NASA Contractor Report 188343

AN AXISYMMETRIC ANALOG TWO-LAYER CONVECTIVE HEATING PROCEDURE WITH APPLICATION TO THE EVALUATION OF SPACE SHUTTLE ORBITER WING LEADING EDGE AND WINDWARD SURFACE HEATING

K. C. Wang
*Lockheed Engineering & Sciences Company*
*Houston, Texas*

Prepared for
Lyndon B. Johnson Space Center
under Contract NAS9-17900

National Aeronautics and Space Administration

1994
Contents

1 Summary 1
2 Introduction 1
3 Inviscid Flowfield 3
4 Surface Streamlines 3
5 Metric Coefficient 4
6 Computation of Surface Streamlines and Metrics 6
7 Backward Streamline Tracing 14
8 Heating Computation 15
  8.1 BLIMP ...................................... 15
  8.2 Approximate Convective-Heating Equations .......... 15
9 Gas Properties 16
10 Results and Discussion 17
  10.1 Wind Tunnel Case .......................... 17
  10.2 STS-5 Flight Case .......................... 18
  10.3 STS-2 Flight Case .......................... 19
11 Conclusions 20
12 References 21
A Appendix A - Input Description 49
B Appendix B - AA2LCH Listing 53
List of Figures

1. Streamline coordinates ........................................... 24
2. Mapping from master element to a physical element .......... 24
3. Mapping from master element to physical domain ............ 25
4. Layout of thin-film heat transfer gauge .......................... 26
5. Pitch plane initial grid ........................................... 27
6. Pitch plane grid after outer boundary adjustment ............ 27
7. Surface streamlines ................................................ 28
8. Comparison of predicted and measured heating rates, $\alpha = 30$ deg. ...................................................... 29
9. Pressure and Heating rate distribution along the leading edge 30
10. Typical pressure distribution along a leading edge streamline 30
11. Comparison of predicted and measured heating rates, $\alpha = 40$ deg. ...................................................... 31
12. Orbiter windward surface thermocouple locations ......... 32
13. Windward centerline heating distribution, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ................................. 33
14. Circumferential heating distribution at $x/L = .1$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 34
15. Circumferential heating distribution at $x/L = .2$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 34
16. Circumferential heating distribution at $x/L = .3$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 35
17. Circumferential heating distribution at $x/L = .4$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 35
18. Circumferential heating distribution at $x/L = .5$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 36
19. Circumferential heating distribution at $x/L = .6$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 36
20. Circumferential heating distribution at $x/L = .7$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 37
21. Circumferential heating distribution at $x/L = .9$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....................... 37
22. Heating distribution on windward surface at $y = 184.8$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. .......... 38
23. Heating distribution on windward surface at $y = 233.6$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. .......... 38
Heating distribution on windward surface at $y = 275.3$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....... 39

Heating distribution on windward surface at $y = 322.7$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....... 39

Heating distribution on windward surface at $y = 369.0$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....... 40

Heating distribution on windward surface at $y = 420.8$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg. ....... 40

Heating distribution along windward centerline, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 41

Circumferential heating distribution at $x/L = .1$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 42

Circumferential heating distribution at $x/L = .2$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 42

Circumferential heating distribution at $x/L = .3$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 43

Circumferential heating distribution at $x/L = .4$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 43

Circumferential heating distribution at $x/L = .5$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 44

Circumferential heating distribution at $x/L = .6$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 44

Circumferential heating distribution at $x/L = .7$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 45

Circumferential heating distribution at $x/L = .9$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 45

Heating distribution on windward surface at $y = 184.8$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 46

Heating distribution on windward surface at $y = 233.6$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 46

Heating distribution on windward surface at $y = 275.3$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 47

Heating distribution on windward surface at $y = 322.7$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 47

Heating distribution on windward surface at $y = 369.0$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 48

Heating distribution on windward surface at $y = 420.8$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............. 48
Nomenclature

c_1, \ldots, c_5 \quad \text{Defined by Equations (108-112)}
\varepsilon_r, \varepsilon_\xi, \varepsilon_\eta \quad \text{Unit vector in streamline coordinates}
F \quad \text{Function describing body surface}
\vec{\nabla} F \quad \text{Surface norm vector}
h \quad \text{Metric coefficient}
H^* \quad \text{Eckert reference enthalpy}
H_e \quad \text{Edge enthalpy}
H_w \quad \text{Wall enthalpy}
H_{aw} \quad \text{Adiabatic wall enthalpy}
i, j, k \quad \text{Unit Cartesian vector}
m \quad \text{Defined by Equation 107}
N \quad \text{Exponent of the power law velocity profile relation}
p \quad \text{Pressure}
Pr_w \quad \text{Wall Prandtl number}
Pr_{stag} \quad \text{Stagnation pressure}
\dot{q}_L \quad \text{Laminar heating rate}
\dot{q}_T \quad \text{Turbulent heating rate}
q_w \quad \text{Surface heating rate}
Re_\theta \quad \text{Momentum thickness Reynolds number}
\vec{\varphi} \quad \text{Position vector}
s \quad \text{Distance along a streamline, measured from stagnation point}
t \quad \text{Time}
u, v, w \quad \text{Cartesian velocity components}
u_e \quad \text{Edge velocity}
V \quad \text{Total velocity}
x, y, z \quad \text{Cartesian coordinates}
\alpha, \beta \quad \text{Coordinates of computational domain}
\theta \quad \text{Momentum thickness}
\delta \quad \text{Boundary layer thickness}
\mu^* \quad \text{Viscosity defined at reference enthalpy}
\mu_e \quad \text{Edge viscosity}
\rho^* \quad \text{Density defined by reference enthalpy}
\rho_e \quad \text{Edge density}
\tau, \xi, \eta \quad \text{Streamline coordinates (see Fig. 1)}
\hat{\psi}_j \quad \text{Finite element shape function}
Acknowledgments

The author wish to thank Mr. Michael An of Lockheed Engineering & Sciences Company, for providing the CFD grids for this work. Thanks are also due to Dr. Carol Davis of Sterling Software, for providing the SAGE code and assisting the running of the code. This work was done under NASA contract No. NAS9-17900. The technical monitor of this work was Mr. Joe M. Caram of NASA Johnson Space Center.
An Axisymmetric Analog Two-Layer Convective Heating Procedure with Application to the Evaluation of Space Shuttle Orbiter Wing Leading Edge and Windward Surface Heating

1 Summary

A numerical procedure for predicting the convective heating rate of hypersonic re-entry vehicles is described. The procedure, which is based on the axisymmetric analog, consists of obtaining the three-dimensional inviscid flowfield solution; then the surface streamlines and metrics are calculated using the inviscid velocity components on the surface; finally, an axisymmetric boundary layer code or approximate convective heating equations are used to evaluate heating rates. This approach yields heating predictions to general three-dimensional body shapes.

The procedure has been applied to the prediction of the wing leading edge heating to the Space Shuttle Orbiter. The numerical results are compared with the results of heat transfer testing (OH66) of an 0.025 scale model of the Space Shuttle Orbiter configuration in the Calspan Hypersonic Shock Tunnel (HST) at Mach 10 and angles of attack of 30 and 40 degrees. Comparisons with STS-5 flight data at Mach 9.15 and angle of attack of 37.4 degrees and STS-2 flight data at Mach 12.86 and angle of attack of 39.7 degrees are also given.

2 Introduction

The calculation of aerodynamic heating on a three-dimensional body at hypersonic speeds is a challenging problem. Since wind tunnel testing cannot simulate the high temperature air environment of hypersonic flight, it is necessary to rely on computational fluid dynamic (CFD) flowfield code predictions.

Numerical methods have been developed for solving thin-layer Navier-Stokes equations over complex three-dimensional geometries to calculate aerodynamic heating[1]. But even with the most advanced modern supercomputer many hours of computer time are still required. A simpler
method to compute the viscous flow uses the "axisymmetric analog" for three-dimensional boundary layers developed by Cooke[2]. Following that approach, the general three-dimensional boundary layer equations are written in a streamline coordinate system, and the cross-flow is assumed to be small and can be neglected. This reduces the three-dimensional boundary layer equations to a form that is identical to those of axisymmetric flow, provided that the distance along a streamline is interpreted as the distance along an "equivalent body", and the metric coefficient that describes the spreading of streamlines is interpreted as the radius of an equivalent body. This allows the existing axisymmetric boundary layer codes or approximate convective heating equations to be used to compute the approximate three-dimensional heating rate along a streamline.

In order to apply the axisymmetric analog technique in computing approximate heating of three-dimensional bodies, the inviscid surface streamline paths and the metric coefficients need to be computed. DeJarnette and Davis[3] calculated the streamlines as the lines of steepest descent emanating from the stagnation point. DeJarnette and Hamilton[4] developed a simple method for calculating streamlines from a known pressure distribution. However, this approach has proven difficult to apply, unless the surface geometry and pressures can be described analytically. More success has been achieved in using the three-dimensional inviscid flowfield solution to compute surface streamlines and metric coefficients[5-9]. But, the disadvantage of this approach is that it requires more computer time than the engineering approximate methods described in references 3 and 4. The majority of the computer time is spent in obtaining the inviscid flowfield solution.

Previous streamline codes[5-9] use three-dimensional flowfield predictions described either in a spherical or cylindrical coordinate system. However, most recent CFD flowfield solutions use either Cartesian or generalized coordinates. In this work, the Inviscid Equilibrium Computation in 3-Dimensions (IEC3D), a three-dimensional, shock-capturing, inviscid CFD code[10], was used to compute the inviscid flowfield solution. A streamline code, which uses surface velocity components in Cartesian coordinates as input, has been developed to trace streamline paths and compute metric coefficients along the path. A boundary layer code, the Boundary Layer Integral Matrix Procedure (BLIMP)[11], was used to evaluate the heating. Because of the failure of convergence while using the BLIMP code to evaluate heating for the flight case, approximate convective heating equations
developed by Zoby et al.[12] were used to obtain heating rates. This code has been designated as the Axisymmetric Analog 2-Layer Convective Heating (AA2LCH) code. A sample input is shown in Appendix A, and the listing of the program is provided in Appendix B.

This document will provide a detailed description of the numerical procedure used. The procedure is applied to an 0.025 scale Shuttle Orbiter at a wind tunnel condition of Mach 10 and angles of attack of 30 and 40 degrees. The predicted heating rates are compared with experimental results, obtained in the OH66[21] test. The procedure is also applied to a trajectory point of the STS-5 flight at an altitude of 159,000 ft, Mach number of 9.15 and angle of attack of 37.4 degrees, and a trajectory point of the STS-2 flight at an altitude of 179,920 ft, Mach number of 12.86 and angle of attack of 39.7 degrees.

3 Inviscid Flowfield

A three-dimensional, shock capturing, inviscid CFD code, IEC3D[10], was used to compute the inviscid flowfield solution. The IEC3D code is a general purpose three-dimensional Euler solution CFD code, which can compute inviscid flowfield solutions around general three-dimensional geometries at a wide range of flight conditions and angles-of-attack. This code utilizes an upwind-biased, finite-volume, high order Total Variation Diminishing (TVD) scheme. Both Van Leer's Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) type flux-vector splitting[13-14] and Roe's characteristic-based flux-difference splitting[15] are considered. An improved implicit solution algorithm called Lower-Upper Symmetric Gauss-Seidel (LU-SGS)[16] has also been incorporated. This code has been validated by wind tunnel and flight data[17].

4 Surface Streamlines

Streamlines in a flow are defined as lines that, at any instant, are tangent to the velocity vector. The streamline equation in Cartesian coordinates can be written as

\[
\frac{dx}{dt} = u
\]

(1)
\[
\frac{dy}{dt} = v \\
\frac{dz}{dt} = w
\]  
(2)  
(3)

Starting at a initial location \((x_0, y_0, z_0)\), equations (1) through (3) can be integrated to obtain streamline locations.

5 Metric Coefficient

A location \((x, y, z)\) can be written in Cartesian coordinates as

\[
\vec{r} = xi + yj + zk
\]  
(4)

Let \((\xi, \eta, \tau)\) represent an orthogonal curvilinear coordinate system, where \(\xi\) is in the direction of the streamline, \(\eta\) is everywhere normal to the streamline on the body surface and \(\tau\) is normal to the body surface as shown in Figure 1. On the body surface we have

\[
\vec{r} = \xi \vec{\xi} + \eta \vec{\eta}
\]  
(5)

where \(\vec{\xi}\) and \(\vec{\eta}\) are unit vectors in the \(\xi\) and \(\eta\) direction, respectively. The unit tangent vector \(\vec{\eta}\) can be written as

\[
\vec{\eta} = \frac{\frac{\partial \vec{r}}{\partial \eta}}{|\frac{\partial \vec{r}}{\partial \eta}|}
\]  
\[
= \frac{\frac{\partial x}{\partial \eta} \vec{i} + \frac{\partial y}{\partial \eta} \vec{j} + \frac{\partial z}{\partial \eta} \vec{k}}{|\frac{\partial \vec{r}}{\partial \eta}|}
\]  
(6)

where \(\left|\frac{\partial \vec{r}}{\partial \eta}\right|\) is the metric coefficient. Since \((\xi, \eta, \tau)\) represent orthogonal coordinates, therefore we have

\[
\vec{\eta} = \vec{\tau} \times \vec{\xi}
\]  
(7)

The velocity on the surface of the body can be written as

\[
\vec{V} = V \vec{\xi}
\]  
\[
= u \vec{i} + v \vec{j} + w \vec{k}
\]  
(8)
where \( u, v, w \) are Cartesian surface velocity components. Therefore

\[
\varepsilon_t = \frac{u}{V} \mathbf{i} + \frac{v}{V} \mathbf{j} + \frac{w}{V} \mathbf{k}
\]

(9)

The points on the body can be described as \( z = x(y, z) \). Therefore, the body surface can be described as

\[
F(x, y, z) = x - x(y, z) = 0
\]

(10)

The unit normal vector to the body surface can then be written as

\[
\varepsilon_r = \frac{\nabla F}{|\nabla F|} = \tau_x \mathbf{i} + \tau_y \mathbf{j} + \tau_z \mathbf{k}
\]

(11)

where

\[
\nabla F = \mathbf{i} - \frac{\partial x}{\partial y} \mathbf{j} - \frac{\partial x}{\partial z} \mathbf{k}
\]

(12)

and

\[
|\nabla F| = \left[ 1 + \left( \frac{\partial x}{\partial y} \right)^2 + \left( \frac{\partial x}{\partial z} \right)^2 \right]^{\frac{1}{2}}
\]

(13)

Equation (7) can now be written as

\[
\varepsilon_\eta = \begin{vmatrix}
\mathbf{i} & \mathbf{j} & \mathbf{k} \\
\tau_x & \tau_y & \tau_z \\
\frac{\partial z}{\partial n} & \frac{\partial y}{\partial n} & \frac{\partial x}{\partial n} \\
\end{vmatrix} \\
= \left( \frac{u \frac{\partial z}{\partial n} - v \frac{\partial x}{\partial n}}{\nabla F} \right) \mathbf{i} - \left( \frac{\nabla F}{V} + \frac{u}{V} \frac{\partial z}{\partial n} \right) \mathbf{j} + \left( \frac{\nabla F}{V} + \frac{v}{V} \frac{\partial x}{\partial n} \right) \mathbf{k}
\]

(14)

From Equations (6) and (14) we have

\[
\frac{\partial y}{\partial n} = -\frac{1}{|\nabla F|} \left( \frac{w}{V} + \frac{u}{V} \frac{\partial z}{\partial n} \right)
\]

(15)
and
\[ \frac{\partial z}{\partial \eta} = \frac{1}{|\nabla F|} \left( \frac{v + u \partial x}{\sqrt{V^2 + V \partial y}} \right) \]  
(16)

Let \( h = \left| \frac{\partial x}{\partial \eta} \right| = \) metric coefficient, then
\[ \frac{\partial y}{\partial \eta} = -\frac{h}{|\nabla F|} \left( \frac{w + u \partial x}{V + V \partial z} \right) \]  
(17)

and
\[ \frac{\partial z}{\partial \eta} = \frac{h}{|\nabla F|} \left( \frac{v + u \partial x}{V + V \partial y} \right) \]  
(18)

Combining Equations (17) and (18), we have
\[ h = \left| \nabla F \right| \frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \eta} \]  
(19)

Equation (19) can now be used to compute the metric coefficient, \( h \), along the streamline. From DeJarnette[18], on the body surface
\[ \frac{d}{dt} \left( \frac{\partial y}{\partial \eta} \right) = \frac{\partial}{\partial \eta} \left( \frac{dy}{dt} \right) = \frac{\partial v}{\partial \eta} = \frac{\partial v}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial v}{\partial z} \frac{\partial z}{\partial \eta} \]  
(20)

\[ \frac{d}{dt} \left( \frac{\partial z}{\partial \eta} \right) = \frac{\partial}{\partial \eta} \left( \frac{dz}{dt} \right) = \frac{\partial w}{\partial \eta} = \frac{\partial w}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial w}{\partial z} \frac{\partial z}{\partial \eta} \]  
(21)

For a given \( h \), initial values of \( \frac{\partial x}{\partial \eta} \) and \( \frac{\partial z}{\partial \eta} \) can be obtained from equations (17) and (18). Equations (20) and (21) can then be integrated to obtain \( \frac{\partial y}{\partial \eta} \) and \( \frac{\partial z}{\partial \eta} \) along the streamline. It can be shown that the heating rate is independent of the initial value of \( h \).

6 Computation of Surface Streamlines and Metrics

A fourth order variable step Runge-Kutta integrator was used to integrate equations (1) through (3), (20) and (21). In order to integrate equations (20) and (21) we need to know the values of \( \frac{\partial v}{\partial y}, \frac{\partial w}{\partial x}, \frac{\partial v}{\partial y}, \) and \( \frac{\partial w}{\partial z} \) at each integration step. To compute \( |\nabla F| \), we need the values of \( \frac{\partial x}{\partial y} \) and \( \frac{\partial z}{\partial z} \) also at each integration step. All of these values can be computed numerically from the coordinates and surface velocity components at each grid point.
Consider a master element \( \hat{\Omega} \) and a mapping to an element \( \Omega_e \) in the physical domain, as shown in Figure 2. Figure 3 shows the mapping from a master element to the entire physical domain. By using the finite element shape functions, the mapping from the \((\alpha, \beta)\) domain to the physical \((x, y, z)\) domain can be described as

\[
x = \sum_{j=1}^{n} x_j \hat{\psi}_j(\alpha, \beta) \tag{22}
\]

\[
y = \sum_{j=1}^{n} y_j \hat{\psi}_j(\alpha, \beta) \tag{23}
\]

\[
z = \sum_{j=1}^{n} z_j \hat{\psi}_j(\alpha, \beta) \tag{24}
\]

Here \((x_j, y_j, z_j)\) are the \(x, y, z\) coordinates of a local nodal point \(j\) in element \(\Omega_e\), \(n\) is the total number of nodal points of the element and \(\hat{\psi}_j\) is the shape function at nodal point \(j\) of the master element \(\hat{\Omega}\). Referring to Figure 2, the shape function at nodal points of each sub-element of the master element are Element I:

\[
\hat{\psi}_1 = \alpha \beta \tag{25}
\]

\[
\hat{\psi}_2 = -(1 + \alpha)\beta \tag{26}
\]

\[
\hat{\psi}_5 = (1 + \alpha)(1 + \beta) \tag{27}
\]

\[
\hat{\psi}_4 = -(1 + \beta)\alpha \tag{28}
\]

Element II:

\[
\hat{\psi}_2 = -(1 - \alpha)\beta \tag{29}
\]

\[
\hat{\psi}_3 = -\alpha \beta \tag{30}
\]

\[
\hat{\psi}_6 = \alpha(1 + \beta) \tag{31}
\]

\[
\hat{\psi}_5 = (1 - \alpha)(1 + \beta) \tag{32}
\]

Element III:

\[
\hat{\psi}_5 = (1 - \alpha)(1 - \beta) \tag{33}
\]

\[
\hat{\psi}_6 = \alpha(1 - \beta) \tag{34}
\]

\[
\hat{\psi}_9 = \alpha \beta \tag{35}
\]

\[
\hat{\psi}_8 = (1 - \alpha)\beta \tag{36}
\]
Element IV:

\[
\begin{align*}
\hat{\Psi}_4 &= -\alpha(1 - \beta) \\
\hat{\Psi}_5 &= (1 + \alpha)(1 - \beta) \\
\hat{\Psi}_8 &= (1 + \alpha)\beta \\
\hat{\Psi}_7 &= -\alpha\beta
\end{align*}
\]  

(37)  

(38)  

(39)  

(40)

The \(x\) coordinate of a point in the physical domain, which is the mapping of sub-element I, can be written as

\[
x = x_1\hat{\Psi}_1 + x_2\hat{\Psi}_2 + x_5\hat{\Psi}_5 + x_4\hat{\Psi}_4
\]

\[
= x_1\alpha\beta - x_2(1 + \alpha)\beta + x_5(1 + \alpha)(1 + \beta) - x_4(1 + \beta)\alpha
\]

(41)

The derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) are

\[
\frac{\partial x}{\partial \alpha} = x_1\beta - x_2\beta + x_5(1 + \beta) - x_4(1 + \beta)
\]

(42)

\[
\frac{\partial x}{\partial \beta} = x_1\alpha - x_2(1 + \alpha) + x_5(1 + \alpha) - x_4\alpha
\]

(43)

At point 5 of element I (\(\alpha = 0, \beta = 0\)) in the physical domain, the derivatives are

\[
\left.\frac{\partial x}{\partial \alpha}\right|_5 = x_5 - x_4
\]

(44)

\[
\left.\frac{\partial x}{\partial \beta}\right|_5 = -x_2 + x_5
\]

(45)

The \(x\) coordinate of a point in the physical domain, which is the mapping of sub-element II, can be written as

\[
x = x_2\hat{\Psi}_2 + x_3\hat{\Psi}_3 + x_6\hat{\Psi}_6 + x_5\hat{\Psi}_5
\]

\[
= -x_2(1 - \alpha)\beta - x_3\alpha\beta + x_6\alpha(1 + \beta) + x_5(1 - \alpha)(1 + \beta)
\]

(46)

The derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) are

\[
\frac{\partial x}{\partial \alpha} = x_2\beta - x_3\beta + x_6(1 + \beta) - x_5(1 + \beta)
\]

(47)

\[
\frac{\partial x}{\partial \beta} = -x_2(1 - \alpha) - x_3\alpha + x_6\alpha + x_5(1 - \alpha)
\]

(48)
At point 5 of element II \((\alpha = 0, \beta = 0)\) in the physical domain, the derivatives are

\[
\frac{\partial x}{\partial \alpha}|_5 = x_6 - x_5 \quad (49)
\]
\[
\frac{\partial x}{\partial \beta}|_5 = -x_2 + x_5 \quad (50)
\]

The \(x\) coordinate of a point in the physical domain which is the mapping of sub-element III, can be written as

\[
x = x_5 \hat{y}_5 + x_6 \hat{y}_6 + x_9 \hat{y}_9 + x_8 \hat{y}_8
\]

\[
= x_5(1 - \alpha)(1 - \beta) + x_6\alpha(1 - \beta) + x_9\alpha\beta + x_8(1 - \alpha)\beta \quad (51)
\]

The derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) are

\[
\frac{\partial x}{\partial \alpha} = -x_5(1 - \beta) + x_6(1 - \beta) + x_9\beta - x_8\beta \quad (52)
\]
\[
\frac{\partial x}{\partial \beta} = -x_5(1 - \alpha) - x_5\alpha + x_9\alpha + x_8(1 - \alpha) \quad (53)
\]

At point 5 of element III \((\alpha = 0, \beta = 0)\) in the physical domain, the derivatives are

\[
\frac{\partial x}{\partial \alpha}|_5 = -x_5 + x_6 \quad (54)
\]
\[
\frac{\partial x}{\partial \beta}|_5 = -x_5 + x_8 \quad (55)
\]

The \(x\) coordinate of a point in the physical domain which is the mapping of sub-element IV, can be written as

\[
x = x_4 \hat{y}_4 + x_5 \hat{y}_5 + x_8 \hat{y}_8 + x_7 \hat{y}_7
\]

\[
= -x_4\alpha(1 - \beta) + x_5(1 + \alpha)(1 - \beta) + x_8(1 + \alpha)\beta - x_7\alpha\beta \quad (56)
\]

The derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) are

\[
\frac{\partial x}{\partial \alpha} = -x_4(1 - \beta) + x_5(1 - \beta) + x_8\beta - x_7\beta \quad (57)
\]
\[
\frac{\partial x}{\partial \beta} = x_4\alpha - x_5(1 + \alpha) + x_8(1 + \alpha) - x_7\alpha \quad (58)
\]
At point 5 of element IV \((\alpha = 0, \beta = 0)\) in the physical domain, the derivatives are

\[
\frac{\partial x}{\partial \alpha}|_5 = -x_4 + x_5 \quad (59)
\]

\[
\frac{\partial x}{\partial \beta}|_5 = -x_5 + x_8 \quad (60)
\]

Therefore, the derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) at point 5 in the physical domain are

\[
\frac{\partial x}{\partial \alpha}|_5 = \frac{1}{4}(x_5 - x_4 + x_6 - x_5 - x_5 + x_6 - x_4 + x_5)
\]

\[
= \frac{1}{2}(x_6 - x_4) \quad (61)
\]

\[
\frac{\partial x}{\partial \beta}|_5 = \frac{1}{4}(-x_2 + x_5 - x_2 + x_5 - x_5 + x_8 - x_5 + x_8)
\]

\[
= \frac{1}{2}(x_8 - x_2) \quad (62)
\]

Similarly, we have

\[
\frac{\partial y}{\partial \alpha} = \frac{1}{2}(y_6 - y_4) \quad (63)
\]

\[
\frac{\partial y}{\partial \beta} = \frac{1}{2}(y_8 - y_2) \quad (64)
\]

\[
\frac{\partial z}{\partial \alpha} = \frac{1}{2}(z_6 - z_4) \quad (65)
\]

\[
\frac{\partial z}{\partial \beta} = \frac{1}{2}(z_8 - z_2) \quad (66)
\]

The partial derivatives of \(x, y, z\) with respect to \(\alpha, \beta\) at all interior grid points can then be obtained from Eqs. (61) through (66). For points on the boundary, partial derivatives can be found as follows: Refering to Figure 2, for points on the left boundary, only elements II and III exist. The derivatives \(\frac{\partial x}{\partial \alpha}\) and \(\frac{\partial x}{\partial \beta}\) at point 5 in the physical domain are

\[
\frac{\partial x}{\partial \alpha}|_5 = \frac{1}{2}(x_6 - x_5 + x_6 - x_5)
\]

\[
= x_6 - x_5 \quad (67)
\]

\[
\frac{\partial x}{\partial \beta}|_5 = \frac{1}{2}(-x_2 + x_5 - x_5 + x_8)
\]

\[
= \frac{1}{2}(x_8 - x_2) \quad (68)
\]

10
For points on the right boundary, only elements I and IV exist. The derivatives $\frac{\partial x}{\partial \alpha}$ and $\frac{\partial x}{\partial \beta}$ at point 5 in the physical domain are

$$\left. \frac{\partial x}{\partial \alpha} \right|_5 = \frac{1}{2}(x_5 - x_4 - x_4 + x_5) = x_5 - x_4$$

(69)

$$\left. \frac{\partial x}{\partial \beta} \right|_5 = \frac{1}{2}(-x_2 + x_5 - x_5 + x_8) = \frac{1}{2}(x_8 - x_2)$$

(70)

For points on the upper boundary, only elements I and II exist. The derivatives $\frac{\partial x}{\partial \alpha}$ and $\frac{\partial x}{\partial \beta}$ at point 5 are

$$\left. \frac{\partial x}{\partial \alpha} \right|_5 = \frac{1}{2}(x_5 - x_4 + x_6 - x_5) = \frac{1}{2}(x_6 - x_4)$$

(71)

$$\left. \frac{\partial x}{\partial \beta} \right|_5 = \frac{1}{2}(-x_2 + x_5 - x_2 + x_5) = x_5 - x_2$$

(72)

For points on the lower boundary, only elements III and IV exist. The derivatives $\frac{\partial x}{\partial \alpha}$ and $\frac{\partial x}{\partial \beta}$ at point 5 are

$$\left. \frac{\partial x}{\partial \alpha} \right|_5 = \frac{1}{2}(-x_5 + x_6 - x_4 + x_5) = \frac{1}{2}(x_6 - x_4)$$

(73)

$$\left. \frac{\partial x}{\partial \beta} \right|_5 = \frac{1}{2}(-x_5 + x_8 - x_5 + x_8) = x_8 - x_5$$

(74)

Similarly, we can calculate $\frac{\partial y}{\partial \alpha}$, $\frac{\partial y}{\partial \beta}$, $\frac{\partial z}{\partial \alpha}$ and $\frac{\partial z}{\partial \beta}$ at all boundary points. By using similar finite element mappings, we can calculate $\frac{\partial y}{\partial \alpha}$, $\frac{\partial y}{\partial \beta}$, $\frac{\partial z}{\partial \alpha}$, $\frac{\partial z}{\partial \beta}$, $\frac{\partial w}{\partial \alpha}$ and $\frac{\partial w}{\partial \beta}$ at all grid points.

From the mapping, we have $x = x(\alpha, \beta)$, $y = y(\alpha, \beta)$ and $z = z(\alpha, \beta)$. Therefore

$$dx = \frac{\partial x}{\partial \alpha} d\alpha + \frac{\partial x}{\partial \beta} d\beta$$

(75)
\[ dy = \frac{\partial y}{\partial \alpha} d\alpha + \frac{\partial y}{\partial \beta} d\beta \]  
\( (76) \)

\[ dz = \frac{\partial z}{\partial \alpha} d\alpha + \frac{\partial z}{\partial \beta} d\beta \]  
\( (77) \)

or

\[
\begin{bmatrix}
    dx \\
    dy \\
    dz
\end{bmatrix} = \begin{bmatrix}
    \frac{\partial x}{\partial \alpha} & \frac{\partial x}{\partial \beta} \\
    \frac{\partial y}{\partial \alpha} & \frac{\partial y}{\partial \beta} \\
    \frac{\partial z}{\partial \alpha} & \frac{\partial z}{\partial \beta}
\end{bmatrix} \begin{bmatrix}
    d\alpha \\
    d\beta
\end{bmatrix}
\]  
\( (78) \)

or

\[
\begin{bmatrix}
    \frac{\partial x}{\partial \alpha} & \frac{\partial x}{\partial \beta} \\
    \frac{\partial y}{\partial \alpha} & \frac{\partial y}{\partial \beta} \\
    \frac{\partial z}{\partial \alpha} & \frac{\partial z}{\partial \beta}
\end{bmatrix} \begin{bmatrix}
    dx \\
    dy \\
    dz
\end{bmatrix} = \begin{bmatrix}
    (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 \\
    (\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2 \\
    \frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}
\end{bmatrix} \begin{bmatrix}
    d\alpha \\
    d\beta
\end{bmatrix}
\]  
\( (79) \)

Assuming there exists an inverse mapping, then we have \( \alpha = \alpha(x, y, z) \) and \( \beta = \beta(x, y, z) \). Therefore

\[ d\alpha = \frac{\partial \alpha}{\partial x} dx + \frac{\partial \alpha}{\partial y} dy + \frac{\partial \alpha}{\partial z} dz \]  
\( (80) \)

\[ d\beta = \frac{\partial \beta}{\partial x} dx + \frac{\partial \beta}{\partial y} dy + \frac{\partial \beta}{\partial z} dz \]  
\( (81) \)

or

\[
\begin{bmatrix}
    d\alpha \\
    d\beta
\end{bmatrix} = \begin{bmatrix}
    \frac{\partial \alpha}{\partial x} & \frac{\partial \alpha}{\partial y} & \frac{\partial \alpha}{\partial z} \\
    \frac{\partial \beta}{\partial x} & \frac{\partial \beta}{\partial y} & \frac{\partial \beta}{\partial z}
\end{bmatrix} \begin{bmatrix}
    dx \\
    dy \\
    dz
\end{bmatrix}
\]  
\( (82) \)

from Equation (79)

\[
\begin{bmatrix}
    d\alpha \\
    d\beta
\end{bmatrix} = \frac{1}{|J|} \begin{bmatrix}
    (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 & -(\frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}) \\
    -(\frac{\partial x}{\partial \beta} \frac{\partial x}{\partial \alpha} + \frac{\partial y}{\partial \beta} \frac{\partial y}{\partial \alpha} + \frac{\partial z}{\partial \beta} \frac{\partial z}{\partial \alpha}) & (\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2
\end{bmatrix} \begin{bmatrix}
    dx \\
    dy \\
    dz
\end{bmatrix}
\]  
\( (83) \)
where \( |J| \) is called the Jacobian of the transformation and is equal to the ratio of areas of element at physical plane to that at \((\alpha, \beta)\) plane. \( |J| \) must be greater than 0 to ensure the inverse mapping exists.

\[
|J| = \left[ \left( \frac{\partial x}{\partial \alpha} \right)^2 + \left( \frac{\partial y}{\partial \alpha} \right)^2 + \left( \frac{\partial z}{\partial \alpha} \right)^2 \right] \left[ \left( \frac{\partial x}{\partial \beta} \right)^2 + \left( \frac{\partial y}{\partial \beta} \right)^2 + \left( \frac{\partial z}{\partial \beta} \right)^2 \right] - \left( \frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta} \right)^2
\]

(84)

From equations (82) and (83) we have

\[
\frac{\partial \alpha}{\partial x} = \frac{1}{|J|} \left[ \frac{\partial x}{\partial \alpha} \left( \frac{\partial x}{\partial \alpha} \right)^2 + \left( \frac{\partial y}{\partial \alpha} \right)^2 + \left( \frac{\partial z}{\partial \alpha} \right)^2 \right] - \frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}
\]

(85)

\[
\frac{\partial \alpha}{\partial y} = \frac{1}{|J|} \left[ \frac{\partial y}{\partial \alpha} \left( \frac{\partial x}{\partial \alpha} \right)^2 + \left( \frac{\partial y}{\partial \alpha} \right)^2 + \left( \frac{\partial z}{\partial \alpha} \right)^2 \right] - \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}
\]

(86)

\[
\frac{\partial \alpha}{\partial z} = \frac{1}{|J|} \left[ \frac{\partial z}{\partial \alpha} \left( \frac{\partial x}{\partial \alpha} \right)^2 + \left( \frac{\partial y}{\partial \alpha} \right)^2 + \left( \frac{\partial z}{\partial \alpha} \right)^2 \right] - \frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}
\]

(87)

\[
\frac{\partial \beta}{\partial x} = \frac{1}{|J|} \left[ \frac{\partial x}{\partial \beta} \left( \frac{\partial x}{\partial \beta} \right)^2 + \left( \frac{\partial y}{\partial \beta} \right)^2 + \left( \frac{\partial z}{\partial \beta} \right)^2 \right] - \frac{\partial x}{\partial \beta} \frac{\partial x}{\partial \alpha} + \frac{\partial y}{\partial \beta} \frac{\partial y}{\partial \alpha} + \frac{\partial z}{\partial \beta} \frac{\partial z}{\partial \alpha}
\]

(88)

\[
\frac{\partial \beta}{\partial y} = \frac{1}{|J|} \left[ \frac{\partial y}{\partial \beta} \left( \frac{\partial x}{\partial \beta} \right)^2 + \left( \frac{\partial y}{\partial \beta} \right)^2 + \left( \frac{\partial z}{\partial \beta} \right)^2 \right] - \frac{\partial y}{\partial \beta} \frac{\partial y}{\partial \alpha} + \frac{\partial z}{\partial \beta} \frac{\partial z}{\partial \alpha}
\]

(89)

\[
\frac{\partial \beta}{\partial z} = \frac{1}{|J|} \left[ \frac{\partial z}{\partial \beta} \left( \frac{\partial x}{\partial \beta} \right)^2 + \left( \frac{\partial y}{\partial \beta} \right)^2 + \left( \frac{\partial z}{\partial \beta} \right)^2 \right] - \frac{\partial x}{\partial \beta} \frac{\partial x}{\partial \alpha} + \frac{\partial y}{\partial \beta} \frac{\partial y}{\partial \alpha} + \frac{\partial z}{\partial \beta} \frac{\partial z}{\partial \alpha}
\]

(90)

Now, the derivatives \( \frac{\partial \alpha}{\partial y}, \frac{\partial \beta}{\partial \alpha}, \frac{\partial \beta}{\partial \gamma}, \frac{\partial \gamma}{\partial \alpha}, \frac{\partial \gamma}{\partial \beta}, \frac{\partial \gamma}{\partial \zeta}, \frac{\partial \zeta}{\partial \alpha}, \frac{\partial \zeta}{\partial \beta}, \frac{\partial \zeta}{\partial \gamma}, \frac{\partial \gamma}{\partial \zeta}, \frac{\partial \zeta}{\partial \gamma}, \frac{\partial \gamma}{\partial \zeta}, \frac{\partial \zeta}{\partial \gamma}, \frac{\partial \zeta}{\partial \gamma}, \frac{\partial \gamma}{\partial \zeta} \) and \( \frac{\partial \zeta}{\partial \gamma} \) can be computed.

\[
\frac{\partial x}{\partial y} = \frac{\partial x}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial x}{\partial \beta} \frac{\partial \beta}{\partial y}
\]

(91)

\[
\frac{\partial x}{\partial z} = \frac{\partial x}{\partial \alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial x}{\partial \beta} \frac{\partial \beta}{\partial z}
\]

(92)

\[
\frac{\partial \alpha}{\partial y} = \frac{\partial \alpha}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial \alpha}{\partial \beta} \frac{\partial \beta}{\partial y}
\]

(93)

\[
\frac{\partial \alpha}{\partial z} = \frac{\partial \alpha}{\partial \alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial \alpha}{\partial \beta} \frac{\partial \beta}{\partial z}
\]

(94)

\[
\frac{\partial \beta}{\partial y} = \frac{\partial \beta}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial \beta}{\partial \beta} \frac{\partial \beta}{\partial y}
\]

(95)

\[
\frac{\partial \beta}{\partial z} = \frac{\partial \beta}{\partial \alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial \beta}{\partial \beta} \frac{\partial \beta}{\partial z}
\]

(96)
The procedure for tracing streamlines and computing metrics is: 1) obtain $\frac{\partial u}{\partial y}, \frac{\partial v}{\partial z}, \frac{\partial u}{\partial x}, \frac{\partial v}{\partial y}, \frac{\partial w}{\partial z}$ and $\frac{\partial w}{\partial x}$ at all grid points, 2) integrate equations (1) through (3), (20) and (21), and 3) use (19) to compute the metric. The streamline path does not always fall on the grid point, therefore interpolation is required to obtain velocity components and derivatives along the path in order to perform the numerical integration. A finite element based linear interpolation scheme is used. Consider a master element as shown in Figure 3. The shape function at nodal points of the master element are

$$
\hat{\psi}_1 = (1 - \alpha)(1 - \beta) \\
\hat{\psi}_2 = \alpha(1 - \beta) \\
\hat{\psi}_3 = \alpha \beta \\
\hat{\psi}_4 = (1 - \alpha)\beta
$$

Equations (22) through (24) describe the mapping from the master element to a physical cell. By knowing the $x, y, z$ and $x_j, y_j, z_j$ values, equations (22) to (24) represent a system of non-linear equations of unknown $\alpha$ and $\beta$, which can be solved by using either Newton's method or successive approximation. The finite element shape function can then be used to interpolate surface velocity components and derivatives required for integration.

7 Backward Streamline Tracing

Surface streamlines emanate from the stagnation point and spread all over the body surface. It is very difficult to start a streamline path from the stagnation point and have it pass through a specific point on the body. Since we are interested in evaluating heating rates at specific locations on the leading edge, a procedure was developed to trace a streamline in a backward fashion, i.e. starting at a specific location and tracing the streamline toward the stagnation point. This can be done easily by reversing the sign of velocity components while integrating equations (1) through (3). Streamline coordinates and the integration step size are saved at each integration step. The process can then be reversed at a point very close to the stagnation point (the stagnation point can never be reached) and re-trace the streamline in a forward fashion and compute metric coefficients along the streamline path until the point of interest is reached. The streamline location, metric coefficient and pressure distribution along the streamline can then be used to evaluate the heating rate.
8 Heating Computation

8.1 BLIMP

The BLIMP code[11] determines the viscous flow field across the boundary layer around a blunt body for either an axisymmetric or planar shape. This code requires as input the distance from the stagnation point, metric coefficient and pressure ratio and computes the heating rate. Ideal gas, equilibrium flow and reacting gas chemistry may be used with the code. Convergence problems were encountered while using the code to evaluate wing leading edge heating rate for the flight case. The difficulty was associated with the rapid pressure rise close to the wing leading edge. Therefore the BLIMP code was only used to obtain the heating rate for the wind tunnel cases.

8.2 Approximate Convective-Heating Equations

The approximate convective-heating equations developed by Zoby et al[12] were used to obtain heating rates for the flight cases. These heating rate relations, valid for both laminar and turbulent flow, have been shown to yield results which compare favorably with the more exact solution obtained from the BLIMP code for both nonreacting and reacting gas mixtures for either constant or variable entropy edge conditions.

The laminar heating rate is computed from an equation which relates heating rate to the momentum thickness Reynolds number

$$\dot{q}_L = 0.22(Re_{\theta,e})^{-1}(\rho^*/\rho)(\mu^*/\mu)\rho_e u_e (H_{aw} - H_w)(Pr_w)^{-0.6} \quad (101)$$

where \((^*)\) quantities are computed using Eckert reference enthalpy relation, subscripts \(w, e\) and \(aw\) represent wall, edge and adiabatic wall respectively, and \(\theta_L\), used to compute the momentum thickness Reynolds number, \(Re_{\theta}\), is given by the equation

$$\theta_L = 0.644(\int_0^s \rho^* \mu^* u_e h^2 dS)^{1/2}/(\mu_e h) \quad (102)$$

where \(h\) is the metric coefficient, and \(s\) is the distance along a streamline. The Eckert's reference enthalpy relation is given by

$$H^* = 0.5(H_w + H_e) + 0.22(H_{aw} - H_e) \quad (103)$$
The boundary-layer thickness is given approximately by the equation

\[
\frac{\delta}{\theta} L = 5.55 \tag{104}
\]

The turbulent heating is also computed from an equation that relates turbulent heating to momentum thickness Reynolds number

\[
\dot{q}_T = c_1 (Re_{\theta,e})^{-m} \left( \frac{\rho^*}{\rho_e} \left( \frac{\mu^*}{\mu_e} \right)^m \rho_e u_e (H_{aw} - H_w) (Pr_w)^{-0.4} \right) \tag{105}
\]

and

\[
\theta_T = \left( c_2 \int_0^{\delta} \frac{\rho^* u_e (\mu^*)^m h^2 dS}{\rho_e u_e h} \right)^{c_4} \tag{106}
\]

\[
m = 2/(N + 1) \tag{107}
\]

\[
c_1 = (1/c_5)^{2N/(N+1)} [N/(N + 1)(N + 2)]^m \tag{108}
\]

\[
c_2 = (1 + m)c_1 \tag{109}
\]

\[
c_3 = (1 + m) \tag{110}
\]

\[
c_4 = 1/c_3 \tag{111}
\]

\[
c_5 = 2.2433 + 0.93N \tag{112}
\]

The value of N which is the exponent in the power law velocity profile relation, \(u/u_e = (y/\delta)^{1/N}\), was computed from the expression

\[
N = 12.67 - 6.5 \log(Re_{\theta,e}) + 1.21(\log(Re_{\theta,e}))^2 \tag{113}
\]

which comes from a curve fit of axisymmetric nozzle-wall data[19], and

\[
\frac{\delta}{\theta} T = N + 1 + \left[ \left( \frac{N + 2}{N} \frac{H_w}{H_{aw}} + 1 \right) (1 + 1.29(Pr_w)^{0.333} \frac{u_e^2}{2H_e}) \right] \tag{114}
\]

Since an inviscid solution is known, the boundary layer edge can be located through an iterative process of the momentum thickness equation, reference enthalpy equation and corresponding approximate ratio of boundary-layer thickness to momentum thickness. The inviscid properties at this location can then be interpolated to obtain the boundary-layer edge properties.

### 9 Gas Properties

The equilibrium air curvefits of Gupta et al[20] were used to evaluate thermodynamic and transport properties of the equilibrium air. The properties include enthalpy, total specific heat at constant pressure, compressibility factor, viscosity, total thermal conductivity, and total Prandtl number.
These curvefits are valid from 500 degrees Kelvin to 30,000 degrees Kelvin over a pressure range of $10^{-4}$ to $10^{2}$ atmospheres.

10 Results and Discussion

10.1 Wind Tunnel Case

The AA2LCH code was used to calculate the leading edge heating to a 0.025 scale model Space Shuttle Orbiter configuration at a wind tunnel condition of Mach 10 and angles of attack of 30 and 40 degrees. The results were then compared with the OH66 heat transfer test data.

Heat transfer test OH66 was conducted on a 0.025 scale semi-span Space Shuttle Orbiter model in September of 1976 in the Calspan 96-inch Hypersonic Shock Tunnel (HST) at a nominal Mach number of 10.2[21]. The right wing and a portion of the aft fuselage were deleted to allow a larger model size. The partial model was constructed of stainless steel. One of the test objectives was to obtain spanwise heat transfer rate distribution on the leading edge of the glove and wing. Figure 4 shows the layout of thin-film heat transfer gauges on the model.

The first step in using the AA2LCH code is to obtain an inviscid flowfield solution. The quality of the CFD solution is highly influenced by the quality of the grid. In order to obtain good wing leading edge heating predictions, it is crucial to have good grid resolution in the shock-shock interaction region near the wing leading edge. Because the location of shock-shock interaction is not known a priori, the following procedure was adapted to ensure good grid resolution at the regions needed. An initial grid as shown in figure 5 was used to get approximate shock shape and location. The grid has dimensions of $81 \times 61 \times 75$ in the $I, J, K$ directions, respectively, where the $I$ direction is from the nose toward tail, the $J$ direction is from the surface of the body toward the outer boundary of the grid, and the $K$ direction is wrapped around the body from top to bottom. An outer grid boundary adjustment program[22] was then used to push the outer boundary inward toward the shock. Figure 6 shows the resulting grid after the adjustment. Finally, the Multidimensional Self-Adaptive Grid Code (SAGE)[23] was used to adapt the grid based on the flowfield solution. Convergent solution is then obtained on the adaptive grid.

After the inviscid flowfield solution was obtained, the streamline and metric coefficients from the stagnation region to each of the heat transfer
gauge location were computed by the backward streamline tracing procedure. Figure 7 shows the streamlines on the body surface. Heating rates were then computed by using the BLIMP code. Figure 8 shows the comparison between predicted and measured heating rates for the 30 degree angle of attack case. The result is plotted as $q_w$ versus $x/L$, where $x$ is the length along the $x$-axis. For the .025 scaled model, $L$ is equal to 32.02 inches. Figure 9 shows the heating rate and pressure distribution along the leading edge of the model. In this double $y$ plot, both $q_w$ and $p/p_{stag}$ are plotted against $x/L$, where $p_{stag}$ is the stagnation pressure. From this figure, one can see that the heating rate distribution follows closely the pressure distribution. Figure 10 shows a typical plot of pressure distribution along a leading edge streamline. In this plot $p/p_{stag}$ along a streamline is plotted against the distance from the stagnation point. The rapid rise in pressure contributes to the higher heating rate in the leading edge area. From figure 8, one can see that the predictions compare well with experimental data except at $z/L = .6620$ and $z/L = .7703$ locations where the predicted value is about 15% less than the measurements. Overall, the predictions match well with the measurements. Figure 11 shows the comparison between computed and measured heating rate for the 40 degree angle of attack case. Again, the predictions compare well with the experimental data except at $z/L = .6530$. A heating rate of more than $135 Btu/ft^2/s$ was measured while the prediction was about $62.5 Btu/ft^2/s$. The measured heating rate at this location is considered unreasonably high.

10.2 STS-5 Flight Case

Since the IEC3D code does not include chemical non-equilibrium effects, a point on the Space Shuttle STS-5 trajectory was chosen for comparison where the flow is expected to be at chemical equilibrium. The point is at an altitude of 159,000 feet. The Mach number is 9.15 and angle of attack is 37.4 degrees. Again, the IEC3D code and the grid adaption procedure described above were used to obtain the inviscid flowfield solution. The initial grid was the same as the wind tunnel case. During the Shuttle flight, the radiometer located near the leading edge panel 9 was used to measure heating rate in the wing shock-shock interaction area. The location viewed by the radiometer is at $x = 1095$ in, $y = 256.488$ in, and $z = 291.94$ in. The predicted heating rate using the approximate convective-heating equations of Ref. [12] is about $8.0 Btu/ft^2/s$ at this location, and the heating rate
backed out from radiometer data is about $14.0\text{Btu/ft}^2/\text{s}[24]$. Because of the large discrepancy between the prediction and measurement, a comparison of windward surface heating rate with flight data was made in order to determine the reason for this under-prediction. Thermocouple locations on the windward surface are shown in figure 12. The detailed description of thermocouple locations and heat transfer analysis of the STS-5 flight data can be found in Hartung and Throckmorton[25]. The comparison of heating rates along the windward centerline is shown in figure 13. The results are plotted as $q_w$ versus $x/L$, where $L$ is taken as 1280.8 inches. Good agreement with the flight data was obtained on the windward centerline. Lateral heating comparisons in planes normal to the centerline of the Shuttle Orbiter on the fuselage at $x/L$ stations of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, and 0.9 are shown in figures 14 through 21. The results are plotted as $q_w$ versus $y$, where $y = 0$ is on the centerline. In general, the prediction is within 15% of the measurement, except at three locations where the difference is larger. The comparison of heating rate on the surface of the wing is shown in figures 22 through 27. The results are plotted as $q_w$ versus $x/L$ at $y$ equal to 184.8, 233.6, 275.3, 322.7, 369.0 and 420.8 inches. There appears to be some scatter in the data. The trajectory point is at 1120 seconds after entry interface. By examining thermocouple data[25], one should realize that boundary-layer transition has occurred, and the flow has already turned turbulent on part of the wing. This contributes to the large scatter of the data.

10.3 STS-2 Flight Case

A point on the Space Shuttle STS-2 trajectory point was also chosen for comparison. The point is at an altitude of 179,920 feet, about 1090 seconds after entry interface. The Mach number is 12.86 and the angle of attack is 39.7 degrees. At this altitude and flight condition, the equilibrium assumption is still valid. The grid used for this case has dimensions of $151 \times 61 \times 110$ in the $I, J, K$ directions, respectively. The inviscid solution was obtained by applying the outer boundary adjustment procedure twice, the grid adaption procedure was not used in this case.

The detailed description of the heat transfer analysis of the STS-2 flight data can be found in Reference [26]. Figure 28 shows the comparison of heating rate along the windward centerline. The results are plotted as $q_w$ versus $x/L$. Again, good agreement with the flight data was obtained on the windward centerline. Lateral heating comparison in planes normal to the
centerline of the Shuttle Orbiter on the fuselage at $x/L$ stations of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, and 0.9 are shown in Figures 29 through 36 respectively. The results are plotted as $q_w$ versus $y$. Good agreement between prediction and measured flight data was obtained again. Figures 37 through 42 show the comparison of heating rate on the wing surface. The results are plotted as $q_w$ versus $x/L$ at $y$ equal to 184.8, 233.6, 275.3, 322.7, 369.0, and 420.8 inches. Agreement between the prediction and flight data for the STS-2 case is better than that for the STS-5 case.

The heating rate prediction at the location viewed by the leading edge panel 9 radiometer is $20.0 Btu/ft^2/s$, and the heating rate backed out from radiometer data is $27.0 Btu/ft^2/s$. This is certainly an improvement as compared with the STS-5 case, but there is still a 25% under-prediction.

From the above comparisons, the under prediction of heating by the two-layer method in the wing leading edge shock-shock interaction area is probably caused by the deficiency of the inviscid flow solver to resolve the strong viscous effect in this region. Also, the flow in the vicinity of the wing leading edge is highly three-dimensional, and neglecting the cross flow in the boundary layer is probably another reason for the under prediction.

11 Conclusions

A procedure for evaluating surface heating rates for Orbiter-like vehicles at high angle of attack has been developed. The axisymmetric analog based procedure uses three-dimensional inviscid solutions in Cartesian coordinates to trace streamlines and compute metric coefficients, and uses either an axisymmetric boundary layer code or Zoby's approximate convective-heating equations for heating rate computation. The procedure has been applied to the prediction of leading edge heating rate of an 0.025 scale shuttle orbiter at a wind tunnel condition of Mach 10 and angles of attack at 30 and 40 degrees. Good agreement was obtained between predictions and measurements.

Heating rate predictions were also compared with flight-deduced data on an STS-5 trajectory point and an STS-2 trajectory point. Good agreement were obtained on the windward surface for both cases. However, the predictions on the wing leading edge shock-shock interaction area were lower than flight-deduced data. This discrepancy is probably caused by the inviscid solver being unable to resolve the strong viscous effect of shock-shock interaction, and the neglect of cross flow in the boundary layer by the two-layer
References


Figure 1: Streamline coordinates

Figure 2: Mapping from master element to a physical element
Figure 3: Mapping from master element to physical domain
Figure 4: Layout of thin-film heat transfer gauge
Figure 5: Pitch plane initial grid

Figure 6: Pitch plane grid after outer boundary adjustment
Figure 7: Surface streamlines
Orbiter Wing Leading Edge Heating Prediction
OH66 Wind Tunnel Condition, Alpha = 30 deg.

Figure 8: Comparison of predicted and measured heating rates, $\alpha = 30\,\text{deg.}$
Figure 9: Pressure and heating rate distribution along the leading edge

Figure 10: Typical pressure distribution along a leading edge streamline
Orbiter Wing Leading Edge Heating Prediction
OH66 Wind Tunnel Condition, Alpha = 40 deg.

Figure 11: Comparison of predicted and measured heating rates, $\alpha = 40$ deg.
Figure 12: Orbiter windward surface thermocouple locations
Figure 13: Windward centerline heating distribution, STS-5 case, Mach = 9.15, Alpha = 37.4 deg.
Figure 14: Circumferential heating distribution at $x/L = .1$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 15: Circumferential heating distribution at $x/L = .2$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 16: Circumferential heating distribution at $x/L = .3$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 17: Circumferential heating distribution at $x/L = .4$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 18: Circumferential heating distribution at $x/L = .5$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 19: Circumferential heating distribution at $x/L = .6$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 20: Circumferential heating distribution at $x/L = .7$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 21: Circumferential heating distribution at $x/L = .9$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 22: Heating distribution on windward surface at $y = 184.8$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 23: Heating distribution on windward surface at $y = 233.6$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 24: Heating distribution on windward surface at $y = 275.3$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 25: Heating distribution on windward surface at $y = 322.7$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 26: Heating distribution on windward surface at $y = 369.0$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 27: Heating distribution on windward surface at $y = 420.8$ in., STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 28: Heating distribution along windward centerline, STS-2 case, $Mach = 12.86, \ Alpha = 39.7$ deg.
Figure 29: Circumferential heating distribution at $x/L = .1$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 30: Circumferential heating distribution at $x/L = .2$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 31: Circumferential heating distribution at $x/L = .3$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 32: Circumferential heating distribution at $x/L = .4$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 33: Circumferential heating distribution at $x/L = 0.5$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7 \, deg$.

Figure 34: Circumferential heating distribution at $x/L = 0.6$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7 \, deg$. 
Figure 35: Circumferential heating distribution at $x/L = .7$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 36: Circumferential heating distribution at $x/L = .9$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 37: Heating distribution on windward surface at $y = 184.8$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 38: Heating distribution on windward surface at $y = 233.6$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 39: Heating distribution on windward surface at $y = 275.3$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 40: Heating distribution on windward surface at $y = 322.7$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 41: Heating distribution on windward surface at $y = 369.0$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 42: Heating distribution on windward surface at $y = 420.8$ in., STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
## A Appendix A - Input Description

Following is a brief description of input records for the AA2LCH code, Table A.1 shows a sample input.

<table>
<thead>
<tr>
<th>record</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Grid file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((x(i,j,k),i=1,idim),j=1,jdim),k=1,kdim), (((y(i,j,k),i=1,idim),j=1,jdim),k=1,kdim), (((z(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>2</td>
<td>Density file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((rho(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>3</td>
<td>X-component of velocity file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((u(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>4</td>
<td>Y-component of velocity file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((v(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>5</td>
<td>Z-component of velocity file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((w(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>6</td>
<td>Pressure file, must be written using FORTRAN UNFORMATTED I/O as follows: write(unit) idim, jdim, kdim write(unit) (((p(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)</td>
</tr>
<tr>
<td>7</td>
<td>streamline output file, contains following: streamline coordinates, metric coefficient, distance from stagnation point and pressure</td>
</tr>
<tr>
<td>8</td>
<td>streamline outfile, suitable for BLIMP input, contains distance from stagnation point, metric coefficient and pressure to stagnation pressure ratio</td>
</tr>
<tr>
<td>record</td>
<td>description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>9</td>
<td>output surface grid file in PLOT3D format (unformatted)</td>
</tr>
<tr>
<td>10</td>
<td>an output PLOT3D q file (unformatted), contains surface heating rate, surface radiation equilibrium temperature, boundary layer thickness, momentum thickness, and skin friction</td>
</tr>
<tr>
<td>11</td>
<td>input file, contains wall temperature</td>
</tr>
<tr>
<td>12</td>
<td>a temporary file</td>
</tr>
<tr>
<td>13</td>
<td>an output file, contains location, distance from stagnation point, heating rate and skin friction along a streamline</td>
</tr>
<tr>
<td>14</td>
<td>output file, contains radiation equilibrium temperature along a streamline</td>
</tr>
<tr>
<td>15</td>
<td>output file, contains surface heating rate at grid point</td>
</tr>
<tr>
<td>16</td>
<td>outfile file, contains error message</td>
</tr>
<tr>
<td>17</td>
<td>grid scale</td>
</tr>
<tr>
<td>18</td>
<td>model scale</td>
</tr>
<tr>
<td>19</td>
<td>free stream pressure (psf)</td>
</tr>
<tr>
<td>20</td>
<td>free stream temperature (deg Rankin)</td>
</tr>
<tr>
<td>21</td>
<td>free stream velocity (fps)</td>
</tr>
<tr>
<td>22</td>
<td>free stream Mach number</td>
</tr>
</tbody>
</table>
| 23     | equilibrium air flag  
  = 1; equilibrium air  
  = 0; perfect gas |
| 24     | turbulent heating flag  
  = 1; turbulent heating computation  
  = 0; laminar heating computation |
| 25     | radiation equilibrium temperature flag  
  = 1; radiation equilibrium wall temperature  
  = 0; constant wall temperature |
| 26     | surface emissivity |
| 27     | wall temperature input flag  
  = 1; wall temperature file is specified in record 11  
  = 0; input initial wall temperature in record 28 |
<p>| 28     | initial wall temperature |
| 29     | angle of attack (deg.) |</p>
<table>
<thead>
<tr>
<th>record</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>distance from stagnation point where streamline tracing starts (not used in backward streamline tracing)</td>
</tr>
<tr>
<td>31</td>
<td>initial step size of integration</td>
</tr>
<tr>
<td>32</td>
<td>maximum relative truncation error allowed in variable step Runge-Kutta integration</td>
</tr>
<tr>
<td>33</td>
<td>minimum step size allowed in variable step Runge-Kutta integration</td>
</tr>
<tr>
<td>34</td>
<td>number of integration steps to be taken (not used in backward tracing)</td>
</tr>
<tr>
<td>35</td>
<td>number of streamlines to be generated (not used in backward tracing)</td>
</tr>
<tr>
<td>36</td>
<td>initial streamline spreading angle (not used in backward tracing)</td>
</tr>
</tbody>
</table>
| 37     | backward streamline tracing flag  
         = 1; yes  
         = 0; no |
| 38     | flag indicates whether initial location of streamline is on a grid point  
         = 1; initial location is on a grid point  
         = 0; initial location is input in the following record |
| 39     | coordinates of initial location of streamline |
| 40     | initial and final grid points for streamline tracing, always in the form of imin, imax, kmin, and kmax, where i is from nose to tail and k is wrapped around the body from top to bottom |
| 41     | character indicates whether boundary layer parameter is available for interpolation along a streamline  
         'y' is available  
         'n' is not available |
| 42     | a PLOT3D q file contains boundary thickness, momentum thickness, displacement thickness and Reynolds number per foot |
| 43     | a PLOT3D q file contains REK values for k equals to .1, .25, .5, and 1. inch |
Table 1: Sample AA2LCH input file

```plaintext
'sts2_nunf.xyz'
'sts2_nunf.q1'
'sts2_nunf.q2'
'sts2_nunf.q3'
'sts2_nunf.q4'
'sts2_nunf.p'
'sts2.ws.dat'
'sts2.wm.dat'
'sts2_basur.xyz'
'sts2_basur.q'
'twallI.dat'
'temp.dat'
'hrate.dat'
'rwall.dat'
'sufheat.dat'
'error.dat'
'grid scale = 1.
'model scale = 1.
'free stream pressure'= .84578
'free stream temperature'= 470.88
'free stream velocity'= 13674.5
'free stream mach no. '= 12.85
'equilibrium air flag '=' 1
'turbulent heating flag '=' 0
'radiator equilibrium flag' 1
'surface emissivity' .907
'wall temperature input flag' 0
'wall temperature deg. R' 540.
39.7
10.
.01
1.0e-4
1.0e-5
130
1
180.
'backward integration flag=' 1
'flag indicate grid point only' 1
'starting location '=' 251.5 6.4 -60.6416
'imix,imax,kmin,kmax' 140 140 110 110
'transition parameter files' 'n'
'/ceg/kwang/stream/roex3.q2'
'/ceg/kwang/stream/roex3.q4'
```

Table 1: Sample AA2LCH input file
Appendix B - AA2LCH Listing
program aa21ch

This program reads in an euler flowfield in Cartesian coordinate, and trace the streamine and compute metric coefficient in a backward or forward fashion from a given starting location. Third order Runge-Kutta integrator is used to perform numerical integration.

common /connty/ il, j, k1
common /stagpt/ xstag, ystag, zstag
common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzta(8)
common /nomxyz/ xn(151,110), yn(151,110), zn(151,110), xnp, ynp, znp
common /grdxyz/ xxyz(151,110,110,3), idim, jdim, kdim,
& .iblank(151,40,110), blank, isubs(2),
&  jsubs(2), ksubs(2)
common /sfxiet/ xixs(151,110), xiys(151,110), xisz(151,110),
&  etxs(151,110), etys(151,110), etzs(151,110),
&  flogsm(151), flogsn(151)
common /dvdwyz/ dvdy(151,110), dwdz(151,110),
&  dwdy(151,110)

modification of 8/16/93

common /duxyz/ dudx(151,110), dudy(151,110), dudz(151,110),
&  dwdx(151,110), dwdx(151,110)
end of modification
common /geom/ dxdy(151,110), dxdz(151,110)
common /surfpr/ vel, surfn, up, vp, surfp, dxzdp, dydzp, xdi, det
common /sfbond/ msp(151), msr(151), nsp(151), nsr(151)
common /qxyz/ rho(151,110,110), u(151,110), v(151,110), w(151,110),
&  p(151,110)
common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav,
&  first, back, second, ifatal
common /freest/ pinf, tinf, rholinf, vinf, vninf, cpinf, igas
common /radeq/ itwall, wallt, irad, eps
common /tempwl/ twall(1000)

boundary and momentum thickness
common /bltrn/ deltal, thetl
dimension y(50), us(151)
dimension xb(1500), yb(1500), zb(1500), hb(1500)
dimension q(151,1,110,5)
dimension qll(151,1,110,5), q12(151,1,110,5)
logical blank, first, back, second, fext
character*80 gfile, rhofle, rhoufl, rhovfl, rhowfl, pfile,
&  sfile, mfile, twfile, xfile, qfile, lqfl, lqf2,
&  ilfile, outfle, rwfile, shfile, errfile
character*30 dummy
character*1 flauna

cconversion factors

data cmin, fps, atm, den /2.54, 30.48, 2116.2, .01601846/
data rtok /1.8/
cStefan-Boltzman constant (w/cm**2/k)
data sigma /5.67e-12/
cinput surface grid locations
read(5,*) gfile
read(5,*) rhofle
read(5,*) rhoufl
read(5,*) rhovfl
read(5,*) rowsfl
read(5,*) pfile
read(5,*) sfile
read(5,*) mfile
read(5,*) xfile
read(5,*) qfile
read(5,*) twfile
read(5,*) ilfile
read(5,*) outflile
read(5,*) rwsfile
read(5,*) shfile
read(5,*) errfile
read(5,*) dummy, scale
read(5,*) dummy, scale
c
input freestream condition

read(5,*) dummy, pinf
read(5,*) dummy, tinf
read(5,*) dummy, vinf
read(5,*) dummy, aminf
pinf = pinf / atm
tinf = tinf / rtok
vinf = vinf*fps
read(5,*) dummy, igas
c
input turbulent flag

read(5,*) dummy, iturb
c
input radiation equilibrium flag

read(5,*) dummy, irad
c
input surface emissivity

read(5,*) dummy, eps
c
input wall temperature flag

read(5,*) dummy, itwall
if(itwall .eq. 0)
  read(5,*) dummy, wallt
endif
c
get free stream properties

iflag = 2
if(igas .eq. 0) iflag = 0
call enthalpy(pinf, hinf, tinf, iflag)
if(ifatal .ne. 0) then
  write(6,*) ' Bad freestream condition encountered'
  stop
endif
write(6,*) ' hinf(curve fit) = ', hinf
call eqprop(pinf, tinf, cpinf, rhoinf, xmuinf, xkinf, prinf, gaminf, &
  arminf, iflag)
if(ifatal .ne. 0) then
  write(6,*) ' Bad freestream condition encountered'
  stop
endif
c
normalization factor for velocity (cm/sec)
c

\[ v_{ninf} = \sqrt{p_{ninf} l/1.013336/rho_{ninf}} \]

c open(unit=7, file=gdfile, form='unformatted',
& status='old')
c open(unit=8, file=rhofile, form='unformatted',
& status='old')
open(unit=9, file=rhoufile, form='unformatted',
& status='old')
open(unit=10, file=rhovfile, form='unformatted',
& status='old')
open(unit=11, file=rhowfile, form='unformatted',
& status='old')
open(unit=12, file=pfile, form='unformatted', status='old')
open(unit=13, file=sfile, form='formatted',
& status='unknown')
open(unit=14, file=mfile, form='formatted', status='unknown',
& err=9999)
open(unit=18, file=tfile, form='formatted',
& status='unknown')
inquire(file=xfile, exist=fext)
if(fext) then
  open(unit=16, file=xfile, form='unformatted', status='old')
  open(unit=17, file=qfile, form='unformatted', status='old')
  read(17) idim, jdim, kdim
  read(17) aminf, alp, re, time
  read(17) (((q(i, j, k, nx), i=1, idim), j=1, jdim), k=1, kdim),
& nx=1, 5)
else
  open(unit=16, file=xfile, form='unformatted', status='new')
  open(unit=17, file=qfile, form='unformatted', status='new')
endif
c additional file
c open(unit=15, file=ilfile, form='unformatted', status='unknown')
open(unit=24, file=outfile, form='formatted', status='unknown')
open(unit=25, file=rfile, form='formatted', status='unknown')
open(unit=30, file=shfile, form='formatted', status='unknown')
open(unit=40, file=errfile, form='formatted', status='unknown')
c idim = 0
jdim = 0
kdim = 0
continue
read(7) idim, jdim, kdim
read(7) (((xyz(i, j, k, 1), i=1, idim), j=1, jdim), k=1, kdim),
& (((xyz(i, j, k, 2), i=1, idim), j=1, jdim), k=1, kdim),
& (((xyz(i, j, k, 3), i=1, idim), j=1, jdim), k=1, kdim)
c multiply by scale factor
do 20 j=1, jdim
do 20 k=1, kdim
do 20 i = 1, idim
xyz(i, j, k, 1) = xyz(i, j, k, 1) * scale
xyz(i, j, k, 2) = xyz(i, j, k, 2) * scale
xyz(i, j, k, 3) = xyz(i, j, k, 3) * scale
continue
c modification of 1/11/94
c pi = atan2(1.0, 0.) * 2.
dangle = pi / kdim
do 30 j = 1, jdim
do 30 k = 1, kdim
angle = (k-1)*dangle
\[ \text{xyz}(1,j,k,2) = 1.e-7 \times \sin(\text{angle}) \times \text{scale} \]
\[ \text{xyz}(1,j,k,3) = 1.e-7 \times \cos(\text{angle}) \times \text{scale} \]

30 continue

c end of modification

c read rho

c read(8) idim, jdim, kdim
read(8) (((rho(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c read u
read(9) idim, jdim, kdim
read(9) (((u(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c read v
read(10) idim, jdim, kdim
read(10) (((v(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c read w
read(11) idim, jdim, kdim
read(11) (((w(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c read p
read(12) idim, jdim, kdim
read(12) (((p(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c alp is angle of attack

c read(5,*) alp
pi = atan2(0.,-1.)
dtr = \pi / 180.
alpr = alp * dtr
cosa = \cos(alpr)
sina = \sin(alpr)
ksdim = kdim
if(alp .lt. 0.) ksdim = 1
c
locate stagnation point (no yaw)

c istag = 0
\text{do 160} i = 1,idim
us(i) = u(i,1,ksdim)
\text{160 continue}
c
locate a pair grid points between which u changes sign.

c \text{do 170} i = idim,2,-1
iml = i - 1
if(\text{us}(i)\times\text{us}(\text{iml}) .lt. 0.) then
is = iml
ispl = is + 1
\text{go to 180}
else if(\text{us}(i) .eq. 0.) then
\text{istag} = 1
is = i
ispl = is + 1
\text{go to 180}
else if(\text{us}(\text{iml}) .eq. 0.) then
\text{istag} = 1
is = iml
ispl = is + 1
\text{go to 180}
170  continue  

c  write(6,*), ' Stagnation point cannot be located'
stop
180  continue  

c  find coordinates of stagnation point

c  xstag = 0.
ystag = 0.
zstag = 0.
if(istag .eq. i) then
  xstag = xyz(ispl,l,ksdim,1)
ystag = xyz(ispl,l,ksdim,2)
zstag = xyz(ispl,l,ksdim,3)
else
  c  
  c use linear interpolation to find coordinates of
  c stagnation point.
  
  ds = sqrt((xyz(is, l,ksdim, 1) - xyz(ispl,l,ksdim, 1))**2 +
            &  (xyz(is, l,ksdim, 3) - xyz(ispl,l,ksdim, 3))**2)
ds1 = - us(is) * ds / (us(ispl) - us(is))
ds2 = ds - ds1
write(6,*), ' ds,ds1,ds2',ds,dsl,ds2

c  xstag = xyz(is,l,ksdim, 1) + ds1/ds*
&  (xyz(ispl, l,ksdim, 1)-xyz(is, l,ksdim, 1))
zstag = xyz(is,l,ksdim, 3) + ds1/ds*
&  (xyz(ispl, l,ksdim, 3)-xyz(is, l,ksdim, 3))
pstag = p(is, l,ksdim)+ds1/ds*(p(ispl, l,ksdim)-p(is, l,ksdim))
write(6,*), ' Interpolated stagnation pressure : ',pstag
endif

c  compare with maximum pressure

do 190 i = 1,idim
if(p(i,l,l) .gt. pstag) pstag = p(i,l,l)
if(p(i,l,kdim) .gt. pstag) pstag = p(i,l,kdim)
190  continue  

c  find starting point location

c  read(5,*),dels
if(dels .gt. ds1) then
  iss = is
  is = is - 1
if(is .le. 0) then
  is = iss
  dels = ds1
endif
else if(dels .gt. ds2) then
  ispl = ispl
  ispl = ispl + 1
if(ispl .gt. idim) then
  ispl = ispl
  dels = ds2
endif
endif
if(alp .gt. 0.) then
  xstagl = xstag + dels/ds2 * (xyz(ispl, l,ksdim, 1)-xstag)
zstagl = zstag + dels/ds2 * (xyz(ispl, l,ksdim, 3)-zstag)
ystagl = 0.
xstag2 = xstag + dels/dsl * (xyz(is,1,ksdim,1)-xstag)
zstag2 = zstag + dels/dsl * (xyz(is,1,ksdim,3)-zstag)
ystag2 = 0.
else
  xstagl = xstag + dels/dsl * (xyz(is,1,ksdim,1)-xstag)
zstagl = zstag + dels/dsl * (xyz(is,1,ksdim,3)-zstag)
ystagl = 0.
xstag2 = xstag + dels/ds2 * (xyz(ispl,1,ksdim,1)-xstag)
zstag2 = zstag + dels/ds2 * (xyz(ispl,1,ksdim,3)-zstag)
ystag2 = 0.
endif
if(is .ne. iss) is = iss
if(ispl .ne. ispl) ispl = ispl

set up boundary values for each element

msper = 0
nsper = 0
msl = 1
ns1 = 1
mdim = idim
ndim = kdim
call bounds(mdim,ndim,msl,mdim,ns1,ndim,msper,nsper)

compute surface metrics

call surmet(mdim,ndim,msl,mdim,ns1,ndim,msper,nsper)

n is number of functions to be integrated

n = 5
modification of 8/16/93

n = 6

h is initial step size

read(5,*) saveh
hsave = saveh
back = .false.

emax is maximum relative truncation error

read(5,*) emax

hmin is minimum step size allowed

read(5,*) hmin

nstep is the number of steps of integration

read(5,*) nstep

nsl is total number of streamlines to be generated

read(5,*) nsl

angi is initial spreading angle between streamlines

read(5,*) angi

read in backward streamline tracing flag
read(5,*) dummy, iback

itype = 1 : multiple streamline, imin, imax, kmin, kmax required
itype = 0 : single streamline

read(5,*) dummy, itype
read(5,*) dummy, xst, yst, zst
read(5,*) dummy, ibmin, ibmax, kbmin, kbmax
read(5,*) dummy, flaura
if (flaura .eq. 'y' .or. flaura .eq. 'Y') then
  read(5,*) lqfl
  read(5,*) lqf2
  open (unit=21, file=lqfl, form='unformatted', status='old')
  open (unit=22, file-lqf2, form='unformatted', status='old')
endif
read(5,*) dummy, nbstep
if (iback .eq. i) then
  back = .true.
else
find coordinates of center
xc = xstag + dels
zc = zstag + dels * tan(alpr)
yc = 0.
sinthe = cos(alpr)
costhe = sqrt(1.-sinthe*sinthe)
rr = dels/cosa
starting location
read(5,*,end=9999) isav, jsav, ksav, xf, yf, zf
  second = .false.
  igsav = 1
  issav = 1
  asav = 0.
  bsav = 0.
  gsav = 0.
  ifatal = 0

nsl is total number of streamlines to be generated
dtr is the conversion factor from degree to radian
dthe is the initial delta angle between two adjacent streamlines
dphi = angi
if (nsl .gt. 1 .and. dphi .le. 0.) then
  dphi = 180. / (nsl - 1)
endif
dphir = dphi * dtr
cang = dphi * nsl
jflag = 0
  if (cang .ge. 180.) jflag = 1
endif
determin number of streamlines to be traced
if(itype .ne. 0) then
    nsl = (ibmax-ibmin+1) * (kbmax-kbmin+1)
    istart = ibmin
    kstart = kbmin
endif

do 300 nl = 1,nsl
second = .false.
if(iback .eq. 1) back = .true.
if(back) then
    if(itype .eq. 0) then
        xin = xst
        yin = yst
        zin = zst
    else
        xin = xyz(istart,1,kstart,1)
        yin = xyz(istart,1,kstart,2)
        zin = xyz(istart,1,kstart,3)
    endif
else
    first = .false.
    if(nl .eq. 1) first = .true.
    c initial position
    if(nl .eq. 1) then
        xin = xstgl
        yin = ystgl
        zin = zstgl
    endif
    if(nsl .eq. 1 .and. jflag .eq. 1) then
        xin = xstag2
        yin = ystag2
        zin = zstag2
    endif
else
    phir = (nl-1)*dphir
    cosp = cos(phir)
    sinp = sin(phir)
write(6,*) ' cosp, sinp ',cosp, sinp
    c coordinate in (xc,yc,zc) system
    xxp = -(rr * costhe)
    yyp = rr * sinthe * sinp
    zzp = -(rr * sinthe * cosp)
write(6,*) ' xxp,yyp, zzp ',xxp, yyp, zzp
    c transfer coordinate to original (x,y,z) system
    xin = xxp*cosa - zzp*sina + xc
    zin = xxp*sina + zzp*cosa + zc
    yin = yyp
    if(jflag .ne. 0 .and. nl .eq. nsl) then
        xin = xstgl
        yin = ystgl
        zin = zstgl
    endif
endif
endif
xin = xs(ispl+1,kdim-nl+1)
yin = ys(ispl+1,kdim-nl+1)
zin = zs(ispl+1,kdim-nl+1)
continue

write(6,*) ' xin,yin,zin ',xin,yin,zin
ms = 1
me = idim
ns = 1
ne = kdim
call fndnrp(ms,me,ns,ne,xin,yin,zin)
istat = 0
delxi = 0.
delret = 0.
call quads(delxi,delret,xin,yin,zin,istat)
if(istat .gt. 1) then
  write(6,*) ' Search failed, istat = ',istat
  write(40,*) ' Location ',istart,kstart,' failed'
go to 240
endif
i1 = isav
k1 = ksav
  if(n1 .eq. 1) then
    if(k1 .eq. 1 .or. k1 .eq. kdim) then
delret = 0.
  endif
  endif
c11 = delxi * delret
c10 = delxi - c11
c01 = delret - c11
c00 = 1. - delxi - delret + c11
ipl = msp(i1)
kpl = nsp(k1)
  uin = u(i1,1,k1)*c00 + u(ipl,1,k1)*c10 + u(ipl,1,kpl)*cll
    & + u(i1,1,kpl)*c01
  vin = v(i1,1,k1)*c00 + v(ipl,1,k1)*c10 + v(ipl,1,kpl)*cll
    & + v(i1,1,kpl)*c01
  win = w(i1,1,k1)*c00 + w(ipl,1,k1)*c10 + w(ipl,1,kpl)*cll
    & + w(i1,1,kpl)*c01
  pin = p(i1,1,k1)*c00 + p(ipl,1,k1)*c10 + p(ipl,1,kpl)*c01
    & + p(i1,1,kpl)*c01
dxdyn = dxdy(i1,1,k1)*c00 + dxdy(ipl,1,k1)*c10 +
    & dxdy(ipl,1,kpl)*c01
  dxdzin = dxdz(i1,1,k1)*c00 + dxdz(ipl,1,k1)*c10 +
    & dxdz(ipl,1,kpl)*c01
find norm vector

  xnin = xn(i1,1,k1)*c00 + xn(ipl,1,k1)*c10 +
    & xn(ipl,1,kpl)*c01
  ynin = yn(i1,1,k1)*c00 + yn(ipl,1,k1)*c10 +
    & yn(ipl,1,kpl)*c01
  znin = zn(i1,1,k1)*c00 + zn(ipl,1,k1)*c10 +
    & zn(ipl,1,kpl)*c01
interpolate boundary layer transition parameters

  if(faura .eq. 'y' .or. flaira .eq. 'Y') then
    repf = qll(i1,1,k1,4)*c00 + qll(ipl,1,k1,4)*c10 +
      & qll(ipl,1,kpl,4)*c01 + qll(i1,1,kpl,4)*c01
  thkm = qll(i1,1,k1,2)*c00 + qll(ipl,1,k1,2)*c10 +
    & qll(ipl,1,kpl,2)*c01 + qll(i1,1,kpl,2)*c01
  thkd = qll(i1,1,k1,3)*c00 + qll(ipl,1,k1,3)*c10 +
    & qll(ipl,1,kpl,3)*c01 + qll(i1,1,kpl,3)*c01
  rek = qll(i1,1,k1,3)*c00 + qll(ipl,1,k1,3)*c10 +
    & qll(ipl,1,kpl,3)*c01 + qll(i1,1,kpl,3)*c01
rem = repf * thkm / 12.
red = repf * thkd / 12.
rex = repf * dist / 12.
endif

c
x = 0.
y(1) = xin
y(2) = yin
y(3) = zin
dist = sqrt((xin-xstag)**2+(yin-ystag)**2+(zin-zstag)**2)

c initial metric coefficient

c
hmet = dist
hmet = sqrt((xin-xstag)**2+(yin-ystag)**2+(zin-zstag)**2)

c initial surface norm vector magnitude

c
surfn = sqrt(1. + dxdyin**2 + dxdzin**2)
vel = sqrt(uin**2+vin**2+win**2)

c initial value for dydtau and dzdtau

c
y(4) = -hmet/surfn * (win/vel + uin/vel * dxdzin)
y(5) = hmet/surfn * (vin/vel + uin/vel * dxdyin)
y(4) = -hmet * (xinin*win/vel - znin*uin/vel)
y(5) = hmet * (xinin*vin/vel - ynin*uin/vel)

c modification of 8/16/93

c
y(6) = hmet * (ynin*win/vel - znin*vin/vel)
c end of modification

c re-compute hmet from dydtau and dzdtau

c
hmet = surfn/vel * (vin*y(5) - win*y(4))
hmet = (vin*y(5) -win*y(4))/(vel*xnin)
hmet1 = -y(4) / (xin*win/vel - znin*uin/vel)
hmet2 = y(5) / (xin*vin/vel - ynin*uin/vel)

hmet=(y(5)-y(4))/(xnin*(vin+win)/vel-uin/vel*(ynin+znin))

hmet = sqrt(y(4)**2 + y(5)**2 + y(6)**2)

c end of modification

c
hmet = surfn/(vel-uin/vel*(uin-vin*dxdyin-win*dxdzin))*
& (vin*y(5) - win*y(4))
hmet1 = -surfn*y(4)/(win/vel+uin/vel*dxdzin)
hmet2 = surfn*y(5)/(vin/vel+uin/vel*dxdyin)
c if(xf .ne. 0. .and. yf .ne. 0. .and. zf .ne. 0.) then
  x = xf
cy(1) = yf
cy(2) = zf
cendif

c output streamline locations

c
write(13,*) ' Streamline No. ',nl
write(14,*) ' Streamline No. ',nl
write(6,*) ' Streamline No. ',nl
write(13,9001) y(1),y(2),y(3),dist,hmet,pin
if(flarga .eq. 'y' .or. flaura .eq. 'Y') then
  write(26,9001) thkm,thkd,rem,red,rek,rex
endif
write(14,9002) dist/12.,hmet/12.,pin/pstag
write(15)y(1),y(2),y(3),-xnin,-ynin,-znin,vel,pin,hmet,isav,ksav
write(17,9010) y(1),y(2),y(3),y(1)-xmin,y(2)-ynin,y(3)-zmin
write(6,*) y(1),y(2),y(3),y(4),y(5),surfn,vel,hmet,pin

integration loop

h = hsave
hmin = hmin
write(6,*) ' real step size',h,hmin
xsave = y(1)
ysave = y(2)
zssave = y(3)
do 250 i = 1,nstep
200 continue
if(second) then
  i = nstep
else
  i = 1
endif
210 continue
if(second) then
  y(1) = xb(i)
y(2) = yb(i)
y(3) = zb(i)
  if(i .eq. 2) then
    h = hb(i)
  else
    h = hb(i) - hb(i-1)
  endif
else
  y(1) = xb(i)
y(2) = yb(i)
y(3) = zb(i)
endif
write(6,*) 'back',y(1),y(2),y(3),h
    psave = surfp
call rk34(n,x,y,h,emax,hmin)
if(ifatal .ne. 0) then
  write(40,*) ' Location ',istart,kstart,' failed'
  ifatal = 0
  go to 240
endif
if (back) then
  if(itype .eq. I) then
    distag = sqrt((y(1)-xstag)**2 + (y(2)-ystag)**2 + (y(3)-zstag)**2)
    if(distag .lt. dels) then
      if(surfp .le. psave) then
        go to 220
      else if(abs(surfp-psave) .le. 1.e-3) then
        go to 220
      endif
    endif
  else
    if(i .gt. nstep) go to 220
    i= i + 1
  endif
  xb(i) = y(1)
yb(i) = y(2)
zb(i) = y(3)
hb(i) = x
else
  i = i - 1
endif
write(6,*) ' step ',i
write(6,*) isptr,xb(isptr),yb(isptr),zb(isptr),hb(isptr)
write(6,*) 'vp,y(5),wp,y(4)',vp,y(5),wp,y(4),surfn/vel
hmet = surfn / vel * (vp*y(5) - wp*y(4))
hmet = (vp*y(5) - wp*y(4))/(vel*xnp)
hmet1 = -y(4) / (xnp*wp/vel - znp*up/vel)
hmet2 = y(5) / (xnp*vp/vel - ynp*up/vel)
hmet = (y(5) - y(4)) / (xnp*(vp+wp)/vel - up/vel*(ynp+znp))

modification of 8/16/93

hmet = sqrt(y(4)**2 + y(5)**2 + y(6)**2)
end of modification

hmet = surfn/(vel-up/vel*(up-vp*dxddyw-wp*dxdzp)) *
  (vp*y(5) - wp*y(4))
& hmet1 = -surfny(4)/(vp/vel+up/vel*dxdzp)
& hmet2 = surfny(5)/(vp/vel+up/vel*dxdy)
dist = dist + sqrt((y(1)-xsave)**2 + (y(2)-ysave)**2 +
  (y(3)-zsave)**2)

xsave = y(1)
ysave = y(2)
zsave = y(3)
hs = h

interpolate boundary layer transition parameters

if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
  il = isav
  k1 = ksav
  cl1 = dxi * det
  c10 = dxi - cl1
  c01 = det - cl1
  c00 = 1. - dxi - det + cl1
  ip1 = msp(il)
  kpl = nsp(kl)
  repf = qll(il,1,k1,4)*c00 + qll(ip1,1,k1,4)*c10 +
  & qll(ip1,1,kpl,4)*c11 + qll(il,1,k1,4)*c01

  thkm = qll(il,1,k1,2)*c00 + qll(ip1,1,k1,2)*c10 +
  & qll(ip1,1,kpl,2)*c11 + qll(il,1,k1,2)*c01

  thkd = qll(il,1,k1,3)*c00 + qll(ip1,1,k1,3)*c10 +
  & qll(ip1,1,kpl,3)*c11 + qll(il,1,k1,3)*c01

  rek = qll(il,1,k1,3)*c00 + qll(ip1,1,k1,3)*c10 +
  & qll(ip1,1,kpl,3)*c11 + qll(il,1,k1,3)*c01

  rem = repf * thkm / 12.
  red = repf * thkd / 12.
  rex = repf * dist / 12.
  write(26,9001) thkm,thkd,rem,red,rex,repf
endif
write(13,9001) y(1),y(2),y(3),dist,hmet,surf
write(14,9002) dist/12.,hmet/12.,surf/sstag
write(15) y(1),y(2),y(3),-xnp,-ynp,-znp,vel,surf,hmet,isav,ksav
write(17,9010) y(1),y(2),y(3),y(1)-xnp,y(2)-ynp,y(3)-znp
write(6,*) ' i,k = ',isav,ksav
write(6,*) y(1),y(2),y(3),y(4),y(5),dist,hmet,surf
if(.not. back) then
  if(i .le. I) go to 230
endif
go to 210

220 continue
if(back) then
  rewind 14
  rewind 13
  rewind 15
if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
  rewind 26
endif
dist=sqrt((y(1)-xstag)**2+(y(2)-ystag)**2+(y(3)-zstag)**2)
write(13,*) ' Streamline No. ', nl
write(14,*) ' Streamline No. ', nl
write(6,*) ' Streamline No. ', nl
back = .false.
second = .true.
xin = xsave
yin = ysave
zin = zsave
hsave = hs
go to 195
endif
continue

call subroutine heat to evaluate convective heating
call heat(qw, qwbtu, wltemp, cf)
rewind 13
rewind 14
rewind 15
rewind 6
rewind 24
rewind 25
write(30,*) istart, kstart, qw, qwbtu
if(ifatal .ne. 0) then
  write(40,*) ' Location ', istart, kstart, ' failed'
  ifatal = 0
  go to 240
endif
    if(itype .ne. 0) then
q(istart,1,kstart,1) = qwbtu
q(istart,1,kstart,2) = wltemp
q(istart,1,kstart,3) = deltal
q(istart,1,kstart,4) = thetl
q(istart,1,kstart,5) = cf
endif
continue
if(istart .eq. ibmax) then
  if(kstart .eq. kbmax) then
    failed'
  else
    kstart = kstart + 1
  endif
else
  kstart = kstart + 1
if(kstart .gt. kbmax) then
  kstart = kbmin
  istart = istart + 1
endif
endif
hsave = saveh
if(itype .eq. 0 .and. .not. fext) then
  jdim3d = 1
  write(16) idim, jdim3d, kdim
  write(16) ((xyz(i, j, k, 1), i=1, idim), j=1, jdim3d, k=1, kdim),
  & ((xyz(i, j, k, 2), i=1, idim), j=1, jdim3d, k=1, kdim),
  & ((xyz(i, j, k, 3), i=1, idim), j=1, jdim3d, k=1, kdim)
c
  write(17) idim, jdim3d, kdim
  write(17) aminf, alp, re, time
  write(17) ((q(i, j, k, nx), i=1, idim), j=1, jdim3d, k=1, kdim),
  & nx=1,5)
endif

9001 format(6(1x, e11.4,1x))
9002 format(3(1x, e15.6,',',1x))
9010 format(6(e12.5,1x))
9999 continue
stop
end

subroutine rk34

subroutine rk34(n,x,y,h,e,hmin)

common /save/ igsav, issav, jsav, ksav, asav, bsav, gsav,
  & first, back, second, ifatal

dimension y(50), ytemp(50), yhat(50), f(50), a(50),
  & b(50), c(50), d(50)
logical second, first, back

data c0, c2, c3, ch0, ch2, ch3, ch4, a1, a2, a3, b10, b20, b21,
  & b30, b31, b32, b40, b42, b43/
  & .1612244898, .5998345284, .2389409818, .1557823129, .6205184777, .1681436539,
  & .0555555556, .2857142857, .4666666667, .9210526316,
  & .2857142857, .0855555556, .3811111111,
  & .5574792244, -1.406455023, 1.770028430, .1612244898,
  & .5998345284, .2389409818/

d continue
  xtemp = x
  do 2 i = 1, n
    ytemp(i) = y(i)
 2 continue
  call funct(xtemp, ytemp, f)
  if(ifatal .ne. 0) return
  xtemp = x + al*h
  do 3 i = 1, n
    a(i) = h * f(i)
    ytemp(i) = y(i) + b10*a(i)
 3 continue
  call funct(xtemp, ytemp, f)
  if(ifatal .ne. 0) return
  xtemp = x + a2*h
  do 4 i = 1, n
    b(i) = h * f(i)
    ytemp(i) = y(i) + b20*a(i) + b21*b(i)
 4 continue
  call funct(xtemp, ytemp, f)
  if(ifatal .ne. 0) return
  xtemp = x + a3*h
  do 5 i = 1, n
    c(i) = h * f(i)
    ytemp(i) = y(i) + b30*a(i) + b31*b(i) + b32*c(i)
 5 continue
call funct(xtemp,ytemp,f)
if(ifatal.ne.0) return
xtemp = x + h
do 6 i = 1,n
d(i) = h * f(i)
ytemp(i) = y(i) + b40*a(i) + b42*c(i) + b43*d(i)
6 continue
call funct(xtemp,ytemp,f)
if(ifatal.ne.0) return
do 7 i = 1,n
ytemp(i) = y(i) + c0*a(i) + c2*c(i) + c3*d(i)
yhat(i) = y(i) + ch0*a(i) + ch2*c(i) + ch3*d(i) + ch4*h*f(i)
a(i) = abs(yhat(i) - ytemp(i))
7 continue
do 8 i = 1,n
c(i) = abs(yhat(i))
if(c(i) .le. a(i)*l.e-3) c(i) = 1.
c if(abs(f(i)) .le. l.e-5) c(i) = 1.
b(i) = a(i) / c(i)
if(.not. second .and. b(i) .gt. e) go to 11
8 continue

c 9 continue
x = x + h
etemp = e / 16.
iflag = 0
do 10 i = 1,n
y(i) = yhat(i)
if(b(i) .gt. etemp) iflag = 1
10 continue
call funct(x,y,f)
if(ifatal.ne.0) return
if(iflag .eq. 1) return
h = h + h
if(h .gt. 1.0e-3) h = 1.0e-3
return
11 continue
if(abs(h) .gt. abs(hmin)) go to 12
write(6,*) 'relative truncation error criterion could
& not be satisfied/",e,hmin
stop
go to 9
12 continue
h = h * 0.5
write(6,*) 'h = ",h
go to 1
end

subroutine funct

subroutine funct(x,y,f)
common /nomxyz/ xn(151,110),yn(151,110),zn(151,110),xnp,ynp,znp
common /sfbond/ msp(151),msr(151),nsp(151),nsr(151)
common /grdxyz/ xyz (151,40,110), idim,jdim,kdim,
& iblank (151,40,110), blank,isubs(2),
& jsubs(2),ksubs(2)
common /qxyz/ u(151,40,110),v(151,40,110),
& w(151,40,110), p(151,40,110)
common /dvdwyz/ dvdy(151,110),dvdz(151,110),dwdy(151,110),
& dwdz(151,110)

modification of 8/16/93
common /duxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
& dwdx(151,110),dwdx(151,110)
end of modification
common /geom/ dx, dy(151,110), dz(151,110)
common /surfpr/ vel, surfn, up, vp, wp, surfp, dxz, dy, dz
common /save/ ig, isav, jasav, jsav, kgsav, asav, bsav, gsav,
& first, back, second, ifatal
logical first, back, second

c
dimension y(50), f(50)

c
sx = y(1)
sy = y(2)
sz = y(3)

c surface interpolation

c
ms = isav - 3
me = isav + 3
if (ms .lt. 1) ms = 1
if (me .gt. idim) me = idim
ns = kgsav - 3
ne = kgsav + 3
if (ns .lt. 1) ns = 1
if (ne .gt. kdim) ne = kdim

c
ns = 1
ne = kdim
call fndnrp(ms, me, ns, ne, sx, sy, sz)

c
istat = 0
delet = 0.
call quads(delxi, delet, sx, sy, sz, istat)
if (istat .gt. 1) then
write (6, *) ' Search failed, istat = ', istat
ifatal = 1
return
endif
il = isav
kl = kgsav
if (first) then
if (kl .eq. 1 .or. kl .eq. kdim) then
delet = 0.
endif
endif
delxi = delxi
det = delet
c11 = delxi * delet
c10 = delxi - c11
c01 = delet - cl1
c00 = 1. - delxi - delet + c11

c
ipl = msp(il)
kpl = msp(kl)
up = u(i1, il, kl) * c00 + u(i1, il, kpl) * c10 + u(i1, kpl, kl) * c01
& + u(i1, il, kpl) * c11
vp = v(i1, il, kl) * c00 + v(i1, il, kpl) * c10 + v(i1, kpl, kl) * c01
& + v(i1, il, kpl) * c11
wp = w(i1, il, kl) * c00 + w(i1, il, kpl) * c10 + w(i1, kpl, kl) * c01
& + w(i1, il, kpl) * c11
surf = p(i1, il, kl) * c00 + p(i1, il, kpl) * c10 + p(i1, kpl, kl) * c01
& + p(i1, il, kpl) * c11
dvdyp = dvdy(i1, il, kl) * c00 + dvdy(i1, il, kpl) * c10 + 
& dvdy(i1, kpl, kl) * c01 + dvdy(i1, kpl, kpl) * c11
& dvdz(i1, il, kl) * c00 + dvdz(i1, il, kpl) * c10 + 
& dvdz(i1, kpl, kl) * c01 + dvdz(i1, kpl, kpl) * c11
dwdyp = dwdy(i1, il, kl) * c00 + dwdy(i1, il, kpl) * c10 + 
& dwdy(i1, kpl, kl) * c01 + dwdy(i1, kpl, kpl) * c11
& dwdz(i1, il, kl) * c00 + dwdz(i1, il, kpl) * c10 + 
& dwdz(i1, kpl, kl) * c01 + dwdz(i1, kpl, kpl) * c11
modification of 8/16/93

dudxp = dudx(i1,kl)*c00 + dudx(ipl,kl)*c10 +
& dudx(ipl,kpl)*c11 + dudx(il,kpl)*c01
dudyp = dudy(il,kl)*c00 + dudy(ipl,kl)*c10 +
& dudy(ipl,kpl)*c11 + dudy(il,kpl)*c01
dudzp = dudz(il,kl)*c00 + dudz(ipl,kl)*c10 +
& dudz(ipl,kpl)*c11 + dudz(il,kpl)*c01
dvdxp = dvdx(il,kl)*c00 + dvdx(ipl,kl)*c10 +
& dvdx(ipl,kpl)*c11 + dvdx(il,kpl)*c01
dwdxp = dwdx(il,kl)*c00 + dwdx(ipl,kl)*c10 +
& dwdx(ipl,kpl)*c11 + dwdx(il,kpl)*c01
dxdyp = dxdy(il,kl)*c00 + dxdy(ipl,kl)*c10 +
& dxdy(ipl,kpl)*c11 + dxdy(il,kpl)*c01
dxdzp = dxdz(il,kl)*c00 + dxdz(ipl,kl)*c10 +
& dxdz(ipl,kpl)*c11 + dxdz(il,kpl)*c01

surface norm vector

xnp = xn(il,kl)*c00 + xn(ipl,kl)*c10 +
& xn(ipl,kpl)*c11 + xn(il,kpl)*c01
ynp = yn(il,kl)*c00 + yn(ipl,kl)*c10 +
& yn(ipl,kpl)*c11 + yn(il,kpl)*c01
znpp = zn(il,kl)*c00 + zn(ipl,kl)*c10 +
& zn(ipl,kpl)*c11 + zn(il,kpl)*c01

vel = sqrt(up*up + vp*vp + wp*wp)
surf = sqrt(1. + dxdyp*dxdyp + dxdzp*dxdzp)

f(I) = up
f(2) = vp
f(3) = wp
f(4) = dvdyp * y(4) + dvdzp * y(5)
f(5) = dvdyp * y(4) + dwdzp * y(5)
f(6) = dvdyp * y(4) + dvdyp * y(4) + dwdzp * y(5)
if(back) then
  f(1) = -f(1)
  f(2) = -f(2)
  f(3) = -f(3)
endif
write(6,*), f(1), f(2), f(3), f(4), f(5)
return
end

subroutine bounds ( mdim, ndim, ms, me, ns, ne,
& msp, rsper, nsper)
common /sfbond/ msp(151), msr(151), nsp(151), nsr(151)

Initialize arrays.

do 10 m = 1, mdim
  msp(m) = 0
  msr(m) = 0
  10 continue

do 11 n = 1, ndim
  nsp(n) = 0
  nsr(n) = 0
  11 continue

Set up +/- index arrays.

do 20 m = ms, me
msp(m) = m+1
msr(m) = m-1
continue
if (msper.eq.0) then
    msp(me) = me
    msr(ms) = ms
else
    msp(me) = ms+1
    msr(ms) = me-1
endif

do 21 n = ns,ne
    nsp(n) = n+1
    nsr(n) = n-1
continue
if (nsper.eq.0) then
    nsp(ne) = ne
    nsr(ns) = ns
else
    nsp(ne) = ns+1
    nsr(ns) = ne-1
endif
return
end

subroutine surmet (mdim,ndim,ms,me,ns,ne,msper,nsper)

Compute metrics for defining surface grid.

common /nomxyz/ x(151,110),y(151,110),z(151,110),xnp,ynp,znp
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
                   & iblank(151,40,110), blank, isubs(2),
                   & jsubs(2), ksubs(2)
common /sfxiet/ xix(151,110),xiys(151,110),xizs(151,110),
                 & etxs(151,110),etys(151,110),etzs(151,110),
                 & flgs(151), flgsn(151)
common /sfbond/ msp(151),msr(151),nsp(151),nsr(151)
common /qxyz/ u(151,40,110),v(151,40,110),
              & w(151,40,110), p(151,40,110)
common /dvwxyz/ dvdy(151,110),dvdz(151,110),dwdy(151,110),
                & dwdz(151,110)

modification of 8/16/93

common /duxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
               & dvdx(151,110), dwdx(151,110)
end of modification
common /geom/ dx(151,110),dy(151,110)
dimension jacm(151),jacn(151)

njac = 0

initialize arrays.

do 10 m = 1,mdim
    flgsm(m) = 0.
    jacm(m) = 0
    jacn(m) = 0
10 continue

do 11 n = 1,ndim
    flgsn(n) = 0.
11 continue

do 12 m = 1,mdim
    do 12 n = 1,ndim
        xixs(m,n) = 0.
12 continue
xiys (m,n) = 0.
xizs (m,n) = 0.
etxs (m,n) = 0.
etys (m,n) = 0.
etzs (m,n) = 0.
dvdy (m,n) = 0.
dvdz (m,n) = 0.
dwdy (m,n) = 0.
dwdx (m,n) = 0.
dwdx (m,n) = 0.
dxdz (m,n) = 0.

12 continue

c Set up weight functions.

do 20 m = ms,me
   flgsm (m) = .5
   continue
20
   if (msper.eq.O) then
      flgsm (me) = 1.
      flgsm (ms) = 1.
   endif

do 21 n = ns,ne
   flgsn (n) = .5
   continue
21
   if (nsper.eq.O) then
      flgsn (ns) = 1.
      flgsn (ne) = 1.
   endif

c Compute metrics.

do 30 m = ms,me
   mp = msp (m)
   mr = msr (m)
   do 30 n = ns,ne
      np = nsp (n)
      nr = nsr (n)
      xxi = (xyz(mp, l,n,1) - xyz(mr, l,n,1)) * flgsm (m)
      yxi = (xyz(mp, l,n,2) - xyz(mr, l,n,2)) * flgsm (m)
      zxi = (xyz(mp, l,n,3) - xyz(mr, l,n,3)) * flgsm (m)
      vxi = (v(mp, l,n) - v(mr, l,n)) * flgsm (m)
      wxi = (w(mp, l,n) - w(mr, l,n)) * flgsm (m)
      xet = (xyz(m,l,np,1) - xyz(m,l,nr,1)) * flgsn (n)
      yet = (xyz(m,l,np,2) - xyz(m,l,nr,2)) * flgsn (n)
      zet = (xyz(m,l,np,3) - xyz(m,l,nr,3)) * flgsn (n)
      vet = (v(m,l,np) - v(m,l,nr)) * flgsn (n)
      wet = (w(m,l,np) - w(m,l,nr)) * flgsn (n)
30

modification of 8/16/93

uxi = (u(mp, l,n) - u(mr, l,n)) * flgsm (m)
uet = (u(m,l,np) - u(m,l,nr)) * flgsn (n)

end of modification

c debug print

if(n .eq. ns .or. n .eq. ne) then
   write(48,*), m,n,xxi,yxi,zxi
   write(49,*), m,n,xet,yet,zet
endif

c compute surface norm

if(m .eq. ms) then
xn(m,n) = 1.
yn(m,n) = 0.
zn(m,n) = 0.

else if(m .eq. me .and. n .ne. ns .and. n .ne. ne) then
  x1 = xyz(m-1,1,n+1,1) - xyz(m,1,n,1)
y1 = xyz(m-1,1,n+1,2) - xyz(m,1,n,2)
z1 = xyz(m-1,1,n+1,3) - xyz(m,1,n,3)
x2 = xyz(m,1,n+1,1) - xyz(m-1,1,n,1)
y2 = xyz(m,1,n+1,2) - xyz(m-1,1,n,2)
z2 = xyz(m,1,n+1,3) - xyz(m-1,1,n,3)
xn1 = y1*z2 - y2*z1
yn1 = x2*z1 - x1*z2
zn1 = x1*y2 - x2*y1

  sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn1 = xn1*sqri
yn1 = yn1*sqri
zn1 = zn1*sqri

else if(n .eq. ns .and. m .ne. ms .and. m .ne. me) then
  x1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1)
y1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2)
z1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3)
x2 = xyz(m,1,n-1,1) - xyz(m-1,1,n,1)
y2 = xyz(m,1,n-1,2) - xyz(m-1,1,n,2)
z2 = xyz(m,1,n-1,3) - xyz(m,1,n-1,3)
xn2 = y1*z2 - y2*z1
yn2 = x2*z1 - x1*z2
zn2 = x1*y2 - x2*y1

  sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
xn2 = xn2*sqri
yn2 = yn2*sqri
zn2 = zn2*sqri

  xnn = 0.5*(xn1+xn2)
ynn = 0.5*(yn1+yn2)
zn = 0.5*(zn1+zn2)

  sqri = 1./sqrt(xnn*xnn + ynn*ynn + zn*zn)
xn(m,n) = xnn*sqri
yn(m,n) = ynn*sqri
zn(m,n) = zn*sqri

else if(n .eq. ns .and. m .eq. me) then
  x1 = xyz(m,1,1,n+1,1) - xyz(m,1,n,1)
y1 = xyz(m,1,1,n+1,2) - xyz(m,1,n,2)
z1 = xyz(m,1,1,n+1,3) - xyz(m,1,n,3)
x2 = xyz(m,1,1,n,1) - xyz(m,1,n,1)
y2 = xyz(m,1,1,n,2) - xyz(m,1,n,2)
z2 = xyz(m,1,1,n,3) - xyz(m,1,n,3)
xn1 = y1*z2 - y2*z1
yn1 = x2*z1 - x1*z2
zn1 = x1*y2 - x2*y1

  sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn1 = xn1*sqri
yn1 = yn1*sqri
zn1 = zn1*sqri

  x2 = xyz(m+1,1,n+1,1) - xyz(m,1,n,1)
y2 = xyz(m+1,1,n+1,2) - xyz(m,1,n,2)
z2 = xyz(m+1,1,n+1,3) - xyz(m,1,n,3)
xn2 = y1*z2 - y2*z1
yn2 = x2*z1 - x1*z2
zn2 = x1*y2 - x2*y1

  sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
xn2 = xn2*sqri
yn2 = yn2*sqri
zn2 = zn2*sqri
xnn = 0.5*(xnl+xn2)
ynn = 0.5*(ynl+yn2)
ynn = 0.
zn = 0.5*(zn1+zn2)
sqri = 1./sqrt(xnn*xnn + ynn*ynn + zn*znn)

xn(m,n) = xnn*sqri
yn(m,n) = ynn*sqri
zn(m,n) = zn*sqri

else if(n .eq. ne .and. m .ne. ms .and. m .ne. me) then

x1 = xyz(m+1,l,n-l,1) - xyz(m,l,n,1)
y1 = xyz(m+1,l,n-l,2) - xyz(m,l,n,2)
z1 = xyz(m+1,l,n-l,3) - xyz(m,l,n,3)
x2 = xyz(m,l,n-l,1) - xyz(m+1,l,n,1)
y2 = xyz(m,l,n-l,2) - xyz(m+1,l,n,2)
z2 = xyz(m,l,n-l,3) - xyz(m+1,l,n,3)
xn1 = yl*z2 - y2*zl
yn1 = x2*zl - xl*z2
zn1 = xl*y2 - x2*yl

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)

xn1 = xn1*sqri
yn1 = yn1*sqri
zn1 = zn1*sqri

else if(m .eq. me .and. n .eq. ns) then

x1 = xyz(m-1,l,n+l,1) - xyz(m,l,n,1)
y1 = xyz(m-1,l,n+l,2) - xyz(m,l,n,2)
z1 = xyz(m-1,l,n+l,3) - xyz(m,l,n,3)
x2 = xyz(m,l,n+l,1) - xyz(m-1,l,n,1)
y2 = xyz(m,l,n+l,2) - xyz(m-1,l,n,2)
z2 = xyz(m,l,n+l,3) - xyz(m-1,l,n,3)
xn2 = yl*z2 - y2*zl
yn2 = x2*zl - xl*z2
zn2 = x1*y2 - x2*yl

sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)

xn2 = xn2*sqri
yn2 = yn2*sqri
zn2 = zn2*sqri

else if(m .eq. me .and. n .eq. ne) then

x1 = xyz(m,l,n-l,1) - xyz(m,l,n,1)
y1 = xyz(m,l,n-l,2) - xyz(m,l,n,2)
z1 = xyz(m,l,n-l,3) - xyz(m,l,n,3)
x2 = xyz(m,l,n+1,1) - xyz(m,l,n,1)
y2 = xyz(m,l,n+1,2) - xyz(m,l,n,2)
z2 = xyz(m,l,n+1,3) - xyz(m,l,n,3)
xn1 = yl*z2 - y2*zl
yn1 = x2*zl - xl*z2
ynl = 0.
zn1 = x1*y2 - x2*yl

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)

xn1 = xn1*sqri
yn1 = yn1*sqri
zn1 = zn1*sqri

else if(m .eq. me .and. n .eq. ne) then

x1 = xyz(m,l,n-1,1) - xyz(m,l,n,1)
y1 = xyz(m,l,n-1,2) - xyz(m,l,n,2)
z1 = xyz(m,l,n-1,3) - xyz(m,l,n,3)
x2 = xyz(m,l,n,1) - xyz(m,l,n,1)
y2 = xyz(m,l,n,2) - xyz(m,l,n,2)
z2 = xyz(m,l,n,3) - xyz(m,l,n,3)

else if(m .eq. me .and. n .eq. ne) then

x1 = xyz(m,l,n-1,1) - xyz(m,l,n,1)
y1 = xyz(m,l,n-1,2) - xyz(m,l,n,2)
z1 = xyz(m,l,n-1,3) - xyz(m,l,n,3)
x2 = xyz(m,l,n,1) - xyz(m,l,n,1)
y2 = xyz(m,l,n,2) - xyz(m,l,n,2)
z2 = xyz(m,l,n,3) - xyz(m,l,n,3)

else if(m .eq. me .and. n .eq. ne) then

x1 = xyz(m,l,n-1,1) - xyz(m,l,n,1)
y1 = xyz(m,l,n-1,2) - xyz(m,l,n,2)
z1 = xyz(m,l,n-1,3) - xyz(m,l,n,3)

else if(m .eq. me .and. n .eq. ne) then

x1 = xyz(m,l,n-1,1) - xyz(m,l,n,1)
y1 = xyz(m,l,n-1,2) - xyz(m,l,n,2)
z1 = xyz(m,l,n-1,3) - xyz(m,l,n,3)
\[ z2 = xyz(m-1,1,n,3) - xyz(m,1,n,3) \]
\[ xn1 = yl*z2 - y2*z1 \]
\[ yn1 = x2*z1 - x1*z2 \]
\[ yn1 = 0. \]
\[ zn1 = x1*y2 - x2*y1 \]
\[ sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1) \]
\[ xn(m,n) = xnl*sqri \]
\[ yn(m,n) = ynl*sqri \]
\[ zn(m,n) = zn1*sqri \]
\[ \text{else} \]
\[ x1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1) \]
\[ y1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2) \]
\[ z1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3) \]
\[ x2 = xyz(m,1,n-1,1) - xyz(m,1,n-1,1) \]
\[ y2 = xyz(m,1,n-1,2) - xyz(m,1,n-1,2) \]
\[ z2 = xyz(m,1,n-1,3) - xyz(m,1,n-1,3) \]
\[ xn1 = yl*z2 - y2*z1 \]
\[ yn1 = x2*z1 - x1*z2 \]
\[ zn1 = x1*y2 - x2*y1 \]
\[ sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1) \]
\[ xn1 = xnl*sqri \]
\[ yn1 = ynl*sqri \]
\[ zn1 = zn1*sqri \]
\[ \text{c} \]
\[ x1 = xyz(m+1,1,n+1,1) - xyz(m,1,n,1) \]
\[ y1 = xyz(m+1,1,n+1,2) - xyz(m,1,n,2) \]
\[ z1 = xyz(m+1,1,n+1,3) - xyz(m,1,n,3) \]
\[ x2 = xyz(m+1,1,n,1) - xyz(m,1,n+1,1) \]
\[ y2 = xyz(m+1,1,n,2) - xyz(m,1,n+1,2) \]
\[ z2 = xyz(m+1,1,n,3) - xyz(m,1,n+1,3) \]
\[ xn3 = yl*z2 - y2*z1 \]
\[ yn3 = x2*z1 - x1*z2 \]
\[ zn3 = x1*y2 - x2*y1 \]
\[ sqri = 1./sqrt(xn3*xn3 + yn3*yn3 + zn3*zn3) \]
\[ xn3 = xn3*sqri \]
\[ yn3 = yn3*sqri \]
\[ zn3 = zn3*sqri \]
\[ \text{c} \]
\[ x1 = xyz(m+1,1,n-1,1) - xyz(m,1,n,1) \]
\[ y1 = xyz(m+1,1,n-1,2) - xyz(m,1,n,2) \]
\[ z1 = xyz(m+1,1,n-1,3) - xyz(m,1,n,3) \]
\[ x2 = xyz(m,1,n-1,1) - xyz(m+1,1,n,1) \]
\[ y2 = xyz(m,1,n-1,2) - xyz(m+1,1,n,2) \]
\[ z2 = xyz(m,1,n-1,3) - xyz(m+1,1,n,3) \]
\[ xn4 = yl*z2 - y2*z1 \]
\[ yn4 = x2*z1 - x1*z2 \]
\[ zn4 = x1*y2 - x2*y1 \]
\[ sqri = 1./sqrt(xn4*xn4 + yn4*yn4 + zn4*zn4) \]
\[ xn4 = xn4*sqri \]
\[ yn4 = yn4*sqri \]
\[ zn4 = zn4*sqri \]
\[ \text{c} \]
\[ x_{nn} = 0.25*(x_{n1}+x_{n2}+x_{n3}+x_{n4}) \]
\[ y_{nn} = 0.25*(y_{n1}+y_{n2}+y_{n3}+y_{n4}) \]
\[ z_{nn} = 0.25*(z_{n1}+z_{n2}+z_{n3}+z_{n4}) \]
\[ s_{qri} = 1./\sqrt{(x_{nn}^2+y_{nn}^2+z_{nn}^2)} \]
\[ x_{n}(m,n) = x_{nn}*s_{qri} \]
\[ y_{n}(m,n) = y_{nn}*s_{qri} \]
\[ z_{n}(m,n) = z_{nn}*s_{qri} \]
\]
\end{c}

\begin{c}
\text{Make sure jacobian is positive. For degenerate or collapsed points, a more rigorous forward or backward scheme should be used. For now, just ignore these points.}
\end{c}

\begin{c}
\text{if (rjac.gt.0. .and. rjac .gt. 1.e-10) then}
\text{if (rjac .gt. 0.) then}
\end{c}

\begin{c}
\text{rd = 1./ rjac}
\text{xixs(m,n) = ( rere*xxi - rxre*xet) * rd}
\text{xiys(m,n) = ( rere*yxi - rxre*yet) * rd}
\text{xizs(m,n) = ( rere*zxi - rxre*zet) * rd}
\text{etxs(m,n) = (-rxre*xxi + rxrx*xet) * rd}
\text{etys(m,n) = (-rxre*yxi + rxrx*yet) * rd}
\text{etzs(m,n) = (-rxre*zxi + rxrx*zet) * rd}
\end{c}

\begin{c}
\text{surface velocity derivatives}
\end{c}

\begin{c}
\text{dvdy(m,n) = vxi*xiys(m,n) + vet*etys(m,n)}
\text{dvdz(m,n) = vxi*xizs(m,n) + vet*etzs(m,n)}
\end{c}

\begin{c}
\text{modification of 8/16/93}
\end{c}

\begin{c}
\text{dudx(m,n) = uxi*xixs(m,n) + uet*etxs(m,n)}
\text{dudy(m,n) = uxi*xiys(m,n) + uet*etys(m,n)}
\text{dudz(m,n) = uxi*xizs(m,n) + uet*etzs(m,n)}
\text{dwdx(m,n) = wxi*xixs(m,n) + wet*etxs(m,n)}
\text{dwdx(m,n) = wxi*xiys(m,n) + wet*etys(m,n)}
\text{dwdx(m,n) = wxi*xizs(m,n) + wet*etzs(m,n)}
\end{c}

\begin{c}
\text{end of modification}
\end{c}

\begin{c}
\text{surface derivatives}
\end{c}

\begin{c}
\text{dx dy(m,n) = xxixiys(m,n) + xet*etxs(m,n)}
\text{dx dz(m,n) = xxixizs(m,n) + xet*etzs(m,n)}
\end{c}

\begin{c}
\text{if(n .eq. ns .or. n .eq. ne) then}
\text{write(50,*)) m,n, dvyd(m,n), dwdz(m,n)}
\text{write(51,*)) m,n, dwdy(m,n), dwdx(m,n)}
\text{write(52,*)) m,n, dx dy(m,n), dwdx(m,n)}
\end{c}

\begin{c}
\text{else}
\text{njac = njac + 1}
\text{jacm(njac) = m}
\text{jacn(njac) = n}
\text{xixs(m,n) = 0.}
\text{xiys(m,n) = 0.}
\text{xizs(m,n) = 0.}
\text{etxs(m,n) = 0.}
\text{etys(m,n) = 0.}
\text{etzs(m,n) = 0.}
\text{dvyd(m,n) = 0.}
\text{dwdy(m,n) = 0.}
\end{c}
\[ dwdz(m,n) = 0. \]
\[ dx(y)(m,n) = 0. \]
\[ dx(z)(m,n) = 0. \]

30  continue

Write questionable grid points to output log file.

if (njac.ne.0) then
  open(2,file='surmet.log',status='unknown',form='formatted')
  do 40 n = 1,njac
    write(2,1010) jacm(n),jacn(n)
  continue
  close(2)
endif

1010 format('Metric computation for grid point ('i2','i2, & failed. This point will be ignored.')

Find point on surface subset that is closest to \((xp, yp, zp)\).

ds = 1.0e8

do 10 n = ns,ne
  do 10 m = ms,me
    dist = (xyz(m,1,n,1)-xp)**2 + (xyz(m,1,n,2)-yp)**2 + (xyz(m,1,n,3)-zp)**2
    if (dist.lt.dsave) then
      dsave = dist
      msave = m
      nsave = n
    endif
  continue
  write(6,*') ' msave, nsave = ',msave, nsave
end

Find quadrant in which the point \((xp,yp,zp)\) lies.
jstart = js
kstart = ks
5 continue

iterm = ((jsmax + ksmax)/2) + 2

do 10 iter = 1,iterm

   delxi = 0.5
   delet = 0.5
   if(js .eq. 1) then
      delxi = .25
      delet = .25
   endif
   do 11 it = 1,10
      if(js .eq. 1) then
         c11 = 1 - delxi - delet
         c12 = delet
         c13 = delxi
         jp = jsp(js)
         kp = ksp(ks)
         xt = xyz(js,1,ks,1)*c11 + xyz(jp,1,ks,1)*c12
            + xyz(jp,1,kp,1)*c13
         yt = xyz(js,1,ks,2)*c11 + xyz(jp,1,ks,2)*c12
            + xyz(jp,1,kp,2)*c13
         zt = xyz(js,1,ks,3)*c11 + xyz(jp,1,ks,3)*c12
            + xyz(jp,1,kp,3)*c13
         xt = xyz(js,1,ks,1)*c11 + xyz(jp,1,ks,1)*c12
            + xyz(jp,1,kp,1)*c13
         xix = xixs(js,ks) * c11 + xixs(jp,ks) * c12
            + xixs(jp,kp) * c13
         xiy = xiys(js,ks) * c11 + xiys(jp,ks) * c12
            + xiys(jp,kp) * c13
         xiz = xizs(js,ks) * c11 + xizs(jp,ks) * c12
            + xizs(jp,kp) * c13
         etx = etxs(js,ks) * c11 + etxs(jp,ks) * c12
            + etxs(jp,kp) * c13
         ety = etys(js,ks) * c11 + etys(jp,ks) * c12
            + etys(jp,kp) * c13
         etz = etzs(js,ks) * c11 + etzs(jp,ks) * c12
            + etzs(jp,kp) * c13
      else
         c11 = delxi * delet
         c10 = delxi - c11
         c01 = delet - c11
         c00 = 1 - delxi - delet + c11
         jp = jsp(js)
         kp = ksp(ks)
         xt = xyz(js,1,ks,1) * c00 + xyz(jp,1,ks,1) * c10
            + xyz(jp,1,kp,1) * c01 + xyz(jp,1,ks,1) * c11
         yt = xyz(js,1,ks,2) * c00 + xyz(jp,1,ks,2) * c10
            + xyz(jp,1,kp,2) * c01 + xyz(jp,1,ks,2) * c11
         zt = xyz(js,1,ks,3) * c00 + xyz(jp,1,ks,3) * c10
            + xyz(jp,1,kp,3) * c01 + xyz(jp,1,ks,3) * c11
         xix = xixs(js,ks) * c00 + xixs(jp,ks) * c10
            + xixs(jp,kp) * c11
         xiy = xiys(js,ks) * c00 + xiys(jp,ks) * c10
            + xiys(jp,kp) * c11
         xiz = xizs(js,ks) * c00 + xizs(jp,ks) * c10
            + xizs(jp,kp) * c11
         etx = etxs(js,ks) * c00 + etxs(jp,ks) * c10
            + etxs(jp,kp) * c11
         ety = etys(js,ks) * c00 + etys(jp,ks) * c10
            + etys(jp,kp) * c11
         etz = etzs(js,ks) * c00 + etzs(jp,ks) * c10
      endif
   enddo 11
10 continue
endif

ddelxi = xix * (xp - xt) + xiy * (yp - yt) + xiz * (zp - zt)
ddelet = etx * (xp - xt) + ety * (yp - yt) + etz * (zp - zt)
delxi = delxi + ddelxi
delel = delet + ddelet

c Test for being far off.
if (max(abs(delxi-0.5),abs(delet-0.5)) .gt. 3.5) goto 12

c Close enough?
error = ddelxi**2 + ddelet**2
error = sqrt(error)
if (error .le. 1.e-3) goto 12
continue

12 continue

c Converged to the right cell?
if (max(abs(delxi-0.5),abs(delet-0.5)) .le. 0.5 + toler) goto 20

jso = js
kso = ks
if (delxi .lt. 0.) then
  if (abs(delxi) .gt. toler) then
    js = jsr(js)
  endif
endif
if (delxi .gt. 1.) then
  if (delxi-1. .gt. toler) then
    js = jsp(js)
  endif
endif
if (delet .lt. 0.) then
  if (abs(delet) .gt. toler) then
    ks = ksr(ks)
  endif
endif
if (delet .gt. 1.) then
  if (delet-1. .gt. toler) then
    ks = ksp(ks)
  endif
endif

if (js,ks) didn't change, we hit the grid boundary. Limit delxi,delet
to the range [0,1].
if (js.eq.jso.and.ks.eq.kso) then
  delxi = min(max(delxi,0.),1.)
  delet = min(max(delet,0.),1.)
  istat = 1
  write(6,*) ' delxi,delet',delxi,delet
  goto 20
endif

continue
write(6,*) 'delxi,delet',delxi,delet
C Convergence failed. (Hope this never happens, but note it.)
C
ifail = ifail + 1
if (ifail.gt.3) then
  istat = 2
  goto 20
endif
js = jstart - 1
if(js .le. 0) js = 1
ks = kstart - 1
if(ks .le. 0) ks = 1
goto 5
20 continue
C write(6,*) 'js,ks', js,ks
C write(6,*) 'delxi delet',delxi,delet
return
end

subroutine heat(qw,qwbtu,wltemp,cf)
C
C This program reads in streamline paths and metric coefficients, uses
C Zoby's approximate heating methodology and used Gupta's curve fit to
C compute equilibrium air thermodynamics and transport properties to
C evaluate convective heating along the streamline.
C
common /connty/ il,jl,kl
common /stagpt/ xstag,ystag,zstag
common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzta(8)
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
& iblank(151,40,110), blank,isubs(2),
& jsubs(2),ksubs(2)
common /qxyz/ u(151,40,110),v(151,40,110),
& w(151,40,110), p(151,40,110)
common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
& first,back,second,ifatal
common /freest/ pinf,tinf,rhoinf,vinf,vninf,cpinf,igas
common /turbml/ iturb
common /radeq/ itwall,wallt,irad,eps
common /tempwl/ twall(1000)
C
C boundary and momentum thickness
C
common /bltrn/ deltai,thetl
C
dimension xs(1000),ys(1000),zs(1000),xn(1000),yn(1000),
& zn(1000),vels(1000),ps(1000),hmet(1000),
& isaves(1000),ksaves(1000)
C
logical blank,back,first,second
C
conversion factors
C
data cmin,fps,atm,den /2.54, 30.48, 2116.2, .01601846/
data rtok /1.8/
C
data Stefan-Boltzman constant(w/cm**2/k)
C
data sigma /5.67e-12/
C
rewind 15
is = 1
continue
read(15,end=90) xs(is),ys(is),zs(is),xn(is),yn(is),zn(is),
& vels(is),ps(is),hmet(is),isaves(is),ksaves(is)
hmet(is) = hmet(is)*cmin
is = is + 1
go to 80
90 continue
it = 1
92 continue
if(itwall .eq. 1) then
  read(18,*,end=95) twall(it)
twall(it) = twall(it) / rtok
  it = it + 1
  go to 92
endif
95 continue
is = is -1
if(itwall .eq. 0) then
  do 97 iu = 1,is
twall(iu) = wallt/rtok
97 continue
endif
iwit = 0
99 continue
isubs(1) = 1
isubs(2) = idim
jsubs(1) = 1
jsubs(2) = jdim
ksubs(1) = 1
ksubs(2) = kdim
blank = .false.
igsav = 1
issav = 1
asav = 0.
bsav = 0.
gsav = 0.
s = 0.
sumO = 0.
f1 = 0.
ite = 0
iweq = 0
do 200 iter = 1,is
  rewind 6
  write(6,*) ' iter = ',iter
  icon = 0
  cc initial edge velocity
  vele = vels(iter) * vninf
  velet = vele
  cc edge pressure
  pe = ps(iter) * pinf
  cc stagnation enthalpy (cal/g)
  hinf = cpinf*tinf + 0.5*vinf**2*2.3901e-8
  write(6,*) ' hinf(computed) = ',hinf
  cc initial search cell
  isav = isaves(iter)
  jsav = 1
ksav = ksaves(iter)
100 continue

edge enthalpy (cal/g)

he = hinf - 0.5*vele**2*2.3901e-8
he = hinf - 0.5*vele**2*2.3901e-8

get edge property

he = 1.0e-3*he
write(6,*) ' he = ', he
iflag = 1
if(igas .eq. 0) then
  iflag = 0
te = he / .24e-3
else if (he .lt .12) then
  te = he / .24e-3
else
  call enthalpy(pe,he,te,iflag)
  if(ifatal .ne. 0) return
endif

call eqprop(pe,te,cpe,rhoe,xmue,xke,pre,gammae,airme,iflag)
if(ifatal .ne. 0) return

get wall properties (tw is input value)

iflag = 2
if(igas .eq. 0) iflag = 0
tw = twall(iter)
call enthalpy(pe,hw,tw,iflag)
if(ifatal .ne. 0) return
call eqprop(pe,tw,cpw,rhow,xmuw,xkw,prw,gammaw,airmw,iflag)
if(ifatal .ne. 0) return

compute Eckert's reference enthalpy

hstar = 0.5*(he+hw) + .11*sqrt(prw)*vele**2 * 2.3901e-11
hstar = 0.5*(he+hw) + .11*sqrt(prw)*vele**2 * 2.3901e-11

get reference properties

iflag = 1
if(igas .eq. 0) then
  tstar = hstar / .24e-3
  iflag = 0
else
  call enthalpy(pe,hstar,tstar,iflag)
  if(ifatal .ne. 0) return
endif

call eqprop(pe,tstar,cpstar,rhostar,xmustar,xkstar,prstar,gammastar, & armstr,iflag)
if(ifatal .ne. 0) return

compute momentum boundary layer thickness

if(iter .eq. 1) then
  ds = sqrt((xstag-xs(iter))**2 + (ystag-ys(iter))**2 & + (zstag-zs(iter))**2)
ds = ds*cmin
  s = s + ds
sum = 0.5*f2*ds

else
  ds = sqrt((xs(iter)-xs(iter-1))**2 + (ys(iter)-ys(iter-1))**2 & + (zs(iter)-zs(iter-1))**2)


\[ ds = ds \times c_{\text{min}} \]
\[ s = s + ds \]
\[ \text{sum} = \text{sum}0 + 0.5 \times (f1 + f2) \times ds \]
\[ \text{endif} \]
\[ \text{c} \]
\[ \text{check if turbulence heating is required} \]
\[ \text{rem} = 1. \]
\[ c3 = 2. \]
\[ c4 = 0.5 \]
\[ c2 = 0.440896 \]
\[ \text{iterb} = 0 \]
\[ \text{thetlo} = 0. \]
\[ \text{110} \]
\[ \text{continue} \]
\[ \text{c} \]
\[ f2 = \text{rhostr} \times \text{xmustr} \times \text{rem} \times \text{vele} \times \text{hmet(\text{iter})} \times c_{3} \]
\[ \text{endif} \]
\[ \text{c} \]
\[ \text{if(\text{iter} .eq. 1)} \]
\[ \text{sum} = 0.25 \times f2 \times ds \]
\[ \text{sum} = 0.5 \times f2 \times ds \]
\[ \text{else} \]
\[ \text{sum} = 0.5 \times (f1 + f2) \times ds + \text{sum0} \]
\[ \text{endif} \]
\[ \text{c} \]
\[ \text{thetl} = (c2 \times \text{sum}) \times c_{4} / (\text{rho} \times \text{vele} \times \text{hmet(\text{iter})}) \times .93 \]
\[ \text{c} \]
\[ \text{thetl} = (c2 \times \text{sum}) \times c_{4} / (\text{rho} \times \text{vele} \times \text{hmet(\text{iter})}) \]
\[ \text{rth} = \text{rhoe} \times \text{velet} \times \text{thetl} / \text{xmue} \]
\[ \text{if(\text{iturb} .ne. 0)} \]
\[ \text{if(abs(thetl-thetlo) .gt. .01 .or. \text{iterb .eq. 0)} \]
\[ \text{rncap} = 12.67 - 6.5 \times \text{alog10}(\text{rth}) + 1.21 \times (\text{alog10}(\text{rth}))^{2} \]
\[ \text{rncap} = \text{anint}(\text{rncap}) \]
\[ \text{rem} = 2. / (\text{rncap} + 1.0) \]
\[ c5 = 2.2433 + 0.93 \times \text{rncap} \]
\[ c3 = 1.0 + \text{rem} \]
\[ c4 = 1.0 / c3 \]
\[ c1 = ((1.0 / c5) \times (2.2 \times \text{rncap}/(\text{rncap}+1.0))) \times \text{rncap}/((\text{rncap}+1.0)) \]
\[ \text{&} \]
\[ c2 = (1.0 + \text{rem}) \times c1 \]
\[ \text{iterb} = \text{iterb} + 1 \]
\[ \text{write(6,*) \ ' \ iterb', iterb} \]
\[ \text{write(6,*) \ ' \ rncap, rem, c1, c2, c3, c4, c5'} \]
\[ \text{if(\text{iterb} .ge. 20)} \]
\[ \text{write(6,*) \ ' \ Too many iterations !!!'} \]
\[ \text{ifatal} = 1 \]
\[ \text{if(\text{iwit} .eq. 0)} \]
\[ \text{qw = 0.} \]
\[ \text{qwbtu = 0.} \]
\[ \text{wtemp = 0.} \]
\[ \text{cf = 0.} \]
\[ \text{endif} \]
\[ \text{return} \]
\[ \text{endif} \]
\[ \text{endif} \]
\[ \text{c} \]
\[ \text{f2 = rhostr \times xmustr \times vele \times hmet(\text{iter})} \times 2 \]
\[ \text{c} \]
\[ \text{thetl} = .664 \times \text{sqrt}(\text{sum}) / (\text{rho} \times \text{vele} \times \text{hmet(\text{iter})}) \times .93 \]
\[ \text{c} \]
\[ \text{write(6,*) \ ' \ Momentum thickness = ', thetl} \]
\[ \text{c} \]
\[ \text{compute boundary layer thickness} \]
\[ \text{c} \]
\[ \text{if(\text{iturb .eq. 0)}} \]
\[ \text{deltal = 5.55 \times thetl} \]
else
  haw = he+0.5*sqrt(prw)*vele**2*2.3901e-11
  haw = he+0.5*sqrt(prw)*velet**2*2.3901e-11
  thedd = rncap + 1. + (((rncap+2.)*bw/(rncap*haw)+1.)*
  & (1.1.29*prw**.333*.5*vele**2*2.3901e-11/he))
  & (1.1.29*prw**.333*.5*velet**2*2.3901e-11/he))
  deltal = thetl * thedd
endif

  deltal = 5.55*thetl*1.18
write(6,*) ' Boundary layer thickness = ',deltal

if(icon .eq.1) go to 150

  xble = xs(iter) + xn(iter)*deltal/cmin
  yble = ys(iter) + yn(iter)*deltal/cmin
  zble = zs(iter) + zn(iter)*deltal/cmin
convert back to inch for interpolation

  xble = xble/cmin
  yble = yble/cmin
  zble = zble/cmin
find cell for interpolation

  isrch = 0
  igr = 1
  nsubs = 1
  call close3(isrch,igr,nsubs,i,j,k,a,b,g,xble,yble,zble,isin, & istat)
  if(istat .ne. 0) then
    write(6,*) ' close3 search failed'
    stop
  i = isaves(iter)
  j = jj
  j = jdim - 1
  if(j .eq. 0) j = 1
  k = ksaves(iter)
  if(ksaves(iter) .eq. kdim) k = k - 1
  write(6,*) 'i,j,k = ',i,j,k
else
  write(6,*) 'i,j,k = ',i,j,k
  jj = j
endif

interpolation of edge properties

  il = i
  j = j
  k = k
  call find(xble,yble,zble)
  if(ifatal .ne. 0) return

  velb = vele
  velb = velet
  ipl = il + 1
  jpl = j + 1
  kpl = k + 1
  ue = u(il,jl,kl)*psi(1) + u(ipl,jl,kpl)*psi(2) + u(ipl,jl,kpl)*psi(3) + u(il,jl,kpl)*psi(4) + u(ipl,jl,kpl)*psi(5) + u(ipl,jl,kpl)*psi(6) + u(ipl,jl,kpl)*psi(7) + u(ipl,jl,kpl)*psi(8)
  ve = v(il,jl,kl)*psi(1) + v(ipl,jl,kpl)*psi(2) + v(ipl,jl,kpl)*psi(3) + v(il,jl,kl)*psi(4) + v(ipl,jl,kpl)*psi(5) + v(ipl,jl,kpl)*psi(6) + v(ipl,jl,kpl)*psi(7) + v(ipl,jl,kpl)*psi(8)
  we = w(il,jl,kl)*psi(1) + w(ipl,jl,kpl)*psi(2) + w(ipl,jl,kpl)*psi(3) + w(il,jl,kl)*psi(4) + w(ipl,jl,kpl)*psi(5) + w(ipl,jl,kpl)*psi(6) + w(ipl,jl,kpl)*psi(7) + w(ipl,jl,kpl)*psi(8)
& + w(i1,j1,kpl)*psi(4)+w(i1,jpl,k1)*psi(5)+w(ipl,jpl,k1)*psi(6)
& + w(ipl,jpl,kpl)*psi(7)+w(i1,jpl,kpl)*psi(8)

pe = p(i1,j1,k1)*psi(1)+p(i1,jpl,k1)*psi(2)+p(i1,jpl,kpl)*psi(3)
& + p(ipl,j1,k1)*psi(4)+p(ipl,jpl,k1)*psi(5)+p(ipl,jpl,kpl)*psi(6)
& + p(ipl,jpl,kpl)*psi(7)+p(i1,jpl,kpl)*psi(8)

c

pe = pe*pinf

vele = sqrt(ue*ue + ve*ve + we*we)

c
use tangential component

velen = ue*xn(iter) + ve*yn(iter) + we*zn(iter)
velet = sqrt(vele*vele - velen*velen)
vele = vele*vninf

vel = vele*30.48

c
write(6,*), ' vele,velb = ', vele, velb

c
tconv = (vele - velb)/velb
if(abs(tconv) .lt. .001) then
  icon = 1
else
  ire = ite + 1
  if(ite .gt. 25) then
    write(6,*), ' edge failed'
    icon = 1
  endif
endif

go to 100

c compute laminar heating
continue

c momentum thickness Reynolds number

reth = rhoe*vele*thetl/xmue
write(6,*), ' reth = ', reth

c compute internal energy per unit mass (m**2/s**2)
iein = he*4.184e10 - pe*1.01336e6/rhoe
eins = ein * l.e-4
rhoes = rhoe * 1.e3

c use tannelhill's curve fit to evaluate entropy

call tgas2(eins,rhoes,entrpy)
c
c adiabatic wall enthalpy (kcal/g)

haw = he +0.5*sqrt(prw)*vele**2*2.3901e-11
haw = he +0.5*sqrt(prw)*velet**2*2.3901e-11
if(iturb .eq. 0) then
  cf = .44 / reth*(rhostr/rhoe)*(xmustr/xmue)
  if(xs(iter) .le. 128.8) then
    cf = 1.534*(rhoe*xmue)**.43*(xmuw*rhow)**.07
  endif
  cf = .44 / reth
else
  qw = 0.22*rhostr*xmustr*rhoe*vele/(rhoe*xmue)*
  qw = 0.22*rhostr*xmustr*rhoe*velet/(rhoe*xmue)*
  & (haw - hw)/(reth*prw**.6)
else
\[ cf = 2 \cdot \frac{c_1}{(\text{reth}^{\text{rem}})} \cdot \left(\frac{\text{rhostr}}{\text{rhoe}}\right) \cdot \left(\frac{\text{xmustr}}{\text{xmue}}\right)^{\text{rem}} \]

\[ qw = \frac{c_1 \cdot \left(\frac{\text{rhostr}}{\text{rhoe}}\right) \cdot \left(\frac{\text{xmustr}}{\text{xmue}}\right)^{\text{rem}} \cdot \text{rhoe} \cdot \text{vele}}{(\text{haw-hw}) / (\text{prw}^{0.4} \cdot \text{reth}^{\text{rem}})}\]

\[ \text{endif} \]

\[ \text{convert kcal/cm}^2/\text{s to watt/cm}^2 \]

\[ qw = qw \cdot 4.18392e3 \]

\[ \text{compute radiation equilibrium wall temperature} \]

\[ \text{if}(qw \leq 0.0) \text{ then} \]
\[ \text{tweq} = 300.0 \]
\[ \text{qw} = 0.0 \]
\[ \text{else} \]
\[ \text{tweq} = \left(\frac{qw}{(\text{eps} \cdot \text{sigma})}\right)^{0.25} \]
\[ \text{endif} \]

\[ \text{if}(\text{abs}(\text{tweq}-\text{twall}(\text{iter})) \gt 5.) \text{ iweq} = 1 \]
\[ \text{twall}(\text{iter}) = \text{tweq} \]
\[ \text{write}(25,*) \text{ tweq} \]

\[ \text{convert to btu/ft}^2/\text{s} \]

\[ \text{qwbtu} = qw \cdot 0.88 \]
\[ s = s + ds \]
\[ \text{sum} = \text{sum} \]
\[ f_1 = f_2 \]
\[ \text{velets} = \text{velet} \]
\[ \text{ite} = 0 \]
\[ \text{wltemp} = \text{twall}(\text{iter}) \]
\[ \text{fss} = \text{xmue}/\text{xmustr} \]
\[ \text{rethss} = \text{fss} \cdot \text{reth} \]
\[ \text{rethl} = \log_{10}(\text{rethss}) \]

\[ \text{cfc} = (\text{te}/\text{tstar})/(17.08 \cdot \text{rethl} \cdot \text{rethl} + 25.11 \cdot \text{rethl} + 6.012) \]

\[ \text{write}(24,9001) \text{ xs}(\text{iter}), s, qw, qwbtu, cf \]
\[ \text{write}(34,9002) \text{ xs}(\text{iter}), \text{ys}(\text{iter}), \text{zs}(\text{iter}), \text{entrpy} \]

\[ 200 \text{ continue} \]
\[ \text{if}(\text{irad} \neq 0) \text{ then} \]
\[ \text{if}(\text{iweq} \neq 0) \text{ then} \]
\[ \text{iweq} = 0 \]
\[ \text{iwit} = \text{iwit} + 1 \]
\[ \text{if}(\text{iwit} \geq 7) \text{ then} \]
\[ \text{write}(6,*) \text{ ' Wall equilibrium temperature iteration failed'} \]
\[ \text{go to 9999} \]
\[ \text{endif} \]
\[ \text{rewind 25} \]
\[ \text{rewind 24} \]
\[ \text{rewind 34} \]
\[ \text{go to 99} \]
\[ \text{endif} \]

\[ 9001 \text{ format}(5(1x, e14.7)) \]
\[ 9002 \text{ format}(4(1x, e14.7)) \]
\[ 9999 \text{ continue} \]
\[ \text{return} \]
\[ \text{end} \]

\[ \text{SUBROUTINE TGAS2(E,R,S)} \]
\[ \text{INPUTS FOR SUBROUTINE:} \]
\[ E=\text{INTERNAL ENERGY IN (M/SEC)**2} \]
\[ R=\text{DENSITY IN KG/M**3} \]
OUTPUT:
S=ENTROPY IN (M/SEC)**2/K

DATA E0,R0,GASCON/78408.4E00,1.292E00,287.06E00/
RRATIO=R/R0
ERATIO=E/E0
Y=ALOG10(RRATIO)
Z=ALOG10(ERATIO)
IF(ABS(Y+4.5E00).LT.2.5E-02) GO TO 10
IF(ABS(Y+0.5E00).LT.0.5E-02) GO TO 40
IFLAG=-1
GO TO 80
10 IFLAG=0
RSAVE=R
YM=Y
Y=-4.5E00+2.5E-02
YHIGH=Y
R=(10.**Y)*R0
JFLAG=-1
GO TO 80
20 SHIGH=S
Y=-4.5E00-2.5E-02
YLOW=Y
R=(10.**Y)*R0
JFLAG=0
GO TO 80
30 SLOW=S
GO TO 70
40 IFLAG=1
RSAVE=R
YM=Y
Y=-0.5E00+0.5E-02
YHIGH=Y
R=(10.**Y)*R0
JFLAG=-1
GO TO 80
50 SHIGH=S
Y=-0.5E00-0.5E-02
YLOW=Y
R=(10.**Y)*R0
JFLAG=0
GO TO 80
60 SLOW=S
70 S=SLOW+(SHIGH-SLOW)/(YHIGH-YLOW)*(YM-YLOW)
R=RSAVE
RETURN

CONTINUE
IF(Z.LE.0.65E00) GO TO 110
IF(Y.GT.-4.5E00) GO TO 90
IF(Z.GT.3.69E00) WRITE(6,1000) R,E
GAS1=-9.91081E-01-5.00277E00*Y
GAS2=(5.46521E01+5.10144E00*Y)*Z
GAS3=(1.76206E-02+2.12002E-02*Z+1.76358E-03*Y)*Y*Y
GAS4=(-2.97001E01-1.84915E00*Y+5.87892E00*Z)*Z*Z
GO TO 120
90 IF(Y.GT.-0.50E00) GO TO 100
IF(Z.GT.3.4E00) WRITE(6,1000) R,E
GAS1=1.0836E01-4.55524E00*Y
GAS2=(2.96473E01+3.90851E00*Y)*Z
GAS3=(-2.05732E-03+3.65982E-02*Z+5.23821E-03*Y)*Y*Y
GAS4=(-1.67001E01-1.44623E00*Y+3.98307E00*Z)*Z*Z
GO TO 120
100 IF(Z.GT.3.0E00) WRITE(6,1000) R,E
GAS1=2.01858E01-3.13458E00*Y
GAS2=(1.03619E01+1.87767E00*Y)*Z
GAS3=(-1.72922E-01+1.12174E-01*Z+1.28626E-02*Y)*Y*Y
GAS4=(-5.43557E00-8.71048E-01*Y+2.01789E00*Z)*Z
GO TO 120
110 DELTZ=Z-0.4E00
DELTs=(2.5E00*DELTZ-Y)*GASCON*2.302585E00
S=6779.2004E00+DELTs
GO TO 130
120 SNON=GAS1+GAS2+GAS3+GAS4
S=GASCON*SNON
130 IF (IFLAG) 160,140,150
140 IF (JFLAG) 20,30,160
150 IF (JFLAG) 50,60,160
160 CONTINUE
1000 FORMAT(/20X,48HWARNING] OUTSIDE VALIDITY RANGE OF CURVE FIT
*,/20X,5HRHO =,1PE15.8,5X,3HE =,1PE15.8,/) RETURN
END
subroutine enthalpy(pinp,hinp, t,iflag)
c This subroutine takes input pressure and enthalpy, uses Gupta's
 equilibrium air thermodynamic and transport properties curve fit
t to compute temperature.
c iflag = 0, perfect gas
c iflag = 1, pressure and enthalpy are input
c iflag = 2, pressure and temperature are input
c
dimension ptb(7), ah(6,7), bh(6,7), ch(6,7), dh(6,7),
 & temp(6,7), tmax(6,7), ntmax(7)
c
common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav,
 & first,back,second,ifatal
common /coef/ coef1, coef2, coef3, coef4, coef5
common /pass/ ipass
c
data ptb / .0001, .001, .01, .1, 1., 10., 100. /
data ntmax / 6, 6, 5, 4, 4, 3 /
data ipmax / 7 /
data tmax / 2250., 4250., 6750., 10750., 17750., 25000. ,
 & 2250., 4250., 6750., 11750., 18750., 28000.,
 & 2750., 5250., 9750., 17750., 30000., 30001.,
 & 3250., 6250., 15250., 30000., 30001., 30001.,
 & 3750., 8250., 17750., 30000., 30001., 30001.,
 & 4250., 9250., 18750., 30000., 30001., 30001.,
 & 6250., 12750., 30000., 30001., 30001., 30001. /
c
data ah / .128180e1, .125380e2, .426138e2, .885088e1, .151569e2,
 & .101799e2, .902850, .237222e2, .880011e2, -.333238e2, .196866e2,
 & .446849e2, .653358, .431122e1, -.126229e1, .209845e2, .26647e2, 0.,
 & .363885, -.865884e1, -.164319e2, -.207249e2, 0., 0.,
 & .209284, -.171560e2, -.134978e2, -.564265e1, 0., 0.,
 & .124937, -.120314e2, -.913636e1, .639208e1, 0., 0.,
 & -.755123e-2, -.117469e1, -.245329e1, 0., 0., 0./
c
data bh / .121182e2, .720107e2, .123001e3, -.207380e2, -.713138e1,
 & -.161956e2, .839944e1, .118014e3, .213329e3, -.316397e2, -.201771e2,
 & -.141088e3, .596886e1, .267604e2, .113432e2, -.181381e2, -.104256e3,
 & 0., .329839e1, -.208034e2, -.285858, .633182e2, 0., 0.,
 & .187458e1, -.416382e2, .801118e1, .262889e2, 0., 0.,
 & .109286e1, -.229170e2, .113969e2, -.149544e2, 0., 0.,
 & .164258e-1, -.592622e1, .371340e1, 0., 0., 0./
data ch / .424907e2, .148949e3, .121801e3, -.134604e2, -.172524, 
   -.336892e1, .289458e2, .214780e3, .181623e3, -.401000e1, .635249e1, 
   .159412e3, .201689e2, .541203e2, .109117e2, -.399635, .145439e3, 
   0., .110641e2, -.132700e2, .447878e1, -.678713e2, 0., 0., 
   .622153e1, -.332532e2, .192371e1, -.396119e2, 0., 0., 
   .355163e1, -.129249e2, -.259796e1, .882252e1, 0., 0., 
   .366590, -.214181e1, -.288683, 0., 0., 0./

data dh / .665524e2, .133853e3, .509305e2, .166408e1, .643645, 
   .161274e2, .448640e2, .171168e3, .661367e2, .379639e1, -.174347, 
   -.738595e2, .309518e2, .462077e2, .400303e1, .308738e1, -.846045e2, 
   0., 
   .173605e2, .248899e1, .196275e1, .312942e2, 0., 0., 
   .101561e2, -.747816e1, .930272, .251297e2, 0., 0., 
   .617946e1, .262066, .114665e2, .258596e1, 0., 0., 
   .210603e1, .251111e1, .421200, 0., 0., 0./

data eh / .385195e2, .451550e2, .995964e1, .356570e1, .356353e1, 
   -.210068e1, 
   .256452e2, .513939e2, .110476e2, .325469e1, .354258e1, 
   .155141e2, 
   .174843e2, .152182e2, .284253e1, .283981e1, .212051e2, 
   0., 
   .999025e1, .417259e1, .256061e1, -.158288e1, 0., 0., 
   .603650e1, .178858e1, .244209e1, -.207198e1, 0., 0., 
   .386028e1, .235363e1, .236890e1, .107086e1, 0., 0., 
   .195195e1, .212013e1, .239842e1, 0., 0., 0./

c pinps = 0.
if(iflag .eq. 1) write(6,*) 'pinp, hinp = ',pinp,hinp
if(iflag .eq. 0) then
   hinp = .24e-3*t
   return
endif

c find pressure range for interpolation

c if(pinp .lt. 1.e-4) then
   pinps = pinp
   pinp = 1.e-4
endif
if(pinp .lt. 1.e-4) then
   if(iflag .eq. 2 .and. t .le. 500.) then
      hinp = .24e-3*t
   else
      write(6,*) ' Pressure is below the lower bound, p = ',pinp
      ifatal = 1
      return
   endif
else if(pinp .gt. 100.) then
   write(6,*) ' Pressure is above the upper bound, p = ',pinp
   ifatal = 1
   return
endif

c find the index which brace the input pressure

c do 10 i = 1,7
   if(pinp .le. ptb(i)) then
      if(abs(pinp-ptb(i)) .le. 1.e-4) then
         pinps = pinp
      endif
   endif
   if(iflag .eq. 1) write(6,*) 'pinp, hinp = ',pinp,hinp
10   continue
intp = 0
else
intp = 1
endif
ip = i
ipml = ip - 1
go to 20
endif
10 continue
write(6,*) ' Pressure is out of range'
20 continue
c write(6,*) ip, ipml ',ip, ipml
if(iflag .eq. 2) then
xi = alog(t*0.001)
if(intp .eq. 0) go to 50
do 30 i = 1,ntmax(ipml)
if(t .gt. tmax(i,ipml)) go to 30
it = i
go to 40
30 continue
write(6,*) ' Temperature ',t,' is outside the range of',
& ' available data'
ifatal = 1
return
40 continue
if(t .lt. 500.) then
hinp = .24e-3*t
if(pinps .ne. 0.) pinp = pinps
return
endif
hl = ah(it,ipml)*xi**4 + bh(it,ipml)*xi**3 + ch(it,ipml)*
& xi**2 + dh(it,ipml)*xi + eh(it,ipml)
c
50 continue
if(t .lt. 500.) then
hinp = .24e-3*t
if(pinps .ne. 0.) pinp = pinps
return
endif
do 60 i = 1,ntmax(ip)
if(t .gt. tmax(i,ip)) go to 60
it = i
go to 70
60 continue
write(6,*) ' Temperature ',t,' is outside the range of',
& ' available data'
ifatal = 1
return
70 continue
h2 = ah(it,ip)*xi**4 + bh(it,ip)*xi**3 + ch(it,ip)*xi**2
& + dh(it,ip)*xi + eh(it,ip)
c
if(intp .eq. 0) then
alogh = h2
else
alogh = (h2 - hl) * (alog(pinp) - alog(ptb(ipml)))
& /
(alog(ptb(ip)) - alog(ptb(ipml))) + hl
endif
c
hinp = exp(alogh)
go to 200
endif
c
c compute maximum enthalpy for each temperature range for each pressure
c
if(ipass .eq. 0) then
ipass = 1
do 80 j = 1,ipmax
do 80 i = 1,ntmax(j)
   xi = alog(tmax(i,j)*.0001)
   hmax(i,j) = exp(ah(i,j)*xi**4 + bh(i,j)*xi**3 +
                ch(i,j)*xi**2 + dh(i,j)*xi + eh(i,j))
   write(6,*)' hmax ',j,i,hmax(i,j)
80 continue
endif
if(intp .eq. 0) go to 100

c find correct temperature range and curve to solve for temperature
c
ntip = ntmax(ipml)
do 90 i = 1,ntip
   if(hinp .lt. hmax(i,ipml)) then
      coef1 = ah(i,ipml)
      coef2 = bh(i,ipml)
      coef3 = ch(i,ipml)
      coef4 = dh(i,ipml)
      coef5 = eh(i,ipml)
      tmin = 500.
      hmin = .24e-3*tmin
      if(i .gt. 1) then
         tmin = tmax(i-1,ipml)
         hmin = hmax(i-1,ipml)
      endif
      ti = (tmax(i,ipml)+tmin)*.5
      ti=tmin+(alog(hinp)-alog(hmin))*(tmax(i,ipml)-tmin)
      &/(alog(hmax(i,ipml))-alog(hmin))
      call newton(ti,hinp,t1)
      go to 100
   else if(hinp .eq. hmax(i,ipml)) then
      t2 = tmax(i,ipml)
      go to 120
   endif
90 continue

c write(6,*)' Enthalpy is out of range', hinp
ifatal = 1
return
100 continue

c ntip = ntmax(ip)
do 110 i = 1,ntip
   if(hinp .lt. hmax(i,ip)) then
      coef1 = ah(i,ip)
      coef2 = bh(i,ip)
      coef3 = ch(i,ip)
      coef4 = dh(i,ip)
      coef5 = eh(i,ip)
      tmin = 500.
      hmin = .24e-3*tmin
      if(i .gt. 1) then
         tmin = tmax(i-1,ip)
         hmin = hmax(i-1,ip)
      endif
      ti = tmin + (alog(hinp)-alog(hmin))*(tmax(i,ip)-tmin)
      &/(alog(hmax(i,ip))-alog(hmin))
      call newton(ti,hinp,t2)
      if(ifatal .ne. 0) return
      go to 120
   else if(hinp .eq. hmax(i,ip)) then
      t2 = tmax(i,ip)
      go to 120
   endif
110 continue
continue
write(6,*) ' Enthalpy is out of range', hinp
ifatal = 1
return
continue
if(intp .eq. 0) then
    t = t2
    go to 200
endif
use logarithm interpolation to find temperature
alogt = (alog(t2) - alog(t1)) * (alog(pinp) - alog(ptb(ipml)))
    & / (alog(ptb(ip)) - alog(ptb(ipml))) + alog(t1)
t = exp(alogt)
continue
if(pinp .ne. 0.) pinp = pinps
return
end
subroutine newton(ti, hinp, t)
common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav,
    & first, second, ifatal
common /coef/  a, b, c, d, e
xi = alog(ti*.0001)
iter = 0
continue
resl = exp(a*xi**4 + b*xi**3 + c*xi**2 + d*xi + e)
res = resl - hinp
if(abs(res) .lt. 1.0e-4) go to 100
compute derivative
fp = resl * (4.*a*xi**3 + 3.*b*xi**2 + 2.*c*xi + d)
dx = -res/fp
xi = xi + dx
iter = iter + 1
if(iter .gt. 50) then
    write(6,*) ' Newton method failed to converge'
    ifatal = 1
    return
endif
go to 100
continue
t = 10000. * exp(xi)
return
end
subroutine eqprop(pinp, tinp, cp, rho, xmu, xk, pr, gamma, airm, iflag)
This routine uses Gupta's curve fit equation to obtain thermodynamics
and transport properties using logarithm interpolation.
Inputs are pressure (pinp) and temperature (tinp)
common /save/  igsav, issav, isav, jsav, ksav, asav, bsav, gsav,
    & first, second, ifatal
dimension acp(9,7), bcp(9,7), ccp(9,7), dcp(9,7), ecp(9,7),
    & az(5,7), bz(5,7), cz(5,7), dz(5,7), ez(5,7), fmu(4,7),
runiv is in cal/g-mole-k

data airm0, runiv /28.96, 1.987/
data ptb /1.e-4, 1.e-3, 1.e-2, 1.e-1, 1., 10., 100. /
data ncpmax /9, 8, 7, 8, 7, 7, 6/
data nzmax /5, 5, 5, 5, 4, 3, 2/
data nmumax /4, 4, 4, 4, 3, 3, 2/
data nkmax /7, 7, 7, 6, 6, 5, 4/
data nprmax /8, 8, 7, 7, 5, 5/
data ipmax /7/
data tcpmax /& 1250., 1750., 2750., 4750., 6250., 9750., 14250., 19750., 25000.,
& 1250., 2250., 3750., 5250., 7250., 10750., 17250., 28000., 28001.,
& 1750., 2750., 4750., 6750., 12750., 19750., 30000., 30001., 30001.,
& 1750., 2750., 4250., 6750., 9750., 15750., 21500., 30000., 30001.,
& 1750., 3250., 4750., 7750., 11750., 20500., 30000., 30001., 30001.,
& 1750., 3250., 2750., 9250., 13750., 22500., 30000., 30001., 30001.,
& 1750., 3750., 6750., 10750., 17750., 30000., 30001., 30001., 30001./
data tzmax /& 2750., 5750., 8750., 17750., 25000.,
& 3250., 6750., 9750., 19750., 28000.,
& 3250., 7250., 11750., 21500., 30000.,
& 3750., 8250., 13750., 23500., 30000.,
& 5750., 9250., 15750., 23500., 30000.,
& 5750., 9750., 17250., 30000., 30001.,
& 8750., 17750., 30000., 30001., 30001./
data tmumax /& 7750., 10750., 16750., 25000.,
& 8250., 12250., 18750., 28000.,
& 8750., 14250., 19750., 30000.,
& 9750., 16750., 24500., 30000.,
& 11250., 19750., 30000., 30001.,
& 12750., 21500., 30000., 30001.,
& 15250., 30000., 30001., 30001./
data tkmax /& 1750., 2750., 4750., 6250., 10250., 17750., 25000.,
& 1750., 2750., 4750., 6250., 11250., 18250., 28000.,
& 2250., 3250., 5750., 7750., 12750., 18750., 30000.,
& 2250., 4250., 6750., 9250., 16750., 30000., 30001.,
& 2250., 4250., 7750., 10750., 19250., 30000., 30001.,
& 3250., 5250., 8750., 13750., 30000., 30001., 30001.,
& 3750., 6250., 10750., 30000., 30001., 30001., 30001./
data tprmax /& 2250., 3750., 5750., 8250., 10750., 14750., 18250., 25000.,
& 2250., 4750., 7250., 10250., 12750., 17250., 20500., 28000.,
& 2750., 5250., 8250., 11750., 14250., 18250., 23500., 30000.,
& 2750., 5250., 7750., 13750., 18250., 25500., 30000., 30001.,
& 2750., 4750., 7750., 13250., 17750., 23500., 30000., 30001.,
& 2750., 5750., 10750., 20500., 30000., 30001., 30001., 30001.,
& 2750., 6750., 12750., 20500., 30000., 30001., 30001., 30001./
data acp /& .349023, .152264e2, -.159675e2, -.108293e3, -.116264e4, 
& -.238707e2, -.209557e2, .762671e3, -.789820e3,
\begin{verbatim}
& .203928e2, -.281133e3, -.107793e3, .457650e2, .180195el,
& .100733e1, .277132e2, 0., 0.,
& .820277e1, -.154364e2, .205793e3, -.168003e2, .838365,
& .987915, .339060e2, 0., 0.,
& .126269e1, .100733e1, .277132e2, 0., 0.,
& .820277e1, -.154364e2, .205793e3, -.168003e2, .838365,
& .987915, .339060e2, 0., 0.,
& .126269e1, .100733e1, .277132e2, 0., 0.,

\textbf{data az} /
& .710750, -.614415el, -.632086e2, -.467833e2, .556705e2,
& .824285, .746758e1, -.385889e2, -.455262e2, .809623e2,
& .873086, -.195828e2, .417508e2, -.208036e3, .915792e2,
& .940213, .124751el, -.325326e2, -.428667e2, .970875,
& .102671e1, .384233ei, .628850el, .743313, .877002,
& .129598e2, 0., 0., 0./

\textbf{data bz} /
& .107229e1, .861656el, .370722e2, .139011e2, -.135009e2,
& .625098, -.460729e2, .209649e2, -.162146e2, .612253,
& .434929, .324383el, .199910e2, .101757e2, -.342626e2,
& .311295, .485004, .137742e2, .888031el, -.309522e2,
& .465274e-1, -.204392e2, .637080el, -.385889e2, -.431463e2,
& .869030e-1, .186655el, .353080el, -.612253, 0.,
& .103304e-1, -.555015el, .202955e2, 0., 0./

\textbf{data cz} /
& -.125673e1, -.370256e1, -.776456el, -.138963e1, .118386el,
& -.689867, .109594el, -.398276el, -.108251e1, .120105el,
& -.454400, -.123210e1, -.334091e1, -.804882, .210825el,
& -.302086, -.321087, -.203163e1, -.620696, .165245el,
& -.972123e-2, .404607e1, -.827695, -.208573, .758996,
& -.737745e-1, -.540958, -.389667, .997321e-1, 0.,
& .237877e-1, -.115055, .203092, 0., 0./

\textbf{data dz} /
& .564944, .681208, .706484, .592861e-1, -.427210e-1,
& .286982, -.89428e-1, .327436, .415356e-1, -.375039e-1,
& .176448, .198816, .243749, .274096e-1, -.563525e-1,
& .107468, .632573e-1, .130377, .188157e-1, -.384201e-1,
& .417402e-2, -.344141, .466769e-1, .395832e-2, .194343e-1,
& .218303e-1, .639254e-1, .187431e-1, -.411847e-2, 0.,
& -.281715e-2, .324023e-2, -.525620e-2, 0., 0./

\textbf{data ez} /
& -.822333e-1, -.443045e-1, -.233636e-1, -.903887e-3, .551468e-3,
& -.376727e-1, .162238e-2, -.970559e-2, -.569596e-3, .421422e-3,
& -.212727e-1, -.111471e-1, -.644569e-2, -.334336e-3, .555405e-3,
& -.116924e-1, -.364522e-2, -.302863e-2, -.206237e-3, .330019e-3,
& -.536830e-3, .107287e-1, -.941988e-3, -.175392e-4, -.182292e-3,
& -.179762e-2, -.255478e-2, -.320898e-3, .542207e-4, 0.,
& .168221e-3, -.18832e-4, .489857e-4, 0., 0./

\textbf{data amu} /
& -.1160076e-4, -.9105422, .1463029e-1, -.2140374e-2,
& .2397194e-4, -.5784272, .1658118e-1, .6903134e-2,
& .5085043e-4, -.341487, .2450600e-1, -.3561146e-1,
& .6394112e-4, -.2376368, .6309492e-3, -.1622687e1,
& .5781887e-4, -.1844238, .2606784e-1, 0.,
& .7256455e-4, -.9524274e-1, .5037513e-1, 0.,
& .7609039e-4, -.7868582e-1, 0., 0./

\textbf{data bmu} /
\end{verbatim}
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.6656010e-3</td>
<td>.4949794</td>
<td>-.5019958e-2</td>
<td>.6529285e-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5564725e-3</td>
<td>.2816531</td>
<td>-.5027652e-2</td>
<td>-.1345295e-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4774840e-3</td>
<td>.1473594</td>
<td>-.6972242e-2</td>
<td>.7255623e-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4385020e-3</td>
<td>.9006170e-1</td>
<td>.6108099e-3</td>
<td>.3035173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4438221e-3</td>
<td>.6040101e-1</td>
<td>-.4562535e-2</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4050530e-3</td>
<td>.2589951e-1</td>
<td>-.8081647e-2</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.3891948e-3</td>
<td>.1820922e-1</td>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-.2933969e-3</td>
<td>-.1060568</td>
<td>.6886543e-3</td>
<td>-.7290226e-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.1970968e-3</td>
<td>-.5377449e-1</td>
<td>.6106363e-3</td>
<td>-.1061916e-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.1322133e-3</td>
<td>-.2471167e-1</td>
<td>.7362709e-3</td>
<td>-.5837678e-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.1024141e-3</td>
<td>-.1315352e-1</td>
<td>-.1286661e-3</td>
<td>-.226401e-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.1020840e-3</td>
<td>-.7566737e-2</td>
<td>.3111533e-3</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.7626766e-4</td>
<td>-.2593217e-2</td>
<td>.5209350e-3</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.6458779e-4</td>
<td>-.1543467e-2</td>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.7427050e-4</td>
<td>.1123425e-1</td>
<td>-.4723839e-4</td>
<td>.3865996e-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4272210e-4</td>
<td>.5058384e-2</td>
<td>-.3715711e-4</td>
<td>-.4234384e-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.2362256e-4</td>
<td>.2030404e-2</td>
<td>-.4070960e-4</td>
<td>-.2324393e-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1654305e-4</td>
<td>.9370344e-3</td>
<td>.9381977e-5</td>
<td>-.8445985e-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1688754e-4</td>
<td>.4609058e-3</td>
<td>-.1018512e-4</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.1114437e-4</td>
<td>.1227957e-3</td>
<td>-.1682098e-4</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.8791566e-5</td>
<td>.6257766e-4</td>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-.6456605e-5</td>
<td>-.5896774e-3</td>
<td>.1623374e-5</td>
<td>-.9908122e-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.2690853e-5</td>
<td>-.2352317e-3</td>
<td>.8514686e-7</td>
<td>-.4324384e-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.8014978e-6</td>
<td>-.8216415e-4</td>
<td>.1134307e-5</td>
<td>-.4590857e-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.5014106e-6</td>
<td>-.3279124e-4</td>
<td>-.2906969e-6</td>
<td>-.1570909e-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.8622324e-6</td>
<td>-.1377229e-4</td>
<td>.1576999e-6</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.5020411e-6</td>
<td>-.2772500e-5</td>
<td>.2731352e-6</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.4216496e-6</td>
<td>-.1234998e-5</td>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.8752161e-7</td>
<td>.1229026e-4</td>
<td>-.2239581e-7</td>
<td>.9916638e-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.3009241e-7</td>
<td>.4336410e-5</td>
<td>-.1397984e-7</td>
<td>-.6893227e-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.6458338e-7</td>
<td>-.1314540e-5</td>
<td>-.1276018e-7</td>
<td>.3600777e-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.3710875e-7</td>
<td>-.4529650e-6</td>
<td>-.3444222e-8</td>
<td>-.1166667e-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.2239193e-9</td>
<td>-.9623637e-6</td>
<td>-.9011456e-9</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.7047486e-10</td>
<td>-.2383398e-7</td>
<td>-.1794872e-8</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-.4509800e-8</td>
<td>.9579999e-8</td>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.395299e1</td>
<td>.119879e2</td>
<td>-.832682e2</td>
<td>-.103603e4</td>
<td>.261125e2</td>
<td></td>
</tr>
<tr>
<td>.246095e2</td>
<td>-.571805e2</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.199665e1</td>
<td>-.831120e2</td>
<td>-.110139e3</td>
<td>.299875e3</td>
<td>.434485e2</td>
<td></td>
</tr>
<tr>
<td>.895136e1</td>
<td>.422029e2</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.198558e1</td>
<td>.595832e2</td>
<td>-.442143e2</td>
<td>-.584437e3</td>
<td>.373716e2</td>
<td></td>
</tr>
<tr>
<td>.143675e2</td>
<td>.502985e1</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.105928e1</td>
<td>.101351e3</td>
<td>.830640e1</td>
<td>-.318301e3</td>
<td>.469099e2</td>
<td></td>
</tr>
<tr>
<td>.154279e2</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.334316</td>
<td>.109992e2</td>
<td>.124072e2</td>
<td>-.189644e3</td>
<td>.298795e2</td>
<td></td>
</tr>
<tr>
<td>.844897e1</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.413573</td>
<td>.821188e2</td>
<td>.113875e2</td>
<td>-.723261e2</td>
<td>-.382696e1</td>
<td></td>
</tr>
<tr>
<td>0.</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.208749</td>
<td>.378677e2</td>
<td>.223116e2</td>
<td>-.792550e1</td>
<td>0.</td>
<td>0.</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.386816e2</td>
<td>.412181e2</td>
<td>-.419438e3</td>
<td>-.242470e4</td>
<td>.411940e1</td>
<td></td>
</tr>
<tr>
<td>.507490e2</td>
<td>.166828e3</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.194822e2</td>
<td>-.560438e3</td>
<td>-.481050e3</td>
<td>.923042e3</td>
<td>.464790e1</td>
<td></td>
</tr>
<tr>
<td>.322183e2</td>
<td>.144838e3</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.189164e2</td>
<td>.288748e3</td>
<td>-.206207e3</td>
<td>-.873106e3</td>
<td>-.115449e2</td>
<td></td>
</tr>
<tr>
<td>.801073e1</td>
<td>-.960227e1</td>
<td>0.</td>
<td>0.</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Values</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ck</strong></td>
<td>.100924e2, .490653e3, -.324274e2, -.306306e3, -.330961e2, -.541310e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.990653e3, -.324274e2, -.306306e3, -.330961e2, -.541310e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.541310e2, .990653e3, -.324274e2, -.306306e3, -.330961e2, .192122e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.123284e2, .336369, -.216552e2, 0., 0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>dk</strong></td>
<td>.140687e3, .711756e1, -.751764e3, -.206135e4, -.186054e2, .359131e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.181577e3, .757873e3, .992814e3, -.155778e2, -.305726e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.164202e2, .196818e1, .500756e3, -.324643e3, -.445088e3, -.113653e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.356709e2, .868620e3, -.942568e2, -.782124e2, -.166070e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.693640e2, .119939e2, .387282e2, -.530293e2, .998789e1, .117041e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.553921e2, .200944e2, .548304e1, -.299886e2, .277739e1, .122011e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.353787e2, .207305e2, .250668e3, -.122339e2, -.233991e1, .107119e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.658813e1, .144224e3, -.142705e2, .204578e2, 0., 0., 0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ek</strong></td>
<td>.127138e3, .810415e2, -.157470e3, -.108281e3, -.621476e1, .623025e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.213335e2, .599079e2, -.632398e3, -.125800e3, .634709e2, -.558790e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.498154e1, .270417e2, .527822e2, .844428e2, -.494556e2, -.128529e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.389761e1, -.896353e1, .249670e2, .180596e3, -.180857e2, -.585083e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.115271e1, .462882e1, -.120166e2, -.961485e1, -.581069e1, .738431e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.274595e1, .427728e1, .475889e2, -.610064e1, -.556444e1, -.717162e1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.658813e1, .684087e1, -.498832e1, -.485454e1, 0., 0., 0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>apr</strong></td>
<td>.7695318, .4081926e2, -.5060907e3, .3172744e2, -.7624622e4, .1034825e3,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.3414530e3, .1398841e2, .7483071, .4355608e2, .6877578e3, -.7024989e3,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.3107328e3, .9153227e2, .2304245e4, -.1424413e2, .7406012, .3206825e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.4409105e3, .2474049e2, -.9639789e3, .9436717e2, -.7123564e2, -.7846718e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.7548377, .6190687e2, -.2641868e3, .7451996e2, .1034771e2, -.1252168e3,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-.5022975e4, 0., .8037721, .9173178e2, .1874806e2, -.1974460e3, .5713095e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.1619271e3, .1996134e3, 0., .8167207, -.1857130e2, -.4287169e2, .6447596e2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.7384318e2, 0., 0., 0., 0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
data fprr /
& .9284843e-1, -.1430131, .2747920, -.5625333e-2, .8174707e-1,
& -.2291528e-3, .4881128e-3, .3740205e-6,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .2544039e-1, -.4395200e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .5734562e-1, -.9453467e-1, .1793128e-2, .3121600e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-5,
& .6541023e-2, -.5926933e-1, .1836850e-4, .2331795e-4,
& .2716070e-4, .3283492e-3, 0.,
& -.2182155e-1,-.1497413,-.3499282e-2, .1622326e-2, -.2015436e-3, 
& -.3092820e-4, .1339615e-4, 0.,
& -.2479595e-1, .1987815e-1, .1844057e-2, -.6127611e-4, -.4987500e-5,
& 0., 0., 0.,
& -.2557178e-1,.4226421e-2,.5732041e-3,.2296542e-4,.4740705e-5,
& 0., 0., 0./

c if temperature is less than 500K

c if(tinp .lt. 500. .or. iflag .eq. 0) then
    cp = .24
    z = 1.
    xmu = 1.4584e-5*tinp**1.5/(tinp+110.33)
    xk = 5.9776e-6*tinp**1.5/(tinp+194.4)
    pr = .24*xmu/xk
    rho = pinp*airm0/(runiv*tinp)*2.4218e-2
    airm = airm0
    gamma = 1./(1.-runiv/airm/cp)
    return
endif

c Find pressure range for interpolation

c if(pinp .lt. 1.e-4 .and. pinp .gt. 100.) then
    write(6,*) ' Input pressure of ',pinp,' is outside range of
    ' available curve fit'
    ifatal = 1
    return
endif

c Find pressures which bound the input pressure

do 10 i = 1,7
    if(pinp .le. ptb(i)) then
        if(pinp .eq. ptb(i)) then
            intp = 0
        else
            intp = 1
        endif
    ip = i
    ipml = ip -1
    go to 20
10 continue

c write(6,*) ' Pressure is outside the range of available data'
    ifatal = 1
    return
20 continue

c xi = alog(tinp*.0001)

c Specific heat

do 30 i = 1,ncpmax(ip)
    if(tinp .gt. tcpmax(i,ip)) go to 30
it = i
30 go to 40

write(6,*) ' Temperature ',tinp,' is outside the range of',
& ' available data - cp high'
ifatal = 1
return
40 continue

ccm2 = acp(it,ip)*xi**4 + bcp(it,ip)*xi**3 + ccp(it,ip)*
& xi**2 + dcp(it,ip)*xi + ecp(it,ip)

if(intp .eq. 0 .or. ipml .le. 0) then
    cp = exp(ccm2)
go to 65
do 50 i = 1,ncpmax(ipml)
    if(tinp .gt. tcpmax(i,ipml)) go to 50
it = i
    go to 60
50 continue

write(6,*) ' Temperature ',tinp, 'is outside the range of',
& ' available data - cp low'
ifatal = 1
return

ccml = acp(it,ipml)*xi**4+bcp(it,ipml)*xi**3+ccp(it,ipml)*xi**2
& + dcp(it,ipml)*xi + ecp(it,ipml)

cc interpolation for specific heat

calogcp = (ccm2 - ccml)/(alog(ptb(ip))-alog(ptb(ipml)))*
& (alog(pinp) - alog(ptb(ipml))) + ccml

cc Compressibility factor

c65 continue
xi = tinp*.001
do 70 i = 1,nzmax(ip)
    if(tinp .gt. tzmax(i,ip)) go to 70
it = i
    go to 80
70 continue

write(6,*) ' Temperature ',tinp, 'is outside the range of',
& ' available data - z high'
ifatal = 1
return

zz2 = az(it,ip) + bz(it,ip)*xi + cz(it,ip)*xi**2
& + dz(it,ip)*xi**3 + ez(it,ip)*xi**4
if(intp .eq. 0 .or. ipml .le. 0) then
    z = zz2
    go to 105
endif

do 90 i = 1,nzmax(ipml)
    if(tinp .gt. tzmax(i,ipml)) go to 90
it = i
    go to 100
90 continue

write(6,*) ' Temperature ',tinp, 'is outside the range of',
& ' available data - z low'
ifatal = 1
return
100 continue
z1 = az(it,ipml) + bz(it,ipml)*xi + cz(it,ipml)*xi**2
& + dz(it,ipml)*xi**3 + ez(it,ipml)*xi**4

/* interpolation of compressibility factor */

z = (z2 - zl)/(ptb(ip) - ptb(ipml)) * (pinp - ptb(ipml)) + zl

/* Viscosity */

105 continue
do 110 i = 1,nmumax(ip)
if(tinp .gt. tmumax(i,ip)) go to 110
it = i
go to 120
110 continue

write(6,*),' Temperature ',tinp, 'is outside the range of',
& ' available data - mu high'
ifatal = 1
return
120 continue

xmu2 = amu(it,ip) + bmu(it,ip)*xi + cmu(it,ip)*xi**2
& + dmu(it,ip)*xi**3 + emu(it,ip)*xi**4 + fmu(it,ip)*xi**5
if(intp .eq. 0 .or. ipml .le. 0) then
  xmu = xmu2
  go to 145
endif

do 130 i = 1,nmumax(ip)
if(tinp .gt. tmumax(i,ipml)) go to 130
it = i
go to 140
130 continue

write(6,*),' Temperature ',tinp, 'is outside the range of',
& ' available data - mu low'
ifatal = 1
return
140 continue

xmul = amu(it,ipml) + bmu(it,ipml)*xi + cmu(it,ipml)*xi**2
& + dmu(it,ipml)*xi**3 + emu(it,ipml)*xi**4 + fmu(it,ipml)*xi**5

/* interpolation of viscosity */

xmu = (xmu2 - xmul)/(ptb(ip) - ptb(ipml)) * (pinp - ptb(ipml))
& + xmul

/* Thermal conductivity */

145 continue
xi = alog(tinp*.0001)
do 150 i = 1,nkmax(ip)
if(tinp .gt. tkmax(i,ip)) go to 150
it = i
go to 160
150 continue

write(6,*),' Temperature ',tinp, 'is outside the range of',
& ' available data - K high'
ifatal = 1
return
160 continue

xk2 = ak(it,ip)*xi**4 + bk(it,ip)*xi**3 + ck(it,ip)*xi**2
& \quad \text{ksi}^2 + \text{dk}(\text{it}, \text{ip})^2 \text{ksi} + \text{ek}(\text{it}, \text{ip})

\text{if(intp .eq. 0 .or. ipml .le. 0) then}
\text{xk} = \exp(xk2)
\text{go to 185}
\text{endif}

do 170 i = 1, nkmax(ipml)
\text{if(tinp .gt. tkmax(i, ipml)) go to 170}
\text{it} = i
\text{go to 180}
170 
\text{continue}

\text{write(6,*) ' Temperature ',tinp, 'is outside the range of', '
& available data - K low'}
\text{ifatal = 1}
\text{return}

180 
\text{continue}
\text{xk1} = \text{ak}(\text{it}, \text{ipml})^2 \text{ksi}^4 + \text{bk}(\text{it}, \text{ipml})^2 \text{ksi}^3 + \text{ck}(\text{it}, \text{ipml})^2 \text{ksi}^2
& + \text{dk}(\text{it}, \text{ipml})^2 \text{ksi} + \text{ek}(\text{it}, \text{ipml})
\text{c}
\text{interpolation for thermal conductivity}
\text{c}
\text{alogk} = (xk2 - xk1)/(\text{alog(pptb(ip))-alog(pptb(ipml))})*
& (\text{alog(pinp) - alog(pptb(ipml))}) + xk1
\text{xk} = \exp(alogk)
\text{c}
\text{Prandtl number}
\text{c}
185 
\text{continue}
\text{xi} = \text{tinp} \times .001
\text{do 190 i = 1, nprmax(ip)}
\text{if(tinp .gt. tprmax(i, ip)) go to 190}
\text{it} = i
\text{go to 200}
190 
\text{continue}
\text{c}
\text{write(6,*) ' Temperature ',tinp, 'is outside the range of', '
& available data - Pr high'}
\text{ifatal = 1}
\text{return}

200 
\text{continue}
\text{pr2} = \text{apr}(\text{it}, \text{ip}) + \text{bpr}(\text{it}, \text{ip})^2 \text{ksi} + \text{cpr}(\text{it}, \text{ip})^2 \text{ksi}^2
& + \text{dpr}(\text{it}, \text{ip})^2 \text{ksi}^3 + \text{epr}(\text{it}, \text{ip})^2 \text{ksi}^4 + \text{fpr}(\text{it}, \text{ip})^2 \text{ksi}^5
\text{if(intp .eq. 0 .or. ipml .le. 0) then}
\text{pr} = \text{pr2}
\text{go to 300}
\text{endif}

\text{c}
\text{do 210 i = 1, nprmax(ipml)}
\text{if(tinp .gt. tprmax(i, ipml)) go to 210}
\text{it} = i
\text{go to 220}
210 
\text{continue}
\text{write(6,*) ' Temperature ',tinp, 'is outside the range of', '
& available data - Pr low'}
\text{ifatal = 1}
\text{return}

220 
\text{continue}
\text{prl} = \text{apr}(\text{it}, \text{ipml}) + \text{bpr}(\text{it}, \text{ipml})^2 \text{ksi} + \text{cpr}(\text{it}, \text{ipml})^2 \text{ksi}^2
& + \text{dpr}(\text{it}, \text{ipml})^2 \text{ksi}^3 + \text{epr}(\text{it}, \text{ipml})^2 \text{ksi}^4 + \text{fpr}(\text{it}, \text{ipml})^2 \text{ksi}^5
\text{c}
\text{interpolation of Prandtl number}
\text{c}
\text{pr} = (\text{pr2} - \text{prl})/(\text{ptb}(ip) - \text{ptb}(ipml)) \times (\text{pinp} - \text{ptb}(ipml))
& + \text{prl}
\text{c}
compute density

```
rho0 = pinn*airm0/(runiv*tinp)*2.4218e-2
if(z .lt. 1.) z = 1.
rho = rho0/z
```

compute molecular weight and effective gamma

```
airm = airm0/z
gamma = 1./(1.-runiv/airm/cp)
```

write(6,*) ' cp = ',cp
write(6,*} ' z = ',z
write(6,*) ' rho = ',rho
write(6,*) ' mu = ',xmu
write(6,*) ' k = ',xk
write(6,*) ' pr = ',pr

return
end

subroutine find(x0, y0, z0)

Given a location (x0,y0,z0), and the grid cell containing this point,
this routine uses Newton's method to find corresponding location
(xi,eta,zeta) in the master element.

```
common /pos/ xi,eta,zeta
common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzeta(8)
common /connty/ il,jl,k1
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
 & isub0(151,40,110), blank,isubs(2),
 & jsubs(2),ksubs(2)
common /save/ igsav,issav,jsav,ksav,asav,bsav,gsav,
 & first,second,ifatal
logical blank,first,second,back
dimension x(8), y(8), z(8), r(3), fj(3,3), dr(3)
```

set initial guess at xi=0, eta=0 and zeta=0

```
write(6,*) 'x0,y0,z0 = ',x0,y0,z0
xi = 0.
eta = 0.
zeta = 0.
ipl = il + 1
jpl = jl + 1
kpl = kl + 1

x(1) = xyz(il,jl,kl,1)
y(1) = xyz(il,jl,kl,2)
z(1) = xyz(il,jl,kl,3)
```

```
x(2) = xyz(ipl,jl,kl,1)
y(2) = xyz(ipl,jl,kl,2)
z(2) = xyz(ipl,jl,kl,3)
```

```
x(3) = xyz(ipl,jl,kpl,1)
y(3) = xyz(ipl,jl,kpl,2)
z(3) = xyz(ipl,jl,kpl,3)
```

```
x(4) = xyz(il,jl,kpl,1)
y(4) = xyz(il,jl,kpl,2)
z(4) = xyz(il,jl,kpl,3)
```

\[
x(5) = xyz(i_1, j_1, k_1, 1) \\
y(5) = xyz(i_1, j_1, k_1, 2) \\
z(5) = xyz(i_1, j_1, k_1, 3) \\
\]
\[
x(6) = xyz(i_2, j_1, k_1, 1) \\
y(6) = xyz(i_2, j_1, k_1, 2) \\
z(6) = xyz(i_2, j_1, k_1, 3) \\
\]
\[
x(7) = xyz(i_2, j_1, k_1, 1) \\
y(7) = xyz(i_2, j_1, k_1, 2) \\
z(7) = xyz(i_2, j_1, k_1, 3) \\
\]
\[
x(8) = xyz(i_1, j_1, k_1, 1) \\
y(8) = xyz(i_1, j_1, k_1, 2) \\
z(8) = xyz(i_1, j_1, k_1, 3) \\
\]
\[
\text{iter} = 0 \\
\]
\[
10 \text{ continue} \\
\]
\[
\text{compute shape function and it's derivatives} \\
\]
\[
\text{call shape} \\
\]
\[
\text{initialize jacobian matrix} \\
\]
\[
\text{do 20 j = 1,3} \\
\text{do 20 i=1,3} \\
fj(i,j) = 0. \\
20 \text{ continue} \\
\]
\[
r(1) = x_0 \\
r(2) = y_0 \\
r(3) = z_0 \\
\]
\[
\text{get jacobian matrix} \\
\]
\[
\text{do 30 i = 1,8} \\
fj(1,i) = fj(1,i) + x(i)*dpdx(i) \\
fj(2,i) = fj(2,i) + y(i)*dpdy(i) \\
fj(3,i) = fj(3,i) + z(i)*dpdz(i) \\
20 \text{ continue} \\
\]
\[
\text{det} = fj(1,1)*fj(2,2)*fj(3,3) + fj(1,2)*fj(2,3)*fj(3,1) \\
+ fj(1,3)*fj(2,1)*fj(3,2) - fj(3,1)*fj(2,2)*fj(1,3) \\
- fj(1,2)*fj(2,1)*fj(3,3) - fj(1,1)*fj(3,2)*fj(2,3) \\
\text{write(6,*)) ' determinant of Jacobian = ' det} \\
\text{res = sqrt(r(1)*r(1) + r(2)*r(2) + r(3)*r(3))} \\
\text{if(res .lt. 1.0e-4) go to 9999} \\
\text{idm = 3} \\
\text{call gauss(fj,r,dr,idm)} \\
\text{if(ifatal .ne. 0) return} \\
\text{write(6,*)) dr} \\
x1 = x1 + dr(1) \\
eta = eta + dr(2) \\
zeta = zeta + dr(3) \\
\text{if(sdr .ge. 1.e-3) then} \\
\text{iter = iter + 1} 
\]
if (iter .gt. 40) then
    write(6,*), ' !!! iteration limit exceeded !!!'
    write(6,*), ' sdr = ',sdr
    ifatal = 1
    return
else
    go to 10
endif
endif
9999 continue
if(abs(xi) .gt. 1.) xi = sign(1.,xi)
if(abs(eta) .gt. 1.) eta = sign(1.,eta)
if(abs(zeta) .gt. 1.) zeta = sign(1.,zeta)
call shape
write(6,*), ' xi,eta, zeta = ',xi,eta,zeta
return
end
subroutine shape

This routine evaluate 3-D linear shape functions and their derivatives.

common /pos/ xi,eta,zeta
common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzta(8)

omxi = 1. - xi
opxi = 1. + xi
ometa = 1. - eta
op eta = 1. + eta
omzeta = 1. - zeta
opzeta = 1. + zeta

omxiet = omxi * ometa
opxime = opxi * ometa
opxiet = opxi * op eta
omxipe = omxi * op zeta

psi(1) = .125 * omxiet * om z eta
psi(2) = .125 * opxime * om z eta
psi(3) = .125 * opxiet * om z eta
psi(4) = .125 * omxipe * om z eta
psi(5) = .125 * omxiet * op zeta
psi(6) = .125 * opxime * op zeta
psi(7) = .125 * opxiet * op zeta
psi(8) = .125 * omxipe * op zeta

ometze = ometa * om z eta
ometpz = ometa * op zeta
opetze = op eta * op zeta
opetmz = op eta * om z eta
omxize = omxi * om z eta
omxipz = omxi * op zeta
opxize = op xi * op zeta
opximz = op xi * om z eta

dpdxi(1) = -.125 * ometze
dpdxi(2) = -dpdxi(1)
dpdxi(3) = .125 * opetmz
dpdxi(4) = -dpdxi(3)
dpdxi(5) = -.125 * ometpz
dpdxi(6) = -dpdxi(5)
dpdxi(7) = .125 * opetze
dpdxi(8) = -dpdxi(7)

dpd eta(1) = -.125 * om xize
\[
\begin{align*}
\text{dpdeta}(2) &= -0.125 \times \text{opximz} \\
\text{dpdeta}(3) &= -\text{dpdeta}(2) \\
\text{dpdeta}(4) &= -\text{dpdeta}(1) \\
\text{dpdeta}(5) &= -0.125 \times \text{omxipz} \\
\text{dpdeta}(6) &= -0.125 \times \text{opxize} \\
\text{dpdeta}(7) &= -\text{dpdeta}(6) \\
\text{dpdeta}(8) &= -\text{dpdeta}(5)
\end{align*}
\]

\[
\begin{align*}
\text{dpdzta}(1) &= -0.125 \times \text{omxiet} \\
\text{dpdzta}(2) &= -0.125 \times \text{opxime} \\
\text{dpdzta}(3) &= -0.125 \times \text{omxiet} \\
\text{dpdzta}(4) &= -\text{dpdzta}(1) \\
\text{dpdzta}(5) &= -\text{dpdzta}(2) \\
\text{dpdzta}(6) &= -\text{dpdzta}(3) \\
\text{dpdzta}(7) &= -\text{dpdzta}(4) \\
\text{dpdzta}(8) &= -\text{dpdzta}(5)
\end{align*}
\]

```
SUBROUTINE GAUSS(FJAC,C,X,IDIM)
    common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav, 
                   first, back, second, ifatal

    C GAUSS ELIMINATION SOLVER FOR SYSTEM AX=C
    C
    C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    DIMENSION FJAC(3,3),c(3),X(3)

    C FORWARD ELIMINATION
    DO 30 I = 1, IDIM-1
       IP1 = I + 1
       IF(FJAC(I,I) .EQ. 0.) THEN
         WRITE(6,9006) I
         ifatal = 1
         return
       ENDIF
       DO 20 J = IP1, IDIM
          IF(FJAC(J,I) .EQ. 0.) GO TO 20
          DO 10 K = IP1, IDIM
             FJAC(J,K) = FJAC(J,K) - FJAC(I,K)*FJAC(J,I)/FJAC(I,I)
          10 CONTINUE
          C(J) = C(J) - C(I)*FJAC(J,I)/FJAC(I,I)
       20 CONTINUE
    30 CONTINUE

    C BACK SUBSTITUTION
    DO 70 I = IDIM, 1,-1
       X(I) = 0.
       DO 60 J = I, IDIM
          C(I) = C(I) - FJAC(I,J)*X(J)
       60 CONTINUE
       X(I) = C(I) / FJAC(I,I)
    70 CONTINUE

9006 FORMAT(' *** ZERO PIVOT, I = ',I3)
RETURN
END
```

```
SUBROUTINE CELL3 (IS,IE,JS,JE,KS,KE,

************************************************************************
************************************************************************

C Find the point (x,y,z) in the grid XYZ and return its (i,j,k) cell
```
number. We assume that the given (i, j, k), (a, b, g), and subset are
valid. These can be checked with STRTxx.

STATUS=1 - Unable to find the point without going out of the
computational domain or active subset. The computational point returned indicates the direction to look..

common /grdxyz/ xyz(151, 40, 110, 3), idim, jdim, kdim,
& iblank(151, 40, 110), blank, isubs(2),
& jsubs(2), ksubs(2)
DIMENSION AMAT(3,3), BMAT(3,3)
LOGICAL BLANK
INTEGER STATUS

PARAMETER (MITER=5)
LOGICAL DIALOG
INTEGER DI, DJ, DK

DATA ISAV1, JSAV1, KSAV1/3*0/, ISAV2, JSAV2, KSAV2/3*0/
DATA DIALOG/.FALSE./
DATA DIALOG/.TRUE./

STATUS=0

Reset saved points used for checking if we’re stuck on the same
point (or ping-ponging back and forth) when searching for the right
cell. This would happen while following a point outside the
computational domain.

ISAV1 = 0
ISAV2 = 0
DI = 1
DJ = 1
DK = 1

Maximum number of steps before we’ll give up is max(IDIM, JDIM, KDIM).

MSTEP = MAX(IDIM, JDIM, KDIM)
NSTEP = 0

20 CONTINUE
NSTEP = NSTEP+1

if(i .ge. idim) i = idim - 1
if(j .ge. jdim) j = jdim - 1
if(k .ge. kdim) k = kdim - 1
I1 = I+1
J1 = J+1
K1 = K+1

X1 = XYZ(I, J, K, 1)
X2 = XYZ(I1, J, K, 1)
X3 = XYZ(I, J1, K, 1)
X4 = XYZ(I1, J1, K, 1)
X5 = XYZ(I, J, K1, 1)
X6 = XYZ(I1, J, K1, 1)
X7 = XYZ(I, J1, K1, 1)
X8 = XYZ(I1, J1, K1, 1)

Y1 = XYZ(I, J, K, 2)
Y2 = XYZ(I1, J, K, 2)
Y3 = XYZ(I, J1, K, 2)
Y4 = XYZ(I1, J1, K, 2)
Y5 = XYZ(I, J, K1, 2)
Y6 = XYZ(I1, J, K1, 2)
Y7 = XYZ(I, J1, K1, 2)
Y8 = XYZ(I1, J1, K1, 2)
Z1 = XYZ(I, J, K, 3)
Z2 = XYZ(I1, J, K, 3)
Z3 = XYZ(I, J1, K, 3)
Z4 = XYZ(I1, J, K1, 3)
Z5 = XYZ(I, J, K1, 3)
Z6 = XYZ(I, J1, K1, 3)
Z7 = XYZ(I1, J1, K1, 3)
Z8 = XYZ(I1, J1, K1, 3)

X0 = X1
XA = X2 - X1
XB = X3 - X1
XG = X5 - X1
XAB = X4 - X3 - X2 + X1
XAG = X6 - X5 - X2 + X1
XBG = X7 - X5 - X3 + X1
XABG = X8 - X7 - X6 + X5 - X4 + X3 + X2 - X1

Y0 = Y1
YA = Y2 - Y1
YB = Y3 - Y1
YG = Y5 - Y1
YAB = Y4 - Y3 - Y2 + Y1
YAG = Y6 - Y5 - Y2 + Y1
YBG = Y7 - Y5 - Y3 + Y1
YABG = Y8 - Y7 - Y6 + Y5 - Y4 + Y3 + Y2 - Y1

Z0 = Z1
ZA = Z2 - Z1
ZB = Z3 - Z1
ZG = Z5 - Z1
ZAB = Z4 - Z3 - Z2 + Z1
ZAG = Z6 - Z5 - Z2 + Z1
ZBG = Z7 - Z5 - Z3 + Z1
ZABG = Z8 - Z7 - Z6 + Z5 - Z4 + Z3 + Z2 - Z1

A = .5
B = .5
G = .5
ITER = 0

30 CONTINUE
ITER = ITER + 1
XH = X0 + XA*A + XB*B + XG*G + XAB*(A*B) + XAG*(A*G) + XBG*(B*G) + XABG*(A*(B*G))
YH = Y0 + YA*A + YB*B + YG*G + YAB*(A*B) + YAG*(A*G) + YBG*(B*G) + YABG*(A*(B*G))
ZH = Z0 + ZA*A + ZB*B + ZG*G + ZAB*(A*B) + ZAG*(A*G) + ZBG*(B*G) + ZABG*(A*(B*G))

AMAT(1, 1) = XA + XAB*B + XAG*G + XABG*(B*G)
AMAT(2, 1) = YA + YAB*B + YAG*G + YABG*(B*G)
AMAT(3, 1) = ZA + ZAB*B + ZAG*G + ZABG*(B*G)

AMAT(1, 2) = XB + XAB*A + XBG*G + XABG*(A*G)
AMAT(2, 2) = YB + YAB*A + YBG*G + YABG*(A*G)
AMAT(3, 2) = ZB + ZAB*A + ZBG*G + ZABG*(A*G)

AMAT(1, 3) = XG + XAG*A + XB*B + XABG*(A*B)
AMAT(2, 3) = YG + YAG*A + YBG*B + YABG*(A*B)
AMAT(3, 3) = ZG + ZAG*A + ZBG*B + ZABG*(A*B)

CALL INV3X3(AMAT, BMAT, ISTAT)
IF (ISTAT.NE.0) THEN
  IF (DIALOG) WRITE(6,*) 'DEGENERATE VOLUME AT INDEX ',I,J,K
See if we’re at the edge of the cell. If so, move away from the (possibly degenerate) edge and recompute the matrix.

\[
\begin{align*}
AS &= A \\
BS &= B \\
GS &= G \\
\text{IF} \ (A \cdot EQ. 0.) \ A &= .01 \\
\text{IF} \ (A \cdot EQ. 1.) \ A &= .99 \\
\text{IF} \ (B \cdot EQ. 0.) \ B &= .01 \\
\text{IF} \ (B \cdot EQ. 1.) \ B &= .99 \\
\text{IF} \ (G \cdot EQ. 0.) \ G &= .01 \\
\text{IF} \ (G \cdot EQ. 1.) \ G &= .99 \\
\text{IF} \ (A \cdot NE. AS \ .OR. B \cdot NE. BS \ .OR. G \cdot NE. GS) \ \text{THEN} \ \text{GOTO 30}
\end{align*}
\]

We’re inside a cell and the transformation matrix is singular. Move to the next cell and try again.

\[
\begin{align*}
\text{ELSE} \\
A &= DI+.5 \\
B &= DJ+.5 \\
G &= DK+.5 \\
\text{GOTO} \ 40 \\
\end{align*}
\]

\[
\begin{align*}
\text{DX} &= X-XH \\
\text{DY} &= Y-YH \\
\text{DZ} &= Z-ZH \\
\text{DA} &= \text{DX} \ast \text{BMAT(1,1)} + \text{DY} \ast \text{BMAT(1,2)} + \text{DZ} \ast \text{BMAT(1,3)} \\
\text{DB} &= \text{DX} \ast \text{BMAT(2,1)} + \text{DY} \ast \text{BMAT(2,2)} + \text{DZ} \ast \text{BMAT(2,3)} \\
\text{DG} &= \text{DX} \ast \text{BMAT(3,1)} + \text{DY} \ast \text{BMAT(3,2)} + \text{DZ} \ast \text{BMAT(3,3)} \\
A &= A+DA \\
B &= B+DB \\
G &= G+DG \\
\end{align*}
\]

If we’re WAY off, don’t bother with the error test. In fact, go ahead and try another cell.

\[
\begin{align*}
\text{IF} \ (\text{ABS}(A-.5).GT.3. \ .OR. \ \text{ABS}(B-.5).GT.3. \ .OR. \ \text{ABS}(G-.5).GT.3.) \ \text{THEN} \ \text{GOTO} \ 40
\end{align*}
\]

Check iteration error and branch out if it’s small enough.

\[
\begin{align*}
\text{ELSE} \\
\text{ERR2} &= \text{DA} * \text{DA} + \text{DB} * \text{DB} + \text{DG} * \text{DG} \\
\text{IF} \ (\text{ERR2}/3. \ LE. 1. \ E-4) \ \text{GOTO} \ 40 \\
\end{align*}
\]

40 CONTINUE

The point is in this cell.

\[
\begin{align*}
\text{IF} \ (\text{ABS}(A-.5).LE.50005 \ .AND. \ \text{ABS}(B-.5).LE.50005 \ .AND. \ \text{ABS}(G-.5).LE.50005) \ \text{THEN} \\
\text{IF} \ (\text{DIALOG}) \ \text{WRITE}(6,*) \ 'MATCH'
\end{align*}
\]

We’ve taken more steps then we’re willing to wait...

\[
\begin{align*}
\text{ELSE IF} \ (\text{NSTEP}.GT.\text{MSTEP}) \ \text{THEN} \\
\text{STATUS} &= 1 \\
\text{IF} \ (\text{DIALOG}) \ \text{WRITE}(6,*) \ 'MORE \ THAN \ ',\text{MSTEP},' \ STEPS'
\end{align*}
\]

..
Update our \((i,j,k)\) guess, keeping it inbounds.

\[
\text{ELSE} \quad 
\begin{align*}
\text{IN} &= I \\
\text{JN} &= J \\
\text{KN} &= K \\
\text{IF} (A.<0.) \quad \text{IN} &= \text{MAX}(\text{IN}-1, \text{IS}) \\
\text{IF} (A.>1.) \quad \text{IN} &= \text{MIN}(\text{IN}+1, \text{IE}-1) \\
\text{IF} (B.<0.) \quad \text{JN} &= \text{MAX}(\text{JN}-1, \text{JS}) \\
\text{IF} (B.>1.) \quad \text{JN} &= \text{MIN}(\text{JN}+1, \text{JE}-1) \\
\text{IF} (G.<0.) \quad \text{KN} &= \text{MAX}(\text{KN}-1, \text{KS}) \\
\text{IF} (G.>1.) \quad \text{KN} &= \text{MIN}(\text{KN}+1, \text{KE}-1) \\
\text{IF} (g.<0.) \quad \text{JN} &= \text{MAX}(\text{JN}-1, \text{JS}) \\
\text{IF} (g.>1.) \quad \text{JN} &= \text{MIN}(\text{JN}+1, \text{JE}-1) \\
\text{IF} (b.<0.) \quad \text{KN} &= \text{MIN}(\text{KN}+1, \text{KS}) \\
\text{IF} (b.>1.) \quad \text{KN} &= \text{MAX}(\text{KN}-1, \text{KE}-1) \\
\text{IF} (\text{DIALOG}) \quad \text{WRITE}(*,*) '\text{TRY CELL INDEX }, \text{IN}, \text{JN}, \text{KN}'
\end{align*}
\]

Check IBLANK for this cell.

\[
\begin{align*}
\text{IF} (\text{BLANK}) \quad \text{THEN} \\
\text{IN1} &= \text{IN}+1 \\
\text{JN1} &= \text{JN}+1 \\
\text{KN1} &= \text{KN}+1 \\
\text{IF} (\text{IBLANK} (\text{IN}, \text{JN}, \text{KN}) .EQ. 0 .OR. \text{IBLANK} (\text{IN1}, \text{JN1}, \text{KN1}) .EQ. 0 .OR. \text{IBLANK} (\text{IN}, \text{JN1}, \text{KN}) .EQ. 0 .OR. \text{IBLANK} (\text{IN1}, \text{JN}, \text{KN1}) .EQ. 0 .OR. \text{IBLANK} (\text{IN}, \text{JN1}, \text{KN1}) .EQ. 0 .OR. \text{IBLANK} (\text{IN1}, \text{JN1}, \text{KN}) .EQ. 0) \quad \text{THEN} \\
\text{STATUS} &= 1 \\
\text{IF} (\text{DIALOG}) \quad \text{WRITE}(6,*) '\text{EXTRAPOLATE}' \\
\text{GOTO} 50
\end{align*}
\]

Endif

EndIf

Not repeating a previous point. Use the new \((i,j,k)\) and try again.

\[
\begin{align*}
\text{IF} (\text{AND} (\text{IN} .NE. \text{ISAV1} .OR. \text{JN} .NE. \text{JSAV1} .OR. \text{KN} .NE. \text{KSAV1})) \quad \text{THEN} \\
\text{ISAV2} &= \text{ISAV1} \\
\text{JSAV2} &= \text{JSAV1} \\
\text{KSAV2} &= \text{KSAV1} \\
\text{ISAV1} &= \text{IN} \\
\text{JSAV1} &= \text{JN} \\
\text{KSAV1} &= \text{KN} \\
\text{DI} &= \text{ISIGN}(1, \text{IN}-1) \\
\text{DJ} &= \text{ISIGN}(1, \text{JN}-1) \\
\text{DK} &= \text{ISIGN}(1, \text{KN}-1) \\
\text{I} &= \text{IN} \\
\text{J} &= \text{JN} \\
\text{K} &= \text{KN} \\
\text{GOTO} 20
\end{align*}
\]

We've been here before...

\[
\text{ELSE} \\
\text{It seems to be outside the domain. We would have to extrapolate to find it.}
\]

\[
\begin{align*}
\text{STATUS} &= 1 \\
\text{IF} (\text{DIALOG}) \quad \text{WRITE}(6,*) '\text{EXTRAPOLATE}' \\
\text{ENDIF}
\end{align*}
\]

EndIf

50 CONTINUE
RETURN
END

*** End of Subroutine Cell3

***************************************************************

SUBROUTINE CLOSE3(ISRCH,IGRID,NSUBS,IX,JK,KA,B,G, 
X,Y,Z,ISUB,STATUS)
***************************************************************

C

Find the cell containing the given (x,y,z) values and return its 
(i,j,k) indices and subset number. We will use several different 
strategies to try to do this efficiently, based on the value of 
ISRCH:

ISRCH= 0 - General search. 
1 - Search from previous point. 
n - Coming from grid n.

STATUS=1 - (x,y,z) point could not be found.

common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim, 
&        iblank(151,40,110), blank,isubs(2), 
&                     jsubs(2),ksubs(2) 
common /save/ igsav,issav,jsav,ksav,asav,bsav,gsav, 
&                     first,back,second,ifatal 
LOGICAL BLANK, first,second, back 
INTEGER STATUS

STATUS= 0

Search from previous point.

IF (ISRCH.EQ.1) THEN
    IF (IGRID.EQ.IGSAV .AND. ISSAV.GE.1 .AND. ISSAV.LE.NSUBS) THEN
        ISUB = ISSAV
        I = ISA
        J = JSAV
        K = KSA
        A = ASA
        B = BSA
        G = GSA
        is = isubs(1)
        ie = isubs(2)
        js = jsubs(1)
        je = jsubs(2)
        ks = ksubs(1)
        ke = ksubs(2)
        CALL PSRCH3(is,ie, js, je,ks,ke,I,J,K,A,B,G,X,Y,Z,ISTAT)
        IF (ISTAT.EQ.0) GOTO 10
    ENDIF

Try edges next.

    CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
    IF (ISTAT.EQ.0) GOTO 10

Try faces next.

    CALL FSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
    IF (ISTAT.EQ.0) GOTO 10

Try hole boundaries next.
CALL HSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10

Coming from grid n...

ELSE IF (ISRCH.LT.0) THEN

Try edges first.

   CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10

If that fails, try closest point with IBLANK = -ISRCH.

   CALL NSRCH3(-ISRCH,I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10

General search.

ELSE

Try edges.

   CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10

Try faces next.

   CALL FSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10

Try hole boundaries next.

   CALL HSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
   IF (ISTAT.EQ.0) GOTO 10
ENDIF

Nothing worked. Sorry.

STATUS= 1
IGSAV = 0

10 CONTINUE
   IGSAV = IGRID
   ISSAV = ISUB
   ISAV = I
   JSAV = J
   KSAV = K
   ASAV = A
   BSAV = B
   GSAV = G
   RETURN
END

*** End of Subroutine Close3

******************************************************************************************************************
SUBROUTINE DMIN3(IS,IE,JS,JE,KS,KE,X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
******************************************************************************************************************
C
C Add to a list of minimum distance points (i,j,k) to (x,y,z). Note that the calling program is responsible for maintaining D2MIN if the IPTS list is to be added to. Don't overflow IPTS, but don't worry about a message or anything.
common /grdxyz/ xyz(151, 40, 110, 3), idim, jdim, kdim,
&   iblank(151, 40, 110), blank, isubs(2),
&  jsubs(2), ksubs(2)
DIMENSION IPTS(3, MPTS)
LOGICAL BLANK

No previous points. Establish a minimum distance.

IF (NPTS.EQ.0) D2MIN=1.E35
DO 10 K= KS, KE
    DO 10 J= JS, JE
        DO 10 I= IS, IE
            IF (.NOT.BLANK .OR. IBLANK(I, J, K).NE.0) THEN
                D2= (X-XYZ(I, J, K, 1))**2+(Y-XYZ(I, J, K, 2))**2
                    +(Z-XYZ(I, J, K, 3))**2
            ELSE IF (D2.EQ.D2MIN) THEN
                IF (NPTS.LT.MPTS) THEN
                    NPTS= NPTS+1
                    IPTS(1, NPTS)= I
                    IPTS(2, NPTS)= J
                    IPTS(3, NPTS)= K
                ENDIF
            ENDIF
        10 CONTINUE
RETURN
END

*** End of Subroutine DMin3

********************************************************************
SUBROUTINE ESRCH3(I, J, K, A, B, G, X, Y, Z, ISUB, STATUS)
********************************************************************
SEARCH from closest point on any edge.

STATUS=1 - Couldn't find the point.

common /grdxyz/ xyz(151, 40, 110, 3), idim, jdim, kdim,
&   iblank(151, 40, 110), blank, isubs(2),
&  jsubs(2), ksubs(2)
LOGICAL BLANK
INTEGER STATUS
PARAMETER (MPTS=5)
DIMENSION IPTS(3, MPTS)

STATUS= 0

Loop through the subsets.
IS = ISUBS(1)
IE = ISUBS(2)
JS = JSUBS(1)
JE = JSUBS(2)
KS = KSUBS(1)
KE = KSUBS(2)

Find the closest point(s) on any edge. Be careful of degenerate subsets.

NPTS = 0
D2MIN = 1.E35

CALL DMIN3(IS, IE, JS, JS, KS, KS, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
IF (JS .NE. JE) THEN
   CALL DMIN3(IS, IE, JE, JE, KS, KS, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
IF (KS .NE. KE) THEN
   CALL DMIN3(IS, IE, JS, JS, KE, KE, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
IF (JS .NE. JE) THEN
   CALL DMIN3(IS, IE, JE, JE, KE, KE, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
CALL DMIN3(IS, IS, JS+1, JE-1, KS, KS, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
IF (IS .NE. IE) THEN
   CALL DMIN3(IE, IE, JS+1, JE-1, KS, KS, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
IF (JS .NE. JE) THEN
   CALL DMIN3(IS, IS, JE, JE, KS+I, KE-I, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
Search from each of these points.

DO 10 II = 1, NPTS
    ISUB = II
    ISUB = 1
    I = IPTS(1, II)
    J = IPTS(2, II)
    K = IPTS(3, II)
    A = 0.
    B = 0.
    G = 0.
    IF (ISTAT .EQ. 0) GOTO 30
10 CONTINUE
C   Failed all searches.
C   STATUS = 1
C
30 CONTINUE
RETURN
END

*** End of Subroutine ESrch3

***********************************************************************************************
SUBROUTINE ESrch3(I,J,K,A,B,G,X,Y,Z,ISUB,STATUS)
***********************************************************************************************
C   Search from closest point on any face.
C
   STATUS=1 - Couldn’t find the point.
C
   common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
   & iblank(151,40,110), blank,isubs(2),
   & jsubs(2),ksubs(2)
   LOGICAL BLANK
   INTEGER STATUS
C
   PARAMETER (MPTS=5)
   DIMENSION IPTS(3,MPTS)
C
   STATUS= 0
C
   Loop through the subsets.
C
   IS= ISUBS(1)
   IE= ISUBS(2)
   JS= JSUBS(1)
   JE= JSUBS(2)
   KS= KSUBS(1)
   KE= KSUBS(2)
C
   Find the closest point(s) on any face. Be careful of degenerate
   subsets.
C
   NPTS = 0
   D2MIN= 1.E35
C
   CALL DMIN3(IS,IE,JS,JE,KS,
   & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
   IF (KS.NE.KE) THEN
      CALL DMIN3(IS,IE,JS,JE,KE,
      & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      CALL DMIN3(IS,IE,JS,JE,KS+1,KE-1,
      & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      IF (JS.NE.JE) THEN
         CALL DMIN3(IS,IE,JE,JS+1,KE-1,
         & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      ENDIF
      CALL DMIN3(IS,JS+1,JE-1,KS+1,KE-1,
      & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      IF (IS.NE.IE) THEN
         CALL DMIN3(IE,IE,JE,JS+1,KE-1,
         & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      ENDIF
   ENDIF
   CALL DMIN3(IS,JS+1,JE-1,KS+1,KE-1,
   & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
   IF (IS.NE.IE) THEN
      CALL DMIN3(IE,IE,JS+1,JE-1,KS+1,KE-1,
      & X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
   ENDIF
ENDIF
C
C   Search from each of these points.
**End of Subroutine FSrch3**

---


Search from closest point on a hole boundary.

- STATUS = 1 - Couldn't find the point.

```fortran
common /grdxyz/ xyz(151,40,110,3), idim, jdim, kdim,
&   iblank(151,40,110), blank, isubs(2),
&   jsubs(2), ksubs(2)
LOGICAL BLANK
INTEGER STATUS
PARAMETER (MPTS=5)
DIMENSION IPTS(3,MPTS)

STATUS = 0
Loop through the subsets.

if (blank) then
  IS = ISUBS(1)
  IE = ISUBS(2)
  JS = JSUBS(1)
  JE = JSUBS(2)
  KS = KSUBS(1)
  KE = KSUBS(2)

Search for the closest point(s) on the boundary of any hole.

NPTS = 0
D2MIN = 1.E35
DO 10 KK = KS, KE
  DO 10 JJ = JS, JE
    DO 10 II = IS, IE
      IF (IBLANK(II, JJ, KK).NE.0 .AND. (II.NE.1 .AND. IBLANK(II-1, JJ, KK).EQ.0) .OR. (II.NE.IDIM .AND. IBLANK(II+1, JJ, KK).EQ.0))
        (II.NE.1 .AND. IBLANK(II-1, JJ, KK).EQ.0) .OR. (II.NE.IDIM .AND. IBLANK(II+1, JJ, KK).EQ.0)
```

---
C .OR. (JJ.NE.1 .AND. IBLANK(II, JJ-1, KK).EQ.0)
C .OR. (JJ.NE.JDIM .AND. IBLANK(II, JJ+1, KK).EQ.0)
C .OR. (KK.NE.1 .AND. IBLANK(II, JJ, KK-1).EQ.0)
C .OR. (KK.NE.KDIM .AND. IBLANK(II, JJ, KK+1).EQ.0)) THEN
D2= (X-XYZ (II, JJ, KK, 1))**2
   +(Y-XYZ (II, JJ, KK, 2))**2
   +(Z-XYZ (II, JJ, KK, 3))**2
IF (D2.LT.D2MIN) THEN
   D2MIN = D2
   NPTS = 1
   IPTS(1,1)= II
   IPTS(2,1)= JJ
   IPTS(3,1)= KK
C If several points have the same minimum distance, keep track of them
C Use PSRCH3 to determine which is correct.
C
ELSE IF (D2.EQ.D2MIN) THEN
   IF (NPTS.LT.MPTS) THEN
      NPTS = NPTS+1
      IPTS(1,NPTS)= II
      IPTS(2,NPTS)= JJ
      IPTS(3,NPTS)= KK
   ENDIF
ENDIF
ENDIF
ENDIF
CONTINUE
C Check each of these points with PSRCH3.
C
DO 20 II= 1,NPTS
   ISUB = 1
   I = IPTS(1,II)
   J = IPTS(2,II)
   K = IPTS(3,II)
   A = 0.
   B = 0.
   G = 0.
   IF (ISTAT.EQ.0) GOTO 30
   CONTINUE
20 CONTINUE
C ENDIF
C Failed from all closest hole boundary points.
C STATUS= 1
C 30 CONTINUE
RETURN
END
*** End of Subroutine HSrch3

*****************************************************************************
SUBROUTINE INV3X3(A,AINV,STATUS)
*****************************************************************************
C Invert the 3x3 matrix A. If A is singular, do our best to find the
C pseudo-inverse.
C
STATUS=1 - A has one dependent column.
STATUS=2 - A has two dependent columns.
STATUS=3 - A is zero.
```c
DIMENSION A(3,3),AINV(3,3)
INTEGER STATUS

DIMENSION TMP(3,3),WORK(3),S(3),E(3),U(3,3),V(3,3),SIU(3,3)

STATUS= 0

AINV(1,1) = A(1,1)*A(2,2) + A(2,1)*A(3,2) - A(3,1)*A(2,2) - A(2,2)*A(3,3)
AINV(1,2) = A(1,2)*A(2,3) + A(2,2)*A(3,3) - A(3,2)*A(2,3) - A(2,3)*A(3,2)
AINV(1,3) = A(1,3)*A(2,2) + A(2,3)*A(3,2) - A(3,3)*A(2,2) - A(2,2)*A(3,3)

AINV(2,1) = A(1,1)*A(2,2) + A(2,1)*A(3,2) - A(3,1)*A(2,2) - A(2,2)*A(3,3)
AINV(2,2) = A(1,2)*A(2,3) + A(2,2)*A(3,3) - A(3,2)*A(2,3) - A(2,3)*A(3,2)
AINV(2,3) = A(1,3)*A(2,2) + A(2,3)*A(3,2) - A(3,3)*A(2,2) - A(2,2)*A(3,3)

AINV(3,1) = A(1,1)*A(2,2) + A(2,1)*A(3,2) - A(3,1)*A(2,2) - A(2,2)*A(3,3)
AINV(3,2) = A(1,2)*A(2,3) + A(2,2)*A(3,3) - A(3,2)*A(2,3) - A(2,3)*A(3,2)
AINV(3,3) = A(1,3)*A(2,2) + A(2,3)*A(3,2) - A(3,3)*A(2,2) - A(2,2)*A(3,3)

DET = A(1,1)*AINV(1,1) + A(2,1)*AINV(1,2) + A(3,1)*AINV(1,3)

Matrix is nonsingular. Finish up AINV.

IF (DET.NE.0.) THEN
  DET = 1./DET
  AINV(1,1) = AINV(1,1)*DET
  AINV(1,2) = AINV(1,2)*DET
  AINV(1,3) = AINV(1,3)*DET
  AINV(2,1) = AINV(2,1)*DET
  AINV(2,2) = AINV(2,2)*DET
  AINV(2,3) = AINV(2,3)*DET
  AINV(3,1) = AINV(3,1)*DET
  AINV(3,2) = AINV(3,2)*DET
  AINV(3,3) = AINV(3,3)*DET
ENDIF

Matrix is singular. Do a singular value decomposition to construct the pseudo-inverse. Use LINPACK routine SSVDC.

ELSE
  CALL COPY(9,A,TMP)
  CALL SSVDC(TMP,3,3,3,S,E,U,V,3,WORK,11,INFO)
  IF (S(1).EQ.0.) THEN
    STATUS= 3
    CALL ZERO(9,AINV)
    GOTO 10
  ENDIF

  Compute V S U.

  S(1) = 1./S(1)
  IF (S(3)*S(1).LT.1.E-5) THEN
    STATUS= 1
    S(3) = 1./S(3)
  ELSE
    S(3) = 1./S(3)
  ENDIF
  IF (S(2)*S(1).LT.1.E-5) THEN
    STATUS= 2
    S(2) = 1./S(2)
  ELSE
    S(2) = 1./S(2)
  ENDIF
ENDIF

Start out assuming S is a diagonal matrix.
```
C
SIU(1,1) = S(1)*U(1,1)
SIU(2,1) = S(2)*U(2,1)
SIU(3,1) = S(3)*U(3,1)
C
SIU(1,2) = S(1)*U(1,2)
SIU(2,2) = S(2)*U(2,2)
SIU(3,2) = S(3)*U(3,2)
C
SIU(1,3) = S(1)*U(1,3)
SIU(2,3) = S(2)*U(2,3)
SIU(3,3) = S(3)*U(3,3)
C
S is upper bidiagonal, with E as the super diagonal.
IF (INFO.GE.1) THEN
  SIU(1,1) = SIU(1,1) - (E(1)*S(1)*S(2))*U(2,1)
  SIU(1,2) = SIU(1,2) - (E(1)*S(1)*S(2))*U(2,2)
  SIU(1,3) = SIU(1,3) - (E(1)*S(1)*S(2))*U(2,3)
ENDIF
IF (INFO.GE.2) THEN
  SIU(1,1) = SIU(1,1) + (E(1)*E(2)*S(1)*S(2)**2*S(3))*U(3,1)
  SIU(1,2) = SIU(1,2) + (E(1)*E(2)*S(1)*S(2)**2*S(3))*U(3,2)
  SIU(1,3) = SIU(1,3) + (E(1)*E(2)*S(1)*S(2)**2*S(3))*U(3,3)
ENDIF
C
SIU(2,1) = SIU(2,1) - (E(2)*S(2)*S(3))*U(3,1)
SIU(2,2) = SIU(2,2) - (E(2)*S(2)*S(3))*U(3,2)
SIU(2,3) = SIU(2,3) - (E(2)*S(2)*S(3))*U(3,3)
C
FINISH UP A = V S U.
C
AINV(1,1)= V(1,1)*SIU(1,1) + V(1,2)*SIU(2,1) + V(1,3)*SIU(3,1)
AINV(2,1)= V(2,1)*SIU(1,1) + V(2,2)*SIU(2,1) + V(2,3)*SIU(3,1)
C
AINV(3,1)= V(3,1)*SIU(1,1) + V(3,2)*SIU(2,1) + V(3,3)*SIU(3,1)
C
AINV(1,2)= V(1,1)*SIU(1,2) + V(1,2)*SIU(2,2) + V(1,3)*SIU(3,2)
AINV(2,2)= V(2,1)*SIU(1,2) + V(2,2)*SIU(2,2) + V(2,3)*SIU(3,2)
C
AINV(3,2)= V(3,1)*SIU(1,2) + V(3,2)*SIU(2,2) + V(3,3)*SIU(3,2)
C
AINV(1,3)= V(1,1)*SIU(1,3) + V(1,2)*SIU(2,3) + V(1,3)*SIU(3,3)
AINV(2,3)= V(2,1)*SIU(1,3) + V(2,2)*SIU(2,3) + V(2,3)*SIU(3,3)
C
AINV(3,3)= V(3,1)*SIU(1,3) + V(3,2)*SIU(2,3) + V(3,3)*SIU(3,3)
ENDIF
C
10 CONTINUE
RETURN
*** End of Subroutine Inv3X3

************************************************************************************************************
************************************************************************************************************
C
Search from closest point coming from grid N.
C
STATUS=1 - Couldn't find the point.
C
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
&   iblank(151,40,110), blank,isubs(2),
&   jsubs(2),ksubs(2)
LOGICAL BLANK
INTEGER STATUS

PARAMETER (MPTS=5)
DIMENSION IPTS(3,MPTS)

STATUS= 0

Loop through the subsets.

IF (BLANK) THEN
  IS= ISUBS(1)
  IE= ISUBS(2)
  JS= JSUBS(1)
  JE= JSUBS(2)
  KE= KSUBS(1)

Search for the closest point(s) on the boundary of any hole.

NPTS = 0
D2MIN = 1.E35
DO 10 KK= KS,KE
  DO 10 JJ= JS,JE
    DO I0 II= IS, IE
      IF (IBLANK(II,JJ, KK).EQ.-N) THEN
        D2= (X-XYZ (II, JJ, KK, I) )**2
            + (Y-XYZ (II, JJ, KK, 2) )**2
            + (Z-XYZ (II, JJ, KK, 3) )**2
        IF (D2.LT.D2MIN) THEN
          D2MIN = D2
          NPTS = 1
          IPTS(1,1)= II
          IPTS(2,1)= JJ
          IPTS(3,1)= KK
        ELSE IF (D2.EQ.D2MIN) THEN
          IF (NPTS.LT.MPTS) THEN
            NPTS = NPTS+I
            IPTS(1,NPTS)= II
            IPTS(2,NPTS)= JJ
            IPTS(3,NPTS)= KK
          ENDIF
        ENDIF
      ENDIF
    CONTINUE
10

If several points have the same minimum distance, keep track of them. Use PSRCH3 to determine which is correct.

ELSE IF (D2.EQ.D2MIN) THEN
  IF (NPTS.LT.MPTS) THEN
    NPTS = NPTS+1
    IPTS(1,NPTS)= II
    IPTS(2,NPTS)= JJ
    IPTS(3,NPTS)= KK
  ENDIF
ENDIF
ENDIF

Check each of these points with PSRCH3.

DO 20 II= 1,NPTS
  ISUB = 1
  I = IPTS(1,II)
  J = IPTS(2,II)
  K = IPTS(3,II)
  A = 0.
  B = 0.
  G = 0.
  CALL PSRCH3(IS, IE, JS, JE, KS, KE,
  IF (ISTAT.EQ.0) GOTO 30
20 CONTINUE

ENDIF
C Failed from all closest hole boundary points.
C
STATUS = 1
C
30 CONTINUE
RETURN
END
*** End of Subroutine NSrch3


C Search for a point using CELL3.
C
C STATUS=1 - Couldn't find the point.
C
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
& iblank(151,40,110), blank,isubs(2),
& jsubs(2),ksubs(2)
LOGICAL BLANK
INTEGER STATUS
C
DIMENSION AMAT(3,3), BMAT(3,3)
C
STATUS= 0
C
CALL CELL3 (IS,IE,JS,JE, KS, KE,
IF (ISTAT.NE.0) THEN
  STATUS = 1
  GO TO 10
ENDIF
C
10 CONTINUE
RETURN
END
*** End of Subroutine PSrch3

SUBROUTINE COPY(LEN,FROM,TO)

C Copy an array from FROM to TO. The user is responsible for making
C sure that FROM doesn't overwrite itself if FROM and TO overlap.
C
DIMENSION FROM(*),TO(*)
C
DO 10 I= 1,LEN
  TO(I) = FROM(I)
10 CONTINUE
RETURN
END
*** End of Subroutine Copy

subroutine ssvdc(x,ldx,n,p,s,e,u,ldu,v,ldv,work,job,info)

************************************************************************
ssvdc is a subroutine to reduce a real nxp matrix x by orthogonal transformations u and v to diagonal form. the diagonal elements s(i) are the singular values of x. the columns of u are the corresponding left singular vectors, and the columns of v the right singular vectors.

on entry

x real(ldx,p), where ldx.ge.n.
x contains the matrix whose singular value decomposition is to be computed. x is destroyed by ssvdc.

ldx integer.
ldx is the leading dimension of the array x.

n integer.
n is the number of rows of the matrix x.

p integer.
p is the number of columns of the matrix x.

ldu integer.
ldu is the leading dimension of the array u. (see below).

ldv integer.
ldv is the leading dimension of the array v. (see below).

work real(n).
work is a scratch array.

job integer.
job controls the computation of the singular vectors. it has the decimal expansion ab with the following meaning

  a.eq.0 do not compute the left singular vectors.
  a.eq.1 return the n left singular vectors in u.
  a.ge.2 return the first min(n,p) singular vectors in u.
  b.eq.0 do not compute the right singular vectors.
  b.eq.1 return the right singular vectors in v.

on return

s real(mm), where mm=min(n+l,p).
the first min(n,p) entries of s contain the singular values of x arranged in descending order of magnitude.

e real(p).
e ordinarily contains zeros. however see the discussion of info for exceptions.
u is a real array of dimension (ldu,k), where ldu.ge.n. if joba.eq.1 then k.eq.n, if joba.ge.2 then k.eq.min(n,p).

u contains the matrix of left singular vectors. u is not referenced if joba.eq.0. if n.le.p or if joba.eq.2, then u may be identified with x in the subroutine call.

v is a real array of dimension (ldv,p), where ldv.ge.p.

v contains the matrix of right singular vectors. v is not referenced if job.eq.0. if p.le.n, then v may be identified with x in the subroutine call.

info is an integer.

the singular values (and their corresponding singular vectors) s(info+1),s(info+2),...,s(m) are correct (here m=min(n,p)). thus if info.eq.0, all the singular values and their vectors are correct. in any event, the matrix b = trans(u)*x*v is the bidiagonal matrix with the elements of s on its diagonal and the elements of e on its super-diagonal (trans(u) is the transpose of u). thus the singular values of x and b are the same.

linpack, this version dated 03/19/79.

correction to shift calculation made 2/85.

g.w. stewart, university of maryland, argonne national lab.

***** uses the following functions and subprograms.

external srot
blas saxpy,sdot,sscal,sswap,snrm2,srotg
fortran abs,amax1,max0,min0,mod,sqrt

internal variables

integer i,iter,j,jobu,k,kase,kx,l,ll,lls,lm1,lp1,ls,lu,m,maxit,
* mm,mml,mm1,nct,nctpl,ncu,nrt,nrtpl
real sdot,t,r
real b,c,cs,el,emml,f,g,snrm2,snrm1,snrm3,shift,sl,sm,sn,smml,t1,test,
* ztest
logical wantu,wantv

set the maximum number of iterations.

maxit = 30

determine what is to be computed.

wantu = .false.
wantv = .false.
jobu = mod(job,100)/10
ncu = n
if (jobu.gt.1) ncu = min0(n,p)
if (jobu.ne.0) wantu = .true.
if (mod(job,10).ne.0) wantv = .true.

reduce x to bidiagonal form, storing the diagonal elements in s and the super-diagonal elements in e.

info = 0
nct = min0(n-1,p)
nrt = max0(0,min0(p-2,n))
lu = max0(nct,nrt)
if (lu .lt. 1) go to 170
do 160 l = 1, lu
   lp1 = l + 1
   if (l .gt. nct) go to 20
   compute the transformation for the l-th column and place the l-th diagonal in s(l).
   s(l) = snrm2(n-l+1,x(l,l),1)
   if (s(l) .eq. 0.0e0) go to 10
   if (x(l,l) .ne. 0.0e0) s(l) = sign(s(l),x(l,l))
call sscal(n-l+1,1.0e0/s(l),x(l,l),1)
x(l,l) = 1.0e0 + x(l,l)
   10 continue
   s(l) = -s(l)
   continue
   if (p .lt. lp1) go to 50
   do 40 j = lp1, p
      if (l .gt. nct) go to 30
      if (s(l) .eq. 0.0e0) go to 30
      apply the transformation.
      t = -sdot(n-l+1,x(l,l),1,x(l,j),1)/x(l,l)
call saxpy(n-l+1,t,x(l,l),1,x(l,j),1)
   continue
   place the l-th row of x into e for the subsequent calculation of the row transformation.
e(j) = x(l,j)
   40 continue
   continue
   if (.not.wantu .or. 1 .gt. nct) go to 70
   place the transformation in u for subsequent back multiplication.
do 60 i = 1, n
   u(i,l) = x(i,l)
   continue
   continue
   if (l .gt. nrt) go to 150
   compute the l-th row transformation and place the l-th super-diagonal in e(l).
e(l) = snrm2(p-l,e(lp1),1)
   if (e(l) .eq. 0.0e0) go to 80
   if (e(lp1) .ne. 0.0e0) e(l) = sign(e(l),e(lp1))
call sscal(p-1,1.0e0/e(l),e(lp1),1)
e(lp1) = 1.0e0 + e(lp1)
   80 continue
   e(l) = -e(l)
   if (lp1 .gt. n .or. e(l) .eq. 0.0e0) go to 120
   apply the transformation.
do 90 i = lp1, n
   work(i) = 0.0e0
   continue
   do 100 j = lp1, p
      call saxpy(n-l,e(j),x(lp1,j),1,work(lp1),1)
   continue
do 110 j = lpl, p
   call saxpy(n-l,e(j)/e(lpl),work(lpl),l,x(lpl,j),l)
110 continue
continue
   if (.not.wantv) go to 140

place the transformation in v for subsequent
back multiplication.

do 130 i = lpl, p
   v(i,1) = e(i)
130 continue
140 continue
150 continue
160 continue
170 continue

set up the final bidiagonal matrix or order m.

m = min0(p,n+1)
nctpl = nct + 1
nrtp1 = nrt + 1
if (nct .lt. p) s(nctpl) = x(nctpl,nctpl)
if (n .lt. m) s(m) = 0.0e0
if (nrtp1 .lt. m) e(nrtp1) = x(nrtp1,m)
e(m) = 0.0e0

if required, generate u.
if (.not.wantu) go to 300
   if (ncu .lt. nctpl) go to 200
      do 190 j = nctpl, ncu
         do 180 i = 1, n
            u(i,j) = 0.0e0
         180 continue
         u(j,j) = 1.0e0
      190 continue
      if (nct .lt. 1) go to 290
      do 280 ll = 1, nct
         if (s(ll) .eq. 0.0e0) go to 250
            lpl = ll + 1
            if (ncu .lt. lpl) go to 220
            do 210 j = lpl, ncu
               t = -sdot(n-l+1,u(1,l),1,u(1,j),1)/u(1,1)
               call saxpy(n-l+1,t,u(1,l),1,u(1,j),1)
            210 continue
            call sscl(n-l+1,-1.0e0,u(1,l),1)
            u(1,l) = 1.0e0 + u(1,l)
            lml = 1 - 1
            if (lml .lt. 1) go to 240
            do 230 i = 1, lml
               u(i,1) = 0.0e0
            230 continue
         240 continue
         go to 270
      250 continue
      do 260 i = 1, n
         u(i,1) = 0.0e0
      260 continue
      u(1,1) = 1.0e0
   270 continue
280 continue
290 continue
300 continue

if it is required, generate v.

if (.not.wantv) go to 350
  do 340 l1 = 1, p
      l = p - l1 + 1
      lpl = l + 1
      if (l.gt.nrt) go to 320
      if (e(l).eq.0.0e0) go to 320
      do 310 j = lpl, p
        t = -sdot(p-l,v(lpl,1),1,v(lpl,j),1)/v(lpl,l)
        call saxpy(p-l,t,v(lpl,1),1,v(lpl,j),1)
      310 continue
  320 continue
  do 330 i = 1, p
      v(i,1) = 0.0e0
  330 continue
  v(l,l) = 1.0e0
  340 continue
  350 continue

main iteration loop for the singular values.

mm = m
iter = 0
360 continue

quit if all the singular values have been found.

...exit
  if (m.eq.0) go to 620
  if too many iterations have been performed, set
  flag and return.
  if (iter.lt.maxit) go to 370
  info = m
  ......exit
  go to 620
  370 continue

this section of the program inspects for
negligible elements in the s and e arrays.  on
completion the variables kase and l are set as follows.

kase = 1  if s(m) and e(l-1) are negligible and 1.lt.m
kase = 2  if s(l) is negligible and 1 lt.m
kase = 3  if e(l-1) is negligible, 1 lt.m, and
          s(l), ..., s(m) are not negligible (qr step).
kase = 4  if e(m-1) is negligible (convergence).

do 390 l1 = 1, m
     l = m - l1
  390 continue

...exit
  if (l.eq.0) go to 400
  test = abs(s(l)) + abs(s(l+1))
  ztest = test + abs(e(l))
  if (ztest.ne.test) go to 380
    e(l) = 0.0e0
  380 continue
  390 continue
  400 continue
  if (l.ne.m - 1) go to 410
kase = 4
410  go to 480
     continue
     lp1 = l + 1
     mpl = m + 1
     do 430 11s = lp1, mpl
             ls = m - 11s + lp1
     c          ... exit
             if (ls .eq. l) go to 440
             test = 0.0e0
             if (ls .ne. m) test = test + abs(e(ls))
             if (ls .ne. l + 1) test = test + abs(e(ls-1))
             ztest = test + abs(s(ls))
             if (ztest .ne. test) go to 420
             s(ls) = 0.0e0
     c          ...... exit
             go to 440
     420  continue
     430  continue
     440  continue
             if (ls .ne. l) go to 450
             kase = 3
             go to 470
     450  continue
             if (ls .ne. m) go to 460
             kase = 1
             go to 470
     460  continue
             kase = 2
             l = ls
     470  continue
     480  continue
             l = l + 1
     c          perform the task indicated by kase.
     c          go to (490,520,540,570), kase
     c          deflate negligible s(m).
     c          490  continue
             mm1 = m - 1
             f = e(m-1)
             e(m-1) = 0.0e0
             do 510 kk = 1, mm1
                     k = mm1 - kk + 1
                     t1 = s(k)
                     call srotg(t1,f,cs,sn)
                     s(k) = t1
                     if (k .eq. l) go to 500
                     f = -sn*e(k-1)
                     e(k-1) = cs*e(k-1)
     500  continue
             if (wantv) call srot(p,v(l,k),l,v(l,m),l,cs,sn)
             continue
     510  continue
     520  continue
             f = e(l-1)
             e(l-1) = 0.0e0
             do 530 k = 1, m
                     t1 = s(k)
                     call srotg(t1,f,cs,sn)
                     s(k) = t1
     c          split at negligible s(1).
$f = -\text{sne}(k)$
$e(k) = \text{cse}(k)$

if (wantu) call srot(n,u(1,k),1,u(1,k-1),1,cs,sn)

continue
go to 610

perform one qr step.

continue

calculate the shift.

scale = amax1(abs(s(m)),abs(s(m-1)),abs(e(m-1)),abs(s(1)),
abs(e(l)))

sm = s(m)/scale
smml = s(m-1)/scale
eeml = e(m-1)/scale
sl = s(l)/scale
el = e(l)/scale

b = ((smml + sm)*(smml - sm) + eeml**2)/2.0e0

c = (sm*emml)**2

shift = 0.0e0

if (b .eq. 0.0e0 .and. c .eq. 0.0e0) go to 550

shift = sqrt(b**2 + c)

if (b .lt. 0.0e0) shift = -shift

shift = c/(b + shift)

continue

f = (sl + sm)*(sl - sm) + shift

g = sl*el

chase zeros.

mm1 = m - 1

do 560 k = 1, mm1

call srot(f,g,cs,sn)

if (k .ne. 1) e(k-1) = f

f = cs*sn + s(e(k))

if (wantv) call srot(p,v(1,k),1,v(1,k+1),1,cs,sn)

call srotg(f,g,cs,sn)

s(k) = f

if (wantu .and. k .lt. n)

call srot(n,u(1,k),1,u(1,k+1),1,cs,sn)

continue

e(m-1) = f

iter = iter + 1

go to 610

convergence.

make the singular value positive.

if (s(1) .ge. 0.0e0) go to 580

s(1) = -s(1)

if (wantv) call sscal(p,-1.0e0,v(1,1),1)

continue

order the singular value.
if (l .eq. mm) go to 600
...exit
if (s(l) .ge. s(l+1)) go to 600
t = s(l)
s(l) = s(l+1)
s(l+1) = t
if (wantv .and. 1 .lt. p) * call sswap(p, v(1,1), 1, v(1,1+l), 1)
if (wantu .and. 1 .lt. n) * call sswap(n, u(1,1), 1, u(1,1+l), 1)
l = l + 1
go to 590

600 continue
iter = 0
m = m - 1
610 continue
go to 360
620 continue
return
end
*** End of Subroutine SSVDC

******************************************************************************
subroutine swap (n,sx,incx,sy,incy)
******************************************************************************
interchanges two vectors.
uses unrolled loops for increments equal to 1.
jack dongarra, linpack, 3/11/78.
real sx(1), sy(1), stemp
integer i, incx, incy, ix, iy, m, mpl, n
if (n .le. 0) return
if (incx .eq. 1 .and. incy .eq. 1) go to 20
code for unequal increments or equal increments not equal to 1
c
ix = 1
iy = 1
if (incx .lt. 0) ix = (-n+1)*incx + 1
if (incy .lt. 0) iy = (-n+1)*incy + 1
do 10 i = 1, n
stemp = sx(ix)
sx(ix) = sy(iy)
sy(iy) = stemp
ix = ix + incx
iy = iy + incy
10 continue
return
c code for both increments equal to 1
c
c clean-up loop
c
20 m = mod(n, 3)
if (m .eq. 0 ) go to 40
do 30 i = 1, m
stemp = sx(i)
sx(i) = sy(i)
sy(i) = stemp
30 continue
   if( n .lt. 3 ) return
40 mpl = m + 1
do 50 i = mpl,n,3
   stemp = sx(i)
sx(i) = sy(i)
sy(i) = stemp
   stemp = sx(i + 1)
sx(i + 1) = sy(i + 1)
sy(i + 1) = stemp
   stemp = sx(i + 2)
sx(i + 2) = sy(i + 2)
sy(i + 2) = stemp
50 continue
return
end

*** End of Subroutine SSwap

******************************************************************************
******************************************************************************
subroutine saxpy(n,sa,sx,incy,sy,incy)
******************************************************************************
******************************************************************************
c
   constant times a vector plus a vector.
c uses unrolled loop for increments equal to one.
c jack dongarra, linpack, 3/11/78.
c
real sx(1),sy(1),sa
integer i,incx,incy,ix,iy,m,tpl,n
c
if(n.le.0) return
   if (sa .eq. 0.0) return
   if(incx.eq.1.and.incy.eq.1)go to 20
   c
c   c code for unequal increments or equal increments
   c not equal to 1
   c
   ix = 1
   iy = 1
   if(invx.lt.0) ix = (-n+1)*incx + 1
   if(incy.lt.0) iy = (-n+1)*incy + 1
   do 10 i = 1,n
      sy(iy) = sy(iy) + sa*sx(ix)
      ix = ix + incx
      iy = iy + incy
10 continue
return
   c
c   code for both increments equal to 1
   c
   clean-up loop
   c
20 m = mod(n,4)
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
      sy(i) = sy(i) + sa*sx(i)
30 continue
   if( n .lt. 4 ) return
40 mpl = m + 1
do 50 i = mpl,n,4
   sy(i) = sy(i) + sa*sx(i)
sy(i + 1) = sy(i + 1) + sa*sx(i + 1)
50 continue
return
sy(i + 2) = sy(i + 2) + sa*sq(i + 2)
sy(i + 3) = sy(i + 3) + sa*sq(i + 3)

50 continue
return
end

*** End of Subroutine SaxPy

***********************************************************************
real function sdot(n,sx,incy,ty,incy)
***********************************************************************
c
forms the dot product of two vectors.
uses unrolled loops for increments equal to one.
jack dongarra, linpack, 3/11/78.
c
real sx(1), sy(1), stemp
integer i, incx, incy, ix, iy, m, mpl, n
c
stemp = 0.0e0
sdot = 0.0e0
if(n.le.0) return
if(incx.eq.1.and.incy.eq.1) go to 20
c
code for unequal increments or equal increments
not equal to 1
c
ix = 1
iy = 1
if(incx.lt.0) ix = (-n+1)*incx + 1
if(incy.lt.0) iy = (-n+1)*incy + 1
do 10 i = 1, n
stemp = stemp + sx(ix)*sy(iy)
ix = ix + incx
iy = iy + incy
10 continue
sdot = stemp
return
c
code for both increments equal to 1
c
0 clean-up loop
c
20 m = mod(n,5)
if( m.eq.0 ) go to 40
do 30 i = 1, m
stemp = stemp + sx(i)*sy(i)
30 continue
if( n.lt.5 ) go to 60
40 mpl = m + 1
do 50 i = mpl, n, 5
stemp = stemp + sx(i)*sy(i) + sx(i + 1)*sy(i + 1) +
* sq(i + 2)*sy(i + 2) + sq(i + 3)*sy(i + 3) + sq(i + 4)*sy(i + 4)
50 continue
60 sdot = stemp
return
end
*** End of Function Sdot
real function snrm2 ( n, sx, incx)
************************************************************************
group next
integer next
real sx(1), cutlo, cuthi, hitest, sum, xmax, zero, one
data zero, one /0.0e0, 1.0e0/
c
euclidean norm of the n-vector stored in sx() with storage
increment incx .
if n .le. 0 return with result = 0.
if n .ge. 1 then incx must be .ge. 1

c.l.lawson, 1978 jan 08

c four phase method using two built-in constants that are
hopefully applicable to all machines.
cutlo = maximum of sqrt(u/eps) over all known machines.
cuthi = minimum of sqrt(v) over all known machines.
where
eps = smallest no. such that eps + 1. .gt. 1.
u = smallest positive no. (underflow limit)
v = largest no. (overflow limit)

to outline of algorithm..

brief outline of algorithm..

phase 1 scans zero components.
move to phase 2 when a component is nonzero and .le. cutlo
move to phase 3 when a component is .gt. cutlo
move to phase 4 when a component is .ge. cuthi/m
where m = n for x() real and m = 2*n for complex.

values for cutlo and cuthi..
from the environmental parameters listed in the imsl converter
document the limiting values are as follows..
cutlo, s.p. u/eps = 2**(102) for honeywell. close seconds are
univac and dec at 2**(-103)
thus cutlo = 2**(-51) = 4.44089e-16
cuthi, s.p. v = 2**127 for univac, honeywell, and dec.
thus cuthi = 2**(63.5) = 1.30438e19
cutlo, d.p. u/eps = 2**(-67) for honeywell and dec.
thus cutlo = 2**(-33.5) = 8.23181d-11
cuthi, d.p. same as s.p. cuthi = 1.30438d19
data cutlo, cuthi / 8.232d-11, 1.304d19 /
data cutlo, cuthi / 4.441e-16, 1.304e19 /
data cutlo, cuthi / 4.441e-16, 1.304e19 /

c
if(n .gt. 0) go to 10
snrm2 = zero
 go to 300
c
10 assign 30 to next
sum = zero
nn = n * incx
begin main loop
c
i = 1
20 go to next,(30, 50, 70, 110)
30 if( abs(sx(i)) .gt. cutlo) go to 85
assign 50 to next
xmax = zero

c
phase 1. sum is zero

c
50 if( sx(i) .eq. zero) go to 200
if( abs(sx(i)) .gt. cutlo) go to 85

c
assign 70 to next
prepare for phase 2.
go to 105

prepare for phase 4.

100 i = j
assign 110 to next
sum = (sum / sx(i)) / sx(i)
105 xmax = abs(sx(i))
go to 115

phase 2. sum is small.
scale to avoid destructive underflow.

70 if( abs(sx(i)) .gt. cutlo ) go to 75

common code for phases 2 and 4.
in phase 4 sum is large. scale to avoid overflow.

110 if( abs(sx(i)) .le. xmax ) go to 115
    sum = one + sum * (xmax / sx(i))**2
    xmax = abs(sx(i))
go to 200

115 sum = sum + (sx(i)/xmax)**2
go to 200

prepare for phase 3.

75 sum = (sum * xmax) * xmax

for real or d.p. set hitest = cuthi/n
for complex set hitest = cuthi/(2*n)

85 hitest = cuthi/float( n )

phase 3. sum is mid-range. no scaling.

do 95 j=i,nn,incx
    if(abs(sx(j)) .ge. hitest) go to 100
95 sum = sum + sx(j)**2
    snrm2 = sqrt( sum )
go to 300

200 continue
    i = i + incx
    if ( i .le. nn ) go to 20

end of main loop.
compute square root and adjust for scaling.

snrm2 = xmax * sqrt(sum)
300 continue
return
end

*** End of Function SNRM2

************************************************************************************

subroutine srot (n,sx,incx, sy,incy,c,3)
************************************************************************************
applies a plane rotation.
jack dongarra, linpack, 3/11/78.

real sx(1),sy(1),stemp,c,s
integer i,incx,incy,ix,iy,n

if(n.le.0) return
if(incx.eq.1.and.incy.eq.1) go to 20

c ode for unequal increments or equal increments not equal
to 1

ix = 1
iy = 1
if(incx.lt.0)ix = (-n+1)*incx + 1
if(incy.lt.0)iy = (-n+1)*incy + 1
do 10 i = 1,n
  stemp = c*sx(ix) + s*sy(iy)
  sy(iy) = c*sy(iy) - s*sx(ix)
  sx(ix) = stemp
  ix = ix + incx
  iy = iy + incy
10 continue
return

c ode for both increments equal to 1

20 do 30 i = 1,n
  stemp = c*sx(i) + s*sy(i)
  sy(i) = c*sy(i) - s*sx(i)
  sx(i) = stemp
30 continue
return
end

*** End of Subroutine SRot

*****************************************************************************
subroutine srotg(sa,sb,c,s)
*****************************************************************************

construct givens plane rotation.
jack dongarra, linpack, 3/11/78.

real sa,sb,c,s,roe,scale,r,z

roe = sb
if( abs(sa) .gt. abs(sb) ) roe = sa
scale = abs(sa) + abs(sb)
if( scale .ne. 0.0 ) go to 10
  c = 1.0
  s = 0.0
  r = 0.0
  go to 20
10 r = scale*sqrt((sa/scale)**2 + (sb/scale)**2)
r = sign(1.0,roe)*r
c = sa/r
s = sb/r
20 z = 1.0
if( abs(sa) .gt. abs(sb) ) z = s
if( abs(sb) .ge. abs(sa) .and. c .ne. 0.0 ) z = 1.0/c
sa = r
sb = z
return
*** End of Subroutine SRotG

************************************************************
subroutine sscal(n,sa,sx,incx)
************************************************************
c scales a vector by a constant.
uses unrolled loops for increment equal to 1.
jack dongarra, linpack, 3/11/78.
c
real sa,sx(1)
integer i,incx,m,mpl,n,nincx
c
if (n.le.0) return
if (inxc.eq.1) go to 20
c code for increment not equal to 1
c
nincx = n*inxc
do 10 i = 1,nincx,inxc
   sx(i) = sa*sx(i)
10 continue
c return
c code for increment equal to 1
c
 clean-up loop
c
20 m = mod(n,5)
   if (m.eq.0) go to 40
do 30 i = 1,m
      sx(i) = sa*sx(i)
30 continue
   if (n.lt.5) return
40 mpl = m + 1
do 50 i = mpl,n,5
   sx(i) = sa*sx(i)
sx(i + 1) = sa*sx(i + 1)
sx(i + 2) = sa*sx(i + 2)
sx(i + 3) = sa*sx(i + 3)
sx(i + 4) = sa*sx(i + 4)
50 continue
return
end
*** End of Subroutine SScal

************************************************************************
SUBROUTINE ZERO(LEN,ARRAY)
************************************************************************
C Just a little routine to zero the array.
C DIMENSION ARRAY(*)
C
DO 10 I= 1,LEN
   ARRAY(I) = 0.
10 CONTINUE
RETURN
*** End of Subroutine Zero

SUBROUTINE TRIM(STRING, LSTRIN)

CHARACTER*(*) STRING
CHARACTER NULL, TAB

Initialize the null and tab characters.

NULL = CHAR(0)
TAB = CHAR(9)

Loop backwards through the character string and find the last nonblank, nonnull character.

LSTRIN = LEN(STRING)
DO 10 L = LSTRIN, 1, -1
   IF (STRING(L:L) .NE. ' ' .AND. STRING(L:L) .NE. NULL .AND. STRING(L:L) .NE. TAB) THEN
      LSTRIN = L
      GOTO 20
   ENDIF
10 CONTINUE

ALL blank or null or tabs!
LSTRIN = 0

20 CONTINUE
RETURN
END

*** End of Subroutine Trim

SUBROUTINE UCASE(STRING)

Character*(*) STRING
CHARACTER*26 LOWER, UPPER

DATA LOWER/'abcdefghijklmnopqrstuvwxyz'/
C UPPER/'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/

Don’t worry about the trailing blanks -
CALL TRIM(STRING, LSTRIN)

Look for lower case letters and substitute upper case ones.

DO 10 I = 1, LSTRIN
   LETTER = INDEX(LOWER, STRING(I:I))
IF (LETTER .NE. 0) STRING(I:I) = UPPER(LETTER:LETTER)
   10 CONTINUE
   RETURN
END

*** End of Subroutine UpCase