FINAL REPORT
FOR
NASA Marshall Space Flight Center
CONTRACT NAS 8-26929

CONJUGATE GRADIENT OPTIMIZATION PROGRAMS
FOR SHUTTLE REENTRY

by

W. F. Powers, R. A. Jacobson, and D. A. Leonard

Department of Aerospace Engineering
The University of Michigan
Ann Arbor, Michigan 48104

May 1972
FINAL REPORT
FOR
NASA Marshall Space Flight Center
CONTRACT NAS 8-26929

CONJUGATE GRADIENT OPTIMIZATION PROGRAMS
FOR SHUTTLE REENTRY

by
W. F. Powers, R. A. Jacobson, and D. A. Leonard

Department of Aerospace Engineering
The University of Michigan
Ann Arbor, Michigan 48104

May 1972
ABSTRACT

Two computer programs for shuttle reentry trajectory optimization are listed and described. Both programs use the conjugate gradient method as the optimization procedure. The Phase I Program is developed in cartesian coordinates for a rotating spherical earth, and crossrange, downrange, maximum deceleration, total heating, and terminal speed and altitude are included in the performance index. The Phase II Program is developed in an Euler angle system for a nonrotating spherical earth, and crossrange, downrange, total heating, maximum heat rate, and terminal speed, altitude, and flight path angle are included in the performance index. The programs make extensive use of subroutines so that they may be easily adapted to other atmospheric trajectory optimization problems.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>The Conjugate Gradient Method</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.1 Finite-Dimensional Conjugate Gradient Method</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.2 Infinite-Dimensional Conjugate Gradient: Unconstrained</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.3 Infinite-Dimensional Conjugate Gradient: Constrained</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Phase I Program</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.1 Basic Description</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.2 Subroutine Map</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.3 Subroutine Description</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>3.4 Phase I Program Notes</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Phase I1 Program</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4.1 Basic Description</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4.2 Subroutine Map</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>4.3 Subroutine Descriptions</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>4.4 Phase I1 Program Notes</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Recommendations</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>5.1 Summary</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>5.2 Conclusions and Recommendations</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>34</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Listing of Phase I Program</td>
<td>36</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Listing of Phase I1 Program</td>
<td>67</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

The forthcoming Space Shuttle Program will involve vehicles which possess both rocket and aircraft characteristics. Because of the interplay of gravitational, thrusting, and aerodynamic forces, the trajectories that the vehicle will fly are more complicated than the trajectories of the Saturn-Apollo class. Thus, the need exists for efficient, reliable shuttle trajectory optimization programs.

This report describes two computer programs which were generated for shuttle reentry. During the time period of this contract, the national emphasis shifted from a small, low-crossrange, straight-wing orbiter to a larger, high-crossrange, delta-wing orbiter. This definitely influences the reentry trajectory in that the straight-wing trajectory usually encounters the 3g deceleration constraint whereas the delta-wing trajectory rarely (if ever) encounters the 3g-constraint but instead has high heat-rate problems. Thus, instead of making a large cumbersome program for all possible vehicles, two programs were developed. Since the Phase II-deck was developed after the Phase I-deck, it has the advantage of some improvements learned in the development of the Phase I-program.

It has been noted by numerous investigators in the last two years that shooting (or initial Lagrange multiplier guessing) iteration schemes have been almost useless in determining shuttle reentry trajectories. There exist other techniques which might be applicable to the problem and they are briefly described below:

1) Classical Gradient Method: This method iterates on the total control function and does not require any second-order information (i.e., second-derivatives of the Hamiltonian). This method is well-known for having excellent properties far away from the solution, but slow convergence near the solution. With respect to boundary conditions, either penalty functions* or projections2 may

*Numbers refer to listings in the References section.
be employed. A modified gradient projection approach for shuttle reentry is under development at TRW-Systems\textsuperscript{3}.

2) Second-Order Gradient Method: This method is essentially a function space Newton's method which iterates on the total control function and requires full second-order information. The method is described in Ref.\textsuperscript{4}, and a shuttle-version of the program is in use at NASA-Manned Spacecraft Center\textsuperscript{5}. It has been found that although this program obtains accurate trajectories and control histories, the deck is difficult to work with and modify, and requires extremely long computer time.

3) Conjugate Gradient\textsuperscript{6} and Function-Space Davidon\textsuperscript{7} Methods: These methods iterate on the total control function and do not require any second-order information. These methods are mainly motivated by deficiencies in the classical gradient and second-order gradient (or function space Newton) methods. That is, they require only first-order information and may have better convergence characteristics near the minimum than the classical gradient method. This study involved the generation of two conjugate gradient programs. It appears that the function-space Davidon method needs further analysis before it should be employed in a shuttle computer program.

4) Parameter Optimization Methods: In the last decade a number of efficient parameter optimization techniques have been popularized, e.g., conjugate gradient (CG), Davidon-Fletcher-Powell (DFP) variable metric. These schemes have proven their worth, and the DFP method is probably the most popular parameter optimization scheme in use today. Both the CG and DFP methods are available in Fortran subroutines\textsuperscript{8}. The DFP method has been applied successfully to shuttle optimization by Johnson and Kamm\textsuperscript{9,10}. They represent the control variables by sequences of straight line segments and then use DFP to iterate for the optimal slopes of the segments subject to continuity and inequality constraints. By computing their gradients numerically, the deck is easily modified to
include additional parameters, different vehicles, and various missions. Thus, for design purposes, this is a very efficient approach.

From the discussion above of the various approaches to shuttle optimization, it would appear at first glance that parameter optimization is the superior iterative procedure. For preliminary design this is probably the case. However, the parameter optimization approach requires either prior knowledge of approximate optimal control histories or an undeterminable amount of working time devoted to selecting workable but accurate representations for the controls. In reentry this may be especially difficult because a change in terminal boundary conditions may cause a completely different bank angle control, and in many cases the bank angle would require a large number of segments to approximate it adequately. Thus, the parameter optimization approach is by no means automatic or even desirable in some cases.

Because of the deficiencies noted above for the parameter optimization approach, the need still exists for a relatively flexible and efficient function space technique. At the present time it appears that both the projected gradient and the conjugate gradient methods are the leading candidates for such a scheme, and which scheme is best is probably problem dependent. For example, the projected gradient technique is probably best for problems which are strongly influenced by boundary conditions and do not contain singular arcs. The conjugate gradient technique is probably best for problems with singular arcs and/or problems which exhibit slow convergence near the minimum with a standard gradient technique. However, not as much work has been done with the conjugate gradient technique as the projected gradient technique, so improvements in the conjugate gradient approach are occurring more frequently than in the projected gradient method. It should be noted that the conjugate gradient and gradient projection technique have been combined\textsuperscript{11}, but the results were not promising. However, there may exist more efficient ways of combining the two techniques, and, thus, a "projected conjugate gradient" technique may be feasible.
In this chapter, a tutorial treatment of the conjugate gradient method will be given in both finite- and infinite-dimensional spaces. The methods for treating inequality constraints in the programs are discussed, also.

2.1 Finite-Dimensional Conjugate Gradient Method

Consider the problem of minimizing

$$f(x_1, \ldots, x_n),$$

where $x = (x_1, \ldots, x_n)$ is an element of a bounded, connected, open subset of $\mathbb{R}^n$ and $f \in C^1$. If equality and/or inequality constraints are present, it is assumed that they are incorporated into (2.1) by means of penalty functions.

Before we develop the algorithm, let us consider a few general remarks about the minimization of a quadratic function. Consider

$$q = x^T Ax,$$

where $x \in \mathbb{R}^n$, $A$ is a positive definite matrix. The contours of constant $q$-values are $n$-dimensional ellipsoids centered on the global minimum $x = 0$. In 2-space, the eccentricity of the elliptical contours is dependent upon the relative magnitudes of the eigenvalues of $A$; the contours are circular if the eigenvalues are equal and the contours become more eccentric as the ratio of the eigenvalues increases from one. Of course, similar results are true in $n$-space.

If the contours of (2.2) are noncircular, the gradient method (with a one-dimensional search) will take an infinite number of iterates to converge to the minimum if the method does not converge on the first iterate. (If the initial guess is on a principal axis of the $n$-dimensional ellipsoid, then a single gradient step results in $x = 0$.) On the other hand, no matter what the eigenvalues are, Newton's method will converge in one iterate.

The reason why the quadratic problem is of interest is that in the terminal stages of an iterative minimization of many nonlinear functions, the
the function may be well-approximated by a second-order expansion. Thus, an efficient algorithm for general functions should have good convergence characteristics for quadratic functions. As noted above for quadratic functions, Newton's method is excellent in all cases, while the gradient method is strongly problem dependent. However, Newton's method requires the computation of second-order information while the gradient method requires only first-order. In addition, for general nonlinear functions, Newton's method may diverge whereas the gradient method will, at least, never result in an iterate which increases the quantity to be minimized.

Because of the properties discussed above, researchers in the 1950's attempted to develop techniques which combined the advantages of the gradient and Newton methods while minimizing their disadvantages. With respect to the quadratic minimization problem, two techniques with the following properties were developed: (i) the methods are stable, (ii) the minimum is obtained in at most $n$ iterations, (iii) no second-order information is required. The methods are the conjugate gradient method\textsuperscript{12} and the Davidon variable metric method\textsuperscript{13} (or Davidon-Fletcher-Powell\textsuperscript{14} method).

With respect to general nonlinear functions, the methods retain properties (i) and (ii) mentioned above. For certain classes of functions, rates of convergence are known for all of the methods mentioned except the DFP method. These show that when the methods work, Newton should be faster than the CG method, and the CG method should be faster than the gradient method. However, Newton's method does not possess either property (i) or (ii) mentioned above.

The CG formula will now be stated, the sequence of steps required in the development of the formula will be outlined, and then the steps will be developed in detail. The CG algorithm is as follows:

1. Guess $x_0$. Define $g = f_x$.

2. $p_0 = g_0$, $p_{J+1} = g_{J+1} + p_J \left( \frac{g_{J+1}^T g_{J+1}}{g_J^T g_J} \right)$, $J = 0, 1, \ldots$. \hspace{1cm} (2.3)

3. $x_{J+1} = x_J - \alpha_J p_J$, $\alpha_J \geq 0$ \hspace{1cm} (2.4)
In the formula above, \( x_j \) is an n-vector, \( g_j \) is the n-vector gradient, \( p_j \) is the n-vector search direction, and \( \alpha_j \) is a scalar.

The formula development involves the following sequence of steps:

(A) Assume \( x_{j+1} = x_j - \alpha_j p_j \) with \( p_j = g_j + b_j \). and devise a means for defining \( b_j \).

(B) Show that \( g_j^T b_j = 0 \) implies the method will be stable.

(C) Show that the largest decrease in \( f \) is obtained if \( g_{j+1}^T p_j = 0 \).

(D) Note that (B) and \( b_j = C_j p_{j-1} \) imply (C), where \( C_j \) is a constant to be defined.

(E) Show that finite convergence for the quadratic function \( f = x^T A x \) is guaranteed if \( p_i^T A p_j = 0 \) (I \( \neq \) J), i.e., the search directions are "A-conjugate."

(F) Combine all of the above steps to show that

\[
  b_j = (g_j, g_j) / (g_{j-1}, g_{j-1}) p_{j-1}
\]

where \( (g_j, g_j) \equiv g_j^T g_j \) is an inner product in \( \mathbb{R}^n \). The inner product notation will be used from here on instead of the transpose notation.

Let us now develop the results noted in steps (A) to (F). First, we assume a form for the update formula

\[
  x_{j+1} = x_j - \alpha_j p_j \tag{2.6}
\]

\[
  p_j = g_j + b_j. \tag{2.7}
\]

The motivation for this form is that the method is basically a gradient method with a correction vector (i.e., \( b_j \)) which, hopefully, will aid the convergence characteristics of the gradient method in the neighborhood of the solution. The only undefined quantities in Eqs (2.6) and (2.7) are \( \alpha_j \) and \( b_j \). The scalar \( \alpha_j \) will be determined by a 1-D search in each iteration, so the only quantity which must be characterized is the n-vector \( b_j \).

**PROPERTY 1:** If

\[
  (g_j, b_j) = 0 \tag{2.8}
\]
then the method is stable.

Proof: Expand \( f(x_{J+1}) \) about \( f(x_J) \) to first-order:

\[
f(x_{J+1}) = f(x_J) + \langle g(x_J), x_{J+1} - x_J \rangle
\]

\[
f(x_{J+1}) = f(x_J) - \alpha_J \langle g_J, g_J \rangle - \alpha_J \langle g_J, b_J \rangle.
\] (2.9)

For \( \alpha_J \) small, a sufficient condition for \( f(x_{J+1}) \) to be less than or equal to \( f(x_J) \) is \( \langle g_J, b_J \rangle = 0 \).

Note that Property 1 is a sufficient condition for stability. Thus, there exist numerous possibilities for techniques which could also be stable; one need only insure that the interaction of the two terms on the right-hand side of Eq. (2.9) be negative (when the first-order expansion is valid).

PROPERTY 2: Let \( \alpha_J \) be the value of the search parameter which minimizes \( f(x_J + \alpha p_J) \). Then,

\[
\langle g_{J+1}, p_J \rangle = 0.
\] (2.10)

Proof: By definition of \( \alpha_J \):

\[
\frac{df}{d\alpha} \bigg|_{\alpha_J} = \left[ \frac{\partial f^T}{\partial x_{J+1}} \frac{\partial x_{J+1}}{\partial \alpha} \right] \alpha_J = \langle g_{J+1}, p_J \rangle = 0.
\]

PROPERTY 3: If Eq. (2.8) holds and

\[
b_J = C_J p_{J-1} \quad (J = 1, 2, \ldots)
\] (2.11)

(where \( C_J \neq 0 \) is a scalar to be defined), then Eq. (2.10) is satisfied.

Proof: By Eq. (2.8):

\[
\langle g_{J+1}, b_{J+1} \rangle = 0 \Rightarrow \langle g_{J+1}, C_{J+1} p_J \rangle = 0.
\]

Thus, Eq. (2.10) is satisfied when \( C_{J+1} \neq 0 \).

Because of Property 3, we shall assume that the correction vector, \( b_J \), is linearly related to the previous search direction, i.e., we shall assume that \( b_J \) is defined by (2.11). In this case, the only thing that remains is the characterization of the constant \( C_J \).
PROPERTY 4: Consider \( f = x^T A x \), where \( A \) is positive definite. If the search directions are \( A \)-conjugate (i.e., \( p^T_i A p_j = 0, \; i \neq j \)) and Eqs. (2.6) and (2.10) hold (or, equivalently, (2.6), (2.8), (2.11)), then the global minimum \( x = 0 \) of \( f \) is obtained in at most \( n \) iterations.

Proof: At the unique global minimum of \( f \), the gradient \( g = Ax \) must equal zero. If in the application of the algorithm either \( g_0, g_1, \ldots \), or \( g_{n-1} = 0 \), then the property is proved. Thus, assume \( g_0, \ldots, g_{n-1} \neq 0 \). At each iterate, we have

\[
g_J = A x_J.
\] (2.12)

By repeated use of Eq. (2.6) we have:

\[
x_n = x_{j+1} + \sum_{i=j+1}^{n-1} \alpha_i p_i
\]

for any \( J \in \{0, \ldots, n-2\} \). From Eq. (2.11):

\[
g_n = g_{j+1} + \sum_{i=j+1}^{n-1} \alpha_i A p_i.
\] (2.13)

The inner product of \( g_n \) and \( p_J \) is

\[
<g_n, p_J> = <g_{j+1}, p_J> + \sum_{i=j+1}^{n-1} \alpha_i <p_i, A p_J>.
\] (2.14)

The first term in this equation vanishes because of Eq. (2.10), and the summation vanishes because of the \( A \)-conjugacy property. Thus,

\[
<g_n, p_J> = 0. \; (J = 0, 1, \ldots, n-2)
\] (2.15)

By Eq. (2.10) we also have

\[
<g_n, p_{n-1}>= 0.
\] (2.16)

Equations (2.15) and (2.16) may be written in matrix form as

\[
[p_0 p_1 \cdots p_{n-1}] g_n = 0.
\] (2.17)

It can be shown that \( n \) \( A \)-conjugate vectors are linearly independent (note that \( A \)-conjugate is a generalization of orthogonality), and thus, Eq. (2.17) implies
We now have enough information to define the constant $C_J$ in Eq. (2.11).

**PROPERTY 5**: Consider $f = x^T Ax$, where $A$ is positive definite. If the update formula is defined by Eqs. (2.6) and (2.11), the search directions are $A$-conjugate, and $\alpha_J$ and $C_J$ are chosen to give the maximum decrease in the function $f$, then

$$C_J = \frac{<g_J, g_J>}{<g_{J-1}, g_{J-1}>}.$$  \hspace{1cm} (2.19)

**Proof**: At a given iteration $f$ is given by

$$f[x_J + \alpha_J g_J + \alpha_J C_J P_{J-1}].$$

At a minimum of $f$ with respect to $\alpha_J, C_J$:

$$f_{\alpha_J} = 0 \Rightarrow <g_{J+1}, P_J> = 0$$ \hspace{1cm} (2.20)

$$f_{C_J} = 0 \Rightarrow <g_{J+1}, P_{J-1}> = 0.$$ \hspace{1cm} (2.21)

Expansion of Eq. (2.20), noting $g_{J+1} = g_J + \alpha_J A p_J$, $P_J = g_J + C_J P_{J-1}$, gives

$$<g_J, g_J> + C_J <p_{J-1}, g_J> + \alpha_J <p_J, A p_J> = 0,$$

which implies

$$\alpha_J = -\frac{<g_J, g_J>}{<p_J, A p_J>}.$$ \hspace{1cm} (2.22)

Before we obtain the desired result, note that Eqs. (2.20) and (2.21) imply

$$<g_{J+1}, g_J> = 0.$$ \hspace{1cm} (2.23)

To obtain the expression for $C_J$, we first form the inner product of $g_J = P_J - C_J P_{J-1}$ with $A p_{J-1}$:

$$<g_J, A p_{J-1}> = <p_J, A p_{J-1}> - C_J <p_{J-1}, A p_{J-1}>.$$ \hspace{1cm} (2.24)

The first inner product on the right vanishes because of $A$-conjugacy. The desired result is obtained by substituting $(g_J - g_{J-1})/\alpha_{J-1}$ for $A p_{J-1}$ on the left and $<g_{J-1}, g_{J-1}>/\alpha_{J-1}$ for $-<p_{J-1}, A p_{J-1}>$ on the right:
As noted previously, the algorithm defined above, along with the Davidon-Fletcher-Powell method, are available as Fortran subroutines in Ref. 8.

2.2 Infinite-Dimensional Conjugate Gradient: Unconstrained

In this chapter the conjugate gradient method is treated separately in finite- and infinite-dimensional spaces because of applications. However, one could describe the method in a Hilbert space setting and, thus, cover both the finite- and infinite-dimensional cases in one development. Such is the approach taken in Refs. 15, 16, and 17.

The main references for Sections 2.2 and 2.3 are Refs. 6 and 18. In this section we shall consider problems which do not possess control or state variable inequality constraints; these will be included in the next section.

The infinite-dimensional problem which we are mainly concerned with is the following:

**BASIC PROBLEM:** Determine the control \( u^*(t), t \in [t_0, t_f] \), which minimizes:

\[
J[u] = \tilde{\phi}(t_f, x_f) + \int_{t_0}^{t_f} L(t, x, u) \, dt
\]

subject to:

\[
\dot{x} = f(t, x, u) \quad , \quad x(t_0) = x_0 \quad (2.26)
\]

\[
\psi(t_f, x_f) = 0 \quad , \quad (p \text{-vector}; \ p \leq n + 1) \quad (2.27)
\]

where \( x \) is an \( n \)-vector, \( u \) is an \( m \)-vector.

The algorithms in this report treat all of the terminal conditions (i.e., Eq. (2.27)) except one by the method of penalty functions; the remaining condition is employed as a stopping condition. Without loss of generality,
assume that

\[
\psi(t_f, x_f) = \begin{bmatrix}
\psi_1(t_f, x_{1f}) - x_{1f} \\
\psi_2(t_f, x_{2f}, \ldots, x_{nf}) \\
\vdots \\
\psi_p(t_f, x_{2f}, \ldots, x_{nf})
\end{bmatrix} = 0,
\]

(2.28)

and that \(x_1(t)\) is a variable which: (i) cannot reach the value \(x_{1f}\) until the terminal portion of the trajectory (e.g., a specified altitude or Mach number in reentry), (ii) will always be reached in a reasonable time, and (iii) will probably have a nonzero derivative at \(t_f\). In this case, \(x_1(t_f) = x_{1f}\) is a suitable stopping condition for the iterations.

Define

\[
\phi(t_f, x_f) \equiv \tilde{\phi}(t_f, x_f) + \sum_{i=2}^{P} P_{i-1} \psi_i(t_f, x_f)^2,
\]

(2.29)

where it is assumed, also, that \(\tilde{\phi}(t_f, x_f)\) does not depend upon \(x_{1f}\) (this is the usual case in trajectory analysis; the assumption is not restrictive, however) and the

\[
P_i > 0 \quad (i = 1, \ldots, p - 1)
\]

(2.30)

are selected by the investigator. With the definitions (2.28) and (2.29) we have the following problem:

**BASIC PROBLEM WITH PENALTY FUNCTIONS:** Determine the control \(u^*(t), t \in [t_0, t_f]\), which minimizes

\[
J[u] = \phi(t_f, x_f) + \int_{t_0}^{t_f} L(t, x, u) dt
\]

(2.31)

subject to:

\[
\dot{x} = f(t, x, u), \quad x(t_0) = x_0 \quad \text{(2.32)}
\]

\[
x_1(t_f) = x_{1f}.
\]

(Note: \(t_f\) is usually not specified.)
Before we list the formulas in the conjugate gradient method, we shall define a Hamiltonian function and adjoint variables which are useful in any function space iteration scheme. First, define

$$H = L(t, x, u) + \lambda^T f(t, x, u), \quad (2.34)$$

where the n-vector $\lambda(t)$ will be characterized later. With this definition we have:

$$J[u] = \phi(t_f, x_f) + \int_{t_0}^{t_f} \left[ H(t, x, u, \lambda) - \lambda^T x \right] dt, \quad (2.35)$$

where the performance index (2.31) has been augmented to include $\int_{t_0}^{t_f} \lambda^T (f - \dot{x}) dt$.

Let $u^{(0)}(t)$ be an initial control estimate, and integrate $x = f[t, x, u^{(0)}(t)]$ forward from $x(t_0) = x_0$ to form a corresponding trajectory, $x^{(0)}(t)$. Suppose there exists a vector $\lambda^{(0)}(t)$ and define

$$u^{(1)}(t) = u^{(0)}(t) + \delta u(t) \quad (2.36)$$
$$x^{(1)}(t) = x^{(0)}(t) + \delta x(t). \quad (2.37)$$
$$t^{(1)}_f = t^{(0)}_f + dt_f \quad (2.38)$$

Expand $J[u^{(1)}]$ about $J[u^{(0)}]$ to first-order:

$$J[u^{(1)}] = J[u^{(0)}] + \phi^{(0)} t_f + \sum_{i=2}^{n} \frac{\partial \phi^{(0)}}{\partial x_{i}} dx_{i}$$
$$+ \left[ H(t^{(0)}_f) - \lambda^{(0)} T (t^{(0)}_f) x^{(0)}(t^{(0)}_f) \right] dt_f$$
$$+ \int_{t_0}^{t_f} \left[ H_x^{(0)} \delta x + H_u^{(0)} \delta u - \lambda^{(0)} T \delta x \right] dt. \quad (2.39)$$

Integration by parts of the third term in the integrand gives

$$\Delta J[\delta u] = J[u^{(1)}] - J[u^{(0)}] = \left( \phi^{(0)} + H^{(0)} t_f \right) + \sum_{i=2}^{n} \frac{\partial \phi^{(0)}}{\partial x_{i}} dx_{i}$$
$$+ \left[ H(t^{(0)}_f) - \lambda^{(0)} T (t^{(0)}_f) x^{(0)}(t^{(0)}_f) \right] dt_f$$
$$+ \int_{t_0}^{t_f} \left[ H_x^{(0)} \delta x + H_u^{(0)} \delta u - \lambda^{(0)} T \delta x \right] dt. \quad (2.40)$$
subject to:

\[ dx_1f = 0. \]  \hfill (2.41)

We now characterize \( \lambda^{(0)}(t) \) so that a stable iterative algorithm is defined.

**SPECIFY:**

\[
\begin{align*}
\lambda_i^{(0)}(t_f^{(0)}) &= \phi_i^{(0)} x_i^{(0)} \quad (i = 2, \ldots, n) \\
\lambda_i^{(0)}(t_f^{(0)}) &= - (\phi_i^{(0)} + H^{(0)} + \sum_{i=2}^{n} \lambda_i^{(0)} x_i^{(0)}) t_f^{(0)} / x_i^{(0)}(t_f^{(0)}) \\
\lambda^{(0)}(t) &= - H_x[t, x^{(0)}(t), u^{(0)}(t), \lambda^{(0)}(t)].
\end{align*}
\]  \hfill (2.42) \hfill (2.43) \hfill (2.44)

Definitions (2.42), (2.43), (2.44) uniquely define the vector \( \lambda^{(0)}(t) \) and it is formed by a backward integration.

Substitution of Eqs. (2.41) - (2.44) into Eq. (2.40) gives

\[ \Delta J[\delta u] = \int_{t_0}^{t_f} H_u^{(0)} T \delta u \, dt. \]  \hfill (2.45)

The quantity \( H_u^{(0)}(t) \) is the gradient in function space for this iteration, and the gradient method is defined by

\[ u^{(J+1)}(t) = u^{(J)}(t) - \alpha J [H_u^{(J)}(t)]. \]  \hfill (2.46)

(Note that if \( t_f^{(J+1)} > t_f^{(J)} \), then a scheme must be devised to define \( u^{(J+1)}(t) \) on the interval \([t_f^{(J)}, t_f^{(J+1)}] \), but there are numerous ways of doing this.)

As with the parameter optimization problem, there exist numerous techniques which result in a stable method, e.g., one need only guarantee that the first-order expansion term dominate the expansion for \( J[u^{(J+1)}] \) and that \( \delta u \) be chosen in such a way that

\[ \int_{t_0}^{t_f} H_u^{(0)}(t) \delta u(t) \leq 0. \]  \hfill (2.47)

In analogy with parameter optimization, a possible choice for \( \delta u \) is

\[ \delta u^{(J)}(t) = - \alpha J [H_u^{(J)}(t) + C_J P^{(J-1)}(t)]. \]  \hfill (2.48)
where $\alpha_J > 0$ is the search parameter, $p^{(J-1)}(t)$ is the previous search direction with the property $\int_{t_0}^{t_f} H_u^{(J)} p^{(J-1)} dt = 0$, and $C_J$ is a constant to be defined. As shown in Ref. 6, the following function space conjugate gradient scheme satisfies these conditions:

**UNCONSTRAINED CONJUGATE GRADIENT ALGORITHM**

1) Guess $u^{(0)}(t)$ on $[t_0, t_f]$. 
2) Compute:

$$\begin{align*}
X^{(J)}(t), \quad (J) \ (t), \quad H_u^{(J)}(t) \\
p^{(J)}(t) &= H_u^{(J)}(t) + \frac{t_f}{t_0} \int_{t_0}^{t_f} H_u^{(J)} H_u^{(J)} dt \\
&= \frac{t_f}{t_0} \int_{t_0}^{t_f} H_u^{(J-1)} H_u^{(J-1)} dt \\
(p^{(0)}(t) &= H_u^{(0)}(t))
\end{align*}$$

(2.49)

3) Perform 1-D search to determine $\alpha_J$ in the formula

$$u^{(J+1)}(t) = u^{(J)}(t) - \alpha_J p^{(J)}(t).$$

(2.50)

4) Check on appropriate cutoff criterion (e.g., $|dJ/\alpha_J|_{\alpha_J=0} \leq \epsilon$);
5) Return to 2).

In Eq. (2.49) above, the constant which multiplies $p^{(J-1)}$ may be written as

$$<H_u^{(J)}, H_u^{(J)}>_1 / <H_u^{(J-1)}, H_u^{(J-1)}>_1,$$

where

$$<a(t), b(t)>_1 \equiv \int_{t_0}^{t_f} a(t) b(t) dt$$

(2.51)

is an inner product on the function space whereas

$$<a, b> \equiv a^T b$$

(2.52)

is an inner product on $R^n$. Thus, the formula is the same as the finite-dimensional formula; one need only interpret properly the gradient and
inner product functions.

In Ref. 6, a few theorems concerned with the convergence of the function space conjugate gradient method are presented for both general functionals and functionals which result from linear-quadratic optimal control problems. For general functionals, the convergence theorem (which is only sufficient for convergence) essentially requires that one show that the second variation is "strongly positive" (i.e., there exists a constant $M > 0$ such that for all admissible $u, \delta u$, $\delta^2 J(u; \delta u, \delta u) \geq M\|\delta u\|^2$).

As with parameter optimization, the quadratic case plays an important role in functional optimization; again, the argument being that when the solution is approached the general optimal control problem may be well-approximated by a linear-quadratic optimal control problem (formed by expanding the differential equations and boundary conditions to first-order, and the performance index to second-order). The theorems in Ref. 6 assume the resultant quadratic functional to be of the form

$$\Delta J = \langle \delta u, A\delta u \rangle,$$  \hspace{1cm} (2.53)

where $A$ is a positive definite, self-adjoint linear operator. Note that since $\delta u$ is infinite-dimensional there is no reason to expect finite convergence even in a small neighborhood of the solution. (Of course, on a digital computer, one is really only interested in a good rate of convergence since problems are never converged to the limit.) Reference 7 shows how one may transform a class of linear-quadratic problems into the form of Eq. (2.53).

For the case of Eq. (2.53), Ref. 6 shows that the conjugate gradient method has certain desirable features which the classical gradient method does not possess. However, it has never been proved mathematically that the conjugate gradient step is better than the gradient step on every iterate. In fact the statement is probably untrue because of numerical experience which indicates that a gradient step every so often in a conjugate gradient algorithm (i.e., a "reset" step) improves the convergence characteristics. Finally, as with general functionals, to show that the linear operator $A$ in...
Eq. (2.53) is positive definite usually requires a conjugate point test if the operator results from linearization of a nonlinear optimal control problem.

2.3 Infinite-Dimensional Conjugate Gradient: Constrained

In this section, the modifications of the Basic Problem With Penalty Functions (Eqs 2.31-2.33) and the Unconstrained Conjugate Gradient Algorithm to include state variable inequality constraints (SVIC) and control inequality constraints will be presented.

First, suppose that in addition to the equality constraints (2.32), (2.33), the problem contains the SVIC's:

\[ S_i(t,x) \geq 0. \quad (i = 1, \ldots, q) \] (2.54)

There are two main ways of treating an SVIC:

(i) Transform the problem into a multiple-arc problem with intermediate point equality constraints; this is the approach of Ref. 19.

(ii) Augment the performance index to include the SVIC's by means of penalty functions; this is the approach of Ref. 1.

The main goal of the computer programs described in this report is to generate reasonable, near-optimal reentry trajectories with a minimal amount of guessing and analysis required of the user. The (i) approach above requires knowledge of the location and the number of times the inequality constraint boundary is encountered, which requires both analysis and additional programming by the user. Thus, the (ii) approach was chosen since this requires no additional programming and only the initial penalty coefficients must be estimated.

If the SVIC's (2.54) are present, then the performance index (2.31) is modified to

\[ J[u] = \phi(t_f,x_f) + \int_{t_0}^{t_f} \left[ L(t,x,u) + \sum_{i=1}^{q} S_i(t,x) H_i(S_i) \right] dt, \] (2.55)

where

\[ H_i(S_i) = \begin{cases} C_i & \text{if } S_i < 0 \\ 0 & \text{if } S_i \geq 0 \end{cases}, \] (2.56)
and the constant penalty coefficients $C_i(i = 1, \ldots, q)$ are selected by the investigator.

Second, suppose that inequality constraints

$$G_i(t, x, u) \geq 0 \quad (i = 1, \ldots, r) \quad (2.57)$$

which satisfy the following constraint condition are present:

$$
\begin{bmatrix}
\frac{\partial G_1}{\partial u_1} & \cdots & \frac{\partial G_1}{\partial u_m} & G_1 & 0 & \cdots & 0 \\
\frac{\partial G_2}{\partial u_1} & \cdots & \frac{\partial G_2}{\partial u_m} & G_2 & 0 & & \\
\vdots & & \vdots & \vdots & \ddots & \vdots & \vdots \\
\frac{\partial G_r}{\partial u_1} & \cdots & \frac{\partial G_r}{\partial u_m} & G_r & 0 & \cdots & 0
\end{bmatrix}
\text{ has rank } r. \quad (2.58)
$$

This condition is required to guarantee that the control may be determined from the appropriate $G_i = 0$ when a boundary is encountered. The condition is satisfied trivially if the constraints only contain control variables, e.g., $G = 1 - u^2 \geq 0$, and are independent.

Since the adjoint variables (or Lagrange multipliers) are continuous across corners where control boundaries are encountered, constraints of the form (2.57) may be treated directly with little modification of the program. Let us first describe the procedure for treating a constraint of the form (2.57) before we justify the method.

CONTROL CONSTRAINED CONJUGATE GRADIENT ALGORITHM

Suppose the control is a scalar and $|u| \leq 1$; the generalization to more than one control and other control constraints is straightforward:

1) At the beginning of the $J^{th}$ iteration, we have a control $u^{(J)}(t)$, \hspace{1cm} $J \in \{0, 1, 2, \ldots\}$. Define $W_J = \{t: |u^{(J)}(t)| = 1\}$, i.e., the set of points where $u^{(J)}(t)$ is on the boundary. Integrate forward $\dot{x} = f[t, x, u^{(J)}(t)]$ to the stopping condition and set $\lambda^{(J)}(t_f^{(J)})$.

2) Integrate $\dot{\lambda}^{(J)}(t) = -H_x[t, x^{(J)}(t), \lambda^{(J)}(t), u^{(J)}(t)]$ backwards from $t_f^{(J)}$. Evaluate $H_u^{(J)}(t)$ in the usual way on $[t_0, t_f^{(J)}]$. However, the inner product $\langle H_u^{(J)}, H_u^{(J)} \rangle$ is defined by:
Perform the 1-D search with Eqs. (2.49), (2.50).

In the search, truncate \( u^{(J+1)} \) at the boundary if \( |u^{(J+1)}(t)| > 1 \), i.e.,

for a trial \( \alpha_J \):

\[
\text{if } u(t) - \alpha_J p(t) > 1, \text{ set } u^{(J+1)}(t) = 1 \tag{2.60}
\]

\[
\text{if } u(t) - \alpha_J p(t) < 1, \text{ set } u^{(J+1)}(t) = -1.
\]

(This step gives us the means for adjusting the set \( W_J \) from iteration to iteration.) Return to (1) after \( \alpha_J \) and \( W_{J+1} \) are determined.

Let us now justify the approach listed above; the method is developed in Ref. 18. The main difficulties in generating a method for treating control constraints are: (i) ensuring that the method is defined in such a way that it can converge to the true minimum (and not a false minimum), and (ii) developing a method consistent with (i) for defining \( <H_u^{(J)}, H_u^{(J)}> / <H_u^{(J-1)}, H_u^{(J-1)}> \) when the iterate has bounded subarcs.

First, we shall consider how the algorithm should behave near the minimum. Suppose that the set \( W \) is known beforehand, i.e., the points \( t \in [t_0, t_f] \) for which \( u^*(t) = \pm 1 \) are known. Then, the algorithm need only be concerned with "fine-tuning" the interior control segments. In this regard, we would want \( H_u^{(J)} \) and \( p^{(J)} \) to be such that it only changes the interior control segments and not the boundary segments. Thus, in the computation of the coefficient of \( p^{(J-1)}(t) \) in Eq. (2.49), the effect of the boundary arcs is not included because of the form of Eq. (2.59), and this rule is consistent with requirement (i) above.

In reality, we do not know the set \( W \) beforehand, so we must devise a mechanism for the sets \( W_J \) to change from iterate to iterate and such that \( W_J \rightarrow W \). This is accomplished by Step (3) of the procedure defined above; that is, the set \( W_J \) is modified in the 1-D search.
CHAPTER 3
PHASE I PROGRAM

3.1 Basic Description

The Phase I Program is designed to minimize a weighted performance index which includes the following effects:

i. Crossrange
ii. Downrange
iii. Aerodynamic loading
iv. Terminal total heat
v. Terminal altitude

The equations of motion are written in a cartesian coordinate system defined by:

\[ \hat{I} = \frac{\bar{r}}{|\bar{r}|} \]
\[ \hat{K} = \frac{\bar{r} \times \bar{V}}{|\bar{r} \times \bar{V}|} \]
\[ \hat{J} = \hat{K} \times \hat{I} \]

The Aerodynamic Angles are defined by the following coordinate system:
The state equations are:

\[ \dot{\mathbf{r}} = \mathbf{V} \]

\[ \dot{\mathbf{V}} = \frac{F}{|F|} \mathbf{r} + \rho A \left[ \frac{C_A}{C_L} \frac{\mathbf{V}_R}{|\mathbf{V}_R|} \left( \frac{\mathbf{V}_R}{|\mathbf{V}_R|} \right)^2 \right] \frac{\mathbf{V}_R}{|\mathbf{V}_R|} + \frac{1}{|\mathbf{V}_R|} \left[ \mathbf{V}_R \mathbf{V}^T_R \right] \frac{\mathbf{u}}{|\mathbf{u}|} \]

\[ \dot{Q} = C_q \rho^2 \mathbf{V}_R^3 \]

\[ \dot{\mathbf{V}}_R = \mathbf{V} - \mathbf{V}_A \]

where \( \mathbf{V}_A \) = inertial velocity of the atmosphere. The equations involve the following assumptions:

a. The relative velocity vector \( \mathbf{V}_R \) is in the plane of the vehicle that produces the greatest lift.

b. No aerodynamic moments exist about the center of mass.

The performance index is

\[ J = C_1 r_c + C_2 r_d + P_1 (h - h_{f})^2 + P_2 (Q)^2 \]

\[ + P_3 \int_{t_0}^{t_f} \left( \frac{L^2 + D^2}{m^2} - 9g^2 \right) \cdot U \left( \frac{L^2 + D^2}{m^2} - 9g^2 \right) dt \]

where

\[ r_c = \text{cross range} \]

\[ r_d = \text{down range} \]

\[ U(\cdot) = \begin{cases} 1 & \text{if } (\cdot) > 0 \\ 0 & \text{if } (\cdot) \leq 0 \end{cases} \]
The Hamiltonian is

\[
H = \lambda T_T + \lambda V_V + \lambda Q Q + P_4 \left( \frac{L^2 + D^2}{m^2} - 9g^2 \right) \cdot U \left( \frac{L^2 + D^2}{m^2} - 9g^2 \right)
\]

3.2 Subroutine Map

MAIN — ADOTB — ACROS

WPRJCG

ACROS — ADOTB ACROS — ADOTB

FWDINT — RK713 — COSTFN — XLAMFN — OUTPUT

ADOTB

DERIV1 — ACROS — ADOTB — ATMOS — AEROD

ADOTB — ACROS

BAKINT — RK713 — GRADFN

DERIV2 — DERIV1 — ADOTB — ACROS

ACROS — ADOTB — ATMOS — AEROD

SEKALF
3.3 Subroutine Descriptions

MAIN: Reads in all necessary data, sets integration coefficients, computes initial values, and calls on the conjugate gradient subroutine (WPRJCG). On Return, MAIN prints out a message concerning the results of the iteration and prints out the control obtained by that iteration.

A. Namelist Input Data

PI = \pi
RE = earth's radius
XMU = \mu, gravitational constant
OMEGE(3) = angular velocity of the earth
AREA = aerodynamic reference area
ECOEF = heating coefficient
XO(3) = initial position vector
VO(3) = initial velocity vector
TO = initial time
ALTF = desired final altitude
XMACH = desired final Mach number
FLTANG = desired final flight path angle
QMAX = desired final heating value
XMASS = vehicle mass
IOUT = print frequency for forward integration
IOUTZ = print frequency for backward integration
IPRINT1 = print control flag
IPRINT2 = print control flag
DELTS = integration stepsize
IKEY = call flag for output (see FWDINT)
ERRMX = error tolerance for integration routine
ERRMN = not used
TCUT = upper time limit on trajectory
EPST = cutoff tolerance for norm of control change
EPSTF = not used
EPSA = cutoff tolerance for integration altitude cutoff
EPSIT = cutoff tolerance on gradient norm
ERR = cutoff tolerance for small cost change
ITMAX = limit on number of conjugate gradient iterations
ITMX = limit on steps in 1-D search
KOUTM = limit on iterations for altitude cutoff
CSTR = guess of final cost value
B = control bound (see SEKALF)
PFUN(4) = penalty coefficient vector
CCOST(2) = coefficients in cost functional
DTFM = maximum allowable final time change
XDTFM = fraction of DTFM used to start 1-D search

B. Control Vector Data
IJKU = total number of control points
U(IJKU,4) = control vector and time point

WPRJC: This subroutine controls the application of the conjugate gradient algorithm. It calls the forward and backward integration routines, directs the one dimensional search, and updates the control vector and terminal time. It checks for algorithm termination on small cost change, total number of iterations, errors in the 1-D search, failure to generate an admissible trajectory on the first trial.

SEKALF (One-Dimensional Search Subroutine): Determines the parameters for the new control value in the conjugate gradient algorithm. Fits a cubic in $\alpha$ to known values of $J(\alpha)$, $\partial J/\partial \alpha$, to obtain $\min J(\alpha)$ and then $\alpha^*$ for $J_{\min}$.

FWDINT: Subroutine performs the forward integration of the state variables and calls the subroutines to evaluate the cost functional and final multiplier values.

RK713: 7th Order Runge-Kutta integration scheme called by both FWDINT and BAKINT.

BAKIWT: Subroutine performs the backward integration of the state variables and multiplier equations, and calls on GRADFW to calculate the
gradients and store the value at each integration step. The subroutine also determines the new search direction.

**DERIV1**: Subroutine which calculates the time derivatives of the state variables.

**DERIV2**: Subroutine which calculates the time derivatives of the multipliers.

**ATMOS**: Calculates atmospheric parameters.

**AEROD**: Calculates aerodynamic parameters.

**XLAMFN**: Computes final multiplier values.

**COSTFN**: Computes cost functional.

**OUTPUT**: Subroutine called by FWDINT which prints out desired trajectory data.

### 3.4 Phase II Program Notes

i) The program obtains the state for the multiplier equations by integrating the state backward from the terminal conditions of the forward state integration (as opposed to storing the state in the forward integration).

ii) Each iterate is terminated on an assumed $t_f$, which is part of the iteration procedure. The value of $t_f$ for the base trajectory is determined by the trajectory as the time when the desired altitude is reached (thus, the program also has an altitude-cutoff capability).

iii) See Appendix A for a listing of the Phase I Program.
CHAPTER 4

PHASE II PROGRAM

4.1 Basic Description

The Phase II Program is designed to minimize a performance index which includes the following effects:

1. Crossrange
2. Downrange
3. Total heat
4. Peak heating rate
5. Final speed and flight path angle boundary conditions.

Phase II Program uses a nonrotating earth centered spherical coordinate system with an Euler angle body-axis system to define the aerodynamic forces.
The equations of motion assuming a nonrotating earth and no aerodynamic moments are

\[
\begin{align*}
\dot{R} &= V \sin \gamma \\
\dot{\theta} &= \frac{V \cos \gamma \cos \psi}{R \cos \phi} \\
\dot{\phi} &= \frac{V \cos \gamma \sin \psi}{R} \\
\dot{v} &= -\frac{\mu \sin \gamma}{R^2} - \frac{D}{m} \\
\dot{\gamma} &= -\frac{\mu \cos \gamma}{R^2 V} + \frac{V \cos \gamma}{R} + \frac{L}{mV \cos \beta} \\
\dot{\psi} &= \left[ -\frac{V \cos \gamma \cos \psi \sin \phi}{R \cos \phi} - \frac{L \sin \beta}{mV \cos \gamma} \right]
\end{align*}
\]

where the drag \(D\) and lift \(L\) are defined by

\[
\begin{align*}
L &= \frac{1}{2} \rho S V^2 C_L(\alpha, M) \\
D &= \frac{1}{2} \rho S V^2 C_D(\alpha, M)
\end{align*}
\]

The cost functional to be minimized is:

\[
J = C(1) R e^2_f + C(2) R e^2_f + C(3) (V(t_f) - \bar{V}_f)^2 + C(4) [\gamma(t_f) - \bar{\gamma}_f]^2
\]

\[
\int_{t_0}^{t_f} \dot{q} \, dt + C(5) \int_{t_0}^{t_f} q^2 \, dt + C(6) \int_{t_0}^{t_f} \dot{q}^2 \, dt
\]

The term \(\int_{t_0}^{t_f} \dot{q}^2 \, dt\) is an approximate method for minimizing the peak heat rate.

The Hamiltonian is

\[
H = C(5) \dot{q}(R, V) + C(6) \dot{q}^2(R, V, \gamma) + \lambda_1(V \sin \gamma) + \lambda_2\left(\frac{V \cos \gamma \cos \psi}{R \cos \phi}\right) + \lambda_3\left(\frac{V \cos \gamma \sin \psi}{R}\right)
\]

\[
+ \lambda_4\left(-\frac{\mu \sin \gamma}{R^2} - \frac{D}{m}\right) + \lambda_5\left(-\frac{\mu \cos \gamma}{R^2 V} + \frac{V \cos \gamma}{R} + \frac{L}{mV \cos \beta}\right)
\]

\[
+ \lambda_6\left(-\frac{V \cos \gamma \cos \psi \sin \phi}{R \cos \phi} - \frac{L \sin \beta}{mV \cos \gamma}\right)
\]
Forward integration of the state variables is cutoff on a desired altitude.

4.2 Subroutine Map

MAIN
  | WPRJCG
  | FWDINT — DERIV1 — ATMOS — SETUP — SPLINE
  |           | XLAMFN — COSTFN — OUTPUT
BAKINT — DERIV2 — ATMOS — SETUP — SPLINE
  |           | GRADFN
SEKALF
4.3 Subroutine Descriptions

**MAIN**: Reads in all necessary input parameters, sets up spline interpolation of aerodynamic coefficients and calls the conjugate gradient subroutine WPRJCG. On Return, MAIN prints out message concerning the results of the iteration and prints out the control obtained by that iteration.

**A. Namelist Data**

- **PI** = $\pi$
- **RE** = radius of the earth
- **XMU** = $\mu$, gravitational constant
- **OMEGE** = not used
- **AREA** = aerodynamic reference area
- **ECOEF** = heating coefficient
- **DELT** = integration stepsize
- **IKEY** = call flag for OUTPUT
- **ERRMX** = not used
- **ERRMN** = not used
- **TCUT** = upper time limit on trajectory
- **EPST** = cutoff tolerance for norm of control change
- **EPSTF** = not used
- **EPSA** = cutoff tolerance for integration altitude cutoff
- **EPSIT** = cutoff tolerance on gradient norm
- **ERR** = cutoff tolerance for small cost change
- **ITMAX** = limit on number of conjugate gradient iterations
- **ITMX** = limit on steps in 1-D search
- **KOUNTM** = limit on iterations for altitude cutoff
- **CSTR** = guess of final cost value
- **B** = control bound (see SEKALF)
- **C(7)** = coefficients in cost functional
- **DTFM** = not used
XDTFM = not used
SVARO(6) = initial state variables
TO = initial time
ALTF = cutoff altitude
XMACH = not used
FLTANG = not used
GAMMF = final flight path angle
VF = final velocity
XMASS = vehicle mass
IOUT = print frequency for forward integration
IOUT2 = print frequency for backward integration
IPRINT1 = print control flag
IPRINT2 = print control flag

B. Control Data
IJKU = total number of control points
U(IJKU, 3) = control vector and time points

C. Aerodynamic Data
N1, N2 = dimensions of coefficient array
Y(N1,N2,2) = coefficient array
(See sample program for input format)

See Chapter 3.3 for descriptions of:
   WPRJCG
   SEKALF
   DERIV1
   DERIV2
   ATMOS
   XLAMFN
   GRADFN
   COSTFN
   OUTPUT
   FWPIWT
   BAKINT
SETUP - SPLINE - Subroutine computes aerodynamic coefficients based upon piecewise cubic spline interpolation. Input is angle of attack ($\alpha$) and Mach number ($M$); returned are the values of $CL, CD, \partial CL/\partial M, \partial CD/\partial M, \partial CD/\partial \alpha$.

(For test runs the aerodynamics were approximated by:

\[ CD = 2.2 \sin^3 \alpha + .08 \]
\[ CL = 2.2 \sin^2 \alpha \cos \alpha + .01. \]

4.4 Phase II Program Notes

i) The state values for the backward integration of the multiplier equations are stored during the forward integration (as opposed to backward integration for the state). The program currently can store the state at 999 time points.

ii) All trajectories terminate at a specified, desired altitude. The modification to the transversality conditions is discussed in Chapter 2. Since the terminal time of the $N+1$ trajectory, say $t_f^{(N+1)}$, may be larger than $t_f^{(N)}$ (since $h_f$ is the cutoff condition), a linear extrapolation of the control is used on $[t_f^{(N)}, t_f^{(N+1)}]$.

iii) See Appendix B for a listing of the Phase II Program.
CHAPTER 5
CONCLUSIONS AND RECOMMENDATIONS

5.1 Summary

Two computer programs for shuttle reentry optimization have been developed. The programs make extensive use of subroutines so that they may be adapted to other atmospheric optimization problems with little difficulty. Because of contract budget restrictions and the long flight times of realistic shuttle reentry trajectories, the programs have only been checked out with respect to programming errors. In the next section, suggestions for a study of the convergence properties of the programs will be presented. Also, our limited experience obtained with the programs will be discussed. However, with respect to these comments, it should be remembered that no controlled study was performed, and thus, the comments are somewhat tenuous.

5.2 Conclusions and Recommendations

1. Before an extensive analysis of the optimization of reentry trajectories is undertaken, it is recommended that a carefully controlled study of numerical integration procedures be performed for reentry problems in which: (a) the controls are piecewise linear (or possibly higher-order splines) in an integration step, (b) the aerodynamic data is given in tabular form, and (c) the vehicle is a relatively low-drag vehicle (e.g., the high-crossrange shuttle). Much of our time was devoted to determining an acceptable numerical integration package while the optimization procedure was the major goal of the study. We found that RK 7-13 was an excellent scheme with constant aerodynamics and smooth controls; however, with piecewise linear controls and spline-fit aerodynamics, its performance was reduced substantially. For this reason, a fourth-order, predictor-corrector scheme with fixed stepsize is employed in the Phase II-Program. Research should be conducted to make the problem suitable for use with RK 7-13 (or some other high-order scheme) to shorten the long integration times.

2. Because of the relatively low-drag characteristics of the
high-crossrange shuttle, arbitrary initial estimates of the controls in any optimization program may cause highly oscillatory trajectories. This is due to the fact that the path angle may become positive (positive path angle is above the local horizontal) and oscillate about zero degrees. Thus, it is recommended that, if possible, initial control estimates be chosen so that $\gamma$ remains negative. Some investigators have used artificial means to insure $\gamma \leq 0$, e.g., impose a state variable inequality constraint, add damping to the initial iterates, increase the drag in the initial iterates. This problem may be accentuated by an inaccurate numerical integration scheme because $\dot{\gamma}$ is essentially the difference between two terms of the same order of magnitude. Thus, $\dot{\gamma}$ may become positive because of numerical error when its true physical value is negative.

3. Neither program uses nondimensional variables. If the rate of convergence is slow in simulations, nondimensionalization of the variables may improve the rate.

4. Most of the investigations which have applied the conjugate gradient method to optimal control problems have been of low-dimension, near-linear, and fixed final time. Two exceptions are Refs. 21 and 22. In these studies, it was found that the method did not perform satisfactorily on a problem with tight terminal conditions\textsuperscript{21} and a free-final time problem\textsuperscript{22}. Since the two programs of this report treat the free final time problem in two different ways, trends as to which method is best would be useful information.

5. In the Phase II-Program, $\int_{t_0}^{t_f} q^2 dt$ is used in the performance index to penalize large heat rate slopes, and, thus, should aid in "flattening-out" the heating rate. This conjecture should be tested since if it serves to flatten the peak heating rate, it might be a simple way of controlling peak heating rate in an on-board, optimization oriented guidance scheme.

6. A convenient test problem for reentry is the maximum crossrange
problem. In this problem, the optimal control should consist of an angle of attack which causes \((L/D)_{\text{max}}\) and a bank angle which is initially near 90° and which decreases (nearly linearly) toward 0° as time increases to \(t_f\). In our limited testing of the Phase II-Program on the IBM 360/67, a typical iterate (including the 1-D search) required about one minute of CPU time for a double precision, 2000 second (real-time) trajectory with a fixed stepsize of four seconds.

7. As noted above, a typical iterate requires approximately one minute of CPU time. Of course, the large amount of computer time is mainly due to the numerical integration requirements. Hopefully, more efficient numerical integration schemes will be developed for use in conjunction with function-space gradient-type algorithms. In this development one should keep in mind that both forward and backward integrations are required, and this heavily influences the choice of a variable stepsize integration scheme. A possibility in this direction is spline numerical integration schemes since they result in "global" information as opposed to discrete data.
REFERENCES


34


APPENDIX A

LISTING OF PHASE I PROGRAM
37

C. READ IN DATA

1 READ(11,AMMFG)
   READ(7,700,1,1)
   READ(7,701,4)(I111,1),I=1,4,1=1,11KU)
   WRITE(6,AMMFG)

C. INITIAL CONDITIONS

C20A AG AMI (X111.X111)
C20B AG AMI (X111,Y111)
C20C AG AMI (X111,Z111)
C20D AG AMI (XI111,Y111,Z111)

C. INITIAL CONDITIONS

C20E AG AMI (X111,Y111,Z111)
C20F AG AMI (X111,Y111,Z111)
C20G AG AMI (X111,Y111,Z111)
C20H AG AMI (X111,Y111,Z111)
C20I AG AMI (X111,Y111,Z111)
C20J AG AMI (X111,Y111,Z111)
C20K AG AMI (X111,Y111,Z111)
C20L AG AMI (X111,Y111,Z111)
C20M AG AMI (X111,Y111,Z111)
C20N AG AMI (X111,Y111,Z111)
C20O AG AMI (X111,Y111,Z111)
C20P AG AMI (X111,Y111,Z111)
C20Q AG AMI (X111,Y111,Z111)
C20R AG AMI (X111,Y111,Z111)
C20S AG AMI (X111,Y111,Z111)
C20T AG AMI (X111,Y111,Z111)
C20U AG AMI (X111,Y111,Z111)
C20V AG AMI (X111,Y111,Z111)
C20W AG AMI (X111,Y111,Z111)
C20X AG AMI (X111,Y111,Z111)
C20Y AG AMI (X111,Y111,Z111)
C20Z AG AMI (X111,Y111,Z111)
C20A AG AMI (X111,Y111,Z111)
C20B AG AMI (X111,Y111,Z111)
C20C AG AMI (X111,Y111,Z111)
C20D AG AMI (X111,Y111,Z111)
C20E AG AMI (X111,Y111,Z111)
C20F AG AMI (X111,Y111,Z111)
C20G AG AMI (X111,Y111,Z111)
C20H AG AMI (X111,Y111,Z111)
C20I AG AMI (X111,Y111,Z111)
C20J AG AMI (X111,Y111,Z111)
C20K AG AMI (X111,Y111,Z111)
C20L AG AMI (X111,Y111,Z111)
C20M AG AMI (X111,Y111,Z111)
C20N AG AMI (X111,Y111,Z111)
C20O AG AMI (X111,Y111,Z111)
C20P AG AMI (X111,Y111,Z111)
C20Q AG AMI (X111,Y111,Z111)
C20R AG AMI (X111,Y111,Z111)
C20S AG AMI (X111,Y111,Z111)
C20T AG AMI (X111,Y111,Z111)
C20U AG AMI (X111,Y111,Z111)
C20V AG AMI (X111,Y111,Z111)
C20W AG AMI (X111,Y111,Z111)
C20X AG AMI (X111,Y111,Z111)
C20Y AG AMI (X111,Y111,Z111)
C20Z AG AMI (X111,Y111,Z111)
C20A AG AMI (X111,Y111,Z111)
C20B AG AMI (X111,Y111,Z111)
C20C AG AMI (X111,Y111,Z111)
C20D AG AMI (X111,Y111,Z111)
C20E AG AMI (X111,Y111,Z111)
C20F AG AMI (X111,Y111,Z111)
C20G AG AMI (X111,Y111,Z111)
C20H AG AMI (X111,Y111,Z111)
C20I AG AMI (X111,Y111,Z111)
C20J AG AMI (X111,Y111,Z111)
C20K AG AMI (X111,Y111,Z111)
C20L AG AMI (X111,Y111,Z111)
C20M AG AMI (X111,Y111,Z111)
C20N AG AMI (X111,Y111,Z111)
C20O AG AMI (X111,Y111,Z111)
C20P AG AMI (X111,Y111,Z111)
C20Q AG AMI (X111,Y111,Z111)
C20R AG AMI (X111,Y111,Z111)
C20S AG AMI (X111,Y111,Z111)
C20T AG AMI (X111,Y111,Z111)
C20U AG AMI (X111,Y111,Z111)
C20V AG AMI (X111,Y111,Z111)
C20W AG AMI (X111,Y111,Z111)
C20X AG AMI (X111,Y111,Z111)
C20Y AG AMI (X111,Y111,Z111)
C20Z AG AMI (X111,Y111,Z111)
C. CONSTANTS FOR INTEGRATION SUBROUTINE

C20A ALPH(0)=0.

C260 C14)=0.
C261 C13)=34./105.
C262 C12)=9./35.
C263 C11)=0./260.
C264 C10)=0./310.
C265 C9)=0./65.
C266 C8)=1./105.
C267 C7)=0./105.
C268 C6)=0./260.
C269 C5)=0./510.
C270 C4)=0./46.
C271 C3)=0./120.
C272 C2)=0./120.
C273 C1)=0./120.
C274 C0)=0./120.
C. CALL CONJUGATE GRADIENT ROUTINE
CALL WPPJGGL(IER)
GO TO (10,20,30,40,50,60,70,80,90,100),IER
10 CONTINUE
**39**

```fortran
20 WRITE(A,520)
   GO TO 101
30 WRITE(A,530)
   GO TO 101
40 WRITE(A,540)
   GO TO 101
50 WRITE(A,550)
   GO TO 101
60 WRITE(A,560)
   GO TO 101
70 WRITE(A,570)
   GO TO 101
80 WRITE(A,580)
   GO TO 101
90 WRITE(A,590)
   GO TO 101
100 WRITE(A,600)
101 CONTINUE
   WRITE(A,625) IJKU
   WRITE(A,650) ((U(I,J),I=1,3),J=1,1)JKU

STOP
500 FORMAT(214)
505 FORMAT(10,6)
520 FORMAT(10,5X,'ONE-D SEARCH FAILED TO FIND A MINIMUM')
530 FORMAT(10,5X,'CONVERGENCE NOT DECREASING IN SEARCH DIRECTION')
540 FORMAT(10,5X,'CONVERGENCE ON SMALL CONTROL CHANGE')
550 FORMAT(10,5X,'LITTLE COST CHANGE IN LAST TWO ITERATIONS')
560 FORMAT(10,5X,'FAILED TO CONVERGE IN ITH MAX ITERATIONS')
570 FORMAT(10,5X,'INITIAL TRAJECTORY FAILED TO REACH CUT-OFF ALT')
580 FORMAT(10,5X,'TOO MANY INTEGRATIONS STEPS REQUIRED')
590 FORMAT(10,5X,'BACKWARD INTEGRATED TRAJECTORY ERRORS')
600 FORMAT(10,5X,'CONVERGENCE ON ZERO GRADIENT NORM')
625 FORMAT(15)
650 FORMAT(13026,16)
700 FORMAT(15)
750 FORMAT(3026,16)
END
```
C SUBROUTINE WRPJC
    SUBROUTINE WRPJC(IF)
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION XJ(8,A),XJLAM(7),SERCH(999,4),IT(999,4),TEMPI(999,3)
    1,PFCN(41),CO(2),GRAD(999,4)
    COMMON/CONS/FLTS.*,FRMX,FRMN,TCUT,EPST,EPSTF,EPSS,EPST,
    1 ERR,TLMX,ITMX,PLUSM,KEY*
    COMMON/CONS3/STK(4),PFUN,CO,DTF,M,DTFM
    COMMON/CNTL/GRAD,SRCH,STR,ASTR,STF,TF,KJIS,IJKU,ISTAR
    ITR=0
    C PERFORM FORWARD INTEGRATION TO ALTITUDE CUT-OFF
    8 ISWAR=0
    1 IFLAG=1
    CALL FWINTC(CST,XJ,TF,XJLAM,FCDTF,IFLAG)
    IF(FLAG.EQ.1) GO TO 94
    C PERFORM BACKWARD INTEGRATION
    9 CALL MAKINTXJ,XJLAM,TF,ITER,DCOST,XNORMS,DCDTF
    IF(ITER-ITMX) 7,99,93
    7 CSAVE=DCOST
    ITHUM = ITER + 1
    WRITE(6,603) ITHUM
    A03 FORMAT(1,H0,5X,'ITERATION NUMBER=',I5,'//')
    C ENTER 1-D SEARCH
    1 ENTER=1
    IFLAG=2
    KHC=0
    JKM=0
    TPS=TF
    IFLAG=0
    IF(DTFM.LT.0.D0) GO TO 10
    C HAVE A RESTRICTION ON FINAL TIME CHANGES, COMPUTE FIRST GUESS
    ASTR=MANS(XDTF*.0,TMF/STF)
    IFLAG=-1
    10 CALL SEKAF(CST,DCOST,ASTR,CSTR,XNORMS,B,TO,TF,ITMX,IFLAG)
    JKM = JKM + 1
    IF(IPST,JG.0) WRITE(6,600) JKM,ASTR
    600 FORMAT(5X,'J=',I5,5X,'PARAMETER=',F14.16)
    IF(FLAG.GT.1) GO TO 11
    DTF=ASTR*STF
    IF(DTF.LT.0.D0) GO TO 15
    IF(DTF.LT.0.D0) GO TO 15
    WRITE(6,200)
    200 FORMAT(1,H0,5X,'PARAMETER VALUE CAUSES LARGE TF CHANGE')
    ASTR=MANS(DTFM/STF)
    DTF=ASTR*STF
    15 TF=TF-NTH
    CALL FWINT(CST,XJ,TF,XJLAM,FCDTF,IFLAG)
    GO TO 10
    11 IF(FLAG.GT.1) GO TO 99
    C CHECK FOR SMALL CONTROL NORM CHANGE
    UNORM = ASTR*XNORMS
    IF(UNORM.LE.EPST) GO TO 98
    DTF = ASTR*STF
    TF = TPS - UTH
    C PFRFWM FORWARD INTEGRATION
    1 IFLAG=3
    CALL FWINT(CST,XJ,TF,XJLAM,FCDTF,IFLAG)
    IF(COST,LT.CSAVE) GO TO 12
    IFLAG = 2
    KFLG = KFLG + 1
IF (KFLG .GE. ITMX) GO TO 99
IF (IFLAG .EQ. 1) GO TO 10
C HAVE FOUND INTERPOLATED VALUE, UPDATE CONTROL AT FREQUENCY OF SEARCH
52 KTAU = KTAU + 1
IF (KTAU .EQ. 4) GO TO 56
IF (KTAU .GE. 1, KJ) GO TO 57
KTAU = KTAU + 1
GO TO 52
C TAU LIES BETWEEN U(KTAU-1,4) AND U(KTAU,4)
54 IF (TAU .LE. TAU(L,4)) GO TO 56
C USE LINEAR INTERPOLATION IN CTRL DIRECTION GENERATION
55 CONTINUE
GO TO 60
C USE LINEAR EXTRAPOLATION
56 KTUM = KTUM + 1
57 GO TO 58
C COMPUTE NEW FINAL TIME
58 CONTINUE
IF (TFS .GT. TSTRSTF) GO TO 62
L = 1, KJ
GO TO 64
61 L(L,1) = TEMPL(L,M)
62 L(L,4) = SEARCH(L,4)
63 CONTINUE
C CHECK CHANGE IN COST VALUES
64 IF (MANS(COST-CSAVF) .LT. ERR) GO TO 96
IF (IFLR .EQ. 1) IFR = 1
GO TO 9
C SEARCH ERRORS
65 IFR = 2
IF (IFLAG .EQ. 60, ITMX+21) IFR = 3
RETURN
C HAVE CONVERGENCE DUE TO SMALL CONTROL NORM CHANGE
68 IFR = 4
RETURN
C HAVE CONVERGENCE DUE TO NO COST CHANGE
69 IFR = 5
RETURN
C HAVE EXCEEDED PERMITTED NUMBER OF CG STEPS
72 IFR = 6
RETURN
C HAVE FAILED TO REACH ALTITUDE CUT-OFF
74 IFR = 7
RETURN
73 IF (ITFR .EQ. (ITMX+21)) 92, 9
C NOT ENOUGH STORAGE SPACE FOR GRADIENT
92 IFR = 8
RETURN
C CANNOT FIND INTEGRATION CUT-OFF POINT
IFR=9
RETURN
C HAVE CONVERGED ON GRADIENT NORM
IFR=10
RETURN
END
COMPUTE FIRST PARAMETER
ASTAR = 2 * NORM(CSTAR - COST) / NORM
IF(ALF. GE. 0.000) GO TO 10
NORM = NORM / (TF - TO) / SNORM
10 IF(ASTAR. LE. 0.000. OR. ASTAR.GT. XNORM) ASTAR = XNORM
FINT(1) = COST
ALF(1) = 0.000
IFLAG = 1
RETURN
1
XNORM = 0.000 / SNORM
GO TO 11
C SLOPE OF COST IS NOT NEGATIVE
15 IFTF (4, 100) ICOST
100 FORMAT (110, 1V., ' THE VALUE OF THE NON-NEGATIVE SLOPE IS', '024.16')
IFLAG = ITMAX + 2
RETURN

C COMPUTE SECOND PARAMETER
20 IF (IFLAG. LT. 31) GO TO 30
ALF(2) = ASTAR
FINT(2) = COST
IF(FINT(2). LT. FINT(1)) GO TO 25
ASTAR = ALF(2) / 2.000
IFLAG = 2
RETURN
25 IFLAG = 3
GO TO 31

C COMPUTE THIRD PARAMETER
30 IF (IFLAG. LT. 3) GO TO 59
ALF(IFLAG) = ASTAR
FINT(IFLAG) = COST
IF(FINT(IFLAG). GT. FINT(IFLAG - 1)) GO TO 50
31 ASTAR = ALF(2) / (2.000) ** (IFLAG - 1)
IF(IFLAG. GT. ITMAX) GO TO 40
IFLAG = IFLAG + 1
RETURN

C CANNOT FIND A MINIMUM
40 FORMAT (110)
41 FORMAT (110, 1V., ' IF THE HAS EXCEEDED MAXIMUM NUMBER OF STEPS')
IFLAG = ITMAX + 2
RETURN

C GET DATA FOR FIRST INTERPOLATION
50 IF(IFLAG. EQ. 31) GO TO 60
IFLAG = IFLAG - 3
HM(1, 1) = 1, 3
HM(1, 2) = ALF(IFLAG) - ALF(IFLAG + 1)
HM(1, 3) = (ALF(IFLAG) + ALF(IFLAG + 1)) * HM(1, 2) / 2.000
HM(1, 1) = ALF(IFLAG) ** 3 - ALF(IFLAG + 1 ** 3) / 3.000
G(1) = FINT(IFLAG) - FINT(IFLAG + 1)
GO TO 70

C GET DATA FOR THIRD POINT AND SLOPE INTERPOLATION
59 ALF(3) = ASTAR
FINT(3) = COST
60 G(1) = (ALF(3) - ALF(2))*ALF(3)*ALF(2)*2
        G(2) = FINT(2)*ALF(3)*#2 - FINT(3)*ALF(2)*#2 - ALF(2)*ALF(3)
        1*(ALF(3)*ALF(2))*DCOST - (ALF(3)*#2 - ALF(2)*#2)*FINT(1)
        G(2) = -3.00000%G(2)/G(1)
        G(3) = FINT(2)*ALF(3)*#3 - FINT(3)*ALF(2)*#3 - ALF(2)*ALF(3)
        1*(ALF(3)*#2 - ALF(2)*#2)*DCOST - (ALF(3)*#3 - ALF(2)*#3)*FINT(1)
        G(3) = 2.00000%G(3)/G(1)
        AA = G(2)
        BB = G(3)
        CC = DCOST
        GO TO 71

C  SOLVE FOR COEFFICIENTS BY CRAMER'S RULE
70 DETERM = RM(1,1)*RM(2,2)*RM(3,3) - RM(1,2)*RM(2,3)
        1 + RM(1,2)*RM(3,3) - RM(3,2)*RM(2,1)
        2 + RM(1,3)*RM(3,2) - RM(3,1)*RM(2,2))
        AA = (G(1) + RM(2,2)*RM(3,3) - RM(3,2)*RM(2,1))
        1 + RM(1,2)*RM(3,3) - RM(3,2)*RM(2,1)
        2 + RM(1,3)*RM(3,2) - RM(3,1)*RM(2,2)) / DETERM
        BB = (G(1) + RM(2,2)*RM(3,3) - RM(3,2)*RM(2,1))
        1 + RM(1,2)*RM(3,3) - RM(3,2)*RM(2,1)
        2 + RM(1,3)*RM(3,2) - RM(3,1)*RM(2,2)) / DETERM
        CC = (G(1) + RM(2,2)*RM(3,3) - RM(3,2)*RM(2,1))
        1 + RM(1,2)*RM(3,3) - RM(3,2)*RM(2,1)
        2 + RM(1,3)*RM(3,2) - RM(3,1)*RM(2,2)) / DETERM

C  COMPUTE MINIMIZING ALPHA
71 IF(RAH.RT.0.000) GO TO 73
        ASTAR = (-AA + OSORT(RH*#2 - 4.00000*AA*CC))/AA/2.000
72 IF(AR = ITMAX + 1
        RETURN
73 ASTAR = -2.00000*CC/(RH + OSORT(RH*#2 - 4.00000*AA*CC))
        GO TO 72
        FNO
C \$UPPRINT \$UPPRINT(\$COST, XI, IF \$NAME, DCTF, IFLAG)
\$UPPRINT \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
1 \$UPPRINT \$UPPRINT \$UPPRINT \$UPPRINT \$UPPRINT(ALPHA, H, I, \$NAME, \$COST, DCTF, IFLAG)
2 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
3 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
4 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
5 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
6 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
7 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
8 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
9 \$UPPRINT \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
10 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
11 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
12 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
13 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
14 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
15 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
16 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
17 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
18 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
19 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
20 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
21 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
22 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
23 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
24 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
25 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
26 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
27 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
28 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
29 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
30 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
31 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
32 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
33 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
34 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
35 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
36 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
37 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
38 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
39 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
40 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
41 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
42 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
43 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
44 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
45 \$UPPRINT(\$I, \$NAME, \$COST, DCTF, IFLAG)
IF(KOUNT .GT. KOUNTM) GO TO 101
IF(TEST) 78, 80, 72
72 M0 73 I=1,A
73 X(1,1)=XJ(I,1)
I=I+1
GO TO 74
74 M0 79 I=1,A
79 X(I,1)=XSAVE(I)
TN=1M
TESTN=TEST
TA=T-T-1
75 IF(1=.TESTP*(TN-1M))/(TESTP-TESTN)
TI=1M
TO=TMD1-TL
CALL RK713(INX,R,200,M,TOL,TI,TM,XJ,XJ,DV,P,DERIV1)
KOUNT=KOUNT+1
AL1=AL1+XJ(1,1)*O2*XJ(1,2)*O2*XJ(3,1)*O2-RE
TEST=ALT-ALTF
GO TO 71
90 GO TO(100,105,91),IFLAG
91 IF(IFLAG .EQ. 1) GO TO 100
C GET FINAL POINT
D0 92 I=1,A
92 X(I,1)=XSAVE(I)
IW=TMD1-TL
IF(T=1F-1M)
I=1M
TM=TMD1-TL
CALL RK713(INX,R,200,M,TOL,TI,TM,XJ,XJ,DV,P,DERIV1)
IF(IFLAG .EQ. 2) GO TO 104
D0 91
80 T=1M
81 COST=COSTFNM(XJ)
C CALL XLMFM(XLMF,XJ)
C COMPUTE COST CHANGE WITH RESPECT TO TF
CALL DERIV1(TF,XJ,P,1,14,1,1,RPAR,IPAR)
IF(DTFM,LE,0,000,AND,XDFM,GT,0,000) GO TO 104
D0 82 I=1,A
82 DCFM=DCF+XLMFM(I,P),I
IF(P(I,1),LT,0,000) GO TO 104
DCF=DCF+PFUN(4,P,R,1)
104 WRITE(6,600) TM,(XJ(I,1),I=1,A),(UTM(J),J=1,3)
IF(IFLAG,NE,2,AND,IPXY,GT,0) CALL OUTPUT(XJ,UTM)
WRITE(6,601) COST,(XLMF(I),I=1,7),ALT
601 FORMAT(1HO,5X,"COST FUNCTION=",1PD24.16/35X,"FINAL MULTIPLIERS/"
104,1P42.16/1A,1P30.16/6X,8ALTITUDE =",1PD24.16)
RETURN
105 COST=COSTFNM(XJ)
IF(IPRNT1 .EQ. 2) GO TO 104
RETURN
100 WRITE(6,210)
IFLAG=IFLAG+1
RETURN
101 WRITE(6,220)
IFLAG=IFLAG+2
RETURN
210 FORMAT(1HO,5X,"CUTOFF TIME ON RUN WITH ALTITUDE CUTOFF")
220 FORMAT(1HO,5X,"CUTOFF VALUE")
END
C SUBROUTINE RAKNT
 Subroutine RAKINT(X,J,XLAMF,TG,ITER,NCOST,XNORMS,NCUTF)
 IMPLICIT REAL*8(A-H,O-Z)
 DIMENSION XS(14,4),RPR(14,4),DPSAVE(14,4),FR(14,4),DV(14,4),
 2 P(14,1),TF(14,1),RPAR(1),XI(1,1),XLAMF(7),XSAVE(14),GKAI(999,4),
 3 G(999),SHRCH(999,4),TEMPS(999,4),U(999,4),IPAR(1)
 4 X3(3),VN(3),URXVH(3)
 COMMON/STAT0/XV,VR,XUPAR,VDMAG,JURXV,TO
 COMMON/CHTRL/GRAD,SRCH,US,ASTR,STF,KJIS,JK supernova,ITST,
 COMMON/CHS/IN,IC11,HVRMF,VI
 COMMON/PRINT/OUT,IC11,IPRT1,IPRT2
 EXTERNAL DERIV2
C INTEGRATION INITIALIZATION
 LNM = 0
 C REMOVED FIRST STEPER CALL
 DELT=DELTS
 10 XS(1,1)=XJ(1,1)
 11 XS(1,1)=XLAMF(1-7)
 TM=0.00D
 C REMOVED THE SECOND STEPER CALL
 INX = 1
 TOL = FRMNM
 CALL DERIV2(TM,XS,P,6,1,14,1,RPAR,IPAR,1)
 IJK=999
 C PERFORM INTEGRATION AND GRADIENT COMPUTATION
 20 CONTINUE
 TEST = TF - TM
 IF (TEST.EQ.0.0) GO TO 70
 CALL GRADFN(XS,TM,JK)
 TEST = MAX(INI,11012)
 IF (TEST.EQ.0.0) WRITE(4,600) TEST,(XS(I,1),I=1,14),(GRAD(IJK,J),
 1 J=1,3)
 4 INI = INI + 1
 50 DO 71 I=1,14
 61 XS(I,1)=XS(I,1)
 71 INJ=IJK-1
 IF (IJK LT. 11) GO TO 90
 C REPLACED STEPER WITH RK713
 TI = TM
 TM = TM + DELT
 CALL RK713(INX,14,200,MK,TOL,ITM,XS,XS,DV,P,DERIV2)
 GO TO 20
 C TERMINAL ITERATION
 70 IF (TEST.EQ.0.0) GO TO 30
 79 DO 79 I=1,14
 81 XM=TM+PLT
 IF (ITM.EQ.0) TM=TF-ITM
 C REPLACED STEPER WITH RK713
 TI = TM
 TM = TM + DELT
 CALL RK713(INX,14,200,MK,TOL,ITM,XS,XS,DV,P,DERIV2)
 TEST = TF - TM
 90 IF (TEST.EQ.0.0) WRITE(4,600) TEST,(XS(I,1),I=1,14),(GRAD(IJK,J),J=1,3)
 100 FMATITH(XM,X,STATE=1,IP)=24.1/47X,STATE VARIABLE/6X,
 11=M4/24.1/6X,-1324.1/6X,-MULTIPLICITY,6X,1P4024.1/6X,
21P3D/12.16/38x, 'GRADIENT'/6x, 1P3D/24.11)

C SHIFT GRADIENT STORAGE
  KJ=1000-IJC
  DO 31 I=1, KJ
  J=1
  31 J=J+1

31 GRAD(L+2)=GRAD(IJK+1-1.M)
C FORM GRADIENT QUADRATURE BY TRAPEZOIDAL RULE
  DO 40 K=1, KJ
  G(K)=GRAD(K,1)**2+GRAD(K,2)**2+GRAD(K,3)**2
  40 CONTINUE

GTR=0.0000
  DO 41 I=1, KJ
  GRAD(I)=G(I)+G(I-1)*(GRAD(I,1+GRAD(I-1,1))/2.000
  GRAD = GRAD + DMTR**2
  IF (GRAD < .1.F, EPS1) GO TO 101

C GET DERIVATIVE OF COST WITH RESPECT TO PARAMETER
  DCOST=8.000
C GET INITIAL SEARCH DIRECTION
  IF (TAN < .1.F, 42) GO TO 42
  XNORM = DMRTAN*(NORMX/DMRTAN)**2
  GO TO 43
42 XNORM = DSORT(NORM)
43 CONTINUE

41 IF (IPCNT2 >= 0.0) WRITE(4, A001) KJ, NORMX, DCOST, DMRTAN

A001 FORMAT(10, 5x, 1.2, 'COST SLOPE IN SEARCH =', DCOST/6x, 'COST DERIVATIVE WITH RESPECT TO TF =', 1.2, 6x, '1P3D/24.11')

C GET NEW SEARCH DIRECTION
  IF (TAN < .1.F, 51) GO TO 51
  DO 50 K=1, KJ
  DO 50 I=1, 4
  50 SEARCH(K, I) = GRAD(K, I)
  IF (DCOST < 0.0) GO TO 80
51 KTAU=2
  DO 60 I=1, KJ
  TAU=GRAD(I, 4)
  60 IF (TAU < .1.F, 52) GO TO 52
  IF (TAU > .9.F, 57) GO TO 57
  IF (TAU < .6.F, KJ) GO TO 56
  KTAU=TAU+1
  GO TO 52

C TAU IS BELOW THE LOWER LIMIT
54 IF (KTAU < .2.F, 59) GO TO 56
  KTAU=-TAU-1
  GO TO 52

C FIND SEARCH DIRECTION BY LINEAR EXTRAPOLATION
56 DO 53 K=1, 3
  TEMPS(K, I) = GRAD(K, I) + (DMRTAN/DMTAN)*SERCH(KTAU, K) + (TAN-SERCH(KTAU, 4)/(SERCH(KTAU, 4)-SERCH(KTAU, 1, 4))
  53 CONTINUE

59 CONTINUE

C FIND SEARCH DIRECTION BY LINEAR INTERPOLATION
57 DO 58 K=1, 3
  TEMPS(K, I) = GRAD(K, I) + (DMRTAN/DMTAN)*((SERCH(KTAU, K)-SERCH(KTAU, 1, K)) + (TAN-SERCH(KTAU, 4)) / (SERCH(KTAU, 4)-SERCH(KTAU-1, 4)) + SERCH(KTAU)
  58 CONTINUE
C STORE SEARCH DIRECTION

      DO 42 L=1,KJI
       DO 41 M=1,3
  41      SEARCH(L,M)=TEMPS(L,M)
  42      SEARCH(L,4)=GRAD(L,4)
      STF=DCDTF+(RETAN/RETAD)*STF
  80      KJIS=KJI
      RETAD=RETAN
      RETURN
  90      WRITE(4,200)
       ITER=ITMAX+1
      RETURN
 200    FORMAT(IHO,5X,'HAVE EXCEEDED ALLOTTED STORAGE SPACE FOR GRADIENT')
 101    WRITE(6,270)
       ITER=ITMAX+3
      RETURN
 220    FORMAT(IHO,5X,'GRADIENT NORM LESS THAN TOLERANCE')
      END
SUBROUTINE RK713
SUBROUTINE RK713(INDX,N,KT,M,TNL,TL,TF,XI,X,NDIM,TE,DERIV)
   C SEVENTH ORDER RUNGE-KUTTA INTEGRATION WITH STEPSIZE CONTROL
   C M IS THE NUMBER OF STEPS NEEDED
   C N IS THE NUMBER OF DIFFERENTIAL EQUATIONS
   C KT IS MAX NUMBER OF ITERATIONS
   C ARRAY F STORES THE 11 EVALUATIONS OF THE DIFFERENTIAL EQUATIONS
   C SUBSCRIPTS FOR ALPHA, MTA, AND CH ARE *1 GREATER THAN FEHLMERGS
   C F(I) IN FEHLERGS REPORT IS IN F(I+1,J)
   C F(I) IS IN F(I+1,J)
   C PARAMETERS FOR DPO SUBROUTINE MUST BE STORED IN COMMON
   C DIMENSIONS MUST AGREE WITH NUMBER OF DIFFERENTIAL EQUATIONS AND
   C NUMBER OF CONSTANTS IN THE PARTICULAR FEHLERGS FORMULA USED
   C
DIMENSION F(13,25),XDIM(N),TF(N),XI(N),X(N),ALPH(13),
   C PHA(13,12),CHI(13),RPAR(1),IPAR(1)
   C
CH=MTA/NAP/MTA,ALPH,MTA,CH

IPAR(1) = 0
T=1
  DU = 1-1
  X = 0
  DU 10  I=1,N
10 X(I)=XI(I)
  DU 20 CALL DERIV(T,X,TF,INDX,M,N,1,RPAR,IPAR,1)
  DU 30 I=1,N
30 F(I+1)=XI(I)
  DU 40 K=2,13
40 DU 50 I=1,N
   DU 60 XDIM(I)=XI(I)
   DU 70 XDUM(I)=XI(I)
   DU 80 XDUM(I)=XI(I)
   DU 90 X=K
   DU 100 I=1,N
   DU 110 X=K
50 DU 120 XDIM(I)=XDIM(I)+NTOTETAK(J)*F(I+1)
   DU 130 T=ALPH(K)*DT
   DU CALL DERIV(TDIM,XDIM,TF,INDX,M,N,1,RPAR,IPAR,1)
   DU 140 I=1,N
60 F(I+1)=TF(I)
70 CONTINUE
   DU 80 I=1,N
80 XDIM(I)=XI(I)
   DU 90 I=1,N
   DU 100 I=1,13
90 X(I)=X(I)+NTOTETAK(J)*F(I+1)
   DU 110 I=1,N
IF X(I) 110,100,110
100 AX=1
GO TO 120
110 AX=1
120 TF(I)=T+X(I)+F(11,1)+F(11,1)-F(12,1)-F(13,1)*41./840./A
   DU 130 TF(I)=T+X(I)+F(11,1)+F(11,1)-F(12,1)-F(13,1)*41./840./A
   DU FR=DAKS(TF(I))
   DU 140 I=2,N
   DU IF (DAKS(TF(I))-FR) 140,140,130
130 FR=DAKS(TF(I))
140 CONTINUE
150 DT=DT+T
160 X=MT
170 A=8
180 DT=AK*AX*(TNL/FR)**.125
IF (ER-TNL) 150,150,180
150 T=1+DT

IF (NOT-(TF-T)) 170,170,160
160 G1=TF-T
170 CONTINUE
G1 TO 200
I=1
190 X(1)=X+UM(I)
200 IF (N-KT) 210,220,220
210 IF (T-TF) 20,220,220
220 RETURN
END
SUBROUTINE DERIV
SUBROUTINE DERIV(T,X,P,L,M,N,NE,RPAR,IPAR,ND)
IMPLICIT REAL*4(A-H,L-Z)
REAL*4 LOAD
DIMENSION X(M,NE),P(N,NE),RPAR(ND),IPAR(ND),OMEGE(3),TEMP(3),
2 VEF(3),VR(3),CIFE(2),CIFE(3,3),GRAD(999,4),SERC(999,4),U(999,4)
3,OMEGE(3,3)
COMMON/CONS1/PVF,XXII,OMEGE,ARFA,FGAFF,GNOT,OMEGE
COMMON/TIME/GRAD,SERC,HI,ASTR,STF,TF,K.II,S,STAR
COMMON/UTRIVS/RHAG,VR,VMMAG,RHOD,VRHOD,VS,IVS,CLA,CA,FTA,DCLA,
1 UCA,UL,TA
CIMDIV/TA1H/TA11,XMSS,TEMP,TEMP
P(1,1)=X(4,1)
P(2,1)=X(6,1)
P(3,1)=X(11,1)
RMAG2=0.00D0
DO 10 I=1,3
10 RMAG2=RMAG2*X(1,1)*X(1,1)
RMAG1=DSORT(RMAG2)
RMAG2=RMAG2*RMAG1
RMAG3=RMAG2*RMAG1
C COMPUTE ACCELERATIONS DUE TO GRAVITY
DO 11 J=1,4
11 P(J,1)=X(10,J)/RMAG3
C COMPUTE RELATIVE VELOCITY
CALL ACROSHP(OMEGE,VR,0,UNITIC)
DO 12 I=1,3
12 VR(I,1)=X(14,I)-VR(1)
VMAG=DSORT(VMAG+VMAG)
C COMPUTE ATMOSPHERIC QUANTITIES AND AERODYNAMIC PARAMETERS
ALT=RMAG1*PF
CALL ATRN(ALT,TFMPK,PRES,RHOD,VS,IVS,DRHO,DPRES)
RMD=HMM(RHOD)
XMASS=WMAG*VMAG
CALL AER(XXII,CLA,CA,FTA,DCLA,DCA,DETA)
C COMPUTE AERODYNAMIC COEFFICIENTS
CIFE(1)=RHOD*ARPA*VMAG*VMAG*CLA/XMASS/2.00D0
CIFE(2)=-(2.00*TA*CA/CLA)/VMAG
DO 13 J=1,3
13 CIFE(M,J)=VR(I,J)*(2.000*ETA-1.000)/VMAG**2
DO 14 I=1,3
14 CIFE(I,J)=CIFE(I,J)+1.00D0
C ADD AERODYNAMIC ACCELERATIONS TO GRAVITY
DO 15 I=1,4
15 P(I,1)=P(I,1)+CIFE(I)*CIFE(2)*VR(I-3)
IF(IPAR(1),EQ,1) GO TO 20
C FIND CONTROL VECTOR FROM TABLE
GO TO (20,70,70,70,70,70,70,70)*L
22 IF(KE,GE,11) GO TO 25
    KT = KT + 1
    IF(TE,GE,4) GO TO 30
    GO TO 22
20 IF(KE,GE,1) GO TO 20
    KT = KT + 1
    IF(TE,GE,4) GO TO 30
    GO TO 22
70 KT = 7
20 IF(KE,GE,1) GO TO 20
    KT = KT + 1
    IF(TE,GE,4) GO TO 30
    GO TO 22
C INITIATE FOR CONTROL WHICH LIES IN INTERVAL U(KT-1,4),U(KT,4)
C TIME LIMITS OUTSIDE CONTROL ARRAY USE LINEAR EXTRAPOLATION
20 DN 26 I=1,3
   TEMP(1)=U(KT,1)-(U(KT,1)-U(KT-1,1))*(T-U(KT-1,4))/(U(KT,4)-U(KT-1,4))
   GO TO 40
C CONTINUE
C GO TO 33
C CHECK ON CONTROL OPTION FLAG
40 IF(ISTAR .LE. 0) GO TO 28
C FIND SEARCH DIRECTION FROM TABLE
GO TO (50,53,61,53,61,60),I
50 IF(T .LT. SERCH(KTS-1,4)) GO TO 60
52 IF(T .LE. SERCH(KTS,4)) GO TO 65
IF(KTS=KTS+1
GO TO 52
50 KTS=2
51 IF(T .LT. SERCH(KTS-1,4)) GO TO 55
GO TO 52
53 IF(T .LE. SERCH(KTS-1,4)) GO TO 55
GO TO 65
C INITIATE FOR SEARCH DIRECTION
60 DO 61 ,I=1,3
   TEMP(I)=SERCH(KTS-1,I)-SERCH(KTS-1,1)*(T-SERCH(KTS-1,1))/SERCH(KTS,4)-SERCH(KTS-1,4)
   GO TO 63
61 CONTINUE
62 GO TO 68
DO 66 M=1,3
   TEMP(I)=SERCH(KTS,1)-SERCH(KTS-1,1)*(T-SERCH(KTS,4))/SERCH(KTS,4)-SERCH(KTS-1,4)
   GO TO 68
66 CONTINUE
C FORM CONTROL
C GO TO 59
69 TEMP(I)=TEMP(I)-ASTR*TEM(I)
   TEMP(I)=SORT(ANTH(TEMP,TEMP))
   DO 67 I=1,3
   TEMP(I)=TEMP(I)/TEMPM
C AND CONTINUE ACCELERATIONS
70 CONTINUE
DO 41 I=1,3
   P(I,1)=P(I,1)+CNEF(I)*(CNEFM(I-3,1)*TEMP(I)+CNEFM(I-3,2)*TEMP(I))
   CNEFM(I-3,3)*TEMPM(3))
   CONTINUE
C COMPUTE HEATING DERIVATIVE
PHIHM = 1.2750
P(I,1) = TEMP(I)*SORT((RHM/RHM)*1.252D-4*VRMAG)**3.45
C COMPUTE INTEGRATED COST DERIVATIVE
C ALL ACCURATE (VIN, TEMP, TEm, 0, UNICL)
DO 42 I=1,3
   TEMP(I)=TEMP(I)/VRMAG
   ALF2=ADITH(TEMP,TEMP)
   LOADF=(KRM*VRMAG**2*ARFA/2.0)**2*(CA**2+(CLA**2+2.0*ETA*CA*CLA)
   P=ALF2 + (ETA*CLA*ALF2)**2
   LOADF = LOADF/XMASS**2
\[ P(R,1) = \text{LOAD} = (3.000) \times \mu \times (\text{RF/RE})^2 \]
\[ \text{IF} (P(R,1) > T, 0.000, P(R,1) = 0.000) \]
RETURN
END
C SUBROUTINE DERIV
SUBROUTINE DERIV2(T,X,P,L,PH,NE,RPAR,IPAR,ND)
IMPLICIT REAL *4(A-H,O-Z)
REAL*4 X(P,L),PH(NE),RPAR(ND),IPAR(ND),UTM(3),XS(11),1PS(11),UNEGH(4,3),UMGFE(3),GRAD(999,4),SFRCH(999,4),UI(999,4),
2V(P),VCRM(3,3),UMMR(3),UMNR(3)
3,DPHH(4,3),DCST(2),VECFR(3),VECFU(3),VCFV(3),UNITC(3)
COMMON/LCONS1/PI,RF,AXH,UMGFE,AREA,FCOFF,GNIT,OMFG
COMMON/CONS2/STR,B,PFMIN,GCST,DTFM,XDTFM
COMMON/DERIVS/RMAG1,VR,VRMAG,RHD,DRHD,VS,OVS,CLA,CA,ETA,DCLA,
1 DCA,DETA
COMMON/STATE/ALT,XMASS,UTM,UTMAG
COMMON/CTRL/GRAD,SFRCH,IA,ASTR,STF,TI,JSI,JKJ,ISTAR
C COMPUTE FORWARD TIME
TM=TM+1
C FIND CONTROL VECTOR
GO TO (10,19,10,19,9,8,10)
5 IF(KT2,0,1JK) GO TO 15
K1=K1+1
10 IF(TM,i,1K(I),4) GO TO 15
GO TO 5
9 K1=1JK
10 IF(TM,i,1K(I),4) GO TO 5
IF(TM,GE,4(K(I)-1,4)) GO TO 15
K1=K1-1
IF(KT2,0,1JK) GO TO 25
GO TO 10
C FIND CONTROL BY INTERPOLATION OR EXTRAPOLATION
15 DO 16 I=1,3
UTM(I)=U(KT1,1)+(I(U(KT1,1)-U(KT1,4))+(TM-U(KT1,4))/U(KT1,4)-U(KT1,3))
6 CONTINUE
17 UTMAG=ABS(ANORM(UTM,UTM))
18 DO 19 I=1,3
UTM(I)=UTM(I)/UTMAG
19 GO TO 19
C FIND CONTROL BY EXTRAPOLATION
25 K2=2
GO TO 15
C PREPARE FOR DERIV CALL
19 DO 20 I=1,7
20 XS(1,1)=X(1,1)
IPAR(I)=1
CALL DERIVI(TM,XS,PS(1,1),XHARH,1,RPAR,IPAR,1)
C COMPUTE BACKWARD STATE DERIVATIVES
30 DO 31 I=1,7
31 P(1,1)=PS(I,1)
C COMPUTE MACH NUMBER DERIVATIVE WITH RESPECT TO R
CALL ARKINS(UMGFE,VR,VECFV,0,UNITC)
33 DO 32 I=1,3
32 DVM2(1)=VRMAG*OVS*X(1,1)/VS**2/RMAG1+VECFV(1)/VS/VRMAG
C COMPUTE MACH NUMBER DERIVATIVE WITH RESPECT TO V
GO TO 32
37 DVM2(N)=VRMAG*OVS*X(1,1)/VS**2/RMAG1+VECFV(N)/VS/VRMAG
C COMPUTE UNIT PRODUCTS
XLVR=0.000
XLVRS=0.000
XLVH=0.000
GO TO 40
J=11,13
C ADD AERO DYNAMIC LOAD TO MULTIPLIERS

DO 90 I=8,10
   P(1,I) = P(1,I) + (LVRH*DRHO*X(I-7,1)/RMAG + TFM*DMYR(I-7))
   1 + DLDOVR*VECVR(I-7)/VRMA + DLDOVR*VECU(I-7))#PFU/4/XMASS**2
90 CONTINUE

DO 92 I=11,13
   P(1,I) = P(1,I) + (TFM*DMYR(I-10) + DLDOVR*VR(I-1))/VRMA
   1 + DLDOVR*UTM(I-10))#PFU/4/XMASS**2
92 CONTINUE

300 CONTINUE
P(14,1)=0.000
RETURN
END
SUBROUTINE GRADEN

IMPLICIT REAL*8 (A-H,I-O)

REAL*8, dimension (100,100) T

INTEGER, dimension (100,100) IJK, N

DATA T, IJK, N /0,0/  

DO 20 J=2,100  

T(J,J-1) = 1.0  

20 CONTINUE

C CHECK GRADIENT TIME POINT - EFFECT OF VARIABLE STEPSIZE

IF (IJG, GF, 999) GO TO 9

IF (IF, ILT, GRAD(IJK+1,4)) GO TO 9

7 IJK = IJK + 1

IF (IF, ILT, GRAD(IJK+1,4)) GO TO 9

IF (IJG, GF, 999) GO TO 8

GO TO 7

8 IJK = 999

C SET TIME

9 GRAD(IJK,4) = TF - TM

C COMPUTE RELATIVE VELOCITY

CALL ACROSHQ(OMEGH, XS, TEMP, 0, UNIC)

DO 10 J=1,5  

10 VR(J,J-1) = TEMP(J)

VIR = VIRMAG = VIRMAG = VIRMAG = VIRMAG

C COMPUTE ALTITUDE

ALI = (ALTUR, ALTUR, ALTUR, ALTUR, ALTUR, ALTUR, ALTUR, ALTUR, ALTUR, ALTUR)

C COMPUTE ATMOSPHERE AND AERODYNAMIC QUANTITIES

CALL ATMOSP(TEMP, PKS, RH, VS, IVS, URH, DPRES)

XMACH = XMACH = XMACH = XMACH = XMACH

CALL AEROH(XMACH, CLA, ETA, CLA, OCA, OFTA)

C COMPUTE DIVERGENCE

OLVR = OLVR = OLVR = OLVR = OLVR = OLVR

OLVR = OLVR = OLVR = OLVR = OLVR = OLVR

OLVR = OLVR = OLVR = OLVR = OLVR = OLVR

C COMPUTE CONSTANTS

CKA = RHO * PAF * VMAG * 2 * CLA * XMACH / 2.000

CKA = (2.0 * ETA - 1.0) * DLVR / VMAG

CKA = -OLVR + (2.0 * ETA - 1.0) * DLVR / VMAG

C COMPUTE GRADIENT

DO 12 J=1,3  

12 GRAD(IJK, J) = (XS(J,J-1) + VR(J) + KCA * UTM/(J) - KCA)

C COMPUTE AERODYNAMIC LOAD

KCA = (RHO * PAF * VMAG / 2 / 2.000)**2

LOAD = CLA**2 + (4.0 * ETA**2 + 1.0) * ETA**2 + CLA**2

LOAD = LOAD - ETA**2 * CLA**2 + 1.0) * ETA**2 + CLA**2

LOAD = LOAD + ETA**2 * ETA**2 * ETA**2 + ETA**2 + ETA**2

C CHECK LOAD MAGNITUDE

IF (LOAD > 1.0) RETURN

C COMPUTE GRADIENT OF LOAD
CKE = (RHO*AREA*VRMAG/2.0)**2/UI MAG
CKE = CKE*7.0*(4.0*ETA**2-1.0)*CLA**2*DVRU*VRMAG
1 -4.0* ETA*CLA*(CA -2.0* ETA*CLA)*VRMAG)
DO 13 I=1,3
13 TEMP(I) = CKE*(VR(I) - DVRU*UTM(I)*VRMAG)
C ADD LOAD GRADIENT TO TOTAL GRADIENT
DO 14 I=1,3
14 GRAD(IJK, I) = GRAD(IJK, I) + TEMP(I)/XMASS**2
RETURN
END
FUNCTION COSTEN

DOUBLE PRECISION FUNCTION COSTEN(X1)

IMPLICIT REAL*(A-H,O-Z)
DIMENSION X(4),XH(3),TEMP(3),URXVD(3),INDR(3),UNZ(3),CCOST(2),
IPHUN(4),VIT(3),XI(K,1),OMFGE(3),OMFGL(3,3),UTM(3)
COMMON/COMP/1/PHI,XH,OMFGE,AREA,ECOEF,GMAT,OMFGL,
COMMON/CMINS/CSTP,PH,PEHH,COST,DTFM,XTDFM,
COMMON/STATE/ALT,XMAS,UTM,UTMAG
COMMON/STATE/XH,VO,XIMAG,WJVMAG,URXVD,T0
COMMON/STATE/ALT,F,XHAG,OMAX,SINCR,COSCR,SINDR,COSDR

10 I=1,3

10 X(1)=XH(I,1)

C COMPUTE NON-RANGE AND CROSS RANGE
CALL ACROSS(URXVD,X,TEMP,1,UNZ)
CALL ACROSS(XH,URXVD,TEMP,1,UNZ)
COSTR=XH(I,UH)/XIMAG
STDR=ADTH(URXVD,XH)/XIMAG
RANG=DATAN2(SINDR,COSTR)
XAG=ACOS(SORT(ADTH(X,X)))
SINCR=ASINR(X,URXVD)/XIMAG
COSCR=ADTH(X,UNZ)/XIMAG
RANGE=DATAN2(SINCR,COSCR)
ORANGE=ORANGE
RANGE=ORANGE

C COMPUTE ALTITUDE
ALT=XIMAG+RF

C FORM COST VALUE
COSTEN=COSTR(I1)*RANGE+CCOST(I2)*ORANGE+PEHH(I1)*(ALT-ALTFF)**2+XH(I,1)
111+PEHH(4)+PEHH(2)*(XI(7,1)-OMAX)**2
COMMON END
SUBROUTINE XLAMF(XLAMF,XJ)
IMPLICIT REAL(A-H,O-Z)
DIMENSION XJ(1,1),XLAMF(7),DHDRT(3),CDRTF(3),NDRTF(3),
1 XG(3),VO(3),PEH(4),CDT(2),OMEG(3),OMEGO(3,3),UTM(3),
3 URXV(3)
COMMON/CONS1,PI,RF,XXH,OMEGE,AREA,ECDF,GNRT,OMEGU
COMMON/CONS3/CTRT,H,PEH,CDT,DTMF,XDTMF
COMMON/STATF/ALT,XTNCH,FLTANG,OMAX,SINH,COSH,SNDR,COSDR
COMMON/STATE/ALT,XHASS,UTM,UTMAG
COMMON/STATO/XO,VO,XUMAG,VOMAG,URXV,D0
C SF1 FINAL VELOCITY MULTIPLIERS
DO 10 JD=1,4,6
10 XLAMF(JD)=0.000
C SF1 FINAL HEATING MULTIPLIER
XLAMF(7)=2.0+PEH(JD)/(XJ(7,1)-OMAX)
C COMPUTE FINAL POSITION MULTIPLIERS
RF = XJ(1,1)/RF
SMT 20 JD=1,3
20 RFJF(JD)=XJ(1,1)/RF
SMT 21 JD=1,3
21 RFJF(JD)=SINH*XJ(1,1)/COSH/RF**2+URXV(JD)/RF
SMT 22 JD=1,3
RFRF(JD)=XJ(1,1)/COSDR/SINDR/RF**2-(ADTH(XO,VO,XJ(1,1))/VOMAG/XOMA
23 CONTINUE
C CONTINUE
C SF1 CONTINUE
RETURN
END
C. VECUAR OPERATIONS SUBPROGRAMS

C. CROSS PRODUCT

SUBROUTINE ACROSS(A,R,C,UNIT,UNIT)

DOUBLE PRECISION A,R,C,UNIT,CMAG,ADOTH

DIMENSION A(1),R(1),C(1),UNIT(1)

C(1)=A(2)*R(3)-A(3)*R(2)
C(2)=A(3)*R(1)-A(1)*R(3)
C(3)=A(1)*R(2)-A(2)*R(1)

IF(UNIT.LE.0) RETURN

CMAG=SQR(T(ADOTH(C,C)))
DO 1 K=1,3
1 UNIT(C(K))=C(K)/CMAG

RETURN
END

C. DOT PRODUCT

DOUBLE PRECISION FUNCTION ADOTR(A,R)

DOUBLE PRECISION A,R,ADOTH

DIMENSION A(1),R(1)

ADOTH=0.0

DO 1 K=1,3
1 ADOTH=ADOTH+A(K)*R(K)

RETURN
END
C.

SUBROUTINE OUTPUT

SUBROUTINE OUTPUT(X, UTM)

IMPLICIT REAL*8(A-H,O-Z)

DIMENSION X(6,1),UTM(3),OMEGF(3),OMEGD(3,3),UNITC(3),VR(3),

1 UNTP(3),UNIT(1,1),UNITK(3),UNITJ(3),XLIFT(3)

COMMON/CONS1/L1,X,KF,OMEGF,AREA,ECOE,F,ANGT,OMEGD

C. COMPUTE UNIT VECTORS

CALL ACROSS(OMEGF,X,VR,0,UNITC)

DO 12 I=1,3

12 VK(I) = X(I+3,1) - VR(I)

CALL ACROSS(X,VR,UNITC,1,UNTP)

CALL ACROSS(UNITP,VR,UNITC,1,UNIT)

VMAG = DSORT(ADINT(0,VR,VR))

DI = ADINT(0,0,VR)/VMAG

D2 = ADINT(1,UNIT,DI)

DI = ADINT(0,UNIT,DI)

C. COMPUTE AERODYNAMIC ANGLES

ALFA = DATAN2(DI,0.1)*180.000/PI

ALTA = DATAN2(DI,0.1)*180.000/PI

ALFA = DATAN2(1.0,DI)*180.000/PI

C. COMPUTE UNIT VECTORS FOR TRAJECTORY PLANES

UNITK(1) = X(2,1)*X(4,1) - X(1,1)*X(5,1)

UNITK(2) = X(3,1)*X(4,1) - X(1,1)*X(6,1)

UNITK(3) = X(1,1)*X(5,1) - X(2,1)*X(4,1)

UNIT = DSORT(ADINT(UNIT,UNITK))

DO 15 I=1,3

15 UNITK(1) = UNITK(1)/UNIT

CALL ACROSS(UNITK,X,UNITC,1,UNIT)

C. COMPUTE UNIT VECTOR IN LIFT DIRECTION

CALL ACROSS(X,UNIT,UNITC,1,XLIFT)

C. COMPUTE UNIT VECTOR OF PLANE ANGLE

PHIUNIT = DATAN2(FETAJ,0.1)*180.000/PI

FETAJ = ADINT(0,UNIT,DI)

FETAJ = ADINT(0,UNIT,DI)

C. COMPUTE VECTORS IN VEHICLES AXES

PSIUNIT = DATAN2(FETAJ,0.1)*180.000/PI

PSIUNIT = DATAN2(FETAJ,FETAJ)*180.000/PI

C. COMPUTE LIFT PATH ANGLE

ZETA = (X(1,1)*X(4,1) + X(2,1)*X(5,1) + X(3,1)*X(6,1))*VMAG

ZETA = (X(4,1)*UNITJ(1) + X(5,1)*UNITJ(2) + X(6,1)*UNITJ(3))*UNITJ(3)

C. FORMAT FOR PRINTING

WRITE(*,600) ALFA,PSIUNIT,DI

500 FORMAT(10,H,F,5.1)

WRITE(*,501) PHIUNIT

501 FORMAT(10,H,F,5.1)

WRITE(*,502) PHIUNIT,PSIUNIT

502 FORMAT(10,H,F,5.1)

RETURN

END
SUBROUTINE AFFORD - POLYNOMIAL FIT

SUBROUTINE AFFORD(XMACH,E,C,AN,CAN,FTAU,DCLAI,DCLO,DFTAU)
DIMEN - PRECISION XMACH,E,C,AN,CAN,FTAU,DCLAI,DCLO,DFTAU
DIMENSION A(*) = (4), C(5), D(4)

DATA A(1), ... / 5.16, 4.13, 3.22, 1.17, 1.26, 0.94, 0.64/.
1 D = 2.18, 3.36, 4.05, 1.70, 2.17, 4.74, 3.02, 2.44, 6.41, 2.10, 0.93, 2.
   XMACH = SNGL(XMACH)
1 IF(XMACH GT 1.0F-1) GO TO 10
2 CA = A(1) + XMACH*A(2) + XMACH*A(3) + XMACH*A(4) + A(5)
   1 XMACH1)
   DCA = A(2) + XMACH*(2.0*A(3) + XMACH*(3.0*A(4) + 4.0*A(5)*XMACH))
   GO TO 11
10 CA = 1.2F-2
   DCA = 0.0F0
11 IF(XMACH GT 9.0F-1) GO TO 20
2 ETA = C(1) + XMACH*C(2) + XMACH*C(3) + XMACH*C(4) + C(5)
   1 XMACH1))
   DETA = C(2) + XMACH*(2.0*C(3) + XMACH*(3.0*C(4) + 4.0*C(5)*XMACH))
   GO TO 21
20 ETA = 1.0F0
   DETA = 0.0F0
21 IF(XMACH GT 5.0F0) GO TO 30
3 ETA = N(1) + XMACH*N(2) + XMACH*N(3) + XMACH*(F(4) + XMACH
   1 N(5) + A(6)*XMACH)))
   DETA = N(2) + XMACH*(2.0*N(3) + XMACH*(3.0*N(4) + XMACH*(4.0*N(5)
   1 + 5.0*N(6)*XMACH)))
   GO TO 40
30 IF(XMACH GT 1.0F1) GO TO 31
3 ETA = N(1) + XMACH*N(2) + XMACH*N(3) + XMACH*N(4) + N(5)
   1 XMACH1))
   DETA = N(2) + XMACH*(2.0*N(3) + XMACH*(3.0*N(4) + 4.0*N(5)*XMACH))
   GO TO 40
31 ETA = 7.6F-1
   DETA = 0.0F0
40 CLAI = DLE(CLA1)
   DETA = DLE(DETA)
60 DTAU = DLE(DTAU)
RETURN
END
5 IF (ALT < R, R = RH0 + ORH0) ALRT
6 PRES = PRES + DPRES RT
7 VS = VS + DV5 RT
8 CONTINUE
9 RETURN
10 END
NOTE: The Phase II Program is built to use either single aerodynamic approximations or spline-fit aerodynamics. Both listings are presented in this Appendix. To use the simple aerodynamic approximations, use the listing of pages 68-90; to use the spline-fit aerodynamics replace MAIN, DERIV1, and DERIV2 by the listings on pages 91-98 and add the subroutine SETUP (pages 93-94).
IMPLICIT REAL(A-H,O-Z)

DIMENSION G(7), SWR(6), STVR(99,4), IN(99,2), TEMP(2), UTM(2),
2X,1(1), XLAM(6), ERAD(99,3), SCHR(99,3), Y(15,15,2)

COMMON / C3K/U, E, XMS, UMDGF, AKPA, ECOFF, GMT

COMMON / IUT(2), TCUT, EPS, EPST, EPSF, EPSA, EPSIT, ERR, ITMAX, ITMX,
2KOM, IVY

COMMON / C3K/ST, R, C, D, IFM, XOTF

COMMON / CNTRL/GRB0, SFRCH, UASTR, STF, KJS, IJKU, ISTAR

COMMON / STATE/ALT, XMASS, UTM, STVR

COMMON / STATE/ALT, XMACH, FLTANG, VR, GAMMF, TF

COMMON / PRINT/IN1T, IN2T, IPRT1, IPRT2

COMMON / STORE/DLSF, KELSD, DFLE, DFLF, DFLG, DELT, KEN

COMMON / CHAM/EP, XCH, UMDGF, AREA, ECOFF, DELTS, IKEY, TCUT, EPS,
2EPST, EPSA, EPSIT, ERR, ITMAX, ITMX, KOMTM, CSTR, H, C, DIFM, SWARN, TF, ALTF

3, XMACH, FLTANG, GAMMF, XMASS, IN1T, IN2T, IPRT1, IPRT2, VR

C READ DATA

READ(7,100) IJKU

READ(7,750) (V(1,J), J=1,3), I=1, IJKU

WRITE(6,ANAME)

C CALL CONJUGATE GRADIENT ROUTINE

CALL JPRC1(FR)

CALL JPRC2(FR, S)

CALL JPRC3(FR, S, D)

GO TO 100

END
SUBROUTINE WPRLCTR (IER)
IMPLICIT REAL (A-H,O-Z)
DIMENSION X(1),R(1),Y(1),R(1),Y(1),Z(1),S(1),T(1)

! Perform forward integration to altitude cut-off
8 ISSTAR=0
10 IFLAG=1
KRX=0
CALL FWINT (COST, X, XLAMF, DCDF, IFLAG)
15 IF (IFLAG .NE. 1) GO TO 94
20 IF (IFTER .EQ. 0) GO TO 9
30 C. Check change in cost values
50 IF (D/COST-CSAVE) .LT. EPS) GO TO 94
40 C. Perform backward integration
90 CALL BWINT (X, XLAMF, TF, ITER, DCOST, XNORMS, DCDF)
50 IF (IFTER-ITMAX) 7,95,93
70 CSAVE=COST
60 TNOM=ITF+1
WRITE (*,603) ITFNM
603 FORMAT (5X, 'ITERATION NUMBER=', I5)
6
C. FTF 1-0 SEARCH
KJLG = 1
IFLAG=7
JTNT = 0
TSGT=TF
IF (IFLAG .NE. 0) GO TO 10
10 CALL SKELF (COST, DCOST, ASTR, STR, XNORMS, R, TN, TFS, ITMX, IFLAG)
15 IF (ITMX .GT. 5) GO TO 100
20 JGMT = JTNT + 1
30 IF (IPRTNT.GT. 0) WRITE(6,600) JGMT, ASTR
600 FORMAT (5X, '1-0 SEARCH TRIAL = ', I5, ' PARAMETER = ', D24.16)
40 IF (IFLAG .GT. ITMX) GO TO 11
50 IF (IFLAG .GT. ITMX+1) GO TO 99
60 IF (IFLAG .GT. 1) CALL FWINT (COST, X, XLAMF, DCDF, IFLAG)
70 IF (G#) 80,80,80
80 IF (COST .GT. CSAVE) GO TO 10
C. Check for small control norm change
14 UNORM = ASTR*XNORMS
IF (UNORM .LT. EPS) GO TO 98
20 XNORMS=0
C. Have found interpolated value: update control at frequency of search
12 KTMJ = 1
51 IF (KTMJ .EQ. 1) GO TO 52
KTMJ=KTMJ+1
52 IF (KTMJ .EQ. 1) GO TO 54
IF (KTMJ .GE. 150) GO TO 57
KTMJ=KTMJ+1
GO TO 52
C Tau lies between \( u(k_{\text{TAU}-1},3) \) and \( u(k_{\text{TAU}},3) \)
54 IF \( k_{\text{TAU}} = 1 \) GO TO 56
C Use linear interpolation in control direction generation
55 IF \( k_{\text{TAU}} > 1 \) \# \( \text{TEMPUL}(L,K) = \text{ASRERCH}(L,K) + (u(k_{\text{TAU}},K) - u(k_{\text{TAU}-1},K)) \times \right( \text{TAU} - u(k_{\text{TAU}-1},K) \right) / \left( (u(k_{\text{TAU}},3) - u(k_{\text{TAU}-1},3)) + u(k_{\text{TAU}-1},K) \right) \)
75 CONTINUE
GO TO 60
C Use linear extrapolation
56 \# \( k_{\text{TAU}} = 2 \)
57 DO 59 \# \( K = 1,2 \)
58 \# \( \text{TEMPUL}(L,K) = \text{ASRERCH}(L,K) + (u(k_{\text{TAU}},K) - u(k_{\text{TAU}-1},K)) \times \right( \text{TAU} - u(k_{\text{TAU}},3) \right) / \left( (u(k_{\text{TAU}},3) - u(k_{\text{TAU}-1},3)) + u(k_{\text{TAU}},K) \right) \)
59 CONTINUE
GO TO 60
C Continue
60 \# \( \text{L} = 1, \text{KJIS} \)
61 \# \( \text{L} = 1,2 \)
62 \# \( \text{L} = 3, \text{KJIS} \)
63 \# \( \text{KJIS} = \text{KJIS} \)
64 CONTINUE
IF \( \text{IRF} = \text{IRF} + 1 \)
GO TO 5
C 1-n search errors
100 \# \( \text{IRF} = 2 \)
101 IF (\! \! \! \text{IFGL} \! \! \! \gt \! \! \! 0.1 \! \! \! \times \! \! \! \text{ITMX} \! \! \! + \! \! \! 2\! \! \! \) \! \! \! \text{IRF} = 3 \)
RETURN
C HAVE CONVERGENCE DUE TO SMALL CONTROL NORM CHANGE
90 IF \( \text{IRF} = 4 \)
RETURN
C HAVE CONVERGENCE DUE TO NO COST CHANGE
96 IF \( \text{IRF} = 5 \)
RETURN
C HAVE EXCEEDED PERMITTED NUMBER OF CG STEPS
95 IF \( \text{IRF} = 6 \)
RETURN
C HAVE FAILED TO REACH ALTITUDE CUT-OFF
94 IF \( \text{IRF} = 7 \)
RETURN
93 IF (\! \! \! \text{ITER} \! \! \! - \! \! \! (\! \! \! \text{ITMX} \! \! \! + \! \! \! 2\! \! \! ) \! \! \! \right) \! \! \! \geq \! \! \! 92,91,90 \)
RETURN
92 IF \( \text{IRF} = 8 \)
RETURN
C CANNOT FIND INTEGRATION CUT-OFF POINT
91 IF \( \text{IRF} = 9 \)
RETURN
C HAVE CONVERGED ON GRADIENT NORM
90 IF \( \text{IRF} = 10 \)
RETURN
99 IF (\! \! \! \text{IFGL} \! \! \! = \! \! \! \text{ITMX} \! \! \! + \! \! \! 3\! \! \! ) \! \! \! \leq \! \! \! 116,105,100 \)
C NEW SEARCH IN GRADIENT DIRECTION
105 IF (\! \! \! \text{KIFG} \! \! \! = \! \! \! 1 \! \! \! ) \! \! \! \text{GO TO 100} \!
KIFG = 1
GO 101 \# \( \text{II} = 1, \text{KJIS} \)
GO 101 \# \( \text{J} = 1,3 \)
101 \# \( \text{SPCH} (\! \! \! \text{II}, \! \! \! \text{JJ} \! \! \! ) = \! \! \! \text{GRAD} (\! \! \! \text{II}, \! \! \! \text{JJ} \! \! \! ) \)
GO 10 6
C CHECK FOR COST DECREASE
110 CONTINUE
IFLAG = 1
CALL EPROBE(COST, XJ, IE, XJAMP, DCODE, IFLAG)
IF (DCODE .GT. ESAME) GO TO 105
GO TO 14
END
SUBROUTINE SFKAIF COST, ICOST, ASTAR, CSTAR, SNORM, R, JU, TF, ITMAX, IFLAG

IFLAG = 1
C CHECK PRECISION COST, ICOST, ASTAR, CSTAR, SNORM, R, TO
IFLAG = 1, H.F, JU, 10, 30, AMS, 10, AMS, 10, AMS, 10, AMS
DIMENSION FUNT(7), ALF(20), AM(3), A(3)
IF (IFLAG GT 0) GO TO 20
IF (ICOST GT 0.0) GO TO 15
IF (JU = NE, 0.000) GO TO 11
C COMPUTE FIRST PARAMETER
ASTAR = 7.000*(CSTAR - COST)/ICOST
ASTAR = ICOST(10, 20) / SNORM
IFLAG = 1, H.F, JU, 10, 30
IF (ASTAR GT NSTAR) ASTAR = NSTAR
XSTAR = 1.0 / SNORM
IF (ASTAR LT 0.000) ASTAR = XSTAR
11 FILL = (1, 1) = 0.000
IFLAG = 1
RETURN
C MINIMIZE COST IS NOT NEGATIVE
15 FILL = (1, 1) = 0.000
RETURN
C COMPUTE SECOND PARAMETER
20 IF (IFLAG GT 1) GO TO 30
ALF(2) = ASTAR
IFLAG = (1, 1) = 0.000
IF (FILL(2) LE FILL(1)) GO TO 25
ASTAR = ALF(2)/2.000
IFLAG = 2
RETURN
25 IFLAG = 2
GO TO 31
C COMPUTE THIRD PARAMETER
30 IF (IFLAG LT 1) GO TO 39
ALF(IFLAG) = ASTAR
IFLAG = (1, 1) = 0.000
IF (FILL(IFLAG) GT FILL(IFLAG - 1)) GO TO 50
31 ASTAR = ALF(2)$*(2.000)**(IFLAG - 1)
IF (IFLAG LT ITMAX) GO TO 40
IFLAG = 1, H.F, JU, 10, 30
RETURN
C COARSEST (TH) A MINIMUM
40 FILL = TH, 10, 30
101 FILL = (1, 1) = 0.000
IFLAG = (1, 1) = 0.000
RETURN
C GET DATA FOUR POINT INTERPOLATION
50 IF (IFLAG LT 3) GO TO 60
IFLAG = (1, 1) = 3
DO 51 I = 1, 3
RM(1, I) = ALF(IFLAG) - ALF(IFLAG + 1)
RM(1, I) = ALF(IFLAG) + ALF(IFLAG + 1)**RM(1, I)/2.000
RM(1, I) = ALF(IFLAG)**3 - ALF(IFLAG + 1)**3/3.000
51 G(I) = FILL(IFLAG) - FILL(IFLAG + 1)
GO TO 70
C GET DATA FOR THREE POINT AND SLOPE INTERPOLATION
59 ALF(3) = ASTAR
FILL(3) = COST
60 G(1) = (ALF(3) - ALF(2))*(ALF(3)/ALF(2))**2
G(2) = FUNT(2) * ALF(3) #2 - FUNT(3) * ALF(2) #2 - ALF(2) * ALF(3)
1 * (ALF(3) - ALF(2)) # CST - (ALF(3) #2 - ALF(2) #2) * FUNT(1)
G(2) = -3.000 * GG(1) / GG(2)
G(3) = FUNT(2) * ALF(3) #3 - FUNT(3) * ALF(2) #3 - ALF(2) * ALF(3)
1 * (ALF(3) #2 - ALF(2) #2) * CST - (ALF(3) #3 - ALF(2) #3) * FUNT(1)
G(3) = 2.000 * GG(3) / GG(1)
AA = G(2)
MM = G(3)
CC = CST
GO TO 71
C SOLVE FOR COEFFICIENTS USING CRAMER'S RULE
70 DETERM = HM(1,1) * HM(2,2) * HM(3,3) - HM(1,2) * HM(3,2) * HM(2,3)
1 + HM(1,2) * HM(3,2) * HM(2,3) - HM(1,3) * HM(2,2)
2 + HM(1,3) * HM(2,2) * HM(3,1) - HM(1,1) * HM(2,3)
AA = (G(1) * HM(3,2) * HM(2,3) - HM(1,3) * HM(2,2)) / DETERM
1 + G(2) * HM(3,2) * HM(1,3) - HM(1,2) * HM(2,3)
2 + G(3) * HM(1,2) * HM(2,3) - HM(2,2) * HM(1,3)) / DETERM
MM = (G(2) * HM(3,2) * HM(3,3) - HM(2,3) * HM(3,1)) / DETERM
1 + G(1) * HM(3,3) * HM(2,1) - HM(2,3) * HM(1,1)
2 + G(3) * HM(1,1) * HM(2,3) - HM(1,2) * HM(2,1)) / DETERM
CC = (G(2) * HM(3,3) * HM(3,3) - HM(3,3) * HM(2,2)) / DETERM
1 + G(1) * HM(2,2) * HM(2,1) - HM(2,2) * HM(1,1)
2 + G(3) * HM(3,1) * HM(2,2) - HM(2,1) * HM(1,1)) / DETERM
C COMPUTE MINIMIZING ALPHA
71 IF (BR > 0.0000) GO TO 73
ASTAR = - (MM + NSORT(1MM #2 - 4.0000 * AA * CC)) / AA / 2.000
72 IF (ALG = IMAX + 1
RETURN
73 ASTAR = -2.0000 * CC / (MM + NSORT(BR #2 - 4.0000 * AA * CC))
GO TO 72
END
SUBROUTINE FPOINT(CNST, XI, XLAMF, DCOIF, IFLAG)

IMPLICIT REAL*8(A-H, O-Z)

DIMENSION YPR(A, 1), DPSAVE(A, 1), DVT(A, 1), P(R, 1), T(R, 1), RPAR(1), 
ZPAR(1), SVAR(R, 1), XLAMF(6), C(7), STVKS(999, A), DEP(R, 1), UTM(2), A(4)

R(4)

COMMON/CONS1/P1, PF, XMTH, OMEGFA, AREA, COEFF, GNT
COMMON/CONS2/DELTS, TC1T, EPS1T, EPS2T, EPS1T, EPS2T, ITMAX, ITMX

COMMON/STATE/ALT, X455S, UTM, STYRS
COMMON/STATO/SVARO, TO
COMMON/STATE/ALT, XMACH, FLTANG, VF, GAMKTF, TF
COMMON/PRINT/IOUT, IUNIT, IPRINT, IPRINT2
COMMON/STORE/DELSV, DELSF, DELGE, DELT, KEN

C initialization

IF(TST=SVRDO(1)-RE)
TEND=TC1T
IML=10/IOUT-1

IPAR(1) = 0
SIX = 24.000
ZER = 0.000

C

A(1) = -9.000/SIX
A(2) = 37.000/SIX
A(3) = -59.000/SIX
A(4) = 55.000/SIX
B(1) = 1.000/SIX
B(2) = -5.000/SIX
B(3) = 19.000/SIX
B(4) = -A(1)

C

RATIN = 19.000/270.000
SIX = 6.000
TWO = 2.000

C

K1 = 1
K2 = 1
K3 = 2
K4 = 3
DEL1 = DELTS
111 DEP(1, 1) = SVRDO(1)
DEP(7, 1) = 0.000
DEP(R, 1) = 0.000

K = 1

TM = 10

20 ASSIGN 100 TO IPL5
NGL = R

DEL2 = DEL1/SIX
DEL3 = DEL1/TWO

C

DO 21 JAY = 1, N

DO 22 J = 1, N

DPSAVE(J, JAY) = DEP(J, JAY)

22 DVT(J, JAY) = DEP(J, JAY)

CALL DERIV1(TM, NSW, P, L, M, N, NE, RPAR, IPAR, 1)
C) EVALUATING THE DERIVATIVES AND STORING THE VALUES

DON 25 JAY = 1, NE
DON 26 J = 1, N

26 YPR(J, M1, JAY) = P(J, JAY)

25 CONTINUE

IF(INX .EQ. 3) GO TO 30

COST = 0.000
NCATF = 0.000
KCONT = 0

C) PERFORM INTEGRATION AND COMPUTE COST

202 CONTINUE

ON 320 X = 1.8

320 UPSAVF(JI, 1) = DEP(JI, 1)
ON 377 X = 1.6

377 SURF(JI, 1) = DEP(JI, 1)

K = K + 1

IF(K .GT. 999) GO TO 451

CNT = CNT + 1

IF(CNT = PFS) N = LAST
 IF(LK, 1, 1) .NE. L11 & L0, MN) WRITE(A, 600) TM, (DEP(I, 1), I = 1, K)

2) WRITE(I, 11, 2)

C) REPLACED BY MK713

30 V = 1.8

1 = V + DEP

01150

C)

GO TO 1PL5, (100, 200)

01160

C) ENTRY Adams PREDICTOR-CORRECTOR

01170

C) APPLICATION OF THE PREDICTOR EQUATION AND THE DERIVATIVE EVALUATION

01180

200 DO 220 JAY = 1, NE

210 J = 1, N

TE(J, JAY) = (A3)YPR(J, M1, JAY) + R(2)YPR(J, M2, JAY) + R(1)YPR(J, M3, JAY)

01190

01200

01210

01220

01230

C)

APPLICATION OF THE CORRECTOR EQUATION AND THE DERIVATIVE EVALUATION

01240

260 DO 270 JAY = 1, NE

270 J = 1, N

DV(J, JAY) = DEP(J, JAY) + DELT*(A4)YPR(J, M1, JAY) + DELT*(A3)YPR(J, M2, JAY)

01250

01260

01270

01280

01290

C)

SECOND APPLICATION OF THE CORRECTOR EQUATION AND COMPUTING

01300

C) THE SINGLE STEP ERROR

01310

C)

DON 250 JAY = 1, NE

250 J = 1, N

YPR(J, M4, JAY) = P(J, JAY)

01320

01330

01340

01350

01360

01370
5000  M0  =  M4
M4  =  M3
M3  =  M2
M2  =  M1
M1  =  0.0

IF(INX  .EQ.  2) GO TO 202
IF(TM  .GE.  TEND) GO TO 400
AL = DEP(I,1) - RH
TEST = AL - ALT
 IF(DAMSTEST) .LT. EPSA) GO TO 80
 IF(TEST) 79, 80, 90
90 IF(INX  .EQ.  1) GO TO 202
C FINAL ALTITUDE ITERATION
70 KOUNT = K(1,1)
19K = 3
 IF(KPONT  .GE.  KOUNT) GO TO 101
 79 END 79  I = 1, 8
79 DEP(I,1) = DEPSAVE(I,1)
 TM = TM - DFLT
MFT = MFT - DAMSTESTP(TESTP - TEST)
 GO TO 20
 80 IH = TM
 MFI = MFI - DFLT
 K = K
 80   K = K + 1
 300 SIMN(I, J) = DEP(J, 1)
K1 (J) = IGN(I,J) - DFLT)
 CALL XIAMLX(I, J, ADF*H*J, IH, IMH)
104 WRITE(6, 495) SIMN(I, J), DEP(J, 1), (UTM(I,J), J = 1, 2)
 IF(INX  .GE.  1) AND (1YX(i, J) = 0) CALL OUTPUT(DFP, UTM)
 120 XIAJ(1, 4I1) (J = YIAM(L), L = 1, 6)
 600 FORMAT(1HO, 5X, 'TIME = ', X, 'P024.16/32X, 'STATE/6X, 1P+24.16/
1 6X, 1P+24.16/35X, 'CONTROL/6X, 1P+24.16/
601 FORMAT(1HO, 5X, 'COST FUNCTION= ', X, 'P024.16/35X, 'FINAL MULTIPLIERS'/
1 6X, 1P+24.16/35X, 'P2024.16/6X)
 RETURN
105 COST = COSTFN(DFP)
105 IF(1P0410.16) 204, 101
101 RETURN
101 WRITE(6, 410)
 IFLAG = IFLAG + 1
 RETURN
110 WRITE(6, 420)
 IFLAG = IFLAG + 3
 RETURN
410 FORMAT(1HQ, 5X, 'EXCEEDED CUTOFF TIME ON RUN WITH ALTITUDE CUTOFF')
420 FORMAT(1HQ, 5X, 'EXCEEDED MAXIMUM NUMBER OF ITERATIONS IN TERMINAL C
20410 C ENTRY RUNG KUTTA
C
100  VV  =  V  +  DELVY2

C   DO 120 JAY = 1, NE
110 JAY = JAY + 1
110 LAY(JAY) = DEP(JAY) + YPR(J, M1, JAY) * DELVY2
 CALL DERIV(3V, VV, 1, L1, M, N, NE, RPAR, IPAR, 1)
120 CONTINUE
C   DO 140 JAY = 1, NE
130 J = J + 1
130  DVI(J,JAY) = DEP(J,JAY) + P(J,JAY) * DELY2
    CALL DFRIV1(VV,OV,TF,L,M,N,NE,RPAR,IPAR,1)
140  CONTINUE
C
150  IF(J,JAY) = 1, M
160  IF(0.0, 2, 0.0, (TF(J,JAY) + P(J,JAY))
    CALL DFRIV1(TK,OV,P,L,M,N,NE,RPAR,IPAR,1)
170  CONTINUE
C
180  IF(J,JAY) = 1, M
190  YPR(J,N,JAY) = P(J,JAY)
180  CONTINUE
C
KOUNT = KOUNT + 1
IF (KOUNT .LT. 3) GO TO 5000
ASSIGN 200 TO IPLS
INX = 1
GO TO 5000
651 WRITE(6,650)
STOP
650 FORMAT(*, 'EXCEED STATE VAR. STORAGE')
END
SUBROUTINE RAINT(XJ,XLAMF,TG,ITER,DCOST,XPHMS,DCUTF)

[Blocks of code and comments]

C  INITIALIZATION

10 6 = 24,000
7 = 0,000

C

A(1) = -9,000/SIX
A(2) = 87,000/SIX
A(3) = 59,000/SIX
A(4) = 50,000/SIX
B(1) = 1,000/SIX
B(2) = -5,000/SIX
B(3) = 19,000/SIX
B(4) = -A(1)

C

RATIO = 19,000/270,000
SIX = 6,000
TWO = 2,000

C

L=1JKH
M=1
N=6

M1 = 4
M2 = 1
M3 = 2
M4 = 3

DELT = DELSV
Nd = 111,6

111 DFP(1,1) = XLAMF(1)
T = 0
NFK = KFN

900 FNXMA(1,1,214)

1JK=999
1=0,1=1,2=2,3=3

20 ASSIGN 100 TO IP5S
KOMN = 0
DFLY = DELT/SIX
DFLY2 = DELT/TWO
C

600 DD 21 JAY = 1,M
600 DD 22 J = 1,N
600 DSAVE(J,JAY) = DFP(J,JAY)
20 DVL(J,JAY) = DFLX(J,JAY)
600 CALL DFNYT2(TM,DVL,P,L,NEK,O,RPAR,IPAR)

71 CONTINUE

C

CALL GRDFN(DEP,TM,1JK)
C ENSURING THE DERIVATIVES AND STORING THE VALUES

C

DO 25 JAY = 1, NE
   DO 26 J = 1, M
      YPR(J,M,JAY) = P(J,JAY)
   26 CONTINUE
C PERFORM INTEGRATION AND COMPUTE GRAD.
      V = TM
      TM = V + DELT
      NEK = NEK - 1
      GO TO IPL5 (100, 200)
      C ENTRY ADAMS PREDICTOR-CORRECTOR
      C APPLICATION OF THE PREDICTOR EQUATION AND THE DERIVATIVE EVALUATION
      DO 220 JAY = 1, NE
         DO 210 J = 1, N
            TF(J,JAY) = R(1) * YPR(J,M,JAY) + R(2) * YPR(J,M2,JAY) + A(1) * YPR(J,M3,JAY)
            DV(J,JAY) = DER(J,JAY) + DELT * R(4) * YPR(J,M,JAY) + TF(J,JAY)
            DV(J,JAY) = DV(J,JAY) + DELT * A(2) * YPR(J,M3,JAY) + A(3) * YPR(J,M2,JAY)
         210 CONTINUE
         CALL DERIV2(TM, DV, P, L, NEK, 0, RPAR, IPAR)
      220 CONTINUE
      C APPLICATION OF THE CORRECTOR EQUATION AND THE DERIVATIVE EVALUATION
      DO 240 JAY = 1, NE
         DO 230 J = 1, N
            DV(J,JAY) = DER(J,JAY) + DELT * R(4) * P(J,JAY) + TF(J,JAY)
         230 CONTINUE
         CALL DERIV2(TM, DV, P, L, NEK, 0, RPAR, IPAR)
      240 CONTINUE
      C SECOND APPLICATION OF THE CORRECTOR EQUATION AND COMPUTING THE SINGLE STEP ERROR
      DO 250 JAY = 1, NE
         DO 250 J = 1, N
            YPR(J,M4,JAY) = P(J,JAY)
            DER(J,JAY) = DER(J,JAY) + DELT * R(4) * P(J,JAY) + TF(J,JAY)
            DV(J,JAY) = DER(J,JAY)
         250 CONTINUE
      5000 M0 = M4
            M4 = M3
            M3 = M2
            M2 = M1
            M1 = M0
      C
      DO 320 J = 1, N
      320 DPSAVE(J,1) = DER(J,1)
      IF (TEST .LE. 1.0D-1) GO TO 500
      CALL DERIV2(TM, DV, P, L, NEK, 0, RPAR, IPAR)
      CALL GRADFIN(DER, TM, 1JK)
(LMK=LM+1)
LTEST=MOD(LMN,40)
IF(LTEST .EQ. 0) WRITE(6,600) TEST,(STVRS(NEK,K),K=1,6),
(DFI(1,1),I=1,6), (GRAD1JK+1,1), (J=1,2)
GO TO 30
C ENTRY RUNGE KUTTA
C
100 VV = V + DELAY2
C
DO 120 JAY=1,NE
DO 110 J = 1,N
110 NV(J,JAY) = DEP(J,JAY) + YPR(J,M1,JAY)*DELAY2
CALL DERIV2(NV,DV,P,L,NEK,1,RPAR,IPAR)
120 CONTINUE
C
DO 140 JAY=1,NE
DO 130 J = 1,N
130 NV(J,JAY) = DEP(J,JAY) + P(J,JAY)*DELAY2
CALL DERIV2(NV,DV,TF,P,L,NEK,1,RPAR,IPAR)
140 CONTINUE
C
DO 160 JAY=1,NE
DO 150 J = 1,N
NV(J,JAY) = DEP(J,JAY) + TE(J,JAY)*DELT
150 TE(J,JAY) = 2.000 * (TF(J,JAY) + P(J,JAY))
CALL DERIV2(TF,DV,P,L,NEK,0,RPAR,IPAR)
160 CONTINUE
C
DO 180 JAY=1,NE
DO 170 J = 1,N
DEP(J,JAY) = DEP(J,JAY) + DELHA*(P(J,JAY) + TE(J,JAY)*YPR(J,M1,JAY))
170 CALL DERIV2(TF,DV,P,L,NEK,0,RPAR,IPAR)
DO 190 J = 1,N
190 YPR(J,M4,JAY) = P(J,JAY)
180 CONTINUE
C
IF(INX .LE. 3) GO TO 800
KINT = CIHRT + 1
IF (KINT .LE. 3) GO TO 5000
ASSIGN 200 TO IPL5
INX=1
GO TO 5000
800 INX=1
DELT=DELT
GO TO 20
500 CALL DERIV2(TF,DV,P,L,NEK,0,RPAR,IPAR)
CALL GRADFN(DEP,TM,1JK)
WRITE(6,600) TEST,(STVRS(NEK,K),K=1,6), (DEP(1,1),I=1,6), (GRAD1JK+
21,1), (J=1,2)
600 FORMAT('TIME=',1PD24.16,'47X',STATE,1PD24.16,16/'
24X,MULTIPLES,16/6X,1PD24.16/16/30X,GRADIENT,16/1P3
2416)
C SHIFT GRADIENT STORAGE
KJI=999-1JK
DO 830 L=1,KJI
DO 830 M=1,3
830 GRAD(L,M)=GRAD1JK+L,M)
C FORM GRADIENT QUADRAUTURE BY TRAPEZODEAL RULE
DO 40 K=1,KJI
40 20
G(Y)=GRAD(K,1)*GRAD(K,1)+GRAD(K,2)*GRAD(K,2)

40 CONTINUE
KFIANG=0.0
J=1
DO 41 L=2,J
41 KFIANG=KFIANG+(G(L)+G(L-1))#DELS/2.000
KFIANG=KFIANG+(G(KJ)+G(KJ-1))#DELSV/2.000
IF(KFIANG#E. EPSIT) GO TO 101
C GET DERIVATIVE OF COST WITH RESPECT TO PARAMETER
C COST=KFIANG
C GET NORM OF SEARCH DIRECTION
IF(10FR#E. 0.0) GO TO 42
XNORMS=ISORT(KFIANG*KFINAN(XNORMS/KFIANG)**2)
GO TO 43
42 XNORMS=ISORT(KFIANG)
43 CONTINUE
IF(LIST#E. 0.0) WRITE(6,601) KFIANG,XNORMS,COST
601 FORMAT(1X,5H10,Y,2,1X,GRADIENT NORM SQUARED =',1PD24.16/
21X, SEARCH DIRECTION NORM =',1PD24.16/X,1X,COST SLOPE IN SEARCH DIRECTION =',1PD24.16)
C GET NEW SEARCH DIRECTION
IF((1IFR/5)<5.0#E. 1FR) GO TO 51
DO 50 K=1,KJ1
50 DO 50 L=1,J
50
51 DELS=DELS
DELS=DELS
DO 60 L=1,KJ1
IF(L#E. KJ1) GO TO 202
IF(L#E. KJ1) GO TO 400
50 105 K=1.2
105 TEMP(L,K)=GRAD(L,K)*KFIANG#FERCH(L,K)
GO TO 60
202 DELS=DELS
IF(L#E. KJ1) DELS=DELS
DO 300 K=1,J
200 DELS=DELS
300 TEMP(L,K)=GRAD(L,K)*KFIANG#FERCH(KJ1,K)
GO TO 60
400 DELS=DELS
DO 500 K=1,J
500 TEMP(L,K)=GRAD(L,K)*KFIANG#FERCH(L-1,K))
2(FERCH(L,K)-FERCH(L-1,K))
60 CONTINUE
C STORE SEARCH DIRECTION
DELS=DELS
DO 62 L=1,KJ1
62 DO 62 M=1,L
62
80 KJ1=KJ1
IFAD=KFIANG
RETURN
101 WRITE(6,225)
11 FR=1#MAX+3
RETURN
225 FORMAT(1H0,5X,*GRADIENT NORM LESS THAN TOLERANCE*)
END
GO TO 21
M=N-1
GO TO 21
10 M=62+1,
61 M=(M+1)*((SERC(M,1)-SERC(M-1,1))/(SERC(M,3)-SERC(M,1)))
45 M=-1, 51:GO TO 10

100 CONTINUE
C
C
C
C
C
C
C
C
SURROUTINE DERIV2(TS,DXM,PL,ML,MR,RPAR,IPAR)

IMPLICIT NONE

DIMENSION XS(A,1),STVRS(999,6),GRADN(999,3),SFRH(999,3),U(999,3),
                  XP(A,1),X(6,1),P(6,1),C(7)

COMMON/CONS1,P,RE,XMH,OMEGE,AREA,ECOE,F,GNT
COMMON/CONS3/STR,HG,DTFM,DFTFM
COMMON/CI,GRADN,GRAM,SECH,UI,ASTR,STF, KJIS,JKJ1,1STAR
COMMON/STATE/ALT,XMASS,TEMP,STVRS

C PRINT STATE VARIABLES FROM STORAGE
C
1 IF (N+1,N) GO TO 10
C
200 X(I,1)=STVRS(K,I)
   C
10 IF (M+1,M) GO TO 200
   C
40 CONTINUE
   C FIND CONTROL
20 IF (L,GE,1JKH) GO TO 60
   C
50 IF ((T,LT,1) GO TO 50
   C
55 L=L+1
   C
60 IF ((T,LT,1) GO TO 55
   C
65 L=L-1
   C
70 IF ((T,LT,1) GO TO 70
   C
75 CONTINUE
   C COMPUTE TRIG QUANTITIES
COS=DCOS(TEMP(2))
SIN=DSIN(TEMP(2))
COS=DCOS(X(3,1))
SIN=DSIN(X(3,1))
COS=DCOS(X(5,1))
SIN=DSIN(X(5,1))
COS=DCOS(X(6,1))
R=X(6,1)
RMAG2=R*R
RMAG3=R*RMAG2
V=X(4,1)
C COMPUTE ATMOSPHERIC PARAMETERS
ALTI=X(1,1)-RE
CALL ATMOS(ALTI,TEMP,PRES,RH0,VS,DVS,DRH0,DPRS)
XMACH=V/VS
RH0=NARS(RH0)
C COMPUTE AERON . COEF.
N=RHO*AREA*V*V/(2.00*XMASS)
G0=2.000*DSIN(TEMP(1))**3+0.00-2
GL=2.000*DSIN(TEMP(1))**2*DCOS(TEMP(1))+1.00-2
NC1=M=0.000
DC1=M=0.000
SUBROUTINE OUTPUT(XJ, UTM)

IMPLICIT REAL*8(A-H,O-Z)
DIMENSION X(1:11,1:UTM(2),SVAR(6))
COMMON/CONS1/PI,RF,XMU,OMGFE,AREA,EG0E,GN0T
COMMON/STATO/SVAR0,T1
CRANG=(XJ(3,1)-SVARN(3))*RE
BANG=(XJ(2,1)-SVARN(2))*RE
ALTI=XJ(1,1)-RF
CALL A105S(ALTI,TEMPR,PRES,RHO,VS,DVS,DRHO,OPRES)
X=ACH=XJ(4,1)/VS
HEAD=XJ(6,1)*180.00/P1
FLTA=XJ(5,1)*180.00/P1
ANGAT=UTM(11)*180.00/P1
HANK=UTM(2)*180.00/P1
WRITE(6,700) ALTI,XH,ACH,CRANG,BRANG,HEAD,FLTA,ANGAT,HANK
700 FORMAT(10X,'FINAL ALTITUDE',19X,'MACH NUMBER',14X,'CROSS RANGE',
2X,'DOWN RANGE',14X,'074.16,10X,024.16,16X,010X,024.16/
310X,'HEADING ANGLE',15X,'FLIGHT PATH ANGLE',15X,'ANGLE OF ATTACK',
4X,'HANK ANGLE',11X,'074.16,10X,024.16,16X,010X,024.16)
RETURN
END
SUBROUTINE XLAMFN(XLAMF, X, L, M)

IMPLICIT REAL*8(A-H,O-Z)

DIMENSION X(1), XLAMF(1), SVARN(6), C(7), UTM(2), P(8), RPAR(1)

COMMON/COMS/PI,KF, XMAX, HMF, AREA, ECNOF, GNOT
COMMON/STATO/SVARN,T0
COMMON/STATE/ALT, XMASH, UTM
COMMON/STATE/ALT, XMAX, FLTANG, VF, GAMMF, TF

C

COMPUTE FINAL MULTIPLIERS

XLAMF(1) = 0.0
XLAMF(5) = 2.0 * C(4) * (X(5) - GAMMF)
XLAMF(4) = 2.0 * C(3) * (X(4) - VF)
XLAMF(3) = C(1) * KF * X(3) + 2.0
XLAMF(2) = C(2) * KF * X(2) + 2.0
CALL DERIV(TF, X, P, L, M, R, 1, RPAR, 0, 1)
XLAMF(1) = P(7, 1) + C(6) * P(8, 1)

100 XLAMF(1) = XLAMF(1) + XLAMF(1) * P(1, 1)
XLAMF(1) = -XLAMF(1) / P(1, 1)
RETURN
END
FUNCTION COSTF(X1)

IMPLICIT REAL(*-A+(H+D-Z)
DIMENSION SWARD(A),C(7),XI(R+1),UTM(2)
COMMON/CONS1/H,XMU,AREA,ECHEF,GNOT
COMMON/CONS2/C,TM,CT,NT,XTFM,XDTFM
COMMON/STATO/STKID
COMMON/STAF/ALT,FH,TANG,VF,GAMMF,TF
ORANGE=RE*X(2,1)*XI(2,1)
ORANGE=RF*X(3,1)*XI(3,1)
FORM COST VALUE
COSTF=C(1)*ORANGE+C(2)*ORANGE+C(3)*XI(4,1)-VF)*2+C(4)*XI(5,1)-
2*RF*VF)*2+C(5)*XI(7,1)+C(6)*XI(8,1)
RETURN
END
IF(ALT>8.A.
RHO=RHO+DRHO@ALT
PRES=PRES+DPRES@ALT
VS=VS+DVS@ALT
CONTINUE
RETURN
END
C CALL CONJUGATE GRADIENT ROUTINE
C CALL WPJAC(FER)
GO TO (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110) FER
10 CONTINUE
20 WRITE(A,520)
GO TO 101
30 WRITE(A,530)
GO TO 101
40 WRITE(A,540)
GO TO 102
50 WRITE(A,550)
GO TO 101
60 WRITE(A,560)
GO TO 101
70 WRITE(A,570)
GO TO 101
80 WRITE(A,580)
GO TO 101
90 WRITE(A,590)
GO TO 101
100 WRITE(A,600)
101 CONTINUE
WRITE(A,625) IJKU
WRITE(A,650) ((IJK, I), J=1, 3), I=1, IJKU
STOP
502 FORMAT(214)
505 FORMAT(150, 5X, 'ONE-D SEARCH FAILED TO FIND A MINIMUM')
530 FORMAT(150, 5X, 'COST IS NOT DECREASING IN SEARCH DIRECTION')
540 FORMAT(150, 5X, 'CONVERGENCE ON SMALL CONTROL CHANGE')
550 FORMAT(1HO,5X,'LITTLE COST CHANGE IN LAST TWO ITERATIONS')
560 FORMAT(1HO,5X,'FAILED TO CONVERGE IN ITMAX ITERATIONS')
570 FORMAT(1HO,5X,'INITIAL TRAJECTORY FAILED TO REACH CUT-OFF ALT')
580 FORMAT(1HO,5X,'TOO MANY INTEGRATIONS STEPS REQUIRED')
590 FORMAT(1HO,5X,'BACKWARD INTEGRATED TRAJECTORY PRUNES')
600 FORMAT(1HO,5X,'CONVERGENCE ON ZERO GRADIENT NORM')
625 FORMAT(1H,15)
650 FORMAT(1H,3076,16)
700 FORMAT(15)
750 FORMAT(3026,16)
END
C INTERPOLATION BY PIECEWISE CUBIC SPLINES
C INPUTS: N1,N2 NUMBER OF DATA POINTS
C Y(N1,N2,2)= DATA TABLE

N2=N1-1
M=N2-1
DO 40 J=1,2
DO 40 I=2,N2
INDX=I-1

11 A(2,INDX,J)=Y(I+1,L,J)-Y(I,L,J)
   DINV(1,INDX,J)=Y(I+1,L,J)/DX(I,INDX,J)
   T(I)=0,0

17 A(I,INDX,J)=A(I-1,INDX,J)-A(I-1,INDX,J)+A(I-1,INDX,J)*T(I-1)

13 A(I,INDX,J)=A(I,INDX,J)*T(I)+A(I-1,INDX,J)

40 CONTINUE
C FIT SPLINE
C YY,XY,CL,DL,DLDA,DLDDA,DCDDA

21 IF(XX .LT. Y(K(INDX)+1,1)) GO TO 23
   IF(XX .GT. Y(K(INDX)+1,1)) GO TO 24
   DO 28 J=1,2
   DO 28 I=2,N2
   J=1
   X=(XX-Y(K(INDX)+1,1))/DX(K(INDX),M,J)
   N=M
   S=M,J
   R(Y(K(INDX)+1,1),Y(K(INDX)+1,1),L,J)=Y(K(INDX)+1,1,M,J)+Y(K(INDX)+1,1,M,J)*T(I-1)
   M=M+1

28 R(Y(K(INDX)+1,1),Y(K(INDX)+1,1),L,J)=Y(K(INDX)+1,1,M,J)+DX(K(INDX)
   1,M,J)*(3,DX=0=V=0=1,NO)*A(K(INDX)+1,M,J)-(3,DX=0=V=0=1,NO)*A(K(INDX)
   1,M,J)
   X=X+1
   XY=YY+Y(K(INDX)+1,1)
   YY=XY

20 CONTINUE
C IF(YY .GT. Y(1,M2,1)) GO TO 250
   IF(YY .LT. Y(1,M2,1)) GO TO 21
   IF(K(INDX) .EQ. N1) XX=Y(N1,1,1)
   IF(K(INDX) .EQ. N1) XX=Y(N1,1,1)
   GO TO 21

25 CONTINUE
C IF(YY .GT. Y(1,M2,1)) GO TO 250
   IF(YY .LT. Y(1,M2,1)) GO TO 21
94

DC 120  [I=1,MM]
DM(I,J)=Y(I+1,J)-Y(I,J)
DH(I,J)=S(I,J)-S(I,J)/DM(I,J)
120 DC(I,J)=R(I,J)-R(I,J)/DM(I,J)
T(I)=0.0
140 IF I=2,MM
PV=Z,DM(I-1,J)+DM(I,J)-DM(I-1,J)*T(I-1)
T(I)=DM(I,J)/PV
R(I,J)=DM(I,J)-DM(I,J)*R(I,J)/PV
140 C(I,J)=DC(I,J)-DC(I,J)-DM(I,J)*C(I,J)/PV
160 IF I=30  \#S2,MM
[\#I=1,1
R(I,J)=T(I,J)-T(I,J)*R(I,J)
130 C(I,J)=C(I,J)-C(I,J)*C(I,J)
400 CONTINUE
210 IF (YY.LT. Y(I+1),K(INDX)+1,1))  GO TO 230
210 IF (YY.GT. Y(I-K(INDX)+1,1))  GO TO 240
180 IF K=1,2
U=(YY-Y(I,K(INDX)+1,1))/DM(K(INDX),J)
Z=1,0
Z=1,0
Z=1,0
220 IF J=1,\#S(K(INDX)+1,1)+Z*\#S(K(INDX),J)+DM(K(INDX),J)*DM(K(INDX),J)
220 IF J=1,\#S(K(INDX)+1,1)+Z*\#S(K(INDX),J)+DM(K(INDX),J)*DM(K(INDX),J)
220 IF J=1,\#S(K(INDX)+1,1)+Z*\#S(K(INDX),J)+DM(K(INDX),J)*DM(K(INDX),J)
GO TO 230
230 IF K(INDX)=K(INDX)-1
IF K(INDX).EQ.0  YY=Y(1,7,1)
GO TO 210
240 K(INDX)=K(INDX)+1
GO TO 210
END
GO TO 21
56 M=M-1
GO TO 21
61 DO 62 I=1,2
62 TFNI=SERCH(M,I)*((SERCH(M,I)-SERCH(M-1,I))/SERCH(M,3)-SERCH(M-1,3))*T-SERCH(M,3)
GO CONTINUE
C FLOW CONTROL
TEMP(1)=TEMP(1)-ASTR*TEM(1)
TEMP(2)=TEMP(2)-ASTR*TEM(2)
100 CONTINUE
C COMPUTE AERON. COFF
CALL SPLINE(TFNP11, XMACH, CL, CGL, DGLM, DGLA, DCOM, DCUA)
C AND CONTROL ACCERTIUMS
P(4,1)=P(4,1)-DCL
P(5,1)=P(5,1)+DCL*2*(TEMP(2)/V
P(6,1)=P(6,1)-DGL*SOSIN(TEMP(2))/(V*TF(2,2))
C COMPUTE HEATING AND HEAT RATE DERIVATIVES
RHNO=1.2250100
P(7,1)=ECNED*DSORT(RHORHNO)*1.2620-4*VRMAG*3.15
VRMAG=P(4,1)
P(8,1)=DSORT(RHORHNO)*3.15D0*1.2620-4*1.2620-4*VRMAG*2.15
10VRMAG
P(8,1)=P(8,1)+1.2620-4*VRMAG*3.15D0*RHONP11)/(2.00*DSORT(RHONP11))
RETURN
END
SUBROUTINE DFRV2(TS, XS, P, L, M, NI, KPAR, IPAR)

IMPLICIT REAL*8(A-H,L-Z)

DIMENSION XS(4,1), STVRS(999,6), GRAD(999,3), SERCH(999,3), U(999,3),
2 TEMP(2,5), XP(4,1), X(6,1), P(4,1), C(7)

COMMON/CONST/PI, EPS, XHU, DME, AREA, ECOEF, GNOT

COMMON/CONST3/CSTR, R, C, DTFM, XDTFM

COMMON/CTRL/GRAV, SERCH, U, ASTR, STF, KJIS, IJKU, ISTAR

COMMON/STATF/ALT, XMASS, TEMP, STVRS

COMMON/STATF/ALT#1, XMACH, FLTANG, VF, GAMMF, TF

T=TF-15

C RETRIVE STATE VARIABLES FROM STORAGE
IF(MI .EQ. 1) GO TO 10

10 X(1,1)=STVRS(1,1)
GO TO 40

C INTERPOLATE FROM STATE VAR.
10 TF (K, FN, 1) GO TO 40

10 X(1,1)=STVRS(M-1,1)*STVRS(M-1,1)/2.*DO+STVRS(M-1,1)
GO TO 40

C FINISH CONTROL

20 IF(1 .GE. IJKU) GO TO 60

IF(U(1,1,3) .LT. T) GO TO 50

IF(U(1,1,3) .GT. T) GO TO 55

 TEMP(1)=U(1,1,3)*((U(1,1,3)-U(1,1,2))/(U(1,1,3)-U(1,1,2)))*T-U(1,1,3)

 L=1+1
GO TO 400

50 T=1+1
GO TO 20

55 L=1-1
GO TO 20

60 IF(U(1,1,3) .GT. T) GO TO 55

 TEMP(1)=U(1,1,3)*((U(1,1,3)-U(1,1,2))/(U(1,1,3)-U(1,1,2)))*T-U(1,1,3)

 TEMP(2)=U(1,1,3)*((U(1,1,3)-U(1,1,2))/(U(1,1,3)-U(1,1,2)))*T-U(1,1,3)
GO TO 400

C COMPUTE TRIG QUANTITIES

COS=DCOS(TMP(2))
SIN=DSIN(TMP(2))

COS=DCOS(X(3,1))
SIN=DSIN(X(3,1))

COS=DCOS(X(5,1))
SIN=DSIN(X(5,1))

COS=DCOS(X(6,1))
SIN=DSIN(X(6,1))

K=X(1,1)

V=K**2**R

V=P*K**2**R

V=P**2**R

C COMPUTE ATMOSPHERIC PARAMETERS
ALT=K*(X(1,1)-T)

CALL ATMOS(ALT, TEMP, PRES, RHO, VS, NVS, DVS, DRHO, DPRES)

XHACH=V/VS

NH=V/RH

C COMPUTE AERO. COEF.

CALL SPLINE(TMP(1), XHACH, CL, CD, DCLM, DCLA, DCDM, DCLA)

C FIND AERON. PARTIALS

DINR=AREAR+V*CL*DRH/(T*DO*XMASS)

DINV=AREAR+V*CL*RH/XMASS+DO*DCLM/YS

C CONTINUE