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, $\alpha = 30\text{ deg.}$ ........................................ 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, $\alpha = 40\text{ deg.}$ ........................................ 31
12. Orbiter windward surface thermocouple locations ........... 32
13. Windward centerline heating distribution, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 33
14. Circumferential heating distribution at $x/L = .1$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 34
15. Circumferential heating distribution at $x/L = .2$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 34
16. Circumferential heating distribution at $x/L = .3$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 35
17. Circumferential heating distribution at $x/L = .4$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 35
18. Circumferential heating distribution at $x/L = .5$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 36
19. Circumferential heating distribution at $x/L = .6$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 36
20. Circumferential heating distribution at $x/L = .7$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 37
21. Circumferential heating distribution at $x/L = .9$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 37
22. Heating distribution on windward surface at $y = 184.8\text{ in.}$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 38
23. Heating distribution on windward surface at $y = 233.6\text{ in.}$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\text{ deg.}$ ......................... 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 $\frac{x}{L} = .1$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 42
Circumferential heating distribution at $\frac{x}{L} = .2$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 42
Circumferential heating distribution at $\frac{x}{L} = .3$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 43
Circumferential heating distribution at $\frac{x}{L} = .4$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 43
Circumferential heating distribution at $\frac{x}{L} = .5$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 44
Circumferential heating distribution at $\frac{x}{L} = .6$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 44
Circumferential heating distribution at $\frac{x}{L} = .7$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg. ............ 45
Circumferential heating distribution at $\frac{x}{L} = .9$, 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

\( c_1, \ldots, c_5 \) Defined by Equations (108-112)

\( e_r, e_\xi, e_\eta \) Unit vector in streamline coordinates

\( F \) Function describing body surface

\( \nabla F \) Surface norm vector

\( h \) Metric coefficient

\( H^* \) Eckert reference enthalpy

\( H_e \) Edge enthalpy

\( H_w \) Wall enthalpy

\( H_{aw} \) Adiabatic wall enthalpy

\( \vec{i}, \vec{j}, \vec{k} \) Unit Cartesian vector

\( m \) Defined by Equation 107

\( N \) Exponent of the power law velocity profile relation

\( p \) Pressure

\( Pr_w \) Wall Prandtl number

\( Pr_{stag} \) Stagnation pressure

\( \dot{q}_L \) Laminar heating rate

\( \dot{q}_T \) Turbulent heating rate

\( q_w \) Surface heating rate

\( Re_\theta \) Momentum thickness Reynolds number

\( \vec{r} \) Position vector

\( s \) Distance along a streamline, measured from stagnation point

\( t \) Time

\( u, v, w \) Cartesian velocity components

\( u_e \) Edge velocity

\( V \) Total velocity

\( x, y, z \) Cartesian coordinates

\( \alpha, \beta \) Coordinates of computational domain

\( \theta \) Momentum thickness

\( \delta \) Boundary layer thickness

\( \mu^* \) Viscosity defined at reference enthalpy

\( \mu_e \) Edge viscosity

\( \rho^* \) Density defined by reference enthalpy

\( \rho_e \) Edge density

\( \tau, \xi, \eta \) Streamline coordinates (see Fig. 1)

\( \Psi_j \) 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

\[ \frac{dx}{dt} = u \]
\[
\frac{dy}{dt} = v \\
\frac{dz}{dt} = w
\]

Starting at an 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

\[
\vec{r} = x\hat{i} + y\hat{j} + z\hat{k}
\]

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

\[
\vec{r} = \xi \vec{e}_\xi + \eta \vec{e}_\eta
\]

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

\[
\vec{e}_\eta = \frac{\frac{\partial \vec{r}}{\partial \eta}}{\left| \frac{\partial \vec{r}}{\partial \eta} \right|} = \frac{\frac{\partial \xi}{\partial \eta} \hat{i} + \frac{\partial \eta}{\partial \eta} \hat{j} + \frac{\partial \tau}{\partial \eta} \hat{k}}{\left| \frac{\partial \vec{r}}{\partial \eta} \right|}
\]

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

\[
\vec{e}_\eta = \vec{e}_\tau \times \vec{e}_\xi
\]

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

\[
\vec{V} = V \vec{e}_\xi
\]

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

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

(9)

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

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

(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

\[
\tilde{e}_\eta = \frac{\nabla F}{|\nabla F|}
\]

From Equations (6) and (14) we have

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

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

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

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

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

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

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

For a given \( h \), initial values of \( \frac{\partial z}{\partial \eta} \) and \( \frac{\partial y}{\partial \eta} \) can be obtained from equations (17) and (18). Equations (20) and (21) can then be integrated to obtain \( \frac{\partial z}{\partial \eta} \) and \( \frac{\partial y}{\partial \eta} \) 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 \( \frac{\partial v}{\partial y} \), \( \frac{\partial w}{\partial z} \), \( \frac{\partial z}{\partial y} \), and \( \frac{\partial w}{\partial x} \) at each integration step. To compute \( |\nabla F| \), we need the values of \( \frac{\partial z}{\partial y} \) and \( \frac{\partial z}{\partial x} \) 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 $\Omega_x$ 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

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

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

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

Here $(x_j, y_j, z_j)$ are the $x, y, z$ coordinates of a local nodal point $j$ in element $\Omega_x$, $n$ 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:

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

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

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

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

Element II:

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

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

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

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

Element III:

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

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

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

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

\[
\begin{align*}
\hat{\psi}_4 &= -\alpha(1 - \beta) \\
\hat{\psi}_5 &= (1 + \alpha)(1 - \beta) \\
\hat{\psi}_6 &= (1 + \alpha)\beta \\
\hat{\psi}_7 &= -\alpha\beta \\
\end{align*}
\]

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

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

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

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

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

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

\[
\begin{align*}
\left.\frac{\partial x}{\partial \alpha}\right|_5 &= x_5 - x_4 \\
\left.\frac{\partial x}{\partial \beta}\right|_5 &= -x_2 + x_5
\end{align*}
\]

The $z$ coordinate of a point in the physical domain, which is the mapping of sub-element $II$, can be written as

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

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

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

\[
\begin{align*}
\frac{\partial x}{\partial \alpha} &= x_2\beta - x_3\beta + x_6(1 + \beta) - x_5(1 + \beta) \\
\frac{\partial x}{\partial \beta} &= -x_2(1 - \alpha) - x_3\alpha + x_6\alpha + x_5(1 - \alpha)
\end{align*}
\]
At point 5 of element II \((\alpha = 0, \beta = 0)\) in the physical domain, the derivatives are

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

\[
\frac{\partial x}{\partial \alpha}\bigg|_5 = x_6(1 - \beta) + x_5(1 - \beta) - x_8\beta + x_9\alpha\beta + x_8(1 - \alpha)\beta
\] (50)

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

\[
x = x_5 \hat{x}_5 + x_6 \hat{x}_6 + x_9 \hat{x}_9 + x_8 \hat{x}_8
\] = \(x_5(1 - \alpha)(1 - \beta) + x_6 \alpha(1 - \beta) + x_9 \alpha \beta + x_8(1 - \alpha)\beta (51)

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

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

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

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

\[
\frac{\partial x}{\partial \alpha}\bigg|_5 = -x_5 + x_6
\] (54)

\[
\frac{\partial x}{\partial \beta}\bigg|_5 = -x_5 + x_8
\] (55)

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

\[
x = x_4 \hat{x}_4 + x_5 \hat{x}_5 + x_8 \hat{x}_8 + x_7 \hat{x}_7
\] = \(-x_4\alpha(1 - \beta) + x_5(1 + \alpha)(1 - \beta) + x_8(1 + \alpha)\beta - x_7\alpha\beta (56)

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

\[
\frac{\partial x}{\partial \alpha} = -x_4(1 - \beta) + x_5(1 - \beta) + x_8\beta - x_7\beta
\] (57)

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

\[
\begin{align*}
\frac{\partial x}{\partial \alpha}|_5 &= -x_4 + x_5 \\
\frac{\partial x}{\partial \beta}|_5 &= -x_5 + x_8
\end{align*}
\]

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

\[
\begin{align*}
\frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{4}(x_5 - x_4 + x_6 - x_5 + x_6 - x_4 + x_5) \\
&= \frac{1}{2}(x_6 - x_4) \\
\frac{\partial x}{\partial \beta}|_5 &= \frac{1}{4}(-x_2 + x_5 - x_2 + x_5 - x_8 + x_8 - x_5 + x_8) \\
&= \frac{1}{2}(x_8 - x_2)
\end{align*}
\]

Similarly, we have

\[
\begin{align*}
\frac{\partial y}{\partial \alpha} &= \frac{1}{2}(y_6 - y_4) \\
\frac{\partial y}{\partial \beta} &= \frac{1}{2}(y_8 - y_2) \\
\frac{\partial z}{\partial \alpha} &= \frac{1}{2}(z_6 - z_4) \\
\frac{\partial z}{\partial \beta} &= \frac{1}{2}(z_8 - z_2)
\end{align*}
\]

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

\[
\begin{align*}
\frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{2}(x_6 - x_5 + x_6 - x_5) \\
&= x_6 - x_5 \\
\frac{\partial x}{\partial \beta}|_5 &= \frac{1}{2}(-x_2 + x_5 - x_5 + x_8) \\
&= \frac{1}{2}(x_8 - x_2)
\end{align*}
\]
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

\[
\frac{\partial x}{\partial \alpha}|_5 = \frac{1}{2}(x_5 - x_4 - x_4 + x_5) \\
= x_5 - x_4
\]  
(69)

\[
\frac{\partial x}{\partial \beta}|_5 = \frac{1}{2}(-x_2 + x_5 - x_5 + x_6) \\
= \frac{1}{2}(x_8 - x_2)
\]  
(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

\[
\frac{\partial x}{\partial \alpha}|_5 = \frac{1}{2}(x_5 - x_4 + x_6 - x_5) \\
= \frac{1}{2}(x_6 - x_4)
\]  
(71)

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

\[
\frac{\partial x}{\partial \alpha}|_5 = \frac{1}{2}(-x_5 + x_6 - x_4 + x_5) \\
= \frac{1}{2}(x_6 - x_4)
\]  
(73)

\[
\frac{\partial x}{\partial \beta}|_5 = \frac{1}{2}(-x_5 + x_8 - x_5 + x_8) \\
= x_8 - x_5
\]  
(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 $\frac{\partial u}{\partial \alpha}, \frac{\partial u}{\partial \beta}, \frac{\partial v}{\partial \alpha}, \frac{\partial v}{\partial \beta}, \frac{\partial w}{\partial \alpha}$ 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)
\[ dy = \frac{\partial y}{\partial \alpha} d\alpha + \frac{\partial y}{\partial \beta} d\beta \quad (76) \]

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

or

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

or

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

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

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

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

or

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

from Equation (79)

\[
\begin{bmatrix}
  d\alpha \\
  d\beta 
\end{bmatrix} = \frac{1}{|J|} \begin{bmatrix}
  (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 & -(\frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \beta}) \\
  -(\frac{\partial x}{\partial \alpha} \frac{\partial x}{\partial \gamma} + \frac{\partial y}{\partial \alpha} \frac{\partial y}{\partial \gamma} + \frac{\partial z}{\partial \alpha} \frac{\partial z}{\partial \gamma}) & (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 
\end{bmatrix}
\begin{bmatrix}
  dx \\
  dy \\
  dz 
\end{bmatrix} \quad (83)
\]
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.

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

(84)

From equations (82) and (83) we have

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

(85)

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

(86)

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

(87)

Now, the derivatives $\frac{\partial \alpha}{\partial y}, \frac{\partial \alpha}{\partial z}, \frac{\partial \beta}{\partial y}, \frac{\partial \beta}{\partial z}, \frac{\partial \gamma}{\partial y}, \frac{\partial \gamma}{\partial z}$ and $\frac{\partial \delta}{\partial x}$ 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)

$$
\frac{\partial \beta}{\partial y} = \frac{\partial \beta}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial \beta}{\partial \beta} \frac{\partial \beta}{\partial y} 
$$

(93)

$$
\frac{\partial \beta}{\partial z} = \frac{\partial \beta}{\partial \alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial \beta}{\partial \beta} \frac{\partial \beta}{\partial z} 
$$

(94)

$$
\frac{\partial \gamma}{\partial y} = \frac{\partial \gamma}{\partial \alpha} \frac{\partial \alpha}{\partial y} + \frac{\partial \gamma}{\partial \beta} \frac{\partial \beta}{\partial y} 
$$

(95)

$$
\frac{\partial \gamma}{\partial z} = \frac{\partial \gamma}{\partial \alpha} \frac{\partial \alpha}{\partial z} + \frac{\partial \gamma}{\partial \beta} \frac{\partial \beta}{\partial z} 
$$

(96)
The procedure for tracing streamlines and computing metrics is: 1) obtain $\frac{\partial x}{\partial y}, \frac{\partial x}{\partial z}, \frac{\partial y}{\partial x}, \frac{\partial y}{\partial z}, \frac{\partial w}{\partial y}$, and $\frac{\partial w}{\partial z}$ 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

$$\hat{\psi}_1 = (1 - \alpha)(1 - \beta)$$  \hspace{1cm} (97)
$$\hat{\psi}_2 = \alpha(1 - \beta)$$  \hspace{1cm} (98)
$$\hat{\psi}_3 = \alpha\beta$$  \hspace{1cm} (99)
$$\hat{\psi}_4 = (1 - \alpha)\beta$$  \hspace{1cm} (100)

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

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

(101)

where \((\cdot)^*\) 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

\[
\theta_L = 0.644 \left( \int_0^s \rho^* \mu^* u_e h^2 dS \right)^{1/2} / (\rho_e \mu_e h)
\]

(102)

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

\[
H^* = 0.5(H_w + H_e) + 0.22(H_{aw} - H_e)
\]

(103)
The boundary-layer thickness is given approximately by the equation

\[
(\delta/\theta)L = 5.55
\]  \hspace{1cm} (104)

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

\[
\dot{q}_T = c_1(Re_{\theta,e})^{-m}(\rho^*/\rho_e)(\mu^*/\mu_e)^m\rho_e u_e (H_{aw} - H_w)(Pr_w)^{-0.4}
\]  \hspace{1cm} (105)

and

\[
\theta_T = (c_2 \int_0^\delta \rho^* u_e (\mu^*)^m h_{\infty} dS)^{c_4}/(\rho_e u_e h)
\]  \hspace{1cm} (106)

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

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

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

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

\[
c_4 = 1/c_3
\]  \hspace{1cm} (111)

\[
c_5 = 2.2433 + 0.93N
\]  \hspace{1cm} (112)

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

\[
N = 12.67 - 6.5\log(Re_{\theta,e}) + 1.21(\log(Re_{\theta,e}))^2
\]  \hspace{1cm} (113)

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

\[
(\delta/\theta)_T = N + 1 + [\left(\frac{N + 2}{N} \frac{H_w}{H_{aw}} + 1\right)(1 + 1.29(Pr_w)^{0.333} \frac{u_e^2}{2H_e})]
\]  \hspace{1cm} (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 $x/L = .6620$ and $x/L = .7703$ 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 $x/L = .6530$. A heating rate of more than $135Btu/ft^2/s$ was measured while the prediction was about $62.5Btu/ft^2/s$. 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 $8.0Btu/ft^2/s$ at this location, and the heating rate
backed out from radiometer data is about 14.0 $Btu/ft^2/s$[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
method.

References


23
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
Figure 8: Comparison of predicted and measured heating rates, $\alpha = 30$ deg.
Figure 9: Pressure and heating rate distribution along the leading edge.

Figure 10: Typical pressure distribution along a leading edge streamline.
Figure 11: Comparison of predicted and measured heating rates, $\alpha = 40\, \text{deg.}$
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 $x/L = 0.5$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 19: Circumferential heating distribution at $x/L = 0.6$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 20: Circumferential heating distribution at $x/L = 0.7$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.

Figure 21: Circumferential heating distribution at $x/L = 0.9$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ deg.
Figure 22: Heating distribution on windward surface at $y = 184.8\,\text{in.}$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\,\text{deg.}$

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

Figure 25: Heating distribution on windward surface at \( y = 322.7 \text{ in.} \), STS-5 case, \( Mach = 9.15, \ Alpha = 37.4 \text{ deg.} \)
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, $Mach = 12.86$, $Alpha = 39.7\ deg$. 

43
Figure 33: Circumferential heating distribution at $z/L = .5$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 34: Circumferential heating distribution at $z/L = .6$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.
Figure 35: Circumferential heating distribution at $z/L = .7$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ deg.

Figure 36: Circumferential heating distribution at $z/L = .9$, 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 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>
<td></td>
</tr>
<tr>
<td>2 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>
<td></td>
</tr>
<tr>
<td>3 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>
<td></td>
</tr>
<tr>
<td>4 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>
<td></td>
</tr>
<tr>
<td>5 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>
<td></td>
</tr>
<tr>
<td>6 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>
<td></td>
</tr>
<tr>
<td>7 streamline output file, contains following: streamline coordinates, metric coefficient, distance from stagnation point and pressure</td>
<td></td>
</tr>
<tr>
<td>8 streamline output file, suitable for BLIMP input, contains distance from stagnation point, metric coefficient and pressure to stagnation pressure ratio</td>
<td></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.qp'
'sts2_ws.dat'
'sts2_wm.dat'
'sts2_basur.xyz'
'sts2_basur.q'
'twall.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
'radiation 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
'imin,imax,kmin,kmax' 140 140 110 110
'transition parameter files' 'n'
'/ceg/kwang/stream/roex3.q2'
'/ceg/kwang/stream/roex3.q4'
```

52
Appendix B - AA2LCH Listing
program aa21ch

This program reads in an Euler flowfield in Cartesian coordinate, and trace the streamline 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, j1, k1
common /stagpt/ xstag, ystag, zstag
common /psidps/ psi(8), dpxdxi(8), dpxdeta(8), dpxdzta(8)
common /nomxyz/ xn(151,110), yn(151,110), zn(151,110), xnp, ynp, znp
common /grdxyz/ xxyz(151,40,110,3), idim, jdim, kdim, & iblank(151,40,110), blank, isubs(2), & jsubs(2), ksubs(2)
common /sfxiet/ xsi(151,110), xiys(151,110), xisz(151,110), & etix(151,110), etys(151,110), etzs(151,110), & flgsn(151), flgsm(151), dvidwyz/ 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/ dxdy(151,110), dxdz(151,110)
common /surfpr/ vel, surfn, up, vp, wp, surfpx, surfpk, dxdyp, dxdy, dxi, det
common /afbond/ ms(151), msr(151), nsp(151), nsr(151)
common /qxyz/ rho(151,40,110), 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 /radeq/ itwall, wall, irad, eps
common /turbml/ iturb
common /tempwl/ twall(1000)

boundary and momentum thickness

common /bltrn/ delt, thet
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), ql1(151,1,110,5)
logical blank, first, back, second, fext
character*80 gdfile, rhofle, rhoul, rhoufl, rhowfl, pfile,
& sfile, mfile, twfile, xfile, qfile, lqfl, lqf2,
& ilfile, outfile, rfile, shfile, errfile
character*30 dummy
character*1 flauna

conversion factors

data cmin, fps, atm, den /2.54, 30.48, 2116.2, 0.01601846/
data rtok /1.8/

Stefan-Boltzman constant (w/cm**2/k)
data sigma /5.67e-12/

input surface grid locations
read(5,*) gdfile
read(5,*) rhofle
read(5,*) rhoul
read(5,*) rhovfl
read(5,*) rowfl
read(5,*) pfile
read(5,*) sfile
read(5,*) mfile
read(5,*) xfile
read(5,*) qfile
read(5,*) twfile
read(5,*) ilfile
read(5,*) outfile
read(5,*) rfwfile
read(5,*) shfile
read(5,*) errfile
read(5,*) dummy, scale
read(5,*) dummy, scale
scale = scale / scale

C input freestream condition
C
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
C
read(5,*) dummy, iturb

C input radiation equilibrium flag
C
read(5,*) dummy, irad

C input surface emissivity
read(5,*) dummy, eps

C input wall temperature flag
C
read(5,*) dummy, itwall
if(itwall .eq. 0)
   read(5,*) dummy, wallt
endif

C get free stream properties
C
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, xmuiinf, 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
vninf = sqrt(pinf*1.0133e6/rhoinf)

open(unit=7,file=gdfile,form='unformatted',
&status='old')
open(unit=8,file=rhofile,form='unformatted',
&status='old')
open(unit=9,file=rhoufile,form='unformatted',
&status='old')
open(unit=10,file=rhowfile,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

additional file
open(unit=15,file=ilfile,form='unformatted',status='unknown')
open(unit=24,file=outfile,form='formatted',status='unknown')
open(unit=25,file=rwfile,form='formatted',status='unknown')
open(unit=30,file=shfile,form='formatted',status='unknown')
open(unit=40,file=errfile,form='formatted',status='unknown')

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)

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
   20

modification of 1/11/94

pi = atan2(1.0,0.0) * 2.
dangle = pi / kdim
do 30 j = 1,jdim
   do 30 k = 1,kdim
      angle = (k-1)*dangle
xyz(1,j,k,2) = 1.e-7 * sin(angle) * scale
xyz(1,j,k,3) = 1.e-7 * cos(angle) * scale
continue

c end of modification

c read rho
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
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)
istag = 0
do 160 i = 1,idim
us(i) = u(i,1,ksdim)
160 continue

c locate a pair grid points between which u changes sign.
do 170 i = idim,2,-1
iml = i - 1
if(us(i)*us(iml) .lt. 0.) then
  is = iml
  ispl = is + 1
  go to 180
else if(us(i) .eq. 0.) then
  istag = 1
  is = i
  ispl = is + 1
  go to 180
else if(us(iml) .eq. 0.) then
  istag = 1
  is = iml
  ispl = is + 1
  go to 180
endif
continue

write(6,*) ' Stagnation point cannot be located'
stop
continue
write(6,*) ' is,ispl',is,ispl

find coordinates of stagnation point

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

compare with maximum pressure

do 190 i = 1,idim
  if(p(i,1,1) .gt. pstag) pstag = p(i,1,1)
  if(p(i,1,ksdim) .gt. pstag) pstag = p(i,1,ksdim)
continue
write(6,*) ' Stagnation point is at', xstag,' 0.',zstag
write(6,*) ' Stagnation pressure is ',pstag

find starting point location

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 = isspl
    dels = ds2
  endif
endif
if(alp .gt. 0.) then
  xstagl = xstag + dels/ds2 * (xyz(ispl,1,ksdim,1)-xstag)
zstagl = zstag + dels/ds2 * (xyz(ispl,1,ksdim,3)-zstag)
ystag1 = 0.
xstag2 = xstag + dels/dsl * (xyz(is, l, ksd1m, 1) - xstag)
zstag2 = zstag + dels/dsl * (xyz(is, l, ksd1m, 3) - zstag)
ystag2 = 0.
else
xstag1 = xstag + dels/dsl * (xyz(is, l, ksd1m, 1) - xstag)
zstag1 = zstag + dels/dsl * (xyz(is, l, ksd1m, 3) - zstag)
ystag1 = 0.
xstag2 = xstag + dels/ds2 * (xyz(ispl, l, ksd1m, 1) - xstag)
zstag2 = zstag + dels/ds2 * (xyz(ispl, l, ksd1m, 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
ms1 = 1
ns1 = 1
mdim = idim
ndim = kdim
call bounds(mdim, ndim, ms1, mdim, ns1, ndim, msper, nsper)

compute surface metrics

call surmet(mdim, ndim, ms1, mdim, ns1, ndim, rasper, 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,*) lqf1
read(5,*) lqf2
endif

open (unit=21, file=lqf1, form='unformatted', status='old')
open (unit=22, file=lqf2, form='unformatted', status='old')

read(21) ildim, jldim, kldim
read(21) dum1, dum2, dum3, dum4
read(21) (((qll(i, i, k, nx), i=1, ildim), k=1, kldim), nx=1, 5)

read(22) ildim, jldim, kldim
read(22) dum1, dum2, dum3, dum4
read(22) (((ql2(i, i, k, nx), i=1, ildim), k=1, kldim), nx=1, 5)

endif

read(5,*) dummy, nbstep
if (iback .eq. 1) 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.
bavs = 0.
qavs = 0.
ifatal = 0

c nsl is total number of streamlines to be generated
c dtr is the conversion factor from degree to radian
c dthe is the initial delta angle between two adjacent streamlines
c
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,l,kstart,1)
    yin = xyz(istart,l,kstart,2)
    zin = xyz(istart,l,kstart,3)
  endif
else
  first = .false.
  if(nl .eq. 1) first = .true.
  c initial position
  c
  if(nl .eq. 1) then
    xin = xstgl
    yin = ystgl
    zin = zstgl
  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
  c
  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
  c
  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 fnndnrp(ms,me,ns,ne,xin,yin,zin)
istat = 0
delxi = 0.
dele = 0.
call quads(delxi,dele,xin,yin,zin,istat)
if(istat .gt. 1) then
   continue
write(6,*) ' Search failed, istat = ',istat
write(40,*) ' Location ',istart,kstart,' failed'
go to 240
endif
ii = isav
ki = ksav
if(nl .eq. 1) then
   if(kl .eq. 1 .or. kl .eq. kdim) then
dele = 0.
endif
cendif
c11 = delxi * dele
c10 = delxi - c11
c01 = dele - c11
c00 = 1. - delxi - dele + c11
ipl = msp(il)
kp1 = nsp(kl)
ui = u(il,1,kl)*c00 + u(ipl,1,kl)*c10 + u(ipl,1,kp1)*c11
& + u(il,1,kp1)*c01
vi = v(il,1,kl)*c00 + v(ipl,1,kl)*c10 + v(ipl,1,kp1)*c11
& + v(il,1,kp1)*c01
wi = w(il,1,kl)*c00 + w(ipl,1,kl)*c10 + w(ipl,1,kp1)*c11
& + w(il,1,kp1)*c01
pi = p(il,1,kl)*c00 + p(ipl,1,kl)*c10 + p(ipl,1,kp1)*c11
& + p(il,1,kp1)*c01
dxdy = dxdy(il,kl)*c00 + dxdy(ipl,kl)*c10 +
& dxdy(ipl,kp1)*c11 + dxdy(il,kp1)*c01
dxdz = dxdz(il,kl)*c00 + dxdz(ipl,kl)*c10 +
& dxdz(ipl,kp1)*c11 + dxdz(il,kp1)*c01
find norm vector
x nin = xn(il,kl)*c00 + xn(ipl,kl)*c10 +
& xn(ipl,kp1)*c11 + xn(il,kp1)*c01
ynin = yn(il,kl)*c00 + yn(ipl,kl)*c10 +
& yn(ipl,kp1)*c11 + yn(il,kp1)*c01
znin = zn(il,kl)*c00 + zn(ipl,kl)*c10 +
& zn(ipl,kp1)*c11 + zn(il,kp1)*c01
interpolate boundary layer transition parameters
if(flaura .eq. 'y' .or. flora .eq. 'Y') then
   repf = qll(il,1,kl,4)*c00 + qll(ipl,1,kl,4)*c10 +
& qll(ipl,1,kp1,4)*c11 + qll(il,1,kp1,4)*c01
   thkm = qll(il,1,kl,2)*c00 + qll(ipl,1,kl,2)*c10 +
& qll(ipl,1,kp1,2)*c11 + qll(il,1,kp1,2)*c01
   thkd = qll(il,1,kl,3)*c00 + qll(ipl,1,kl,3)*c10 +
& qll(ipl,1,kp1,3)*c11 + qll(il,1,kp1,3)*c01
   rek = qll(ipl,1,kl,3)*c11 + qll(ipl,1,kp1,3)*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)

modification of 8/16/93

c
y(6) = hmet * (ynin*win/vel - znin*vin/vel)
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) / (xinin*win/vel - znin*uin/vel)
hmet2 = y(5) / (xinin*vin/vel - ynin*uin/vel)
hmet=(y(5)-y(4))/(xinin*(vin+win)/vel-uin/vel*(ynin+znin))

modification of 8/16/93

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

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

c output streamline locations

c
write(13,*) ' Streamline No. ',nl
write(14,*) ' Streamline No. ',nl
write(6,*t) ' Streamline No. ',nl
write(13,9001) y(1),y(2),y(3),dist,hmet,pin
if(flaura .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
c    write(17,9010) y(1),y(2),y(3),y(4)-xmin,y(2)-ynin,y(3)-zmin
write(6,*), y(1),y(2),y(3),y(4),surf,vel,hmet,pin

integration loop

h = hsave
hmin = hmin
write(6,*), 'real step size', h, hmin
xsave = y(1)
ysave = y(2)
zsave = y(3)
do 250 i = 1,nstep
    continue
    if(second) then
        i = nstep
    else
        i = 1
    endif
    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
    endif
write(6,*), 'back', y(1), y(2), y(3), h
endif
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
    endif
    i = i + 1
    nstep = i
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), surf / vel

hmet = surf / 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)

hmet = surfn/(vel-up/vel*(up-vp*dxdyp-wp*dxdzp))*
& (vp*y(5) - wp*y(4))
hmet1 = -surfny(4)/(vp/vel+up/vel*dxdyp)
& + (y(5)-ysave)**2 + (y(2)-ysave)**2
& + (y(3)-zsave)**2)
xsave = y(1)
ysave = y(2)
zsav = y(3)
hs = h

interpolate boundary layer transition parameters

if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
    il = isav
    k1 = ksav
    c11 = dxi * det
    c10 = dxi - c11
    c01 = det - c11
    c00 = 1. - dxi - det + c11
    ip1 = msp(il)
    kpl = nsp(k1)
    repf = qll(il,1,k1,4)*c00 + qll(ip1,1,k1,4)*c10 +
& qll(ip1,1,k1,4)*c11 + qll(il,1,k1,4)*c01
    thkm = qll(il,1,k1,2)*c00 + qll(ip1,1,k1,2)*c10 +
& qll(ip1,1,k1,2)*c11 + qll(il,1,k1,2)*c01
    thkd = qll(il,1,k1,3)*c00 + qll(ip1,1,k1,3)*c10 +
& qll(ip1,1,k1,3)*c11 + qll(il,1,k1,3)*c01
    rek = qll(il,1,k1,3)*c00 + qll(ip1,1,k1,3)*c10 +
& qll(ip1,1,k1,3)*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,surfp
write(14,9002) dist/12.,hmet/12.,surfp/pstag
write(15) y(1),y(2),y(3),-xnp,-ynp,-znp,vel,surfp,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,*)' i(1),y(2),y(3),y(4),y(5),dist,hmet,surfp
if(.not.back) then
    if(i .le. I) go to 230
endif
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

230
c call subroutine heat to evaluate convective heating
c
!call heat(qw, qwbtu, wt, 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) = wt
   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'
   endif
   kstart = kstart + 1
   endif
endif
hsave = saveh
continue
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)
    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, lx))
9999 continue
stop
end

subroutine rk34

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

common /save/ igsav, issav, isav, 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,
& .5547922244, -1.406455023, 1.770028430, .1612244898,
& .5998345284, .2389409818/

1 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 + a1*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)*1.e-3) c(i) = 1.
c if(abs(f(i)) .le. 1.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
c if(h .gt. 1.0e-3) h = 1.0e-3
11 continue
if(abs(h) .gt. abs(hmin)) go to 12
write(6,*) ' relative truncation error criterion could not be satisfied/''e,hmin'
c stop
12 continue
h = h * 0.5
write(6,*)' h = ',h
go to 1
end

c subroutine funct

c 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, Ii0,3) , 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 /dvdwyz/ dvdy(151,110),dvdz(151,110),dwdy(151,110),
& dwdx(151,110)
c modification of 8/16/93
c common /duxxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
& dwdx(151,110),dwdx(151,110)
c end of modification
common /geom/ dxdy(151,110),dxdz(151,110)
common /surfpr/ vel,surfn,up,wp,surfp,dxdzp,dxdyp,dxi,det
common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
 &     first,back,second,ifatal
   logical first,back,second
   dimension y(50),f(50)
   sx = y(1)
   sy = y(2)
   sz = y(3)

surface interpolation
   ms = isav - 3
   me = isav + 3
   if(ms .lt. 1) ms = 1
   if(me .gt. idim) me = idim
   ns = ksav - 3
   ne = ksav + 3
   if(ns .lt. 1) ns = 1
   if(ne .gt. kdim) ne = kdim
   ns = 1
   ne = kdim
   call fndnrp(ms,me,ns,ne,sx,sy,sz)

   istat = 0
   delxi = 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 = ksav
   if(first) then
      if(kl .eq. 1 .or. kl .eq. kdim) then
         delet = 0.
      endif
   endif
   c
   dxi = delxi
   det = delet
   c11 = delxi * delet
   c10 = delxi - c11
   c01 = delet - c11
   c00 = 1. - delxi - delet + c11
   ip1 = msp(il)
   kpl = nsp(kl)
   up = u(il,1,k1) *c00 + u(ip1,1,k1) *c10 + u(ip1,1,kpl) *c11
   & + u(il,1,kpl) *c01
   vp = v(il,1,k1) *c00 + v(ip1,1,k1) *c10 + v(ip1,1,kpl) *c11
   & + v(il,1,kpl) *c01
   wp = w(il,1,k1) *c00 + w(ip1,1,k1) *c10 + w(ip1,1,kpl) *c11
   & + w(il,1,kpl) *c01
   surfp = p(il,1,k1) *c00 + p(ip1,1,k1) *c10 + p(ip1,1,kpl) *c11
   & + p(il,1,kpl) *c01
   dxdyp = dxdy(il,1,k1) *c00 + dxdy(ip1,1,k1) *c10 +
   & dxdy(ip1,1,kpl) *c11 + dxdy(il,1,kpl) *c01
   dxdzp = dxdz(il,1,k1) *c00 + dxdz(ip1,1,k1) *c10 +
   & dxdz(ip1,1,kpl) *c11 + dxdz(il,1,kpl) *c01
   dxdyp = dxdy(il,1,k1) *c00 + dxdy(ip1,1,k1) *c10 +
   & dxdy(ip1,1,kpl) *c11 + dxdy(il,1,kpl) *c01
   dxdzp = dxdz(il,1,k1) *c00 + dxdz(ip1,1,k1) *c10 +
   & dxdz(ip1,1,kpl) *c11 + dxdz(il,1,kpl) *c01
modification of 8/16/93

dudxp = dudx(il,kl)*c00 + dudx(ipl,kpl)*c01 +
& dudyp = dudy(il,kl)*c00 + dudy(ipl,kpl)*c01 +
& dudzp = dudz(il,kl)*c00 + dudz(ipl,kpl)*c01 +
& dvdxp = dvdx(il,kl)*c00 + dvdx(ipl,kpl)*c01 +
& dwdx = dwdx(il,kl)*c00 + dwdx(ipl,kpl)*c01 +
& dvdy = dvdy(il,kl)*c00 + dvdy(ipl,kpl)*c01 +
& dwdyp = dwdyp(il,kl)*c00 + dwdyp(ipl,kpl)*c01 +
& dwdzp = dwdz(il,kl)*c00 + dwdz(ipl,kpl)*c01 +

surface norm vector

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

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

f(1) = up
f(2) = vp
f(3) = wp
f(4) = dvdxp * y(6) + dwdx * y(5)
f(5) = dvdxp * y(6) + dwdx * y(5)
f(6) = dvdxp * y(6) + dwdx * 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(1),f(2),f(3),f(4),f(5)
return
end

subroutine bounds ( mdim, ndim, ms, me, ns, ne, 
& msper, 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
```fortran
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/ xn(151,110), yn(151,110), zn(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/ xixs(151,110), xiys(151,110), xizz(151,110),
  & etxs(151,110), etys(151,110), etzs(151,110),
  & flgsm(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 /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),
  & dvdx(151,110), dwdx(151,110)
end of modification
common /geom/ dxdy(151,110), dxdz(151,110)
dimension jacm(151), jacn(151)

njac = 0

Initialize arrays.
do 10 m = 1, mdim
  flgsm(m) = 0.
  jacm(m) = 0
  jacn(n) = 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.
```

xiys(m,n) = 0.
xisz(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.
dwdz(m,n) = 0.
dxdy(m,n) = 0.
dxdz(m,n) = 0.
continue

Set up weight functions.

do 20 m = ms,me
   flgs(m) = .5
20 continue
   if (msper.eq.0) then
      flgs(me) = 1.
      flgs(ms) = 1.
   endif

do 21 n = ns,ne
   flgs(n) = .5
21 continue
   if (nsper.eq.0) then
      flgs(ns) = 1.
      flgs(ne) = 1.
   endif

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,l) - xyz(mr, l,n,l)) * flgs(m)
      yxi = (xyz(mp, l,n,2) - xyz(mr, l,n,2)) * flgs(m)
      zxi = (xyz(mp, l,n,3) - xyz(mr, l,n,3)) * flgs(m)
      vxi = (v(mp, l,n) - v(mr, l,n)) * flgs(m)
      wxi = (w(mp, l,n) - w(mr, l,n)) * flgs(m)
      xet = (xyz(m,l,np, l) - xyz(m,l,nr, l)) * flgs(n)
      yet = (xyz(m,l,np,2) - xyz(m,l,nr,2)) * flgs(n)
      zet = (xyz(m,l,np,3) - xyz(m,l,nr,3)) * flgs(n)
      vet = (v(m,l,np) - v(m,l,nr)) * flgs(n)
      wet = (w(m,l,np) - w(m,l,nr)) * flgs(n)
30 continue

modification of 8/16/93

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

end of modification

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

c if(m .eq. ms) then
xn(m,n) = 1.
yn(m,n) = 0.
zn(m,n) = 0.
else if(m .eq. me .and. n .eq. 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)
xnl = yl*z2 - y2*zl
ynl = x2*zl - xl*z2
zn = xl*y2 - x2*yl
sqri = 1./sqrt(xnl*xnl + ynl*ynl + znl*zn)
xnl = xnl*sqri
ynl = ynl*sqri
zn1 = zn1*sqri
  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 = yl*z2 - y2*zl
yn2 = x2*zl - xl*z2
zn2 = xl*y2 - x2*yl
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)
znn = 0.5*(zn1+zn2)
sqri = 1./sqrt(xnn*xnn + ynn*ynn + znn*znn)
xn(m,n) = xnn*sqri
yn(m,n) = ynn*sqri
zn(m,n) = znn*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-1,1,n,3)
xnl = yl*z2 - y2*zl
ynl = x2*zl - xl*z2
zn1 = xl*y2 - x2*yl
sqri = 1./sqrt(xnl*xnl + ynl*ynl + znl*zn)
xnl = xnl*sqri
ynl = ynl*sqri
zn1 = zn1*sqri
  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)
xn2 = yl*z2 - y2*zl
yn2 = x2*zl - xl*z2
zn2 = xl*y2 - x2*yl
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)
ynn = 0.
znn = 0.5*(zn1+zn2)

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

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

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

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

xnn = 0.5*(xn1+xn2)
ynn = 0.5*(yn1+yn2)
ynn = 0.
znn = 0.5*(zn1+zn2)

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

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

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

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn(m,n) = xn1*sqri
yn(m,n) = yn1*sqri
zn(m,n) = zn1*sqri

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

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

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn(m,n) = xn1*sqri
yn(m,n) = yn1*sqri
zn(m,n) = zn1*sqri

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

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

xnn = 0.5*(xn1+xn2)
ynn = 0.5*(yn1+yn2)
ynn = 0.
znn = 0.5*(zn1+zn2)

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

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

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

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn(m,n) = xn1*sqri
yn(m,n) = yn1*sqri
zn(m,n) = zn1*sqri

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

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

xnn = 0.5*(xn1+xn2)
ynn = 0.5*(yn1+yn2)
ynn = 0.
znn = 0.5*(zn1+zn2)

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

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

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

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn(m,n) = xn1*sqri
yn(m,n) = yn1*sqri
zn(m,n) = zn1*sqri

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

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

xnn = 0.5*(xn1+xn2)
ynn = 0.5*(yn1+yn2)
ynn = 0.
znn = 0.5*(zn1+zn2)

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

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

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

sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
xn(m,n) = xn1*sqri
yn(m,n) = yn1*sqri
zn(m,n) = zn1*sqri

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

x1 = xyz(m,1,n-1,1) - xyz(m,1,n,1)
y1 = xyz(m,1,n-1,2) - xyz(m,1,n,2)
z1 = xyz(m,1,n-1,3) - xyz(m,1,n,3)
x2 = xyz(m,1,n,1) - xyz(m,1,n,1)
y2 = xyz(m,1,n,2) - xyz(m,1,n,2)
z2 = xyz(m,1,n,3) - xyz(m,1,n,3)
\[ z_2 = xyz(m-1,1,n,3) - xyz(m,1,n,3) \]
\[ x_{n1} = y_1 z_2 - y_2 z_1 \]
\[ y_{n1} = x_2 z_1 - x_1 z_2 \]
\[ y_{n1} = 0. \]
\[ z_{n1} = x_1 y_2 - x_2 y_1 \]
\[ sqri = 1./sqrt(x_{n1} x_{n1} + y_{n1} y_{n1} + z_{n1} z_{n1}) \]
\[ x_{n}(m,n) = x_{n1} * sqri \]
\[ y_{n}(m,n) = y_{n1} * sqri \]
\[ z_{n}(m,n) = z_{n1} * sqri \]

\textbf{else}

\[ x_1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1) \]
\[ y_1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2) \]
\[ z_1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3) \]
\[ x_2 = xyz(m-1,1,n,1) - xyz(m,1,n-1,1) \]
\[ y_2 = xyz(m-1,1,n,2) - xyz(m,1,n-1,2) \]
\[ z_2 = xyz(m-1,1,n,3) - xyz(m,1,n-1,3) \]
\[ x_{n1} = y_1 z_2 - y_2 z_1 \]
\[ y_{n1} = x_2 z_1 - x_1 z_2 \]
\[ z_{n1} = x_1 y_2 - x_2 y_1 \]
\[ sqri = 1./sqrt(x_{n1} x_{n1} + y_{n1} y_{n1} + z_{n1} z_{n1}) \]
\[ x_{n2} = x_2 * z_1 - x_1 * z_2 \]
\[ y_{n2} = x_1 * y_2 - x_2 * y_1 \]
\[ sqri = 1./sqrt(x_{n2} x_{n2} + y_{n2} y_{n2} + z_{n2} z_{n2}) \]
\[ x_{n3} = x_{n2} * sqri \]
\[ y_{n3} = y_{n2} * sqri \]
\[ z_{n3} = z_{n2} * sqri \]

\textbf{c}

\[ x_1 = xyz(m+1,1,n+1,1) - xyz(m,1,n,1) \]
\[ y_1 = xyz(m+1,1,n+1,2) - xyz(m,1,n,2) \]
\[ z_1 = xyz(m+1,1,n+1,3) - xyz(m,1,n,3) \]
\[ x_2 = xyz(m+1,1,n,1) - xyz(m,1,n+1,1) \]
\[ y_2 = xyz(m+1,1,n,2) - xyz(m,1,n+1,2) \]
\[ z_2 = xyz(m+1,1,n,3) - xyz(m,1,n+1,3) \]
\[ x_{n3} = y_1 z_2 - y_2 z_1 \]
\[ y_{n3} = x_2 z_1 - x_1 z_2 \]
\[ z_{n3} = x_1 y_2 - x_2 y_1 \]
\[ sqri = 1./sqrt(x_{n3} x_{n3} + y_{n3} y_{n3} + z_{n3} z_{n3}) \]
\[ x_{n3} = x_{n3} * sqri \]
\[ y_{n3} = y_{n3} * sqri \]
\[ z_{n3} = z_{n3} * sqri \]

\textbf{c}

\[ x_1 = xyz(m+1,1,n-1,1) - xyz(m,1,n,1) \]
\[ y_1 = xyz(m+1,1,n-1,2) - xyz(m,1,n,2) \]
\[ z_1 = xyz(m+1,1,n-1,3) - xyz(m,1,n,3) \]
\[ x_2 = xyz(m,1,n-1,1) - xyz(m+1,1,n,1) \]
\[ y_2 = xyz(m,1,n-1,2) - xyz(m+1,1,n,2) \]
\[ z_2 = xyz(m,1,n-1,3) - xyz(m+1,1,n,3) \]
\[ x_{n4} = y_1 z_2 - y_2 z_1 \]
\[ y_{n4} = x_2 z_1 - x_1 z_2 \]
\[ z_{n4} = x_1 y_2 - x_2 y_1 \]
\[ sqri = 1./sqrt(x_{n4} x_{n4} + y_{n4} y_{n4} + z_{n4} z_{n4}) \]
\[ x_{n4} = x_{n4} * sqri \]
\[ y_{n4} = y_{n4} * sqri \]
\[ z_{n4} = z_{n4} * sqri \]

\textbf{c}
\[ x_{nn} = 0.25(x_{n1} + x_{n2} + x_{n3} + x_{n4}) \]
\[ y_{nn} = 0.25(y_{n1} + y_{n2} + y_{n3} + y_{n4}) \]
\[ z_{nn} = 0.25(z_{n1} + z_{n2} + z_{n3} + z_{n4}) \]
\[ sqri = 1./\sqrt{(x_{nn}^2 + y_{nn}^2 + z_{nn}^2)} \]
\[ x(n, m) = x_{nn} \times sqri \]
\[ y(n, m) = y_{nn} \times sqri \]
\[ z(n, m) = z_{nn} \times sqri \]
\]  
\]  
endif

\[ r_{xx} = x_{xi}^2 + y_{xi}^2 + z_{xi}^2 \]
\[ r_{xx} = x_{xi} \times x_{et} + y_{xi} \times y_{et} + z_{xi} \times z_{et} \]
\[ r_{re} = x_{et}^2 + y_{et}^2 + z_{et}^2 \]
\[ r_jac = (r_{xx} \times r_{re} - r_{xx}^2) \]

* 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.

if (r_jac.gt.0. .and. r_jac .gt. 1.e-10) then
  if (r_jac .gt. 0.) then
    rd = 1./r_jac
    xixs(m,n) = (r_{re} \times x_{xi} - r_{xx} \times x_{et}) \times rd
    xiys(m,n) = (r_{re} \times y_{xi} - r_{xx} \times y_{et}) \times rd
    xisz(m,n) = (r_{re} \times z_{xi} - r_{xx} \times z_{et}) \times rd
    etxs(m,n) = (-r_{xx} \times x_{xi} + r_{rx} \times x_{et}) \times rd
    etys(m,n) = (-r_{xx} \times y_{xi} + r_{rx} \times y_{et}) \times rd
    etzs(m,n) = (-r_{xx} \times z_{xi} + r_{rx} \times z_{et}) \times rd
  end if

* surface velocity derivatives

\[ dv_{dy}(m, n) = v_{xi} \times xiys(m, n) + v_{et} \times etys(m, n) \]
\[ dv_{dz}(m, n) = v_{xi} \times xizs(m, n) + v_{et} \times etzs(m, n) \]
\[ dw_{dy}(m, n) = w_{xi} \times xiys(m, n) + w_{et} \times etys(m, n) \]
\[ dw_{dz}(m, n) = w_{xi} \times xizs(m, n) + w_{et} \times etzs(m, n) \]

* modification of 8/16/93

\[ dud_{x}(m, n) = u_{xi} \times xiys(m, n) + u_{et} \times etxs(m, n) \]
\[ dud_{y}(m, n) = u_{xi} \times xiys(m, n) + u_{et} \times etys(m, n) \]
\[ dud_{z}(m, n) = u_{xi} \times xizs(m, n) + u_{et} \times etzs(m, n) \]
\[ dwd_{x}(m, n) = w_{xi} \times xiys(m, n) + w_{et} \times etxs(m, n) \]
\[ dwd_{y}(m, n) = w_{xi} \times xiys(m, n) + w_{et} \times etzs(m, n) \]

* end of modification

* surface derivatives

\[ dx_{dy}(m, n) = xx_{i} \times xiys(m, n) + xx_{et} \times etys(m, n) \]
\[ dx_{dz}(m, n) = xx_{i} \times xizs(m, n) + xx_{et} \times etzs(m, n) \]

if(n .eq. ns .or. n .eq. ne) then
  write(50,*), m, n, dv_{dy}(m, n), dv_{dz}(m, n)
  write(51,*), m, n, dw_{dy}(m, n), dw_{dz}(m, n)
  write(52,*), m, n, dx_{dy}(m, n), dx_{dz}(m, n)
endif

else
  njac = njac + 1
  jacm(njac) = m
  jacn(njac) = n
  xixs(m,n) = 0.
  xiys(m,n) = 0.
  xisz(m,n) = 0.
  etxs(m,n) = 0.
  etys(m,n) = 0.
  etzs(m,n) = 0.
  dv_{dy}(m,n) = 0.
  dv_{dz}(m,n) = 0.
  dw_{dy}(m,n) = 0.
dwdz(m,n) = 0.
dxdy(m,n) = 0.
dxdz(m,n) = 0.
endif

30  continue

c  Write questionable grid points to output log file.
c  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.')
      return
      end

c subroutine fndnrp ( ms,me,ns,ne,
     & xp,yp,zp)
     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,nsave,asav,bsav,gsav,
     & first,back,second,ifatal
     logical blank, first, back, second
     c Find point on surface subset that is closest to (xp, yp, zp).
     dsave = 1.0e8
     do 10 n = ns,ne
        dist = (xyz (m,1,n,1)-xp)**2 +
        1 (xyz (m,1,n,2)-yp)**2 +
        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
      return
    end

c subroutine quads ( delxi,delet,xp, yp, zp, istat)
c  parameter (toler=0.1)
  common /save/ igsav,issav,js,jsav,ks,asav,bsav,gsav,
  & first,back,second,ifatal
  common /grdxyz/ xyz(151,40,110,3),jsmax,ismax,kmax,
  & iblank(151,40,110), blank,isubs(2),
  & jsubs(2),ksubs(2)
  common /sfxiet/ xixs (151,110) ,xiys (151,110),xizs (151,110),
  & etxs (151,110),etys (151,110),etzs (151,110),
  & flgsm(151),flgsm(151)
  common /sfbond/ jsp(151),jsr(151),ksp(151),ksr(151)
  logical first,back,second
  ifail = 0
  c 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
  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,kp,2) * c11
  yt = xyz(js,1,ks,2) * c00 + xyz(jp,1,ks,2) * c10
       + xyz(jp,1,kp,2) * c01 + xyz(jp,1,kp,3) * c11
  zt = xyz(js,1,ks,3) * c00 + xyz(jp,1,ks,3) * c10
       + xyz(jp,1,kp,3) * c01 + xyz(jp,1,kp,4) * 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
       + etzs(jp,kp) * c11
else
  xix = xixs(js,ks) * c00 + xiys(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
       + etzs(jp,kp) * c11
endif
enddo
endif

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

11 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

Update cell number.

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.)
dele = min(max(delet,0.),1.)
istat = 1
  write(6,* ) ' delxi,delet',delxi,delet
goto 20
endif

10 continue
write(6,*) 'delxi,delet',delxi,delet
Convergence failed. (Hope this never happens, but note it.)
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
write(6,*), 'js,ks', js,ks
write(6,*), 'delxi delet', delxi, delet
return
end
subroutine heat(qw, qwbtu, wlttemp, cf)
This program reads in streamline paths and metric coefficients, uses
Zoby's approximate heating methodology and used Gupta's curve fit to
calculate equilibrium air thermodynamics and transport properties to
evaluate convective heating along the streamline.
common /connty/, il, j1, k1
common /stagpt/, xtag, ystag, zstag
common /psidps/, psi(8), dpdxi(8), dpdeta(8), dpdzta(8)
common /grdxyz/, xz(151,40,110,3), idim, jdim, kdim,
  iblank(151,40,110), blank, isubs(2),
  jaubs(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, 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)
boundary and momentum thickness
common /bltrn/, deltal, thetl
dimension xs(1000), ys(1000), zs(1000), xn(1000), yn(1000),
  zn(1000), vels(1000), ps(1000), hmet(1000),
  isaves(1000), k saves(1000)
logical blank, back, first, second
conversion factors
data cmin, fps, atm, den /2.54, 30.48, 2116.2, .01601846/
data rtok /1.8/
Stefan-Boltzmann constant(w/cm**2/k)
data sigma /5.67e-12/
rewind 15
is = 1
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

is = is + 1

if(itwall .eq. 1) then
  read(18, *, end=95) twall(it)
  twall(it) = twall(it) / rtok
  it = it + 1
  go to 92
endif

is = is - 1
if(itwall .eq. 0) then
  do 97 iu = 1, is
    twall(iu) = wallt / rtok
  enddo
  ivit = 0
  isav = isaves(iter)
  jsav = 1
  asav = 0.
  bsav = 0.
  gsav = 0.
  s = 0.
  sum0 = 0.
  f1 = 0.
  ite = 0
  iweq = 0
  do 200 iter = 1, is
    rewind 6
    write(6, *) ' iter = ', iter
    icon = 0
    velle = vels(iter) * vninf
    velet = velle
    velle = vels(iter) * 30.48
    pe = ps(iter) * pinf
    hinf = cpinf * tinf + 0.5 * vinf ** 2 * 2.3901e-8
    write(6, *) ' hinf(computed) = ', hinf
  enddo
  isav = isaves(iter)
  jsav = 1
ksav = ksaves(iter)
continue

c edge enthalpy (cal/g)

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

c 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

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

iflag = 2
if(igas .eq. 0) iflag = 0

iflag = 2
if(igas .eq. 0) iflag = 0
tw = twall(iter)
call enthalpy(pe,hw,tw,iflag)
if(ifatal .ne. 0) return

c 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)*velet**2 * 2.3901e-11

c 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

c 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*cmin
s = s + ds
c
sum = sum0 + 0.5*(f1+f2)*ds
c
endif
c
c
check if turbulence heating is required
c
rem = 1.
c3 = 2.
c4 = 0.5
c2 = .440896
iterb = 0
thetlo = 0.
110 continue
c
f2 = rhostr*xmustr**rem*vele*hmet(iter)**c3
f2 = rhostr*xmustr**rem*vele*hmet(iter)**c3
if(iter .eq. 1) then
sum = 0.25*f2*ds
sum = 0.5*f2*ds
else
sum = 0.5 * (f1 + f2) * ds + sum0
endif
c
thetl = (c2*sum)**c4/(rho*vele*hmet(iter)) ** .93
c
thetl = (c2*sum)**c4/(rho*vele*hmet(iter))
c
thetl = (c2*sum)**c4/(rho*vele*hmet(iter))
c
write(6,*) ' thetlo, thetl = ',thetlo,thetl
reth = rho*vele*thetl
reth = rho*vele*thetl
if(iturb .ne. 0) then
if(abs(thetl-thetlo) .gt. .01 .or. iterb .eq. 0) then
rncap = 12.67-6.5*alog10(reth)+1.21*(alog10(reth))**2
rncap = anint(rncap)
rem = 2. / (rncap + 1.)
c5 = 2.2433 + 0.93 * rncap
c3 = 1. + rem
c4 = 1. / c3
c1 = ((1./c5)**(2.*rncap/(rncap+1.))*(rncap/((rncap+1.)*
&
c2 = (1.+rem)*c1
iterb = iterb + 1
write(6,*) ' iterb',iterb
write(6,*)'rncap,rem,c1,c2,c3,c4,c5',rncap,rem,c1,c2,c3,c4,c5
if(iterb .ge. 20) then
write(6,*) ' Too many iterations !!!'
ifatal = 1
if(iwit .eq. 0) then
qw = 0.
qwbtu = 0.
wltemp = 0.
cf = 0.
endif
return
endif
thetlo = thetl
go to 110
endif
c
f2 = rhostr*xmustr*vele*hmet(iter)**2
c
thetl = .664*sqrt(sum)/(rho*vele*hmet(iter))*.93
c
thetl = .664*sqrt(sum)/(rho*vele*hmet(iter))
c
write(6,*) ' Momentum thickness = ',thetl

compute boundary layer thickness

if(iturb .eq. 0) then
deltal = 5.55*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 = thet1 * thedd
endif

  deltal = 5.55*thet1*1.18
write(6,*) ' Boundary layer thickness = ',deltal

  deltal = 0.
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
  igrd = 1
  nsubs = 1
  call close3(isrch,igrd,nsubs,i,j,k,a,b,g,xble,yble,zble,isub,
  & 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
  jl = j
  kl = k
  call find(xble,yble,zble)
  if(ifatal .ne. 0) return
  
  velb = vele
  velb = velet
  
  ipl = il + 1
  jpl = jl + 1
  kpl = kl + 1
  
  ue = u(il, jl, kl)*psi(1)+u(ipl, jl, kl)*psi(2)+u(ipl, jpl, kl)*psi(3)
  & + u(il, jpl, kl)*psi(4)+u(il, jpl, kpl)*psi(5)+u(ipl, jpl, kpl)*psi(6)
  & + u(ipl, jpl, kpl)*psi(7)+u(ii, jpl, kpl)*psi(8)
  
  ve = v(il, jl, kl)*psi(1)+v(ipl, jl, kl)*psi(2)+v(ipl, jpl, kl)*psi(3)
  & + v(il, jpl, kl)*psi(4)+v(il, jpl, kpl)*psi(5)+v(ipl, jpl, kpl)*psi(6)
  & + v(ipl, jpl, kpl)*psi(7)+v(ii, jpl, kpl)*psi(8)
  
  we = w(il, jl, kl)*psi(1)+w(ipl, jl, kl)*psi(2)+w(ipl, jpl, kl)*psi(3)
pe = p(il, j1, k1)*psi(1)+p(ip1, j1, k1)*psi(2)+p(ip1, jpl, kpl)*psi(3)
& + p(ip1, jpl, kpl)*psi(4)+p(il, jpl, kpl)*psi(5)+p(il, jpl, kpl)*psi(6)
& + p(ip1, jpl, kpl)*psi(7)+p(il, jpl, kpl)*psi(8)

pe = pe*pinf

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

velen = ue*xn(iter) + ve*yn(iter) + we*zn(iter)

velet = sqrt(vele*vele - velen*velen)

vele = vele*vninf

vel = vele*30.48

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

tconv = (vele - velb)/velb

tconv = (velet - velb)/velb

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

go to 100

compute laminar heating

continue

momentum thickness Reynolds number

reth = rhoe*vele*thetl/xmue

write(6,*) ' reth = ',reth

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

tunnelhill's curve fit to evaluate entropy

call tgas2(eins,rhoes,entrpy)

adiabatic wall enthalpy (kcal/g)

haw = he +0.5*sqrt(prw)*vele**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
else
    cf = .44 / reth
    qw = 0.22*rhostr*xmustr*rhoe*vele/(rhoe*xmue)*
    qw = 0.22*rhostr*xmustr*rhoe*velet/(rhoe*xmue)*
        (haw - hw)/(reth*prw**.6)
endif

else

```fortran
    cf = 2.*c1/(reth**rem)*(rhostr/rhoe)*(xmustr/xmue)**rem
    qw = c1*(rhostr/rhoe)*(xmustr/xmue)**rem*rhoe*vele*

endif

    qw = qw*4.18392e3

    if(qw .le. 0.) then
        tweq = 300.
        qw = 0.
    else
        tweq = (qw/(eps*sigma))**.25
    endif
    if(abs(tweq-twall(iter)) .gt. 5.) iweq = 1
    twall(iter) = tweq
    write(25,*) tweq

    qwbtu = qw*.88
    s = s + ds
    sum0 = sum
    f1 = f2
    velets = velet
    ite = 0
    wltemp = twall(iter)
    fss = xmue/xmustr
    rethss = fss*reth
    rethl = alog10(rethss)
    cfc = (te/tstar)/(17.08*rethl*rethl + 25.11*rethl + 6.012)
    write(24,9001) xs(iter),s,qw,qwbtu,cf
    write(34,9002) xs(iter),ys(iter),zs(iter),entrpy

continue
    if(irad .ne. 0) then
        if(iweq .ne. 0) then
            iweq = 0
            iwit = iwit + 1
            if(iwit .gt. 7) then
                write(6,*), ' Wall equilibrium temperature iteration failed'
                go to 9999
            endif
            rewind 25
            rewind 24
            rewind 34
            go to 99
        endif
    endif

9001 format(5(1x,el4.7))
9002 format(4(1x,el4.7))
9999 continue
return
end

SUBROUTINE TGAS2(E,R,S)

INPUTS FOR SUBROUTINE:
E=INTERNAL ENERGY IN (M/SEC)**2
R=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
IFLAG=-1
GO TO 80
20 SHIGH=S
Y=-4.5E00-2.5E-02
YLOW=Y
R=(10.**Y)*R0
IFLAG=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
IFLAG=-1
GO TO 80
50 SHIGH=S
Y=-0.5E00-0.5E-02
YLOW=Y
R=(10.**Y)*R0
IFLAG=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*Z
GO TO 120
110 DELT2=Z-0.4E00
DELT3=(2.5E00*DELT2-Y)*GASCON*2.302585E00
S=6779.204E00+DELT3
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,SX, 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
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), hmax(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.,
&
data ah / .128180el, .125380e2, .426138e2, .885088e1, .151569e2,
& .101759e2,
& .902850, .237222e2, .880011e2, -.333238e2, .196866e2,
& .446849e2,
& .653358, .431122e1, -.126229el, .209845e2, .268647e2, 0.,
& .363885, -.865884el, -.164319e2, -.207249e2, 0., 0.,
& .209284, -.171560e2, -.134978e2, -.564265e1, 0., 0.,
& .124937, -.120314e2, -.913636el, .639208e1, 0., 0.,
& -.755123e-2, -.117469e1, -.245329el, 0., 0., 0./
data bh / .121182e2, .720107e2, .123000e3, -.207380e2, -.713138e1,
& -.161956e2,
& .839944e1, .118014e3, .213329e3, -.316397e2, -.201771e2,
& -.141086e3,
& .596886e1, .267604e2, .113432e2, -.181381e2, -.104256e3,
& .0.,
& .329839e1, -.208034e2, -.285858, .633182e2, 0., 0.,
& .187458e1, -.416138e2, .801118e1, .262889e2, 0., 0.,
& .109286e1, -.229170e2, .113996e2, -.149544e2, 0., 0.,
& .164258e-1, -.592622e1, .371340e1, 0., 0., 0./
data ch/.424907e2, .148949e3, .121801e3, -.134604e2, -.172524, 
-.336892el, .289458e2, .214780e3, .181623e3, -.401000el, .635249el, 
.159412e3, .201689e2, .541203e2, .109117e2, -.399635, .145439e3, 
0., .110641e2, -.132700e2, .447878el, -.678713e2, 0., 0., 0., 
.622153e1, -.332532e2, .192371e1, -.396119e2, 0., 0., 0., 
.355163el, -.129249e2, -.259796el, .882252el, 0., 0., 0., 
.366590, -.214181el, -.288683, 0., 0., 0./

data dh/.665524e2, .133853e3, .509305e2, .166408el, .643645, 
.161274e2, .448640e2, .171168e3, .661367e2, .379639e1, -.174347, 
-.738595e2, .309518e2, .462077e2, .400303e1, .387388e1, -.846045e2, 
0., 0., 0., 0., 0., 0., 0./
data eh/.385195e2, .451550e2, .995964el, .356570e1, .356353e1, 
-.201068e1, .256452e2, .513939e2, .110476e2, .325469e1, .354258e1, 
.155141e2, .174843e2, .152182e2, .284253e1, .283981e1, .212051e2, 
0., 0., 0., 0., 0., 0., 0., 0., 0./

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

do 10 i = 1,7
  if(pinp .le. ptb(i)) then
    if(abs(pinp-ptb(i)) .le. 1.e-4) then

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*.0001)
   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
c   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)
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
   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)
  continue
  endif
if(intp .eq. 0) go to i00
find correct temperature range and curve to solve for temperature
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
  continue
write(6,*), ' Enthalpy is out of range', hinp
ifatal = 1
return
continue
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
  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(tl)) * (alog(pinp) - alog(ptb(ipml)))
& / (alog(ptb(ip)) - alog(ptb(ipml))) + alog(tl)
t = exp(alogt)

if(pinps .ne. 0.) pinp = pinps
return
end

subroutine newton(ti, hinp, t)
common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav,
& first, back, 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
  go to 10
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, back, 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),
ainu(4,7), bmu(4,7), cmu(4,7), chnu(4,7), emu(4,7),
ak(7,7), bk(7,7), ck(7,7), dk(7,7), ek(7,7),
apr(8,7), bpr(8,7), cpr(8,7), dpr(8,7), epr(8,7), fpr(8,7),
ptb(7), tcpmax(9,7), tzmax(9,7), tmumax(9,7),
tkmax(7,7), tprmax(8,7), ncpmax(7), nzmax(7),
nmumax(7), nkmax(7), nprmax(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 nkmax /7, 7, 7, 6, 6, 5, 4/
data nprmax /8, 8, 7, 7, 5, 5, 4/
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., 3750., 9250., 13750., 22500., 30000., 30001., 30001.,
& 1750., 3750., 6750., 10750., 17750., 30000., 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.,
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,
.199532, .345376e1, -.369572e2, -.146237e3, -.758521e3,
-.330240e2, -.618098e2, .125063e3, 0.,
.669436, -.453138e2, -.150135e3, .539167e3, .217707e2,
-.122810e3, .162348e3, 0., 0.,
.935059e2, -.473648e2, 0., 0.,
1.11751, 252675, .450386e2, .231376e2, -.799940e2,
.491698e2, -.253231e3, 0., 0.,
.986591e-1, .974261e-1, .210207e2, .143729e2, -.347606e2,
.450529e2, 0., 0., 0./
data bcp/
.344158e1, .129277e3, -.136508e3, -.515276e3, -.266973e4,
-.890568e2, .213355e2, .130713e4, -.326378e4,
.694347e1, .107177e3, -.129698e3, -.848597e3, -.900003e3,
-.489572e2, -.262275e2, .210795e3, 0.,
.127165e2, -.696036e3, -.835692e3, .106221e4, -.192634e2,
-.138486e3, .525270e3, 0., 0.,
.992221e1, -.779456e2, .101945e4, -.187832e3, -.478605e2,
.127166e2, -.407311e3, -.117775e4, 0.,
.552429e1, -.101598e3, .561712e3, -.893740e2, .217674e2,
.753693e2, .169726e2, 0., 0.,
.368846e1, .313529e2, .82425e3, -.271807e2, .187072e2,
.889977e2, -.132457e4, 0., 0.,
.323392e1, .573647e3, .778089e2, -.173603e2, .148249e1,
.161302e3, 0., 0., 0./
data ccp/
.126715e2, .411057e3, -.411567e3, -.882748e3, -.221802e4,
.890568e2, .213355e2, .130713e4, -.326378e4,
.694347e1, .107177e3, -.129698e3, -.848597e3, -.900003e3,
-.489572e2, -.262275e2, .210795e3, 0.,
.123531e2, -.696036e3, -.835692e3, .106221e4, -.192634e2,
-.138486e3, .525270e3, 0., 0.,
.992221e1, -.779456e2, .101945e4, -.187832e3, -.478605e2,
.127166e2, -.407311e3, -.117775e4, 0.,
.552429e1, -.101598e3, .561712e3, -.893740e2, .217674e2,
.753693e2, .169726e2, 0., 0.,
.368846e1, .313529e2, .82425e3, -.271807e2, .187072e2,
.889977e2, -.132457e4, 0., 0.,
.323392e1, .575473e3, .778089e2, -.173603e2, .148249e1,
.161302e3, 0., 0., 0./
data dcp/
.208154e2, .580300e3, -.525250e3, -.642505e3, -.791376e3,
-.182697e2, -.128857e3, -.422349e3, .176381e4,
.112521e2, .160586e3, -.169299e2, -.532403e3, -.238528e3,
-.182071e1, -.850086e1, -.295269e2, 0.,
.365225e2, -.730849e3, -.502696e3, .370582e3, .517928e1,
.225676e2, -.216688e3, 0., 0.,
.157475e2, -.627915e2, .757047e3, -.100614e3, .326629e1,
.595886e1, .141182e3, -.828838e3, 0.,
.879873e1, -.897230e2, .376565e3, -.403342e2, .314736e1,
-.936668e1, -.836769e2, 0., 0.,
.591907e1, .203259e2, .128924e3, -.102436e2, .350311e2,
-.242382e2, .798459e3, 0., 0.,
.519284e1, .896395e3, .381171e2, -.137585e1, -.510951e1,
.752038e2, 0., 0., 0./
data ecp/
.116592e2, .305728e3, -.241298e3, -.166628e3, -.102433e3,
.138792e1, .135712e1, .482128e2, -.348874e3,
.570825e1, .884544e2, .207647e2, -.119389e3, -.216169e2,
.229104e1, .253250e1, -.792067e1, 0.,
\begin{verbatim}
& .203928e2, -.281133e3, -.107793e3, .457650e2, .180195el,
& .100733e1, .277132e2, 0., 0.,
& .820277e1, -.154364e2, .205793e2, -.168003e2, .838365,
& .412806el, -.100733el, .277132e2, 0., 0.,
& .100733e1, .277132e2, 0., 0.,
& .244269el, .384233ei, .628850el, .743313, .877002,
& .710750, -.614415el, -.632086e2, -.467833e2, .556705e2,
& .820277e1, -.154364e2, .205793e2, -.168003e2, .838365,
& .412806el, -.100733el, .277132e2, 0., 0.,
& .100733e1, .277132e2, 0., 0.,
& .244269el, .384233ei, .628850el, .743313, .877002,

\textbf{data az} / 
& -.614415el, -.632086e2, -.467833e2, .556705e2,
& .820277e1, -.154364e2, .205793e2, -.168003e2, .838365,
& .412806el, -.100733el, .277132e2, 0., 0.,
& .100733e1, .277132e2, 0., 0.,
& .244269el, .384233ei, .628850el, .743313, .877002,

\textbf{data bz} / 
& -.107229el, .861656el, .370272e2, .139011e2, -.135009e2,
& .625098, -.460729el, .209649e2, -.162146e2,
& .434929, .324383el, .199010e2, -.134636e2,
& .311295, .485004, .137742e2, .888031el, -.309522e2,
& -.465274e-1, -.204396el, .630744e2, -.255245e2,

\textbf{data cz} / 
& -.125673el, -.370256e2, -.776456el, -.138693e3, .118386el,
& -.68867, .10954el, -.38276el, -.108251e1, .120156el,
& -.45440, -.123210el, -.334091el, -.804882, .210825el,
& -.30208, -.321087, -.203163el, -.620966, .165245el,
& .972123e-2, .404607el, -.827695, -.208573, .758996,

\textbf{data dz} / 
& .564944, .681308, .706848, .592861e-1, -.427210e-1,
& .286982, -.894328e-1, .327436, .415356e-1, -.375039e-1,
& .176448, .198816, .243749, .274096e-1, -.563525e-1,
& .104768, .632573e-1, .130377, .188157e-1, -.384201e-1,
& .417402e-2, -.344141, .466769e-1, .395382e-2, .194343e-1,
& .218303e-1, .639254e-1, .187431e-1, -.411847e-2, 0.,
& -.281715e-2, .324023e-2, -.525620e-2, 0., 0./

\textbf{data ez} / 
& -.823333e-1, -.443045e-1, -.233636e-1, -.903887e-3, .551468e-3,
& -.376722e1, .162238e-2, -.970559e-2, -.569596e-3, .424122e-3,
& -.212722e-1, -.110471e-1, -.644569e-2, -.334336e-3, .555405e-3,
& -.116924e-1, -.364522e-2, -.302863e-2, -.206237e-3, .330019e-3,
& -.536803e-3, .107287e-1, .941988e-3, -.175392e-4, -.182292e-3,
& -.179762e-2, -.255478e-2, -.320898e-3, .542207e-4, 0.,
& .168222e-3, -.18832e-4, .489857e-4, 0., 0./

\textbf{data amu} / 
& -.1160076e-4, -.9105422, .1463029e-1, -.2140374e-2,
& .2397194e-4, -.5784272, .1659118e-1, .6903314e-2,
& .5085043e-4, -.341487, .2450900e-1, -.356146e-1,
& .6394112e-4, -.2376368, .6309492e-3, -.1622687e1,
& .578188e-4, -.184423e8, .2606784e-1, 0.,
& .7256455e-4, -.9524274e-1, .5037513e-1, 0.,
& .7609039e-4, -.7868582e-1, 0., 0./

\textbf{data bmu} / 
\end{verbatim}
\[.6656010e-3, .4949794, -.5019958e-2, .6529285e-3,\]
\[.5564725e-3, .2816531, -.5019958e-2, .6529285e-3,\]
\[.4774840e-3, .4949794, -.5019958e-2, .6529285e-3,\]
\[.4385020e-3, .9006170e-1, .6108099e-3, .3035173,\]
\[.4438221e-3, .6040101e-1, -.4562535e-2, 0.,\]
\[.4050530e-3, .2589951e-1, -.8081647e-2, 0.,\]
\[.3891948e-3, .1820922e-1, 0., 0./\]
\[-.2933969e-3, -.1060568, .6886543e-3, -.7290226e-4,\]
\[-.1970968e-3, -.5377449e-1, .6106363e-3, .1061916e-3,\]
\[-.1322133e-3, -.2471167e-1, .7362709e-3, -.5837678e-3,\]
\[-.1024141e-3, -.1315352e-1, -.1286661e-3, -.2266401e-1,\]
\[-.1020840e-3, -.7566737e-2, .3111533e-3, 0.,\]
\[-.7626766e-4, -.2593217e-2, .5209350e-3, 0.,\]
\[-.6458779e-4, -.1543467e-2, 0., 0./\]
\[.7427050e-4, .1123425e-1, -.4723839e-4, .3865996e-5,\]
\[.4272210e-4, .5058384e-2, -.3715711e-4, -.4234384e-5,\]
\[.2362256e-4, .2030404e-2, -.4070960e-4, .2324839e-4,\]
\[.1654305e-4, .4609505e-3, -.1018512e-4, 0.,\]
\[.114437e-4, .1227975e-3, -.1682098e-4, 0.,\]
\[.8791566e-5, .6257766e-4, 0., 0./\]
\[-.6456605e-5, -.5897674e-3, .1623374e-5, -.9908122e-7,\]
\[-.2690853e-5, -.2352317e-3, .1135683e-5, .8514686e-7,\]
\[-.8014978e-6, -.8216415e-4, .1134307e-5, -.4590857e-6,\]
\[-.5014106e-6, -.3279129e-4, -.2960958e-6, .1570909e-4,\]
\[-.6822324e-6, -.1377229e-4, .1576999e-6, 0.,\]
\[-.5020411e-6, -.2772500e-5, .2731352e-6, 0.,\]
\[-.4216496e-6, -.1234998e-5, 0., 0./\]
\[.8752161e-7, .1229026e-4, -.2239581e-7, .9916638e-9,\]
\[-.3009241e-7, .4336410e-5, -.1397984e-7, -.6893227e-9,\]
\[-.6458338e-7, .1314540e-5, -.1276018e-7, .3600777e-8,\]
\[-.3710875e-7, .4329650e-6, .3444222e-8, .1166667e-6,\]
\[-.2239193e-9, .1623637e-6, -.9011456e-9, 0.,\]
\[-.7074486e-10, .2383398e-7, -.1794872e-8, 0.,\]
\[-.4509800e-8, .9579999e-8, 0., 0./\]
\[.395299e1, .119879e2, -.832682e2, -.103603e4, .261125e2,\]
\[.246095e2, -.571805e2,\]
\[.199665e1, -.831120e2, -.110139e3, .299875e3, .434485e2,\]
\[.895136e1, -.422029e2,\]
\[.198558e1, -.595826e2, -.442143e2, .584437e3, .373716e2,\]
\[.143675e2, .502985e1,\]
\[.105928e1, .101351e3, .830640e1, -.318301e3, .469099e2,\]
\[.154279e2, 0.,\]
\[.334316, .109992e2, .124072e2, -.189644e3, .298795e2,\]
\[.844897e1, 0.,\]
\[.413573, .821184e2, .113875e2, -.723261e2, -.382696e1,\]
\[0., 0.,\]
\[.208749, .378677e2, .223116e2, .792550e1, 0., 0., 0./\]
\[.386816e2, .412181e2, -.419438e3, -.242470e4, .411940e1,\]
\[-.507490e2, .168628e3,\]
\[.194822e2, -.560438e2, -.481050e3, .923042e3, .464790e1,\]
\[.322183e2, .144383e3,\]
\[.189164e2, .288748e3, -.206207e3, -.873106e3, -.115449e2,\]
\[.801073e1, -.960227e1,\]
\begin{verbatim}
& 0.100924e2, 0.490653e3, -0.324274e2, -0.306306e3, -0.330961e2, \\
& -0.541310e2, 0., \\
& 0.328202e1, 0.387106e2, -0.147438e2, -0.828711e2, -0.381078e2, \\
& -0.358117e2, 0., \\
& 0.328202e1, 0.387106e2, -0.147438e2, -0.828711e2, -0.381078e2, \\
& -0.358117e2, 0., \\
& 0.192122e1, 0.123284e3, 0.336369, -0.216552e2, 0., 0., 0./
\end{verbatim}

\begin{verbatim}
data ck /
& 0.140687e3, 0.711756e1, -0.751764e3, -0.206135e4, -0.186054e2, \\
& 0.369131e2, -0.181577e3, \\
& 0.706404e2, -0.140314e4, -0.757873e3, 0.992814e3, -0.155778e2, \\
& 0.350726e2, -0.182586e3, \\
& 0.668834e2, 0.500756e3, -0.324643e3, -0.445088e3, -0.113653e2, \\
& 0.146420e2, 0.196818e1, \\
& 0.356709e2, 0.868620e3, -0.942568e2, -0.146607e1, 0., 0., 0./
\end{verbatim}

\begin{verbatim}
data dk /
& 0.226110e3, -0.911924e2, -0.566912e3, -0.757541e3, -0.645054e1, \\
& -0.897288e1, 0.859388e2, \\
& 0.113538e3, -0.154128e4, -0.505860e3, 0.442621e3, -0.220224e1, \\
& -0.129798e2, 0.101698e3, \\
& 0.104546e3, 0.363789e3, -0.204871e3, -0.927269e2, 0.135799e1, \\
& -0.117248e2, 0.643952e1, \\
& 0.561818e2, 0.667923e3, -0.647282e2, -0.466313e1, 0.306898e1, \\
& -0.366810e2, 0., \\
& 0.200944e2, 0.548304e1, -0.299886e2, 0.227739e1, 0.122011e1, \\
& -0.353787e2, 0., \\
& 0.207305e2, 0.250668e3, -0.122339e2, -0.233991e2, 0., 0., 0./
\end{verbatim}

\begin{verbatim}
data ek /
& 0.127138e3, -0.810415e2, -0.157470e3, -0.108281e3, -0.621476e1, \\
& -0.623025e2, 0.213335e2, \\
& 0.599079e2, -0.632398e3, -0.125800e3, 0.634709e2, -0.558790e1, \\
& -0.498154e1, 0.270417e2, \\
& 0.527822e2, 0.844428e2, -0.494556e2, -0.128529e2, -0.542822e1, \\
& -0.389761e2, -0.896353e1, \\
& 0.249670e2, 0.180596e3, -0.180857e2, -0.585083e1, -0.562490e1, \\
& 0.115271e1, 0., \\
& 0.462882e1, -0.120166e2, -0.961485e1, -0.581069e1, -0.578171e1, \\
& 0.274595e1, 0., \\
& 0.427728e1, 0.475889e2, -0.610064e1, -0.556444e1, -0.717162e1, \\
& 0., 0., \\
& -0.127699e1, 0.684807e1, -0.498832e1, -0.485454e1, 0., 0., 0./
\end{verbatim}

\begin{verbatim}
data apr /
& 0.7695318, 0.4081926e2, -0.5060907e3, 0.3172744e2, -0.7624622e4, \\
& 0.1034825e3, -0.3414530e3, 0.1398841e2, \\
& 0.7483071, 0.4355608e2, 0.6877578e3, -0.7024989e3, -0.3107328e3, \\
& 0.9153227e2, 0.2304245e4, -0.1424413e2, \\
& 0.7406012, 0.3206825e2, 0.4409105e3, 0.2474049e2, -0.9639789e3, \\
& 0.9436717e2, -0.7123564e2, -0.7846718e2, \\
& 0.7548377, 0.6190687e2, -0.2641868e3, 0.7451996e2, 0.1034771e2, \\
& -0.1252168e3, -0.5022975e4, 0., \\
& 0.8037721, 0.9173717e2, 0.1874806e2, -0.1974440e3, 0.5713095e2, \\
& 0.161271e3, 0.1396134e3, 0., \\
& 0.8167207, -0.1857130e2, -0.4287169e2, 0.6447596e2, 0.7384318e2, \\
& 0., 0., 0./
\end{verbatim}
<table>
<thead>
<tr>
<th>Data bpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>-.2487156, -.6945482e2, .5765441e3, -.1923593e2, .3908646e4,</td>
</tr>
<tr>
<td>.0., 0., 0./</td>
</tr>
<tr>
<td>Data cpr</td>
</tr>
<tr>
<td>-.3371200e-2, .4642510e2, -.2594581e3, .6031312e1, -.7963915e3,</td>
</tr>
<tr>
<td>.5742599e1, -.1639670e2, .1254939,</td>
</tr>
<tr>
<td>-.4548542e-1, .4101816e2, .2059622e3, -.1153420e3, -.398423e2,</td>
</tr>
<tr>
<td>.4035104e1, .6397478e2, -.3586053,</td>
</tr>
<tr>
<td>-.618676, .2610712e2, .3082544e2, -.1345184e2, .9815976e2,</td>
</tr>
<tr>
<td>-.3657252e2, .786187e2, 0.,</td>
</tr>
<tr>
<td>-.1629922, .9087650e1, .1729215e2, -.1351549e2, .1123889e2,</td>
</tr>
<tr>
<td>0., 0., 0./</td>
</tr>
<tr>
<td>Data dpr</td>
</tr>
<tr>
<td>.5237761, -.1500926e2, .5768434e2, -.1191714e1, .8069517e2,</td>
</tr>
<tr>
<td>.4251452, .1109347e1, -.2782377e-2,</td>
</tr>
<tr>
<td>-.3843113, -.1204527e2, -.3502408e2, .1460894e2, -.4000552e1,</td>
</tr>
<tr>
<td>.2667702, .3358410e1, .1651461e-1,</td>
</tr>
<tr>
<td>-.1480977, -.6755087e1, -.1623127e2, .8375614, .5082427e1,</td>
</tr>
<tr>
<td>.1885223, .1451363, .4430242e-1,</td>
</tr>
<tr>
<td>-.1019887, -.1025306e2, .1198777e2, -.2590555, -.3418909e-1,</td>
</tr>
<tr>
<td>.1290359, .2463703e1, 0.,</td>
</tr>
<tr>
<td>-.6243393, -.2015462e2, -.1040290e1, .1841274e1, -.3686820,</td>
</tr>
<tr>
<td>-.1501584, .8742029e-1, 0.,</td>
</tr>
<tr>
<td>-.6989158, .3448930e1, .1132069e1, -.1705427, -.3633532e-1,</td>
</tr>
<tr>
<td>0., 0., 0.,</td>
</tr>
<tr>
<td>-.7111724, .9588901, .4761765, -.8122753e-1, .3771865e-1,</td>
</tr>
<tr>
<td>0., 0., 0./</td>
</tr>
<tr>
<td>Data epr</td>
</tr>
<tr>
<td>.4272376, .2350767e1, -.6333573e1, .1301576, -.4069244e1,</td>
</tr>
<tr>
<td>.1556552e-1, -.3701973e-1, .6736752e-5,</td>
</tr>
<tr>
<td>-.2847560, .1712608e1, .2943975e1, -.9148849, -.1930037,</td>
</tr>
<tr>
<td>.8878147e-2, .8795057e-1, -.3685522e-3,</td>
</tr>
<tr>
<td>-.1294933, .8752933, .1221500e1, -.6543669e-1, -.2000122,</td>
</tr>
<tr>
<td>.5488061e2, -.4016437e-2, -.8360113e-3,</td>
</tr>
<tr>
<td>-.1326718e-1, .1246034e1, -.9677752, .5423023e-2, -.5284942e-3,</td>
</tr>
<tr>
<td>-.2973274e-2, .4499464e-1, 0.,</td>
</tr>
<tr>
<td>.1903222, .2763213e1, .958832e-1, -.8709938e-1, .1391085e-1,</td>
</tr>
<tr>
<td>.3407133e-2, -.1716422e-2, 0.,</td>
</tr>
<tr>
<td>.2151091, -.4205156, -.7326105e-1, .5177594e-2, .6720911e-3,</td>
</tr>
<tr>
<td>0., 0., 0.,</td>
</tr>
<tr>
<td>-.2215290, -.1026406, -.2655841e-1, .2173037e-2, -.6355462e-3,</td>
</tr>
<tr>
<td>0., 0., 0./</td>
</tr>
</tbody>
</table>
c
data fpr /
& .9284843e-1, -.1430131, .2747920, -.5625333e-2, .8174707e-1,
& -.2291528e-3, .4881128e-3, .3740205e-6,
& .5734562e-1, -.9453467e-1, -.9787067e-1, .2261774e-1, .3624133e-2,
& -.1150000e-3, -.9193691e-3, .3216955e-1, -.9453467e-1, -.9787067e-1,.2261774e-1,.3624133e-2,
& -.6365539e-4, .4349632e-4, .6273942e-5,
& .541023e-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,.184057e-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
if(tinp < 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./((l.-runiv/airm/cp)
return
endif

c Find pressure range for interpolation
if(pinp < 1.e-4 .and. pinp > 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
  endif
10 continue
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 > tcpmax(i,ip)) go to 30
it = i
30 go to 40
continue

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

40 continue

$cpm2 = acp(it,ip) * xi**4 + bcp(it,ip) * xi**3 + ccp(it,ip) *$
& $xi**2 + dcp(it,ip) * xi + ecp(it,ip)$

if(intp .eq. 0 .or. ipml .le. 0) then
   cp = exp(cpm2)
   go to 65
endif

50 continue

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

$cinterpolation for specific heat$
c
alogcp = (cpm2 - cpml)/(alog(ptb(ip))-alog(ptb(ipml)))*
& (alog(pinp) - alog(ptb(ipml))) + cpml

$cp = exp(alogcp)$

$cCompressibility factor$
c
65 continue

xi = tinp*.001
70 do 90 i = 1,nzmax(ip)
   if(tinp .gt. tzmax(i,ip)) go to 70
   it = i
   go to 80
90 continue

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

$z2 = az(it,ip) + bz(it,ip) * xi + cz(it,ip) * xi**2$
& $+ dz(it,ip) * xi**3 + ez(it,ip) * xi**4$

if(intp .eq. 0 .or. ipml .le. 0) then
   z = z2
   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

c interpolation of compressibility factor
c

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

Viscosity
c

105 continue

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

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

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
c

interpolation of viscosity
c

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

Thermal conductivity
c

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
c

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) *
& \quad \text{xi}^{**2} + \text{dk}(\text{it}, \text{ip}) \times \text{xi} + \text{ek}(\text{it}, \text{ip})
\text{if(intp .eq. 0 .or. ipml .le. 0) then}
\quad \text{xk} = \exp(\text{xk2})
\text{go to 185}
\text{endif}

\text{do } 170 \text{ 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}
\quad \text{xk1} = \text{ak}(\text{it}, \text{ipml}) \times \text{xi}^{**4} + \text{bk}(\text{it}, \text{ipml}) \times \text{xi}^{**3} + \text{ck}(\text{it}, \text{ipml}) \times \text{xi}^{**2}
\& + \text{dk}(\text{it}, \text{ipml}) \times \text{xi} + \text{ek}(\text{it}, \text{ipml})
\text{c interpolation for thermal conductivity}
\text{c alogk} = (\text{xk2} - \text{xk1})/(\text{alog(ptb(ip))}-\text{alog(ptb(ipml))}) *
\& (\text{alog(pinp)} - \text{alog(ptb(ipml))}) + \text{xk1}
\text{\text{xk} = \exp(\text{alogk})}
\text{c Prandtl number}
\text{c 185 continue}
\quad \text{xi} = \text{tinp} * .001
\text{do } 190 \text{ i = 1,nprmax(ip)}
\text{if(tinp .gt. tprmax(i,ip)) go to 190}
\quad \text{it} = \text{i}
\quad \text{go to 200}

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

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

\text{do } 210 \text{ i = 1,nprmax(ipml)}
\text{if(tinp .gt. tprmax(i,ipml)) go to 210}
\text{it} = \text{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}
\quad \text{prl} = \text{apr}(\text{it}, \text{ipml}) + \text{bpr}(\text{it}, \text{ipml}) \times \text{xi} + \text{cpr}(\text{it}, \text{ipml}) \times \text{xi}^{**2}
\& + \text{dpr}(\text{it}, \text{ipml}) \times \text{xi}^{**3} + \text{epr}(\text{it}, \text{ipml}) \times \text{xi}^{**4} + \text{fpr}(\text{it}, \text{ipml}) \times \text{xi}^{**5}
\text{c interpolation of Prandtl number}
\text{c pr = (pr2 - prl)/(ptb(ip) - ptb(ipml)) * (pinp - ptb(ipml))}
\& + prl
300 continue
c compute density
c rho0 = pinp*airm0/(runiv*tinp)*2.4218e-2
if(z .lt. 1.) z = 1.  
rho = rho0/z
c compute molecular weight and effective gamma
c airm = airm0/z
gamma = 1./((1.-runiv/airm/cp)
c 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
c subroutine find(x0,y0,z0)
c Given a location (x0,y0,z0), and the grid cell containing this point, 
c this routine uses Newton’s method to find corresponding location 
c (xi,eta,zeta) in the master element.
c common /pos/ xi,eta,zeta
common /psidps/ psi(8), dpdx(8), dpdeta(8), dpdz(8)
common /connty/ il,jl,kl
common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
  & blank(151,40,110), blank,isubs(2),
  & jsubs(2),ksubs(2)
common /save/ isav,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)
c set initial guess at xi=0, eta=0 and zeta=0
c write(6,*) 'x0,y0,z0 = ',x0,y0,z0
xi = 0.
eta = 0.
zeta = 0.
ipl = il + 1
jpl = jl + 1
kpl = kl + 1
c x(1) = xyz(il,jl,kl,1)
y(1) = xyz(il,jl,kl,2)
z(1) = xyz(il,jl,kl,3)
c x(2) = xyz(ipl,jl,kl,1)
y(2) = xyz(ipl,jl,kl,2)
z(2) = xyz(ipl,jl,kl,3)
c x(3) = xyz(ipl,jl,kpl,1)
y(3) = xyz(ipl,jl,kpl,2)
z(3) = xyz(ipl,jl,kpl,3)
c x(4) = xyz(il,jl,kpl,1)
y(4) = xyz(il,jl,kpl,2)
z(4) = xyz(il,jl,kpl,3)
x(5) = xyz(il, jpl, kl, 1)
y(5) = xyz(il, jpl, kl, 2)
z(5) = xyz(il, jpl, kl, 3)

x(6) = xyz(ipl, jpl, kl, 1)
y(6) = xyz(ipl, jpl, kl, 2)
z(6) = xyz(ipl, jpl, kl, 3)

x(7) = xyz(ipl, jpl, kpl, 1)
y(7) = xyz(ipl, jpl, kpl, 2)
z(7) = xyz(ipl, jpl, kpl, 3)

x(8) = xyz(il, jpl, kpl, 1)
y(8) = xyz(il, jpl, kpl, 2)
z(8) = xyz(il, jpl, kpl, 3)

iter = 0
10 continue

compute shape function and its derivatives

call shape

initialize jacobian matrix

do 20 j = 1, 3
do 20 i=1,3
fj(i,j) = 0.
20 continue

r(1) = x0
r(2) = y0
r(3) = z0

get jacobian matrix

do 30 i = 1, 8
fj(1,1) = fj(1,1) + x(i)*dpdxi(i)
fj(2,1) = fj(2,1) + y(i)*dpdxi(i)
fj(3,1) = fj(3,1) + z(i)*dpdxi(i)
fj(1,2) = fj(1,2) + x(i)*dpdeta(i)
fj(2,2) = fj(2,2) + y(i)*dpdeta(i)
fj(3,2) = fj(3,2) + z(i)*dpdeta(i)
fj(1,3) = fj(1,3) + x(i)*dpdzeta(i)
fj(2,3) = fj(2,3) + y(i)*dpdzeta(i)
fj(3,3) = fj(3,3) + z(i)*dpdzeta(i)
r(1) = r(1) - x(i)*psi(i)
r(2) = r(2) - y(i)*psi(i)
r(3) = r(3) - z(i)*psi(i)
30 continue

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)

write(6,*) ' determinant of Jacobian = ',det

res = sqrt(r(1)*r(1) + r(2)*r(2) + r(3)*r(3))
if(res .lt. 1.0e-4) go to 9999

idm = 3

call gauss(fj,r,dr,idm)
if(ifatal .ne. 0) return

call gauss(fj,r,dr,idm)
write(6,*) dr
sdr = abs(dr(1)) + abs(dr(2)) + abs(dr(3))
xi = xi + dr(1)
et = eta + dr(2)
zeta = zeta + dr(3)
if(sdr .ge. 1.e-3) then
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
  c
  c This routine evaluate 3-D linear shape functions and their derivatives.
  c
  common /pos/ xi, eta, zeta
  common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzeta(8)
  c
  omxi = 1. - xi
  opxi = 1. + xi
  ometa = 1. - eta
  opeta = 1. + eta
  omzeta = 1. - zeta
  opzeta = 1. + zeta
  c
  omxiet = omxi * ometa
  opxime = opxi * ometa
  opxiet = opxi * opeta
  omxipe = omxi * opeta
  c
  psi(1) = .125 * omxiet * omzeta
  psi(2) = .125 * opxime * omzeta
  psi(3) = .125 * opxiet * omzeta
  psi(4) = .125 * omxipe * omzeta
  psi(5) = .125 * omxiet * opzeta
  psi(6) = .125 * opxime * opzeta
  psi(7) = .125 * opxiet * opzeta
  psi(8) = .125 * omxipe * opzeta
  c
  ometze = ometa * omzeta
  ometpz = ometa * opzeta
  opetze = opeta * opzeta
  opetmz = opeta * omzeta
  omxize = omxi * omzeta
  omxipz = omxi * opzeta
  opxize = opxi * opzeta
  opximz = opxi * omzeta
  c
  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)
  c
  dpdeta(1) = -.125 * omxize
dpdeta(2) = -.125 * opximz
dpdeta(3) = -dpdeta(2)
dpdeta(4) = -dpdeta(1)
dpdeta(5) = -.125 * omxipz
dpdeta(6) = -.125 * opxije
dpdeta(7) = -dpdeta(6)
dpdeta(8) = -dpdeta(5)

c

dpdztzta(1) = -.125 * omxiet
dpdztzta(2) = -.125 * opxime
dpdztzta(3) = -.125 * omxiet
dpdztzta(4) = -dpdztzta(1)
dpdztzta(5) = -dpdztzta(2)
dpdztzta(7) = -dpdztzta(3)
dpdztzta(8) = -dpdztzta(4)

c

return
end

SUBROUTINE GAUSS(FJAC,C,X,IDIM)
common /save/ igsav, issav, isav, jsav, ksav, asav, bsav, gsav, & first, back, second, ifatal
C
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
C FORWARD ELIMINATION
C
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
C BACK SUBSTITUTION
C
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
C
9006 FORMAT (’ *** ZERO PIVOT, I = ’,I3)
RETURN
END

************************************************************************
SUBROUTINE CELL3 (IS,IE,JS,JE,KS,KE,
************************************************************************
C
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, J1, K, 3)
Z5 = XYZ(I, J, K1, 3)
Z6 = XYZ(I1, J, K1, 3)
Z7 = XYZ(I, 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 + XBA*A + XB*G + XABG*(A*G)
AMAT(2,2) = YB + YBA*A + YBG*G + YABG*(A*G)
AMAT(3,2) = ZB + ZBA*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.

```plaintext
AS = A
BS = B
GS = G
IF (A.EQ.0.) A = .01
IF (A.EQ.1.) A = .99
IF (B.EQ.0.) B = .01
IF (B.EQ.1.) B = .99
IF (G.EQ.0.) G = .01
IF (G.EQ.1.) G = .99
IF (A.NE.AS .OR. B.NE.BS .OR. G.NE.GS) THEN
    GOTO 30

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

ELSE
    A = DI+.5
    B = DJ+.5
    G = DK+.5
    GOTO 40
ENDIF
ENDIF
```

```plaintext
DX = X-XH
DY = Y-YH
DZ = Z-ZH
DA = DX*BMAT(1,1) + DY*BMAT(1,2) + DZ*BMAT(1,3)
DB = DX*BMAT(2,1) + DY*BMAT(2,2) + DZ*BMAT(2,3)
DG = DX*BMAT(3,1) + DY*BMAT(3,2) + DZ*BMAT(3,3)
A = A+DA
B = B+DB
G = G+DG
```

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

```plaintext
IF (ABS(A-.5).GT.3..OR. ABS(B-.5).GT.3..OR. ABS(G-.5).GT.3.) THEN
    GOTO 40
```

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

```plaintext
ELSE
    ERR2= DA*DA + DB*DB + DG*DG
    IF (ERR2/3.LE.1.E-4) GOTO 40
ENDIF
IF (ITER.LT.MITER) GOTO 30
```

40 CONTINUE

The point is in this cell.

```plaintext
IF (ABS(A-.5).LE..50005 .AND. ABS(B-.5).LE..50005 .AND. ABS(G-.5).LE..50005) THEN
    IF (DIALOG) WRITE(6,*) 'MATCH'
```

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

```plaintext
ELSE IF (NSTEP.GT.MSTEP) THEN
    STATUS= 1
    IF (DIALOG) WRITE(6,*) 'MORE THAN ',MSTEP,' STEPS'
```
Update our (i,j,k) guess, keeping it inbounds.

ELSE
    IN = I
    JN = J
    KN = K
    IF (A.LT.0.) IN = MAX(IN-1,IS)
    IF (A.GT.1.) IN = MIN(IN+1,IE-1)
    IF (B.LT.0.) JN = MAX(JN-1,JS)
    IF (B.GT.1.) JN = MIN(JN+1,JE-1)
    IF (G.LT.0.) KN = MAX(KN-1,KS)
    IF (G.GT.1.) KN = MIN(KN+1,KE-1)
    IF (g.LT.0.) JN = MAX(JN-1,JS)
    IF (g.GT.1.) JN = MIN(JN+1,JE-1)
    IF (b.LT.0.) KN = min(KN+1,KS)
    IF (b.GT.1.) KN = max(KN-1,KE-1)
    IF (DIALOG) WRITE(*,*) 'TRY CELL INDEX ',IN, JN, KN
    Check IBLANK for this cell.
    IF (BLANK) THEN
        IN1 = IN+1
        JN1 = JN+1
        KN1 = KN+1
        IF (IBLANK(IN, JN, KN) .EQ. 0 .OR. IBLANK(IN1, JN1, KN1) .EQ. 0)
            STATUS= 1
        IF (DIALOG) WRITE(6,*) 'EXTRAPOLATE'
        GOTO 50
    ENDIF
ENDIF
ELSE
    Not repeating a previous point. Use the new (i,j,k) and try again.
    IF (
        (IN.NE.ISAV1 .OR. JN.NE.JSAV1 .OR. KN.NE.KSAV1)
        .AND. (IN.NE.ISAV2 .OR. JN.NE.JSAV2 .OR. KN.NE.KSAV2)
    ) THEN
        ISAV2 = ISAV1
        JSAV2 = JSAV1
        KSAV2 = KSAV1
        ISAV1 = IN
        JSAV1 = JN
        KSAV1 = KN
        DI = ISIGN(1,IN-I)
        DJ = ISIGN(1,JN-J)
        DK = ISIGN(1,KN-K)
        I = IN
        J = JN
        K = KN
        GOTO 20
    We've been here before...
ELSE
    It seems to be outside the domain. We would have to extrapolate to find it.
    STATUS= 1
    IF (DIALOG) WRITE(6,*) 'EXTRAPOLATE'
ENDIF
ENDIF
50 CONTINUE
*** End of Subroutine Cell3

SUBROUTINE CLOSE3(ISRCH, IGRID, NSUBS, I, J, K, A, B, G,
X, Y, Z, ISUB, STATUS)

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 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

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 = ISAV
J = JSAV
K = KSAV
A = ASAV
B = BSAV
G = GSAV
is = isubs(1)
je = jsubs(2)
js = jsubs(1)
ke = ksubs(2)
k = ksubs(1)
CALL PSRCH3(is,je, 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
C that the calling program is responsible for maintaining D2MIN if the
C IPTS list is to be added to. Don't overflow IPTS, but don't worry
C 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 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
         ENDIF
      END
   CONTINUE
10 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)
  IF (JS .NE. JE) THEN
    CALL DMIN3(IS, IE, JE, JE, KE, KE, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
  ENDIF
ENDIF
CALL DMIN3(IS, IS, JS + I, JE - 1, KS, KS, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
IF (IS .NE. IE) THEN
  CALL DMIN3(IE, IE, JS + I, 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)
  IF (IS .NE. IE) THEN
    CALL DMIN3(IE, IE, JE, JE, KS + I, KE - I, X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
  ENDIF
ENDIF

CALL DMIN3(IS, IS, JS, JS, KS + 1, KE - 1,
  X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
IF (IS .NE. IE) THEN
  CALL DMIN3(IE, IE, JS, JS, KS + 1, KE - 1,
  X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
ENDIF
IF (JS .NE. JE) THEN
  CALL DMIN3(IS, IS, JE, JE, KS + 1, KE - 1,
  X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
  IF (IS .NE. IE) THEN
    CALL DMIN3(IE, IE, JE, JE, KS + 1, KE - 1,
    X, Y, Z, MPTS, NPTS, IPTS, D2MIN)
  ENDIF
ENDIF

Search from each of these points.

DO 10 II = 1, NPTS
  ISUB = II; SUB
  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
Failed all searches.

C

STATUS = 1

C

30 CONTINUE

RETURN

END

*** End of Subroutine ESrch3

***************************************************************
SUBROUTINE FSrch3(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

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,KS,

& X,Y,Z,MPTS,NPTS,IPTS,D2MIN)

IF (KS.NE.KE) THEN

CALL DMIN3(IS,IE,JS,JE,KE,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,JE,KS+1,KE-1,

& X,Y,Z,MPTS,NPTS,IPTS,D2MIN)

ENDIF

CALL DMIN3(IS,IS,JE+1,JE-1,NPTS,IPTS,D2MIN)

IF (IS.NE.IE) THEN

CALL DMIN3(IE,IE,JE+1,JE-1,NPTS,IPTS,D2MIN)

ENDIF

ENDIF

C

Search from each of these points.
**End of Subroutine FSrch3**
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
10 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.
   CALL PSRCH3(IS, IE, JS, JE, KS, KE,
   IF (ISTAT.EQ.0) GOTO 30
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.
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(2,2)*A(3,3) - A(3,2)*A(2,3)
AINV(1,2) = A(3,2)*A(1,3) - A(1,2)*A(3,3)
AINV(1,3) = A(1,2)*A(2,3) - A(2,2)*A(1,3)

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

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

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(2,1) = AINV(2,1)*DET
  AINV(3,1) = AINV(3,1)*DET
  AINV(1,2) = AINV(1,2)*DET
  AINV(2,2) = AINV(2,2)*DET
  AINV(3,2) = AINV(3,2)*DET
  AINV(1,3) = AINV(1,3)*DET
  AINV(2,3) = AINV(2,3)*DET
  AINV(3,3) = AINV(3,3)*DET
  ELSE
  CALL COPY(9,A,TMP)
  CALL SSVDC(TMP,3,3,3,S,E,U,3,V,3,WORK,11,INFO)
  IF (S(1) .EQ. 0.) THEN
    STATUS = 3
    CALL ZERO(9,AINV)
    GOTO 10
  ENDIF
ENDIF

Compute V S U.

S(1) = 1./S(1)
IF (S(3)*S(1) .LT. 1.E-5) THEN
  STATUS = 1
  S(3) = 0.
ELSE
  S(3) = 1./S(3)
ENDIF
IF (S(2)*S(1) .LT. 1.E-5) THEN
  STATUS = 2
  S(2) = 0.
ELSE
  S(2) = 1./S(2)
ENDIF

Start out assuming S is a diagonal matrix.
SIU(1,1) = S(1)*U(1,1)
SIU(2,1) = S(2)*U(2,1)
SIU(3,1) = S(3)*U(3,1)

SIU(1,2) = S(1)*U(1,2)
SIU(2,2) = S(2)*U(2,2)
SIU(3,2) = S(3)*U(3,2)

SIU(1,3) = S(1)*U(1,3)
SIU(2,3) = S(2)*U(2,3)
SIU(3,3) = S(3)*U(3,3)

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(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)
ENDIF

Finish up A = V S U.

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)
AINV(3,1) = V(3,1)*SIU(1,1) + V(3,2)*SIU(2,1) + V(3,3)*SIU(3,1)

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)
AINV(3,2) = V(3,1)*SIU(1,2) + V(3,2)*SIU(2,2) + V(3,3)*SIU(3,2)

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)
AINV(3,3) = V(3,1)*SIU(1,3) + V(3,2)*SIU(2,3) + V(3,3)*SIU(3,3)

10 CONTINUE
RETURN

*** End of Subroutine Inv3X3

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

Search from closest point coming from grid N.

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.

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 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+1
                    IPTS(1,NPTS)= II
                    IPTS(2,NPTS)= JJ
                    IPTS(3,NPTS)= KK
                ENDIF
            ENDIF
        END
    END
10 CONTINUE

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

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

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

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

C
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
C DIMENSION FROM(*), TO(*)
C
C DO 10 I = 1, LEN
C TO(I) = FROM(I)
10 CONTINUE
RETURN
END
*** End of Subroutine Copy
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+1,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.
real(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.

real(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 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,11,1ls,lml,lp1,ls,lu,m,maxit,*
        mm,mml,mpl,nct,nctpl,ncu,nrt,nrtpl
real sdot,t,r
real b,c,cs,el,emml,f,g,snrm2,snrml,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 sscl̂al(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)
   20
c
   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)
   30
c
   place the l-th row of x into e for the
   subsequent calculation of the row transformation.
      e(j) = x(l,j)
   40
c
   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)
      60
c
   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 sscl̂al(p-l,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(l) = 0.0e0
      90      continue
      do 100 j = lp1, p
         call saxpy(n-l,e(j),x(lp1,j),1,work(lp1),1)
      100      continue
do 110 j = lpl, p
   call saxpy(n-1,-e(j)/e(lpl),work(lpl),1,x(lpl,j),1)
110  continue
120  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
200  continue
   if (nct .lt. 1) go to 290
   do 280 ll = 1, nct
      l = nct - ll + 1
      if (s(l) .eq. 0.0e0) go to 250
         lp1 = l + 1
      if (ncu .lt. lp1) go to 220
         do 210 j = lp1, ncu
            t = -sdot(n-l+l,u(i,1),1,u(i,j),1)/u(i,i)
call saxpy(n-l+1,t,u(1,1),1,u(l,j),1)
210  continue
220  continue
call ssca1(n-l+1,-1.0e0,u(1,1),1)
u(1,1) = 1.0e0 + u(1,1)
lml = l - 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(l,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 i = 1, p
        l = p - i + 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 l.lt.m
kase = 2  if s(l) is negligible and l.lt.m
kase = 3  if e(l-1) is negligible, l.lt.m, and
          s(l), ..., s(m) are not negligible (qr step).
kase = 4  if e(m-1) is negligible (convergence).

do 390 i = 1, m
    l = m - i
...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
......exit
    go to 400
380 continue
390 continue
400 continue
if (l .ne. m - 1) go to 410
```
kase = 4
410  go to 480
    continue
    lpl = l + 1
    mpl = m + 1
    do 430 1ls = lpl, mpl
        ls = m - 1ls + lpl
        ...exit
        if (ls .eq. 1) 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
    if (ls .ne. 1) go to 450
    kase = 3
    go to 470
430  continue
    if (ls .ne. 1) go to 460
    kase = 1
    go to 470
440  continue
        kase = 2
        l = ls
450  continue
        kase = 2
        l = ls
460  continue
        l = l + 1
    c
    perform the task indicated by kase.
    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. 1) go to 500
        f = -sn*e(k-1)
        e(k-1) = cs*e(k-1)
        continue
        if (wantv) call srot(p,v(1,k),1,v(1,m),1,cs,sn)
500  continue
    go to 610
    c
    split at negligible s(1).
    c
520  continue
    f = e(1-1)
    e(1-1) = 0.0e0
    do 530 k = l, m
        t1 = s(k)
        call srotg(t1,f,cs,sn)
        s(k) = t1
```
\[ f = -\sin(e(k)) \]
\[ e(k) = \cos(e(k)) \]

530 if \( \text{wantu} \) call srot \((n,u(1,k)),l,u(1,l-1),1,cs,sn) continue

540 go to 610

540 perform one qr step.

540 continue

540 calculate the shift.

\[
\text{scale} = \text{amax1}(\text{abs}(s(m)),\text{abs}(s(m-1)),\text{abs}(e(m-1)),\text{abs}(s(1))),\text{abs}(e(1)))
\]
\[
\text{sm} = \frac{s(m)}{\text{scale}}
\]
\[
\text{smml} = \frac{s(m-1)}{\text{scale}}
\]
\[
\text{emml} = \frac{e(m-1)}{\text{scale}}
\]
\[
\text{sl} = \frac{s(1)}{\text{scale}}
\]
\[
\text{el} = \frac{e(1)}{\text{scale}}
\]
\[
b = \frac{(\text{smml} + \text{sm})*(\text{smml} - \text{sm}) + \text{emml}**2}{2.0e0}
\]
\[
c = (\text{sm} \cdot \text{emml})**2
\]
\[
\text{shift} = 0.0e0
\]

550 if \( b \cdot \text{eq.} \ 0.0e0 \cdot \text{and.} \ c \cdot \text{eq.} \ 0.0e0 \) go to 550

\[
\text{shift} = \sqrt{b**2+c}
\]

550 shift = \( -\text{shift} \)

550 shift = \( c/(b + \text{shift}) \)

550 continue

550 \( f = (\text{sl} + \text{sm})*(\text{sl} - \text{sm}) + \text{shift} \)

550 \( g = \text{sl} \cdot \text{el} \)

550 continue

550 go to 610

550 convergence.

570 continue

570 make the singular value positive.

570 if \( s(1) \cdot \geq \ 0.0e0 \) go to 580

570 \( s(1) = -s(1) \)

570 if \( \text{wantv} \) call sscal \((p,-1.0e0,v(1,1)),l)

580 continue

580 order the singular value.
**Routine SSVDC**

```fortran
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 ssowap(p,v(l,l),1,v(l,l+1),1)
   if (wantu .and. 1 .lt. n)
     * call ssowap(n,u(l,l),1,u(l,l+1),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 ssowap (n,sx,incy,un,incy)**

```
c c interchanges two vectors.
c c uses unrolled loops for increments equal to 1.
c c jack dongarra, linpack, 3/11/78.
c c
real sx(l),sy(l),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
   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
   code for both increments equal to 1
   clean-up loop
   20 m = mod(n,3)
   if( m .eq. 0 ) go to 40
   do 30 i = 1,m
     stemp = sx(i)
     sx(i) = sy(i)
30 continue
```


```plaintext
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,incy,iy,ix,m,ml,n
c
if (n .le. 0) return
if (sa .eq. 0.0) return
if(incy.eq.1.and.incy.eq.1)go to 20
c 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
        sy(iy) = sy(iy) + sa*sx(ix)
        ix = ix + incx
        iy = iy + incy
    10 continue
return

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
end
```

---

This text contains a subroutine for performing a vector operation where a constant times a vector plus another vector is computed. The subroutine `saxpy` is designed to efficiently handle vector operations with unrolled loops for increments equal to one. It includes checks for handling different cases of increments, including both equal and unequal increments, and provides clean-up logic for the final iterations of the vector operations.
sy(i + 2) = sy(i + 2) + sa*sx(i + 2)  
sy(i + 3) = sy(i + 3) + sa*sx(i + 3)  
50 continue  
return  
end  
*** End of Subroutine SaxPy

*********************************************************************  
real function sdot(n,sx,incy,SY,incy)  
*********************************************************************  
  c  
  c  forms the dot product of two vectors.  
  c  uses unrolled loops for increments equal to one.  
  c  jack dongarra, linpack, 3/11/78.  
  c
  real sx(1), sy(1), stemp  
  integer i, incx, incy, ix, iy, m, mlp, n  
  
stemp = 0.0e0  
sdot = 0.0e0  
if(n.le.0) return  
if(incx.eq.1.and.incy.eq.1) go to 20
  c  
  c  code for unequal increments or equal increments  
  c 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)  
iy = iy + incy  
iy = iy + incy
  10 continue  
  sdot = stemp  
return
  c  
  c  code for both increments equal to 1
  c
  c
  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
  mlp = m + 1
  do 50 i = mlp, n, 5
     stemp = stemp + sx(i)*sy(i) + sx(i+1)*sy(i+1) +  
     * sx(i+2)*sy(i+2) + sx(i+3)*sy(i+3) + sx(i+4)*sy(i+4)  
  50 continue
  60 sdot = stemp
  return
  end
  *** End of Function Sdot

*********************************************************************
real function snrm2 ( n, sx, incx)
************************************************************************
intrinsic 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

two 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)
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) = 4.4089e-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) = 8.232d-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 /

if(n .gt. 0) go to 10
snrm2 = zero
  go to 300

10 assign 30 to next
snrm2 = zero
nn = n * incx

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

phase 1. sum is zero

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

assign 70 to next

begin main loop

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.

300 continue
    snrm2 = xmax * sqrt(sum)
    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), temp, 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

code 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

code 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 sroth(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
*** End of Subroutine SRotG

************************************************************************
SUBROUTINE ZERO(LEN,ARRAY)
************************************************************************
C
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
*** End of Subroutine Zero

******************************************************************************
SUBROUTINE TRIM(STRING,LSTRIN)
******************************************************************************
C C Return the length of STRING after trailing blanks, nulls, and tabs C C have been removed.
C C CHARACTER(*) STRING
C CHARACTER NULL,TAB
C C Initialize the null and tab characters.
C C NULL= CHAR(0)
TAB = CHAR(9)
C C Loop backwards through the character string and find the last C C nonblank, nonnull character.
C LSTRIN= LEN(STRING)
DO 10 L= LSTRIN,1,-1
  IF (STRING(L:L).NE.' ').AND. STRING(L:L).NE.NULL C C .AND. STRING(L:L).NE.TAB) THEN
    LSTRIN= L
    GOTO 20
10 CONTINUE
ALL blank or null or tabs!
LSTRIN= 0
20 CONTINUE
RETURN
END
*** End of Subroutine Trim

******************************************************************************
SUBROUTINE UPCA(: STRING)
******************************************************************************
C C Convert this character string to upper case.
C C CHARACTER(*) STRING
CHARACTER*26 LOWER,UPPER
C DATA LOWER/'abcdefghijklmnopqrstuvwxyz'/
C UPPER/'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/
C C Don’t worry about the trailing blanks -
C CALL TRIM(STRING,LSTRIN)
C C Look for lower case letters and substitute upper case ones.
C 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