LOW THRUST OPTIMAL ORBITAL TRANSFERS
NAS8-38609 D.0. 64

Dr. Shannon S. Cobb, Principal Investigator
Department of Mathematical Sciences
University of Alabama in Huntsville
Huntsville, Alabama 35899

Final Report
June 1994

Submitted by the University of Alabama in Huntsville,
Huntsville, Alabama

(NASA-CR-193984) LOW THRUST
OPTIMAL ORBITAL TRANSFERS Final
(Alabama Univ.) 59 p

N95-10866
Unclas
63/13 0019679
For many optimal transfer problems it is reasonable to expect that the minimum time solution is also the minimum fuel solution. However, if one allows the propulsion system to be turned off and back on, it is clear that these two solutions may differ. In general, high thrust transfers resemble the well known impulsive transfers where the burn arcs are of very short duration. The low and medium thrust transfers differ in that their thrust acceleration levels yield longer burn arcs and thus will require more revolutions. In this research, we considered two approaches for solving this problem; a powered flight guidance algorithm previously developed for higher thrust transfers was modified and an “averaging technique” was investigated.
INTRODUCTION

The problem considered in this research is the optimal transfer of a space vehicle from a circular low Earth orbit (LEO) to a circular geosynchronous (GEO) orbit using a low thrust propulsion system. Because of the advantages of using a low thrust propulsion system for lunar operations, low thrust transfer was researched extensively in the 1960's. Most of the early work done in this area was analytical or crudely approximate; the results obtained for the LEO-to-GEO transfer were for sub-optimal trajectories because of the simplifying assumptions and approximations made to reduce the computations. The flexibility of the algorithms were thus limited and more preflight analysis was required. Thus, it remained to find an efficient numerical scheme to compute an optimal low thrust trajectory using a low thrust propulsion system.

One of the most recent papers which addresses this problem computationally is that of Redding and Breakwell [1]. They compute the gravity losses for a fixed acceleration maneuver as a way of measuring the “transfer efficiency”; the variance in the applied delta v is compared with the impulsive solution. Several authors have studied low thrust trajectories using an averaging approach. Jasper [2] and Sackett et. al. [3] used averaging techniques for the low thrust trajectory minimum-time problem. Sackett et. al. [3] attempted to generalize their results to include the minimum fuel problem.

As with previous research, we expect the results to be beneficial to space exploration. The benefits of low thrust transfers to NASA include: (i) an efficient guidance scheme for orbital maneuvering vehicles, (ii) savings of fuel for the proposed long duration missions, (iii) enable the operation of large fragile space structures, by keeping small the forces they experience, and (iv) the reduction in size of the transfer propulsion system.

OBJECTIVE

The objective of this research was to address the problem of orbital transfers for low thrust propulsion systems. The approach taken was to modify an existing algorithm which was known to work for “higher” thrust
propulsion systems and apply it to the low thrust problem; at some lower thrust level, we expected this algorithm would not be effective, and then an averaging technique would be applied. In addition to the fact that the "averaging" approach (as outlined in Appendix 1) was not converging and the previous known algorithm was proving effective for lower thrust, the method of approach was modified. The original powered flight guidance algorithm, OPGUID, was developed in the 1960's and later extended to the multiburn algorithm, SWITCH. (All extensions of the original algorithm are referred to as OPGUID in this report). It is expected that OPGUID can be used as an on-board guidance scheme for future missions (6). The algorithm was known to be effective for intermediate to high thrust maneuvers, and with modifications, it has proven effective for low thrust transfers. The brief formulation described below is taken from that of the authors of OPGUID, Johnson and Brown [4].

Formulation of OPGUID

The equations of motion for a space vehicle are given by

\[ \dot{r} = v \]
\[ \dot{v} = \frac{\mu r}{r^3} - \frac{c\dot{m}}{m} u \]  

(1)

where \( r \) is the position vector, \( v \) is the velocity vector, the unit vector \( u \) is the control vector and \( \dot{m} \) satisfying \( \dot{m}_{\text{max}} \leq \dot{m} \leq 0 \), is the vehicle mass rate of change (which represents the magnitude of thrust, and thus is part of control).

A performance index appropriate for minimizing fuel usage is

\[ J = \int_{t_0}^{t_f} -\dot{m} \, dt. \]

(2)

Letting \( x = (r, v, m) \) be the state vector and \( p = (q, s, w) \) the costate vector, the Hamiltonian is

\[ H = L + p^T \dot{x} = -\dot{m} + q^T v + s^T \left( \frac{\mu r}{r^3} - \frac{c\dot{m}}{m} u \right) + w\dot{m}. \]
According to Pontryagin's Minimum Principle, the optimal thrust direction (the control vector) is

\[ \frac{u}{u} = -\frac{s}{s}, \]

which minimizes the Hamiltonian \( H \). That is,

\[ \min_{\text{u}} H = \left( -1 + w + \frac{s_c}{m} \right) \dot{m} + q^T v - \frac{\mu}{r^3} (s^T r). \]

Letting \( S \) denote the "switching function", which we define as \( (1 - w - \frac{s_c}{m}) \), we see that the Hamiltonian is minimized with respect to the thrust direction if \( \dot{m} = \alpha \) for \( S \leq 0 \), and \( \dot{m} = 0 \) for \( S > 0 \). It follows by definition that the costate equations are given by

\[ \dot{q}^T = \frac{-3 \mu r^T s}{r^5} r^T + \frac{\mu b^T}{r^3}, \]
\[ \dot{s}^T = -q^T, \]
\[ \dot{w} = -\frac{c m}{m^2}. \]

For high thrust multiburn optimization, the following assumptions are made:
(i) Apart from thrust acceleration, motion is Keplerian.
(ii) Thrust is proportional to mass rate; hence mass loss is zero when thrust acceleration is zero.
(iii) No terminal constraint is time-dependent.
(iv) The number of separate burns are limited to \( k \) in order to obtain a realistic optimal solution; otherwise with no penalty, one could insert as many separate burns or coasts as desired.

Thus the boundary value problem requires that trajectory which achieves the desired orbit with minimum fuel expenditure subject to a limit on the total number of separate burn arcs. The dynamical necessary conditions for this boundary value problem are given by (1), (4) and (6). Meanwhile, the boundary conditions for this problem are given at the left end by the initial position, velocity, and mass of the vehicle and at the right end by six
conditions defining the desired characteristics of the destination orbit. One of the intermediate point constraints is that the switching function $S$ be zero at each interior switching time. We require that $S > 0$ on coast arcs and $S < 0$ on burn arcs. As an illustration, we will consider a circular-to-circular coplanar transfer. Therefore, the initial arc must be a burn arc, i.e. the switching function $S$ is negative and the propulsion system is on. We cannot start a circular transfer with a coast because any initial coast time is equivalent to another and thus no optimization of fuel occurs.

Let $t_0$ be the initial time for which initial position, velocity and mass are given, $r(t_0) = r_0$, $v(t_0) = v_0$, $m(t_0) = m_0$. Also, let $n$ be the number of separate burn arcs. For the initial burn arc, the switching time for cutoff must be optimized. Clearly, the switching function $S$ must be zero at all switching times. Along the optimal trajectory, the optimality condition on $H^*$,

$$H^* = S\alpha U(-S) + q^T v - \frac{\mu S^T f}{r^3}$$  \hspace{1cm} (7)

implies that $H$ is identically a constant (does not depend explicitly on $t$). Suppose that at each switching time $t_i$, we denote a transversality variable

$$Tv \equiv q^T v - \frac{\mu S^T v}{r^3}.$$ \hspace{1cm} (8)

Since $H$ is constant and the switching function $S$ is zero at each switching time $t_i$, $i = 1, ..., n-1$, we obtain the following $2n-2$ conditions

$$Tv(t_{2j}) = Tv(t_{2j+1}), j = 1,...,n - 1$$
$$s(t_{2j}) = s(t_{2j-1}), j = 1...n - 1.$$ \hspace{1cm} (9)

The advantage of the last equation is that it allows us to remove the costate variable $w$ from the computations. Clearly, we could not merely let $H(t_i) = H(t_{i+1})$ at each switching time since along coast arcs, we gain no information as far as optimality is concerned.

Thus the boundary value problem is to find the values for the six components of initial costate and the $2n-1$ switching times $t_1...t_{2n-1}$ such that the results of integrating (1) and (6) forward in time satisfies the six
right end mission conditions at the final cutoff time, the 2n-2 intermediate necessary conditions in (9) and the condition $|u_0| = 1$.

At the final time $t_f$, $k \leq 6$ terminal state constraints are imposed. If fewer than 6 conditions define the destination orbit, then supplementary transversality conditions requiring that the unconstrained orbit parameters be chosen optimally are included to make a total of 6 independent conditions. The most fully defined orbit transfer mission is the 5 orbital constant mission. It is possible, however, to constrain less than 5 orbital constants. Various orbital missions are defined and given by different modes; e.g. mode = 2 constrains the semi-major axis and eccentricity (a and e) of the target orbit. Whereas, mode=5 constrains a, e, and inclination (i), argument of perigee ($\omega$) and right ascension ($\Omega$). For the case of mode 5, if the k constraint functions of final state are given by $g_i(x_f) = 0, i = 1...5$, the conditions of optimality requires that the final costate $p_f$ lie in the space spanned by the gradients of the constrained functions. Thus is $a_i$ span the space orthogonal to the space spanned by the gradients $\frac{\partial g_i}{\partial x}$ of $g_i$, then $p_f$ must be orthogonal to the single vector $a_6$. We can see that this transversality vector must be orthogonal to the gradient of every final constraint, and thus we can take

$$a_6 = \left( v, -\frac{\mu r}{r^2} \right)^T.$$

MODIFICATION FOR LOW THRUST ACCELERATION

The long length of the burn arcs for lower thrust acceleration can introduce great sensitivity in an orbital transfer, especially in the initial guesses for costate variables. Because of this sensitivity in the many costate variables, as well as the initial guesses of the coast and burn times, the multiburn case for low thrust acceleration has proven difficult to converge. The first attempts in applying this algorithm for transfers shorter than LEO-to-GEO, using multiple burn-coast arcs did not converge; and in several cases, the converged solution was exactly the same as the single burn arc solution depending upon the initial array of times.
Example.

The savings in fuel and burn time can be seen in the following example, where we have convergence. We compute the circular orbital transfer from $a = 6656$ km to $6756$ km using MODE 2 of OPGUID. Thrust = 2.646 kn, initial mass = 270000 kg, mass rate = .60, and $isp = 450$ were the vehicle capability parameters used.

For the single burn case, an initial cutoff time of 20,000 seconds was used to obtain the following results within 26 iterations using the velocity vector direction in the initial guess of costate: semimajor axis $a = 6755.9$, eccentricity = .00050, total burn time of 18,948.8 seconds = 5.26 hours, final mass = 258630.70 kg.

For the two burn case, the initial times array representing a 12,000s burn-5000s coast-5000s burn, with the same initial guess of costate above, converged within 85 iterations to the orbit with semimajor axis $a = 6755.6$, eccentricity = .0010, total burn time of 17,089.9 seconds = 4.75 hours, final mass = 259746.07 kg. The final times array is: 0, 0, 0, 13771.50, 17462.90, 20781.28.

Since we expect long burn arcs, it is not reasonable to expect that a single burn LEO-to-GEO transfer is the “best” approach to fuel efficiency, so we take advantage of the success we found in applying OPGUID to “short” burns in the single burn case. We used the following approach:

A circular transfer LEO —GEO transfer is obtained by:

(i) optimizing a burn of specified length at perigee
(ii) coast around the orbit and center the next burn about perigee
(iii) raise apogee to desired semimajor axis by successive burns centered at perigee
(iv) coast to apogee and optimize a burn of specified length at apogee
(v) raise perigee by successive burns centered about apogee in order to circularize orbit.
The results obtained for \( \text{MODE} = 2 \), where the semimajor axis and eccentricity are constrained is given in the table and graphs below. As expected, there is a trade off between time of transfer and savings in fuel.
### LEO-TO-GEO CIRCULAR TRANSFERS USING VARIOUS BURN LENGTHS

<table>
<thead>
<tr>
<th>Burn length (minutes)</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of perigee burns</td>
<td>152</td>
<td>101</td>
<td>75</td>
<td>64</td>
</tr>
<tr>
<td>Number of apogee burns</td>
<td>59</td>
<td>38</td>
<td>27</td>
<td>20</td>
</tr>
<tr>
<td>Burntime (hours)</td>
<td>77.47</td>
<td>79.35</td>
<td>82.90</td>
<td>92.07</td>
</tr>
<tr>
<td>Totaltime (hours)</td>
<td>1344.04</td>
<td>871.33</td>
<td>620.65</td>
<td>446.51</td>
</tr>
<tr>
<td>Final Mass (kg)</td>
<td>102,655</td>
<td>98,163</td>
<td>90,946</td>
<td>71,641</td>
</tr>
</tbody>
</table>
Burnlength versus Total Burntime
Burnlength versus Final Mass times .001
CONCLUSIONS

It is apparent that the problem of finding the initial costate is the most difficult aspect of computing optimal low thrust orbital transfers, whether one uses OPGUID or MINFUEL. In both cases, for circular-to-circular transfers, taking the initial thrust direction (which is aligned with the initial costate vector) in the direction of the velocity vector has proven the most effective method of finding a solution. One of the disadvantages of using OPGUID is that one has to guess the final cutoff time (and intermediate switch times for multiburn cases). In addition to using the impulsive transfer to guess the final cutoff time, we also looked at other algorithms, such as SCOOT (Simplex Computation Of Orbital Transfers) to guess the length of burn arcs.

We were able to use OPGUID effectively to compute the low thrust transfer from LEO-TO-GEO using successive single burn transfers; however, we were unable to obtain convergence for a similar multiburn approach. One would expect that there would be more savings in fuel if we added a coast arc to the transfer; this is supported in the above example. For the circular transfer case, a possible multiburn approach would be to circularize at various radii as we approach geosynchronous orbit.
CALLING PROGRAM CALLTOGUIDE

Purpose: This is the calling program to the modules which computes the minimum fuel transfer. CALLTOGUIDE actually makes a call to the subroutine G715P_G which then makes the call to GUIDE which computes the trajectory across burn and coast arcs. CALLTOGUIDE reads in the necessary data for the transfer, such as vehicle capabilities and initial and target orbits. Then G715P_G is called to make successive burns centered around a "fixed" perigee and then to coast around in order to center the next burn about perigee; when the desired apogee is reached, a coast is made to center the next burn around this apogee and successive burns and coasts are performed until the desired perigee has been reached. In the cases of circular target orbits, we circularize at this apogee. Also, note that the initial data is given in terms of orbital elements and is then changed to a state vector using the subroutine KEPSTATE.

INPUT VARIABLES:

THLIQ—thrust level in kilonewtons

NCOAST—the number of coast arcs per leg of transfer

NBURN—the number of burn arcs per leg of transfer

TMODE—the mode of transfer; the desired constraints on the target orbit determines the mode; e.g. TMODE=2 constrains the semimajor axis and the eccentricity of the final orbit; same as GUID_OPTION

ITARG—the value of 2 tells G715P_G to compute a Cartesian target vector

DINCL, DNODE, THT, ARGPER—target values of inclination,
ascending node, flight path angle and argument of perigee

IFLAG—the value of 0 implies no initial coast arc

VEH(I,J)—Ith leg, Jth vehicle characteristic in that leg
  J=1 — initial mass of leg, kg
  J=2 — fuel flow rate, kg/s
  J=3 — total burn time of leg, s (set to a very large number)
  J=4 — thrust magnitude, kilonewtons
  J=5 — 0
  J=6 — 0
  J=7 — initial Runge-Kutta integration step size for ith leg, s

TIMES(1:6) — array of engine on/off times, s

ICIRC — the value of 1 indicates a noncircular target orbit; while 0 indicates a circular target orbit

DELTABURN — the length of burn arcs to be optimized, s

RPER, RAPO — the radius of perigee and apogee of initial orbit, km

CEXV — the calculated exhaust velocity

SMATARG — the target semimajor axis, km

Note that there are some variables in the input list which are either constant or are set to certain values for convenience in the computation of an orbital transfer.
MAIN VARIABLES:

DELTAM — the change in mass for a burn of length deltaburn

DELTAV — the velocity increment caused by a burn of length deltaburn

R0, V0 — the current radius and velocity for each intermediate transfer

RT, VT — the target radius and velocity for each intermediate transfer

PERRAD, APORAD — the current radius of perigee and apogee

TIMEPER — the time elapsed since perigee passing of vehicle

APOTIME — the necessary time to reach apogee of current orbit from current position

COASTTIME — the calculated length of coast arcs needed to center burn about perigee and apogee

BURNTIME — the length of actual burn arc after optimizing the burn length deltaburn

TOTALTIME — total time for transfer, which is the sum of each intermediate transfer

OP_GUID_DATA_INIT

Purpose: This routine initializes data needed in the integration process and the earth data.

G715P_G

Purpose: This subroutine has been substantially changed from the original module of Dukeman which was used as a calling program with many mission
components, including ascent and descent, engine throttling and so forth. Although the comments have been left unchanged, the purpose is to set up the target orbit in Cartesian coordinates, set up the initial costate and compute yaw and pitch angles, if desired, and of course to call the subroutine GUIDE to compute the trajectory over burn and coast arcs.
REFERENCES


APPENDIX 1

AVERAGING APPROACH

The formulation of the problem follows that of Horsewood et. al. [7], where the state of the spacecraft is given in terms of the slowly varying equinoctial elements, which are expressed in terms of the classical elements $a, e, i, \omega, \Omega$ as

$$z = (a, e \sin (\omega + \Omega), e \cos (\omega + \Omega), \tan (i/2) \sin \Omega, \tan (i/2) \cos \Omega)^T.$$ 

The spacecraft mass, $m$, is also a state variable. In addition, the position of the spacecraft within an orbit is given by the eccentric longitude, $F = E + \omega + \Omega$.

The equations of motion are

$$\dot{z} = \frac{2P}{mc} M \dot{u}, \quad \dot{m} = -\frac{2P}{c^2}$$  \hspace{1cm} (1)

where $M = \frac{\partial z}{\partial F}$ is a $5 \times 3$ matrix calculated by treating $F$ as an independent parameter, such that the variation of $F$ with respect to the other state variables is zero; $P$ is the power due to the thrusters, $c$ is the exhaust velocity and $\dot{u}$ is the unit vector in the direction of thrust.

It follows from the well-known maximum principle that the fuel optimal trajectory from a given state to some desired final state is found by thrusting at every point along the trajectory in the direction which maximizes the Hamiltonian function $H$. $H$ can be written in terms of the state variables and their corresponding costate (adjoint) variables $\lambda_z$ and $\lambda_m$ as

$$H = \lambda_z^T \dot{z} + \lambda_m \dot{m}$$  

$$H = \frac{2P}{mc} \left( \lambda_z^T M \dot{u} - \frac{m \lambda_m}{c} \right)$$  \hspace{1cm} (2)

where the costate variables satisfy a first order linear system of ordinary differential equations. Now, to maximize $H$ we need only thrust in the direction given by $\dot{u} = \frac{M^T \lambda_m}{|M^T \lambda_m|}$. Note that if the quantity in parenthesis in [2], call it $\sigma$, is negative, then $H$ is negative and hence $H$ is maximized by letting $P = 0$, which amounts to turning the propulsion system off, i.e. “coasting”; on the other hand, if $\sigma$ is positive, then $H$ is maximized by letting $P$ take on its maximum value, i.e. “thrusting”.

(i)
Because of the many orbit revolutions of a long duration transfer, the intensive computations can be reduced by an averaging technique. We can compute an "averaged" Hamiltonian function by holding the state and costate variables constant over an orbital period of duration \( \tau \), i.e. we assume Keplerian motion, and integrating the actual Hamiltonian function as follows:

\[
\dot{H} = \frac{1}{\tau} \int_0^\tau H(\ddot{z}, \lambda_z, \ddot{m}, \lambda_m, F) \, dt \\
= \frac{2\pi}{\tau} \int_0^{2\pi} H(\ddot{z}, \lambda_z, \ddot{m}, \lambda_m, F) s(\ddot{z}, F) \, dF
\]

where \( s(\ddot{z}, F) = \frac{1}{2\pi} \). The "averaged" equations of motion can now be computed using this "averaged" Hamiltonian, and since the Hamiltonian and its derivatives are zero during coast phases, we need only integrate these equations over the predetermined thrust intervals.

**NUMERICAL APPROACH**

A FORTRAN program MINFUEL has been written using the averaging approach as outlined above for the minimum propellant low thrust circular transfer. The initial orbit: semimajor axis \( a = 6656 \) km, eccentricity \( e = 0 \), inclination \( i = 10 \) deg, ascending node \( \Omega = 0 \) deg, argument of perigee \( \omega = 0 \) deg, and eccentric longitude \( F = 0 \) deg. The initial mass \( m = 270000 \) kg, exhaust velocity \( c_{e,v} = 4.41 \) km/s, mass rate \( m_{rate} = -0.6 \) kg/s and thrust \( T = 5.843443 \) kn. The final state desired is \( a = 6756 \) km, \( e = 0 \), and the final transversality condition desired is costate mass = 1. The algorithm of MINFUEL is given below.

(1) In order to obtain the desired final conditions, we must iterate on the unknown initial costate. An initial costate guess can be found by using the fact that it is fuel-optimal to thrust in the direction of motion, or tangential direction. This known direction can be used to help guess the initial costate values since \( \ddot{u} = \frac{M^T \lambda}{|M^T \lambda|} \). Given any state, this optimal thrust direction can be calculated in terms of the costate values by running the program XFORM to compute the transpose of the matrix \( M \) and the velocity vector. These values were scaled upon input to MINFUEL. Since we know that it is optimal to start a circular transfer with a thrust arc, the initial value of costate mass was found using

\[(ii)\]
the scaled values of costate to ensure that the switch function was a "small" positive number. Module used is SWITCH2.

(2) The iterator is called to begin finding the desired initial costate; The SECANT algorithm is the one currently encoded, with a call to the singular value decomposition subroutines, SVDCMP and SVBKSB, to compute the NEWTON corrections for the calculated Jacobian matrix.

(3) A trajectory corresponding to each of the initial costate guesses is found by calling the subroutine FUNCT, which in turn calls the Runge-Kutta routine to integrate the averaged equations of motion over a time step. While performing the integration, a quadrature routine is called to average the state, costate and Hamiltonian function, and hence the averaged equations of motion; it is essential that first the switch points, the zeroes of the switch function, around the orbit be determined a priori in order that this integration is valid. The switch function can be expressed in terms of the fast variable $F$ around an orbit and the zeroes of the switch function $\sigma = \left[ \lambda_x^T M M^T \lambda_x \right]^{\frac{1}{2}} - \frac{m \Delta m}{c}$ are found, which defines the thrusting subintervals of $[0, 2\pi]$ for integration. Again for the circle-to-circle transfer we know that initially the switch function should be positive.

(4) The final conditions are evaluated to determine if they have been satisfied. The Newton corrections are then computed using the Jacobian matrix. We continue until final conditions are within desired tolerance.

Test Results

The result of running MINFUEL is that it does not converge to the desired final state and costate mass. Initially, it appears to converge, then it diverges. Whether, this is the initial costate guess, or some element of the code has not been resolved.
CALLING SEQUENCE AND VARIABLES

SUBROUTINE INPUT—Initializes orbits and transfer data

MAIN VARIABLES:
ZCUR — Current state variables of equinoctial elements a, h, k, p, q, and mass m
COCUR — Current costate variables, input variables as Lam(6)

SUBROUTINE ITERN—Calls the numerical iterator for the initial costate
(currently SECANT)

SUBROUTINE SECANT—Written by Greg Dukeman of MSFC

SUBROUTINE FUNCT—Calculates the optimal trajectory corresponding
to the given initial costate guess; calls the Runge-Kutta routine and evaluates
the final conditions

SUBROUTINE SWCHPTS—Compute a priori the zeroes of the switching
function which are needed in the averaging process in
QUAD to integrate about the orbit

MAIN VARIABLES:
NROOT — number of roots found
ROOTS — actual zeroes found
DERZ12 — averaged derivatives

SUBROUTINE INTEG—Integrate the averaged equations of motion and
Hamiltonian

SIGMAF — The switching function, which is the magnitude of the primer
vector
PROGRAM CALLTOGUIDE

C******************************************************************************
C CALLTOGUIDE is the calling program to the modules of OPGUIDE to compute the
C optimal trajectory of transfer by raising apogee using successive perigee
C burns and then raise perigee in order to attain the desired semi-major axis.
C******************************************************************************

C Many of the main variables are defined in module G715P_G of OPGUIDE
C TIMEPER--Time elapsed since perigee passing
C APOTIME--Time needed to reach apogee of an orbit
C COASTTIME--Length of coast arc needed to center burn arcs at perigee and
C        --apogee points
C DELTABURN--Length of burns to be optimized in transfers
C TOTALTIME--Total time of burn and coast arc lengths
C******************************************************************************

IMPLICIT REAL*8 (A-H, O-Z)
REAL*8 ISP, NMEAN, MANOM
INTEGER TGT_SET, GUID_OPTION
PARAMETER (MXXBURN'600_XMU'3.9860064D+I4,PI'3.14159654)

COMMON /LINEAR/ LINEAR SOLUTION METHOD
COMMON /OPGUID_DATA/ ISP, THLIQ, NCOAST, NBURN,
        ALPHA_MIN, ITARG, MITR, G0, CEXV, AZ, PHIL, C30A, DECOA, RAOA,
        GUID_OPTION
COMMON /G715P/ ISP, THLIQ, NCOAST, NBURN, TMODE, GUID_OPTION,
        ITARG, MITR, SOFT_CONSTRAINTS, LINEAR SOLUTION METHOD, OBLATE,
        NORM VARS, DINCL, DNODE, THT, ARGPER, TIME, IFLAG,
        VEH, TIMES, ICIRC, TGT_SET, DELTABURN, RPER, RAPO,
        GO, CEXV, AZ, PHIL, C30A, DECOA, RAOA, ALPHA_MIN, Q0_SUB, Q1_SUB, SMATARG

CALL OP_GUID_DATA_INIT

READ(100, G715P_C)

UK = XMU/1.D+9

C INICIALIZE ORBIT DATA
TOTALTIME = 0.
NUM_APO_MAX = 200
NUM_PER_MAX = 400
NUM_PER_BURN = 0
NUM_APO_BURN = 0
LAST_BURN = .FALSE.
PERBURN = .TRUE.
SMA0=(RPER+RAPO)/2.0
ECC0=1.0 - (RPER/SMA0)
TRUAN = 0.
CALL KEPSTATE(X0,X0(4),R0,V0,SMA0,ECC0,DINCL, ARGPER, DNODE, TRUAN)

C THE ANGLES ARE RETURNED IN RADIANS FROM KEPSTATE
RADDEG = 180./PI
DEGRAD = PI/180.

NECESSARY DATA FOR G715P_G TO COMPUTE TARGET ORBIT
XMASS = XMASSINIT
DELTAM = VEH(1,2)*DELTABURN
DELTAV = CEXV*(DLOG(XMASS/(XMASS-DELTAM)))
RT = RPER
RAPOTARG = SMATARG
VT = V0 + DELTAV
CALL G715P_G

NUM_PER_BURN = NUM_PER_BURN + 1

DO I = 1, 6
   XCUR(I) = XCUR(I)*1000.
END DO
CALL CSVTOORBELE(XCUR, XCUR(4), AC, EC, DINC, OGAC, APEC, FC)
DO I = 1, 6
   XCUR(I) = XCUR(I)/1000.
   X0(I) = XCUR(I)
END DO
WRITE(30,*) ' ' THIS IS THE END OF BURN ',NUM_PER_BURN,' AT PERIGEE.'
WRITE(30,*) ' CURRENT ELEMENTS: SMA',AC*.001,' ECC',EC,' INC ',DINC,
& ' RAN',OGAC',' ARGOFF',' APEC',' TAN',' FC',' RAPO ',AC*.001*(1.+EC),
& ' RPER ',AC*.001*(1.-EC)

AC = AC/1000.
FC = FC*DEGRAD
PERRAD = AC*(1.0 - EC)
APORAD = AC*(1.0 + EC)
TOTALTIME = TOTALTIME + TIMES(6)
BURNTIME = TIMES(6) - TIMES(5)
TOTAL_BURNTIME = TIMES(6)
WRITE(30,*) ' NUMBER OF ITERATIONS: ',ITER
WRITE(30,*) ' TOTALTIME AND BURNTIME ARE: ',TOTAL_BURNTIME

FOR NONCIRCULAR TARGETS, THEN TARGET PERIGEE AND APOGEE POINTS ARE GIVEN
BELOW FOR A TARGET ECCENTRICITY
DO I = 1, MAXBURN
   DELTAF = PI - FC
   T0 = 0.
   TIME = 0.
   XMASSINIT = XMASS
   TANE2 = SQRT((1.0-EC)/(1.0+EC))*TAN(FC/2.0)
   EANOM = ATAN(TANE2)*2.0
   MANOM = EANOM - EC*SIN(EANOM)
   NMEAN = SQRT(UK)/AC**(1.5)
   TIMEPER = (MANOM)/NMEAN
   APOTIME = PI/NMEAN - TIMEPER
   PERIODT = PI*SQRT((AC**3)/UK)
   DELTAM = VEH(1,2)*DELTABURN
   DELTAV = CEXV*(DLOG(XMASS/(XMASS-DELTAM)))

AFTER FIRST BURN AT PERIGEE, BEGIN ITERATION PROCESS OF BURNS AROUND
PERIGEE AND COASTING TO APOGEE AND PERFORM BURNS ABOUT APOGEE UNTIL
DESIRED SEMI-MAJOR AXIS IS REACHED.

IF (PERBURN) THEN
   NUM_PER_BURN = NUM_PER_BURN + 1
   WRITE(30,*) ',
   WRITE(30,*) 'AT THE END OF COAST-BURN',NUM_PER_BURN,' AT PERIGEE.'
   COASTTIME = APOTIME + (PERIODT - DELTABURN/2.0)
   V0 = SQRT((UK/AC)*(1.0+EC)/(1.0-EC))
   VT = V0 + DELTAV
   IF (APORAD .GE. (RAPOTARG)) PERBURN = .FALSE.
APOGEE_REACHED = APORAD
IF (NUM_PER_BURN .GT. NUM_PER_MAX) PERBURN = .FALSE.
ELSE
   NUM_APO_BURN = NUM_APO_BURN + 1
   WRITE(30,*) , , AT THE END OF COAST-BURN , ,NUM_APO_BURN , , AT APOGEE.
   RT = APOGEE_REACHED
   IF (AC .GE. (SMATARG)) THEN
      CIRCULARIZE AT APOGEE
      ICIRC = 0
      RT = SMATARG
      VT = SQRT(UK/SMATARG)
      LAST_BURN = .TRUE.
   ELSE
      V0 = SQRT((UK/AC)*(1.0-EC)/(1.0+EC))
      VT = V0 + DELTAV
   ENDIF
   COASTTIME = APOTIME - (DELTABURN/2.0)
ENDIF

IF ((NUM_APO_BURN .EQ. 0) .AND. (.NOT. PERBURN)) THEN
   NUM_PER_BURN = NUM_PER_BURN - 1
   GO TO 15
ENDIF

TIMES(5) = COASTTIME
TIMES(6) = COASTTIME + DELTABURN
CALL G715P_G
BURNTIME = TIMES(6) - TIMES(5)
TOTAL_BURNTIME = TOTAL_BURNTIME + BURNTIME
TOTALTIME = TOTALTIME + TIMES(6)
DO J = 1, 6
   XCUR(J) = XCUR(J)*1000.
END DO
CALL CSVOORBELE(XCUR, XCUR(4), AC, EC, DINC, OGAC, APEC, FC)
DO J = 1, 6
   XCUR(J) = XCUR(J)/1000.
   X0(J) = XCUR(J)
END DO
WRITE(30,*) 'CURRENT ELEMENTS: SMA ',AC*.001,' ECC ',EC,' INC ',
& DINC,' RAN ',OGAC,' ARGOFP ',APEC,' TAN ',FC,' RAPO ',
& AC*.001* (1.0+EC) , ' RPER ',AC*.001*(1.-EC)
WRITE(30,*) 'THE NUMBER OF ITERATIONS WERE: ',ITER
WRITE(30,*) 'THE TOTAL TIME TAKEN WAS: ',TIMES(6)
WRITE(30,*) 'BUT THE BURNTIME WAS ONLY: ', BURNTIME
WRITE(30,*) '

AC = AC/1000.
FC = FC*DEGRAD
PERRAD = AC*(1.0 - EC)
APORAD = AC*(1.0 + EC)
IF (LAST_BURN .OR. (NUM_APO_BURN .GE. NUM_APO_MAX)) THEN
   WRITE(30,*) '
   WRITE(30,*) 'A FINAL ORBIT HAS BEEN REACHED WITH SEMIMAJOR AXIS'
   WRITE(30,*) 'AND ECCENTRICITY OF ',AC,EC
   WRITE(30,*) 'THE TOTAL TIME FOR TRANSFER WAS',TOTALTIME
   WRITE(30,*) 'THE TOTAL BURNTIME WAS',TOTAL_BURNTIME
   WRITE(30,*) 'THE FINAL MASS IS ',XMASS
   WRITE(30,*) 'THE NUMBER OF PERIGEE BURNS WERE ',NUM_PER_BURN
   WRITE(30,*) 'THE NUMBER OF APOGEE BURNS WERE ',NUM_APO_BURN
   GO TO 10
ENDIF

CONTINUE
END DO
SUBROUTINE OP_GUID_DATA_INIT
IMPLICIT REAL*8(A-H, O-Z)
Earth constants are IAU, 1964
DATA REARTH/6378.160/, OBLATE/.10827E-2/, $ OMEGA/0.729211585D-4/, ETA/1.0/ ! Kilo units
PARAMETER ( PI = 3.141592654D0 )
COMMON /CLEG/ ATP(7), TL, HMAX, EVT, H0
COMMON /CPHY/ UK, REARTH, RHO0, OMEGA, OBLATE, OBLATE_FACTOR
COMMON /CWT/ WT (6)
WT(1) = 0.95
WT(2) = 0.95
WT(3) = 0.95
WT(4) = 0.95
WT(5) = 0.95
WT(6) = 0.
EVT = 1.D-5
RETURN
END

SUBROUTINE G715P_G
C ********************************************************************
C INPUTS: X013(1:3) - vehicle inertial position vector, km, AP13 coord.
C X013(4:6) - vehicle inertial velocity vector, km/s, AP13 coord.
C AZ - launch azimuth, rad
C PHIL - launch latitude of launch site, rad
C ARGPER - target argument of perigee measured from the
C descending node positive in direction of flight, rad
C DINCL - target inclination, rad
C DNODE - target descending node, rad
C RT - target radius magnitude, km
C VT - target velocity magnitude, km/s
C THT - target flight path angle, rad
C TIMES'(1:6) - array of engine on/off times, referenced to
C beginning of mission, s
C VEH(I,J) - Ith leg, Jth vehicle characteristic in that leg
C J=1 - initial mass of leg, kg
C J=2 - fuel flow rate during non-g-limited legs, kg/s
C J=3 - total burn time of leg, s
C J=4 - when no g-limit exists, exhaust velocity
C * abs(fuel rate), kg km/s^2
C when g-limit exists, thrust acceleration magnitude,
C km/s^2
C J=5 - when not g-limited, = 0
C when g-limited, -(g-limit/exhaust velocity), s^-1
C J=6 - 0, for exoatmospheric legs
C reference area of vehicle, km^2
C J=7 - initial Runge Kutta integration step size for
C ith leg, s
C DELP - the most recent pitch gimbal angle from routine GIMBAL
C DELY - the most recent yaw gimbal angle from routine GIMBAL
C EXIT AREA - total engine exit area, m^2
C TCSAVE(1:20) - coast times after each stage, s
C ITIME - ?
C DTN - ?
C E2 - target orbit eccentricity
C FM - magnitude of vehicle acceleration excluding gravity, m/s^2
GT - negative of magnitude of grav acc at target radius, m/s^2
GX - gravity vector in NLS plumbline coord system, m/s^2
IBURNG - thrust stage indicator
DT GUID - guidance cycle time, s
IOPI - IGM parameter
NBURNG - total # of thrust stages
NCYL - ?
PHIT - ?
TAU(1:20) - (vehicle mass / mass flow rate), s
TCG(1:20) - coast times after each stage ?, s
TFG(1:20) - time to enforce fixed attitude-rate for each stage,
TGG(1:20) - burn time for each stage, s
TI - ?, s
TPHIT - ?
XPMI, XYMI - past values of pitch and yaw, rad
XDLIMP, XDLIMY - maximum allowable values for pitch and yaw rate
GLIMIT - g-limit, g's
MCT - ?
IG2(1:20,1:2) - ?
CHIP - ?
CHIY - ?
TIME - time elapsed since launch, s

OUTPUTS: CHIPCN - commanded pitch angle, radians
CHIYCN - commanded yaw angle, radians
UT0 - unit thrust direction in AP13 coord.
UDT0 - time derivative of unit thrust direction in AP13 coord., s^-1
CHIDP, CHIDY - commanded inertial yaw rate with respect to NLS plumbline coord sy

************************************************************************************
IMPLICIT REAL*8(A-H,O-Z)
REAL*8 ISP,M0,INCT
INTEGER TGT_SET, TGT_SET_OLD
DATA TGT_SET_OLD / 0/
PARAMETER(PI=3.141592654D0,XMU=3.9860064D+14)
LOGICAL SAVED_GUID_PARAMS
COMMON/GIDIN/XT(6),TTG,XO(6),T0,M0,VEH(10,7),Q0(6),TIMES(6),CC(6)
COMMON /CAIN/ W3,CLA0,CA0,ETA,AX3,RHO C1,RHO C2,RHO C3,
RHO C4, PRES C1, PRES C2, PRES C3, VSOULD C1,
VSOUND C2, VSOUND C3, VSOUND C4, CA C1, CA C2,
CA C3, CA C4, CA C5, CNA C1, CNA C2, CNA C3,

COMMON /CINDEX/ NARC,IAARC,JMAX,JM,JMAX1,JLAST,NO,NOP,NRKGOS
COMMON /CLEG/ ATP(7), TL, HMAX, EVT, H0
COMMON /CPHYS/ UK, REARTH, RHO0, OMEGA, OBLATE, OBLATE_FACTOR
LOGICAL SOFT_CONSTRAINTS
COMMON /CMODE/ MODE, IFREZ, ISTOP, SOFT_CONSTRAINTS, MEANTG
COMMON/GIDOUT/DQ0 (6), DTIME(6), E(12,12), DC(12), XG(12),
Z(12,12), DD(6), SM(5), CK
COMMON/GUID_PARAMS/SAVED_GUID_PARAMS,VR 0(3),CSV1 0,CSV2 0
COMMON /TABLE SIZES/ NP_CFBST,NP_CFBST_GUID,NP_CAT,
NP_CAT_GUID,NP_CHTABL,NP_CLABT_GUID,NP_CMAT,
NP_CMAT_GUID,NP_CNTABL, NP_CLABT_GUID,NP_CNNTB_GUID,
NP_CNNTB_GUID,NP_CBT, NP_CBT_GUID,NP_RWNTDBL,NP_PIT,NP_P3T,NP_RAMPFI,
NP_ROLL,NP_SOLEFLOW, NP_SOLEFLOW_GUID,NP_SOLSTRT,
NP_SOLSTRT_GUID,NP_THRTSTCUT,NP_RWNTDBL, NP_RW_GUID,
NP_RWNTDBL, NP_THRTSTCUT, NP_RW_GUID,
NP_XCG, NP_XCG_GUID,NP_XMASSO_GUID,NP_YIT,
NP_Y3T,NP_YCG,NP_YCG_GUID,NP_ZCG,NP_ZCG_GUID

DIMENSION XTF(6), QF(6), A13EQ(3,3), A13EQT(3,3), QOS(6),
X013(6), UT0(3), UDT0(3)
DIMENSION A(80),GX(3),TAU(20),TCG(20),TFG(20),TGG(20),VEX(20)
DOUBLE PRECISION ACC(3), XCG_GUID(36), YCG_GUID(36),
DOUBLE PRECISION PR(15), WIND(3), VRX(3), FENG(10),
$ VREL EST(3), VRMAG
DOUBLE PRECISION VB(3), CPP(3), AL(3,3), CPY(3)

DIMENSION X(3), XD(3), XDDG(3), XF(3), FL(20), FJJ(20), S(20), Q(20),
1 UU(20), XIT(3), XIDT(3), XIDD(3), PHIM(3,3), XI(3),
2 XID(3), XIDD(3), DXIDS(3), DXID(3), G(3,3), FK(3,3)
DIMENSION ABCD(8), VEXSV(20), TFSV(20), IG2(20,2)
DIMENSION A1(3,3), B1(3,3), TCSAVE(20)
DOUBLE PRECISION TIMES OLD(6), Q0 OLD(6), DTIMES OLD(6),
$ DQ0_OLD(6), A_TARG_TO_EQ(3,2)

DATA ALPHA_MIN / 0.8 /
LOGICAL GUID CONVERGED, First Time/.true./
DOUBLE PRECISION LAMBDA(3), LAMBDA_13(3), VR(3), Q0 LVLH(3),
$ Q LVLH TO I(4), NWND, MACH, E_NUM(8,8),
$ Q0 NOM(6), CI(6), DC NOM(8),
$ Z NUM(12,12), XG NOM(12)
DATA C1 / 3*I.D-4, 3*I.D-7 /
COMMON/LINEAR/LINEAR SOLUTION METHOD
COMMON/NORM UNITS/NORM VARS,D_UNIT,T_UNIT
DOUBLE PRECISION VINF(3), DX(3)
COMMON/PLANETARY/VINF
COMMON/MAXKOUNT/ITER
 COMMON/AA FORPRINT/AATARG
LOGICAL NORM VARS
COMMON /OPGUID_DATA/ ISP, THLIQ, NCOAST, NBURN, ALPHA_MIN,
$ ITARG, MITR, GO, CEXV, AZ, PHIL, C3OA, DECOA, RAOA, GUID_OPTION
COMMON/ORBIT_INFO/ DINC, DNODE, THF, ARGPER, TIME, IFLAG,
$ ICIRC, TGT SET, DELTABURN, R0, V0, RT, VT, XMASS, TMODE
COMMON/QOSUB/ Q0_SUB(6), Q1_SUB(6)
COMMON /CURRENT_STATE/ XCUR(6)

NORM_VARS = .FALSE.
SOFT_CONSTRAINTS = .FALSE.
MODE = TMODE
UK = XMU/1.D+9
RADDEG=180./PI
DEGRAD=PI/180.
MEAN TGT=0
MEANTGT = MEAN_TGT
IMODE = MODE
TOLC = .00001

C THESE VALUES ARE FOR MODE = 12
VINFMAG = SQRT(C3OA)
VINF(1) = VINFMAG * COSD(DECOA) * COSD(RAOA)
VINF(2) = VINFMAG * COSD(DECOA) * SIND(RAOA)
VINF(3) = VINFMAG * SIND(DECOA)
THT = THT*DEGRAD
AZ = AZ*DEGRAD
PHIL = PHIL*DEGRAD

IF ( TGT_SET .NE. TGT_SET_OLD ) THEN
C ---------------------------------------------------------------
C TARGETING HAS ISSUED A NEW SET OF TARGETS. RECOMPUTE TARGET FUNCTIONS
C ---------------------------------------------------------------
BETA=0.
BMT=0.
DO I = 1, 6
  CC(I) = 0.
  DD(I) = 0.
END DO

IF ( ITARG .EQ. 2 ) THEN
  C -------------------------------
  C need to construct a cartesian target vector
  C -------------------------------
  IF ( ICIRC .EQ. 1 ) THEN
  C --------------------------------
  C target orbit is non-circular
  C --------------------------------
    P = (RT**2/UK) * VT**2 * COS(THT)**2 ! SEMI-PARAMETER
    TRUAN = ATAN2 (P*TAN(THT), P-RT)
    BETA = ARGPER + TRUAN ! ARGUMENT OF LATITUDE
    BMT = BETA - THT
  END IF
  CNOD = COS (DNODE)
  SNOD = SIN (DNODE)
  CBETA = COS (BETA)
  SBETA = SIN (BETA)
  CINC = COS (DINCL)
  SINC = SIN (DINCL)
  CBMT = COS (BMT)
  SBMT = SIN (BMT)

  IF (mode .ne. 6 .and. mode .ne. 7 .and. mode .ne. 13 .and. mode .ne. 14) then
    XT(1) = RT * CBETA
    XT(2) = RT * CINC * SBETA
    XT(3) = - RT * SINC * SBETA
    XT(4) = - VT * SBMT
    XT(5) = VT * CINC * CBMT
    XT(6) = - VT * SINC * CBMT
  end if
END IF !(ITARG = 2)

Do I = 1, 6
  XTF(I) = XT(I)
End do

IF ( IFLAG .NE. 0 ) THEN
  DT = TIMES(6) - TIME
  CALL COAST716 (XT, E, DT, XTF, DUMY, DUMY, DUMY)
END IF
DO I = 1, 6
  XG(I) = XT(I)
END DO

C -------------------------------
C compute orbital element target functions, i.e., the DD()
C -------------------------------
CALL BOUNDF (DUMMY, DUMMY, 1)

C compute a convergence tolerance for future use in convergence test
C
Converg_limit = 0.
DO I = 1, 6
  CC(I) = DD(I) ! CC = Vector of Target Boundary Conditions
  Converg_limit = Converg_limit + cc(i)**2
END DO
CONVERG_LIMIT = TOLC * TOLC * CONVERG_LIMIT
END IF
IF ( ABS(TIME - T_LAST) .GT. 20. .OR. TGT_SET .NE. TGT_SET_OLD) THEN
  ...
C -------------------------------
EITHER AT THE BEGINNING OF A NEW TRAJECTORY OR NEW TARGETS HAVE JUST BEEN ISSUED. RECOMPUTE INITIAL COSTATE GUESSES

\[ VR(1) = X0(4) + X0(2) * OMEGA \]
\[ VR(2) = X0(5) - X0(1) * OMEGA \]
\[ VR(3) = X0(6) \]
\[ V0 = VMAG(X0(4)) \]
\[ V0 = VMAG(VR) \]
\[ VF = VMAG(XTF(4)) \]
\[ RMAG = VMAG(X0) \]

DO I=1,3
\[ QF(I) = XTF(I+3)/VF \]
\[ Q0(I) = X0(I+3)/V0 \]
IF ( TIME .LT. -10. ) THEN
\[ Q0(I) = X0(I) / RMAG \]
END IF
End do

DT = TIMES(6) - TO
DO I=1,3
\[ Q0(I+3) = .001 * (QF(I) - Q0(I)) / DT \]
End do

GUID CONVERGED = .FALSE.
MAX IT = MITR
END IF

END COSTATE GUESS COMPUTATION

T_NAV = T0

PRINT 111, TT, T0
PRINT 112, ((VEH(I,J), J=1,7), I=1,10)
PRINT 113, X0
PRINT 114, XT
PRINT 115, Q0
PRINT 116, TIMES
PRINT 117, CC

ITERATIVE MODE = 1
DCM = 1.0 + 30
IF ( (TIMES(6) - T0) .GT. 2. ) THEN
IF ( ITERATIVE_MODE .EQ. 1 ) THEN
DO 300 I=1, MAX_IT
IF ( DCM .LE. Converg_limit ) GO TO 310
COMPUTE JACOBIAN MATRIX
END IF
300 CONTINUE
END IF

IF ( I .EQ. 1 ) DT_PRED = DT_GUID ! compute a guidance solution one guid c

CALL QUAT_LVLH_TO_I(X0, X0(4), Q_LVLH_TO_I)
CALL QUATFORM(Q_LVLH_TO_I, Q0, Q0_LVLH)

PITCH_LVLH = DATAN2(-Q0_LVLH(3), Q0_LVLH(1))*57.3
YAW_LVLH = DATAN2(Q0_LVLH(2), DSQRAT(Q0_LVLH(1)**2 + Q0_LVLH(3)**2))*57.3

WRITE(30,*) '_LVLH PITCH ', PITCH_LVLH, '_LVLH YAW ', YAW_LVLH
WRITE(76,*) sngl(t0), sngl(pitch_lvlh), sngl(yaw_lvlh)

ENSURE THAT THE OBLATE FACTOR IS ZERO IN GUIDE
OBLATE = 0.
NOP = 1
DT_PRED = 0.

CALL GUIDE(DT_PRED)
DCM = 0.
DO IL = 1, 6
   DCM = DCM + DC(IL)**2
END DO
DCM_PAST = DCM
C PRINT 349, iter
C Print 355, (xG(j), j=1,12), cc, dd
C Print 351, (DC(j), j=1,6)
C ------------------
C Convergence Test
C ------------------

If ( DCM .le. Converg_limit ) GO TO 310
Iter = 1
IF ( GUID_CONVERGED ) THEN
   DO J = 1, 6
      TIMES(J) = TIMES(J) + DTIMES(J)/I.
      Q0(J) = Q0(J) + DQ0(J)/I.
   END DO
ELSE
   ALPHA = 1.
   DO K = 1, 6
      Q0_OLD(K) = Q0(K)
      TIMES_OLD(K) = TIMES(K)
      DQ0_OLD(K) = DQ0(K)/I.
      DTIMES_OLD(K) = DTIMES(K)/I.
   END DO
   DO WHILE ( (DCM .GE. DCM_PAST) .AND. (ALPHA .GT. ALPHA_MIN) )
      DO J = 1, 6
         TIMES(J) = TIMES_OLD(J) + ALPHA * DTIMES_OLD(J)
         Q0(J) = Q0_OLD(J) + ALPHA * DQ0_OLD(J)
      END DO
C NON-FLIGHT CODE
    CALL QUAT_LVLH_TO_I(X0, X0(4), Q_LVLH_TO_I)
    CALL QUATFORM(Q_LVLH_TO_I, Q0, Q0_LVLH)
    PITCH_LVLH = DATAN2(-Q0_LVLH(3), Q0_LVLH(1))*57.3
    YAW_LVLH = DATAN2(Q0_LVLH(2), DSQRT(Q0_LVLH(1)**2 + Q0_LVLH(3)**2))*57.3
    WRITE(30,*) ' LVLH PITCH ', PITCH_LVLH, ' LVLH YAW ', YAW_LVLH
   IF ( ALPHA / 2. .LE. ALPHA_MIN ) GO TO 300
   NOP = 0
   CALL GUIDE(0.D0)
   DCM = 0.
   DO IL = 1, 6
      DCM = DCM + DC(IL)**2
   END DO
   ALPHA = ALPHA / 2.D0
   END DO
   DCM_PAST = DCM
END IF
300 CONTINUE
310 Continue
C -------------------
C NON-FLIGHT CODE
C -------------------
ELSE ! compute partials numerically to test OP GUID formulation of
   NOP = 1 ! variational equations
   CALL GUIDE(0.D0)
   NOP = 0
   WRITE(75,*)' FOLLOWING Z MATRIX COMPUTED USING VARIATIONS '
   WRITE(75,101)((SNGL(Z(K,J)),J=1,9),K=1,12)
   WRITE(75,*)' FOLLOWING E MATRIX COMPUTED USING VARIATIONS '

WRITE(75,101)((SNGL(E(K,J)),J=1,9),K=1,9)
FORMAT(9E13.6)
DO I = 1, 9
   DC_NOM(I) = DC(I)
END DO
DO I = 1, 12
   XG_NOM(I) = XG(I)
END DO
DO I = 1, 6
   Q0_NOM(I) = Q0(I)
END DO
TF_NOM = TIMES(6)
BETA_NOM = 0.
DO I = 1, 6
   DO J = 1, 9
      E_NUM(J,I) = (DC(J) - DC_NOM(J)) / (CI(I))
   END DO
   DO J = 1, 12
      Z_NUM(J,I) = (XG(J) - XG_NOM(J)) / (CI(I))
   END DO
END DO
END DO

C ------------------------------------------
C NOW COMPUTE PARTIALS WRT SWITCHING TIMES
C ------------------------------------------

NSWITCHES = 1

IF ( TIMES(3) .GT. TIME ) THEN
   T3NOM = TIMES(3)
   TIMES(3) = T3NOM + 1.
   CALL DCOPY(6,Q0_NOM,1,Q0,1)
   CALL GUIDE(0.D0)
   DO J = 1, 10
      E_NUM(J,6+NSWITCHES) = (DC(J) - DC_NOM(J)) / (TIMES(3) - T3NOM)
   END DO
   TIMES(3) = T3NOM
   NSWITCHES = NSWITCHES + 1
END IF

IF ( TIMES(4) .GT. TIME ) THEN
   T4NOM = TIMES(4)
   TIMES(4) = T4NOM + 1.
   DO J = 1, 6
      Q0(J) = Q0_NOM(J)
   END DO
   CALL GUIDE(0.D0)
   DO J = 1, 9
      E_NUM(J,6+NSWITCHES) = (DC(J) - DC_NOM(J)) / (TIMES(4) - T4NOM)
   END DO
   TIMES(4) = T4NOM
   NSWITCHES = NSWITCHES + 1
END IF

IF ( TIMES(5) .GT. TIME ) THEN
   T5NOM = TIMES(5)
   TIMES(5) = T5NOM + 1.
   DO J = 1, 6
      Q0(J) = Q0_NOM(J)
   END DO

END IF
END DO
CALL GUIDE(0.D0)
DO J = 1, 9
    E_NUM(J, 6+NSWITCHES) = (DC(J) - DC_NOM(J)) / (TIMES(5) - T5NOM)
END DO
DO J = 1, 12
    Z_NUM(J, 6+NSWITCHES) = (XG(J) - XG_NOM(J)) / (TIMES(5) - T5NOM)
END DO
TIMES(5) = T5NOM
NSWITCHES = NSWITCHES + 1
END IF
TIMES(6) = TF_NOM + 1.
DO J = 1, 6
    Q0(J) = Q0_NOM(J)
END DO
CALL GUIDE(0.D0)
DO J = 1, 10
    E_NUM(J, 6+NSWITCHES) = (DC(J) - DC_NOM(J)) / (TIMES(6) - TF_NOM)
END DO
DO J = 1, 12
    Z_NUM(J, 6+NSWITCHES) = (XG(J) - XG_NOM(J)) / (TIMES(6) - TF_NOM)
END DO
TIMES(6) = TF_NOM

WRITE(75,*)' Z MATRIX GENERATED USING FINITE DIFFERENCES:
WRITE(75,101) ((SNGL(Z_NUM(K,J)),J-1,9),K=1,12)
WRITE(75,*)' E MATRIX GENERATED USING FINITE DIFFERENCES:
WRITE(75,101) ((SNGL(E_NUM(K,J)),J-1,9),K=1,9)
END IF
END IF
IF ( DCM .le. Converg_limit ) THEN
    GUID_CONVERGED = .TRUE.
    MAX_IT = 1
END IF

C -----------------------------------------------
C construct the guidance direction using a command predicted one guidance cycle ahead
C -----------------------------------------------

C IF ( SAVED_GUID_PARAMS ) THEN
C -----------------------------------------------
C construct the guidance pointing vector using atmospheric formulation
C -----------------------------------------------
VRU = VR_0(1) * Q0(1) + VR_0(2) * Q0(2) + VR_0(3) * Q0(3)
SY = CSV2_0 * VRU
LAMBDA(1) = CSV1_0 * Q0(1) + SY * VR_0(1)
LAMBDA(2) = CSV1_0 * Q0(2) + SY * VR_0(2)
LAMBDA(3) = CSV1_0 * Q0(3) + SY * VR_0(3)
ELSE
C -----------------------------------------------
C construct the guidance pointing vector using vacuum formulation
C -----------------------------------------------
LAMBDA(1) = Q0(1)
LAMBDA(2) = Q0(2)
LAMBDA(3) = Q0(3)
END IF

C NON-FLIGHT CODE
CALL QUAT_LVLH_TO_I(X0, X0(4), Q_LVLH_TO_I)
CALL QUATFORM(Q_LVLH_TO_I, LAMBDA, Q_LVLH)
PITCH_LVLH = DATAN2(-Q_LVLH(3), Q_LVLH(1))*57.3
YAW_LVLH = DATAN2(Q_LVLH(2), DSQRT(Q_LVLH(1)**2)
WRITE(30,*), ' LVLH PITCH ', PITCH_LVLH, ' LVLH YAW ', YAW_LVLH
WRITE(76,*), SNGL(T0), SNGL(PITCH_LVLH), SNGL(YAW_LVLH)

construct commanded pitch and yaw angles and rates

CALL ATT_COMP(ITIME,TGG,TFG,DT_GUID,TCHIT,TI,CHITP,CHITY,
1 FK,CHIDP,CHIDY,IDIMEN,IBURNG,
2 CHIPCN,CHIYCN,XPM1,XYM1,XDLIMP,XDLIMY,LAMBDA_13,
$ DPITCH, DYAW)
WRITE(76,*), SNGL(T0), SNGL(CHIPCN), SNGL(CHIYCN)

***this section of code inserted to correct the commanded steering
***angle for the moment balance gimbal angle
***so that the commanded thrusting direction will be as desired.
C IF ( IMOMENT .EQ. 1 ) THEN
C***assume that the most recent gimbal value
C***is available (in flight,
C***it might be the averaged value over the last guidance cycle).
C
C MODIFY PITCH AND YAW ANGLES

PITCH = ATAN2(LAMBDA_13(1),LAMBDA_13(3))
YAW = ATAN2(LAMBDA_13(2), DSQRT(LAMBDA_13(1)**2+
$ LAMBDA_13(3)**2))
PITCH = PITCH - DELP
YAW = YAW - DELY

LAMBDA_13(1) = COS(YAW) * SIN(PITCH)
LAMBDA_13(2) = SIN(YAW)
LAMBDA_13(3) = COS(YAW) * COS(PITCH)

CALL ATT_COMP(ITIME,TGG,TFG,DT_GUID,TCHIT,TI,CHITP,CHITY,
1 FK,CHIDP,CHIDY,IDIMEN,IBURNG,
2 CHIPCN,CHIYCN,XPM1,XYM1,XDLIMP,XDLIMY,LAMBDA_13,
$ DPITCH, DYAW)

END IF

*** Here, have desired thrust direction. steering angle has been
*** modified so that thrust will be as desired.
C IF ( IAEROOPT .EQ. 1 ) THEN
C PR(1)=HEIGHT
C IF ( IATMOS GUID .EQ. 1 ) THEN
C CALL PRA63(PR,ERROR)
C END IF
C CALL LINLUM(156,HEIGHT,HT_GUID(1),RW_GUID(1),RWND)
C CALL LINLUM(156,HEIGHT, HT引导(I),THTW_GUID(I),THTWND)

C***need r,theta calculation**********************
C*****************************************************************************

DO I = 1, 3
WIND(I)=NWND*NORTH(I)+EWND*EAST(I)
VRX(I)=VEL(I)-RR(I)-WIND(I)
END DO
VRMAG=VMAG(VRX)
RHO=PR(6)
VSOUND=PR(9)
MACH = VMAG/VSOUND
PRES=PR(2)*10000.
DYNPRS = 0.5*VRMAG*VMAG*RHO

C***note fm includes drag and base force, but asssumes they
C***will be constant in the future.
C THRUST = FM*XMASSO
C J=NENG
C IF ( IEOUSV .EQ. 1 ) J = J - 1
C DO I = 1,NENG
C IF(IEOUTSV.EQ.1.AND.I.EQ.IENG) THEN
C FENG(I)=0.
C ELSE
C FENG(I)=THRUST/J
C END IF
C END DO
C***GET AERODYNAMIC ANGLES FOR MAX QALPHA AND QBETA TESTS:
C DO I=1,3
C VREL_EST(I)=VRX(I)+ACC(I)*DT_GUID
C END DO
C CALL AEROGUIDSUB(MACH,DYNPRS,VREL_EST,CHIPCN,CHIYCN,CHIR,
C & IAFLG,VB,FA,CPP,MAA,AL,ALPHAP,ALPHAY,IBURN,CPY,DIAM)
C QA = ALPHAP*RADDEG*DYNPRS*0.2048/GO
C QB = ALPHAY*RADDEG*DYNPRS*0.2048/GO
C DELTAA LP = 0.
C DELTABETA = 0.
C IF (DABS(QA).GT.QAMAXLIM) THEN
C AMAX = QAMAXLIM/RADDEG/DYNPRS/0.2048*GO
C IF (ALPHAP.LT.0.) AMAX = -AMAX
C DELTAA LP = ALPHAP - AMAX
C END IF
C IF (DABS(QB).GT.QBMAXLIM) THEN
C BMAX = QBMAXLIM/RADDEG/DYNPRS/0.2048*GO
C IF (ALPHAY.LT.0.) BMAX = -BMAX
C DELTABETA = ALPHAY - BMAX
C END IF
C***MODIFY PITCH BY AMOUNT TO GET DESIRED LOADS AND BY RATE LIMITED
C***AMOUNT
C PITCH = ATAN2(LAMBDA_13(1),LAMBDA_13(3))
C YAW = ATAN2(LAMBDA_13(2),DSQRT(LAMBDA_13(1)**2+
C & LAMBDA_13(3)**2))
C PITCH = PITCH - DELTAA LP + DPITCH
C YAW = YAW - DELTABETA + DYAW
C LAMBDA_13(1) = COS(YAW)*SIN(PITCH)
C LAMBDA_13(2) = SIN(YAW)
C LAMBDA_13(3) = COS(YAW)*COS(PITCH)
C CALL ATT__COMP(ITIME,TGG,TFG,DT_GUID,TCHIT,TI,CHITP,CHITY,
C 1 FK,CHIDP,CHIDY,IDIMEN,IBURNG,
C 2 CHIPCN,CHIYCN,XPM1,XYM1,XDLIMP,XDLIMY,LAMBDA_13,
C $ DPITCH, DYAW)
C END IF
C WRITE(76,*),SNGL(T0),SNGL(CHIPCN),SNGL(CHIYCN)
C T0 = T_NAV
C T_LAST = T_NAV
C
C PRINT 349, Iter, 2. * ALPHA, DCM
C PRINT*, ' CURRENT TIME ', T0, ' T CUTOFF ', TIMES(6)
C PRINT 351, (DC(J),J=1,6)
C PRINT 353
C PRINT 350, Q0
C print 354, ut0, udt0
C PRINT 352, TIMES
C PRINT 361
C 96 FORMAT(1H1)
C 111 FORMAT(1H , 3H T-,F10.4,5H , T0=,F10.4)
C 112 FORMAT(1H , ',VEH =',1P7E15.6)
C 113 FORMAT(1H , ',X0(Eq) =',1P6E15.6)
C 114 FORMAT(1H , ',XT(Eq) =',1P6E15.6)
C 115 FORMAT(1H , ',Q0(Eq) =',1P6E15.6)
C 116 FORMAT(1H , ',TIMES =',1P6E15.6)
C 117 FORMAT(1H , ',CC(Eq) =',1P6E15.6)
C 349 FORMAT( ' ITER ',I5, ' ALPHA ', D12.3, ' DCM ', D12.3)
C 350 FORMAT( ' Q0(Eq) =',6E15.6)
C 354 format(' Q0(AP13) =',6e15.6)
FIRST_TIME = .FALSE.
XMASS = SM(5)
DO J = 1, 6
   XCUR(J) = XG(J)
END DO
RETURN
END
THE FOLLOWING DATA FILE IS ONE THAT WAS USED TO COMPUTE THE MINIMUM FUEL TRANSFER FROM LEO TO GEO USING DELTABURN LENGTHS OF 1200 SECONDS.

SG715P_C

ARGPER = 0.,
ISP = 450.,
G0 = .0098, !KILOMETERS
NBURN = 1,
NCOAST = 1,
TIME = 0.,
THLIQ = 2.646, !KILONEWTONS
TO = 0., !INITIAL TIME
XMASSINIT = 270000.,
THT = 0.,
DINCL = 10.,
DNODE = 0.,
AZ = 55.2506,
PHIL = 28.6084,
IFLAG = 0,
GUID_OPTION = 2, !2=OPGUID
TIMES = 0., 0., 0., 0., 0., 1200.,
VEH(1, 1) = 270000.,
VEH(1, 2) = .60,
VEH(1, 3) = 1.D30,
VEH(1, 4) = 2.646,
VEH(1, 5) = 0.,
VEH(1, 6) = 0.,
VEH(1, 7) = 30.,
MITR = 250,
ALPHA_MIN = .8,
SOFT_CONSTRAINTS = .FALSE.,
TMODE = 2,
ITARG = 2,
ICIRC = 1,
TGT_SET = 1,
OBLATE = 0.,
LINEAR_SOLUTION_METHOD = 5,
DELTABURN = 1200.,
CEXV = 4.41,
RPER = 6656.,
RAPO = 6656.,
SMATARG = 42164.,
CTAOA = 160.,
DECOA = 0.,
RAOA = 120.001,

$END
SUBROUTINE KEPSTATE(R, V, RMAG, VMAG, SMA, ECC, INC, APE, OGA, TAN)
C THIS PROGRAM TRANSFORMS THE SIX KEPLERIAN ELEMENTS A, E, I, W, O, AND
F (true anomaly) IN KILOMETERS TO A STATE VECTOR OF POSITION AND
VELOCITY IN KILOMETERS
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION R(3), V(3), INC, MU
PARAMETER(MU = 398601.2D0, PI = 3.141592654D0)

DEGRAD = PI/180.
RADDEG = 180./PI

INC = INC*DEGRAD
APE = APE*DEGRAD
OGA = OGA*DEGRAD
TAN = TAN*DEGRAD

SLA = SMA*(1.0 - ECC**2)
FACTOR = SLA / (1.0 + ECC*DCOS(TAN))
SQMUP = DSQRT(MU/SLA)
COSFE = DCOS(TAN) + ECC

CWCO=DCOS(APE)*DCOS(OGA) - DCOS(INC)*DSIN(OGA)*DSIN(APE)
SWCO=-DSIN(APE)*DCOS(OGA) - DCOS(INC)*DSIN(OGA)*DCOS(APE)
R (1) = (DCOS(TAN)*CWCO+DSIN(TAN)*SWCO)*FACTOR

CWSO=DCOS(APE)*DSIN(OGA)+DCOS(INC)*DSIN(OGA)*DSIN(APE)
SWSO=-DSIN(APE)*DSIN(OGA)+DCOS(INC)*DCOS(OGA)*DCOS(APE)
R (2) = (DCOS(TAN)*CWSO+DSIN(TAN)*SWSO)*FACTOR

R (3) = (DCOS(TAN)*DSIN(INC)*DSIN(APE)+DSIN(TAN)*DSIN(INC)*DCOS(APE))

& *FACTOR

V (1) = SQMUP*(COSFE*SWCO-DSIN(TAN)*CWCO)
V (2) = SQMUP*(COSFE*(-DSIN(APE)*DCOS(OGA)+DCOS(INC)*DSIN(OGA)*DCOS(APE))
& -DSIN(TAN)*CWSO)
V (3) = SQMUP*(COSFE*(DSIN(INC)*DCOS(APE)-DSIN(TAN)*DSIN(INC)*DSIN(APE)))

WRITE(19, 100) 'POSITION', 'VELOCITY'
DO I = 1, 3
   WRITE(19, 101) R(I), V(I)
END DO

RMAG = DSQRT(R(1)**2+R(2)**2+R(3)**2)
VMAG = DSQRT(V(1)**2+V(2)**2+V(3)**2)

WRITE(19,*) 'IN KILOMETERS, RMAG IS ', RMAG
WRITE(19,*) 'IN KILOMETERS, VMAG IS ', VMAG

100 FORMAT (IH, A, 32X, A)
101 FORMAT (IH, F28.16, 8X, F28.16)
END
PROGRAM OPTI
C THIS IS THE CALLING PROGRAM FOR THE MINIMUM PROPELLANT PROGRAM.
CALL INPUT
CALL ITERN
END

SUBROUTINE INPUT
C******************************************************************************
C INPUT CALCULATES THE INITIAL AND FINAL ORBIT IN TERMS OF THE
C EQUINOCTIAL ELEMENTS USING THE INPUT OF CLASSICAL ELEMENTS.
C
C MAIN VARIABLES: -ZCUR(6), Z0(6) IS THE CURRENT AND INITIAL STATE
C -COCUR(6), LAM(6) IS THE CURRENT AND INITIAL COSTATE
C -ZF IS THE DESIRED FINAL STATE
C -POWR, C, MRATE ARE POWER, EXHAUST VELOCITY AND MASS
C RATE; POWER = -MRATE/C**2.
C******************************************************************************
PARAMETER (N = 6)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION INC0, INCF, M0, LAM(N), MRATE
DIMENSION Z0(N), ZCUR(N), ZF(N), CO0(N), COCUR(N), COF(N)
COMMON /STATE/ ZCUR, COCUR
COMMON /CONST/ POWR, C, AMU
COMMON /MASSES/ ZM0, ZMCUR, ZMF
COMMON /ZFINAL/ ZF
COMMON /ECC_LONG/ FECC1, FECC2
NAMELIST /OP_DATA/ A0, E0, INC0, W0, OMEGA0, FECCI, M0, AF, EF, INCF, WF, &
OMEGAF, FECC2, LAM, AMU, MRATE, C, POWR
C THESE ARE VALUES USED FOR COMPILATION OF THE PROGRAM.
READ(98, OP_DATA)
C
CALCULATE THE INITIAL AND FINAL STATES
Z0(1) = A0
Z0(2) = E0 * DSIN(W0 + OMEGA0)
Z0(3) = E0 * DCOS(W0 + OMEGA0)
Z0(4) = DTAN(INC0 / 2.D0) * DSIN(OMEGA0)
Z0(5) = DTAN(INC0 / 2.D0) * DCOS(OMEGA0)
Z0(6) = M0
ZF(1) = AF
ZF(2) = EF * DSIN(WF + OMEGA0)
ZF(3) = EF * DCOS(WF + OMEGA0)
ZF(4) = DTAN(INCF / 2.D0) * DSIN(OMEGAF)
ZF(5) = DTAN(INCF / 2.D0) * DCOS(OMEGAF)
CALL DCOPY(N, LAM, 1, COCUR, 1)
C
INITIALIZE THE MASS AND ASSIGN DATA TO ALL VECTORS
ZMCUR = M0
DO 5 I = 1, N
  ZCUR(I) = Z0(I)
5 RETURN
END

SUBROUTINE ITERN
C******************************************************************************
C ITERN IS THE CALLING ROUTINE FOR A THE NUMERICAL INTEGRATOR WHICH WILL
C FIND THE INITIAL COSTATE VECTOR WHICH SATISFIES THE TARGET CONDITIONS.
C AT THIS TIME THE SUBROUTINE SECANT AND CALLING ROUTINES (WRITTEN
C BY GREG DUKEMAN) ARE USED TO ITERATE ON THE COSTATE.
C******************************************************************************
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
PARAMETER (ATHRESH = 7500.D0, N = 6, ITMAX = 250, MAXCALLS = 5)
DIMENSION XMI(N), X0(N), F(N), ZCUR(N), COCUR(N), XVAL(N), FOLD(N)
LOGICAL AOKAY
EXTERNAL FUNCT, SECANT
COMMON /STATE/ ZCUR, COCUR

AOKAY = .TRUE.
NWRITE = 36
EPSI = .005D0

 INPUT THE FIRST GUESSES OF THE COSTATE VARIABLES, XM1 AND X0.
XM1(6) REPRESENTS THE COSTATE MASS.

DO 20 I = 1, N
   XM1(I) = COCUR(I)
   X0(I) = COCUR(I) + EPSI
CONTINUE

CALL SECANT(XM1, X0, EPSI, N, NWRITE, F, FUNCT)

WRITE(NWRITE, 10) 'THE CURRENT STATE IS:', (ZCUR(I), I=1,N)
WRITE(NWRITE, 10) 'THE CURRENT COSTATE IS: ', (COCUR(I), I=1,N)
WRITE(NWRITE, 10) 'THE FINAL CONDITIONS DIFFERENCES IS: ', (F(I), I=1,N)

Determine if the semimajor axis is out of range
IF (ZCUR(1) .GT. ATHRESH) AOKAY = .FALSE.
IF (.NOT. AOKAY) THEN
   WRITE(NWRITE, *) 'TRY A NEW INITIAL PAIR OF GUESSES SINCE A
                  IS OUT OF RANGE.'
   RETURN
ENDIF

SUBROUTINE SECANT(XOLD, X, EPSI, N, NWRITE, F, FUNCT)

This subroutine is the n-dimensional extension of the well-known secant method (which solves the nonlinear scalar equation F(x) = 0).

***************INPUTS****************************
XOLD - old estimate of solution vector x
X - current estimate of solution vector x
EPSI - termination criterion
N - the dimension of the system (currently max of 100)
NWRITE - the logical unit number to send output to
FUNCT - the user-supplied routine used to evaluate the function values corresponding to trial solution vectors

************ Outputs ***********************
X - the solution vector
F - the function vector at the solution X

INTEGER I, IERROR, K, N, NFEVAL, NITER, NTRYS, NTRYSMAX,
$ NWRITE
DOUBLE PRECISION J(100, 100), DIFFX(100), F(N), XOLD(N),
$ X(N), XTRY(100), FTRY(100), C(100),
$ FOLD(100), POLD, DOTN, RNORM, ALPHA, EPSI,
$ P, DELTAX, TEMP

IF ( NWRITE .GT. 0 ) WRITE(NWRITE, 100)
NITER = 0
NFEVAL = 0
NTRYSMAX = 100
CALL FUNCT(X, F)
NFEVAL = NFEVAL + 1
P = DOTN(F, F, N)
CALL FUNCT(XOLD, FOLD)
NFEVAL = NFEVAL + 1
FOLD = DOTN(F, F, N)
IF ( P .GT. POLD ) THEN
   DO I = 1, N
      TEMP = XOLD(I)
      XOLD(I) = X(I)
      X(I) = TEMP
      TEMP = FOLD(I)
      FOLD(I) = F(I)
      F(I) = TEMP
   END DO
ELSE
   POLD = P
END IF
DO 2 I = 1, N
   DIFFX(I) = X(I) - XOLD(I)
2 IF ( NWRITE .GT. 0 ) WRITE(NWRITE, I01)
   $ NITER, NFEVAL, (F(I), X(I), I = 1, N)
   CALL VECTORASSIGNN(XTRY, X, N)
   Compute the pseudo Jacobian matrix (dF/dX)
   IF ( N .GT. 1 ) THEN
      DO 4 I = 1, N
         XTRY(I) = XOLD(I)
         CALL FUNCT(XTRY, FTRY)
         NFEVAL = NFEVAL + 1
         DELTAX = DIFFX(I)
         XTRY(I) = X(I)
         DO 5 K = 1, N
            J(K, I) = (F(K) - FTRY(K)) / DELTAX
         END DO
         CONTINUE
      END DO
      ELSE
         J(1,1) = (F(1) - FOLD(1)) / DIFFX(1)
   END IF
   Compute the n-vector J^-1 F ( = C )
   CALL GAUSS(J, F, C, N, 100, IERROR, RNORM)
   IF ( IERROR .EQ. 2 ) WRITE(NWRITE, *)' ERROR IN GAUSS ROUTINE'
   Save the current design vector for the next iteration
   CALL VECTORASSIGNN(XOLD, X, N)
   CALL VECTORASSIGNN(FOLD, F, N)
   Update the design vector and function vector
   ALPHA = .8D0
   NTRYS = 0
17 DO 6 I = 1, N
6   X(I) = XOLD(I) - ALPHA * C(I)
   CALL FUNCT(X, F)
   NFEVAL = NFEVAL + 1
   COMPUTE THE PERFORMANCE INDEX P AND THEN SEE IF IT HAS DECREASED.
   IF IT HAS INCREASED, THEN REDUCE THE STEP SIZE PARAMETER ALPHA
   IN AN ATTEMPT TO OBTAIN A DECREASE IN P.
P = DOTN(F, F, N)
IF ( P .GE. POLD ) THEN
  NTRYS = NTRYS + 1
  IF ( NTRYS .LT. NTRYSMAX ) THEN
    ALPHA = 0.5D0 * ALPHA
    GOTO 17
  ELSE
    WRITE(NWRITE, *)' NO CONVERGENCE'
    RETURN
  END IF
ELSE
  POLD = P
  DO 11 I = 1, N
    DIFFX(I) = - ALPHA * C(I)
  END IF
NITER = NITER + 1

C Check termination criterion
C
IF ((P .GT. EPSI) .AND. (DOTN(DIFFX, DIFFX, N) .GT. EPSI)) THEN
  GOTO 3
ELSE
  IF ( NWRITE .GT. 0 ) THEN
    WRITE(NWRITE, *)' CONVERGED SOLUTION'
    WRITE(NWRITE, 101) NITER, NFEVAL, ( F(I), X(I),
      I = 1, N )
  END IF
  RETURN
END IF

C Format statements
C
100 FORMAT(1H1,4X,12H# ITERATIONS,4X,15H# $ F EVALUATIONS,4X,20HBEST FUNCTION VALUES,4X,16H DESIGN $ VARIABLES///)
101 FORMAT(/1H ,I10,I17,D29.10,D23.10/(I ,D56.10,D23.10))

SUBROUTINE FUNCT(XVAL, FVAL)
*******************************************************************************
C FUNCT EVALUATES THE SYSTEM OF NONLINEAR EQUATIONS FOR A GIVEN
C SET OF INITIAL COSTATE VALUES IN ORDER TO SOLVE THE SYSTEM F(X)
C = 0 BY COMPUTING TRAJECTORIES AND EVALUATING THE FUNCTION F(X) WHICH
C IS COMPOSED OF THE FINAL CONDITIONS--IN THIS CASE, TARGET SEMIMAJOR
C AXIS, TARGET ECCENTRICITY AND TRANVERSALITY CONDITION MASS COSTATE.
*******************************************************************************
C*******************************************************************************
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
PARAMETER (N=6, NWRITE=36)
DIMENSION XVAL(N), FVAL(N), ZCUR(N), COCUR(N),
  AMX(5,3), AMXP(5,3,5), ZI2(2*N), DERZI2(2*N), ZF(N), TSTEP(3)
LOGICAL AVG, CHECK
COMMON /STATE/ ZCUR, COCUR
COMMON /MASSES/ ZM0, ZMCRU, ZMF
COMMON /COMASS/ COM0, COMCUR, COMF
COMMON /AVG_CHECK/ CHECK
COMMON /CONST/ POWR, C, AMU
COMMON /DERIV RK4/ DERZ12, Z12
COMMON /ZFINA/ ZF
COMMON /ECC LONG/ FECC1, FECC2
EXTERNAL RKFCT, RK4

TWOPI = 8.D0*DATAN(1.D0)
CHECK = .FALSE.
DO 5 I = 1, N
C  
C  COCUR(I) = XVAL(I) 
COMCUR = XVAL(N)  
C
DO 6 I = 1, N  
   Z12(I) = ZCUR(I)  
   Z12(I+N) = COCUR(I)  
6 CONTINUE  

NDIM = 12  
TSTEP(1) = 0.D0  
TSTEP(2) = .005D0  
TSTEP(3) = .005D0  

T = TSTEP(1)  
TF = TSTEP(2)  
DT = TSTEP(3)  

C***INTEGRATE STATE AND COSTATE ACROSS A STEP***  
DO WHILE (T LT. TF)  
   CALL RK4 (T, DT, TF, Z12, NDIM, RKFCT)  
   WRITE(NWRITE,*) ' FOR TIME T - ',T  
   WRITE(NWRITE,10) ' THE CURRENT STATE IS:', (Z12(I),I=1,N)  
   WRITE(NWRITE,10) ' THE CURRENT COSTATE IS:', (Z12(I),I=7,2*N)  
END DO  

DO 7 I = 1, N  
   ZCUR(I) = Z12(I)  
   COCUR(I) = Z12(N+I)  
7 CONTINUE  

C CHECK FINAL CONDITIONS  
FVAL(1) = DABS(ZCUR(1) - ZF(1))  
FVAL(2) = DSQRT(ZCUR(2)**2+ZCUR(3)**2)  
FVAL(3) = DABS(COMCUR - 1.D0)  
FVAL(4) = 0.D0  
FVAL(5) = 0.D0  
FVAL(6) = 0.D0  

FNORM = DOTN(FVAL, FVAL, N)  
WRITE(NWRITE,*) 'THE VALUE OF FNORM IS: ',FNORM,' AT T = ',T  

10 FORMAT (/A/6(F25.12/)/)  
RETURN  
END  

SUBROUTINE RK4(T, DT, TF, X, NX, DESUB)  
C  
C This is the classical fourth-order Runge Kutta numerical integration method.  
C  
C Variable definitions . . . .  
C  
C T (scalar) The current value of the independent variable  
C DT (scalar) The integration step size  
C TF (scalar) The final point in the independent variable to integrate to  
C X (vector) The vector of states or dependent values  
C NX (integer) The number of first-order differential equations to integrate  
C DESUB The differential equations subroutine  
C  
DOUBLE PRECISION X(30),XP(30),F1(30),F2(30),F3(30),F4(30), T, $  
DT, TP, TF  
INTEGER I, NX  
CALL DESUB(T,X,F1)  
IF(ABS(DT) .GT. ABS(TF-T)) DT=TF-T
DO 10 I=1,NX
10 XP(I)=X(I)+DT*F1(I)/2.D0
   TP=T+DT/2.D0
   CALL DESUB(TP,XP,F2)
   DO 20 I=1,NX
20 XP(I)=X(I)+DT*F2(I)/2.D0
   CALL DESUB(TP,XP,F3)
   DO 30 I=1,NX
30 XP(I)=X(I)+DT*F3(I)
   TP=T+DT
   CALL DESUB(TP,XP,F4)
   DO 40 I=1,NX
40 XP(I)=X(I)+DT*(F1(I)/6.D0+F2(I)/3.D0+F3(I)/3.D0+F4(I)/6.D0)
   T=T+DT
RETURN
END

SUBROUTINE RKFCT(T,Z12,DERZ12)
C
RKFCT IS CALLED BY RK4--RUNGE-KUTTA INTEGRATOR--TO COMPUTE THE
C AVERAGED DERIVATIVES TO BE INTEGRATED.
C
PARAMETER(N = 6)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
LOGICAL AVG, CHECK
DIMENSION Z12(12), DERZ12(12), ZCUR(N), COCUR(N), ZSUM(12),
   AMX(5,3), AMXP(5,3,5), DZ(N), ZDOTT(N), COZDOTT(N), G(12), H(12)
COMMON /STATE/ ZCUR, COCUR
COMMON /VEC12/ ZVEC, COVEC
COMMON /MASS/ ZM0, ZMCUR, ZMF
COMMON /COMASS/ COM0, COMCUR, COMF
COMMON /MPROD5/ DZ, PVLAM
COMMON /AVG_CHECK/ CHECK
COMMON /AVERAGE/ AVG
COMMON /CONST/ POWR, C, AMU
COMMON /DERIV/ ZDOTT, ZMDOT, COZDOTT, COMDOT
COMMON /ORBIT2/ XI,YI,RA, PZ20,PZ26,PZ29,PZ35
COMMON /ECC_LONG/ FECCI, FECC2
EXTERNAL EVALMP, PRIMER, AVGTST, INTEG, QTRAP, FCT

TWOPI=8.D0*DATAN(1.D0)
IFLAG = 2

C
CALL EVALMP(ZCUR,FECCI,AMU,AMX,AMXP,IFLAG)
C
CALL PRIMER(AMX,AMXP)
AVG = .TRUE.
C
DO 3 I = 1, N
   ZDOTT(I)=0.D0
   COZDOTT(I)=0.D0
   ZSUM(I)=0.D0
   ZSUM(N+I)=0.D0
3 CONTINUE

C***THIS CHECK IS BASED ON THE NUMBER OF CALLS IN RK4
C
IF (CHECK) THEN
   CALL AVGTST(AVG)
ENDIF

IF (AVG) THEN
   CALL SWCHPTS
   CALL INTEG
ELSE
   PRINT*, 'NOT AVERAGING NOW'
C***THIS CHANGE IS TO COMPUTE A 1 BURN ONLY 5/28/93
C
   CALL QTRAP(FCT,0.D0,TWOPI,ZSUM,2*N,N)
   CALL QUAD(0.D0,TWOPI,FCT,ZSUM,12,G,H,2*N)
   DO 5 I = 1, N
   C
5 CONTINUE
C
ZDOTT(I) = ZDOTT(I) + ZSUM(I)
COZDOTT(I) = COZDOTT(I) + ZSUM(N+I)

5 CONTINUE
ENDIF

DO 10 I = 1, N
   DERZ12(I) = ZDOTT(I)
   DERZ12(I+6) = COZDOTT(I)
10 CONTINUE
IF (ABS(ZDOTT(1)) .GT. 200.) AVG = .FALSE.

2 RETURN
END

SUBROUTINE AVGTST (AVG)

C EVALUATE HOW FAST THE EQUINOCTIAL ELEMENTS ARE CHANGING

IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION ZDOT(5), DZ(5)
LOGICAL AVG
COMMON /MPROD5/ DZ, PVLAM
COMMON /CONST/ POWER, C, AMU
COMMON /MASS/ ZM0, ZMCUR, ZMF

C CALCULATE PARAMETER FOR STOPPING
PARAMETER (THRESH = 100.D0)
PARAMETER (IR=5, IC=3)

AVG = .TRUE.
DO 5 I = 1, 5
   ZDOT(I) = ((2.D0 * POWER) / ZMCUR * C) * DZ(I)
5 CONTINUE
IF (ZDOT(1) .GT. THRESH) THEN
   AVG = .FALSE.
C PRINT*, 'THE ORBITAL ELEMENTS ARE CHANGING TOO RAPIDLY TO AVERAGE.'
ENDIF
RETURN
END

SUBROUTINE SWCHPTS

C PURPOSE: FIND THE ZEROES OF THE SWITCH FUNCTION SIGMA(F)
PARAMETER(N = 6)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION MESHPT(100), LOWER, MDPT, NP1, NP2, INCR
INTEGER PTON, FKOUNT
DIMENSION LHS(100), FNP(100), ROOTS(100), ZCUR(N), LCUR(N)
LOGICAL STARTPOS, ENDPOS
COMMON /STATE/ ZCUR, LCUR
COMMON /CONST/ POWER, C, AMU
COMMON /GREEKS/ ALPHA, BETA, DELTA, GAMMA
COMMON /ENDDATA/ NP1, NP2, STARTPOS, ENDPOS
COMMON /SWITCH/ ROOTS,NROOT, FNP, FKOUNT, NKOUNT
EXTERNAL SIGMAF, MYSIGN, DZBREN

TWOPI = 8.D0*DATAN(1.D0)
NP1 = 0.D0
NP2 = TWOPI
MAXFN = 100
ERRABS = 1.D-3
ERRREL = 1.D-4

ALPHA = ZCUR(3) * LCUR(2) - ZCUR(2) * LCUR(3)
BETA = (1.D0 + DSQRT(1.D0 - ZCUR(2)**2 + ZCUR(3)**2))**(-1)
DELTA = DSQRT(AMU * ZCUR(1) * (1.D0 - ZCUR(2)**2 + ZCUR(3)**2))
GAMMA = (0.5D0) * (1.D0 + ZCUR(4)**2 + ZCUR(5)**2)

STARTPOS = .FALSE.
ENDPOS = .FALSE.
nroot = 0
NTOP = 0
PTON = 0
FKOUNT = 0
NKOUNT = 61
INCR = (NP2 - NP1) / NKOUNT
MESHPT(1) = NP1

LHS(1) = MYSIGN(SIGMAF(MESHPT(1)))
DO 10 I = 2, NKOUNT
   MESHPT(I) = MESHPT(I-1) + INCR
   LHS(I) = MYSIGN(SIGMAF(MESHPT(I)))
10 CONTINUE

DO 20 I = 1, NKOUNT - 1
   IF (LHS(I) .EQ. LHS(I+1)) THEN
      IF (LHS(I) .EQ. 0) THEN
         NROOT = NROOT + 1
         ROOTS(NROOT) = MESHPT(I)
         ENDIF
      ELSE
         LOWER = MESHPT(I)
         UPPER = MESHPT(I+1)
         CALL DZBREN(SIGMAF, ERRABS, ERRREL, LOWER, UPPER, MAXFN)
         NROOT = NROOT + 1
         ROOTS(NROOT) = UPPER
      ENDIF
   ELSE
      NTOP = NTOP + 1
      FNP(NTOP) = ROOTS(NROOT)
      FKOUNT = FKOUNT + 1
      ELSE
      PTON = PTON + 1
      FNP(PTON) = ROOTS(NROOT)
   ENDIF
20 CONTINUE

IF (NROOT .EQ. 0) ROOTS(1) = TWOPI

IF (LHS(1) .GT. 0) STARTPOS = .TRUE.
MDPT = (ROOTS(NROOT) + NP2) / 2.D0
CHECKSIGN = MYSIGN(SIGMAF(MDPT))
IF (CHECKSIGN .GT. 0) ENDPOS = .TRUE.
RETURN
END

SUBROUTINE INTEG

C PURPOSE: PREPARE INTEGRANDS FOR QUADRATURE AND THEN CALL QUADRATURE
C ROUTINE, EITHER QUAD4, QUAD8, QUAD16 OR QUAD32

IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION NP1, NP2, N
PARAMETER (NN = 6)
DIMENSION ZDOTT(NN), COZDOTT(NN), DERZ12(12), Z12(12), ZSUM(12),
& ROOTS(100), FNP(100), G(12), H(12)
INTEGER FKOUNT
LOGICAL STARTPOS, ENDPOS, THRUST
COMMON /STATE/ ZCUR, COCUR
COMMON /MPROD5/ DZ, PVLAM
COMMON /AVERAGE/ AVG
COMMON /MASSES/ ZM0, ZMCUR, ZMF
COMMON /DERIV/ ZDOTT, ZMDOT, COZDOTT, COMDOT
COMMON /DERIV_RK4/ DERZ12, Z12
COMMON /ENDDATA/ NP1, NP2, STARTPOS, ENDPOS
COMMON /SWITCH/ ROOTS, NROOT, FNP, FKOUNT, NKOUNT
EXTERNAL QUAD, FCT, QTRAP

IF (NROOT .EQ. 0) THRUST = .FALSE.
IF (NROOT .EQ. 0) STARTPOS = .TRUE.
HAVG = 0.D0
COMDOT = 0.D0
ZMDOT = 0.D0
DO 3 I = 1, N
   ZDOTT(I) = 0.D0
   COZDOTT(I) = 0.D0
   ZSUM(I) = 0.D0
   ZSUM(NN+I) = 0.D0
3 CONTINUE
IF (STARTPOS) THEN
   CALL QUAD (NP1, ROOTS(1), FCT, ZSUM, Z12, G, H, 2*NN)
   CALL QTRAP (FCT, NP1, ROOTS(1), ZSUM, 2*NN, M)
DO 20 I = 1, NN
   ZDOTT(I) = ZDOTT(I) + ZSUM(I)
   COZDOTT(I) = COZDOTT(I) + ZSUM(NN+I)
20 CONTINUE
ENDIF
IF (ENDPOS) THEN
   CALL QUAD (ROOTS(NROOT), NP2, FCT, ZSUM, Z12, G, H, 2*NN)
   CALL QTRAP (FCT, ROOTS(NROOT), NP2, ZSUM, 2*NN, M)
DO 30 I = 1, NN
   ZDOTT(I) = ZDOTT(I) + ZSUM(I)
   COZDOTT(I) = COZDOTT(I) + ZSUM(NN+I)
30 CONTINUE
ENDIF
IF (THRUST) THEN
   DO 40 I = 1, FKOUNT
      CALL QUAD (ROOTS(2*I), ROOTS(2*I+1), FCT, ZSUM, Z12, G, H, 2*NN)
      CALL QTRAP (FCT, ROOTS(2*I), ROOTS(2*I+1), ZSUM, 2*NN, M)
   DO 45 J = 1, NN
      ZDOTT(J) = ZDOTT(J) + ZSUM(J)
      COZDOTT(J) = COZDOTT(J) + ZSUM(NN+J)
   45 CONTINUE
   40 CONTINUE
ELSE
   DO 50 I = 1, FKOUNT
      CALL QUAD (ROOTS(2*I-1), ROOTS(2*I), FCT, ZSUM, Z12, G, H, 2*NN)
      CALL QTRAP (FCT, ROOTS(2*I-1), ROOTS(2*I), ZSUM, 2*NN, M)
   DO 55 J = 1, NN
      ZDOTT(J) = ZDOTT(J) + ZSUM(J)
      COZDOTT(J) = COZDOTT(J) + ZSUM(NN+J)
   55 CONTINUE
   50 CONTINUE
ENDIF
ZMDOT = ZDOTT(NN)
COMDOT = COZDOTT(NN)
RETURN
END

DOUBLE PRECISION FUNCTION SIGMAF(F)
C PURPOSE: TO COMPUTE THE EXPRESSION OF THE FUNCTION LAMBDA
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION LAMSQ, LAMOFF, LMCUR, LCUR, LVEC
DIMENSION ZCUR(6), LCUR(6), RVEC(3), VELVEC(3)
INTEGER I, ILEN
PARAMETER (ILEN=3)
COMMON /STATE/ ZCUR, LCUR
COMMON /CONST/ POWR, C, AMU
COMMON /MASSES/ ZM0, ZMCUR, ZMF
COMMON /COMASS/ COM0, LMCUR, COMF
COMMON /GREGGS/ ALPHA, BETA, DELTA, GAMMA
EXTERNAL DNRM2

X1 = ZCUR(1) * ((1.0 - (ZCUR(2)**2) * BETA) * DCOS(F) + ZCUR(2) * 
     ZCUR(3) * BETA * DSIN(F) - ZCUR(3))
Y1 = ZCUR(1) * ((1.0 - (ZCUR(3)**2) * BETA) * DSIN(F) + ZCUR(2) * 
     ZCUR(3) * BETA * DCOS(F) - ZCUR(2))

RVEC(1) = X1
RVEC(2) = Y1
RVEC(3) = 0.0
R = DNRM2(ILEN, RVEC, 1)

XDOT = (ZCUR(2) * ZCUR(3) * BETA * DCOS(F) - (1.0 - (ZCUR(2)**2) 
     * BETA) * DSIN(F)) * DSQRT(AMU * ZCUR(1)) / R
YDOT = ((1.0 - ZCUR(3)**2 * BETA) * DCOS(F) - ZCUR(2) * 
     ZCUR(3) * BETA * DSIN(F)) * DSQRT(AMU * ZCUR(1)) / R

VELVEC(1) = XDOT
VELVEC(2) = YDOT
VELVEC(3) = 0.0
VEL = DNRM2(ILEN, VELVEC, 1)

PROD1 = 2.0 * ZCUR(1)**2 * XDOT * LCUR(1) + (Y1 * XDOT - DELTA) * LCUR(2) 
     + Y1 * YDOT * LCUR(3)
PROD2 = 2.0 * ZCUR(1)**2 * YDOT * LCUR(1) - (X1 * XDOT * LCUR(2)) 
     + (X1 * YDOT + DELTA) * LCUR(3)

LAMSQ = (PROD1**2) / AMU**2 + (PROD2**2) / AMU**2 + (PROD3**2) / DELTA**2
LAMOFF = DSQRT(LAMSQ)
SIGMAF = LAMOFF - (ZMCUR * LMCUR / C)

RETURN
END

DOUBLE PRECISION FUNCTION SZFF(F)
C PURPOSE: CALCULATE THE FUNCTION S(Z,F) USED IN THE INTEGRATION IN
C AVERAGING
C
PARAMETER(N = 6)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DOUBLE PRECISION LCUR
DIMENSION ZCUR(N), LCUR(N)
COMMON /STATE/ ZCUR, LCUR

FACTOR = (1.0 - ZCUR(3)*DCOS(F) - ZCUR(2)*DSIN(F))
SZFF = FACTOR / (8.0 * DATAN(1.0))
RETURN
END

SUBROUTINE PRIMER(AMX,AMXP)
C PRIMER CONSTRUCTS THE PRIMER VECTOR AND THE OPTIMAL THRUST
C DIRECTION U^.`
C
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
PARAMETER (IM=5, IN=3, N=6)
DIMENSION ZCUR(N), COCUR(N), COCUR5(5), UHAT(3), AMX(5, 3), 
     AMXP(5, 3, 5), DZ(5), UVEC(3), UPROJ(3)
COMMON /STATE/ ZCUR, COCUR
COMMON /MPRODS/ DZ, PVLAM
COMMON /HAT/ UHAT
COMMON /ORBIT2/ X1, Y1, RA, PZ20, PZ26, PZ29, PZ35
COMMON /ORBIT3/ X1DOT, Y1DOT

EXTERNAL DNRM2, DMURRV, DDOT

DO 3 I = 1, IM
     COCUR5(I) = COCUR(I)
3
CALL DMURRV(IM, IN, AMX, IM, IM, COCUR5, 2, IN, UVEC)
UVAL = (UVEC(1)*X1 + UVEC(2)*Y1)/(RA**2)
UPROJ(1)=UVAL*X1/ZCUR(1)
UPROJ(2)=UVAL*Y1/ZCUR(1)
UPROJ(3)=0.
DO 4 I=1, 3
UVEC(I) = UVEC(I)-UPROJ(I)
UVEC(1)=X1DOT/ZCUR(1)
UVEC(2)=Y1DOT/ZCUR(1)
UVEC(3)=0.
ULEN = DNRM2(IN, UVEC, 1)
DO 5 I = 1, IN
  UHAT(I) = UVEC(I)/ULEN
CALL DMURRV(IM, IN, AMX, IM, IN, UHAT, 1, IM, DZ)
PVLAM = DDOT(IM, COCUR5, I, DZ, I)
RETURN
END

SUBROUTINE VECTORASSIGNN(X, Y)
C
C This subroutine assigns the 3-element vector Y to the 3-element vector X
C
DOUBLE PRECISION X(3), Y(3)
X(1) = Y(1)
X(2) = Y(2)
X(3) = Y(3)
RETURN
END

DOUBLE PRECISION FUNCTION DOTN(F,G,N)
INTEGER N
DOUBLE PRECISION F(N), G(N)
DOTN = 0.D0
DO 10 I = 1, N
  DOTN = DOTN + F(I)*G(I)
10 CONTINUE
RETURN
END

SUBROUTINE GAUSS(A,B,X,N,MAINDM,IERROR,RNORM)
INTEGER NM1, NP1, I, J, K, N, IP1, IERROR, IPIVOT, MAINDM
DOUBLE PRECISION A(MAINDM, MAINDM), B(N), X(N), RNORM
REAL*16 AUG(100, 101), Q, RSQ, RESI, Rmag,
   PIVOT, TEMP
NM1=N-1
NP1=N+1

C
C SET UP THE AUGMENTED MATRIX FOR AX=B.
C
DO 2 I=1,N
DO 1 J=1,N
AUG(I,J)=A(I,J)
1 CONTINUE
AUG(I,NP1)=B(I)
2 CONTINUE

C
C THE OUTER LOOP USES ELEMENTARY ROW OPERATIONS TO TRANSFORM
C THE AUGMENTED MATRIX TO ECHelon FORM.
C
DO 8 I=1,NM1
SEARCH FOR THE LARGEST ENTRY IN COLUMN I, ROWS I THROUGH N.

PIVOT IS THE ROW INDEX OF THE LARGEST ENTRY.

PIVOT=0.
DO 3 J=I,N
  TEMP=ABS(AUG(J,I))
  IF(PIVOT .GE. TEMP)GOTO 3
  PIVOT=TEMP
  IPIVOT=J
3 CONTINUE
  IF(PIVOT .EQ. 0.)GOTO 13
  IF(IPIVOT .EQ. I)GOTO 5

INTERCHANGE ROW I AND ROW IPIVOT.

DO 4 K=I,NPI
  TEMP=AUG(I,K)
  AUG(I,K)=AUG(IPIVOT,K)
  AUG(IPIVOT,K)=TEMP
4 CONTINUE

ZERO ENTRIES (I+1,I), (I+2,I), ..., (N,I) IN THE AUGMENTED MATRIX.

5 IPI=I+1
  DO 7 K=IPI,N
    Q=-AUG(K,I)/AUG(I,I)
    AUG(K,I)=0.
  DO 6 J=IPI,NPI
    AUG(K,J)=Q*AUG(I,J)+AUG(K,J)
  6 CONTINUE
  7 CONTINUE
  8 CONTINUE
  IF(AUG(N,N) .EQ. 0.)GOTO 13

BACKSOLVE TO OBTAIN A SOLUTION TO AX=B.

X(N)=AUG(N,NPI)/AUG(N,N)
DO 10 K=1,NM1
  Q=0.
  DO 9 J=1,K
    Q=Q+AUG(N-K,NPI-J)*X(NPI-J)
  9 CONTINUE
  X(N-K)=(AUG(N-K,NPI)-Q)/AUG(N-K,N-K)
10 CONTINUE

CALCULATE THE NORM OF THE RESIDUAL VECTOR, B-AX.

SET IERROR=1 AND RETURN.

RSQ=0.
DO 12 I=1,N
  Q=0.
  DO 11 J=1,N
    Q=Q+A(I,J)*X(J)
  11 CONTINUE
  RESI=B(I)-Q
  RMAG=ABS(RESI)
  RSQ=RSQ+RMAG**2
12 CONTINUE
  RNORM=SQRT(RSQ)
  IERROR=1
RETURN

ABNORMAL RETURN --- REDUCTION TO ECHELON FORM PRODUCES A ZERO ENTRY ON THE DIAGONAL. THE MATRIX A MAY BE SINGULAR.

13 IERROR=2
INTEGER FUNCTION MYSISGN(X)
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C CALCULATE THE SIGN OF THE FUNCTION SIGMAF TO HELP DEFINE ENDPOINTS.

MYSISGN = 0
IF (X .GT. 0.D0) MYSISGN = 1
IF (X .LT. 0.D0) MYSISGN = -1
RETURN
END

SUBROUTINE FCT(F1,F2,ZI2,H,G)
C CALLED BY QUAD4 TO COMPUTE STATE AND COSTATE INTEGRANDS
PARAMETER(N = 6)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
PARAMETER(IFLAG-I)
DIMENSION ZCUR(N), COCUR(N), AMX(5,3), HZ(5),
+ AMXP(5,3,5), DZ(5), DHDLZ(5), DSDZ(5), DHDZ(5), UHMH(1,5),
+ ANEWMX(5,3), H(12), G(12), UHAT(3), PROD(5), ZI2(12)
LOGICAL AVG
COMMON /STATE/ ZCUR, COCUR
COMMON /CONST/ POWR, C, AMU
COMMON /AVERAGE/ AVG
COMMON /MASSES/ ZM0, ZMCUR, ZMF
COMMON /COMASS/ COM0, COMCUR, COMF
COMMON /MPROD5/ DZ, PVLAM
COMMON /HAT/ UHAT
COMMON /ORBIT2/ X1,Y1,RA,PZ20,PZ26,PZ29,PZ35
EXTERNAL COMPHZ, EVALMP, PRIMER, SZFF, DDOT, DMURRV
TWOPI = 8.D0 * DATAN(1.D0)
DO 5 I = 1, 5
   DO 6 J = 1, 3
      ANEWMX(I,J) = 0.D0
6 CONTINUE
5 CONTINUE
INTFLAG = 1
F = F1
8 CALL EVALMP (ZCUR, F, AMU, AMX, AMXP, IFLAG)
CALL PRIMER (AMX,AMXP)
DO 10 I = 1, 5
   UHMH(I,1) = DZ(I)
10 CONTINUE
DO 20 I = 1, 5
   IF (AVG) THEN
      DHDLZ(I) = ((2.D0 * POWR) / ZMCUR*C) * UHMH(I,1) * SZFF(F)
   ELSE
      DHDLZ(I) = ((2.D0 * POWR) / ZMCUR*C) * UHMH(I,1)
   ENDIF
20 CONTINUE
DO 30 I = 1, 5
   DO 35 J = 1, 3
      DO 40 K = 1, 5
         ANEWMX(K,J) = AMXP(K,J,I)
40 CONTINUE
35 CONTINUE
C CHECK THE PREVIOUS CALCULATIONS USING ANOTHER IMSL ROUTINE
CALL COMPHZ(AMXP,COCUR(5),UHAT,HZ)
C HZ(I) = DBLINF(5,3,ANEWMX,5,COCUR,UHAT)
C CALL DMURRV(5,3,ANEWMX,5,3,UHAT,1,5,PROD)
C HZ(I) = DDOT(5,COCUR,1,PROD,1)
30 CONTINUE

DSDZ(1) = 0.D0
DSDZ(2) = - DSIN(F) / TWOPI
DSDZ(3) = - DCOS(F) / TWOPI
DSDZ(4) = 0.D0
DSDZ(5) = 0.D0

C CALCULATE THE HAMILTONIAN OR THE AVERAGED HAMILTONIAN.

IF (AVG) THEN
    HAM = ((2.D0 * POWR) / (ZMCUR * C)) * (PVLAM - ZMCUR*COMCUR/C) + * SZFF(F)
ELSE
    HAM = ((2.D0 * POWR) / (ZMCUR * C)) * (PVLAM - ZMCUR*COMCUR/C)
ENDIF

DO 50 I = 1, 5
    IF (AVG) THEN
        DHDZ(I) = - (((2.D0*POWR)/ZMCUR*C)*HZ(I)*SZFF(F) + HAM*DSDZ(I))
    ELSE
        DHDZ(I) = - (((2.D0*POWR)/ZMCUR*C)*HZ(I) + HAM*DSDZ(I))
    ENDIF
50 CONTINUE

C THIS PART CALCULATES THE PARTIAL DERIVATIVE DH/DLM.
IF (AVG) THEN
    DHDLM = - ((2.D0 * POWR) / C**2) * SZFF(F)
ELSE
    DHDLM = - ((2.D0 * POWR) / C**2)
ENDIF

C THIS PART CALCULATES THE PARTIAL DERIVATIVE DH/DM.
IF (AVG) THEN
    DHDM = (2.D0 * POWR) / (ZMCUR**2 * C) * PVLAM * SZFF(F)
ELSE
    DHDM = (2.D0 * POWR) / (ZMCUR**2 * C) * PVLAM
ENDIF

C IF (INTFLAG .EQ. 0) GOTO 65

DO 60 I = 1, N-1
    H(I) = DHDLZ(I)
    H(N+I) = DHDZ(I)
60 CONTINUE
H(N) = DHDLM
H(2*N) = DHDM

C IF (M .EQ. 1) THEN
    F = F2
    INTFLAG=0
    GO TO 8
C ENDIF
65 CONTINUE

C 60 I = 1, N-1
C    G(I) = DHDLZ(I)
C    G(N+I) = -DHDZ(I)
70 CONTINUE
G(N) = DHDLM
G(2*N) = DHDM

RETURN
END
SUBROUTINE COMPHZ(A,Y,X,HZ)
C THIS SUBROUTINE COMPUTES THE TENSOR NEEDED IN PARTIAL OF COSTATE
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION A(5,3,5),Y(5),X(3),HZ(5),KIJ(5,3,5)

DO 10 I=1,5
DO 20 J=1,3
DO 30 K=1,5
     KIJ(I,J,K) = Y(K)*A(I,J,K)
10 CONTINUE
20 CONTINUE
30 CONTINUE

DO 32 I=1,5
     HZ(I) = 0.D0
32 CONTINUE

DO 35 I=1,5
DO 40 J=1,3
DO 45 K=1,5
     HZ(I) = HZ(I) + KIJ(K,J,I)*X(J)
35 CONTINUE
40 CONTINUE
45 CONTINUE

RETURN
END

SUBROUTINE EVALMP(X, THETA, AMU, AM, PAM, IMFLAG)
C EVALMPEVALMPC
C THIS SUBROUTINE EVALUATES THE 5X3 MATRIX M AND THE
C 5X3X5 PARTIAL OF M WRT X
C
C IF IMFLAG=1, BOTH M (AM) AND ITS PARTIAL (PAM) ARE EVALUATED
C IF IMFLAG=2, ONLY M (AM) IS EVALUATED
C IF IMFLAG=3, ONLY THE PARTIAL OF M (PAM) IS EVALUATED
C
IMPLICIT DOUBLE PRECISION(A-H,O-Z), INTEGER (I-N)
DIMENSION X(5), AM(5,3), PAM(5,3,5)
COMMON /ORBIT2/ X1,Y1,RA, PZ20,PZ26,PZ29,PZ35
COMMON /ORBIT3/ XIDOT, YIDOT
EN=DSQRT (AMU/X(1)**3)
if ((X(2)**2 + X(3)**2) .ge. 1.d0) then
   print*, 'trouble in evalmp'
   return
endif
RHO= DSQRT(1.D0- X(2)**2- X(3)**2)
BETA= 1.D0/(1.D0 + RHO)
CT= DCOS (THETA)
ST= DSIN (THETA)
RA= 1.D0-X(3)*CT -X(2)*ST
ZETA= X(3)*ST-X(2)*CT
BETA3= BETA**3/(1.D0 -BETA)
X1= X(1)*((1.D0 -X(2)**2*BETA)*CT +X(2)*X(3)*BETA*ST -X(3))
Y1= X(1)*((1.D0 -X(3)**2*BETA)*ST +X(2)*X(3)*BETA*CT -X(2))
X11=X1
Y11=Y1
X1DOT= -((1.D0 -X(2)**2*BETA)*ST -X(2)*X(3)*BETA*CT)*EN*X(1)/RA
Y1DOT= ((1.D0 -X(3)**2*BETA)*CT -X(2)*X(3)*BETA*ST)*EN*X(1)/RA
PZ1= X1*(ZETA*(BETA+X(2)**2*BETA3) - (X(2)*BETA -ST)*CT/RA)
PZ2= -X1*(ZETA*X(2)*X(3)*BETA3 +1.D0 + (ST-X(2)*BETA)*ST/RA)
PZ3= X1*(-ZETA*X(2)*X(3)*BETA3-1.D0 + (X(3)*BETA -CT)*CT/RA)
PZ4= X1*(-ZETA*(BETA +X(3)**2*BETA3) +(CT-X(3)*BETA)*ST/RA)
IF (IMFLAG .EQ. 3) GO TO 10

C

IF DO NOT WANT TO EVALUATE PARTIAL OF M, BRANCH TO 10

AM(1,1) = 2.D0*X1DOT/(EN**2*X(1))
AM(1,2) = 2.D0*Y1DOT/(EN**2*X(1))
AM(1,3) = 0.D0

DUM = RH0/(EN*X(1)**2)

AM(2,1) = DUM*(PZ2 - X(2)*BETA*X1DOT/EN)
AM(2,2) = DUM*(PZ4 - X(2)*BETA*Y1DOT/EN)
AM(2,3) = DUM*(X(3) * (X(5) * Y1 - X(4) * X1))/RH0**2
AM(3,1) = -DUM*(PZ1 + X(3)*BETA*X1DOT/EN)
AM(3,2) = -DUM*(PZ3 + X(3)*BETA*Y1DOT/EN)
AM(3,3) = -DUM*(X(2) * (X(5) * Y1 - X(4) * X1))/RH0**2
AM(4,1) = 0.D0
AM(4,2) = 0.D0

DUM = (1.D0 + X(4)**2 + X(5)**2)/(2.D0*EN*X(1)**2*RHO)

AM(4,3) = DUM*Y1
AM(5,1) = 0.D0
AM(5,2) = 0.D0
AM(5,3) = DUM*X1

IF (IMFLAG .EQ. 2) RETURN

C

IF WE ONLY WISH TO EVALUATE M THEN PROGRAM RETURNS HERE

CA = DSQRT(AMU/X(1))/RA

PZ5 = X(2)*BETA3
PZ6 = X(3)*BETA3
PZ9 = CA*ST/RA
PZ10 = CA*CT/RA

PZ20 = X(1)*(-2.D0*X(2)*BETA*CT + X(3)*BETA*ST + PZ5*ZETA*X(2))
PZ26 = X(1)*X(2)*BETA*ST - 1.D0 + PZ6*X(2)*ZETA)
PZ29 = X(1)*X(3)*BETA*CT - 1.D0 - PZ5*X(3)*ZETA)
PZ35 = X(1)*(-2.D0*X(3)*BETA*ST + X(2)*BETA*CT - PZ6*X(3)*ZETA)
PZ11 = -X1DOT/(2.D0*X(1))
PZ12 = -Y1DOT/(2.D0*X(1))

DUM1 = 1.D0 - RA
PZ13 = -CA*(-2.D0*X(2)*BETA*ST - X(3)*BETA*CT - PZ5*X(2)*DUM1) + PZ9

PZ14 = -CA*(-X(2)*BETA*CT - PZ6*X(2)*DUM1) + PZ10*X1DOT/CA
PZ15 = -CA*X(3)*BETA*ST + PZ5*X(3)*DUM1) + PZ9*Y1DOT/CA
PZ16 = -CA*X(2)*BETA*CT + X(2)*BETA*ST + PZ6*DUM1*X(3))

DUM = BETA + X(2)*PZ5
PZ17 = 1.D0 + PZ5*X(2)* (3.D0/BETA + 1.D0/(1.D0 - BETA))
PZ18 = (2.D0 + PZ17)*PZ5
PZ19 = PZ17*PZ6

DUM2 = X(2)*BETA - ST

PZ21 = -X(1)* (CT*DUM - ZETA*PZ18 + CT*DUM + CT*ST*DUM2/RA**2)
PZ22 = X(1) * (ST*DUM + ZETA*PZ19 - CT*X(2)*PZ6/RA - CT**2*DUM2/RA**2)
PZ23 = BETA3* (3.D0/BETA + 1.D0/(1.D0 - BETA))
PZ24 = PZ23*PZ5
PZ25 = PZ23*PZ6

PZ27 = X(1)*(-CT*X(2)*X(3)*BETA3 + ZETA*X(3)* (BETA3 + X(2)*PZ24)

1 + (ST* (BETA + X(2)*PZ5))/RA + ST**2*DUM2/RA**2)
PZ28 = X(1)* (ST*X(2)*X(3)*BETA3 + ZETA*X(2)* (BETA3 + X(3)*PZ25)

1 +(X(2)*ST*PZ6/RA + ST*CT*DUM2/RA**2)

DUM2 = X(3)*BETA -CT

PZ30 = X(1)* (CT*X(2)*X(3)*BETA3 - ZETA*X(3)* (BETA3 + X(2)*PZ24)

1 + CT*X(3)*PZ5/RA + CT*ST*DUM2/RA**2)
PZ31 = X(1)*(-ST*X(2)*X(3)*BETA3 - ZETA*X(2)* (BETA3 + X(3)*PZ25)

1 + CT* (BETA + X(3)*PZ6)/RA + CT**2*DUM2/RA**2)

DUM = BETA + X(3)*PZ6

PZ32 = 1.D0 + PZ6*X(3)* (3.D0/BETA + 1.D0/(1.D0 - BETA))
PZ33 = PZ32*PZ5
PZ34 = PZ32*PZ6 + 2.D0*X(3)*BETA3

PZ36 = X(1) * (CT*DUM - ZETA*PZ33 - ST*X(3)*PZ5/RA - ST**2*DUM2/RA**2)
PZ37 = X(1)*(-ST*DUM - ZETA*PZ34 - ST*(BETA + X(3)*PZ6)/RA - ST*CT

1 + DUM2/RA**2)

DO 20 J=1,2
PAM(1, I, J) = 3.0*D0*AM(1, J) / (2.0*D0*X(1))
DUM = 3.0*D0*X(1) ** 2 / AMU
PAM(1, 1, 2) = PZ13*DUM
PAM(1, 1, 3) = PZ14*DUM
PAM(1, 2, 2) = PZ15*DUM
PAM(1, 2, 3) = PZ16*DUM
DUM = DSQRT (AMU*X(1))
CB = RHO/DUM
PZ38 = -X(2) * CB/RHO**2
PZ39 = -X(3) * CB/RHO**2
PAM(2, 1, 1) = AM(2, 1) / (2.0*D0*X(1))
PAM(2, 1, 2) = -CB*BETA*X1DOT/EN + PZ38*AM(2, 1) / CB + CB*(PZ27
1 - X(2)**2) / X1DOT*PZ25/EN
PAM(2, 1, 3) = PZ39*AM(2, 1) / CB + CB*(PZ28 - X(2)**2)*X1DOT/EN
1 - X(2)**2)
PAM(2, 2, 1) = AM(2, 2) / (2.0*D0*X(1))
PAM(2, 2, 2) = PZ38*AM(2, 2) / CB + CB*(PZ36 - BETA*Y1DOT/EN - X(2)
1 * Y1DOT*PZ25/EN - X(2)**2) / BETA*PZ15/EN
PAM(2, 2, 3) = PZ39*AM(2, 2) / CB + CB*(PZ37 - X(2)**2)*Y1DOT/PZ6/EN
1 - X(2)**2)
PAM(2, 3, 1) = AM(2, 3) / (2.0*D0*X(1))
DUM1 = X(5)*Y1 - X(4)*X1
PAM(2, 3, 2) = X(3) * (X(5)*PZ29 - X(4)*PZ20) / (RHO*DUM) + X(2)**2
1 + X(3)**2) / RHO**3*DUM)
PAM(2, 3, 3) = DUM1 / (RHO**3*DUM) + X(3)**2*DUM1 / (RHO**3*DUM)
PAM(2, 3, 4) = X(2)**2*DUM1 / (RHO**3*DUM)
PAM(2, 3, 5) = X(2)**2*DUM1 / (RHO**3*DUM)
PAM(3, 1, 1) = AM(3, 1) / (2.0*D0*X(1))
PAM(3, 1, 2) = PZ38*AM(3, 1) / CB - CB*(PZ21 + X(3)**2)*X1DOT/PZ5/EN
1 + X(3)**2) / (RHO*DUM)
PAM(3, 1, 3) = PZ39*AM(3, 1) / CB - CB*(PZ22 + (BETA*Y1DOT + X(3)
1 * Y1DOT*PZ5/EN + BETA*PZ14) / EN
PAM(3, 2, 1) = AM(3, 2) / (2.0*D0*X(1))
PAM(3, 2, 2) = PZ38*AM(3, 2) / CB - CB*(PZ30 + X(3)**2)*Y1DOT/PZ5
1 + X(3)**2)*Y1DOT/PZ5
PAM(3, 2, 3) = PZ39*AM(3, 2) / CB - CB*(PZ31 + BETA*Y1DOT + X(3)
1 * Y1DOT*PZ6 + X(3)**2)*BETA*PZ16) / EN
PAM(3, 3, 1) = AM(3, 3) / (2.0*D0*X(1))
PAM(3, 3, 2) = -DUM1 / (RHO*DUM) - X(2)**2) / (RHO**3*DUM)
PAM(3, 3, 3) = X(2)**2) / (RHO*DUM) - X(2)**2) / (RHO**3*DUM)
PAM(3, 3, 4) = X(2)**2) / (RHO*DUM)
PAM(3, 3, 5) = X(2)**2) / (RHO*DUM)
DUM = (1.0*D0 + X(5)**2 + X(4)**2) / (2.0*D0*DUM*RHO)
PZ40 = -Z5 / (2.0*D0*X(1))
PZ41 = X(2)**2*Z5/RHO**2
PZ42 = X(3)**2*Z5/RHO**2
PZ43 = X(4)**2*Z5/RHO**2
PZ44 = X(5)**2*Z5/RHO**2
PAM(4, 3, 1) = AM(4, 3) / (2.0*D0*X(1))
PAM(4, 3, 2) = PZ41*Y1 + Z5*PZ29
PAM(4, 3, 3) = PZ42*Y1 + Z5*PZ35
PAM(4, 3, 4) = PZ43*Y1
PAM(4, 3, 5) = PZ44*Y1
PAM(5, 3, 1) = AM(5, 3) / (2.0*D0*X(1))
PAM(5, 3, 2) = PZ41*X1 + Z5*PZ20
PAM(5, 3, 3) = PZ42*X1 + Z5*PZ26
PAM(5, 3, 4) = PZ43*X1
PAM(5, 3, 5) = PZ44*X1
DO 30 K = 1, 5
DO 30 I = 4, 5
DO 30 J = 1, 2
PAM(I, J, K) = 0.0*D0
SUBROUTINE QUAD(XL,XU,FCT,Y,Z,G,H,N)

C
C QUAD
C
C THIS IS A MODIFIED QUADRATURE PROGRAM FOR VECTOR VALUED FUNCTIONS.
C COMPUTES INTEGRAL OF THE FUNCTION G (OR H) OVER X FROM XL TO XU.
C THE RESULT IS Y. ROUTINE USES A 4 POINT GAUSS QUADRATURE.
C
C IMPLICIT DOUBLE PRECISION(A-H,O-Z), INTEGER (I-N)
DIMENSION Y(1),H(1),G(1),Z(1)
EXTERNAL FCT

A=.5D0*(XU+XL)
B=XU-XL
C=.43056815579702629D0*B
K=1
GO TO 50
10 DO 20 I=1,N
20 Y(I)=.17392742256872693D0*G(I)
C=.16999052179242813D0*B
K=2
GO TO 50
30 DO 40 I=1,N
40 Y(I)=B*(Y(I)+.32607257743127307D0*G(I))
RETURN
CALL FCT(A-C,A+C,Z,H,G)
DO 60 I=1,N
60 G(I)=G(I)+H(I)
GO TO (10,30), K
END
SUBROUTINE QTRAP(FUNC,A,B,S,N)
RETURNS S AS THE VECTOR INTEGRAL OF FUNCT OVER [A,B]
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER(_EPS=1.D-4, JMAX=20)
DIMENSION S(12),SOLDS(12),G(12),H(12),SUM(12)
EXTERNAL FUNC

DO 10 I=1,N
10 SOLDS(I)=-1.D30

DO 11 J=1, JMAX
M=J
IF (M .EQ. 1) THEN
CALL FUNC(A,B,H,G,M)
DO 12 K=1,N
12 S(K)=0.5D0*(B-A)*(G(K)+H(K))
IT=1
ELSE
ITNM=IT
DEL=(B-A)/ITNM
X=A+0.5D0*DEL
DO 13 K=1,N
13 SUM(K)=0.DO
DO 20 K=1, IT
CALL FUNC(X,B,H,G,M)
DO 25 I=1,N
25 SUM(I)=SUM(I)+H(I)
X = X + DEL

20 CONTINUE
DO 30 I = 1, N
30 S(I) = 0.5D0 * (S(I) + (B - A) * SUM(I) / ITNM)
   IT = 2 * IT
ENDIF
DO 35 I = 1, N
IF (DABS(S(I) - SOLDS(I)) .LT. EPS * DABS(SOLDS(I))) RETURN
SOLDS(I) = S(I)
35 CONTINUE
11 CONTINUE
IF (IT .GT. 50) PRINT*, 'TOO MANY STEPS'

END
For many optimal transfer problems it is reasonable to expect that the minimum time solution is also the minimum fuel solution. However, if one allows the propulsion system to be turned off and back on, it is clear that these two solutions may differ. In general, high thrust transfers resemble the well known impulsive transfers where the burn arcs are of very short duration. The low and medium thrust transfers differ in that their thrust acceleration levels yield longer burn arcs and thus will require more revolutions. In this research, we considered two approaches for solving this problem; a powered flight guidance algorithm previously developed for higher thrust transfers was modified and an "averaging technique" was investigated.