that the optimal value function be convex and differentiable at least at \( \varepsilon_1 = \bar{\varepsilon}_{i1} \) and \( \varepsilon_1 = \bar{\varepsilon}_{i2} \). As it was given in Section 2, Fiacco, under the assumptions A1 - A4 of Lemma 2.1, has established the differentiability of the optimal value function in a neighborhood of \( \varepsilon = 0 \) for the general parametric programming problem \( P(\varepsilon) \). Thus, not
only Problem CR(ε), but all classes of parametric nonlinear programming problems possessing a convex optimal value function and, in particular, meeting assumptions A1 - A4 at $\epsilon_1 = c_{11}$ and $\epsilon_1 = c_{12}$, can be analyzed for bounds according to Algorithm 3.1.

3.3 Parametric Bounds on the Optimal Value Function of the Problem SR(ε)

In this section we show how Algorithm 3.1 can be extended to calculate piecewise linear parametric upper and lower bounds on the optimal value function of a separable nonconvex right hand side perturbation problem, given the over- and underestimating problems.

3.3.1 Separable right hand side perturbation problems.

The separable nonconvex problem addressed here has the following structure:

$$\begin{align*}
\text{minimize} & \quad f(x) = \sum_{j=1}^{n} f_j(x_j) \\
\text{subject to} & \quad g_i(x) = \sum_{j=1}^{n} g_{ij}(x_j) \geq \epsilon_i, \ i=1, \ldots, m \\
& \quad L_j \leq x_j \leq U_j, \ j=1, \ldots, n,
\end{align*}$$

where each $f_j(x_j)$ and $g_{ij}(x_j)$ is a function of a single variable $x_j$ and is differentiable; $L_j$ and $U_j$ are lower and upper bounds on the variable $x_j$, respectively. Note that the discussion that follows is also valid when Problem SR(ε) involves linear equality constraints. Let

$$\begin{align*}
G & = \left\{ x : g_1(x) = \sum_{j=1}^{n} g_{1j}(x_j) \geq \epsilon_1, \ j=1, \ldots, m \right\}, \\
B_j & = \left\{ x_j : L_j \leq x_j \leq U_j \right\}, \ j=1, \ldots, n, \\
B & = \{ x : B_j \}, \ j=1, \ldots, n.
\end{align*}$$

- 14 -
and

\[ S = G \cap B. \]

The set \( G \), and therefore set \( S \), depends on \( \varepsilon \). Thus the problem \( \text{SR}(\varepsilon) \) will read as follows:

\[
\begin{align*}
\text{minimize } & \quad f(x) = \sum_{j=1}^{n} f_j(x_j) \\
\text{subject to } & \quad x \in B \cap G = \bar{S} \subseteq \mathbb{R}^n.
\end{align*}
\]

3.3.2 Convex underestimating problem.

Let \( \tilde{f}_j(x_j) \) be the convex envelope of \( f_j(x_j) \), \( j=1,...,m \) and \( \tilde{g}_{ij}(x_j) \) be the concave envelope of \( g_{ij}(x_j) \), \( i=1,...,n, j=1,...,m \) over the interval \( B_j \). Thus, for \( x_j \in B_j \),

\[
\begin{align*}
\tilde{f}_j(x_j) & \leq f_j(x_j) \\
\tilde{g}_{ij}(x_j) & \geq g_{ij}(x_j), \quad i=1,...,n; \quad j=1,...,m.
\end{align*}
\]

**Definition:** Define the problem \( \text{CSR}(\varepsilon) \) as follows:

\[
\begin{align*}
\text{minimize } & \quad \tilde{f}(x) = \sum_{j=1}^{n} \tilde{f}_j(x_j) \\
\text{subject to } & \quad x \in G \cap B = \bar{S},
\end{align*}
\]

where

\[
\bar{S} = \left\{ x : \tilde{g}_i(x) = \sum_{j=1}^{n} \tilde{g}_{ij}(x) \geq \varepsilon_i , \quad i=1,...,m \right\}.
\]

It follows easily that problem \( \text{CSR}(\varepsilon) \) is a separable convex RHS perturbation problem whose optimal value function underestimates the global optimal value function of the problem \( \text{SR}(\varepsilon) \).

Thus the basic ingredients required to formulate the convex underestimating problem \( \text{CSR}(\varepsilon) \) are the convex envelope of each term in the
objective function and the concave envelope of each term in the constraints of the problem \( SR(\varepsilon) \) over the rectangular polytope \( B \).

In the next section we will discuss the formulation of a convex overestimating problem \( CSR(\varepsilon) \) of the problem \( SR(\varepsilon) \).

3.3.3 Convex overestimating problem.

Let \( f_j(x_j) \) be a convex function which does not underestimate \( f_j(x_j) \), \( j=1,\ldots,n \), and \( \tilde{g}_{ij}(x_j) \) be a concave function which does not overestimate \( g_{ij}(x_j) \), \( j=1,\ldots,n \), \( i=1,\ldots,m \), over the interval \( B_j \).

Thus, for \( x_j \in B_j \),

\[
\tilde{f}_j(x_j) > f_j(x_j) \quad \text{and} \quad \tilde{g}_{ij}(x_j) \leq g_{ij}(x_j), \quad i=1,\ldots,m; \quad j=1,\ldots,n.
\]

Henceforth in discussion, \( \tilde{f}_j(x_j) \) will be abbreviated as convex "nuf" (nonunderestimating function) of the single variable function \( f_j(x_j) \), and \( \tilde{g}_{ij}(x_j) \) will be abbreviated as concave "nof" (nonoverestimating function) of \( g_{ij}(x_j) \) over the interval \( B_j \), \( i=1,\ldots,m \), \( j=1,\ldots,n \).

Definition: Let us define the problem \( CSR(\varepsilon) \) as follows:

\[
\text{minimize } f(x) = \sum_{j=1}^{n} \tilde{f}_j(x_j) \\
\text{subject to } x \in (\tilde{S} \cap B) \equiv \tilde{S},
\]

where

\[
\tilde{S} = \left\{ x : \tilde{g}_i(x) = \sum_{j=1}^{n} \tilde{g}_{ij}(x_j) \geq \varepsilon_i , \quad i=1,\ldots,m \right\}.
\]

It is easy to show that problem \( CSR(\varepsilon) \) is a separable convex RHS problem whose optimal value function underestimates the global optimal value function of the problem \( SR(\varepsilon) \).
Thus, in order to formulate an overestimating convex problem 
CSR(\(\epsilon\)) of the problem SR(\(\epsilon\)), one must calculate convex nuf, \(f_j(x_j)\),
for each component of \(f(x)\) and concave nof, \(\tilde{g}_{ij}(x_j)\), for each com-
ponent of \(g_i(x)\), \(i=1,\ldots,m\), over the interval \(B_j\). It must be noted
that convex nuf and concave nof of a given function, unlike its enve-
lopes, are not unique. Thus the overestimating problem CSR(\(\epsilon\)) of prob-
lem SR(\(\epsilon\)) is not unique. For further treatment of convex nuf and con-
cave nof of single variable functions, readers may refer to [11].

An important point to be noted here is that formulation of the
convex problems CSR(\(\epsilon\)) and CSR(\(\epsilon\)) enable one to calculate parametric
upper and lower bounds on the (global) optimal value function of the
separable nonconvex programming problem SR(\(\epsilon\)).

3.4 Algorithm for Calculation of Bounds on
the (Global) Optimal Value Function
of the Problem SR(\(\epsilon\))

In the following an algorithm is presented for calculation of up-
per and lower bounds on the (global) optimal value function of the prob-
lem SR(\(\epsilon\)) as a function of each RHS parameter \(\epsilon_i\), where \(\epsilon_i \in [\bar{\epsilon}_{i1}, \bar{\epsilon}_{i2}]\).
Implicit in this algorithm is the assumption that the overestimating
problem CSR(\(\epsilon\)) has a nonempty interior, which is a basic requirement of
almost any nonlinear programming algorithm.

Algorithm 3.2:
Step 1: Calculate \(f_j(x_j)\), a convex nuf for each \(f_j(x_j)\), \(j=1,\ldots,n\),
and \(\tilde{g}_{ij}(x_j)\), a concave nof for each \(g_{ij}(x_j)\), \(i=1,\ldots,m\),
j=1,\ldots,n, in closed interval \(B_j\), and construct the cor-
responding overestimating problem CSR(\(\epsilon\)).

Step 2: Solve the problem CSR(\(\epsilon\)) at \(\epsilon_i = \bar{\epsilon}_{i1}\) and \(\epsilon_i = \bar{\epsilon}_{i2}\) and ob-
tain \(f^*(\epsilon_i)\) at these values of \(\epsilon_i\).
Step 3: Derive the equation for the line passing through the points 
\((\varepsilon_1 = \tilde{\varepsilon}_{11}, \quad f^*(\varepsilon_1) = f^*(\tilde{\varepsilon}_{11}))\) and \((\varepsilon_1 = \tilde{\varepsilon}_{12}, \quad f^*(\varepsilon_1) = f^*(\tilde{\varepsilon}_{12}))\). This line provides a parametric upper bound \(\tilde{f}^*(\varepsilon_1)\) on the (global) optimal value function \(f^*(\varepsilon_1)\) of the problem \(SR(\varepsilon)\) as a function of \(\varepsilon_1\) over the interval \([\tilde{\varepsilon}_{11}, \tilde{\varepsilon}_{12}]\).

Step 4: Calculate \(f_j(x_j)\), the convex envelope for each \(f_j(x_j)\), \(i = 1, \ldots, n\) and \(\tilde{g}_{ij}(x_j)\), the concave envelope for each \(g_{ij}(x_j)\), \(i = 1, \ldots, m\), \(j = 1, \ldots, n\) in closed interval \(B_j\), and construct the corresponding underestimating problem \(CSR(\varepsilon)\).

Step 5: Solve the problem \(CSR(\varepsilon)\) at \(\varepsilon_1 = \tilde{\varepsilon}_{11}\) and \(\varepsilon_1 = \tilde{\varepsilon}_{12}\) and obtain \(f^*(\varepsilon_1)\) and \(\tilde{u}_i(\varepsilon_i) = df^*(\varepsilon_i)/d\varepsilon_i\) at these values of \(\varepsilon_1\).

Step 6: Derive the equation for the tangent lines to \(f^*(\varepsilon_1)\) at \(\varepsilon_1 = \tilde{\varepsilon}_{11}\) and \(\varepsilon_1 = \tilde{\varepsilon}_{12}\), with respective slopes of \(\tilde{u}_1(\varepsilon_{11})\) and \(\tilde{u}_1(\varepsilon_{12})\) derived in Step 5. These two lines over the interval \([\tilde{\varepsilon}_{11}, \tilde{\varepsilon}_{12}]\) jointly provide a parametric lower bound \(f^*(\varepsilon_1)\) on the (global) optimal value function \(f^*(\varepsilon_1)\) of the problem \(SR(\varepsilon)\) as a function of \(\varepsilon_1\).

The implicit assumption in Steps 5 and 6 of this algorithm is that assumptions A1 - A4 of Lemma 2.1 hold at the solution points when \(\varepsilon_1\) takes values of \(\tilde{\varepsilon}_{11}\) and \(\tilde{\varepsilon}_{12}\).

Before delving into the next section, we must point out that if over- and underestimating problems with concave optimal value functions could be formulated, then the algorithm above, with minor alterations, can be used to calculate the desired bounds. In this instance Step 6, when applied to the overestimating problem, will provide an upper bound.
while Step 3, when applied to the underestimating problem, will provide a lower bound on the optimal value function. For calculation of envelopes, convex nuf and concave nof of single variable functions, readers may refer to [11]. The entries in Table 1 summarize the concept of convex nuf and concave nof for a variety of single variable functions.

4. **SENSUMT Input Specifications**

The coding procedure for nonlinear programming problems for solution, sensitivity analysis, and bound calculation using SENSUMT closely follows the coding procedure for SUMT-Version 4 [13]. In order to code a given problem for the above calculations, one must have (i) a user-supplied subroutine, and (ii) user's information cards.

4.1 User-supplied Subroutines

User-supplied subroutines will generally consist of four subroutines called READIN, RESTNT, GRAD1 and MATRIX. These subroutines are the only subroutines for SENSUMT that are problem dependent. The communication between these subroutines and the rest of the subroutines of SENSUMT is mostly through the COMMON blocks, but partly via their arguments. Each of the user-supplied subroutines must contain the double precision card

```implied real*8(a-h,o-z).```

The subroutines RESTNT, GRAD1, and MATRIX must contain the COMMON card

```common/share/x(45), del(45), a(45,45), n, m, mn, npl, nm1.```  

Also, depending on the problem being solved, the following COMMON card may be required in some or all of the user subroutines:

```common/par(45), par(45), npar, isumps.```  

When bound analysis is desired, subroutine READIN must contain the COMMON card

```common/abg/ly, lz, per(45).```
### TABLE 1

**Convex nuf and Concave nuf of the Single Variable Function $T(z)$ Over the Closed Interval $a \leq z \leq b$**

<table>
<thead>
<tr>
<th>$T(z)$</th>
<th>$\hat{T}(z)$ [convex nuf of $T(z)$]</th>
<th>$\tilde{T}(z)$ [concave nuf of $T(z)$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \leq z \leq b$ Graph</td>
<td>$a \leq z \leq b$ Graph</td>
<td>$a \leq z \leq b$ Graph</td>
</tr>
<tr>
<td>Convex</td>
<td>$\hat{T}(z) = T(z)$</td>
<td>$\tilde{T}(z) = \text{any line not overestimating } T(z)$</td>
</tr>
<tr>
<td>Concave</td>
<td>$\hat{T}(z) = \text{any line not underestimating } T(z)$</td>
<td>$\tilde{T}(z) = T(z)$</td>
</tr>
<tr>
<td>Neither convex nor concave</td>
<td>$\hat{T}(z) = \text{any line not underestimating } T(z)$</td>
<td>$\tilde{T}(z) = \text{any line not overestimating } T(z)$</td>
</tr>
</tbody>
</table>
In the following section we discuss the purpose and coding procedure of each of these user-supplied subroutines. The general description and listing of the rest of the subroutines comprising SENSUMT will be given in Section 6.

4.1.1 READIN

This subroutine allows the user to read in the data which is necessary for evaluation of the problem functions and convey it, through the COMMON blocks, to the related subroutines. The problem parameters for which sensitivity and bound calculation are desired must also be defined and initialized in this subroutine. The initial values of the parameters must be placed in array PAR. The corresponding perturbation values for bound calculation must be placed in array PER. NPAR in this subroutine must be initialized to the total number of problem parameters on which sensitivity is desired. Notice that the presence of array PER in this subroutine will trigger SENSUMT to calculate the desired bound calculation. READIN may also be used to read in and print out any pertinent information of the user's choice, such as problem title, date, and so on. This subroutine is called only once for each problem being solved. For clarity, see Example 1 and its code in Figure 2. Any input data read by subroutine READIN must be placed immediately after the first option card.

4.1.2 RESTNT (IN, VAL)

This subroutine is used to read in the problem functions. For IN=0, set VAL=f(x) and for IN≠0 set

\[ \text{VAL} = g_{IN}(x), \quad IN=1,2,...,m \]
\[ \text{VAL} = h_{IN}(x), \quad IN=m+1,...,n+p \]

For clarity see Example 1 and Figure 2.
4.1.3 GRADI (IN)

This subroutine is used to read in the gradients of the problem functions. For IN=0 set

\[ \text{DEL}(J) = \frac{\partial f(x)}{\partial x_J}, \quad J=1,2,...,n \]

and for IN≠0 set

\[ \text{DEL}(J) = \begin{cases} \frac{\partial g_{\text{IN}}(x)}{\partial x_J}, & J=1,2,...,n \\ \frac{\partial h_{\text{IN}}(x)}{\partial x_J}, & J=1,2,...,n \end{cases} \]

For an illustration, see Example 1 and Figure 2.

SEPSUMT can internally compute numerical approximations for some or all of the gradients. To utilize this option, instead of coding the gradients code the statements

CALL DIFF1(IN)
RETURN

for any desired value of the variable IN (the index of the problem functions, with IN=0 corresponding to the objective function \( f(x) \)). Because of the computational effort involved in numerical differencing, this option may be prohibitive for large problems.

4.1.4 MATRIX (IN,K)

This subroutine reads in the upper triangle and diagonal elements of the Hessian matrix of the problem functions. For IN=0, set

\[ A(I,J) = \frac{\partial^2 f(x)}{\partial x_I \partial x_J}, \quad I=1,2,...,n; \quad J=1,2,...,n; \quad \text{and} \quad I < J \]

For IN≠0 set

\[ A(I,J) = \begin{cases} \frac{\partial^2 g_{\text{IN}}(x)}{\partial x_I \partial x_J}, & IN=1,2,...,m \\ \frac{\partial^2 h_{\text{IN}}(x)}{\partial x_I \partial x_J}, & IN=m+1,...,p \end{cases} \]

where I=1,2,...,n; J=1,2,...,n; and I ≤ J. Before the call is made to this subroutine, all entries of the matrix \( A(I,J) \) are set to zero.
The second argument $K$ of the subroutine MATRIX is provided so that the user may communicate to SENSUMT that the Hessian of a problem function is identically zero. Set $K=1$ if the second partial derivatives of the $IN$th constraint are zero ($IN=0$ corresponds to the objective function). For an illustration, see Example 1 and Figure 2.

SENSUMT can internally compute numerical approximations for some or all of the Hessian matrices. To utilize this option, instead of coding the Hessian matrices, code the statements

```fortran
CALL DIFFZ(IN)
RETURN
```

for any desired value of the variable IN (the index of the problem function, with $IN=0$ designating the objective function). Although this option can be used in calculating an optimal solution, the current sensitivity routines require explicit coding of the closed form of the Hessian matrices. Thus, this option cannot be used for sensitivity and bound calculations. Again, because of the computational effort involved in numerical differencing, use of this option is not advised for large problems.

### 4.2 User's Information Cards

The other inputs required by SENSUMT are the user's information cards, i.e.,

- PARAMETER CARD
- INITIAL VECTOR CARD(S)
- FIRST OPTION CARD
- TOLERANCE CARD
- SECOND OPTION CARD.

In the following we specify the type of information, along with the corresponding format, name, and description, which must be provided by the user to SENSUMT via the above cards. All of this information is read by the program MAIN, which sets up the SENSUMT algorithm to solve and
analyze the problem under study. Tables 2, 3, and 4, which follow, are taken from [13].

4.2.1 Parameter card.

Input specifications of the parameter card are shown in Table 2.

<table>
<thead>
<tr>
<th>Columns</th>
<th>Format</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-12</td>
<td>E12.0</td>
<td>EPSI (e)</td>
<td>Tolerance used to decide if an unconstrained minimum has been achieved for each subproblem [see Option 9]</td>
</tr>
<tr>
<td>13-24</td>
<td>E12.0</td>
<td>RHOIN (r₁)</td>
<td>Possible initial value of r (often set at 1.0) [see Option 1]</td>
</tr>
<tr>
<td>25-36</td>
<td>E12.0</td>
<td>THETAO (θ₀)</td>
<td>Tolerance used to decide if the solution to the NLP problem P(ε) has been approximated [see Option 5]</td>
</tr>
<tr>
<td>37-48</td>
<td>E12.0</td>
<td>RATIO (c)</td>
<td>Parameter (&gt;1) used to compute consecutive values of r; rᵢ₊₁ = rᵢ/c (often set at 16.0)</td>
</tr>
<tr>
<td>49-60</td>
<td>E12.0</td>
<td>TMMAX</td>
<td>Maximum amount of time for solving problem (in seconds)</td>
</tr>
<tr>
<td>61-64</td>
<td>I4</td>
<td>M</td>
<td>Number (integer) of nontrivial constraints [see Option 2] (M+MZ) ≤ 200*</td>
</tr>
<tr>
<td>65-68</td>
<td>I4</td>
<td>N</td>
<td>Number (integer) of variables, N ≤ 45</td>
</tr>
<tr>
<td>69-72</td>
<td>I4</td>
<td>MZ</td>
<td>Number (integer) of equality constraints</td>
</tr>
</tbody>
</table>

*The limits on M+MZ and N are governed by the size of the arrays in SENSUMT.

4.2.2 Initial vector card(s).

The cards designating the initial starting point immediately follow the parameter card. There are six components per card, requiring n/6 cards for the initial vector. Each card has the format 6E12.6.
4.2.3 First option card.

The input specifications for the first option card are shown in Table 3. This card must immediately follow the user-supplied subroutines.

4.2.4 Tolerance card.

The input specifications for the tolerance card are given in Table 4. This card must immediately follow the first option card.

4.2.5 Second option card.

The input specifications for the second option card, which immediately follows the tolerance card, are given in Table 5.

Figure 1 shows the arrangement of the data together with JOB and JCL cards in a coded deck.
<table>
<thead>
<tr>
<th>Option</th>
<th>Column</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>=1</td>
<td>The value for ( r ) is made by finding an approximate solution ( \min { [VW(x^0, r)][V^2W(x^0, r)]^{-1}VW(x^0, r) } ) which is a good approximation only when ( x^0 ) is close to the boundary (i.e., for some ( i ), ( g_i(x) ) is close to zero) or when ( V^2f(x^0) = 0 ) and when ( MZ = 0 ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>( r_1 ) is given by formula 8.65 [10, p. 191] (can only be used when ( MZ = 0 )).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>( r_1 = RH0IN ) (see parameter card).</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>=1</td>
<td>The requirements (trivial constraints) that ( x_i \geq 0 ) for ( i = 1, \ldots, n ) are to be automatically included in the problem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>The only constraints on the problem are those inputted by the user.</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>=1</td>
<td>Standard printout (this includes a call to OUTPUT after the solution of every subproblem). Also the estimates of the &quot;Lagrange multipliers&quot; and first and second order solution estimates are printed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>For additional printout (includes standard printout and every intermediate point, gradient of ( P ) and the vector ( S )).</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>=1</td>
<td>Final convergence is determined on the basis of current solution to the subproblem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Final convergence is determined on the basis of the first order estimates. The first order estimate of the solution vector must be close to feasible. See below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>Final convergence is determined on the basis of the second order estimates. The second order estimate of the solution vector must be close to feasible before the convergence check is made. If ( \bar{x} ) is a solution</td>
</tr>
<tr>
<td>Option</td>
<td>Column</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>=1</td>
<td>The convergence criterion determining the NLP problem has been solved (only use =1, when NT4#1). The convergence criterion determining the NLP problem has been solved (only use =1, when NT4#1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>estimate it is considered close to being feasible if ( g_i(x) + \theta_0 \geq 0 ), ( i=1,2,\ldots,m ), where ( \theta_0 ) is defined on the parameter card. estimate it is considered close to being feasible if ( g_i(x) + \theta_0 \geq 0 ), ( i=1,2,\ldots,m ), where ( \theta_0 ) is defined on the parameter card.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Quit when ( \frac{G(x(r_k), \mu(r_k), \lambda(r_k))}{G(x(r_k), \mu(r_k), \lambda(r_k))} &lt; \theta_0 ) Quit when ( \frac{G(x(r_k), \mu(r_k), \lambda(r_k))}{G(x(r_k), \mu(r_k), \lambda(r_k))} &lt; \theta_0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>Quit when (</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>first order estimate of ( v_0 ) ( \frac{G(x(r_k), \mu(r_k), \lambda(r_k))}{G(x(r_k), \mu(r_k), \lambda(r_k))} - 1 &lt; \theta_0 ) first order estimate of ( v_0 ) ( \frac{G(x(r_k), \mu(r_k), \lambda(r_k))}{G(x(r_k), \mu(r_k), \lambda(r_k))} - 1 &lt; \theta_0 )</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>=1</td>
<td>After final convergence the program reads in new data and solves the next problem. After final convergence the program reads in new data and solves the next problem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>After final convergence has been determined a call to PUNCH is made before proceeding on to the next problem. After final convergence has been determined a call to PUNCH is made before proceeding on to the next problem.</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>=1</td>
<td>First move after a minimum to a subproblem is achieved First move after a minimum to a subproblem is achieved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>No extrapolation. No extrapolation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>Extrapolate through last two minima. Extrapolate through last two minima.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Extrapolate through last three minima. Extrapolate through last three minima.</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
<td></td>
<td>Not used. Not used.</td>
</tr>
<tr>
<td>9</td>
<td>63</td>
<td>=1</td>
<td>Subproblem convergence criterion, or when to stop minimizing P function for fixed value of ( r ) (see parameter card). Subproblem convergence criterion, or when to stop minimizing P function for fixed value of ( r ) (see parameter card).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Quit when ( \left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Quit when ( \left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( \left</td>
</tr>
</tbody>
</table>

\[ \frac{W(x^{i-1}) - W(x^i)}{5} \]
Table 3--continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Column</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^a</td>
<td>70</td>
<td>=3</td>
<td>Quit when $</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1</td>
<td>At least one nonlinear constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Linear constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>Linear constraints and linear objective function (i.e., a linear programming problem)</td>
</tr>
</tbody>
</table>

^aWhen option 10=3, MATRIX (the user subroutine supplying the second partial derivatives) will not be called, and when option 10=2 it will be called only to get the second partials of f(x).

TABLE 4
TOLERANCE CARD

<table>
<thead>
<tr>
<th>Columns</th>
<th>Format</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-12</td>
<td>E12.6</td>
<td>XEP1</td>
<td>If some first or second derivatives are to be gotten by numerical differencing (see addition option card) this is the value used to compute them. See the description of DIFF1. (Usually setting XEP1 equal to .0001 is satisfactory, although a good value is dependent on the scaling of the problem.)</td>
</tr>
<tr>
<td>13-24</td>
<td>E12.6</td>
<td>XEP2</td>
<td>When minimizing the $W$ function for a given value of $r$ (RHO) the value of $W$ must decrease by an amount exceeding XEP2 for each iteration after the first. If it does not, then the code prints out the message &quot;apparently roundoff errors prevent a more accurate determination of the minimum of this subproblem,&quot; and it is assumed a minimum has been found. (Usually we set XEP2 equal to 0.)</td>
</tr>
</tbody>
</table>
### TABLE 5
SECOND OPTION CARD

<table>
<thead>
<tr>
<th>Option</th>
<th>Column</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>=1</td>
<td>Solve problems without checking derivatives</td>
</tr>
<tr>
<td>(normally set to 4)</td>
<td></td>
<td>=2</td>
<td>Solve problem after checking only first derivatives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>Do not solve problem after checking only first derivatives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=4</td>
<td>Solve problem after checking first and second derivatives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=5</td>
<td>Do not solve problem after checking first and second derivatives</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>=1</td>
<td>The method for minimizing the unconstrained penalty function is to be the generalized Newton-Raphson method as modified to handle indefinite Hessian matrices. This method requires function values, first and second derivatives.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Same as 1, except that when an &quot;orthogonal move&quot; is made because of an indefinite Hessian matrix, $-\nabla P$ is added to the orthogonal move vector. Steepest descent is used to minimize $P$-function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>The method for minimizing the unconstrained penalty function is McCormick's modification of the Fletcher-Powell method as reported in [10]. This requires function values and first derivatives.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=4</td>
<td>Do not conduct a sensitivity analysis.</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>=0</td>
<td>Conduct a sensitivity analysis at the final subproblem with a fixed value for the differencing interval.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1</td>
<td>Conduct a sensitivity analysis at each subproblem along the minimizing trajectory with the value of the differencing interval depending on the particular subproblem.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Conduct a sensitivity analysis at the final subproblem for a range of differencing intervals.</td>
</tr>
</tbody>
</table>
Table 5--continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Column</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>28</td>
<td>=0</td>
<td>Do not estimate the partial derivatives of the estimates of the Lagrange multipliers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1</td>
<td>Estimate the partial derivatives of the estimates of the Lagrange multipliers whenever a sensitivity analysis of the solution point is conducted.</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>=0</td>
<td>Estimate the partial derivatives of the optimal value function and eliminate those parameters which do not affect the optimal value functions from subsequent sensitivity calculations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1</td>
<td>Estimate the partial derivatives of the optimal value function with respect to all parameters, but continue all subsequent sensitivity calculations with respect to all parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>Do not estimate the partial derivatives of the optimal value function first. Conduct the sensitivity analysis with respect to all parameters.</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>=0</td>
<td>Do not transform the results.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1</td>
<td>The problem being solved, ( P(x) ), is a convex equivalent of a geometric programming problem ( G(t) ) obtained by transformation ( c_i = e^{-x_i} ). Thus back transform the results to ( t ) space.</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>=1</td>
<td>( f^<em>(e) ) is convex. Derive parametric upper and lower bounds on ( f^</em>(e) ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=2</td>
<td>( f^<em>(e) ) is convex. Derive parametric upper bound on ( f^</em>(e) ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=3</td>
<td>( f^<em>(e) ) is convex. Derive parametric lower bound on ( f^</em>(e) ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=4</td>
<td>( f^<em>(e) ) is concave. Derive parametric upper and lower bounds on ( f^</em>(e) ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=5</td>
<td>( f^<em>(e) ) is concave. Derive parametric upper bound on ( f^</em>(e) ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=6</td>
<td>( f^<em>(e) ) is concave. Derive parametric lower bound on ( f^</em>(e) ).</td>
</tr>
</tbody>
</table>
NOTE: This data deck arrangement assumes that the user subroutines are linked with the rest of the SENSUMT subroutines (which are loaded in a data set) via the JCL cards.

Figure 1.—Data deck structure for SENSUMT.
5. Coded Examples and Input/Output Illustrations

In this section we provide the input listing and output description of two illustrative examples which are taken from [11].

5.1 Example 1

Consider the following convex RHS programming problem:

\[
\begin{align*}
\text{minimize} & \quad f(x) = (x_1-4)^2 + (x_2-2)^2 \\
\text{subject to} & \quad g_1(x) = -x_1^2 + x_2 \geq \epsilon_1 \\
& \quad g_2(x) = -x_1 - x_2 \geq \epsilon_2.
\end{align*}
\]

It is desired to solve and analyze this problem for sensitivity when \( \epsilon_1 = 0 \) and \( \epsilon_2 = -3 \). Moreover, it is desired to derive parametric upper and lower bounds on the optimal value function of this problem when

\[(i)\quad -1 \leq \epsilon_1 \leq 0 \text{ while } \epsilon_2 = -3, \text{ and}
\]

\[(ii)\quad -3 \leq \epsilon_2 \leq -1 \text{ while } \epsilon_1 = 0.
\]

5.1.1 Computer listing of the code deck.

Figure 2 shows the computer listing of the code of Example 1. The letters in circles categorizing the input data correspond to those indicated in the code deck structure depicted in Figure 1. The format of the listed data is given in Section 4.

5.1.2 Selected pages from the computer output.

Figure 3 lists an annotated computer output for Example 1. An explanation of the meaning of the corresponding steps follows.
Figure 2.—Computer listing of the code for Example 1.
Figure 3.—Annotated computer output for Example 1.
Figure 3.--continued
**Figure 3.—continued**
<table>
<thead>
<tr>
<th>POINT</th>
<th>DOT</th>
<th>RHO</th>
<th>MAGNITUDE</th>
<th>PHASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3082480-05</td>
<td>0.70276280-01</td>
<td>0.3146870-01</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.3083690-01</td>
<td>0.73467030-01</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.3083690-01</td>
<td>0.73467030-01</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.—continued
Figure 3.—continued
Figure 3—continued
Sensitivity Analysis

The value of $\rho_{k1}^0$ is 0.238420-05

The point at which the estimate of sensitivity will be made is

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>Differentiating Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.100000-09</td>
</tr>
<tr>
<td>2</td>
<td>0.100000-09</td>
</tr>
</tbody>
</table>

Optimal Value Function Sensitivity

<table>
<thead>
<tr>
<th>$\frac{df}{dx}$, $x_1 = 1.305098$</th>
<th>$\frac{df}{dx}$, $x_2 = 1.903824$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{df}{dx}$ (x1) = 1.305098</td>
<td>$\frac{df}{dx}$ (x2) = 1.903824</td>
</tr>
</tbody>
</table>

Detailed sensitivity results follow for parameters 1 and 2.

X-derivatives are with respect to parameter 1:

<table>
<thead>
<tr>
<th>$\frac{dx_1}{dx}$ (x1) = 0.2773500</th>
<th>$\frac{dx_2}{dx}$ (x1) = 0.2773500</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{dx_1}{dx}$ (x2) = 0.5063357</td>
<td>$\frac{dx_2}{dx}$ (x2) = 0.46223350-01</td>
</tr>
</tbody>
</table>

X-derivatives are with respect to parameter 2:

<table>
<thead>
<tr>
<th>$\frac{dx_1}{dx}$ (x1) = 0.2773497</th>
<th>$\frac{dx_2}{dx}$ (x1) = 0.7226484</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{dx_1}{dx}$ (x2) = 0.46223350-01</td>
<td>$\frac{dx_2}{dx}$ (x2) = 1.308971</td>
</tr>
</tbody>
</table>

Figure 3.—continued
Figure 3.--continued
1ST ORDER ESTIMATES
P = 0.62643750 01
G = 0.62643670 01
SIGMA = 0.0
H = 0.0

LAGRANGE MULTIPLIERS
P = 0.62643750 01
G = 0.62643670 01
SIGMA = 0.0
H = 0.0

THE CURRENT VALUE OF X IS
0.15615533 01
0.14384470 01

THE CONSTRAINT VALUES
0.89499520 00
0.20043080 00

SENSITIVITY ANALYSIS

THE VALUE OF R(RO) IS 0.238420-05

THE POINT AT WHICH THE ESTIMATE OF SENSITIVITY WILL BE MADE IS
X(1) = 1.5615532
X(2) = 1.4384472

PARAMETER VALUE DIFFERENCING INTERVAL
1 -1.000000 0.100000-09
2 -3.000000 0.100000-09

OPTIMAL VALUE FUNCTION SENSITIVITY

DF/DX(1) = 0.8949953
DF/DX(2) = 2.004307

Figure 3.--continued
Figure 3.—continued
Figure 3.--continued
Optimal value function round when part 2) is performed

Point 1 (updated solution)

\[ F(2) = 0.3345180 \]

Point 2 (updated solution)

\[ F(2) = 0.14055730 \]

---

Line passing through points 1 and 2 and over estimating F#

\[ F = 0.3345180 \] \[ \text{part 2) } + 0.17600250 \]

---

Line under estimating F# at point 1

\[ F = 0.19038260 \] \[ \text{part 2) } + 0.13078170 \]

---

Line under estimating F# at point 2

\[ F = 0.19038260 \] \[ \text{part 2) } + 0.13078170 \]

---

Quadratic estimation of F# through points 1 and 2

\[ F = 0.72034670 \] \[ \text{part 2) } + 0.62259050 \] \[ \text{part 2) } + 0.19561290 \]

---

F# round evaluation at ten equidistance points between points 1 and 2

<table>
<thead>
<tr>
<th>Part 2</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>Quad Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.3345180 01</td>
<td>0.73666930 01</td>
<td>0.73666930 01</td>
</tr>
<tr>
<td>-2.000</td>
<td>0.73666930 01</td>
<td>0.90356960 01</td>
<td>0.77762710 01</td>
</tr>
<tr>
<td>-2.400</td>
<td>0.87045000 01</td>
<td>0.93796030 01</td>
<td>0.82437899 01</td>
</tr>
<tr>
<td>-2.600</td>
<td>0.93796030 01</td>
<td>1.00472110 02</td>
<td>0.93507740 01</td>
</tr>
<tr>
<td>-2.000</td>
<td>0.93796030 01</td>
<td>0.97171710 02</td>
<td>0.99808640 01</td>
</tr>
<tr>
<td>-1.800</td>
<td>0.97171710 02</td>
<td>1.10387110 02</td>
<td>1.06885940 02</td>
</tr>
<tr>
<td>-1.600</td>
<td>1.00472110 02</td>
<td>1.12049020 02</td>
<td>1.14439330 02</td>
</tr>
<tr>
<td>-1.400</td>
<td>1.12049020 02</td>
<td>1.12719207 02</td>
<td>1.12569000 02</td>
</tr>
<tr>
<td>-1.200</td>
<td>1.12719207 02</td>
<td>1.13886820 02</td>
<td>1.13277097 02</td>
</tr>
<tr>
<td>-1.000</td>
<td>1.13886820 02</td>
<td>1.14055730 02</td>
<td>1.14055730 02</td>
</tr>
</tbody>
</table>

Figure 3—continued
Output for solution of the unperturbed problem, i.e., \( \varepsilon_1 = 0 \), \( \varepsilon_2 = -3 \) (Step 1 of Algorithm 3.1).

1. Printout of parameter card data.
2. Printout of first option card data.
3. Printout of tolerance card data.
4. Printout of second option card data.
5. Elapsed time since the start of the problem. (This information is not accurate. Subroutine TIMEC, which monitors the elapsed time, has to be modified.)
6. Initial starting point \( x^0 \) and corresponding problem function values.
7. Feasibility of \( x^0 \) is verified. Note that if \( x^0 \) is not feasible, then SENSUMT invokes the subroutines FEAS and BODY to find a feasible starting point.
8. The solution to the first subproblem. Program took four inverse product moves to minimize \( W \) for \( \text{RH} \Phi = 10, (r=10) \)

\[
D\Phi T = \nabla_{x} W(x(r))^T \left[ \frac{\partial^2 W(x(r))}{\partial x_i \partial x_j} \right]^{-1} \nabla_{x} W(x(r)) < 10^{-7}
\]

**MAGNITUDE** = \( \| \nabla_{x} W(x(r)) \| \) (the magnitude of the gradient of \( W \))

\[
F = f(x(r)) \quad \text{(the value of the objective function)}
\]

\[
W = f(x(r)) - r \sum_{j=1}^{M+N} g_j(x(r)) + \sum_{j=M+1}^{N+M} h_j(x(r)) \quad \text{the value of the P-function)
\]

Note: The \( W \)-function coded in SENSUMT differs from that given in (2.6) in the last term by a factor of 2.
\[ \text{RESIGMA} = -r \sum_{j=1}^{M} \ln g_j[x(r)] \quad \text{(note that this may be negative)} \]

\[ H = \sum_{j=M+1}^{M+MZ} h_j^2[x(r)] / r \]

\[ G = \text{dual value} = f[x(r)] - r^*N + 2^*H \]

(in a convex problem \( F \geq v^* \geq G \))

where the current value of \( x \) is \( x(r) \) and the current constraint values are \( g_1^*[r] \) and \( g_2^*[x(r)] \).

9. Values of \( F, W \) (designated by \( P \) in the computer output), \( G \), \( RSIGMA \), and \( x \) are repeated. Current constraint values (i.e., \( u_j \) and \( w_j \)) are \( r/g_j \), \( j=1, M \), and \( 2n_j(x)/r \), \( j=M+1, M+MZ \). Note: Since Example 1 has no equality constraint, the entries relating to \( h_j \) in 8 and 9 are zero.

10. The solution to the second subproblem (\( r = 10/4 = 2.5 \)).

11. The current first order estimates of \( x \) (obtained by first order extrapolation, using the solution of the first and second subproblems) and the corresponding problem functions.

12. The solution to the third subproblem (\( r = 2.5/4 = .625 \)).

13. The current second order estimates of \( x \) (obtained by second order extrapolation using solutions of the first, second, and third subproblems) and the corresponding problem functions.

14. The solution to the fourth subproblem.

15. The solution to the fifth subproblem.

16. The solution to the sixth subproblem.

17. The solution to the seventh subproblem.

18. The solution to the eighth subproblem.

19. The solution to the ninth subproblem.

20. The solution to the tenth subproblem.
The solution to the eleventh subproblem.

The solution to the twelvth (final) subproblem.

The general information about the sensitivity analysis including the value of \( r \), the estimated final solution point and the parameter values and associated differencing intervals.

Estimates of the sensitivity of the optimal value function obtained by taking the gradient of the Lagrangian with respect to the parameters as described in Step 9 of Algorithm 2.1.

The parameters for which detailed sensitivity results follow. Here it is both parameters since the optimal value function is sensitive to both.

Estimates of the partial derivatives of the solution point taken with respect to parameter one as in Steps 1-3 of Algorithm 2.1.

Estimates of the partial derivatives of the Lagrange multipliers taken with respect to parameter one as in Step 5 of Algorithm 2.1.

Estimate of the partial derivative of the optimal value function taken with respect to parameter one and obtained by using the chain rule as in Step 8 of Algorithm 2.1.

Same as 26, 27, and 28 but with respect to parameter two.

Output for solution of the first perturbed problem, i.e., \( \epsilon_1 = -1 \), \( \epsilon_2 = -3 \) (Step 2 of Algorithm 3.1).

Printout of the input data and subproblem solutions for the first perturbed problem.

Printout of the final subproblem for the first perturbed problem.

Same as 29 but for the first perturbed problem.

Same as 24 but for the first perturbed problem.
III  Output for optimal value function bounds as a function of first parameter (Steps 3 and 4 of Algorithm 3.1).

34  \((f^*(\varepsilon_1), \varepsilon_1)\) of the unperturbed and first perturbed problems [see \((22), (23), \) and \((31), (32), \) respectively].

35  Equations of parametric upper bound on \(f^*(\varepsilon_1)\).

36  Equations of parametric lower bounds on \(f^*(\varepsilon_1)\).

37  Maximum of these equations over the range of \(-1 \leq \varepsilon_1 \leq 0\) provides a lower bound on \(f^*(\varepsilon_1)\).

38  Equation of a quadratic estimate of \(f^*(\varepsilon_1)\) [see the description of subroutine \(\text{BOUND}\) in Section 6 for the method of deriving this equation].

39  Value of bounds and the quadratic estimate of \(f^*(\varepsilon_1)\) at eleven equidistant points over the perturbation range of the first parameter \(\varepsilon_1\).

IV  Output for solution of the second perturbed problem, i.e., \(\varepsilon_1 = 0, \varepsilon_2 = -1\) (Step 2 of Algorithm 3.1).

   Description of the output for the second perturbed problem is similar to that of the first perturbed problem, i.e., \((30) - (33)\).

V   Output for optimal value function bounds as a function of the second parameter (Steps 3 and 4 of Algorithm 3.1).

   The description here again closely parallels that of the first parameter, i.e., \((34) - (39)\).

   The graphical depiction of the bounds derived for \(f^*(\varepsilon)\) as a function of \(\varepsilon_2\), together with a plot of the analytical solution and quadratic estimates of \(f^*(\varepsilon)\) at different values of \(\varepsilon_2\) is in Figure 4.
Figure 4.—Graph of bounds on \( f^*(\varepsilon_2) \) of Example 1 (computer solution).

5.2 Example 2

Minimize \( f(x) = x_1^2 + .5e^x_2 - .125x_3 + .25(x_4-40)^2 - .1e^x_5 \)

\( x_1, \ldots, x_5 \)

Subject to

\( g_1(x) = -.5x_1^2 + 6x_2 - 5e^x_3 - .05x_4^2 - .5/x_5 \geq \epsilon_1 \)

\( g_2(x) = 5e^{-x_1} - 2x_2^2 + 3x_3 + x_4 + 3x_5 \geq -12 \)

\( g_3(x) = 3x_1 + x_2 - x_3^2 + x_4 - x_5^2 \leq 2 \)

\( 0 < x_3 < 5, \ 0 < x_5 < 5, \ 0 < x_j < \infty, \ j=1,2,4. \)
It is desired to calculate bounds on \( f^*(\epsilon_1) \) of this problem for \( \epsilon_1 \in [-10,5] \).

**(i) Problem SR(\epsilon).**

* Standard format:

\[
\begin{align*}
\text{Min} \quad f(x) &= \sum_{j=1}^{5} f_j(x_j) = x_1^2 + .5e^{x_2} - .125x_3^2 \\
&\quad + .25(x_4-40)^2 - .1e^{x_5} \\
\text{s.t.} \quad x &\in G \cap B, \text{ where} \\
G &= \{x : g_1(x) \geq \epsilon_1, g_2(x) \geq -12, g_3(x) \geq 2\}, \quad -10 \leq \epsilon_1 \leq 5 \\
\text{and} \\
B &= \{0 \leq x_3 \leq 5, 0 \leq x_5 \leq 5, 0 \leq x_i \leq \infty, i=1,2,4\}.
\end{align*}
\]

**(ii) Problem \( \tilde{\text{CSR}}(\epsilon) \), (convex overestimating problem of SR(\epsilon)).**

* Problem formulation:

\[
\begin{align*}
\tilde{f}_3(x_3) &= -3.125x_3 + 6.014 \\
\tilde{f}_5(x_5) &= -2.948x_5 + 7.027 \\
\text{Remaining terms are identical to those of problem SR(\epsilon); thus problem } \tilde{\text{CSR}}(\epsilon) \text{ reads} \\
\text{Min} \quad \tilde{f}(x) &= x_1^2 + .5e^{x_2} - 3.125x_3^2 + .25(x_4-40)^2 \\
&\quad - 2.948x_5 + 13.041 \\
\text{s.t.} \quad x &\in \tilde{G} \cap B, \quad -10 \leq \epsilon_1 \leq 5, \\
\text{where} \quad \tilde{G} &= G. \\
\text{Note: } \tilde{f}_3(x_3) \text{ and } \tilde{f}_5(x_5) \text{ were chosen here to be the lowest nonunderestimating lines parallel to the convex envelopes of } f_3(x_3) \text{ and } f_5(x_5), \text{ respectively.}
\end{align*}
\]
(iii) Problem CSR(c), (convex underestimating problem).

- Problem formulation:

\[ f_3(x_3) = -3.125x_3 \]

\[ f_5(x_5) = -2.948x_5 \]

Remaining terms are identical to those of problem SR(c); thus problem CSR(c) reads

\[ \min f(x) = x_1^2 + 0.5e^{-x_2} - 3.12x_3 + 0.25(x_4 - 40)^2 - 2.948x_5 + 0.1 \]

S.t. \[ x \in \mathbb{C} \cap B, \quad -10 \leq x_1 \leq 5 \]

where \( \mathbb{C} \cap G \).

The computer listing of the code for problem CSR(c) is shown in Figure 5. The listing of the code for CSR(c) is identical to that of the problem CSR(c) except for the constant term in the objective function, which is +13.041, rather than -0.1.

(iv) Bounds.

Figures 6 and 7 depict the calculated upper and lower bounds via the analysis of the problems CSR(c) and CSR(c), respectively. These results are summarized below.

Upper bound:

\[ \overline{f}(\varepsilon_1) = 4.825\varepsilon_1 + 136.628 \]

Lower bound:

\[ \underline{f}(\varepsilon_1) = \max[1.880\varepsilon_1 + 94.032, 5.466\varepsilon_1 + 120.284] \]

The graphical depiction of these bounds is in Figure 8.
Figure 5.--Computer listing of the code for the convex overestimating problem of Example 2.
RETURN

1   DEL(1)=-3.
2   DEL(2)=-1.
3   DEL(3)=-2.*X(3)
4   DEL(4)=-1.
5   DEL(5)=-3.*X(5)
RETURN

4   DEL(3)=-1.
RETURN

5   DEL(5)=-1.
RETURN

END

SUBROUTINE MATRIX(IN,K)
IMPLICIT REAL*(A-H,O-Z)
COMMON/SHARE/X(45),DEL(45),A(45,45),N,M,NPI,NI
COMMON/COM/PAR(45),DPAR(45),PAR,ISENS
L=IN+1
GO TO (20,1,2,3,4,4),L

20   A(1,1)=2.
20   A(2,2)=5.*DEXP(X(2))
20   A(4,4)=.5
RETURN

1   A(1,1)=-1.
1   A(3,3)=5.*DEXP(X(3))
1   A(4,4)=-1.
1   A(5,5)=1./X(5)**3
RETURN

2   A(1,1)=-5.*DEXP(-X(1))
2   A(2,2)=-4.
RETURN

3   A(3,3)=-2.
3   A(5,5)=-2.
RETURN

4   K=1
RETURN

END

EXEC FOR6,DSN='OR79961,BOUNDS45',PRG=MAIN
//G0,SYSLIB (X)
// DD
// UN DSN=GW,FWLIB,DIS=SHR
//G0,SYSLIN DO
// UN DSN=TEMPUNCH,DIS=(OLD,DELETE)
//F107F01 DO SYMTAB=A
1.0E-08 10.0 1.0E-04 4.0 900.0 5 5 0
1.5 3 3.4 1.4 1.1 1 1
3.0E-10 0.0001
1 4 1 1 0 2

//

Figure 5.--continued
Figure 6.--Parametric upper bound on $f^*(\epsilon_1)$ via problem $CSR(\epsilon)_1$, Example 2 (computer solution).
Figure 7.--Parametric lower bound on $f^*(\varepsilon_1)$ via problem CSR(\varepsilon_i), Example 2 (computer solution).
Figure 8.--Parametric bounds on $f^*(\epsilon_1)$, Example 2 (computer solution).
6. General Description and Listing of the SENSUMT Subroutines

The subroutines comprising SENSUMT fall into four categories:

(i) user subroutines,
(ii) SUMT subroutines,
(iii) sensitivity subroutines, and
(iv) bound subroutines.

As mentioned before, user subroutines are generally composed of four subroutines, i.e., READIN, RSTNT, GRAD1, and MATRIX. These subroutines respectively provide pertinent information about the problem, functions of the problem, their gradients and Hessian matrices. A detailed description of these subroutines, together with the corresponding codes for Examples 1 and 2, were given in Section 4 and Figures 2 and 5.

SUMT subroutines implement the Sequential Unconstrained Minimization Technique of Fiacco and McCormick [10]. The subroutines, along with program MAIN, comprising this group are:

- BODY
- CHECKER
- CONVRG
- DIFF1
- DIFF2
- ESTIM
- EVALU
- FEAS
- FINAL
- GRAD
- INVERS
- MAIN
- OPT
- OUTPUT
- PEVALU
- PUNCH
- REJECT
- RHOCOM
- SECOND
- SECORD
- SET
- STORE
- TCHECK
- TIMEC
- TRANS
- XMOVE

With the exception of the subroutine TRANS, the above routines have been developed by Mylander, Holmes, and McCormick [13]. Some of
these routines, in particular program MAIN and subroutines BODY and OUTPUT, have been modified for implementing sensitivity and bound calculation routines. Subroutine TRANS was recently developed to aid the user when analyzing a convex equivalent of geometric programming problems by SENSUMT. See page 104 for a more detailed description of this subroutine.

Sensitivity subroutines implement the sensitivity analysis Algorithm 2.1 and interface it with the SUMT subroutines. A brief history of the development of these subroutines was given in Section 1. The latest version of the codes comprising this group, due to Armacost [1], are subroutines SENS, PARDIF, IMULT, and PRESEN. Subroutines SENS and PRESEN have been slightly modified in implementing the bound calculation routines.

Bound subroutines, developed in [11], implement the bound calculation Algorithms 3.1 and 3.2 given in Section 3. The two subroutines in this group are BOUND and PERT.

All of the subroutines comprising SENSUMT are dimensioned to solve problems having at most 45 variables, 45 parameters, and 200 constraints. However, if the computer capacity permits, they may readily be re-dimensioned to solve problems of larger size. All of these subroutines are separately filed in the Conversational Monitor System (CMS) component of the IBM Virtual Machine facility 37C (VM/370) at The George Washington University Center for Academic and Administrative Computing under their corresponding names.

To make this manual self-contained, the computer listing and a general description of each subroutine is provided (in alphabetical order by name). The descriptions of the SUMT subroutines are largely taken from [13], and those of the sensitivity subroutines are taken from [1]. The user subroutines are problem-dependent, thus they are included in the following listings. For familiarity with the user subroutines, the reader should refer to the coding of Examples 1 and 2 provided in Figures 2 and 5, respectively.
6.1 BODY

Subroutine BODY coordinates the flow among the subroutines that actually do the calculations required by the various phases of the algorithm. The flow in this routine is slightly different when the program is in the feasibility phase (solving the entry problem) rather than the normal phase (solving the stated NLP problem). The listing appears as Figure 9.

6.2 BOUND

The subroutine BOUND, called by the program MAIN, generates the equations for the upper and lower bounds of the optimal value function as functions of the problem parameters. When analyzing a problem which is known to have a convex or concave optimal value function, BOUND also derives the equation for a quadratic function which approximates the optimal value function. If \( f^*(\varepsilon) \) is convex (concave), it corresponds to the higher (lower) of the two quadratic functions, in the perturbation range of the parameter of interest, which passes through the two calculated points in \( f^*(\varepsilon), \varepsilon \) space and has the calculated slope at each of these points. Evaluation and printout of this quadratic function (when applicable), and the upper and/or lower bound at eleven equi-distant points over the perturbation range of the parameters of interest are also programmed in subroutine BOUND. The listing appears as Figure 10.

6.3 CHECKER

Subroutine CHECKER evaluates the first partial derivatives for all the functions at the starting point first by calling the user-supplied subroutine GRAD1 and then by calling DIFF1. The results are printed out to aid the user in debugging the subroutines used to describe the NLP problem he wishes to solve.

The matrix of second partial derivatives of each function is also evaluated by the two methods; first by calling MATRIX and then calling
Figure 9.—Subroutine $\text{BODY}$. 

- 62 -
C SUBROUTINE BOUND WAS DEVELOPED BY GHAEINI (1979) TO CALCULATE
C UPPER AND/OR LOWER PIECEWISE LINEAR PARAMETRIC BOUNDS ON
C OPTIMAL VALUE FUNCTIONS WHEN THESE FUNCTIONS ARE KNOWN TO BE
C CONVEX OR CONCAVE OVER THE RANGE OF A GIVEN PARAMETER.

0001 SUBROUTINE BOUND

0002 IMPLICIT REAL*4(A-H,O-Z)

0003 COMMON/OPY/NEXOP7,NPAR(41,NPAR),SENS

0005 COMMON/ARG/LAT,LZ,FE2

0006 COMMON/ARGZ/DS(F4.5,DF2145)

0007 COMMON/OP7/NEQP7

0008 DIMENSION E(20),FQ(20),FQI2(20),FUXZ(20)

0009 E1=PARI(L1)

0010 E2=PARI(L2) + PER(21)

C LINE FIT THROUGH POINTS 1 AND 2

0011 SIZ = (FQI1/E1)/E2 -E2)

0012 SIZ = FE1-5IZE)

C TANGENT LINE UNDER ESTIMATING F AT POINT 1

0013 S1=PARI(L1)

0014 B=FQI1-S2(E2)

C TANGENT LINE UNDER ESTIMATING F AT POINT 2

0015 S2=PARI(L2)

0016 B=FQI1-S2(E2)

C INTERSECTION OF THE ABOVE TWO LINES

0017 E1=I*BU(11/11-52)

0018 FE3=I*BU(11-521)/I1-52)+ B1

C QUADRATIC FIT THROUGH POINTS 1 AND 2

0019 A=(FE2-E251-1)-E1111/(IE1-E1111#2)

0020 C=(11#11)-FE2-E211)-IE1111*#2-2.011E1111/IE1-E1111#2)

0021 B=11-11*11011

0022 AA=(FE111151-11-2)-E2111)*((E111111-E11#2)*E1111111/(E11-E1111#2)

0023 CC=(11#111)-((E111151-11-2)-E2111)*IE1111111/(E11-E1111#2)

0024 BB=11-11*11011

C BOUND EVALUATION AT 10 DISCRETE POINTS

0025 E1111=E1

0026 E=PER1(L111/10.

0027 DO 580 I=1,10

0028 IF(11.EQ.11) GO TO 750

0029 E11=11*1111+DE

C QUADRATIC ESTIMATE


C UPPER LINEAR BOUND

0032 FULL(11)=S12*E111111111

C LOWER LINEAR BOUND

0033 IF(PER1(L1111)/10.) GO TO 585

0034 IF(E1111/11.EQ.1111) GO TO 570

0035 FL12(11)=S12*E111111111

0036 GO TO 585

0037 FL12(11)=S12*E111111111

0038 GO TO 585

0039 595 IF(E1111/11.EQ.1111) GO TO 590

0040 FL12(11)=S12*E111111111

0041 GO TO 590

0042 590 FL12(11)=S12*E111111111

0043 595 CONTINUE

0044 580 CONTINUE

C CHOICE ON PROPER QUADRATIC FIT

0045 IF(NEQP7 .GE.0) GO TO 810

0046 IF(FQI2(11) .GT. FQI(11)) GO TO 800

0047 A=AA

0048 B=BB

0049 C=CC

0050 DO 100 I=1,10

0051 FQI11=FQ111

0052 100 CONTINUE

0053 GO TO 800

0054 810 IF(FQI2(11) .LT. FQI(11)) GO TO 800

0055 A=AA

0056 B=BB

0057 C=CC

0058 DO 110 I=1,10

0059 FQI11=FQI11

0060 110 CONTINUE

0061 800 CONTINUE

0062 WRITE(6,600) L1

0063 600 FORMAT(1X,25X,*OPTIMAL VALUE FUNCTION BOUNDS WHEN PAR'1,12,

0064 *S1 IS PERTURBED*,/24S,57(*8*1,,1/)

0065 WRITE(6,610) L2,E11111,FL1Z,Z,LZ,FE2

0066 610 FORMAT(1X,*POINT 1 (UNPERTURBED SOLUTION) ^1/2,2,PAR'1,12,

0067 *S1**=',E15.7,10X,*PAR'1,12,=*=',E15.7,10X,*POINT 2 (PERTURBED)

0068 S1**=',E15.7,10X,*PAR'1,12,=*=',E15.7,10X, *PAR'1,12=*

0069 *=',E15.7,10X, *PAR'1,12=*

0070 TO 810)

0080 GO TO 810

C OUTPUT WHEN NEQP7=1

0081 WRITE6,620) L1,L2,E111111111

0082 520 FORMAT(1X,*LINE PASSING THROUGH POINTS 1 AND 2 OVER*

0083 Figure 10.—Subroutine BOUND.
Figure 10.--continued

- 64 -
DIFF2. Both results are printed out. If it is found that MATRIX creates a nonzero element below the main diagonal of A, then a switch is set to cause the statement STOP to be executed in MAIN. Subroutine CHECKER appears as Figure 11.

6.4 CONVRG

After each iteration of the algorithm to locate the minimum of the unconstrained function (the W function) subroutine CONVRG is called to determine if the current point is an acceptable approximation of a point giving the minimum value of the W function. The argument N2 of this routine is given a value of 1 if the point is close enough; otherwise, it is given a value of 2. CONVRG appears as Figure 12.

6.5 DIFF1

Subroutine DIFF1 computes the first derivatives by numerical differencing. The user-supplied subroutine RESTNT is called 2n times for each gradient evaluated by DIFF1. For the function $f$, DIFF1 computes the $i$th component of the gradient using the formula

$$(\nabla f(x^0))_i = \frac{f(x^0 + \theta e_i) - f(x^0 - \theta e_i)}{2\theta},$$

where $e_i$ is a vector of zeroes with a 1 in the $i$th component and $\theta$ is a small number whose value is assigned by the user. DIFF1 is shown as Figure 13.

6.6 DIFF2

Subroutine DIFF2 computes the second derivatives by numerical differencing. The matrix of second partials is computed using central differencing. The differences are calculated using the gradients of the function. The user-supplied subroutine GRADI1 is called 2n times for each matrix of second partial derivatives evaluated by DIFF2. DIFF2 is Figure 14.
SUBROUTINE CHECKER

MARCH 1971


IMPLICIT REAL*8, I, N

REAL*4 X(4), X(2), X(2)

COMMON /SHARE/X(4), XI, MIN, MAX

COMMON /EXCEPTON/EXOP1, EXOP2, EXOP3, EXOP4, NEPOX4, NEXO, NEP1, NEX01, NEP2, NEX02

M=(1+X)

DO 5 J=1, N

DEL(J)=1.2345678

CONTINUE

WRITE (6,1701 IN

CALL GRADIENT (IN)

WRITE (6,1603 4DEL6)

CALL DIFF2 (IN)

CONTINUE

IF (NEOPI.L.T. 41 GO TO 160

WRITE (6,1901

150 IN-1, N

WRITE (6,1703 IN

IN-2

DO 30 K=1, N

D0 20 AIKJ, 0

CONTINUE

CALL MATRIX (MIN, IT)

IF (IT.EQ.1) GO TO 150

DO 50 K=1, N

XMI-K-1

DO 30 J=1, N

IF (AIKJ.EQ.0.0) GO TO 40

NEXOPI=5

WRITE (6,2101 K, J

GO TO 60

40 CONTINUE

GO TO 60

DO 100 J=1, N

DO 110 CONTINUE

WRITE (6,2001 K

GO TO 90

60 DO 10 K=1, N

DO 70 J=1, N

IF (AIKJ.NE.0.0) GO TO 40

CONTINUE

WRITE (6,2201 K

GO TO 90

100 AIKJ=0.

110 CONTINUE

WRITE (6,2119) IN

115 FORMAT (13HO CALL DIFF2 (IN, IT) )

120 CONTINUE

125 WRITE (6,2201 K

GO TO 90

130 WRITE (6,2101 K, J)

DO 140 K=1, N

DO 150 J=1, N

140 CONTINUE

WRITE (6,2301 K

DO 160 J=1, N

150 CONTINUE

WRITE (6,2401 K

DO 170 J=1, N

160 CONTINUE

RETURN

170 FORMAT (12HO CHECKER....CONSTRAINT NO. 131)

180 FORMAT (11HO. CHECKER....1ST PARTIALS/E10.6, E10.6, E10.6, E10.6, E10.6, E10.6, E10.6)

190 FORMAT (10HO. CHECKER....2ND PARTIALS)

END

Figure 11.---Subroutine CHECKER.
Figure 12.—Subroutine CONVRG.
SUBROUTINE DIFF1 (IN)
C FEBRUARY 1971
C DIFF1 COMPUTES THE FIRST DERIVATIVES AT NUMERICAL DIFFERENCING.
C USER CAN CALL FOR DIFFERENCING OF SELECTED FUNCTIONS
C
IMPLICIT REAL*8 (A-H, O-Z)
REAL*8 XEP1, XEP2
COMMON/Xören/(45), DEL(45), A45, A51, N4, N5, N1, N7
COMMON/XEP1/OEX2P, NEX2P, NEXOP5, NEXOP7, XEP1, XEP2
COMMON/STRT/XSTR(45), XSTR(45), XSTR(45), XSTR(45), XSTR(45)
DO 10 J=1, N
DO 30 J=1, N
IF ( J.EQ.1 ) GO TO 20
JN=J-1
X(JN)=XSTR(JN)
X(J)=XSTR(J)+XEP1
CALL RESTNT (IN, ZZ1)
X(J)=XSTR(J)-XEP1
30 DEL(XJ)=ZZ2-ZZ1/ZZ2*XEP1
RETURN
END

Figure 13.—Subroutine DIFF1.
SUBROUTINE DIFF2 (IN)
DIFF2 COMPUTES THE SECOND DERIVATIVES BY NUMERICAL DIFFERENCING.

IMPLICIT REAL*8 (A-H,O-Z)
REAL*4 XEP1,XEP2
COMMON/SHARE/XEPI,XEP2,DEL1(45),DEL2(45),N(45,45),M,IN,INP1,INP2
COMMON/EXOP/NEOP1,NEOP2,NEOP3,NEOP4,NEOP5,NEOP6,NEOP7
COMMON/STIRA/XSRI(45),XSSS(45),DOLL(45)
COMMON/EXOPT/P4EXPI,NEXOP1,NEXOP2,NEXOP3,NEXOP4,NEXOP5,XEP1,XEP2

DO 10 J=1,N
DO 50 J=1,N
IF (J.EQ.1) GO TO 20
JM=J-1

10 XSSS(JM)=XSSS(J)

DO 50 J=1,N
X(JM)=XSSS(JM)
CALL GRAD1 (IN)
DO 10 J=1,N

30 DOLL(J)=DEL(J)
X(J)=XSSS(J)+XEP1
CALL GRAD1 (IN)
DO 40 J=1,N

40 AIJ,J=(DOLL(J)-DEL(J))/(XEP1)
CONTINUE
50 CALL GRAD1 (IN)
50 CONTINUE
10 RETURN

END

Figure 14.—Subroutine DIFF2.
6.7 ESTIM

After the minimum of the W function for a given value of \( r \) has been located, ESTIM is called. When the minimum of the W function for a given value of \( r \) is determined a subproblem is said to be solved. ESTIM performs the computations to estimate the Lagrange multipliers and make the first- and second-order estimates of the final solution of the problem. ESTIM is Figure 15.

6.8 EVALU

In the normal phase, EVALU calls the user-supplied routines to evaluate the objective function and the constraint functions at the current point \( x \). As the constraint functions are being computed, the code performs the calculations to compute the penalty terms of the penalty function. In the feasibility phase this routine puts the negative sum of the violated constraints, which is the objective function of the entry problem, in location \( F \). In location \( F \) the current value of the objective function is stored for use by the program. After \( F \) has been computed, the value of the penalty function is computed. Also subroutine EVALU computes a value for the dual objective function. This value is only correct at the solution to a subproblem and the value is stored in the location labeled \( G \). EVALU appears as Figure 16.

6.9 FEAS

Subroutine FEAS determines whether the starting point is an interior feasible point or not. If the variables of a problem are supposed to be nonnegative and some of the components of the starting point vector are nonpositive, then those components are changed to small positive numbers. A modification of FEAS must be made if it is desired to change this number. If the starting point does not satisfy the nontrivial constraints (the constraints other than the nonnegativity constraints), then the program goes into the feasibility phase. In this phase the negative of the sum of all the violated inequality constraints
Figure 15.—Subroutine ESTIM.
Figure 15.--continued
SUBROUTINE EVALU

IMPLICIT REAL*(A-H,O-Z)

REAL RHOIN,NRATIO,EPsi,THETA0
COMMON SHARE/X(45),DIAG1(45),F
CALL RSIGMA-RSIGMA-RHO*DFLOAT4I
DO 58 (J,RJ(J))
RSIGMA.RSIGMA-RHO*DFLOAT4I
RETURN
END

Figure 16.-- Subroutine EVALU.
C--- STRAIGHT FUNCTION EVALUATION (MAIN*FEAS ONLY)
0063 200 CONTINUE
0064 IF (N.EQ.0) GO TO 220
0065 DO 210 I=1,N
0066 CALL RESTNT (1,N,I)
0067 210 CONTINUE
0068 220 CALL RESTNT (0,F)
C EQUALITY CONSTRAINTS
0069 IF (NZ) 250,250,230
0070 230 DO 240 I=1,NZ
0071 NZ=NZ+1
0072 240 CALL RESTNT (KZ,RA(KZ))
0073 250 RETURN
C CONSTRAINTS VIOLATED NOT SO BEFORE
0074 260 NSATIS=3
0075 P0=10.035
0076 RETURN
0077 END

Figure 16.--continued
is labeled for use as the objective function in the entry problem. Then
the routine calls BØDY to minimize this auxiliary function subject to
the set of satisfied constraints. In the feasibility phase if a vio-
lated constraint is fortuitously satisfied an immediate return to FEAS
is made and a new entry problem is begun, including the newly satisfied
constraint in the set of constraints active for the entry problem. Thus
the entry problem can result in a series of NLP problems being partly
solved.

FEAS also contains tests that indicate when a problem does not
contain a feasible starting point. Such information is printed out, and
control is returned to MAIN with indicators set so that MAIN begins to
try the next NLP problem in a stack of problems. When the entry problem
is not a convex programming problem the test indicates only that the
program is in a region from which it will be unable to locate a feasible
starting point. The user, by supplying another starting point, may
cause the algorithm to generate another sequence of points that does
lead to a feasible starting point. Figure 17 shows FEAS.

6.10 FINAL

Subroutine FINAL contains the test used to determine if a point
satisfies the final convergence criterion of the algorithm. If a point
does satisfy the criterion chosen by use of option 5 then N2, the argu-
ment of this routine, is given a value of 1; otherwise it is given a
value of 2. FINAL is called following the computation of the solution
estimates, which are made after the solution of the subproblem. FINAL
appears as Figure 18.

6.11 GRAD

The gradient of the W function is given by the formula
\[ W(x, r) = Vf(x) - \sum_{i=1}^{m} \frac{r}{g_i(x)} Vg_i(x) \sum_{i=m+1}^{m+p} \frac{2h_i(x)}{r} Vh_i(x). \]
SUBROUTINE FEAS
C
AUGUST 1971
C
FEAS DETERMINES WHETHER THE STARTING POINT IS FEASIBLE, IF IT IS NOT, FEAS LOOKS FOR A FEASIBLE ONE. IF NONE EXISTS, A MESSAGE IS PRINTED AND CONTROL RETURNS TO MAIN.
C
IMPLICIT REAL*BIA-HO-ZL
REAL04 RHOIN,RATIO,EPSPHETAO
COMMON/SHAREIX/E,SIhDEL(4S)A(.5,*5,NNMNNP*N11
COMMON/OPTNS/NTINT2,NT3,NT4,NT5,NT6,NT7,NT8,NT9,NT10
COMMON/VALUE/FGPORSIGNAeRJ(90|,RHO
COMMON/CRSTIOELX44S3,DELXOI4ShRHOINRATIOEPSI THETAO,
1RSIG1,G1,X1,E453,21453,X31453,XR21453,XR11453,PR1,
2PR2,P1,F1,K110V0,10T1,PGAR1453,DIAS1453,
3 PREV3,ADELX, NCIR, NUMINI, NPHASE, NSATIS
GO TO (10,1501 NTZ
1 DO 30 I-1,N
2 IF IXllI) 20v20O30
3 XII6I.E-0S
4 CONTINUE
5 GO TO (ISO,403. NFIX
6 NPHASE.*
7 CALL EVALU
8 NPHASE71
9 :RITE
10 CALL OUTPUT
11 IF IMI 90.90,60
12 IF IRJI413 100,100,70
13 CONTINUE
14 CALL TIME
15 WRITE (6,140)
16 G=0.0
17 CALL RESTNT (0,F)
18 CALL OUTPUT (2)
19 90 RETURN
20 CALL BODY
21 IF(NPHASE .EQ. 5) RETURN
22 DO 110 I=1,N
23 IF (KJII) 120,120110
24 IF (KJII) 100,100,70
25 CONTINUE
26 CALL TIME
27 WRITE (6,140)
28 130 RETURN
29 CALL BODY
30 IF (KJII) 120,120,110
31 CONTINUE
32 GO TO 90
33 WRITE (6,150)
34 C TO INDICATE TO MAIN TO START ON NEXT PROBLEM.
35 NPHASE=5
36 GO TO 90
37 C
38 FORMAT (110,2X,40MMMADE VIOLATED NON-NEGATIVITIES SLIGHTLY POSITIVEFEA00560)
39 )
40 FORMAT (51MMTHE FEASIBLE STARTING POINT TO BE USED IS ...)
41 END
42 FORMAT (3X,09MHIS PROBLEM POSSESSES NO FEASIBLE STARTING POINT, WFEA00590)
43 END
44 Figure 17.--Subroutine FEAS.
SUBROUTINE FINAL (N2)
C
C OCTOBER 1970
C
C FINAL CONTAINS THE TESTS USED TO DETERMINE WHETHER A POINT SATISFIES
C THE FINAL CONVERGENCE CRITERION CHOOSEN TO DETERMINE IF THE NLP
C PROBLEM HAS BEEN SOLVED.
C N2 SET EQUAL TO 1 IF CONVERGENCE CRITERION IS SATISFIED.
C N2 SET EQUAL TO 2 OTHERWISE.
C
0002 REAL+ RHONRATIOEPSI THETA0
0003 COMMON/SHARE/R(45),DEL(45),X(45),Z(45),N,NH,M,M,NPT,NMI
0005 COMMON/VALUE/,GN, opioid, Z(45), RHONRATIOEPSI THETA0,
0006 COMMON/CAST/DEL(45),DEL(45),RHONRATIOEPSI THETA0,
0007 EPSI, THETA0, Del(45), X(45), Z(45), N, NPT, NMI
0009 3 PREV.ADELX. NUMIN1 NUMASE NSATIS
0010 GO TO 10,20,30, N75
0012 IF (EPSI THETA0) 50,50,70
0013 IF (EPSI THETA0) 50,50,70
0014 IF (EPSI THETA0) 50,50,70
0016 GO TO 80,40, N75
0018 CALL PUNCH
0020 RETURN
0021 END

Figure 18.--Subroutine FINAL.
Subroutine GRAD calls the user-supplied subroutine GRAD1 to compute \( Vf(x) \), \( Vg_i(x) \), and \( Vh_i(x) \) and performs the computations to evaluate \( VW(x, r) \). The negative of the gradient of the \( W \) function (i.e., \(-VW\)) is left in the array DELX0 when GRAD returns control to the calling routine. When the argument of GRAD has a value of 2 this is all that is done. However, when it has a value of 1, GRAD also does part of the computations needed to evaluate the matrix of second partial derivatives of \( W \) at \( x \). Subroutine GRAD is shown in Figure 19.

6.12 INVERS

When Newton's method is used to minimize the \( W \) function for a given value of \( r \), it maps the negative of the gradient of \( W \) with the inverse of the matrix of second partial derivatives of \( W \) evaluated at \( x \). A positive definite matrix will always give a vector along which the value of \( W \) will initially decrease. That is, for iteration \( i \), a move is made along the vector \( S^i \), given by the formula

\[
S^i = -\left[V^2W(x, r)\right]^{-1} VW(x, r).
\]

This is equivalent to solving the set of simultaneous linear equations

\[
[V^2W(x, r)]S^i = -VW(x, r)
\]

for \( S^i \), where \( V^2W \) and \( VW \) are respectively the matrix of second derivatives and gradient vector of \( W \) evaluated at \( (x, r) \).

Subroutine INVERS solves this set of equations for \( S^i \) using an \( L-U \) decomposition method (the Crout procedure). If it is determined that the matrix of second partials is not positive definite, a different procedure is used to obtain a direction \( S^i \). A complete discussion of this procedure is given on page 167 of Fiacco and McCormick [10]. When

*In the program the vector \( S^i \) is called DELX.*
SUBROUTINE GRAD (15)
C
C OCTOBER 1970
C
C GRAD COMPUTES THE GRADIENT OF THE PENALTY FUNCTION AND THE OUTER
C PRODUCT FACTORS OF THE MATRIX OF SECOND PARTIALS OF P.
C IF (IS=1) ACCUM. MATRIX OF 2ND PARTIALS IF (IS=2) DONT
C
C IMPLICIT REAL*8, I implicat reaI*8. N, M.

002 130 DO 140 I = 1, N
003 140 150 CONTINUE
004 150 160 CONTINUE
005 160 ICX = ICX + 1
006 170 CONTINUE

007 180 IF (N*7 + 180) GO TO 250 GRAOO190
008 190 DO 200 I = 1, N
009 200 CONTINUE

010 210 DO 220 J = 1, M
011 220 CONTINUE

012 230 DO 240 K = 1, M
013 240 CONTINUE

014 250 DO 260 L = 1, M
015 260 CONTINUE

016 270 DO 280 M = 1, M
017 280 CONTINUE

018 290 DO 300 N = 1, N
019 300 CONTINUE

020 310 CONTINUE

021 320 DO 330 K = 1, M
022 330 CONTINUE

023 340 DO 350 L = 1, M
024 350 CONTINUE

025 360 DO 370 M = 1, M
026 370 CONTINUE

027 380 DO 390 N = 1, N
028 390 CONTINUE

C THIS SECTION WORKS CORRECTLY IN FEASIBILITY PHASE AS WELL AS NORMAL
C

030 400 DO 410 I = 1, N
031 410 CONTINUE

032 420 DO 430 J = 1, M
033 430 CONTINUE

034 440 DO 450 K = 1, M
035 450 CONTINUE

C IF DEL(1) IS NOT A VIOLATED CONSTRAINT GRAD ADDS TO OBJ. FUNCTION
C

040 460 DO 470 J = 1, M
041 470 CONTINUE

C EQUALITY CHANGES FOR GRAD
C

045 480 DO 490 J = 1, M
046 490 CONTINUE

C LEAVES THE NEG. GRAD OF P IN DELX
C

050 500 DO 510 J = 1, M
051 510 CONTINUE

052 520 DO 530 L = 1, M
053 530 CONTINUE

054 540 DO 550 M = 1, M
055 550 CONTINUE

C LEAVES THE NEG. GRAD OF P IN DELX
C

059 560 DO 570 J = 1, M
060 570 CONTINUE

061 580 DO 590 L = 1, M
062 590 CONTINUE

063 600 DO 610 M = 1, M
064 610 CONTINUE

C LEAVES THE NEG. GRAD OF P IN DELX
C

068 620 DO 630 J = 1, M
069 630 CONTINUE

C LEAVES THE NEG. GRAD OF P IN DELX
C

073 640 DO 650 J = 1, M
074 650 CONTINUE

C LEAVES THE NEG. GRAD OF P IN DELX
C

END

078 660 DO 670 J = 1, M
079 670 CONTINUE

Figure 19.—Subroutine GRAD.
this occurs the program prints out "ORTHOGONAL MOVE." This also warns the user that the problem is probably not a convex program. Subroutine INVERS appears as Figure 20.

6.13 LMULT

Subroutine LMULT, called by subroutine SENS, calculates the Lagrange multiplier sensitivities according to steps 5 and 6 of Algorithm 2.1 (Section 2). LMULT is given in Figure 21.

6.14 MAIN

MAIN is the program that initiates the SENSUMT algorithm to solve a nonlinear programming (NLP) problem; it is not a subroutine. The input of parameters and options, as well as a starting point, is done in MAIN, and the call to READIN (a user-supplied subroutine) is made to allow the user to read in data needed to evaluate his objective function and constraint functions. Starting point data or blank cards should always be supplied by the user because starting point data cards are always read in MAIN. Subroutine FEAS is called to obtain a feasible starting point by making use of the SENSUMT algorithm to solve the entry problem if the user-supplied point is not feasible. The actual solution of the NLP problem using the SENSUMT algorithm is supervised by subroutine BØDY. The calls to SENS (sensitivity subroutine) and BØUND (bound subroutine) are also done in MAIN. When calculating bounds, after the solution and sensitivity analysis of the unperturbed problem, MAIN reiterates SENSUMT to solve the subject problem with the perturbed data. Perturbations and readjustments in the problem data are done in PERT (perturbation subroutine), on call by MAIN. Subroutine MAIN is shown as Figure 22.

6.15 ØPT

The purpose of subroutine ØPT is to obtain a $\bar{\theta} > 0$ such that $W(\bar{x} + \bar{\theta}\bar{s})$ is a minimum with respect to $\theta$ along the vector $\bar{x} + \theta\bar{s}$.
**Figure 20.—Subroutine INVERS.**
C --- COMPUTE ORTHOGONAL MOVE

DO 350 II=1,N

IF (II .GE. I) GO TO 350

BI{II} = 0.0
GO TO 350

IF (II .GE. I) GO TO 350

Bi{II} = 0.0
GO TO 350

IF (II .GE. I) GO TO 350

Bi{II} = 0.0
GO TO 350

DO 340 J=IPI{I}+1,N

Bi{II} = Bi{II} - Ai{I}*Bi{J}
GO TO 350

IF (IPL.GT.N) GO TO 350

DO 340 J=IPI{I}+1,N

Bi{II} = Bi{II} - Ai{I}*Bi{J}
GO TO 350

IF (IPL.GT.N) GO TO 350

DO 340 J=IPI{I}+1,N

Bi{II} = Bi{II} - Ai{I}*Bi{J}
GO TO 350

END

C --- CHECK MAYBE DO DIFF FOR P.S.O.

DO 370 I=1,N

DG 370 I=1,N

DO 370 I=1,N

DG 370 I=1,N

IF (IPL.GT.N) GO TO 250

C MCC ZANGWILL. ONE MOD

IF (IPL.GT.N) GO TO 250

END

C

WRITE 16,4501

FORMAT (TE17.8)

FORMAT (1HO,6X,12D06 P VECTOR)

FORMAT (1HO,6X,2SH06 SECOND ORDER MOVE VECTOR)

FORMAT (1HO,6X,1SHORT ORTHOGONAL MOVE)

END

Figure 20.--continued
SUBROUTINE LMULT(ING,DELMU,DEM,DU)

1 MARCH 1976

SUBROUTINE LMULT IS USED TO ESTIMATE THE PARTIAL DERIVATIVES OF THE LAGRANGE MULTIPLIERS. IT IS CALLED BY SENS WHEN NEXOP = 1. USING LMU00090

THE DIFFERENTIABILITY OF L21, IT TAKES ADVANTAGE OF THE RELATIONS LUI1 = RHO/111902 AND WIJ = 2WIJ/11190, DIFFERENTIATING THEM WITH LMU00120

RESPECT TO THE PARAMETERS. THIS SUBROUTINE WAS CODED BY R.L. ARMACOSTLMOU00130

IMPLICIT REALA-H.0-11

COMMON/SHARE/XIASDELS),A145.4S),N.M,PRNNONNI

COMMON/VALUE/FoGPOPRSI

COMMON/EQAL/

INTCTR.NUMINIX1451,X21451,X31451,1R21451,XR11451,PR2,PL,FER41903:DOTTPGRAO(451,DIAG44SI,

DIMENSION DELMUI9O),0Ul901,KTEST1As1

MMZ= 0

GO TO 1.2*3,3it
END

C IND = 0

DO 100 I=1,MMZ

RETURN

C IND = 1

DO 50 I=1,MMZ

DELMU(I) = 0.

RETURN

C IND = 2

DO 60 I=1,MMZ

DELMU(I) = RJ(I)

RETURN

DO 70 I=1,MMZ

DELMU(I) = (DELMU(I) - RJ(I))/DEM

RETURN

DO 30 I=1,MMZ

CALL GRAD(I)

CALL RESTNT(VAL)

SUM = 0.

DO 24 J=1,NN

SUM = SUM + DELXJ*DELXJJ

IF (IND.EQ.4) GO TO 80

DO 27 J=1,NN

SUM = SUM + DELXJ*DELXJJ

DU(I) = (SUM + DELMU(I))/RHO/VAL**2

GO TO 70

END
MAIN IS THE PROGRAM THAT INITIATES THE SUM ALGORITHM. THE INPUT OF
PARAMETERS, OPTIONS, AND STARTING POINT IS DONE IN MAIN AFTER THE
SOLUTION OF ONE M.P. PROBLEM MAIN LOCKS FOR DATA FOR ANOTHER M.P. PROB.
MAIN HAS BEEN MODIFIED BY AMMOCOST (1976) TO INCLUDE SENSITIVITY
ROUTINES AND BY SHAREL (1976) TO INCLUDE BOUND CALCULATIONS.

FOR I = 1 to 41
MAIN DATE = 02/24/82

0001 IMPLICIT REAL*4(EA+H=2-3)
0002 REAL*4 RH110,RH120,EP1,EP2,THETA0,
0003 COMM/SHARE/6415,DEL/451,455,N,M,N,M,N,P,NI1
0005 COMM/NEG/*: M, NI2, NT2, NT2, NT2, NT2, NT2, NT2, NT2, NT2
0006 COMM/VALUE/FG, PG, SI, THETA0, RHO
0007 COMM/CLIOOF/DEL/451,DEL/451,455,THETA0, RHOIN, RADIUS, EPS1, THETA0,
0008 RI51, GL, X1451, X2451, X3451, X2451, X1451, PR1
0009 ZP2, P1, P1, P1, P1, P1, P1, P1, P1, P1, D1, DIFF, PCAID(451), D1, D1, D1
0010 3 PREV3, ADEX, NTCTR, NMIN1, NHPHASE, NSATIS
0011 COMM/NE/PORT(451), OPAR(451), NPAR, ISENS
0012 COMM/NE/PORT/NEKOP1, NEKOP2, NEXOP3, NEXOP4, NEXOP5, NEKOP1, NEKOP2
0013 COMM/ARG/Y, L1, L2, L3, L4
0017 DIRECTION(X(451)
0018 PARAMETER CARD
0019 10 READ(5,50,END=40) EPS1, RH110, THETA0, RATIO, THMAK, N, M
0020 VAR=0
0021 X=RH110
0022 20 CONTINUE
0023 IF(EP1.EQ.0.) GO TO 15
0024 IF(NEKOP3.EQ.0.) GO TO 40
0025 15 CALL SET (THMAK)
0026 C INITIAL X VECTOR CARD FORMAT
0027 IF(THMAK.EQ.1) GO TO 200
0028 READ (5,60) (X(I),I=1,N)
0029 DO 4000 (I=1,N)
0030 4000 CONTINUE
0031 VAR=1
0032 200 RH110=VAR
0033 NTCTR=0
0034 N=0
0035 N1=N
0036 CALL READIN IS UNDER PROGRAMMER CONTROL
0037 IF(EP1.EQ.0.) GO TO 210
0038 C CALL READIN
0039 C OPTION CARD FOLLOWS PROGRAMMERS DATA
0040 READ (5,50) NT1, NT2, NT3, NT4, NT5, NT6, NT7, NT8, NT9, NT10
0041 GO TO 215
0042 210 CALL READIN
0043 CALL PERT
0044 IF(EP1.EQ.0.) GO TO 40
0045 IF(NEKOP3.EQ.0.) GO TO 40
0046 215 WRITE (6,110)
0047 WRITE (6,120) N, M, N, THMAK, RH110, RADIUS, EPS1, THETA0
0048 WRITE (6,130)
0049 WRITE (6,140)
0050 WRITE (6,150) N, M, N, NT1, NT2, NT3, NT4, NT5, NT6, NT7, NT8, NT9, NT10
0051 C—READ FULVALUES
0052 IF(EP1.EQ.0.) GO TO 220
0053 READ (5,60) NEKOP1, NEKOP2, NEKOP3, NEKOP4, NEKOP5, NEKOP6, NEKOP7
0054 WRITE (6,160)
0055 C—READ FULVALUES
0056 WRITE (6,170) NEKOP1, NEKOP2, NEKOP3, NEKOP4, NEKOP5, NEKOP6, NEKOP7
0057 CALL TIMEC
0058 NHPHASE=0
0059 C—JUST TO GET AN INITIAL PRINTOUT
0060 CALL EVALU
0061 PO=0.0
0062 G=0.0
0063 W=0.0
0064 RSIGMA=0.0
0065 CALL OUTPUT (1)
0066 CALL STORE
0067 IF(NEKOP1.EQ.1) CALL CHECKER
0068 IF(NEKOP1.EQ.3) STOP 01072
0069 IF(NEKOP1.EQ.5) STOP 01104
0070 CALL FEAS
0071 NPMASS=9
0072 C NPMASS=9 INDICATES EITHER THE RUN OUT OR NO FEASIBLE POINT WAS FOUND
0073 GO TO 30,30,30,10,10, NHPHASE
0074 10 NHPHASE=0

Figure 22.—MAIN
0069 TCTR=0
0070 CALL EXIT
0071 IF(NPHASE.EQ.5) GO TO 17
0072 IF(NEXUP3.EQ.11) GO TO 18
0073 IF(NEXUP3.EQ.31) GO TO 19
0074 IF(I EXUP3.EQ.31) GO TO 19
0075 IF(INT8.EQ.5) CALL PROF
0076 18 ISENS = 4
0077 CALL SENS
0078 IF(LY.LT.0.1) CALL BOUND
0079 GO TO 17
0080 19 ISENS = 2
0081 GO 21 I=1,12
0082 CALL SENS
0083 IF(LY.LT.0.1) CALL BOUND
0084 GO TO 17
0085 21 CONTINUE
0086 17 GO 300 I=1,N
0087 300 CONTINUE
0088 300 CONTINUE
0089 GO TO 20
0090 40 STOP
0091 C PARAMETER CARD
0092 50 FORMAT (5E12.0,3I4)
0093 C INITIAL X VECTOR CARD FORMAT
0094 70 FORMAT (5E12.0,3I4)
0095 C OPTION CARD FORMAT
0096 80 FORMAT (1017)
0097 90 FORMAT(13HD TOLERANCES )
0098 100 FORMAT (26N0 SECOND SET OF OPTIONS )
0099 110 FORMAT (56HI NONLINEAR PROGRAMMING ROUTINE-SUNT VERSION 4 03/01/77)
0100 120 FORMAT (110,H5,KX,2NM=13,6X,2NM=13,6X,3NMZ=13/8X,10MAX. TIME=E14.7)
0101 130 FORMAT (26N0 OPTIONS SELECTED )
0102 C
0103 C
0104 C
0105 C
0106 Figure 22.--continued
<table>
<thead>
<tr>
<th>UNCLASSIFIED SERIAL-T-434</th>
<th>ARO-16229.9-M NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The vector $\mathbf{s}$, along which the search for the minimum is made, is determined in subroutine XMOVE. The method used to locate the minimum along the vector is the Golden Section search method, which is based on the Fibonacci search method (see [10, p. 193]).

The Golden Section method is a one-dimensional search method to find the minimum of the function that does not require the computation of derivatives. Figure 23 shows $\phi_{PT}$.

6.16 OUTPUT

Subroutine OUTPUT contains most of the "write" statements used to print out information on the results of solving an NLP problem. It is used to print out information after each iteration and also to print out the solution estimates and the estimates of the Lagrange multipliers. OUTPUT is given in Figure 24.

6.17 PARDIF

The subroutine PARDIF is called by subroutine SENS to assign a differencing increment for use in the central differencing formulas indicated in Steps 2 and 4 of Algorithm 2.1. It assigns a fixed value to the differencing interval when the sensitivity analysis is conducted at the final subproblem, or assigns a value over a specified interval for sensitivity analysis at the final subproblem. When a trajectory sensitivity analysis is performed, PARDIF returns a differencing interval which is dependent on the particular subproblem involved, refining the value as the subproblem approaches the final one. Subroutine PARDIF is shown in Figure 25.

6.18 PERT

Subroutine PERT, called by the program MAIN, performs the bookkeeping regarding various parametric changes and adjustments required in the process of bound calculation. Figure 26 is subroutine PERT.
FORTRAN IV G LEVEL 21
OPT
DATE = 80243
10/14/70

0001 SUBROUTINE OPT
C MARCH 1971
C OPT LOOKS FOR A MINIMUM ALONG THE SEARCH VECTOR USING THE GOLDEN
C SECTION SEARCH METHOD.
0002 IMLPLICIT REAL*4,A-H,O-Z
0003 REAL K,O,RATI0,EP1,THETA
0004 COMMON SHAPE,ES/451,DELX/451,ES+451,ES,ES+451,SA,SA,ES
0005 COMMON VALUE,F,G,P,PS,GAMMA,X1,M0,451
0006 COMMON /CSR/ DELX/451,DELX/451,SH/451,SH/451,SHRX/451,
0007 ZP2,P1,X11111011011,OUT,2451145114511451145114511451
0008 3 PREV3,DELX, NICT, NUMINE, NPHASE, NSATIS
0009 K=1
0010 N405=1
0011 P1=0
0012 IS=0
0013 CALL DOTT
0014 GO TO 10
0015 20 DO 30 I=1,N
0016 30 DELXI=DELXI
0017 40 CONTINUE
0018 N404=0
0019 N=NUMNI
0020 C MN IS NUM NOWN OF POINTS AFTER MIN ACHIEVED
0021 NICT=NICT+1
0022 DO 50 I=1,N
0023 50 X2(I)=X2(I)
0024 PX1=0
0025 N401=0
0026 DO 70 J=1,N
0027 70 X1(J)=X2(I)+DELX(J)
0028 CALL EVALU
0029 C I MEANS SATIS,DF CONSTRAINT NT.PFEV. ZMEANS INCHANGE MEANS VIOLATION
0030 C IF POINT IS NOT FEASIBLE GIVE IT AN ARBITRARILY HIGH VALUE
0031 GO TO 50
0032 80 PX2=10.E35
0033 PX=10.E35
0034 GU TO 100
0035 90 CONTINUE
0036 PX2=0
0037 IF (PX1-PX2) 100,100,150
0038 100 IF (N401-2) 110,110,110
0039 DO 120 I=1,N
0040 120 X1(I)=X1(I)
0041 P1=PX
0042 GO TO 40
0043 CALL ONE POINT SO FAR COMPUTED
0044 GO TO 130
0045 DO 140 I=1,N
0046 140 X2(I)=X2(I)
0047 PREV=PX1
0048 GO TO 180
0049 150 DO 160 I=1,N
0050 160 X1(I)=X2(I)
0051 170 DELXI=1.0333999*DELXI
0052 PREV=PX1
0053 PX=PX2
0054 GU TO 60
0055 C GOLDEN SECTION SEARCH METHOD.
0056 C B VECTOR GOES TO X1(I)
0057 180 190 X2(I)=X1(I)
0058 200 X1(I)=X1(I)
0059 210 CALL EVALU
0060 GO TO 960
0061 CALL EVALU
0062 220 GU TO (X405,E70,210), NSATIS
0063 230 CALL EVALU
0064 CALL EVALU
0065 210 IF (N405.E70,300) GO TO 170
0066 300 CONTINUE
C THERE IS NO REFERENCE TO 211, THE ABOVE STATEMENT IS A DUMMY STATEMENT.
C-- IT IS POSSIBLE NO FEASIBLE POINT EXISTS. IF NOT TRY MOVING ON DELXI
C IF IT IS NOT POSSIBLE TO MOVE ON DELXI THEN WE MUST BE AT A
C-- SOLUTION OF MIP PROBLEM.
C-- SUBLATION OF NEP PROBLEM.
0067 220 IF (X405.E70,100) GU TO 240
0068 230 DO 320 I=1,N
0069 320 IF (X405.E70,310) 330,330,330
0070 330 CONTINUE
0071 240 GU TO 1290,2801, N405

Figure 23.--Subroutine OPT.
Figure 23. -- continued

- 88 -
SUBROUTINE OUTPUT (I)

REAL*4 RHOIN, RATIO, EPSI, THETAO

COMMON /EQAL/ H0, H1, NZ

COMMON /OPTNS/ NT1, NT2, NT3, NT4, NT5, NT6, NT7, NT8, NT9, NT10

COMMON /VALUE/ FVPO, SIGMA, R

CON/4ON/ABG/LYL, PERI

CON/4ON/ABG1/FEL, FE2

CON/4ON/ABG2/F(45), DF(45)

COMMON/ABG3/TX(45), DELX(45)

COMMON/ABG4/X(45)

COMMON/ABG5/NEXOP6

COMMON/ABG6/NZ

GO TO 110

WRITE (6,601) NTCTR, RHOIN, RATIO, EPSI, THETAO

WRITE (6,80) NTCTR, RHOIN, RATIO, EPSI, THETAO

IF (FALT > 1.0) GOTO 200

DO 210 I = 1, N

WRITE (6,90) I, RHOIN, RATIO

GO TO 50

WRITE (6,100) I, RHOIN

RETURN

C CALCULATION OF GP VARIABLES

IF (FALT < 0.0) GOTO 310

WRITE (6,290) I, X(I) / 1.0

WRITE (6,120) I, RHOIN

GO TO 50

WRITE (6,130) I, RHOIN

RETURN

C FORTRAN IV LEVEL 21 OUTPUT DATE = 08242 12/12/70

FORTRAN IV LEVEL 21 OUTPUT DATE = 08242 12/12/70

Figure 24.—Subroutine OUTPUT.
6.19 PEVALU

Subroutine PEVALU is used to compute the value of the \( W \) function and the \( G \) function. It makes use of the values of the objective function and the constraint function, which must have been computed before PEVALU is called. The \( G \) function is interpreted as the value of the dual objective function when the \( W \) function has been minimized for a given value of \( r \) (RHO). PEVALU is given as Figure 27.
Figure 27.--Subroutine PEVALU.

6.20 PRESEN

This subroutine calculates the gradient of the optimal value function, using the gradient of the Lagrangian when taken with respect to the parameters (Step 9 of Algorithm 2.1). In addition, PRESEN (when invoked) performs a preliminary screening of the problem parameters to which the optimal value function is practically insensitive. The criterion used for this purpose is that if the change in optimal value function is less than 0.1 percent of its current value as a result of the introduced increment of a given parameter, then that parameter is
eliminated from further consideration. This parameter screening is invoked when the fifth variable in the second option card takes a value of zero, as shown in Table 4. Figure 28 lists PRESEN.

6.21 PUNCH

When a call to PUNCH is made, the current value of the array \( x \) is punched, along with some of the control cards that can be used to restart SENSUMT. Subroutine PUNCH is shown in Figure 29.

6.22 REJECT

Subroutine REJECT puts values of the point at \( x \) and the associated values of the objective function, constraint functions, and \( W \) function back into their normal locations. These values were temporarily stored in other locations by subroutine STORE. STORE appears as Figure 30.

6.23 RHOCOM

This subroutine is used to compute the initial value of \( r \) (RH\( \Phi \)) for each NLP problem. As previously stated, the search for a feasible starting point can result in a sequence of NLP problems. The initial value of \( r \) is computed by the formula specified for the use of option 1 (NT1). RHOCOM is given as Figure 31.

6.24 SECOND

SECOND is used to query about the computer's clock. This is made possible by an internal clock which is called by this subroutine. The elapsed time obtained and monitored by this subroutine is used by the subroutines TIMEC, TCHECK, and SET. Figure 32 is subroutine SECOND.
FORTRAN IV G LEVEL 21
PRESEN DATE = 80242 12/13/50

0001 SUBROUTINE PRESEN(DU,KTEST) PRE00040
0002 C 1 MARCH 1976 PRE00070
0003 C SUBROUTINE PRESEN IS USED TO CALCULATE THE GRADIENT OF THE OPTIMAL
0004 C VALUE FUNCTION USING THE GRADIENT OF THE LAGRANGIAN TAKEN WITH
0005 C RESPECT TO THE PARAMETERS. PRESEN IS CALLED BY SENS WHEN VARIABLE
0006 C NEQPS = 0 OR = 1. ADDITIONALLY, WHEN NEQPS = 0, THE PARAMETERS WHICH
0007 C AFFECT THE OPTIMAL VALUE FUNCTION BY LESS THAN 0.001 TIMES ITS
0008 C CURRENT VALUE ARE ELIMINATED FURTHER CONSIDERATION. THIS
0009 C SUBROUTINE WAS CODED BY R. L. ARMACOST. PRE01160
0010 IMPLICIT REAL*4(A-H,O-Z) PRE00170
0011 COMMON/SHARX/I(45),DEL(45),J(45),N,N,M,N,P1,N1,NI
0012 COMMON/VALU*ES,F,F,P,J,ASigma,RAJ(90),RH
0013 COMMON/EN/PAR(45),NPAR(45),SENS
0014 COMMON/AGB,L,PER(45)
0015 COMMON/AGB1/PAR1(FE1,FE2)
0016 COMMON/AGB2/OF4(45,OF2(45)
0017 DIMENSION GX(90),GXI(90),KTEST(45),KLIST(45)
0018 NPAR = N * M
0019 FILE = 0.031
0020 PTES = 0.001 * DABS(FI)
0021 DO 100 J = 1, NPAR
0022 RTST(J) = 0 FILE
0023 PARAM = PARAM + DPAR(J)
0024 CALL RESTT(I,DF)
0025 IFMPLZ.EQ.01 GO TO 20
0026 DO 10 J = 1, NPAR
0027 CALL RESTT(I,GXI(J))
0028 30 CONTINUE
0029 DFEPS = (DF - XF) / XF
0030 IFMPLZ.EQ.01 GO TO 60
0031 SUM = DFEPS
0032 IFMPLZ.EQ.01 GO TO 80
0033 DO 70 J = 1, M
0034 DO 70 J = 1, M
0035 SUM = SUM - MX(RHI(J)*DUIJ)
0036 DO 70 J = 1, M
0037 SUM = SUM - MX(RHI(J)*DUIJ)
0038 SUM = SUM + MX(RHI(J)*DUIJ)
0039 DEL(J) = SUM
0040 CALL DPAR(J) = PARAM + DPAR(J)
0041 DTST = DABS(DEL(J))
0042 IFKSTT.GE.FTEST.TKST(T) = 1
0043 ELSE WRITE(*,6400)
0044 6400 FORMAT(1060) PRE00620
0045 600 FORMAT(22,4H0OPTIMAL VALUE FUNCTION SENSITIVITY
0046 400 CONTINUE
0047 400 IFDFLY.TE.GE.0.1 GO TO 700
0048 700 JJ = 0
0049 700 JJ = 0
0050 700 JJ = 0
0051 700 JJ = 0
0052 700 JJ = 0
0053 700 JJ = 0
0054 700 JJ = 0
0055 700 JJ = 0
0056 700 JJ = 0
0057 700 JJ = 0
0058 700 JJ = 0
0059 700 JJ = 0
0060 700 JJ = 0
0061 700 JJ = 0
0062 700 JJ = 0
0063 700 JJ = 0
0064 700 JJ = 0
0065 700 JJ = 0
0066 700 JJ = 0
0067 700 JJ = 0
0068 700 JJ = 0
0069 700 JJ = 0
0070 700 JJ = 0
0071 700 JJ = 0
0072 700 JJ = 0
0073 700 JJ = 0
0074 700 JJ = 0
0075 700 JJ = 0
0076 700 JJ = 0
0077 700 JJ = 0
0078 700 JJ = 0
0079 700 JJ = 0
0080 700 JJ = 0
0081 700 JJ = 0
0082 700 JJ = 0

Figure 28.—Subroutine PRESEN.
Figure 29.--Subroutine PUNCH.

Figure 30.--Subroutine REJECT.
Figure 3.1—Subroutine RHOCOM.
Subroutine **SECOND** evaluates the matrix of second partials of the \( W \) function. It calls on the user-supplied subroutine **MATRIX** to evaluate the upper triangle of the matrix of the second partials of the objective and the constraint functions. Subroutine **SECOND** is listed in Figure 33.

**SENS**

This subroutine calculates the solution point and optimal value function sensitivities for each subproblem (if required), as well as the final problem, by implementing Steps 1-3 and 7-8 of Algorithm 2.1 of Section 2. The various choices for sensitivity calculations are given in options 3, 4, and 5 of the second option card (see Table 4). SENS is called by the program MAIN and the subroutine BODY. It calls subroutine **LMULT** to calculate the Lagrange multiplier sensitivities. SENS is shown as Figure 34.

**SET**

Subroutine **SET** is called once by MAIN at the start of the attempt to solve a problem. It obtains and stores the value of the computer's clock. It obtains the value of the clock by calling a system library or user coded subroutine that queries the computer's clock and returns its value as a floating point number in seconds. Figure 35 lists subroutine **SET**.
Figure 33.—Subroutine SECORD.
0074 DD 300 J=1, I
0075 IF (A(J,1)) .NE. 0.0 GO TO 0085
0076 300 CONTINUE
0077 300 CONTINUE
0078 300 CONTINUE
0079 DU 320 I=1, N
0080 IF (A(I,1)) .LE. 0.0 GO TO 0090
0081 310 DIAG(I)=DIAG(I)+TA(I,1)
0082 320 CONTINUE
0083 330 CONTINUE
0084 340 CONTINUE
0085 350 IF (ML.EQ.0) GO TO 420
0086 GO TO (520, 350, 520, NPHASE)
0087 350 CONTINUE
0088 IF (INTO.GE.2) GO TO 420
0089 DO 610 I=1, N
0090 I=I+1
0091 LORN=2
0092 CALL MATRIX (I, LORN)
0093 IF (LORN.LT.2) GO TO 410
0094 T=2.*RJNI/RHO
0095 DO 360 J=1, N
0096 J=J+1
0097 DO 370 J=1, N
0098 J=J+1
0099 360 A(I,J)=A(I,J)+T*A(J)
0100 A(I,J)=0.0
0101 370 CONTINUE
0102 380 CONTINUE
0103 DO 400 I=1, N
0104 IF (A(I,1)) .LE. 0.0 GO TO 410
0105 390 DIAG(I)=DIAG(I)+TA(I,1)
0106 A(I,1)=0.0
0107 400 CONTINUE
0108 410 CONTINUE
0109 C GET MATRIX OF 2ND PARTIALS OF OBJECTIVE FUNCTION
0110 420 LLLL=2
0111 CALL MATRIX (0, LLL)
0112 IF (LLLL.LT.2) GO TO 490
0113 DO 440 J=1, N
0114 J=J+1
0115 IF (A(J,1)) .LE. 0.0 GO TO 480
0116 430 A(J,J)=A(J,J)+T*A(J)
0117 440 A(J,J)=A(J,J)
0118 DO 470 J=1, N
0119 J=J+1
0120 450 A(J,J)=DIAG(J)+A(J,J)
0121 GO TO 470
0122 460 A(J,J)=DIAG(J)
0123 470 CONTINUE
0124 480 RETURN
0125 490 DO 500 J=1, N
0126 J=J+1
0127 500 A(J,J)=DIAG(J)
0128 GO TO 480
0129 510 DO 530 J=1, N
0129 510 GO TO (920, 390, 350, NPHASE)
0130 520 DO 530 J=2, N
0131 530 A(J,J)=A(J,J)
0132 540 A(J,J)=DIAG(J)
0133 GO TO 480
0134 550 A(J,1)=-A(J,1)
0135 560 GO TO 480
0136 570 END
0137

Figure 33.--continued
SUBROUTINE SENS

IMPLICIT REAL*4 (A-H,O-Z)
REAL*4 RHON,TAB,EPSTHETA,EPST,EPX,EPZ
COMMON/X/RAEX,DELX1,DELX2,DELX3,DELX4,DELX5,DELX6,
       /N/N1,N2,N3,N4,N5,N6
COMMON/EEJ/LSE,BSE,SE0,SE1,SE2,SE3,SE4,SE5,SE6,SE7,SE8,SE9,SE10
COMMON/LL/X1,X2,X3,X4,X5,X6
REAL*4 X1,X2,X3,X4,X5,X6
COMMON/COMMON/EQAL/HRL,H1,MZ
COMMON/OPTS/N1,N2,N3,N4,N5,N6,N7,N8,N9,N10
COMMON/W,PAR,RI,RI0,R10,PI,PI0,PI1,PI2,PI3,PI4
COMMON/OF1,OF2,OF3,OF4,OF5,OF6
COMMON/AG1,AG2,AG3,AG4,AG5,AG6
COMMON/DIF/D1,D2,D3,D4,D5,D6
COMMON/DER/D11,D12,D13,D14,D15,D16

COMMON/SH//1X,2HX,2HM
COMMON/CRST//X1,2X,2H

COMMON/CS/SHA,SH0,SHA1,SHA2,SHA3,SHA4,SHA5,SHA6,SHA7,SHA8,SHA9

COMMON/CST1,N1,T1,T2,T3,T4,T5,T6,T7

COMMON/IS/IS1,IS2,IS3,IS4,IS5,IS6,IS7,IS8,IS9,IS10,IS11,IS12,IS13,IS14,IS15,IS16

COMMON/INT/INT1,INT2,INT3,INT4,INT5,INT6,INT7,INT8,INT9,INT10,INT11,INT12,INT13,INT14,INT15,INT16,INT17,INT18,INT19,INT20

REAL*4 RHOINRATIOEPSI,THETAPHI,THETAPHI2

10 FORMAT(6X,6HTHE VALUE OF RHON IS ,E12.3)
20 FORMAT(6X,6H THE POINT AT WHICH THE ESTIMATE OF SENSITIVITY WILL BE MADE IS   )
50 FORMAT(6X,6HDIFFERENCING INTERVAL IS E14.6)
**Figure 34.---continued**

- 100 -
C: CALCULATION OF GP VARIABLE SENSITIVITY

IF(INEXOP6.NE.11) GO TO 670

IF(IN.EQ.0) GO TO 301

CALL L*JLTI44.ELMU.0EMOUI

WRITE(6,360) OF,CONT

Figure 34.--continued
6.28 STORE

Subroutine STORE stores the values of the current point $x$ and the associated values of $F$, $W$, and $G$ in a temporary storage area. These values are restored by a call to REJECT. The listing for subroutine STORE appears as Figure 36.
6.29 TCHECK

If the print option has been set so that printouts occur only after a subproblem has been called, then TCHECK is called after each iteration of the algorithm used to minimize the W function. The elapsed time is computed as is done in TIMEC but it is not printed out. If the elapsed time exceeds 90 percent of the estimated time limit, then the print option is changed so a printout occurs after each iteration of the procedure being used to minimize the W function. If the elapsed time exceeds the user specified time limit, then the routine will cause termination of the attempt to solve the problem by setting NSWW equal to 2. Subroutine TCHECK appears as Figure 37.

Figure 37.—Subroutine TCHECK.

6.30 TIMEC

Subroutine TIMEC calls the same routine as called by SET to obtain the current status of the computer's time clock. It then prints out the elapsed time since the call to SET. If the elapsed time is greater than the user specified maximum time limit, TIMEC will cause the termination of the attempt to solve the problem by setting NSWW
equal to 2. The listing for subroutine TIMEC is shown below in Figure 38.

**Figure 38.--Subroutine TIMEC.**

6.31 TRANS

TRANS is a special purpose subroutine that performs an exponential transformation of the solution vector and sensitivity results. This subroutine is useful when solving problem $c(x)$, the convex equivalent of the geometric programming problem $G(t)$. The former problem is obtained from the latter via the transformation $t = e^{-x_i}$, $i=1,...,n$. TRANS is used to transform the results to the original space of the variable $t$. The motivation for coding this subroutine was the computational difficulty experienced by some users while solving geometric programming problems with SUMT. TRANS is called by the subroutines OUTPUT and SENS. Figure 39 is a listing of subroutine TRANS.

6.32 XMOLVE

Subroutine XMOLVE contains most of the logic of the algorithm used to minimize the $W$ function for a given value of $r$ (RHO). First, a
Subroutine TRANS is chosen in which it is believed the W function will initially decrease. Then OPT is used to find a minimum of the W function in that direction. Having generated a new point giving a lower value of the W function, XM0VE returns control to BODY.

Subroutine XM0VE contains three procedures for generating a direction vector. The choice of the procedure used is controlled by experimental option 2 (NEXP2). If NEXP2 = 1, then Newton's method is used to choose the direction vector S, that

$$ S^i = -\left( V^2 W(x_i, r_k) \right)^{-1} V W(x_i) $$

If $V^2 W(x_i, r_k)$ is not a positive definite matrix, $S^i$ is generated by rules outlined on page 167 in Fiacco and McCormick [10].

If NEXP2 = 2, then the negative of the gradient is used as the direction vector, $S^i = -V W(x_i, r_k)$.

When NEXP2 = 3 a variable matrix method is used to generate $S^i$. This method is McCormick's modification of the Fletcher-Power-Davidon, as described on pages 170-175 in Fiacco and McCormick [10]. Subroutine XM0VE is shown as Figure 40.
Figure 40.—Subroutine XMOVE.
0059  DO 170 I=1,N
0060  170 DELX(I)=DELX(I)/IC1
0061  CALL STORE
0062  CALL OPT
0063     IT=IT+1
0064  RETURN
0065  180 CONTINUE
0066   C STEEPEST DESCENT
0067   CALL GRAD (2)
0068  190 DO 180 I=1,N
0069  190 DELX(I)=DELX(I)
0070  CALL STORE
0071  CALL OPT
0072  RETURN
0073  END

Figure 40.—continued
REFERENCES


To cope with the expanding technology, our society must be assured of a continuing supply of rigorously trained and educated engineers. The School of Engineering and Applied Science is completely committed to this objective.