FINAL REPORT for NAG-1-2246
Control Law — Control Allocation Interaction

F/A-18 PA Simulation Test-Bed

Dr. Wayne Durham & Mark Nelson

Department of Aerospace & Ocean Engineering
Virginia Polytechnic Institute & State University
Blacksburg, Virginia 24061
1 Introduction

This report documents the first stage of research into Control Law — Control Allocation Interactions. A three-year research effort was originally proposed:

1. Create a desktop flight simulation environment under which experiments related to the open questions may be conducted.

2. Conduct research to determine which aspects of control allocation have impact upon control law design that merits further research.

3. Conduct research into those aspects of control allocation identified above, and their impacts upon control law design.

Simulation code was written utilizing the F/A-18 airframe in the power approach (PA) configuration. A dynamic inversion control law was implemented and used to drive a state-of-the-art control allocation subroutine.

2 Simulation

The airframe used was derived from the F/A-18 model already implemented in CASTLE. The airframe is not realistic, but is intended to be a test-bed for further research. The greatest area in which the test-bed simulation differs from the original airframe is in the treatment of control deflections. There are essentially two sets of control effectors:

1. The original control effectors in the F/A-18 airframe model. These are used only for initial trim and subsequent scheduling.
2. A duplicate set of control effectors that have linear effectiveness. This control set is the input to the control allocator, and the forces and moments they generate are superimposed on those of the bare airframe and original control set. Rate limits of the duplicate set are the no-load rate limits of the original controls. Position limits of the duplicate control effectors are referenced from the trim or scheduled positions of their counterparts in the original controls.

The rationale for incorporating a duplicate control set was to provide a constant, linear control effective matrix with flexibility for future variations and modifications. The control deflections are calculated for the trimmed/scheduled flight condition using the original F/A-18 nonlinear table lookups. The control deflections calculated from the allocator to produce desired moments use the control effectiveness matrix obtained from linearizing the F/A-18 aero database.

2.1 Simulation Code

There are six files that are used in the simulation of the airframe: Aero.f, Aeropa.f, Control.f, Constants.f, Engine.f, and Alloc.f.

2.1.1 Aero.f

The aero code first calls Aeropa.f to calculate the aerodynamics of the scheduled/trimmed flight condition. The code then combines the aerodynamics from the non-linear scheduled/trimmed flight condition and from the control deflections calculated in the allocator to produce the desired moments.
2.1.2 Aeropa.f

This code is taken from the F/A-18 simulation and modified slightly to include the control positions that are used in the table lookups later in the code. This is the only code that gives the airframe F/A-18 like characteristics. All added code is at the top of the RUN section.

2.1.3 Control.f

Stick and rudder pedal commands are taken as inputs and converted into an alpha command \(\alpha_{\text{cmd}}\), beta command \(\beta_{\text{cmd}}\), and a roll rate command \(p_{\text{cmd}}\). These commands are input to a simple dynamic inversion control law that generates desired moments for the control allocation subroutine. First, \(\alpha_{\text{cmd}}\) and \(\beta_{\text{cmd}}\) are converted to desired accelerations \(\dot{\alpha}_{\text{des}}\) and \(\dot{\beta}_{\text{des}}\):

\[
\begin{align*}
  \dot{\alpha}_{\text{des}} &= \lambda_\alpha (u - \dot{\alpha}_{\text{cmd}}) \\
  \dot{\beta}_{\text{des}} &= \lambda_\beta (v - \dot{\beta}_{\text{cmd}}) \\
  \dot{\alpha}_{\text{cmd}} &= V \sin \beta_{\text{cmd}} \\
  \dot{\beta}_{\text{cmd}} &= \lambda_\beta (v - \dot{\beta}_{\text{cmd}})
\end{align*}
\]

Next, \(\dot{\alpha}_{\text{des}}\) and \(\dot{\beta}_{\text{des}}\) are applied to inversions of the body-axis force equations (treating \(q\) and \(r\) as controls):

\[
\begin{align*}
  q_{\text{cmd}} &= \frac{\dot{\alpha}_{\text{des}} + pu + g \cos \theta \cos \phi - Z/m}{u} \\
  r_{\text{cmd}} &= \frac{-\dot{\beta}_{\text{des}} + pu + g \cos \theta \sin \phi + Y/m}{u}
\end{align*}
\]

These two inversions are made as perfect as possible by using actual aircraft states, and the last calculated values of the body-axis forces \(Y\) and \(Z\) from the
aerodynamic calculations. First-order responses are specified for the desired angular accelerations,

\[ \dot{\hat{p}}_{des} = \lambda_p (p - p_{cmd}) \]
\[ \dot{\hat{q}}_{des} = \lambda_q (q - q_{cmd}) \]
\[ \dot{\hat{r}}_{des} = \lambda_r (r - r_{cmd}) \]

Finally, the desired body-axis moments, required to obtain the desired accelerations, are calculated from inversions of the body-axis moment equations:

\[ C_{\ell_{des}} = -C_\ell + \frac{I_{\ell \ell_{des}} + I_{\ell r_{des}} + (I_{\ell z} - I_{\ell y}) \tau r - I_{\ell z} pq}{q S_b} \]
\[ C_{m_{des}} = -C_m + \frac{I_{\mu \mu_{des}} + (I_{\mu z} - I_{\mu y}) \tau r + I_{\mu z} (p^2 - r^2)}{q S_c} \]
\[ C_{n_{des}} = -C_n + \frac{-I_{n \mu_{des}} + I_{n r_{des}} + (I_{n x} - I_{n y}) pq + I_{n z} \tau}{q S_b} \]

The moment coefficients \( C_\ell, C_m, \) and \( C_n \) are the last calculated values of the body-axis moments. Since control-generated moments are superimposed on these values, they are the moments generated by the bare-airframe plus scheduled control deflections. The trimmed flight control deflections are used to calculate moments for the current flight condition to be used in the restoring algorithm. The attained moments are calculated next using the control deflections from the last time step for comparison purposes with the desired moments. The desired moments, along with the required inputs, are input to the allocator to produce the required control deflections. The last step is to check the control deflections against the limits and reset them accordingly.

2.1.4 Constants.f

This section of code sets the model specific constants.
2.1.5 Engine.f

The engine model is taken from the Stevens & Lewis F-16 model [1].

2.1.6 Alloc.f

This code is the control allocator that produces required control deflections for desired moments. This code is explained in detail later.

3 Desktop Simulation

The F/A-18 PA model was first implemented on the UNIX-based CASTLE. The conversion of the simulation to the desktop PC required the CASTLE offline help menu provided with the PC version of CASTLE. Some additional steps were taken to complete the compilation of the airframe. The steps are as follows:

1. The directory structure from UNIX was copied to the CASTLE airframes folder.

2. A project was created in Microsoft Studio 6.0 following the F/A-18 project already included with the PC version of CASTLE. All custom builds were set up in the same way the F/A-18 project had them set up\(^1\). The custom builds were implemented on `symbols.sdf` and all the FTP data files.

3. In `symbols.sdf` the realtime CDF section was changed to resemble the

\(^1\) The offline CASTLE help explains a different way of setting up custom builds, but did not work.
F/A-18 realtime CDF section in the corresponding symbols.sdf. The reason is just a difference in structure between the PC CASTLE and UNIX CASTLE.

4 Control Allocation Algorithm

4.1 Introduction

The control allocation algorithm is a FORTRAN implementation of the bi-secting, edge-searching algorithm. The theory behind the allocation code is explained in detail in [2]. Following is a step-by-step explanation of the code. Line numbers correspond to those in the attached file “Alloc.f”.

4.2 Subroutine DA3

4.2.1 Diagnostics

The sections of code that depend on the DIAGS flag are debugging tools that can be used to dump several relevant variables. Because a great volume of output is generated, the DIAGS flag should be used sparingly.

4.2.2 Code Description

**Lines 0126–0146**: Array CSPHI is a table of sines and cosines of angles, beginning at 45° and proceeding through 20 bisections.

**Lines 0191–0208**: The desired moments are checked for zero length, and a vector of zero control deflections returned if they are; otherwise the
vector is normalized.

**Lines 0210–0219:** The initial rotation is performed using the transformation generated by subroutine DCGEN to align the desired moment direction $y_{3_d}$ with the $y_1$ axis. Subroutine DCGEN is an implementation of the method described in [2, Section 5.1]. Lines 0212–0219 perform the matrix multiplication, $B_3 = TB_{3...}$. 

**Lines 0221–0231:** The controls that generate the moment with the maximum $y_1$ component are found by examining the sign of the first row of $B$ and setting the control to its maximum or minimum, depending on that sign. The controls are first set to ±1 (object notation) and then set to their actual limits by subroutine SETU.

**Lines 0233–0243:** This section of code was added to deal with the finite precision of computer math. The variable TOL is a distance in moment space that is related to the smallest bisection angle to be used, at the distance from the origin of the vertex just determined (maximum $y_1$ component). TOL is used in subsequent code to resolve near-zero numbers.

**Lines 0264–0265:** Subroutine R20 solves the 2-D problem for the projection onto the current $y_1$-$y_2$ plane. R20 returns the object-notation vector of controls of the intersecting edge in variable U1, the control that defines that edge in variable IU, and a ±1 value in variable INFRONT that is +1 if the edge is in front of, and −1 if it is behind the $y_1$-$y_2$ plane.

The three variables TEMP2, TEMP3, and TEMP4 contain respectively...
the sorted list of controls (ITHETA) with an additional zero between the two controls at the ends of the intersecting edge, the number of vertices in the list (NANGS), and the index of the position in the list of the additional zero (INDX). Finally, the logical variable ISVERTEX signals that the desired moment points directly at a vertex.

**Lines 0274–0293:** This section of code has no counterpart in reference [2]. It was added during debugging and found to improve the success rate of the algorithm (decrease the number of estimations required). The most recently found edges that were in front (Last In Front, LIF) and behind (Last In Back, LIB) are saved. Theoretically the last two edges found will be LIF and LIB, but in some cases they were not.

**Lines 0295–0299:** If R20 reports a vertex in ISVERTEX, the controls that determine that vertex, and the saturation of the desired moment, are calculated by a call to DOVERT, and the subroutine is exited.

**Lines 0304–0322:** This section of code initializes several variables, including the rotation matrix T22.

**Lines 0333–0510:** This is the main loop, in which the 2-D problem is repeatedly solved for different rotations about the \( y_1 \) axis.

**Lines 0335–0342:** Used during debugging, retained for possible future use.

**Lines 0344–0349:** Rotation about \( y_1 \). B1 is the operative \( B \) matrix throughout. Code performs operation \( T \cdot B \).
Lines 0360–0364: The last returned values of ITHETA, NANGS, and INDX are assigned to those variables to be saved when TEMP2, TEMP3, and TEMP4 are overwritten by R20.

Lines 0366–0367: Call to R20 to solve the 2-D problem for the current orientation of B1 about the y1-axis.

Lines 0376–0395: The edge identified by R20 is assigned to LIF or LIB according to the sign of the variable INFRONT.

Lines 0397–0401: Another vertex check.

Lines 0411–0495: Executed when the most recent and the prior edges differ in sign of their y3 component, as indicated by the variables INFRONT and WASINF. This section of code is the implementation of the description given in [2, Section 5.3]. Through line 0436 the code is doing housekeeping and (possibly) diagnostics.

Lines 0438–0457: This section reflects a subtlety in the implementation of the algorithm not described in [2]. The prior edge was identified using a different B matrix than the most recent edge. All relevant information regarding the prior edge is contained in the saved variables ITHETA, NANGS, and INDX. At lines 0456–0457 a call is made to subroutine GETEDGE, which is also called as the last step of subroutine R20.

Lines 0486–0493: Check the last two edges identified to see if they comprise the solution facet. If they do not, the LIF and LIB edges are checked. Both sets of edges are checked using subroutine ISFACET, described below. Output from ISFACET consists of the logical ISOK, numbers of the two defining controls in IUOUT and JUOUT, and controls (in object notation) at three vertices of the facet as columns of the array U123.

Lines 0518–0519: If the variable ISOK is false, the correct facet has not been determined and the maximum number of bisections has been performed. One last check of LIF and LIB is performed.

Lines 0520–0574: If ISOK is true, the solution is calculated. Otherwise (lines 0572–0573) the solution is estimated.

Lines 0521–0565: A straightforward implementation of [2, Equations (13) and (14)]. M123 is the matrix $\tilde{e}_{3,1} (v_1^y - v_2^y) (v_1^y - v_3^y)]$ in [2, Equation (13)]; variables AA, BB, and CC correspond to $\alpha_3$, $C_{1,2}$, and $C_{1,3}$ respectively; and MTEMP is $v_1^y$. The variable UDA is the same as $u^a$ in [2, Equation (14)], except that it has been scaled as necessary.

Lines 0572–0573: The estimator is called.
4.3 Subroutine DCGEN

This subroutine is a straightforward implementation of the initial transformation algorithm described in [2, Section 5.1].

**Lines 0789–0798:** The desired moments are normalized using double precision math.

**Lines 0810–0824:** If one or more of the leading components of the normalized moment vector are zero, the size of the problem is reduced.

**Lines 0829–0833:** The first row of the transformation matrix is set to the normalized desired moments.

**Lines 0837–0850:** The remaining terms are calculated in the three nested do-loops in [2, Equation (4)].

**Lines 0858–0868:** The last section of DCGEN ensures that the determinant of the transformation matrix is +1.

4.4 Subroutine R20

To find the edge that the desired moments direction is pointing to, the subroutine R20 is implemented. The theory behind this subroutine is in [2, Section 5.2.2]. All calculations in this subroutine are done in the $y_1y_2$ plane.

**Lines 0928–0961:** The $y_2$ component of the point with the maximum $y_1$ component (UMAX in object notation, XUMAX in control notation) is calculated to determine its sign. The desired moment is checked to see
if its direction points towards a vertex of the attainable moment subset.
If it is a vertex the subroutine is exited and the allocation carries on.

**Lines 0976–0992:** Implementation of [2, Step 1, Page 20]. The array THETA is the needed part of the set $L_{\phi}$, and ITHETA that of $L_u$. Once the angle is found, $\pi$ is added or subtracted from it if the absolute value is greater than $\pi/2$ and depending on the sign of the angle. In this way, the angles of just the vertices with positive $y_1$ components are generated.

**Lines 0998–1010:** The angles are sorted in a clockwise or counterclockwise manner starting with the vertex that has the largest $y_2$ component. The manner in which they are sorted depends on the sign of the $y_2$ component of the maximum vertex, recorded in SY.

**Lines 1013–1025:** A zero is inserted in THETA and ITHETA to mark the point at which the angle changes sign.

**Lines 1034–1036:** THETA, ITHETA, and NANGS (the number of angles generated) are sent to subroutine GETEDGE to finish the solution to the 2-D problem. Subroutine GETEDGE is provided separately so that it could be called independently from DA3, as described above.

### 4.5 Subroutine GETEDGE

This subroutine is part of the explanation in [2, Section 5.2.2].

**Lines 1090–1127:** The first loop in this subroutine is looking for a sign change in the $y_2$ component between ordered vertices. Since the vertices
were sorted in the manner described, the solution edge will be the first one encountered in traversing the edges starting with the first vertex. The list is stepped through in the proper direction by the index $\text{IX} = \text{IX} + \text{SY}$.

The previous $y_2$ value is stored before the next $y_2$ value is calculated. This new value is compared to the previous one determining whether the edge crosses the $y_1$ axis. If the do loop continues, $U_2$ is set to the next vertex by changing the sign of the control that is defining the current edge. The index is updated accordingly with the sign of $y_2$ and the process starts again until the edge is found. The do loop is exited when a new point is found that has a different sign than the point before.

**Lines 1129–1181:** This section deals with possible failure of the previous loop to find an edge, as indicated by $(\text{SY.EQ.SSY})$. The starting values of relevant variables are restored, and the vertex list is traversed in the opposite direction. The first loop should always find the proper edge when \texttt{GETEDGE} is called from \texttt{R20}, but the first loop may fail when called from within \texttt{DA3}. The list is traversed in the opposite direction by the index $\text{IX} = \text{IX} + \text{SY}$. Implementation of this section of code was the reason for inserting a zero in the ordered list of vertices.

**Lines 1190–1200:** One or the other of the previous two loops will have identified $U_2$ (a vertex in object notation) and $JU$ (the number of the control that defines the edge). $U_2$ is converted to control notation using the subroutine \texttt{SETU}. The third row of the $B$ matrix is applied to the two
vertices that define the solution edge to determine the \( y_3 \) component in moment space at the point where the \( y_2 \) component of the edge is zero. If the \( y_3 \) component is positive, the edge is described as "in front", whereas if the \( y_3 \) component is negative, the edge is "behind" the line defined by the direction of the desired moments \( \ell_3 \).

**Lines 1202–1217:** A final vertex check is made and the subroutine is exited.

### 4.6 Subroutine DOVERT

**Lines 693–728:** If it was determined that the desired moments points directly to a vertex the subroutine **DOVERT** is called. **DOVERT** uses the maximum or minimum controls that make up the vertex and calculates the total moment from there, scaling it appropriately. The allocator subroutine is then exited and the simulation carries on. This case is rare during simulation, but may occur.

### 4.7 Subroutine EST

The theory behind the estimator subroutine is explained in [2, Section 5.4.2].

**Lines 0604–629** The subroutine starts with the last two edges that the allocator had found and creates a facet by setting the appropriate control to \(-1\) or \(+1\). **SETU** is used to assign actual control limits to these points which are then put into moment space using the control effectiveness matrix.

**Lines 631–669** An interpolation is then made with the estimated facet vertices to determine the solution.
The moments are calculated using the estimated control positions and then scaled with the saturation limits.

4.8 Subroutine ISFACET

The subroutine is used to test the facet found by DA3. The subroutine uses the two defining controls from DA3 to find a facet from scratch that these two controls define. This algorithm is the subject of reference [4].

**Lines 1236–1251:** Zeros are set in the appropriate positions of the vertex arrays so that two edges are defined for the facet. The dimension of the union (see [2, Section 4.2]) of the two edges is determined. If the union is not two dimensional, then the edges can not form a facet; ISOK is set to false and the subroutine is exited.

**Lines 1253–1320:** For the two dimensional case the routine begins to calculate from scratch the facet that is determined by the two defining controls. The method used is completely independent of the edge-searching method and is explained in [4].

**Lines 1255–1287:** This section of code was lifted from earlier FORTRAN implementations of the facet-searching allocation method described in reference [4]. The facet defined by the two controls is in the variable TESTFACET.

**Lines 1291–1311:** The facet TESTFACET is compared with the object OBJ that was generated by R20. If they are different, the
facet opposite TESTFACET (also generated by the same two controls) is tested (lines 1300-1311).

**Lines 1322–1336:** If the facet just found is the same facet as the one that was found from the allocator, then U123, which is the matrix whose columns correspond to controls that generate three of the vertices that make up the solution facet, is assembled and returned.

### 4.9 Miscellaneous Subroutines

#### 4.9.1 MINNORM

The purpose of the minimum-norm restoring solution is to keep the controls as close to their trimmed control position as possible. The usual minimum-norm solution keeps the controls as close to zero as possible, however, in this application the zero position is redefined as the trimmed/scheduled control positions.

**Lines 1496–1531:** The subroutine is started by finding the total control position for the current time step and calculating the total attained moment. **Lines 1533–1554:** If the control limits are zero, the routine is returned and no restoring takes place. Otherwise, the difference between the pseudo-inverse solution redefined at the trim condition, and the controls given by the allocation routine are used to find a delta control position that will drive the controls towards the trimmed position. This delta control position is scaled according to the control limits and a new restored control position is returned.

For more information on control restoring, refer to Bolling. [3, Ch. 4]
4.9.2 SORTC

Lines 1351–1492: A sorting subroutine downloaded from the National Institute of Standards and Technology (NIST) GAMS (Guide to Available Mathematical Software) at http://gams.nist.gov/. This particular algorithm was chosen for its efficiency, and for the fact that it returns a sorted index vector along with the sorted vector.

4.9.3 INVMAT3

Lines 0740–767: A brute force matrix inversion subroutine. Good only for $3 \times 3$ matrices.

5 Verification Data

Sample runs are included to verify the airframe. The four tests cases used are a trimmed flight condition, a step in the longitudinal stick, and step doublets in the lateral stick and rudder pedals. The MANGEN command in CASTLE was implemented to produce the desired stick commands. Complete MATLAB flies of the four cases are attached as trim_dec11.mat, long_dec11.mat, lat_dec11.mat, and dir_dec11.mat.

The plots include selected states of the airframe along with the trimmed/scheduled control positions and the allocated control positions.

Figure 1 shows the time histories of the six global controls in a trimmed flight condition at 8.1 degrees angle of attack, 1200 ft, and $231.52 \frac{ft}{sec}$. These settings
are the default when the airframe is loaded. Some settling of the controls to achieve steady state is noted.

Figure 2 shows time histories for a step input in longitudinal stick of 2.5 inches aft from center. The airframe was initialized to the trim conditions described above and the stick step implemented at time $= 1$sec for 1 second.

Figure 3 shows the time histories for a step douplet in lateral stick. The lateral stick was driven right 2 inches from center at time $= 1$sec for 1 second and then left of center 2 inches for 1 second.

Figure 4 shows the time histories for a step douplet in rudder pedals. The pedals were driven right 2 inches from center at time $= 1$sec for 1 second and then left of center 2 inches for 1 second.
Figure 1: Global Control Deflections, Trimmed Flight (Degrees)
Figure 2: Longitudinal Stick Step Input
Figure 3: Lateral Stick Step Doublet
Figure 4: Lateral Rudder Pedal Step Doublet
References


C*******************************************************************************
C TITLE: DA3
C
C FUNCTION: 3 Moment Control Allocator
C Direct Allocation for the 3 objective problem
C using bisecting edge searching algorithm
C
C DESIGNED BY: Bull Durham
C CODED BY: Kevin Scalera
C MAINTAINED BY: VPI SIMULATIONS
C
C MODIFICATION HISTORY:
C
C GLOSSARY
C
C ASSIGNMENTS:
C NONE
C
C INPUTS:
C IMODE Sim. mode: -2=init,-1=reset,0=hold,1=ru
C
C OUTPUTS:
C NONE
C
C LOCALS:
C NONE
C
C OTHER LOCALS:
SUBROUTINE DA3(UDA, SAT, IERR, & B, MDES, U_MIN, U_MAX, M, NBI, TIME, DIAGS)

C DECLARATION SECTION

C IMPLICIT NONE

** Parameters

INPUTS:
INTEGER*4 IMODE

OTHER LOCALS:
BYTE CONPAR, CTLBUF
LOGICAL*4 DIAGS, DIDSWITCH, INITIALIZED, ISOK, ISVERTEX, STUCK
INTEGER*4 I, I_COUNT, IERR, INDX, I_THETA(21), IU, IUOUT
INTEGER*4 IU_TEMP(20), I_LIB, I_LIF, J, JU, JUOUT, K, M
INTEGER*4 MAXSTEPS, NA_NGS, NBI, NMu_X, STEPS, SY, TEMP2(21)
INTEGER*4 TEMP3, TEMP4, U1(20), U123(20,3), U2(20), U_MAX(20)
INTEGER*4 U_LIB(20), U_LIF(20)
REAL*4 MINV(3,3), MTEMP(3), MXMAX(3), COSPHI, CSPHI(2,20)
REAL*4 ABC(3), NORM, PI, SAT, SINPHI, DET, AA, INFRONT, T(3,3)
REAL*4 T22(2,2), BB, BTEMP(2), CC, TIME, TOL, TOLANG, B(3,20)
REAL*4 M123(3,3), MAXNORM, UDA(20), B1(3,20), MD(3), MDES(3)
REAL*4 U_MAX(20), U_MIN(20), WASINF, XU123(20,3), XUMAX(20)
REAL*4 XUTEMP(20), Y

C COMMON SECTION

COMMON/ SHELL1 / CONPAR(424)

C EQUIVALENCE SECTION

EQUIVALENCE( CONPAR(1), IMODE)

** Inputs:
DATA SECTION

DATA INITIALIZED/.FALSE./
DATA PI/3.141592653589793/

Table of cosines and sines of bisection angle

DATA CSPHI/

& 7.071067811865475e-01, 7.071067811865476e-01,
& 9.238795325112867e-01, 3.826834323650898e-01,
& 9.807852804032304e-01, 1.950903220161282e-01,
& 9.951847266721969e-01, 9.801714032956060e-02,
& 9.987954562051724e-01, 4.906767432741801e-02,
& 9.996988186962042e-01, 2.454122852291229e-02,
& 9.999247018391445e-01, 1.227153828571993e-02,
& 9.999811752826011e-01, 6.135884649154475e-03,
& 9.999952938095762e-01, 3.067956762965976e-03,
& 9.999988234517019e-01, 1.533980186284766e-03,
& 9.999997058628822e-01, 7.669903187427045e-04,
& 9.999999264657179e-01, 3.834951875713956e-04,
& 9.999999981614293e-01, 1.917475973107033e-04,
& 9.999999999041073e-01, 9.587379909597734e-05,
& 9.9999999998510269e-01, 4.793689960306688e-05,
& 9.99999999997127567e-01, 2.396844980841822e-05,
& 9.99999999999281892e-01, 1.198422490506971e-05,
& 9.999999999999820472e-01, 5.992112452642428e-06,
& 9.99999999999995118e-01, 2.996056226334661e-06,
& 9.999999999999988780e-01, 1.498028113169011e-06/

INITIALIZATION SECTION

RESET SECTION

RUN SECTION
IF (DIAGS) THEN
    WRITE(*, '(A50)') 'Entering DA3'

    WRITE(*, '(A50)') 'Calling arguments'

    WRITE(*, '(A50,6E13.6)') ' *DA4* B(1,:) = ', (B(1,I), I=1,M)
    WRITE(*, '(A50,6E13.6)') ' *DA4* B(2,:) = ', (B(2,I), I=1,M)
    WRITE(*, '(A50,6E13.6)') ' *DA4* B(3,:) = ', (B(3,I), I=1,M)

    WRITE(*, '(A50,6E13.6)') ' *DA5* MDES = ', (MDES(I), I=1,M)
    WRITE(*, '(A50,6E13.6)') ' *DA5* U_MIN = ', (U_MIN(I), I=1,M)
    WRITE(*, '(A50,6E13.6)') ' *DA5* U_MAX = ', (U_MAX(I), I=1,M)

    WRITE(*, '(A50,13)') ' *DA6* M = ', M

    WRITE(*, '(A50,13)') ' *DA7* NBI = ', NBI

    WRITE(*, '(A50,13)') ' *DA8* TIME = ', TIME

ENDIF

INFRONT = 1.0

NORM = 0.0

DO I = 1,3
    NORM = NORM + MDES(I)*MDES(I)
ENDDO

DO I = 1,3
    MD(I) = MDES(I)/NORM
ENDDO

CALL DCGEN(T, MD)

DO I = 1,3
    DO J = I,M
        BI(I,J) = 0.0
        DO K = 1,3
            BI(I,J) = BI(I,J) + T(I,K)*B(K,J)
        ENDDO
    ENDDO
ENDDO

DO I = 1,M
    IF (BI(I,I) EQ 0.0) THEN
        UMAX(I) = 0
    ELSEIF (BI(I,I) LT 0.0) THEN
        UMAX(I) = -1
    ELSE
        UMAX(I) = 1
    ENDIF
ENDDO
ENDDO
C
CALL SETU(XUMAX, UMAX, U_MIN, U_MAX, M)
C
TOLANG = CSPI(2, MIN(20, 2*NBI))
DO I=1,3
    MXMAX(I) = 0.
    DO J=I,M  
        MXMAX(I) = MXMAX(I) + BI(I, J) * XUMAX(J)
    ENDDO
ENDDO
MAXNORM = SQRT(MXMAX(1) * MXMAX(1)
& +MXMAX(2) * MXMAX(2)
& +MXMAX(3) * MXMAX(3))
TOL = MAXNORM * TOLANG

C
IF (DIAGS) THEN
    WRITE(*, '(ASC)') ' Preliminary Calcs'
    WRITE(*, '(A30, E18.6)') 'DA* NORM = ', NORM
    WRITE(*, '(A30, F14.6)') 'DA* MD = ', (MD(I), I=1,3)
    WRITE(*, '(A30, F14.6)') 'DA* T(1,:) = ', (T(1, I), I=1,3)
    WRITE(*, '(A30, F14.6)') 'DA* T(2,:) = ', (T(2, I), I=1,3)
    WRITE(*, '(A30, F14.6)') 'DA* T(3,:) = ', (T(3, I), I=1,3)
    WRITE(*, '(A30, E18.6)') 'DA* B1(1,:) = ', (B1(1, I), I=1, M)
    WRITE(*, '(A30, E18.6)') 'DA* B1(2,:) = ', (B1(2, I), I=1, M)
    WRITE(*, '(A30, E18.6)') 'DA* B1(3,:) = ', (B1(3, I), I=1, M)
    WRITE(*, '(A30, E18.6)') 'DA* UMAX = ', (UMAX(I), I=1, M)
    WRITE(*, '(A30, E18.6)') 'DA* XUMAX = ', (XUMAX(I), I=1, M)
    WRITE(*, '(A30, E18.6)') 'DA* TOLANG = ', TOLANG
    WRITE(*, '(A30, E18.6)') 'DA* MMXXAX = ', (MMAX(I), I=1,3)
    WRITE(*, '(A30, E18.6)') 'DA* MXXNORM = ', MAXNORM
    WRITE(*, '(A30, E18.6)') 'DA* TOL = ', TOL
    WRITE(*, '(A30)') ' First call to R20'
ENDIF
C
CALL R20(UI, IU, INFRONT, TEMP2, TEMP3, TEMP4, ISVERTEX,
& B1, UMAX, UMAX, U_MIN, U_MAX, TOL, M, DIAGS)
C
IF (DIAGS) THEN
    WRITE(*, '(ASC)') ' After 1st R20'
    WRITE(*, '(A30, F13.6)') 'DA* TEMP2 (ITHETA) = ', (TEMP2(I), I=1, TEMP3+1)
    WRITE(*, '(A30, F13.6)') 'DA* TEMP3 (INDX) = ', TEMP3
    WRITE(*, '(A30, F13.6)') 'DA* TEMP4 (INDX) = ', TEMP4
ENDIF
C
IF (IU.NE.0) THEN
    IF (INFRONT.EQ.1.) THEN
        DO I=1, M
            U_LIF(I) = UI(I)
        ENDDO
        I_LIF = IU
    ELSEIF (INFRONT.EQ.-1.) THEN
        DO I=1, M
            U_LIB(I) = UI(I)
        ENDDO
        I_LIB = IU
    ENDIF
ENDIF

IF (DIAGS) THEN
  WRITE(*, '(/AS0)') ' After 1st LIF/LIB'
ENDIF

IF (ISVERTEX) THEN
  WRITE(*,*) TIME = ', TIME, ' FIRST CALL TO R20'
  CALL DOVERT(UDA,SAT,U1,B,U_MIN,U_MAX,M,NORM)
  RETURN
ENDIF

C 1st rotation about x-axis
IF (M.GE.8) THEN
  ICOUNT = 1
ELSE
  ICOUNT = 2
ENDIF
COSPHI = CSPHI(1,ICOUNT)
SINPHI = INFRONT*CSPHI(2,ICOUNT)
T22(I,1) = COSPHI
T22(I,2) = -SINPHI
T22(2,1) = SINPHI
T22(2,2) = COSPHI
MAXSTEPS = 2*INT(ABS(PI/ASIN(SINPHI)))
WASINF = INFRONT
ISOK = .FALSE.
NMAX = NBI + 1
DIDSWITCH = .FALSE.
STEPS = 0
STUCK = .FALSE.
IF (DIAGS) THEN
  WRITE(*, '(/AS0)') ' Before Main Loop'
  WRITE(*, '(/A30,015)') 'DA3* COSPHI = ', COSPHI
  WRITE(*, '(/A30,E18.6)') 'DA3* SINPHI = ', SINPHI
  WRITE(*, '(/A30,E18.6)') 'DA3* MAXSTEPS = ', MAXSTEPS
ENDIF

C MAIN LOOP ***********************************************
DO WHILE ((ICOUNT.LT.NMAX).AND. (.NOT.ISOK))
  STEPS = STEPS+1
  IF (STEPS.GE.MAXSTEPS) THEN
    STUCK = .TRUE.
  ELSE
    WRITE(*, '/A30') '************
    WRITE(*, '/A30,L3') 'DA3* STUCK = ', STUCK
    WRITE(*, '/A30,F14.6') 'DA3* TIME = ', TIME
    ENDIF
  ENDIF

ENDIF
DO J = 1,M
  BTEMP(1) = T22(1,1)*B1(2,J) + T22(1,2)*B1(3,J)
  BTEMP(2) = T22(2,1)*B1(2,J) + T22(2,2)*B1(3,J)
  B1(2,J) = BTEMP(1)
  B1(3,J) = BTEMP(2)
ENDDO

IF (DIAGS) THEN
  WRITE(*, '/AS01') ' In DA3 DOWHILE'
  WRITE(*, '/A30,13') ' *DA3* IOCOUNT = ', IOCOUNT
  WRITE(*, '/A30,13') ' *DA3* STEPS = ', STEPS
  WRITE(*, '/A30,13') ' *DA3* MAXSTEPS = ', MAXSTEPS
  WRITE(*, '/A30,43') ' *DA3* B1(2,:) = ', (B1(2,I),I=1,M)
  WRITE(*, '/A30,43') ' *DA3* B1(3,:) = ', (B1(3,I),I=1,M)
ENIF

NANGS = TEMP3
INDX = TEMP4
DO I=1,21
  ITHETA(I) = TEMP2(I)
ENDDO

CALL R20(UI,IU, INFRONT TEMP2,TEMP3,TEMP4,ISVERTEX,
& B1,UMAX,XUMAX,U_MIN,U_MAX,TOL,M,DIAGS)

IF (DIAGS) THEN
  WRITE(*, '/A30') ' After Loop R20'
  WRITE(*, '/A30,13') ' *DA3* TEMP2 (ITHETA) = ', (TEMP2(I), I=1,TEMP3+1)
  WRITE(*, '/A30,13') ' *DA3* TEMP3 (NANGS) = ', TEMP3
  WRITE(*, '/A30,43') ' *DA3* TEMP4 (INDX) = ', TEMP4
ENIF

IF (IU.NE.0) THEN
  IF (INFRONT.EQ.1.) THEN
    DO I=1,M
      U_LIF(I) = U1(I)
    ENDDO
    I_LIF = IU
  ELSEIF (INFRONT.EQ.-1.) THEN
    DO I=1,M
      U_LIB(I) = UI(I)
    ENDDO
    I_LIB = IU
  ENDIF
  IF (DIAGS) THEN
    WRITE(*, '/A30') ' After Loop R20 LIF_LIB'
    WRITE(*, '/A30,61') ' *DA3* U_LIF = ', (U_LIF(I), I=1,M)
    WRITE(*, '/A30,61') ' *DA3* I_LIF = ', I_LIF
    WRITE(*, '/A30,61') ' *DA3* U_LIB = ', (U_LIB(I), I=1,M)
    WRITE(*, '/A30,61') ' *DA3* I_LIB = ', I_LIB
  ENIF
  ENIF

IF (ISVERTEX) THEN
  WRITE(*, '*') ' TIME = ', TIME, ' LOOP CALL TO R20'
  CALL DOVERT(UDA,SAT,UI,B1,U_MIN,U_MAX,TOL,M,DIAGS)
RETURN
ENDIF
ENDIF
IF (DIAGS) THEN
  WRITE(*,'(/ASC0:\') ' Before testing reversal'
  WRITE(*,'(A30,18.6)) ' *CA3* INFRONT = ', INFRONT
  WRITE(*,'(A30,18.6)) ' *CA3* WASINF = ', WASINF
ENDIF

DIDSWITCH = .FALSE.
IF (INFRONT.NE.WASINF) THEN
  REVERSE DIRECTION
  CIDSWITCH = .FALSE.
  IF (INFRONT.NE.WASINF) THEN
    WRITE(*,'(/ASC0:\') ' Reversing'
    WRITE(*,'(A3,3,i3)') ' *SA3* Stems _aken :
    WRITE(*,'(A3,13)') ' *DA3* Angle = ', 180.*ASIN(SINPHI)/PI
    WRITE(*,'(A3,8)') ' *DA3* MAXSTEPS = ', MAXSTEPS
    WRITE(*,'(A3,8)') ' *DA3* STUCK = ', STUCK
  ENDIF
  CIDSWITCH = .TRUE.
  WASINF = INFRONT
  ICOUNT = ICOUNT+1
ENDIF

C Bisection and next transformation
COSPHI = CSPHI(1,ICOUNT)
SINPHI = INFRONT*CSPHI(2,ICOUNT)
T22(I,1) = COSPHI
T22(I,2) = -SINPHI
T22(2,1) = SINPHI
T22(2,2) = COSPHI
MAXSTEPS = 2*INT(ABS(PI/ASIN(SINPHI)))
IF (DIAGS) THEN
  WRITE(*, '(/ASC0:\') ' Bisection and next transformation'
  WRITE(*, ' (A30,18.6)) ' *DA3* COSPHI = ', COSPHI
  WRITE(*, ' (A30,18.6)) ' *DA3* SINPHI = ', SINPHI
  WRITE(*, ' (A30,18.6)) ' *DA3* MAXSTEPS = ', MAXSTEPS
ENDIF

STEPS = 0
Y = 0.0
DO I = I,M
  Y = Y + BI(2,1)*XUMAX(I)
ENDDO
SY = 1
IF (Y.LT.0.0) SY = -1
IF (DIAGS) THEN
  WRITE(*, '(/ASC0:\') ' Before GETEDGE'
  WRITE(*, '(A30,6E18.6)) ' *DA3* BI(2,:) = ', (BI(2,I),I=1,M)
  WRITE(*, '(A30,6E18.6)) ' *DA3* BI(3,:) = ', (BI(3,I),I=1,M)
  WRITE(*, '(A30,6E18.6)) ' *DA3* XMAX = ', (UMAX(I),I=1,M)
  WRITE(*, '(A30,6E18.6)) ' *DA3* Y = ', Y
  WRITE(*, '(A30,13)') ' *DA3* SY = ', SY
  WRITE(*, ' (A30,13)') ' *DA3* ITHETA = ', (ITHETA(I),I=1,NANGS+1)
  WRITE(*, ' (A30,13)') ' *DA3* NANGS = ', NANGS
ENDIF
CALL GETEDGE(U2, JU, INFRONT, ISVERTEX,
& B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDX, TOL, M, DIAGS)

IF (JU.NE.0) THEN
  IF (INFRONT.EQ.1.) THEN
    DO I=1,M
      U_LIF(I) = U2(I)
    ENDDO
    I_LIF = JU
  ELSEIF (INFRONT.EQ.-1.) THEN
    DO I=1,M
      U_LIB(I) = U2(I)
    ENDDO
    I_LIB = JU
  ENDIF
  IF (DIAGS) THEN
    WRITE(*,'(/Af0! ') 'After SETEDGE LIF/LIB'
    WRITE(*,'(A30,6I3)') 'U_LIF = ', (U_LIF(I), I=I,M)
    WRITE(*,'(A30,13)') 'U_LIB = ', I_LIB
    WRITE(*,'(A30,6I3)') 'U_LIB = ', (U_LIB(I), I=I,M)
    WRITE(*,'(A30,23)') 'I_LIB = ', I_LIB
  ENDIF
ENDIF

IF (ISVERTEX) THEN
  WRITE(*,*) 'TIME = ', TIME, ' POL GETEDGE'
  CALL DOVERT(UDA,SAT,U2,B,U_MIN,U_MAX,M,NORM)
  RETURN
ENDIF

IF (JU.NE.0) THEN
  CALL ISFACET(ISOK, IUOUT, JUOUT, U123, &
  IU, JU, U1, U2, B1, M, TOL)
  IF (.NOT.ISOK) CALL ISFACET(ISOK, IUOUT, JUOUT, U123, &
  I_LIF, I_LIB, U_LIF, U_LIB, B1, M, TOL)
ELSE
  ISOK = .FALSE.
ENDIF
ENDIF  ! IF (INFRONT.NE.WASINF) THEN

ENDO  ! End of do while statement

END MAIN LOOP ********************************
IF (DIAGS) THEN
  WRITE(*,'(A)') ' Exited from EAB'
ENDIF

IF (.NOT.ISOK) CALL ISFACET(ISOK, IUOUT, JUOUT, U123,
  & I_LIF, I_LIB, U_LIF, U_LIB, B1, M, TOL)

IF (ISOK) THEN
  DO I=1,3
    DO J=1,M
      IUTEMP(J) = U123(J,I)
    ENDDO
    CALL SETU(XUTEMP, IUTEMP, U_MIN, U_MAX, M)
    DO J=1,M
      XU123(J,I) = XUTEMP(J)
    ENDDO
  ENDDO

DO I=1,3
  DO J=1,3
    M123(I,J) = 0.
  DO K=1,M
    M123(I,J) = M123(I,J) + B(I,K) * XU123(K,J)
  ENDDO
  ENDDO
  ENDDO

DO I=1,3
  DO J=2,3
    M123(I,J) = M123(I,I) - M123(I,J)
  ENDDO
  MTEMP(I) = M123(I,I)
  M123(I,I) = MDES(I)
ENDDO

CALL INVMAT3(M123, MINV, DET)

DO I=1,3
  ABC(I) = 0.
  DO J=1,3
    ABC(I) = ABC(I) + MINV(I,J) * MTEMP(J)
  ENDDO
ENDDO

AA = ABC(1)
BB = ABC(2)
CC = ABC(3)
SAT = 1./AA

IF (AA.LT.1.) AA = 1.

DO I=1,M
  UDA(I) = (XU123(I,1) & +BB*(XU123(I,2) - XU123(I,1)) & +CC*(XU123(I,3) - XU123(I,1)))/AA
ENDDO

IERR = 0

CALL estimate subroutine to estimate solution if facet not found
ELSE
CALL EST(UDA, SAT, IERR,
& U_LIF, I_LIF, U_LIB, I_LIB, B1, U_MIN, U_MAX, NORM, M)
ENDIF
RETURN
END

SUBROUTINE EST(UDA, SAT, IERR,
& U1, IU, U2, JU, B, U_MIN, U_MAX, NORM, M)
IMPLICIT NONE

C INPUTS
REAL*4 B(3,20), U_MAX(20), U_MIN(20), NORM
INTEGER*4 UI(20), IU, U2(20), JU, M

C OUTPUTS
REAL*4 SAT, UDA(20)
INTEGER*4 IERR

C LOCALS
REAL*4 XUI(20), XU2(20), XU3(20), XU4(20), XMOM(3)
REAL*4 UPPER1(3), UPPER2(3), LOWER1(3), LOWER2(3), XNORM
REAL*4 XKI, XK2, XK3, XVI(3), XV2(3), XW2(20)
INTEGER*4 U3(20), U4(20)

C OTHER LOCALS
INTEGER*4 I, J, K

IERR = 1

U1(IU) = -1
U2(JU) = -1
DO I=1,M
U3(I) = U1(I)
U4(I) = U2(I)
ENDDO
U3(IU) = 1
U4(JU) = 1

CALL SETU(XU1, U1, U_MIN, U_MAX, M)
CALL SETU(XU2, U2, U_MIN, U_MAX, M)
CALL SETU(XU3, U3, U_MIN, U_MAX, M)
CALL SETU(XU4, U4, U_MIN, U_MAX, M)
DO I=1,3
LOWER1(I) = 0.
LOWER2(I) = 0.
UPPER1(I) = 0.
UPPER2(I) = 0.
DO J=1,M
LOWER1(I) = LOWER1(I) + B(I,J) * XU1(J)
LOWER2(I) = LOWER2(I) + B(I,J) * XU2(J)
UPPER1(I) = UPPER1(I) + B(I,J) * XU3(J)
UPPER2(I) = UPPER2(I) + B(I,J) * XU4(J)
IF (LOWER1(2).NE.UPPER1(2)) THEN
  XK1 = LOWER1(2)/(LOWER1(2) - UPPER1(2))
ELSE
  XK1 = 0.
ENDIF

IF (LOWER2(2).NE.UPPER2(2)) THEN
  XK2 = LOWER2(2)/(LOWER2(2) - UPPER2(2))
ELSE
  XK2 = 0.
ENDIF

DO I = i, 3
  X VI(I) = XK1*UPPER1(I) + (1. - XK1)*LOWER1(I)
  XV2(I) = XK2*UPPER2(I) + (1. - XK2)*LOWER2(I)
ENDDO

IF (XV2(3).NE.XVI(3)) THEN
  XK3 = XV2(3)/(XV2(3) - XV1(3))
ELSE
  XK3 = 0.
ENDIF

DO I = I, M
  X WI(I) = XK1*XU1(I) + (1. - XK1)*XU1(I)
  X W2(I) = XK2*XU2(I) + (1. - XK2)*XU2(I)
ENDDO

DO I = I, M
  UDA(I) = XK3*XW1(I) + (1. - XK3)*XW2(I)
ENDDO

DO 1 = 1, 3
  XMOM(I) = 0.
  DO J = I, M
    XMOM(I) = XMOM(I) + B(I, J) * UDA(J)
  ENDDO
ENDDO

XNORM = SQRT(XMOM(1)*XMOM(1) + XMOM(2)*XMOM(2) + XMOM(3)*XMOM(3))

IF (XNORM.NE.0.) THEN
  SAT = NORM/XNORM
  XNORM = SAT
ELSE
  SAT = 0.
ENDIF

IF (XNORM.GT.1.) XNORM = 1.
SUBROUTINE DOVERT(UDA,SAT,
 & U1,B,U_MIN,U_MAX,M,NORM)
 IMPLICIT NONE
 REAL*4 UDA(20), SAT, U_MIN(20), U_MAX(20), B(3,20), NORM
 INTEGER*4 U1(20), M
 REAL*4 XMOM(3), XNORM
 INTEGER*4 I, J
 WRITE(*,*) ' V_ZRI'EX
 CALL SETU(UDA,U1,U_MIN,U_MAX,M)
 DO 1=1,3
 XMOM(I) = 0.
 DO J=1,I
 XMOM(I) = XMOM(I)+B(I,J)*UDA(J)
 ENDDO
ENDDO
XNORM = SQRT(XMOM(1)*XMOM(1)
 & +XMOM(2)*XMOM(2)
 & +XMOM(3)*XMOM(3))
 SAT = NORM/XNORM
 XNORM = SAT
 IF (XNORM.GT.1.) XNORM = 1.
 DO I = 1,M
 UDA(I) = XNORM*UDA(I)
 ENDDO
RETURN
END

SUBROUTINE INVMAT3(MATIN, MATOUT, DET)
 IMPLICIT NONE
 INTEGER*4 I, J
 REAL*4 DET, MATIN(3,3), MATOUT(3,3)
 C Zero out the output matrix
 DO I = 1,3
 DO J = 1,3

C Calculate the determinant of the input matrix

C

0748 DET = MATIN(1,1)*MATIN(2,2)*MATIN(3,3)
0749 &   + MATIN(1,2)*MATIN(2,3)*MATIN(3,1)
0750 &   + MATIN(1,3)*MATIN(2,1)*MATIN(3,2)
0751 &   - MATIN(1,3)*MATIN(2,2)*MATIN(3,1)
0752 &   - MATIN(1,2)*MATIN(2,1)*MATIN(3,3)
0753 &   - MATIN(1,1)*MATIN(2,3)*MATIN(3,2)

C Find the matrix inverse

C

0758 IF (DET.NE.0.0) THEN
0759 MATOUT(1,1) = (MATIN(2,2)*MATIN(3,3)-MATIN(2,3)*MATIN(3,2))/DET
0760 MATOUT(1,2) = - (MATIN(1,2)*MATIN(3,3)-MATIN(1,3)*MATIN(3,2))/DET
0761 MATOUT(1,3) = (MATIN(1,2)*MATIN(2,3)-MATIN(1,3)*MATIN(2,2))/DET
0762 MATOUT(2,1) = - (MATIN(2,1)*MATIN(3,3)-MATIN(2,3)*MATIN(3,1))/DET
0763 MATOUT(2,2) = (MATIN(2,1)*MATIN(3,3)-MATIN(2,3)*MATIN(3,1))/DET
0764 MATOUT(2,3) = - (MATIN(1,1)*MATIN(2,3)-MATIN(1,3)*MATIN(2,1))/DET
0765 MATOUT(3,1) = (MATIN(2,1)*MATIN(3,3)-MATIN(2,3)*MATIN(3,1))/DET
0766 MATOUT(3,2) = - (MATIN(1,1)*MATIN(3,3)-MATIN(1,3)*MATIN(3,1))/DET
0767 MATOUT(3,3) = (MATIN(1,1)*MATIN(2,3)-MATIN(1,2)*MATIN(2,1))/DET

ENDIF

RETURN
END

C Calculate the norm of the moments

C

0783 DETNUM = DREC(DREC(MD(1))*DREC(MD(1))
0784 &   +DREC(MD(2))*DREC(MD(2))
0785 &   +DREC(MD(3))*DREC(MD(3))

C Normalize the desired moments

C

0791 DO I = 1,3
0792 V(I) = dble(MD(I))/VNORM
0793 ENDDO
Zero out the transformation matrix

DO I = 1,3
DO J = 1,3
T(I,J) = 0.0
ENDDO
ENDDO

Check to see if V(I), 3 to 1 is approx equal to zero => reduce size of problem

DO I = 3,1,-1
IF (ABS(V(I)) .LE. DTOL) THEN
T(I,I) = 1.0
ELSE
GOTO 5
ENDIF
ENDDO

T(1,1) = 1.0 or -1.0 for rotation about x-axis (depends on direction of rot.)

DO I = 3,1,-1
IF (I .EQ. i) THEN
T(I,I) = 1.0
IF (dble(MD(1)) .LT.0.0D0) T(I,I) = -1.0
RETURN
ENDIF
ENDDO

Set the 1st row of T equal to the normalized desired moments and calculate the square of each of these values

MLOCAL = I
DO I = 1,3
T(1,I) = V(I)
VLEN(I) = V(I)*V(I)
ENDDO

Developing orthogonal tranformation with V as 1st row

DO JCOL = I,MLOCAL-1
IROW = MLOCAL + 1 - JCOL
T(IROW,JCOL) = sngl(DSQRT(I.0D0*VLEN(JCOL)))
DO KCOL = JCOL+I,MLOCAL
XDOT DC = T(I,JCOL)*T(I,KCOL)
IF (IROW .NE. MLOCAL) THEN
DO I = IROW+I,MLOCAL
XDOT DC = XDOT DC + T(I,JCOL)*T(I,KCOL)
ENDDO
ENDIF
T(IROW,KCOL) = -XDOT_DC/T(IROW,JCOL)
VLEN(KCOL) = VLEN(KCOL) + dble(T(IROW,KCOL))*dble(T(IROW,KCOL))
ENDDO
ENDDO

Tricky stuff here!

DETNUM = int(mod(MLOCAL,4)/2.0)
C Necessary to do, but not easy to explain

IF (DETNUM.EQ.0.0) THEN
  IF (T(1,MLOCAL) .LT. 0.0) THEN
    T(2,MLOCAL-1) = -T(2,MLOCAL-1)
    T(2,MLOCAL)   = -T(2,MLOCAL)
  ENDIF
ELSE
  IF (T(1,MLOCAL) .GT. 0.0) THEN
    T(2,MLOCAL-1) = -T(2,MLOCAL-1)
    T(2,MLOCAL)   = -T(2,MLOCAL)
  ENDIF
ENDIF
RETURN
END

SUBROUTINE SETU(XU_SETU, IU_SETU, U_MIN, U_MAX, M)
IMPLICIT NONE
REAL*4 U_MAX(20), U_MIN(20)
INTEGER*4 IMODE
INTEGER*4 I, IU_SETU(20)
REAL*4 XU_SETU(20)
DO I = 1,M
  IF (IU_SETU(I) .EQ. 1) THEN
    XU_SETU(I) = U_MAX(I)
  ELSEIF (IU_SETU(I) .EQ.-1) THEN
    XU_SETU(I) = U_MIN(I)
  ELSE
    XU_SETU(I) = 0.
  ENDIF
ENDDO
RETURN
END

SUBROUTINE R20(U1, IU, INFRONT, ITHETA, NANGS, INDX, ISVERTEX,
  & B1, U_MAX, XUMAX, U_MIN, U_MAX, TOL, M, DIAGS)
DATA PIOVR2/1.570796326794897/
DATA PI/3.141592653589793/

IF (DIAGS) THEN
WRITE(*, '(/AS0/)') ' Entering P2C'
WRITE(*, '(/AS0/)') ' Calling arguments'
WRITE(*, '(/A30,6E13.6/)') ' *R2D* BI(1,:) = ', (B1(1:), I=1,M)
WRITE(*, '(/A30,6E13.6/)') ' *R2D* BI(2,:) = ', (B1(2:), I=1,M)
WRITE(*, '(/A30,6E13.6/)') ' *R2D* BI(3,:) = ', (B1(3:), I=1,M)
WRITE(*, '(/A30,6E13.6/)') ' *R2D* UMAX = ', (UMAX(1:), I=1,M)
WRITE(*, '(/A30,6F14.6/)') ' *R2D* XUMAX = ', (XUMAX(1:), I=1,M)
WRITE(*, '(/A30,6E13.6/)') ' *R2D* TOL = ', TOL
WRITE(*, '(/A30,13/)') ' *R2D* M = ', M
ENDIF

C Calculate Y

Y = 0.0
DO I = 1,M
UL(I) = UMAX(I)
Y = Y + BI(2,I)*XUMAX(I)
ENDDO

C VERTEX CHECK

ISVERTEX = .FALSE.
IF (ABS(Y).LT.TOL) THEN
Y = 0.
Z = 0.
DO I=1,M
Z = Z+BI(3,I)*XUMAX(I)
ENDDO
ENDIF
IF (ABS(Z).LT.TOL) THEN
C WRITE(*,*) 'Z ',Z,
ISVERTEX = .TRUE.
IU = 0
INFRONT = 0.
NANGS = 0
RETURN
ENDIF

C Get the angle

IF (DIAGS) THEN
WRITE(*, '(/AS0/)') ' First calculations'
WRITE(*, '(/A30,13/)') ' *R2D* Y = ', Y
WRITE(*, '(/A30,13/)') ' *R2D* SY = ', SY
ENDIF

C Get the angle
IF (DIAGS) THEN
WRITE(*,'(A33)') 'Getting angles'
ENDIF

NANGS = 0
DO I=1, M
ANG = ATAN2(-B1(1,I),B1(2,I))
IF (ABS(ANG) > PI/2.) THEN
IF (ANG < 0.) THEN
ANG = ANG + PI
ELSE
ANG = ANG - PI
ENDIF
ENDIF
NANGS = NANGS + I
THETA(NANGS) = ANG
ITHETA(NANGS) = I
IF (DIAGS) THEN
WRITE(*,'(A30,3,AZ0,E18.6)') 'R20* I = ', I, ', ANG = ', ANG
ENDIF
ENDDO

C THETA and ITHETA now sorted by control number
C Sort THETA by magnitude. ITHETA gets shuffled the same way
C
IF (DIAGS) THEN
WRITE(*,'(A50)') 'Before sorting'
WRITE(*,'(A30)') 'R20* NANGS = ', NANGS
WRITE(*,'(A30)') 'R20* ITHETA = ', (ITHETA(I), I=1,NANGS)
WRITE(*,'(A30)') 'R20* THETA = ', (THETA(I), I=1,NANGS)
ENDIF
CALL SORTC(THETA, ITHETA, NANGS, THETA, ITHETA)
IF (DIAGS) THEN
WRITE(*,'(A50)') 'After sorting'
WRITE(*,'(A30)') 'R20* NANGS = ', NANGS
WRITE(*,'(A30)') 'R20* ITHETA = ', (ITHETA(I), I=1,NANGS)
WRITE(*,'(A30)') 'R20* THETA = ', (THETA(I), I=1,NANGS)
ENDIF

C FIND INDEX OF ZERO
DO I=1, NANGS
IF (THETA(I) > 0.) THEN
INDX = I
DO J=NANGS, I, -1
THETA(J+1) = THETA(J)
ITHETA(J+1) = ITHETA(J)
ENDDO
THETA(I) = 0.
ITHETA(I) = 0
GOTO 193
ENDIF
ENDDO
C CONTINUE
IF (DIAGS) THEN
    WRITE(*, '/(ASO0)') ' After existing'
    WRITE(*, '/(ASO,130)') ' RDC* INDEX = ', INDEX
    WRITE(*, '/(ASO,130)') ' RDC* THETA = ', (ITHETA(I), I=1,NANGS+1)
    WRITE(*, '/(ASO,130)') ' RDC* THETA = ', (THETA(I), I=1,NANGS+1)
ENDIF

CALL GETEDGE(U1, IU, INFRONT, ISVERTEX,

& B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDEX, TOL, M, DIAGS)

C AS FUNCTIONS OF

SUBROUTINE GETEDGE(U2, JU, INFRONT, ISVERTEX,

& B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDEX, TOL, M, DIAGS)

IMPLICIT NONE
INTEGER*4 M, SY, UMAX(20), INDEX, NANGS, ITHETA(21)
REAL*4 B1(3,20), U_MAX(20), U_MIN(20), XU(20), Y
REAL*4 INFRONT, TOL
INTEGER*4 U2(20), JU, SSY
LOGICAL*4 ISVERTEX, DIAGS STUCK
INTEGER*4 I, J, I_COUNT, IX
REAL*4 XK, XU2(20), Y_PREV, Z, Z1, Z2, SAVE_Y, MOM(3)

IF (DIAGS) THEN
    WRITE(*, '/(ASO0)') ' Entering GETEDGE'
ENDIF

DO I=1,M
    U2(I) = I/MAX(I)
ENDDO
SAVE Y = Y
IX = INDEX-SY
SSY = SY
STUCK = .FALSE.
ICOUNT = 0
IF (DIAGS) THEN
    WRITE(*, '/(ASO0)') ' Beginning GETEDGE COMPILE'
ENDIF

DO I=1,M
    U2(I) = UMAX(I)
ENDDO
SAVE_Y = Y
IX = INDEX-SY
SSY = SY
STUCK = .FALSE.
ICOUNT = 0
IF (DIAGS) THEN
    WRITE(*, '/(ASO0)') ' Beginning GETEDGE COMPILE'
ENDIF
DO WHILE ((SY.EQ.SSY).AND.(IX.GT.0).AND.(IX.LE.NANGS))
  ICOUNT = ICOUNT+1
  YPREV = Y
  JU = ITHETA(IX)
  Y = Y-U2(JU)*B1(2, JU)*(U_MAX(JU)-U_MIN(JU))
  SSY = 1
  IF (Y.LT.0.0) SSY = -1
  IF (ABS(Y).LT.TOL) THEN
    Y = 0.
    SSY = 0
  ENDIF
  U2(JU) = -U2(JU)
  IF (SY.NE.0.) THEN
    IX = IX-SY
  ELSE
    IX = IX-I
  ENDIF
  ICOUNT = ICOUNT+I
  IF (ICOUNT GT.20) STUCK = .TRUE
  IF (DIAGS) THEN
    WRITE(*, ' ;A30, L33') ' Bottom of GETEDGE DOWHILE'
    WRITE(*, ' ;A30, L33') ' *GETEDGE* SAVE_Y = ', SAVE_Y
    WRITE(*, ' ;A30, L33') ' *GETEDGE* IX = ', IX
    WRITE(*, ' ;A30, L33') ' *GETEDGE* SSY = ', SSY
    WRITE(*, ' ;A30, L33') ' *GETEDGE* NANGS = ', NANGS
    WRITE(*, ' ;A30, L33') ' ITHETA(IX) = ', ITHETA(IX)
    WRITE(*, ' ;A30, L33') ' (SY.EQ.SSY) = ', (SY.EQ.SSY)
    WRITE(*, ' ;A30, L33') ' (U2(I), I=1,M) = ', (U2(I), I=1,M)
    WRITE(*, ' ;A30, L33') ' ICOUNT = ', ICOUNT
  ENDIF
  IF (STUCK) THEN
    PAUSE('Stuck in GETEDGE in DOWHILE')
    STUCK = .FALSE.
  ENDIF
  ENDDO
  IF (SY.EQ.SSY) THEN
    STUCK = .FALSE.
    ICOUNT = 0
  ENDIF
  IF (DIAGS) THEN
    WRITE(*, ' ;A30, L33') ' Beginning Alt. GETEDGE DOWHILE'
    WRITE(*, ' ;A30, L33') ' *GETEDGE* U2 = ', (U2(I), I=1,M)
    WRITE(*, ' ;A30, L33') ' *GETEDGE* SAVE_Y = ', SAVE_Y
    WRITE(*, ' ;A30, L33') ' *GETEDGE* IX = ', IX
    WRITE(*, ' ;A30, L33') ' *GETEDGE* SSY = ', SSY
    WRITE(*, ' ;A30, L33') ' *GETEDGE* NANGS = ', NANGS
1141   ENDIF
1142 C
1143   Y = SAVE_Y
1144   IX = INDX+SY
1145   DO WHILE ((SY.EQ.SSY).AND.(IX.GT.0).AND.(IX.LE.NANGS+1))
1146     YPREV = Y
1147     JU = ITHETA(IX)
1148     Y = Y-U2(JU)*BI(2,JU)*(U_MAX(JU)-U_MIN(JU))
1149     SSY = 1
1150     IF (Y.LT.0.0) SSY = -1
1151     IF (ABS(Y).LT.TOL) THEN
1152       Y = 0.
1153       SSY = 0
1154     ENDIF
1155     U2(JU) = -U2(JU)
1156     IF (SY.NE.0.) THEN
1157       IX = IX+SY
1158     ELSE
1159       IX = IX+1
1160     ENDIF
1161 C
1162   ICOUNT = ICOUNT+1
1163   IF (ICOUNT.GT.20) STUCK = .TRUE.
1164   IF (DIAGS) THEN
1165     WRITE(*,'(/A65c)') ' Bottom of Alt. GETEDGE DOWHILE'
1166     WRITE(*,'(A30,13)') ' *GETEDGE* ICOUNT = ', ICOUNT
1167     WRITE(*,'(A30,13)') ' *GETEDGE* STUCK = ', STUCK
1168     WRITE(*,'(A30,13)') ' *GETEDGE* YPREV = ', YPREV
1169     WRITE(*,'(A30,13)') ' *GETEDGE* JU = ', JU
1170     WRITE(*,'(A30,13)') ' *GETEDGE* Y = ', Y
1171     WRITE(*,'(A30,13)') ' *GETEDGE* SSY = ', SSY
1172     WRITE(*,'(A30,13)') ' *GETEDGE* SY.EQ.SSY = ', (SY.EQ.SSY)
1173     WRITE(*,'(A30,13)') ' *GETEDGE* U2 = ', (U2(I), I=1,M)
1174     WRITE(*,'(A30,13)') ' *GETEDGE* IX = ', IX
1175   ENDIF
1176   IF (STUCK) THEN
1177     PAUSE('Stuck in GETEDGE in Alt. DOWHILE')
1178   ENDIF
1179 C
1180   ENDDO
1181   ENDIF
1182 C
1183   IF (SY.EQ.SSY) THEN
1184     JU = 0
1185   ENDIF
1186 C
1187   INFRONT = 0.
1188   ISVERTEX = .FALSE.
1189   RETURN
1190   ENDIF
1191 C
1192   XK = YPREV/(YPREV-Y)
1193   CALL SETU(XU2, U2, U_MIN, U_MAX, M)
1194   Z2 = 0.
1195   DO I=1,M
1196     Z2 = Z2+B1(3,I)*XU2(I)
1197   ENDDO
1198   Z1 = Z2-U2(JU)*B1(3,JU)*(U_MAX(JU)-U_MIN(JU))
Z = XK*Z2+(1.-XK)*Z1

IF (Z.LT.0.) INFRONT = -1.

ISVERTEX = ((Y.EQ.0.) .AND. (ABS(Z) .LT.TOL))

IF (DIAGS) THEN
  WRITE(*,13) ' Leaving GETEDGE'
  WRITE(*,14) ' ', (U2(I), I=1,M)
  WRITE(*,15) ' ', (XU2(I), I=1,M)
  WRITE(*,16) ' ', JU
  WRITE(*,17) ' ', XK
  WRITE(*,18) ' ', ZZ
  WRITE(*,19) ' ', Z
  WRITE(*,20) ' ', Z
  WRITE(*,21) ' ', JU
  WRITE(*,22) ' ', ISVERTEX
ENDIF

RETURN
END

SUBROUTINE ISFACET(ISOK, IUOUT, JUOUT, U123, IU, JU, U1, U2, B, M, TOL)

IMPLICIT NONE
LOGICAL*4 ISOK
INTEGER*4 IU, JU, M, UI(20), U2(20), IUOUT, JUOUT, U123(20,3)
REAL*4 B(3,20), TOL

LOCALS
INTEGER*4 UXI(20), UX2(20), I, J, K, II, JJ, KK
REAL*4 THEMAT(2,2), MATINV(2,2), MATDET, T2(2), T1(3), TESTFACET(20)
REAL*4 TEMP
INTEGER*4 DIM, OBJ(20), UDEF(20), ITF(20), THEOBJ(20)

ISOK = .FALSE.
DO I=1,M
  UX1(I) = U1(I)
  UX2(I) = U2(I)
ENDDO
U1(IU) = 0
U2(JU) = 0
DIM = 0
DO I = 1,M
  OBJ(I) = U1(I)
  IF ((I.EQ.IU) .OR. (I.EQ.JU) .OR. (U2(I).NE.U1(I))) THEN
    OBJ(I) = 0
    DIM = DIM + 1
    UDEF(DIM) = I
  ENDIF
ENDDO
IF (DIM.EQ.2) THEN
  ISOK = .TRUE.
DO II=1,3
  JJ = MOD(II,3)+1
  KK = MOD(JJ,3)+1
  THEMAT(1,1) = B(II,UDEF(1))
  THEMAT(1,2) = B(JJ,UDEF(1))
  THEMAT(2,1) = B(II,UDEF(2))
  THEMAT(2,2) = B(JJ,UDEF(2))
  MATDET = THEMAT(1,1)*THEMAT(2,2)-THEMAT(1,2)*THEMAT(2,1)
  IF (MATDET.NE.0.) THEN
    MATINV(1,1) = THEMAT(2,2)/MATDET
    MATINV(1,2) = -THEMAT(1,2)/MATDET
    MATINV(2,1) = THEMAT(1,1)/MATDET
    MATINV(2,2) = -THEMAT(2,1)/MATDET
    T2(1) = -MATINV(1,1)*B(KK,UDEF(1)) - MATINV(1,2)*B(KK,UDEF(2))
    T2(2) = -MATINV(2,1)*B(KK,UDEF(1)) - MATINV(2,2)*B(KK,UDEF(2))
    T1(KK) = 1.
    T1(II) = T2(1)
    T1(JJ) = T2(2)
  DO I=1,M
    TESTFACET(I) = 0.
    ITF(I) = 0
    UDEF(I) = 0
  DO J=1,3
    TESTFACET(I) = TESTFACET(I)+T1(J)*B(J,I)
  ENDDO ! DO J=1,3
  IF (ABS(TESTFACET(I)).LT.TOL) THEN
    ITF(I) = 0
  ELSEIF (TESTFACET(I).GT.0.) THEN
    ITF(I) = 1
  ELSEIF (TESTFACET(I).LT.0.) THEN
    ITF(I) = -1
  ENDIF
  ENDDO ! DO I=1,M
  ENDIF ! IF (DIM.NE.2) THEN
  DIM = 0
  ENDIF ! IF (DIM.NE.2) THEN
  DO I=1,M
    THEOBJ(I) = OBJ(I)
  IF ((OBJ(I).EQ.0) .OR. (ITF(I).EQ.0) .OR. (ITF(I).NE.OBJ(I))) THEN
    THEOBJ(I) = 0
    DIM = DIM + 1
    UDEF(DIM) = I
  ENDIF
  ENDIF ! IF (DIM.NE.2) THEN
  IF (DIM.NE.2) THEN
    DIM = 0
    J = 0
  DO I=1,M
    THEOBJ(I) = OBJ(I)
  IF ((OBJ(I).EQ.0) .OR. (ITF(I).EQ.0) .OR. (-ITF(I).NE.OBJ(I))) THEN
    THEOBJ(I) = 0
    DIM = DIM + 1
    UDEF(DIM) = I
  ENDIF
  ENDIF ! IF (DIM.NE.2) THEN
  ENDIF ! IF (DIM.NE.2) THEN
1312 IF (DIM.NE.2) ISOK = .FALSE.
1313 GOTO 194
1314
1315 ENDIF ! IF (MATDET.NE.0.) THEN
1316 ENDDO ! DO II=1,3
1317 194 CONTINUE
1318 ENDDO ! IF (DIM.EQ.2)
1319 C
1320 IF (ISOK) THEN
1321 C
1322 IUOUT = UDEF(1)
1323 JUOUT = UDEF(2)
1324 DO I=1,M
1325 DO J=1,3
1326 U123(I,J) = OBJ(I)
1327 END DO
1328 IUOUT = 0
1329 JUOUT = 0
1330 DO I=1,M
1331 DO J=1,3
1332 U123(I,J) = 0
1333 END DO
1334 ENDIF
1335 RETURN
1336 END
1337 C
1338 SUBROUTINE SORTC(X, IY, N, XS, IYC)
1339 IMPLICIT NONE
1340 INTEGER*4 I, IL_SORTC(36), IMED, IP1, IPR, ITY, IU_SORTC(36)
1341 INTEGER*4 IY(20), IYC(20), J, JMI, JMK, K, L, LMI, M, MID, N
1342 INTEGER*4 NM1
1343 REAL*4 HOLD, AMED, TX, X(20), XS(20)
1344 C
1345 C
1346 CHECK THE INPUT ARGUMENTS FOR ERRORS
1347 C
1348 IPR=11
1349 IF(N.LT.1)GOTO50
1350 IF(N.EQ.1)GOTO55
1351 HOLD=X(1)
1352 DO60I=2,N
1353 IF(X(I) .NE.HOLD)GOTO90
1354 60 CONTINUE
WRITE(*, 9) HOLD
DO61I=1,N
XS(I)=X(I)
IYC(I)=IY(I)
61 CONTINUE
RETURN
50 WRITE(*,15)
WRITE(*,47)N
50 RETURN
55 WRITE(*,18)
XS(1)=X(1)
IYC(1)=IY(1)
RETURN
90 CONTINUE
9 FORMAT(IX, '**** NON-FATAL DIAGNOSTIC--THE FIRST INPUT ARGUMENT TO THE SORTC SUBROUTINE HAS ALL ELEMENTS = ',E15.8,
1 *****')
15 FORMAT(IX, '**** FATAL ERROR--THE SECOND INPUT ARGUMENT TO THE SORTC SUBROUTINE IS NON-POSITIVE *****')
18 FORMAT(IX, '**** NON-FATAL DIAGNOSTIC--THE SECOND INPUT ARGUMENT TO THE SORTC SUBROUTINE HAS THE VALUE 1 *****')
47 FORMAT(IX, '**** THE VALUE OF THE ARGUMENT IS ',I8, ' *****')

COPY THE VECTOR X INTO THE VECTOR XS
DO100I=1,N
XS(I)=X(I)
100 CONTINUE
COPY THE VECTOR IY INTO THE VECTOR IYC
DO150I=1,N
IYC(I)=IY(I)
150 CONTINUE
CHECK TO SEE IF THE INPUT VECTOR IS ALREADY SORTED
NM1=N-1
DO200I=1,NM1
IP1=I+1
IF(XS(I).LE.XS(IP1))GOTO200
GOTO250
200 CONTINUE
RETURN
M=1
I=1
J=N
IF(I.GE.J)GOTO370
305 IF(I.LE.J)GOTO370
310 K=1
MID=(I+J)/2
AMED=XS(MID)
IMED=IYC(MID)
IF(XS(I).LE.XS(IP1))GOTO200
GOTO250
200 CONTINUE
RETURN
250 M=1
310 K=1
MID=(I+J)/2
AMED=XS(MID)
IMED=IYC(MID)
IF(XS(I).LE.XS(IP1))GOTO320
GOTO250
IYC(MID)=IYC(I)
XS(I)=AMED
IYC(I) = IMED
AMED = XS(MID)
IMED = IYC(MID)

320 L = J

IF (XS(J) .GE. AMED) GOTO 340
XS(MID) = XS(J)
IYC(MID) = IYC(J)
XS(J) = AMED
IYC(J) = IMED
AMED = XS(MID)
IMED = IYC(MID)

IF (XS(I) .LE. AMED) GOTO 340
XS(MID) = XS(I)
IYC(MID) = IYC(I)
XS(I) = AMED
IYC(I) = IMED
AMED = XS(MID)
IMED = IYC(MID)

GOTO 340

330 XS(L) = XS(K)
IYC(L) = IYC(K)
XS(K) = TX
IYC(K) = ITY

340 L = L - 1

IF (XS(L) .GT. AMED) GOTO 340
TX = XS(L)
ITY = IYC(L)

350 K = K + 1

IF (XS(K) .LT. AMED) GOTO 350
IF (K .LE. L) GOTO 330

LMI = L - I
JMK = J - K

IF (LMI .LE. JMK) GOTO 360
IL_SORTC(M) = I
IU_SORTC(M) = L
I = K
M = M + 1
GOTO 380

360 IL_SORTC(M) = K
IU_SORTC(M) = J
J = L
M = M - 1
GOTO 380

370 M = M - 1

IF (M .EQ. 0) RETURN
I = IL_SORTC(M)
J = IU_SORTC(M)

380 JMI = J - I

IF (JMI .GE. 11) GOTO 310
IF (I .EQ. 1) GOTO 305
I = I - L

390 I = I + 1

IF (I .EQ. J) GOTO 370
AMED = XS(I + 1)
IMED = IYC(I + 1)

IF (XS(I) .LE. AMED) GOTO 390
K = I
C..............................................................................

SUBROUTINE MINNORM(UMINNORM, SCALE,
C AS A FUNCTION OF
&P, U, UTRIM, MTRIM, UDA, B, U_MIN, U_MAX, M, TIME, XSCALE)

IMPLICIT NONE

** Parameters
** INPUTS:
REAL*4 P(20,3), U(20), UTRIM(20), UDA(20)
REAL*4 B(3,20), MTRIM(3), U_MAX(20), U_MIN(20)
REAL*4 TIME, XSCALE
INTEGER*4 M, IMODE

** OUTPUTS:
REAL*4 UMINNORM(20), SCALE

** LOCALS:
REAL*4 UKDA(20), UP(20), UDELTA(20), M_ATT(3)
INTEGER*4 I, J, K

SCALE = 1.

DO I=1,M
    UKDA(I) = U(I) + UDA(I)
    UP(I) = UTRIM(I)
    ENDDO

DO I=1,3
    M_ATT(I)=0.
    DO J=1,M
        M_ATT(J)=M_ATT(J)+B(I,J)*UKDA(J)
        UP(I) = UP(I) + P(I,J)*(M_ATT(J)-MTRIM(J))
    ENDDO
    ENDDO

DO I=1,M
    IF ((U_MIN(I).EQ.0.).OR.(U_MAX(I).EQ.0.)) SCALE = 0.
    UKDA(I) = U(I) + UDA(I)
    UP(I) = UTRIM(I)
    DO J=1,3
        UP(I) = UP(I) + P(I,J)*(M_ATT(J)-MTRIM(J))
    ENDDO
    ENDDO

UDELTA(I) = SCALE*(UP(I) - UKDA(I))

IF (UDELTA(I) .NE. 0.) THEN
  IF ((UDELTA(I) .GT. U_MAX(I)) .AND. (U_MAX(I) .GT. 0.)) THEN
    SCALE = U_MAX(I) / UDELTA(I)
  ELSEIF ((UDELTA(I) .LT. U_MIN(I)) .AND. (U_MIN(I) .LT. 0.)) THEN
    SCALE = U_MIN(I) / UDELTA(I)
  ENDIF
ENDIF
ENDDO

SCALE = XSCALE*SCALE

DO I=1,M
  UMINNORM(I) = UKDA(I) + SCALE*(UP(I) - UKDA(I))
ENDDO

RETURN
END