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, Aeropaf, Control.f, Constants.f, Engine.f, and Alloc.f.

2.1.1 Aero.f

The aero code first calls Aeropaf 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 $\alpha_{cmd}$, beta command $\beta_{cmd}$, and a roll rate command $\rho_{cmd}$. These commands are input to a simple dynamic inversion control law that generates desired moments for the control allocation subroutine. First, $\alpha_{cmd}$ and $\beta_{cmd}$ are converted to desired accelerations $\dot{\alpha}_{des}$ and $\dot{\beta}_{des}$:

$$
\dot{w}_{cmd} = u \tan \alpha_{cmd}
$$

$$
\dot{\alpha}_{des} = \lambda_w (w - w_{cmd})
$$

$$
\dot{\beta}_{cmd} = V \sin \beta_{cmd}
$$

$$
\dot{\beta}_{des} = \lambda_V (v - v_{cmd})
$$

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

$$
\dot{q}_{cmd} = \frac{\omega_{des} + pu + g \cos \theta \cos \phi - Z/m}{u}
$$

$$
\dot{r}_{cmd} = \frac{-\omega_{des} + pu + g \cos \theta \sin \phi + Y/m}{u}
$$

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,

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

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

\[ C_{\ell,des} = -C_{\ell} + \frac{I_{xx} \dot{\theta}_{des} + I_{xx} \dot{\phi}_{des} + \dot{r} + (I_{xx} - I_{yy}) \dot{r} - I_{xx} \dot{p}}{Q_{sb}} \]
\[ C_{m,des} = -C_{m} + \frac{I_{yy} \dot{\phi}_{des} + (I_{xx} - I_{yy}) \dot{r} + I_{xx} (p^2 - r^2)}{Q_{se}} \]
\[ C_{n,des} = -C_{n} + \frac{I_{yy} \dot{\phi}_{des} + I_{yy} \dot{r}_{des} + (I_{yy} - I_{xx}) \dot{p} + I_{xx} \dot{r}}{Q_{sb}} \]

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 up¹. 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

¹The 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 bisecting, 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, $B_3 = TB_{3r,y}^o$.

**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 $U_1$, the control that defines that edge in variable $I_U$, 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 $y_1$. $B_1$ 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 $y_1$-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 $y_3$ 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 \[\hat{e}_{3,1}(v^p_1 - v^p_2)(v^p_1 - v^p_3)\] in [2, Equation (13)]; variables AA, BB, and CC correspond to \(\alpha_3\), \(C_{1,2}\), and \(C_{1,3}\) respectively; and MTEMP is \(v^p_1\). 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 \( y_1-y_2 \) 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 $\mathcal{L}_\phi$, and ITHETA that of $\mathcal{L}_u$. 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 \( \text{IX} = \text{IX-SY} \).

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, \( U2 \) 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 \text{GETEDGE} is called from \text{R20}, but the first loop may fail when called from within \text{DA3}. The list is traversed in the opposite direction by the index \( \text{IX} = \text{IX-SY} \). 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 \( U2 \) (a vertex in object notation) and \( JU \) (the number of the control that defines the edge). \( U2 \) is converted to control notation using the subroutine \text{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 x 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 = 1sec 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 = 1sec 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 = 1sec 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


**TITLE: DA3**

**FUNCTION:** 3 Moment Control Allocator

Direct Allocation for the 3 objective problem using bisecting edge searching algorithm

**DESIGNED BY:** Bull Durham

**CODED BY:** Kevin Scalera

**MAINTAINED BY:** VPI SIMULATIONS

**MODIFICATION HISTORY:**

**INPUTS:**

IMODE Sim. mode: -2=init, -1=reset, 0=hold, 1=ru

**OUTPUTS:**

NONE

**LOCALS:**

NONE

**OTHER LOCALS:**

NONE
SUBROUTINE DA3(UDA, SAT, IERR, & B, MDES, U_MIN, U_MAX, M, NBI, TIME, DIAGS)

C ..............................................................................

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, ITHETA(21), IU, IUOUT

INTEGER*4 IUTEMP(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), UMAX(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

COMMON SHELL1 / CONPAR( 424)

COMMON SECTION

COMMON SECTION

COMMON SECTION

EQUIVALENCE SECTION

EQUIVALENCE CONPAR(1), IMODE
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, 9.238795325112868e-01,
& 9.807852804032304e-01, 9.807852804032305e-01,
& 9.951847266721969e-01, 9.951847266721969e-01,
& 9.987954562051724e-01, 9.987954562051724e-01,
& 9.996988186962042e-01, 9.996988186962042e-01,
& 9.999247018391445e-01, 9.999247018391445e-01,
& 9.999811752826011e-01, 9.999811752826011e-01,
& 9.999970586828822e-01, 9.999970586828822e-01,
& 9.99999264657179e-01, 9.99999264657179e-01,
& 9.999999816164293e-01, 9.999999816164294e-01,
& 9.999999997127567e-01, 9.999999997127567e-01,
& 9.999999999820472e-01, 9.999999999820472e-01,
& 9.999999999955118e-01, 9.999999999955118e-01,
& 9.999999999999878e-01, 9.999999999999878e-01/

INITIALIZATION SECTION

IF( (IMODE.LE.-2) .OR. .NOT.INITIALIZED ) THEN
ENDIF

RESET SECTION

IF ((IMODE.LE.-1).OR.(.NOT.Initialized)) THEN
Initialized = .TRUE.
ENDIF

IERR = 0 FACET FOUND, ABC OK
IERR = 1 FACET NOT FOUND, INTERPOLATED SOLUTION

RUN SECTION
IF (DIAGS) THEN
  WRITE(*, '(A50)') ' Entering DA3'
  WRITE(*, '(A50)') ' Calling arguments'
  WRITE(*, '(A30,2E13.6)') ' DA1* B(1, :) = ', (B(1, I), I=1,M)
  WRITE(*, '(A30,2E13.6)') ' DA1* B(2, :) = ', (B(2, I), I=1,M)
  WRITE(*, '(A30,2E13.6)') ' DA1* B(3, :) = ', (B(3, I), I=1,M)
  WRITE(*, '(A30,2E13.6)') ' DA1* U_MIN = ', (U_MIN(I), I=1,M)
  WRITE(*, '(A30,2E13.6)') ' DA1* U_MAX = ', (U_MAX(I), I=1,M)
  WRITE(*, '(A30,13)') '* DA3* X = ', M
  WRITE(*, '(A30,13)') '* DA3* NBI = ', NBI
  WRITE(*, '(A30,13)') '* DA3* TIME = ', TIME
ENDIF

INFRONT = 1.0

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

IF (NORM .EQ. 0.0) THEN
  IERR = 0
  SAT = 0.0
  DO I = 1, M
    UDA(I) = 0.0
  ENDDO
  RETURN
ENDIF

NORM = SQRT(NORM)

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

CALL DCGEN(T, MD)

DO I = 1, M
  IF (BI(I,I) .EQ. 0.) THEN
    UMAX(I) = 0
  ELSEIF (BI(I,I) .LT. 0.0) THEN
    UMAX(I) = -1
  ELSE
    UMAX(I) = 1
  ENDIF
ENDDO
ENDDO
CALL SETU(XUM/L_, UMAX, U_MIN, U_MAX, M)
TOLANG = CSPHI(2, MIN(20, 2*NBI))
DO I=1, M
    MXMAX(I) = 0.
    DO J=1, M
        MXMAX(I) = MAX(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
IF (DIAGS) THEN
    WRITE(*, ' A30,E18.6') ' Preliminary Calcs'
    WRITE(*, ' A30,E18.6') ' DA1* NORM = ', NORM
    WRITE(*, ' A30,E18.6') ' DA1* XD = ', (MD(I), I=1,3)
    WRITE(*, ' A30,E18.6') ' DA1* T(1,: ) = ', (T(1, I), I=1,3)
    WRITE(*, ' A30,E18.6') ' DA1* T(2,: ) = ', (T(2, I), I=1,3)
    WRITE(*, ' A30,E18.6') ' DA1* T(3,: ) = ', (T(3, I), I=1,3)
    WRITE(*, ' A30,E18.6') ' DA1* B1(1,:) = ', (B1(1, I), I=1, M)
    WRITE(*, ' A30,E18.6') ' DA1* B1(2,:) = ', (B1(2, I), I=1, M)
    WRITE(*, ' A30,E18.6') ' DA1* B1(3,:) = ', (B1(3, I), I=1, M)
    WRITE(*, ' A30,E18.6') ' DA1* UMAX = ', (XUMAX(I), I=1, M)
    WRITE(*, ' A30,E18.6') ' DA1* MXMAX = ', (MXMAX(I), I=1, M)
    WRITE(*, ' A30,E18.6') ' DA1* TOLANG = ', TOLANG
    WRITE(*, ' A30,E18.6') ' DA1* MAXNORM = ', MAXNORM
    WRITE(*, ' A30,E18.6') ' DA3* TOL = ', TOL
    WRITE(*, ' A30,E18.6') ' First call to R20'
ENDIF
CALL R20(U1, IU, INFRONT, TEMP2, TEMP3, TEMP4, ISVERTEX,
& B1, UMAX, XUMAX, U_MIN, U_MAX, TOL, M, DIAGS)
IF (DIAGS) THEN
    WRITE(*, ' A30,E18.6') ' After 1st R20'
    WRITE(*, ' A30,E18.6') ' DA3* TEMP2 (THETA) = ', (TEMP2(I), I=1,TEMP3+1)
    WRITE(*, ' A30,E18.6') ' DA3* TEMP3 (NANGS) = ', TEMP3
    WRITE(*, ' A30,E18.6') ' DA3* TEMP4 (INDX) = ', TEMP4
ENDIF
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) = U1(I)
        ENDDO
        I_LIB = IU
    ENDIF
ENDIF
0286 IF (DIAGS) THEN
0287 WRITE(*,'(A)') ' After 1st LIF/LIB'
0288 WRITE(*,'(A30,6I3)') ' *DA* U_LIF = ', (U_LIF(I), I=1,M)
0289 WRITE(*,'(A30,13I3)') ' *DA* _LIF = ', _LIF
0290 WRITE(*,'(A30,6I3)') ' *DA* U_LIB = ', (U_LIB(I), I=1,M)
0291 WRITE(*,'(A30,I13)') ' *DA* _LIB = ', _LIB
0292 ENDIF
0293 ENDIF
0294 C
0295 IF (ISVERTEX) THEN
0296 C WRITE(*,'(A)') ' Time = ', TIME, ' FIRST CALL TO R29'
0297 CALL DOVERT(UDA,SAT,U1,UB,U_MIN,U_MAX,M,NORM)
0298 RETURN
0299 ENDIF
0300 C
0301 C 1st rotation about x-axis
0302 C
0303 C IF (M.GE.8) THEN
0304 ICOUNT = 1
0305 ELSE
0306 ICOUNT = 2
0307 ENDIF
0308 C ICOUNT = 1
0309 COSPHI = CSPHI(1,ICOUNT)
0310 S3PHI = INFRONT*CSPHI(2,ICOUNT)
0311 T22(1,1) = COSPHI
0312 T22(1,2) = -S3PHI
0313 T22(2,1) = S3PHI
0314 T22(2,2) = COSPHI
0315 MAXSTEPS = 2*INT(ABS(PI/ASIN(S3PHI)))
0316 WASINF = INFRONT
0317 ISOK = .FALSE.
0318 NMAX = NBI + 1
0319 DIDSWITCH = .FALSE.
0320 STEPS = 0
0321 STUCK = .FALSE.
0322 C IF (DIAGS) THEN
0323 WRITE(*,'(A)') ' Before Main Loop
0324 WRITE(*,'(A30,6I3)') ' *DA* COSPHI = ', COSPHI
0325 WRITE(*,'(A30,6I3)') ' *DA* S3PHI = ', S3PHI
0326 WRITE(*,'(A30,I13)') ' *DA* MAXSTEPS = ', MAXSTEPS
0327 ENDIF
0328 C MAIN LOOP ******************************************
0329 DO WHILE ((ICOUNT.LT.NMAX).AND. (.NOT.ISOK))
0330 C STEPS = STEPS+1
0331 IF (STEPS.GE.MAXSTEPS) THEN
0332 STUCK = .TRUE.
0333 WRITE(*,'(A)') ' Stuck at Step = ', STEPS
0334 WRITE(*,'(A)') ' Stuck
0335 WRITE(*,'(A30,6I3)') ' *DA* STUCK = ', STUCK
0336 WRITE(*,'(A30,6I3)') ' *DA* TIME = ', TIME
0337 ENDIF
0338 C
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 DO WHILE'
  WRITE(*, '(/AS02)') ' DA3, COUNT = ', IOCOUNT
  WRITE(*, '(/AS03)') ' DA3, STEPS = ', STEPS
  WRITE(*, '(/AS04)') ' DA3, MAXSTEPS = ', MAXSTEPS
  WRITE(*, '(/AS05)') ' DA3, B1(2,I) = ', (B1(2,I),I=1,M)
  WRITE(*, '(/AS06)') ' DA3, B1(3,I) = ', (B1(3,I),I=1,M)
ENDIF

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

CALL R20(U1,IU,INFRONT,TEMP2,TEMP3,TEMP4,ISVERTEX,
  & B1,UMAX,XUMAX,U_MIN,U_MAX,TOL,M,DIAGS)

IF (DIAGS) THEN
  WRITE(*, '(/AS07)') ' After Loop R20'
  WRITE(*, '(/AS08)') ' DA3, TEMP2 (ITHETA) = ', (TEMP2(I), I=1,TEMP3+1)
  WRITE(*, '(/AS09)') ' DA3, TEMP3 (NANGS) = ', TEMP3
  WRITE(*, '(/AS10)') ' DA3, TEMP4 (INDX) = ', TEMP4
ENDIF

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_LIF(I) = U1(I)
    ENDDO
    I_LIF = IU
  ENDIF
ENDIF

IF (DIAGS) THEN
  WRITE(*, '(/AS11)') ' After Loop R20 LIF:LIB'
  WRITE(*, '(/AS12)') ' DA3* U_LIF = ', (U_LIF(I), I=1,M)
  WRITE(*, '(/AS13)') ' DA3* I_LIF = ', I_LIF
  WRITE(*, '(/AS14)') ' DA3* U_LIB = ', (U_LIB(I), I=1,M)
  WRITE(*, '(/AS15)') ' DA3* I_LIB = ', I_LIB
ENDIF

IF (ISVERTEX) THEN
  WRITE(*, '(/AS16)') ' TIME = ', TIME, ' LOOP CALL TO R20'
  CALL DOVERT(UDA,SAT,U1,B1,UMIN,U_MAX,M,NORM)
0400      RETURN
0401      ENDIF
0402
0403      IF (DIAGS) THEN
0404      WRITE(*,'(/A50:)') ' Before testing reversal'
0405      WRITE(*,'(/A30,F14.6:}') '*CA3* INFRONT = ', INFRONT
0406      WRITE(*,'(/A30,F14.6:}') '*CA3* WASINF = ', WASINF
0407      ENDIF
0408
0409      DIDSWITCH = .FALSE.
0410
0411      IF (INFRONT.NE.WASINF) THEN ! REVERSE DIRECTION
0412      IF (DIAGS) THEN
0413      WRITE(*,'(/A50:)') ' Reversing'
0414      WRITE(*,'(/A30,I3:}') '*CA3* Steps taken = ', STEPS
0415      WRITE(*,'(/A30,F14.6:}') '*CA3* Angle = ', 180.*ASIN(SINPHI)/PI
0416      WRITE(*,'(/A30,I3:}') '*CA3* MAXSTEPS = ', MAXSTEPS
0417      WRITE(*,'(/A30,I3:}') '*CA3* STUCK = ', STUCK
0418      ENDIF
0419      DIDSWITCH = .TRUE.
0420      WASINF = INFRONT
0421      ICOUNT = ICOUNT+1
0422      C Bisection and next transformation
0423      COSPHI = CSPHI(I,ICOUNT)
0424      SINPHI = INFRONT*CSPHI(2,ICOUNT)
0425      T22(1,1) = COSPHI
0426      T22(1,2) = -SINPHI
0427      T22(2,1) = SINPHI
0428      T22(2,2) = COSPHI
0429      MAXSTEPS = 2*INT(ABS(PI/ASIN(SINPHI)))
0430      IF (DIAGS) THEN
0431      WRITE(*,'(/A50:)') ' Bisection and next transformation'
0432      WRITE(*,'(/A30,E13.6:}') '*CA3* COSPHI = ', COSPHI
0433      WRITE(*,'(/A30,E13.6:}') '*CA3* SINPHI = ', SINPHI
0434      WRITE(*,'(/A30,E13.6:}') '*CA3* MAXSTEPS = ', MAXSTEPS
0435      ENDIF
0436      STEPS = 0
0437      C Check last edge with new B1
0438      Y = 0.0
0439      DO I = I,M
0440      Y = Y + BI(2,1)*XUMAX(I)
0441      ENDDO
0442      SY = 1
0443      IF (Y.LT.0.0) SY = -1
0444
0445      IF (DIAGS) THEN
0446      WRITE(*,'(/A50:)') ' Before GETEDGE'
0447      WRITE(*,'(/A30,E13.6:}') '*CA3* BI(2, :) = ', (BI(2, I), I=1,M)
0448      WRITE(*,'(/A30,E13.6:}') '*CA3* BI(3, :) = ', (BI(3, I), I=1,M)
0449      WRITE(*,'(/A30,E13.6:}') '*CA3* UMAX = ', (UMAX(I), I=1,M)
0450      WRITE(*,'(/A30,E13.6:}') '*CA3* Y = ', Y
0451      WRITE(*,'(/A30,E13.6:}') '*CA3* SY = ', SY
0452      WRITE(*,'(/A30,E13.6:}') '*CA3* ITHETA = ', (ITHETA(I), I=1,NANGS+1)
0453      WRITE(*,'(/A30,E13.6:}') '*CA3* NANGS = ', NANGS
0454      ENDIF
0455
0456      CALL GETEDGE(U2, JU, INFRONT, ISVERTEX,
& B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDX, TOL, M, DIAGS)
0459 C
0459 IF (JU.NE.0) THEN
0460 IF (INFRONT.EQ.1.) THEN
0461 DO I=1,M
0462 U_LIF(I) = U2(I)
0463 ENDDO
0464 I_LIF = JU
0465 ELSEIF (INFRONT.EQ.-1.) THEN
0466 DO I=1,M
0467 U_LIB(I) = U2(I)
0468 ENDDO
0469 I_LIB = JU
0470 ENDF
0471 IF (DIAGS) THEN
0472 WRITE(*, ' /Af0! ') ' After GETEDGE LIF/LIB'
0473 WRITE(*, 'A30,13)') ' U_LIF = ', (U_LIF(I), I=1,M)
0474 WRITE(*, 'A30,13)') ' U_LIF = ', I_LIF
0475 WRITE(*, 'A30,13)') ' U_LIB = ', (U_LIB(I), I=1,M)
0476 WRITE(*, 'A30,13)') ' U_LIB = ', I_LIB
0477 ENDF
0478 C
0479 IF (ISVERTEX) THEN
0480 WRITE(*, ' A30$') ' TIME = ', TIME, ' FOL GETEDGE'
0481 CALL DOVERT(UDA, SAT, U2, B, U_MIN, U_MAX, M, NORM)
0482 RETURN
0483 ENDF
0484 C
0485 IF (JU.NE.0) THEN
0486 CALL ISFACET(ISOK, IUOUT, JUOUT, U123,
0487 IU, JU, U1, U2, B1, M, TOL)
0488 IF (.NOT.ISOK) CALL ISFACET(ISOK, IUOUT, JUOUT, U123,
0489 I_LIF, I_LIB, U_LIF, U_LIB, B1, M, TOL)
0490 ELSE
0491 ISOK = .FALSE.
0492 ENDF
0493 C
0494 C
0495 ENDF ! IF (INFRONT.NE.WASINF) THEN
0496 C
0497 C Must leave on a switch
0498 C
0499 C
0500 C
0500 IF ((ICOUNT.EQ.NMAX).AND.(.NOT.ISOK).AND.(.NOT.DIDSWITCH)) THEN
0501 C
0502 C
0503 C
0504 IF (STUCK) THEN
0505 C
0506 C
0507 C
0508 C
0509 C
0510 END DD: End of do while statement
0511 C
0512 C END MAIN LOOP *******************************
0513 C
IF (DIAGS) THEN
WRITE(*, '(/AS0)') ' Exit from BAA'
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
C
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
C
DO I=1,3
DO J=2,3
MI23(I,J) =MI23(I,I) -MI23(I,J)
ENDDO
MTEMP(I) =MI23(I,I)
MI23(I,I) =MDES(I)
ENDDO
C
CALL INVMA3(M123,MINV,DET)
C
DO I=1,3
DO J=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
C
Call estimate subroutine to estimate solution if facet not found
C
SUBROUTINE 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)
  ENDDO
ENDDO
ENDDO

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
   XV1(I) = XK1*UPPER1(I)+(1.-XK1)*LOWER1(I)
   XV2(I) = XK2*UPPER2(I)+(1.-XK2)*LOWER2(I)
ENDDO

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

DO I = I,M
   XWI(I) = XK1*XU3(I)+(1.-XK1)*XU1(I)
   XW2(I) = XK2*XU4(I)+(1.-XK2)*XU2(I)
ENDDO

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

DO I = I,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
C ..............................................................................
WRITE(*,*) ' V_ZRI'EX
CALL SETU(UDA, U1, U_MIN, U_MAX, M)
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))
SAT = NORM/XNORM
XNORM = SAT
IF (XNORM.GT.1.) XNORM = 1.
DO I = 1,M
UDA(I) = XNORM*UDA(I)
ENDDO
RETURN
END
C ..............................................................................
C Zero out the output matrix
DO I = 1,3
DO J = 1,3
0742       MATOUT(I, J) = 0.0
0743       ENDDO
0744       ENDDO
0745 C Calculate the determinant of the input matrix
0746 C
0747 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)
0754 C
0755 C Find the matrix inverse
0756 C
0757 IF (DET.NE.0.0) THEN
0758       MATOUT(1,1) = (MATIN(2,2) * MATIN(3,3) - MATIN(2,3) * MATIN(3,2)) / DET
0759       MATOUT(1,2) = - (MATIN(1,2) * MATIN(3,3) - MATIN(1,3) * MATIN(3,2)) / DET
0760       MATOUT(1,3) = (MATIN(1,2) * MATIN(3,1) - MATIN(1,3) * MATIN(3,2)) / DET
0761       MATOUT(2,1) = - (MATIN(2,1) * MATIN(3,3) - MATIN(2,3) * MATIN(3,1)) / DET
0762       MATOUT(2,2) = (MATIN(1,1) * MATIN(3,3) - MATIN(1,3) * MATIN(3,2)) / DET
0763       MATOUT(2,3) = - (MATIN(1,1) * MATIN(2,3) - MATIN(1,3) * MATIN(2,1)) / DET
0764       MATOUT(3,1) = (MATIN(2,1) * MATIN(3,2) - MATIN(2,2) * MATIN(3,1)) / DET
0765       MATOUT(3,2) = - (MATIN(1,1) * MATIN(3,2) - MATIN(1,2) * MATIN(3,1)) / DET
0766       MATOUT(3,3) = (MATIN(1,1) * MATIN(2,2) - MATIN(1,2) * MATIN(2,1)) / DET
0767       ENDF
0768
0771       RETURN
0772       END
0773
0774 C---------------------------------------------------------------
0775 C
0776 SUBROUTINE DCGEN(T, MD)
0777 C
0778 IMPLICIT NONE
0779 REAL*4 MD(3)
0780 REAL*4 T(3,3)
0781 INTEGER*4 MLOCAL
0782 REAL*8 V(3), VNORM, XDOT_DC
0783 REAL*4 DETNUM, AMIN_DC
0784 INTEGER*4 J, JCOL, KCOL, I, MOM_FLAG, IROW
0785 REAL*8 DTOL
0786 C
0787 C Calculate the norm of the moments
0788 C
0789 DTOL = 1.D-5
0790 VNORM = DSQRT(dble(MD(1)) + dble(MD(1)))
0791 & + dble(MD(2)) * dble(MD(2))
0792 & + dble(MD(3)) * dble(MD(3))
0793 C
0794 C Normalize the desired moments
0795 C
0796 DO I = 1, 3
0797 V(I) = dble(MD(I)) / VNORM
0798 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(I,I) = V(I)
   VLEN(I) = V(I)*V(I)
ENDDO

Developing orthogonal transformation 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+1,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

IMPLICIT NONE
REAL*4 U MAX(20), U MIN(20)
INTEGER*4 IMODE, M
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 SETU(XU_SETU, IU_SETU, U_MIN, U_MAX, M)

SUBROUTINE R20(UI,IU, INFRONT, ITHETA, NANGS,INDX, ISVERTEX, & B1,UMAX,XUMAX,U_MIN,U_MAX,TOL,M,DIAGS)
IMPLICIT NONE
REAL*4 U_MIN(20), U_MAX(20)
INTEGER*4 IMODE ITHETA(21), IU, UMAX(20), M, N, NANGS, SY
INTEGER*4 UI(20)
REAL*4 PIOVR2, BI(3,20) , PI, XU(20), Y, XUMAX(20)
REAL*4 INFRONT, TOL
REAL*4 THETA(21)
LOGICAL*4 ISVERTEX, DIAGS
INTEGER*4 INDX, SSY
INTEGER*4 I, J
REAL*4 ANG, YPREV, XU1(20), K, Z, Z1, Z2
DATA PIOVR2/1.570796326794897/
DATA PI/3.141592653589793/

IF (DIAGS) THEN
  WRITE(*,'(/AS0)') ' Entering R2C'
  WRITE(*,'(/A50)') ' Calling arguments'
  WRITE(*,'(/A50)') ' R2C* B1(1,;) = ', (B1(1,I), I=1,M)
  WRITE(*,'(/A50)') ' R2C* B1(2,;) = ', (B1(2,I), I=1,M)
  WRITE(*,'(/A50)') ' R2C* B1(3,;) = ', (B1(3,I), I=1,M)
  WRITE(*,'(/A50)') ' R2C* XUMAX = ', (XUMAX(I), I=1,M)
  WRITE(*,'(/A30,6F14.6)') ' Y : ', Y, ' TOL = ', TOL
  WRITE(*,'(/A30,6F14.6)') ' SY : ', SY
  WRITE(*,'(/A30,6F14.6)') ' SY : ', SY
  WRITE(*,'(/A30,6F14.6)') ' SY : ', SY

ENDIF

C Calculate Y

Y = 0.0
DO I = 1,M
  UI(I) = UMAX(I)
  Y = Y + B1(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+B1(3,I)*XUMAX(I)
  ENDDO
  IF (ABS(Z).LT.TOL) THEN
    C WRITE(*,*) 'Z : ',Z,
    ISVERTEX = .TRUE.
    IU = 0
    NANGS = 0
    RETURN
  ENDIF
ENDIF

C Get the angle

IF (ABS(Y).LT.TOL) THEN
  SY = 0
ELSEIF (Y.LT.0.0) THEN
  SY = -1
ELSE
  SY = 1
ENDIF

IF (DIAGS) THEN
  WRITE(*,'(/A50)') ' First calculations'
  WRITE(*,'(/A50)') ' R2C* Y = ', Y
  WRITE(*,'(/A50)') ' R2C* SY = ', SY
ENDIF

C Get the angle
0970 C
0971 C
0972 IF (DIAGS) THEN
0973 WRITE(*,:'/ASC') ' Getting angles'
0974 ENDIF
0975 C
0976 NANGS = 0
0977 DO I=1, M
0978 ANG = ATAN2(-B1(I,1),B1(2,I))
0979 IF (ABS(ANG).GT.PIOVR2) THEN
0980 IF (ANG.LT.0.) THEN
0981 ANG = ANG+PI
0982 ELSE
0983 ANG = ANG-PI
0984 ENDIF
0985 ENDIF
0986 NANGS = NANGS+1
0987 THETA(NANGS) = ANG
0988 ITHETA(NANGS) = I
0989 IF (DIAGS) THEN
0990 WRITE(*,'(A30,13,A10,218.6)') '*R20* I = ', I, ', ANG = ', ANG
0991 ENDIF
0992 ENDDO
0993 C
0994 C THETA and ITHETA now sorted by control number
0995 C Sort THETA by magnitude. ITHETA gets shuffled the same way
0996 C
0997 C
0998 IF (DIAGS) THEN
0999 WRITE(*,'(AS0)') ' Before sorting'
1000 WRITE(*,'(A30,I3)') '*R20* NANGS = ', NANGS
1001 WRITE(*,'(A30,6F13.6)') '*R20* ITHETA = ', (ITHETA(I), I=1,NANGS)
1002 WRITE(*,'(A30,6F13.6)') '*R20* THETA = ', (THETA(I), I=1,NANGS)
1003 ENDIF
1004 CALL SORTC(THETA, ITHETA, NANGS, THETA, ITHETA)
1005 IF (DIAGS) THEN
1006 WRITE(*,'(AS0)') ' After sorting'
1007 WRITE(*,'(A30,I3)') '*R20* NANGS = ', NANGS
1008 WRITE(*,'(A30,6F13.6)') '*R20* ITHETA = ', (ITHETA(I), I=1,NANGS)
1009 WRITE(*,'(A30,6F13.6)') '*R20* THETA = ', (THETA(I), I=1,NANGS)
1010 ENDIF
1011 C FIND INDEX OF ZERO
1012 C
1013 DO I=1,NANGS
1014 IF (THETA(I).GT.0.) THEN
1015 INDX = I
1016 DO J=NANGS,I,-1
1017 THETA(J+1) = THETA(J)
1018 ITHETA(J+1) = ITHETA(J)
1019 ENDDO
1020 THETA(I) = 0.
1021 ITHETA(I) = 0
1022 GOTO 193
1023 ENDIF
1024 ENDDO
1025 193 CONTINUE
1026 C
IF (DIAGS) THEN
  WRITE(*, /A50/) ' After entering GETEDGE'
  WRITE(*, /A30,13/) '   *R20* INDEX = ', INDX
  WRITE(*, /A30,713/) '   *R20* THETA = ', (ITHETA(I), I=1,NANGS+1)
  WRITE(*, /A30,728/) '   *R20* ITHETA = ', (THETA(I), I=1,NANGS+1)
ENDIF

CALL GETEDGE(U1, IU, INFRONT, ISVERTEX, & B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDX, TOL, M, DIAGS)

SUBROUTINE GETEDGE(U2, JU, INFRONT, ISVERTEX, & B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDX, TOL, M, DIAGS)

IMPLICIT NONE

INTEGER*4 M, SY, UMAX(20), INDX, 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, STUCK
INTEGER*4 I, J, ICOUNT, IX
REAL*4 XK, XU2(20), YPREV, Z, Z1, Z2, SAVE_Y, MOM(3)

IF (DIAGS) THEN
  WRITE(*, /A50/) ' Exiting GETEDGE'
  WRITE(*, /A30,611/) '   *R20* U1 = ', (U1(I), I=1,M)
  WRITE(*, /A30,13/) '   *R20* IU = ', IU
  WRITE(*, /A30,611/) '   *R20* INFRONT = ', INFRONT
  WRITE(*, /A30,13/) '   *R20* ISVERTEX = ', ISVERTEX
ENDIF

RETURN
END

SUBROUTINE GETEDGE(U2, JU, INFRONT, ISVERTEX, & B1, UMAX, Y, SY, ITHETA, NANGS, U_MIN, U_MAX, INDX, TOL, M, DIAGS)
1084   WRITE(*,'(A30,E13.6)')  'GETEDGE* SAVE_Y = ', SAVE_Y
1085   WRITE(*,'(A30,I3)')  'GETEDGE* X = ', IX
1086   WRITE(*,'(A30,I3)')  'GETEDGE* SSY = ', SSY
1087   WRITE(*,'(A30,I3)')  'GETEDGE* NANGS = ', NANGS
1088   ENDIF
1089 C
1090 DO WHILE ((SY.EQ.SSY).AND.(IX.GT.0).AND.(IX.LE.NANGS))
1091   ICOUNT = ICOUNT+1
1092   YPREV = Y
1093   JU = ITHETA(IX)
1094   Y = Y-U2(JU)*B1(2,JU)*(U_MAX(JU)-U_MIN(JU))
1095   SSY = 1
1096   IF (Y.LT.0.0) SSY = -1
1097   IF (ABS(Y).LT.TOL) THEN
1098     Y = 0.
1099     SSY = 0
1100   ENDF
1101   U2(JU) = -U2(JU)
1102   IF (SY.NE.0.) THEN
1103     IX = IX-SY
1104 ELSE
1105     IX = IX-1
1106   ENDF
1107 C
1108   ICOUNT = ICOUNT+1
1109   IF (ICOUNT.GT.20) STUCK = .TRUE.
1110   IF (DIAGS) THEN
1111     WRITE(*,':A3C,ELg.6 ')  'Bottom of GETEDGE DOWHILE'
1112     WRITE(*,'(A30,I3)')  'GETEDGE* ICOUNT = ', ICOUNT
1113     WRITE(*,'(A30,I3)')  'GETEDGE* STUCK = ', STUCK
1114     WRITE(*,'(A30,E18.6)')  'GETEDGE* YPREV = ', YPREV
1115     WRITE(*,'(A30,I3)')  'GETEDGE* JU = ', JU
1116     WRITE(*,'(A30,E18.6)')  'GETEDGE* Y = ', Y
1117     WRITE(*,'(A30,I3)')  'GETEDGE* SSY = ', SSY
1118     WRITE(*,'(A30,I3)')  'GETEDGE* SY.EQ.SSY = ', (SY.EQ.SSY)
1119     WRITE(*,'(A30,I3)')  'GETEDGE* U2 = ', (U2(I), I=1,M)
1120     WRITE(*,'(A30,I3)')  'GETEDGE* IX = ', IX
1121   ENDF
1122   IF (STUCK) THEN
1123     PAUSE('Stuck in GETEDGE in DOWHILE')
1124     STUCK = .FALSE.
1125   ENDF
1126 C
1127 ENDDO
1128 C
1129 IF (SY.EQ.SSY) THEN
1130 C
1131   STUCK = .FALSE.
1132   ICOUNT = 0
1133 C
1134 IF (DIAGS) THEN
1135   WRITE(*,'(A30)')  'Beginning Alt. GETEDGE DOWHILE'
1136   WRITE(*,'(A30,I3)')  'GETEDGE* U2 = ', (U2(I), I=1,M)
1137   WRITE(*,'(A30,E18.6)')  'GETEDGE* SAVE_Y = ', SAVE_Y
1138   WRITE(*,'(A30,I3)')  'GETEDGE* IX = ', IX
1139   WRITE(*,'(A30,I3)')  'GETEDGE* SSY = ', SSY
1140   WRITE(*,'(A30,I3)')  'GETEDGE* NANGS = ', NANGS
ENDIF

Y = SAVE_Y
IX = INDX+SY
DO WHILE ((SY.EQ.SSY).AND.(IX.GT.0).AND.(IX.LE.NANGS+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.
  ENDIF
  WRITE(*,*) 'Bottom of Alt. GETEDGE DOWHILE'
ENDIF

IF (SY.EQ.SSY) THEN
  JU = 0
  INFRONT = 0.
  ISVERTEX = .FALSE.
  RETURN
ENDIF

XK = YPREV/(YPREV-Y)
CALL SETU(XU2, U2, U_MIN, U_MAX, M)

Z2 = 0.
DO I=1,M
  Z2 = Z2+B1(3,I)*XU2(I)
ENDDO
Z1 = Z2-U2(JU)*B1(3,JU)*(U_MAX(JU)-U_MIN(JU))
1198 \[ Z = \text{XK} \cdot Z2 + (1. - \text{XK}) \cdot Z1 \]
1199 \[ \text{INFRONT} = 1. \]
1200 IF \((Z \LT 0.)\) INFRONT = -1.
1201 C
1202 ISVERTEX = ((Y.EQ.0.) .AND. (ABS(Z) .LT. TOL))
1203 C
1204 IF (DIAGS) THEN
1205 WRITE(*, 'A9,2 ') 'Leaving GETEDGE'
1206 WRITE(*, 'A35,61 .') 'GETEDGE* X2 = ', (U2(I), I=1,M)
1207 WRITE(*, 'A35,614.6') 'GETEDGE* XU2 = ', (XU2(I), I=1,M)
1208 WRITE(*, 'A35,13') 'GETEDGE* JU = ', JU
1209 WRITE(*, 'A35,418.6') 'GETEDGE* XK = ', XK
1210 WRITE(*, 'A35,418.6') 'GETEDGE* Z2 = ', Z2
1211 WRITE(*, 'A35,418.6') 'GETEDGE* Z1 = ', Z1
1212 WRITE(*, 'A35,418.6') 'GETEDGE* Z = ', Z
1213 WRITE(*, 'A35,418.6') 'GETEDGE* INFRONT = ', INFRONT
1214 WRITE(*, 'A35,418.6') 'ISVERTEX = ', ISVERTEX
1215 ENDIF
1216 C
1217 RETURN
1218 END
1219
1220 C---------------------------------------------------------------
1221
1222 SUBROUTINE ISFACET(ISOK, IUOUT, JUOUT, U123, & IU, JU, U1, U2, B, M, TOL)
1223 & IU, JU, U1, U2, B, M, TOL)
1224 C
1225 IMPLICIT NONE
1226 C
1227 LOGICAL*4 ISOK
1228 INTEGER*4 IU, JU, U1(20), U2(20), IUOUT, JUOUT, U123(20,3)
1229 REAL*4 B(3,20), TOL
1230 C LOCALS
1231 INTEGER*4 UX1(20), UX2(20), I, J, K, II, JJ, KK
1232 REAL*4 THEMAT(2,2), MATINV(2,2), MATDET, T2(2), T1(3), TESTFACET(20)
1233 REAL*4 TEMP
1234 INTEGER*4 DIM, OBJ(20), UDEF(20), ITF(20), THEOBJ(20)
1235 C
1236 ISOK = .FALSE.
1237 DO I=1,M
1238 UX1(I) = U1(I)
1239 UX2(I) = U2(I)
1240 ENDDO
1241 U1(IU) = 0
1242 U2(JU) = 0
1243 DIM = 0
1244 DO I = 1,M
1245 OBJ(I) = U1(I)
1246 IF ((I.EQ.IU) .OR. (I.EQ.JU) .OR. (U2(I).NE.U1(I))) THEN
1247 OBJ(I) = 0
1248 DIM = DIM + 1
1249 UDEF(DIM) = I
1250 ENDIF
1251 ENDDO
1252 C
1253 IF (DIM.EQ.2) THEN
1254 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(2,1)*B(KK,UDEF(2))
        T2(2) = -MATINV(2,1)*B(KK,UDEF(1)) -MATINV(2,2)*B(KK,UDEF(2))
        T1(KK) = 1.
        T1(JJ) = 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
        END DO
        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
    ENDIF
    DIM = 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
    ENDDO
    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
        ENDDO
    ENDIF
ENDIF
ENDDO
1312 C
1313 IF (DIM.NE.2) ISOK = .FALSE.
1314 GOTO 194
1315 ENDIF ! IF (MATDET.NE.0.) THEN
1316 ENDDO ! DO II=1,3
1317 CONTINUE
1318 ENDIF ! IF (DIM.EQ.2)
1319
1320 IF (ISOK) THEN
1321 IUOUT = UDEF(1)
1322 JUOUT = UDEF(2)
1323 DO I=1,M
1324 DO J=1,3
1325 U123(I,J) = OBJ(I)
1326 ENDDO
1327 ENDDO
1328 IU23(IUOUT,2) = -I
1329 U123(JUOUT,2) = 1
1330 U123(IUOUT,3) = 1
1331 U123(JUOUT,3) = -I
1332 ELSE
1333 IUOUT = 0
1334 JUOUT = 0
1335 DO I=1,M
1336 DO J=1,3
1337 U123(I,J) = 0
1338 ENDDO
1339 ENDDO
1340 ENDIF
1341 RETURN
1342 END
1343
1344 SUBROUTINE SORTC(X,IY,N,XS,IYC)
1345 IMPLICIT NONE
1346 INTEGER*4 I, IL_SORTC(36), IMED, IP1, IPR, ITY, IU_SORTC(36)
1347 INTEGER*4 IY(20), IYC(20), J, JMI, JMK, K, L, MID, N
1348 REAL*4 HOLD, AMED, TX, X(20), XS(20)
1349 C
1350 C
1351 C
1352 C
1353 C
1354 C
1355 C
1356 C
1357 C
1358 C
1359 C
1360 C
1361 C
1362 IPR=11
1363 IF(N.LT.1)GOTO50
1364 IF(N.EQ.1)GOTO55
1365 IF(N.EQ.1).NE.HOLD)GOTO90
1366 60 CONTINUE
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 MID=(I+J)/2
AMED=XS(MID)
IMED=IYC(MID)
IF(XS(I).LE.AMED)GOTO320
XS(MID)=XS(I)
IYC(MID)=IYC(I)
GOTO320
310 K=1
312 MID=(I+J)/2
AMED=XS(MID)
IMED=IYC(MID)
IF(XS(I).LE.AMED)GOTO320
XS(MID)=XS(I)
IYC(MID)=IYC(I)
GOTO320
```
1426    IYC(I) = IMED
1427    AMED = XS(MID)
1428    IMED = IYC(MID)
1429      320  L = J
1430      IF (XS(J) . GE. AMED) GOTO 340
1431      XS(MID) = XS(J)
1432      IYC(MID) = IYC(J)
1433      XS(J) = AMED
1434      IYC(J) = IMED
1435      AMED = XS(MID)
1436      IMED = IYC(MID)
1437      IF (XS(I) . LE. AMED) GOTO 340
1438      XS(MID) = XS(I)
1439      IYC(MID) = IYC(I)
1440      XS(I) = AMED
1441      IYC(I) = IMED
1442      AMED = XS(MID)
1443      IMED = IYC(MID)
1444      GOTO 340
1445      330  XS(L) = XS(K)
1446      IYC(L) = IYC(K)
1447      XS(K) = TX
1448      IYC(K) = ITY
1449      340  L = L - 1
1450      IF (XS(L) . GT. AMED) GOTO 340
1451      TX = XS(L)
1452      ITY = IYC(L)
1453      350  K = K + 1
1454      IF (XS(K) . LT. AMED) GOTO 350
1455      IF (K . LE. L) GOTO 330
1456      LMI = L - I
1457      JMK = J - K
1458      IF (LMI . LE. JMK) GOTO 360
1459      IL_SORTC(M) = I
1460      IU_SORTC(M) = L
1461      I = K
1462      M = M + 1
1463      GOTO 380
1464      360  IL_SORTC(M) = K
1465      IU_SORTC(M) = J
1466      J = L
1467      M = M + 1
1468      GOTO 380
1469      370  M = M - 1
1470      IF (M . EQ. 0) RETURN
1471      I = IL_SORTC(M)
1472      J = IU_SORTC(M)
1473      380  JMI = J - I
1474      IF (JMI . GE. 11) GOTO 310
1475      IF (I . EQ. 1) GOTO 305
1476      I = I - 1
1477      390  I = I + 1
1478      IF (I . EQ. J) GOTO 370
1479      AMED = XS(I + 1)
1480      IMED = IYC(I + 1)
1481      IF (XS(I) . LE. AMED) GOTO 390
1482      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 = I.
DO I=1,M
UKDA(I) = U(I) + UDA(I)
ENDDO
DO I=1,3
M_ATT(I) =0.
DO J=1,M
M_ATT(I) = M_ATT(I) + B(I,J)*UKDA(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
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