NASA Contractor Report 188343

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

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

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

National Aeronautics and
Space Administration

1994

Contents

List of Figures

Nomenclature

$c_1, \ldots, c_5$	Defined by Equations (108-112)
$\vec{e_\tau}, \vec{e_\xi}, \vec{e_\eta}$	Unit vector in streamline coordinates
F	Function describing body surface
$\vec{\nabla} F$	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	Degined by Equation 107
N	Exponent of the power law velocity profile relation
p	Pressure
Pr_w	Wall Prandtl number
p_{stag}	Stagnation pressure
$\dot{q}_L$	Laminar heating rate
$\dot{q}_T$	Turbulent heating rate
q_w	Surface heating rate
Re_θ	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
α, β	Coordinates of computational domain
θ	Momentum thickness
δ	Boundary layer thickness
μ^*	Viscosity defined at reference enthalpy
μ_e	Edge viscosity
ρ^*	Density defined by reference enthalpy
ρ_e	Edge density
τ, ξ, η	Streamline coordinates (see Fig. 1)
$\hat{\Psi}_j$	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 runing 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

$$\frac{dx}{dt} = u \tag{1}$$

$$\frac{dy}{dt} = v \tag{2}$$

$$\frac{dz}{dt} = w \tag{3}$$

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

5 Metric Coefficient

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

$$\vec{r} = x\vec{i} + y\vec{j} + z\vec{k} \tag{4}$$

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

$$\vec{r} = \xi\vec{e}_\xi + \eta\vec{e}_\eta \tag{5}$$

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

$$\begin{aligned} \vec{e}_\eta &= \frac{\frac{\partial \vec{r}}{\partial \eta}}{\left|\frac{\partial \vec{r}}{\partial \eta}\right|} \\ &= \frac{\frac{\partial x}{\partial \eta}\vec{i} + \frac{\partial y}{\partial \eta}\vec{j} + \frac{\partial z}{\partial \eta}\vec{k}}{\left|\frac{\partial \vec{r}}{\partial \eta}\right|} \end{aligned} \tag{6}$$

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

$$\vec{e}_\eta = \vec{e}_\tau \times \vec{e}_\xi \tag{7}$$

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

$$\begin{aligned} \vec{V} &= V\vec{e}_\xi \\ &= u\vec{i} + v\vec{j} + w\vec{k} \end{aligned} \tag{8}$$

where u, v, w are Cartesian surface velocity components. Therefore

$$\vec{e}_\xi = \frac{u}{V}\vec{i} + \frac{v}{V}\vec{j} + \frac{w}{V}\vec{k} \tag{9}$$

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

$$F(x,y,z) = x - x(y,z) = 0 \tag{10}$$

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

$$\begin{aligned} \vec{e}_\tau &= \frac{\vec{\nabla}F}{\left|\vec{\nabla}F\right|} \\ &= \tau_x\vec{i} + \tau_y\vec{j} + \tau_z\vec{k} \end{aligned} \tag{11}$$

where

$$\vec{\nabla}F = \vec{i} - \frac{\partial x}{\partial y}\vec{j} - \frac{\partial x}{\partial z}\vec{k} \tag{12}$$

and

$$\left|\vec{\nabla}F\right| = \left[1 + \left(\frac{\partial x}{\partial y}\right)^2 + \left(\frac{\partial x}{\partial z}\right)^2\right]^{\frac{1}{2}} \tag{13}$$

Equation (7) can now be written as

$$\begin{aligned} \vec{e}_\eta &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \tau_x & \tau_y & \tau_z \\ \frac{u}{V} & \frac{v}{V} & \frac{w}{V} \end{vmatrix} \\ &= \frac{\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & -\frac{\partial x}{\partial y} & -\frac{\partial x}{\partial z} \\ \frac{u}{V} & \frac{v}{V} & \frac{w}{V} \end{vmatrix}}{\left|\vec{\nabla}F\right|} \\ &= \frac{\left(\frac{v}{V}\frac{\partial x}{\partial z} - \frac{w}{V}\frac{\partial x}{\partial y}\right)\vec{i} - \left(\frac{w}{V} + \frac{u}{V}\frac{\partial x}{\partial z}\right)\vec{j} + \left(\frac{v}{V} + \frac{u}{V}\frac{\partial x}{\partial y}\right)\vec{k}}{\left|\vec{\nabla}F\right|} \end{aligned} \tag{14}$$

From Equations (6) and (14) we have

$$\frac{\frac{\partial y}{\partial \eta}}{\left|\frac{\partial \vec{r}}{\partial \eta}\right|} = -\frac{1}{\left|\vec{\nabla}F\right|}\left(\frac{w}{V} + \frac{u}{V}\frac{\partial x}{\partial z}\right) \tag{15}$$

and

$$\frac{\frac{\partial z}{\partial \eta}}{\left|\frac{\partial \vec{r}}{\partial \eta}\right|} = \frac{1}{\left|\vec{\nabla} F\right|}\left(\frac{v}{V} + \frac{u}{V}\frac{\partial x}{\partial y}\right) \tag{16}$$

Let $h = \left|\frac{\partial \vec{r}}{\partial \eta}\right|$ = metric coefficient, then

$$\frac{\partial y}{\partial \eta} = -\frac{h}{\left|\vec{\nabla} F\right|}\left(\frac{w}{V} + \frac{u}{V}\frac{\partial x}{\partial z}\right) \tag{17}$$

and

$$\frac{\partial z}{\partial \eta} = \frac{h}{\left|\vec{\nabla} F\right|}\left(\frac{v}{V} + \frac{u}{V}\frac{\partial x}{\partial y}\right) \tag{18}$$

Combining Equations (17) and (18), we have

$$h = \frac{\left|\vec{\nabla} F\right|}{V}\left(v\frac{\partial z}{\partial \eta} - w\frac{\partial y}{\partial \eta}\right) \tag{19}$$

Equation (19) can now be used to compute the metric coefficient, h, along the streamline. From DeJarnette[18], on the body surface

$$\frac{d}{dt}\left(\frac{\partial y}{\partial \eta}\right) = \frac{\partial}{\partial \eta}\left(\frac{dy}{dt}\right) = \frac{\partial}{\partial \eta}(v) = \frac{\partial v}{\partial y}\frac{\partial y}{\partial \eta} + \frac{\partial v}{\partial z}\frac{\partial z}{\partial \eta} \tag{20}$$

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

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

6 Computation of Surface Streamlines and Metrics

A fourth order variable step Runge-Kutta integrator was used to integrate equations (1) through (3), (20) and (21). In order to integrate equations (20) and (21) we need to know the values of $\frac{\partial v}{\partial y}$, $\frac{\partial v}{\partial z}$, $\frac{\partial w}{\partial y}$, and $\frac{\partial w}{\partial z}$ at each integration step. To compute $\left|\vec{\nabla} F\right|$, we need the values of $\frac{\partial x}{\partial y}$ and $\frac{\partial x}{\partial z}$ also at each integration step. All of these values can be computed numerically from the coordinates and surface velocity components at each grid point.

Consider a master element $\hat{\Omega}$ and a mapping to an element Ω_e in the physical domain, as shown in Figure 2. Figure 3 shows the mapping from a master element to the entire physical domain. By using the finite element shape functions, the mapping from the (α, β) domain to the physical (x, y, z) domain can be described as

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

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

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

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

$$\hat{\Psi}_1 = \alpha\beta \tag{25}$$

$$\hat{\Psi}_2 = -(1+\alpha)\beta \tag{26}$$

$$\hat{\Psi}_5 = (1+\alpha)(1+\beta) \tag{27}$$

$$\hat{\Psi}_4 = -(1+\beta)\alpha \tag{28}$$

Element II:

$$\hat{\Psi}_2 = -(1-\alpha)\beta \tag{29}$$

$$\hat{\Psi}_3 = -\alpha\beta \tag{30}$$

$$\hat{\Psi}_6 = \alpha(1+\beta) \tag{31}$$

$$\hat{\Psi}_5 = (1-\alpha)(1+\beta) \tag{32}$$

Element III:

$$\hat{\Psi}_5 = (1-\alpha)(1-\beta) \tag{33}$$

$$\hat{\Psi}_6 = \alpha(1-\beta) \tag{34}$$

$$\hat{\Psi}_9 = \alpha\beta \tag{35}$$

$$\hat{\Psi}_8 = (1-\alpha)\beta \tag{36}$$

Element IV:

$$\hat{\Psi}_4 = -\alpha(1-\beta) \quad (37)$$
$$\hat{\Psi}_5 = (1+\alpha)(1-\beta) \quad (38)$$
$$\hat{\Psi}_8 = (1+\alpha)\beta \quad (39)$$
$$\hat{\Psi}_7 = -\alpha\beta \quad (40)$$

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

$$\begin{aligned} x &= x_1\hat{\Psi}_1 + x_2\hat{\Psi}_2 + x_5\hat{\Psi}_5 + x_4\hat{\Psi}_4 \\ &= x_1\alpha\beta - x_2(1+\alpha)\beta + x_5(1+\alpha)(1+\beta) - x_4(1+\beta)\alpha \end{aligned} \quad (41)$$

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

$$\frac{\partial x}{\partial \alpha} = x_1\beta - x_2\beta + x_5(1+\beta) - x_4(1+\beta) \quad (42)$$
$$\frac{\partial x}{\partial \beta} = x_1\alpha - x_2(1+\alpha) + x_5(1+\alpha) - x_4\alpha \quad (43)$$

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

$$\frac{\partial x}{\partial \alpha}|_5 = x_5 - x_4 \quad (44)$$
$$\frac{\partial x}{\partial \beta}|_5 = -x_2 + x_5 \quad (45)$$

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

$$\begin{aligned} x &= x_2\hat{\Psi}_2 + x_3\hat{\Psi}_3 + x_6\hat{\Psi}_6 + x_5\hat{\Psi}_5 \\ &= -x_2(1-\alpha)\beta - x_3\alpha\beta + x_6\alpha(1+\beta) + x_5(1-\alpha)(1+\beta) \end{aligned} \quad (46)$$

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

$$\frac{\partial x}{\partial \alpha} = x_2\beta - x_3\beta + x_6(1+\beta) - x_5(1+\beta) \quad (47)$$
$$\frac{\partial x}{\partial \beta} = -x_2(1-\alpha) - x_3\alpha + x_6\alpha + x_5(1-\alpha) \quad (48)$$

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

$$\frac{\partial x}{\partial \alpha}|_5 = x_6 - x_5 \tag{49}$$

$$\frac{\partial x}{\partial \beta}|_5 = -x_2 + x_5 \tag{50}$$

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

$$\begin{aligned} x &= x_5\hat{\Psi}_5 + x_6\hat{\Psi}_6 + x_9\hat{\Psi}_9 + x_8\hat{\Psi}_8 \\ &= x_5(1-\alpha)(1-\beta) + x_6\alpha(1-\beta) + x_9\alpha\beta + x_8(1-\alpha)\beta \end{aligned} \tag{51}$$

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

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

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

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

$$\frac{\partial x}{\partial \alpha}|_5 = -x_5 + x_6 \tag{54}$$

$$\frac{\partial x}{\partial \beta}|_5 = -x_5 + x_8 \tag{55}$$

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

$$\begin{aligned} x &= x_4\hat{\Psi}_4 + x_5\hat{\Psi}_5 + x_8\hat{\Psi}_8 + x_7\hat{\Psi}_7 \\ &= -x_4\alpha(1-\beta) + x_5(1+\alpha)(1-\beta) + x_8(1+\alpha)\beta - x_7\alpha\beta \end{aligned} \tag{56}$$

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

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

$$\frac{\partial x}{\partial \beta} = x_4\alpha - x_5(1+\alpha) + x_8(1+\alpha) - x_7\alpha \tag{58}$$

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

$$\frac{\partial x}{\partial \alpha}|_5 = -x_4 + x_5 \tag{59}$$

$$\frac{\partial x}{\partial \beta}|_5 = -x_5 + x_8 \tag{60}$$

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

$$\begin{aligned} \frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{4}(x_5 - x_4 + x_6 - x_5 - x_5 + x_6 - x_4 + x_5) \\ &= \frac{1}{2}(x_6 - x_4) \end{aligned} \tag{61}$$

$$\begin{aligned} \frac{\partial x}{\partial \beta}|_5 &= \frac{1}{4}(-x_2 + x_5 - x_2 + x_5 - x_5 + x_8 - x_5 + x_8) \\ &= \frac{1}{2}(x_8 - x_2) \end{aligned} \tag{62}$$

Similarly, we have

$$\frac{\partial y}{\partial \alpha} = \frac{1}{2}(y_6 - y_4) \tag{63}$$

$$\frac{\partial y}{\partial \beta} = \frac{1}{2}(y_8 - y_2) \tag{64}$$

$$\frac{\partial z}{\partial \alpha} = \frac{1}{2}(z_6 - z_4) \tag{65}$$

$$\frac{\partial z}{\partial \beta} = \frac{1}{2}(z_8 - z_2) \tag{66}$$

The partial derivatives of x, y, z with respect to α, β 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

$$\begin{aligned} \frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{2}(x_6 - x_5 + x_6 - x_5) \\ &= x_6 - x_5 \end{aligned} \tag{67}$$

$$\begin{aligned} \frac{\partial x}{\partial \beta}|_5 &= \frac{1}{2}(-x_2 + x_5 - x_5 + x_8) \\ &= \frac{1}{2}(x_8 - x_2) \end{aligned} \tag{68}$$

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

$$\begin{aligned} \frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{2}(x_5 - x_4 - x_4 + x_5) \\ &= x_5 - x_4 \qquad (69) \\ \frac{\partial x}{\partial \beta}|_5 &= \frac{1}{2}(-x_2 + x_5 - x_5 + x_8) \\ &= \frac{1}{2}(x_8 - x_2) \qquad (70) \end{aligned}$$

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

$$\begin{aligned} \frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{2}(x_5 - x_4 + x_6 - x_5) \\ &= \frac{1}{2}(x_6 - x_4) \qquad (71) \\ \frac{\partial x}{\partial \beta}|_5 &= \frac{1}{2}(-x_2 + x_5 - x_2 + x_5) \\ &= x_5 - x_2 \qquad (72) \end{aligned}$$

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

$$\begin{aligned} \frac{\partial x}{\partial \alpha}|_5 &= \frac{1}{2}(-x_5 + x_6 - x_4 + x_5) \\ &= \frac{1}{2}(x_6 - x_4) \qquad (73) \\ \frac{\partial x}{\partial \beta}|_5 &= \frac{1}{2}(-x_5 + x_8 - x_5 + x_8) \\ &= x_8 - x_5 \qquad (74) \end{aligned}$$

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

$$dx = \frac{\partial x}{\partial \alpha} d\alpha + \frac{\partial x}{\partial \beta} d\beta \qquad (75)$$

$$dy = \frac{\partial y}{\partial \alpha} d\alpha + \frac{\partial y}{\partial \beta} d\beta \tag{76}$$

$$dz = \frac{\partial z}{\partial \alpha} d\alpha + \frac{\partial z}{\partial \beta} d\beta \tag{77}$$

or

$$\begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \alpha} & \frac{\partial x}{\partial \beta} \\ \frac{\partial y}{\partial \alpha} & \frac{\partial y}{\partial \beta} \\ \frac{\partial z}{\partial \alpha} & \frac{\partial z}{\partial \beta} \end{bmatrix} \begin{bmatrix} d\alpha \\ d\beta \end{bmatrix} \tag{78}$$

or

$$\begin{bmatrix} \frac{\partial x}{\partial \alpha} & \frac{\partial y}{\partial \alpha} & \frac{\partial z}{\partial \alpha} \\ \frac{\partial x}{\partial \beta} & \frac{\partial y}{\partial \beta} & \frac{\partial z}{\partial \beta} \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} =$$

$$\begin{bmatrix} (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 & \frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta} \\ \frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta} & (\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2 \end{bmatrix} \begin{bmatrix} d\alpha \\ d\beta \end{bmatrix} \tag{79}$$

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

$$d\alpha = \frac{\partial \alpha}{\partial x} dx + \frac{\partial \alpha}{\partial y} dy + \frac{\partial \alpha}{\partial z} dz \tag{80}$$

$$d\beta = \frac{\partial \beta}{\partial x} dx + \frac{\partial \beta}{\partial y} dy + \frac{\partial \beta}{\partial z} dz \tag{81}$$

or

$$\begin{bmatrix} d\alpha \\ d\beta \end{bmatrix} = \begin{bmatrix} \frac{\partial \alpha}{\partial x} & \frac{\partial \alpha}{\partial y} & \frac{\partial \alpha}{\partial z} \\ \frac{\partial \beta}{\partial x} & \frac{\partial \beta}{\partial y} & \frac{\partial \beta}{\partial z} \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{82}$$

from Equation (79)

$$\begin{bmatrix} d\alpha \\ d\beta \end{bmatrix} = \frac{1}{|J|} \begin{bmatrix} (\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2 & -(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta}) \\ -(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta}) & (\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial x}{\partial \alpha} & \frac{\partial y}{\partial \alpha} & \frac{\partial z}{\partial \alpha} \\ \frac{\partial x}{\partial \beta} & \frac{\partial y}{\partial \beta} & \frac{\partial z}{\partial \beta} \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{83}$$

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 (α, β) plane. $|J|$ must be greater than 0 to ensure the inverse mapping exists.

$$\begin{aligned} |J| &= [(\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2][(\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2] \\ &\quad -(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})^2 \end{aligned} \tag{84}$$

From equations (82) and (83) we have

$$\frac{\partial \alpha}{\partial x} = \frac{1}{|J|}[\frac{\partial x}{\partial \alpha}[(\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2] - \frac{\partial x}{\partial \beta}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{85}$$

$$\frac{\partial \alpha}{\partial y} = \frac{1}{|J|}[\frac{\partial y}{\partial \alpha}[(\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2] - \frac{\partial y}{\partial \beta}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{86}$$

$$\frac{\partial \alpha}{\partial z} = \frac{1}{|J|}[\frac{\partial z}{\partial \alpha}[(\frac{\partial x}{\partial \beta})^2 + (\frac{\partial y}{\partial \beta})^2 + (\frac{\partial z}{\partial \beta})^2] - \frac{\partial z}{\partial \beta}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{87}$$

$$\frac{\partial \beta}{\partial x} = \frac{1}{|J|}[\frac{\partial x}{\partial \beta}[(\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2] - \frac{\partial x}{\partial \alpha}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{88}$$

$$\frac{\partial \beta}{\partial y} = \frac{1}{|J|}[\frac{\partial y}{\partial \beta}[(\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2] - \frac{\partial y}{\partial \alpha}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{89}$$

$$\frac{\partial \beta}{\partial z} = \frac{1}{|J|}[\frac{\partial z}{\partial \beta}[(\frac{\partial x}{\partial \alpha})^2 + (\frac{\partial y}{\partial \alpha})^2 + (\frac{\partial z}{\partial \alpha})^2] - \frac{\partial z}{\partial \alpha}(\frac{\partial x}{\partial \alpha}\frac{\partial x}{\partial \beta} + \frac{\partial y}{\partial \alpha}\frac{\partial y}{\partial \beta} + \frac{\partial z}{\partial \alpha}\frac{\partial z}{\partial \beta})] \tag{90}$$

Now, the derivatives $\frac{\partial x}{\partial y}$, $\frac{\partial x}{\partial z}$, $\frac{\partial v}{\partial y}$, $\frac{\partial v}{\partial z}$, $\frac{\partial w}{\partial y}$ and $\frac{\partial w}{\partial z}$ can be computed.

$$\frac{\partial x}{\partial y} = \frac{\partial x}{\partial \alpha}\frac{\partial \alpha}{\partial y} + \frac{\partial x}{\partial \beta}\frac{\partial \beta}{\partial y} \tag{91}$$

$$\frac{\partial x}{\partial z} = \frac{\partial x}{\partial \alpha}\frac{\partial \alpha}{\partial z} + \frac{\partial x}{\partial \beta}\frac{\partial \beta}{\partial z} \tag{92}$$

$$\frac{\partial v}{\partial y} = \frac{\partial v}{\partial \alpha}\frac{\partial \alpha}{\partial y} + \frac{\partial v}{\partial \beta}\frac{\partial \beta}{\partial y} \tag{93}$$

$$\frac{\partial v}{\partial z} = \frac{\partial v}{\partial \alpha}\frac{\partial \alpha}{\partial z} + \frac{\partial v}{\partial \beta}\frac{\partial \beta}{\partial z} \tag{94}$$

$$\frac{\partial w}{\partial y} = \frac{\partial w}{\partial \alpha}\frac{\partial \alpha}{\partial y} + \frac{\partial w}{\partial \beta}\frac{\partial \beta}{\partial y} \tag{95}$$

$$\frac{\partial w}{\partial z} = \frac{\partial w}{\partial \alpha}\frac{\partial \alpha}{\partial z} + \frac{\partial w}{\partial \beta}\frac{\partial \beta}{\partial z} \tag{96}$$

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

$$\hat{\Psi}_1 = (1-\alpha)(1-\beta) \quad (97)$$

$$\hat{\Psi}_2 = \alpha(1-\beta) \quad (98)$$

$$\hat{\Psi}_3 = \alpha\beta \quad (99)$$

$$\hat{\Psi}_4 = (1-\alpha)\beta \quad (100)$$

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 α and β, 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

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

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

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

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

$$H^* = 0.5(H_w + H_e) + 0.22(H_{aw} - H_e) \qquad (103)$$

The boundary-layer thickness is given approximately by the equation

$$(\delta/\theta)_L = 5.55 \tag{104}$$

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

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

and

$$\theta_T = (c_2 \int_0^s \rho^* u_e (\mu^*)^m h^{c_3} dS)^{c_4}/(\rho_e u_e h) \tag{106}$$

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

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

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

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

$$c_4 = 1/c_3 \tag{111}$$

$$c_5 = 2.2433 + 0.93N \tag{112}$$

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

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

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

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

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

9 Gas Properties

The equilibrium air curvefits of Gupta *et al*[20] were used to evaluate thermodynamic and transport properties of the equilibrium air. The properties include enthalpy, total specific heat at constant pressure, compressibility factor, viscosity, total thermal conductivity, and total Prandtl number.

These curvefits are valid from 500 degrees Kelvin to 30,000 degrees Kelvin over a pressure range of 10^{-4} to 10^{2} atmospheres.

10 Results and Discussion

10.1 Wind Tunnel Case

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

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

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

After the inviscid flowfield solution was obtained, the streamline and metric coefficients from the stagnation region to each of the heat transfer

gauge location were computed by the backward streamline tracing procedure. Figure 7 shows the streamlines on the body surface. Heating rates were then computed by using the BLIMP code. Figure 8 shows the comparison between predicted and measured heating rates for the 30 degree angle of attack case. The result is plotted as q_w versus x/L, where x is the length along the $x - axis$. For the .025 scaled model, L is equal to 32.02 inches. Figure 9 shows the heating rate and pressure distribution along the leading edge of the model. In this double y plot, both q_w and p/p_{stag} are plotted against x/L, where p_{stag} is the stagnation pressure. From this figure, one can see that the heating rate distribution follows closely the presure 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 $135 Btu/ft^2/s$ was measured while the prediction was about $62.5 Btu/ft^2/s$. The measured heating rate at this location is considered unreasonably high.

10.2 STS-5 Flight Case

Since the IEC3D code does not include chemical non-equilibrium effects, a point on the Space Shuttle STS-5 trajectory was chosen for comparison where the flow is expected to be at chemical equilibrium. The point is at an altitude of 159,000 feet. The Mach number is 9.15 and angle of attack is 37.4 degrees. Again, the IEC3D code and the grid adaption procedure described above were used to obtain the inviscid flowfield solution. The initial grid was the same as the wind tunnel case. During the Shuttle flight, the radiometer located near the leading edge panel 9 was used to measure heating rate in the wing shock-shock interaction area. The location viewed by the radiometer is at $x = 1095\, in, y = 256.488\, in$, and $z = 291.94\, in$. The predicted heating rate using the approximate convective-heating equations of Ref. [12] is about $8.0 Btu/ft^2/s$ at this location, and the heating rate

backed out from radiometer data is about $14.0 Btu/ft^2/s$[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

[1] Gnoffo, P. A., "An Upwind-Biased, Point-Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flow," NASA TP-2953, February, 1990.

[2] Cooke, J. C., "An Axially Symmetric Analogue for General Three-Dimensional Boundary Layers," British A.R.C., R&M No. 3200, 1959.

[3] DeJarnette, F. R., and Davis, R. M.,"A Simplified Method for Calculating Laminar Heat Transfer Over bodies at an Angle of Attack," NASA TN D-4720, 1968.

[4] DeJarnette, F. R., and Hamilton, H. H. II, "Inviscid Surface Streamlines and Heat Transfer on Shuttle-Type Configurations," Journal of Spacecraft and Rockets, Vol. 10, May 1973, pp. 314-321.

[5] Goodrich, W. D., Li, C. P., Houston, D. K., Chiu, P. B., and Olmedo, L., "Numerical Computations of Orbiter Flow Fields and Laminar Heating Rates," Journal of Spacecraft and Rockets, Vol. 14, May 1977, pp. 257-264.

[6] Rakich, J. V., and Lanfranco, M. J., "Numerical Computation of Space Shuttle Laminar Heating and Surface Streamlines," Journal of Spacecraft and Rockets, Vol. 14, May 1977, pp. 265-272.

[7] Hamilton, H. H. II, "Calculation of Laminar Heating Rates on Three-Dimensional Configurations Using the Axisymmetric Analogue," NASA TP-1698, Sept. 1980.

[8] Hamilton, H. H. II, Dejarnette, F. R., and Weilmuenster, K. J., "Application of Axisymmetric Analog for Calculating Heating in Three-Dimensional Flows," AIAA Paper 85-0245, Reno, NV, January 1985.

[9] Hamilton, H. H. II, Greene, F. A., and Weilmuenster, K. J., "Comparison of Heating Rate Calculations with Experimental Data on a Modified Shuttle Orbiter at Mach 6," AIAA Paper 91-1347, Honolulu, Hawaii, June 1991.

[10] Tam, L. T., "LU-SGS Implicit Scheme for Entry Vehicle Flow Computation and Comparison with Aerodynamic Data," AIAA Paper 92-2671, June 1992.

[11] Murray, A. L., "Further Enhancements of the BLIMP Computer Code and User's Guide," AFWAL-TR-88-3010, June, 1988.

[12] Zoby, E. V., Moss, J. N., and Sutton, K. S., "Approximate Convective Heating Equations for Hypersonic Flows," Journal of Spacecraft and Rockets, Vol. 18, Jan. 1981, pp 64-70.

[13] Van Leer, B., "Flux-Vector Splitting for the Euler Equations," Lecture Notes in Physics, Vol. 170, 1982, pp. 507-512.

[14] Anderson, W. K., Thomas, L. J., and Van Leer B., "Comparison of Finite Volume Flux Vector Splittings for the Euler Equations," AIAA Journal, Vol. 24, No. 9, Sept. 1986, pp. 1453-1460.

[15] Roe, P. L., "Characteristic-Based Schemes for the Euler Equations," Annual Review of Fluid Mechanics, pp. 337-356, 1986.

[16] Jamerson, A. and Yoon, S., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," AIAA Journal, Vol. 25, 1987, pp. 929-935.

[17] An, M. Y., Wang, K. C., and Tam L. T., "Computation of Inviscid Flowfield Around 3-D Aerospace Vehicles and Comparison with Experimental & Flight Data," AIAA Paper 93-0885, Reno NV., January 1993.

[18] Dejarnette, F. R., Private Communication, August 1991.

[19] Johnson, C. B. and Bushnell, D. M., "Power-Law Velocity Profile Exponent Variations with Reynolds Number, Wall Cooling, and Mach Number in a Turbulent Boundary Layer," NASA TN D-5753, April, 1970.

[20] Gupta, R. N., Lee, K. P., Thompson, R. A., and Yos, J. M., "Calculations and Curve Fits of Thermodynamic and Transport Properties for Equilibrium Air to 30,000 K," NASA Reference Publication 1260, Oct. 1991.

[21] Wittliff, C. E. and Berthold C. L., "Results of Heat Transfer Testing of an 0.025-Scale Model(66-0) of the Space Shutter Orbiter Configuration 140B in the Calspan Hypersonic Shock Tunnel(OH66)," NASA CR-151,405, January 1978.

[22] Campbell C. H., Private Communication, September, 1992.

[23] Davies, C. B. and Venkatapathy, E., "The Multidimensional Self-Adaptive Grid Code, SAGE," NASA TM-103905, July 1992.

[24] Chao, D. C., Battley, H. H., Gallegos, J. J., and Curry, D. M., "Thermal Mathematical Modeling and System Simulation of Space Shuttle LESS Subsystem," AIAA Paper 84-1772, Snowmass, CO. June 1984.

[25] Hartung, L. C., and Throckmorton, D. A., "Space Shuttle Entry Heating Data Book," Vol. III - STS-5, NASA Reference Publication 1193, Parts 1 & 2, May 1988.

[26] Hartung, L. C., and Throckmorton, D. A., "Space Shuttle Entry Heating Data Book," Vol. I - STS-2, NASA Reference Publication 1191, Parts 1 & 2, May 1988.

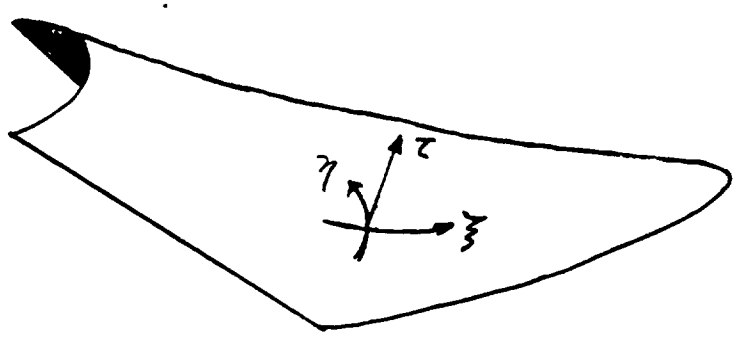

Figure 1: Streamline coordinates

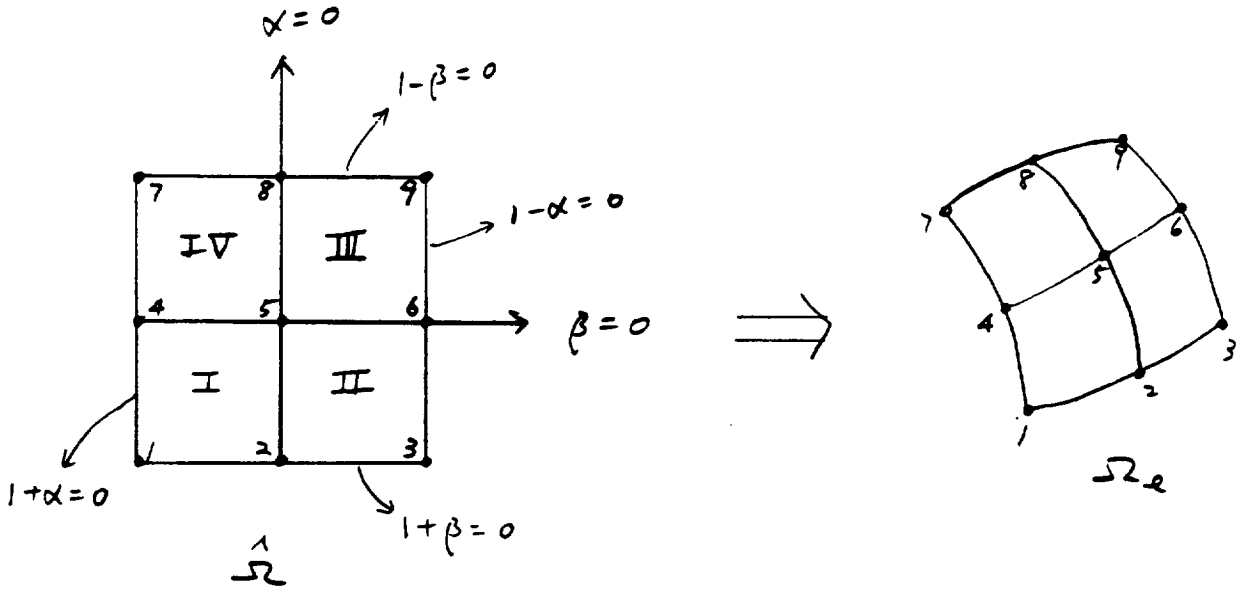

Figure 2: Mapping from master element to a physical element

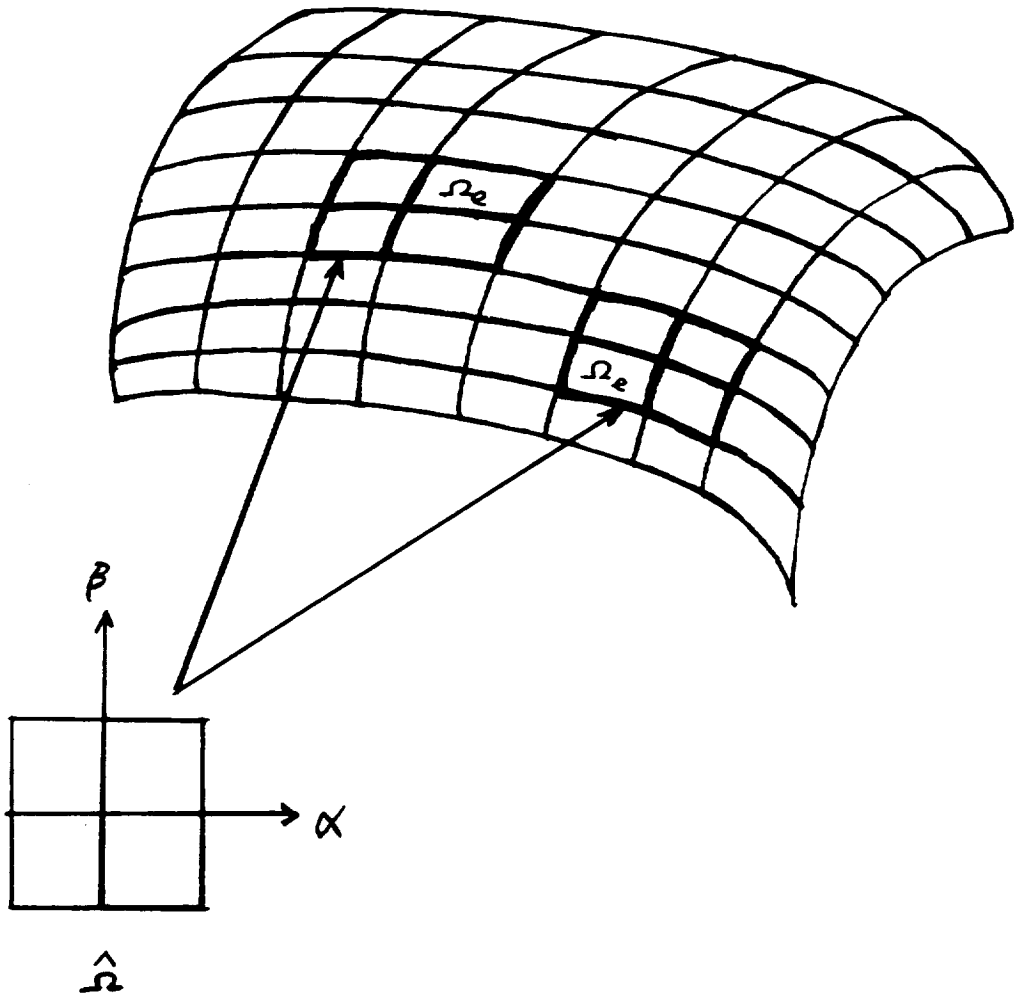

Figure 3: Mapping from master element to physical domain

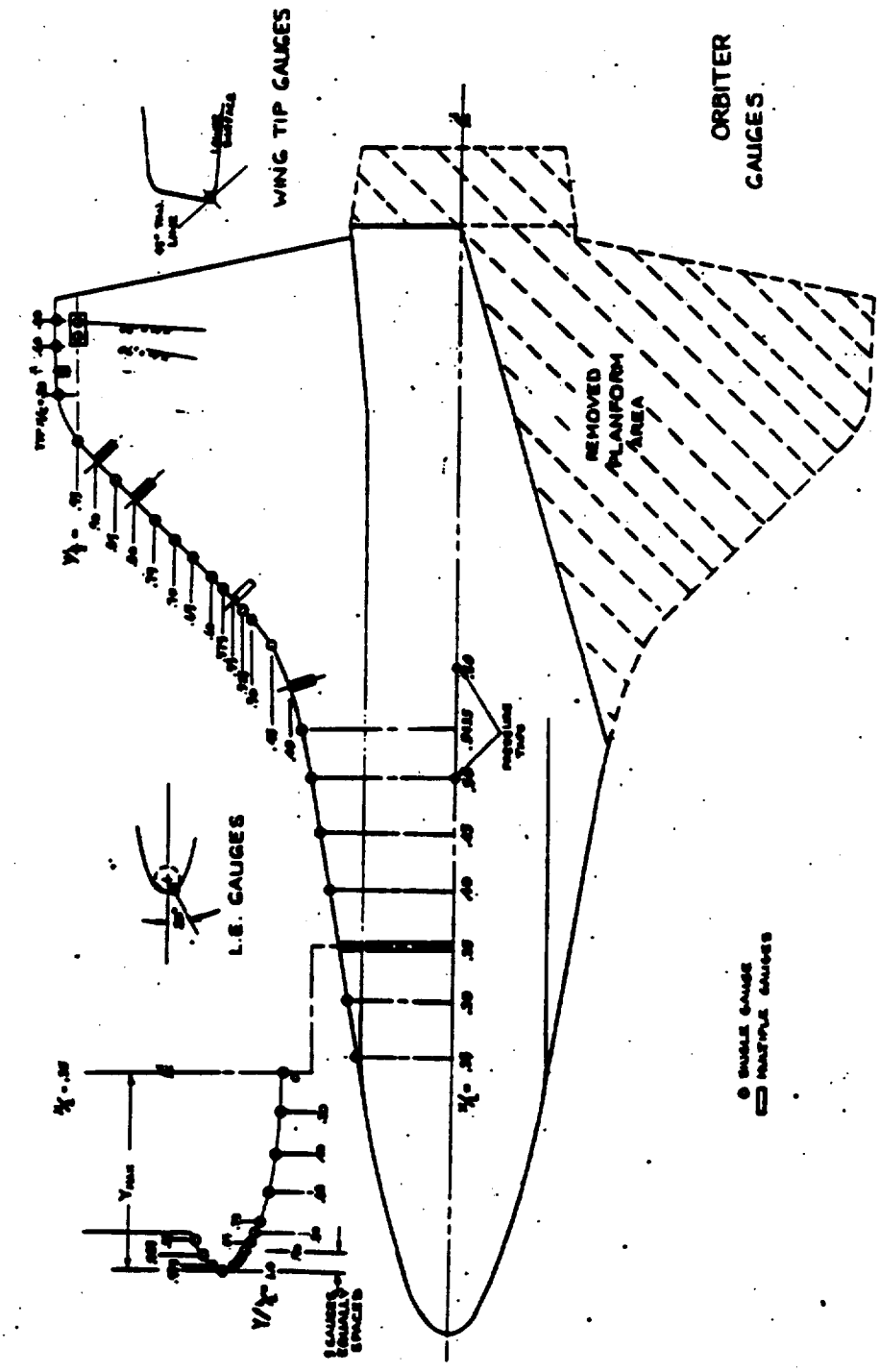

Figure 4: Layout of thin-film heat transfer guage

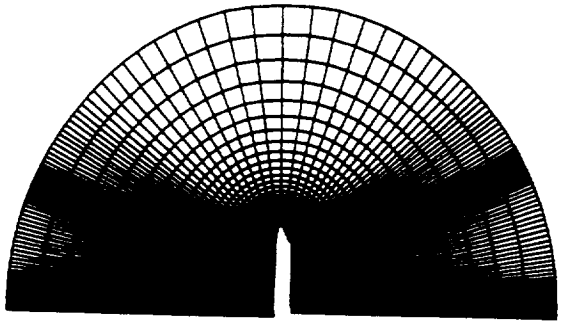

Figure 5: Pitch plane initial grid

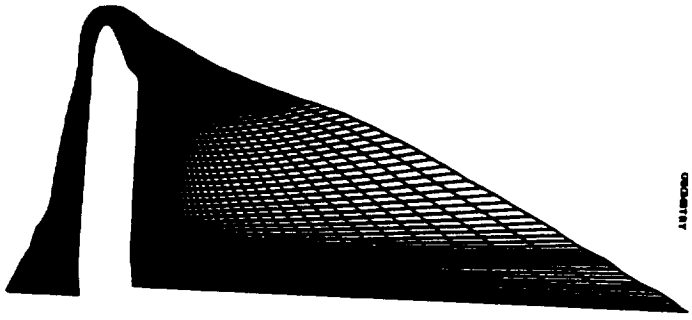

Figure 6: Pitch plane grid after outer boundary adjustment

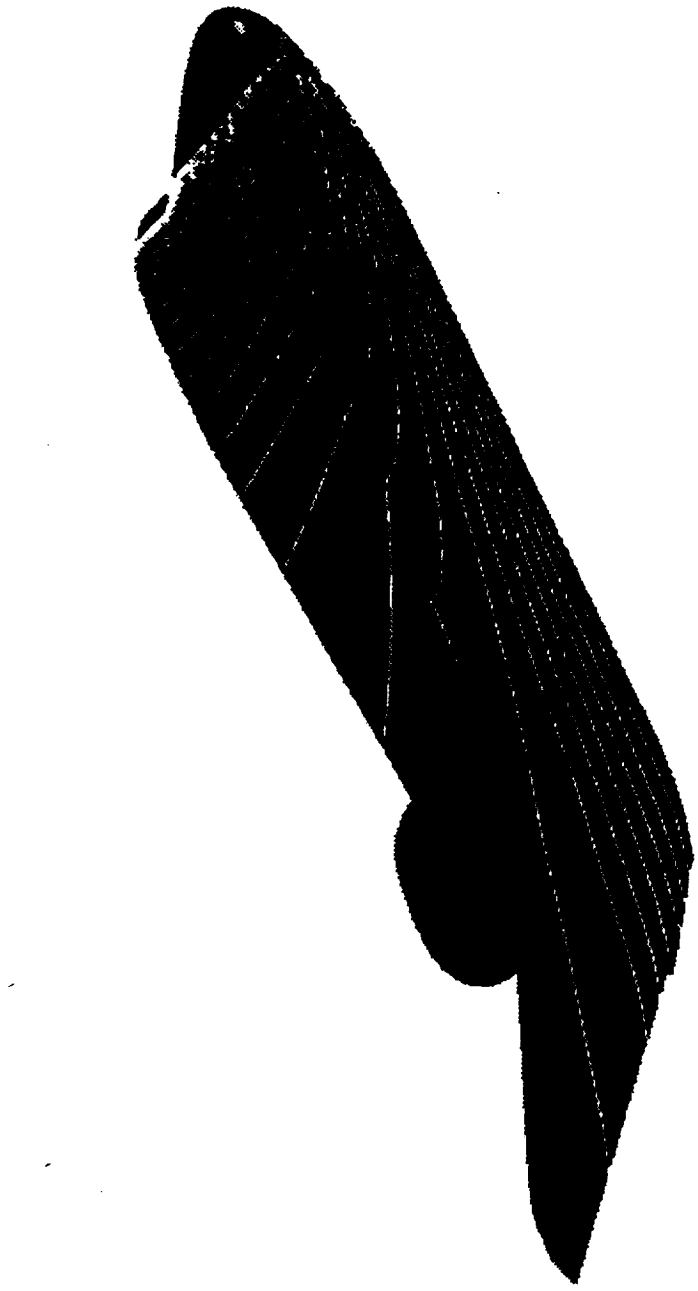

Figure 7: Surface streamlines

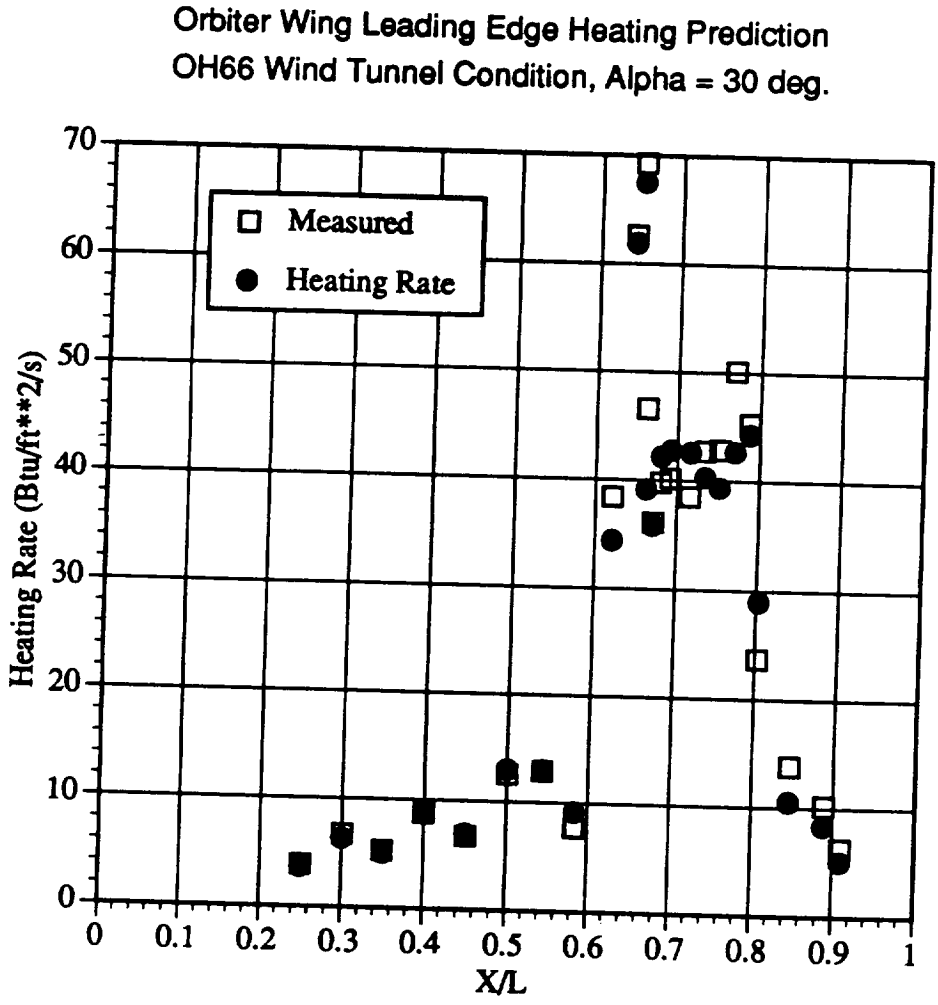

Figure 8: Comparison of predicted and measured heating rates, $\alpha = 30$ *deg*.

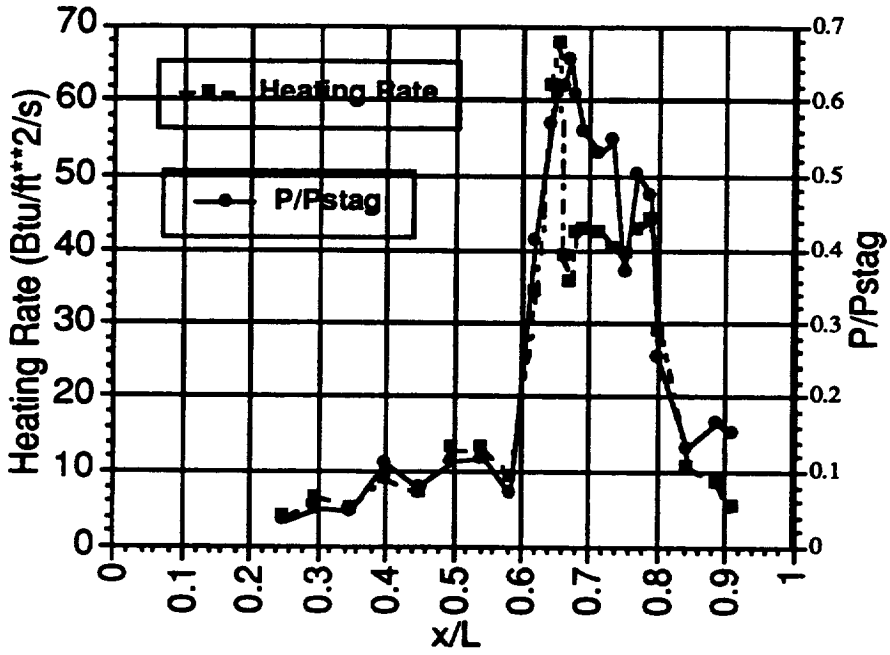

Figure 9: Pressure and Heating rate distribution along the leading edge

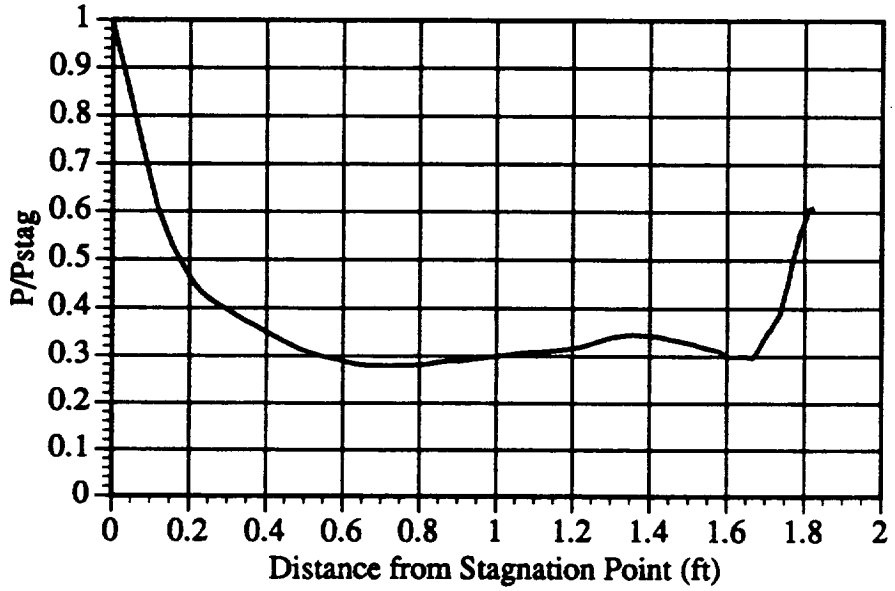

Figure 10: Typical pressure distribution along a leading edge streamline

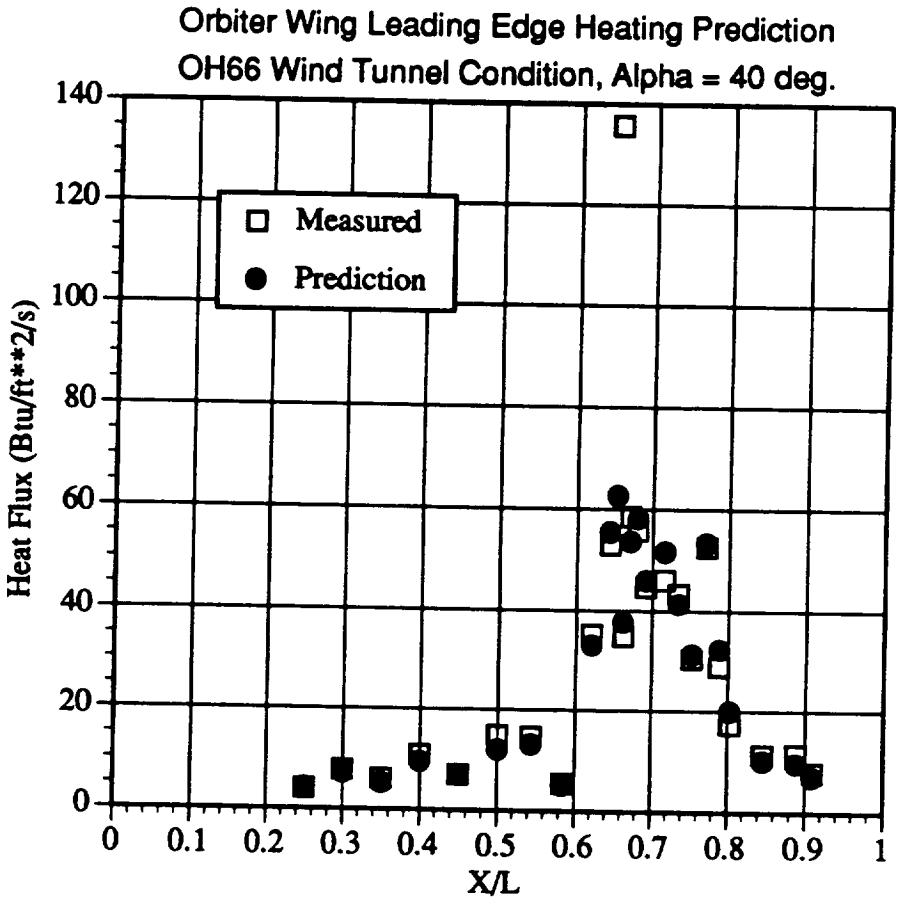

Figure 11: Comparison of predicted and measured heating rates, $\alpha = 40\ deg.$

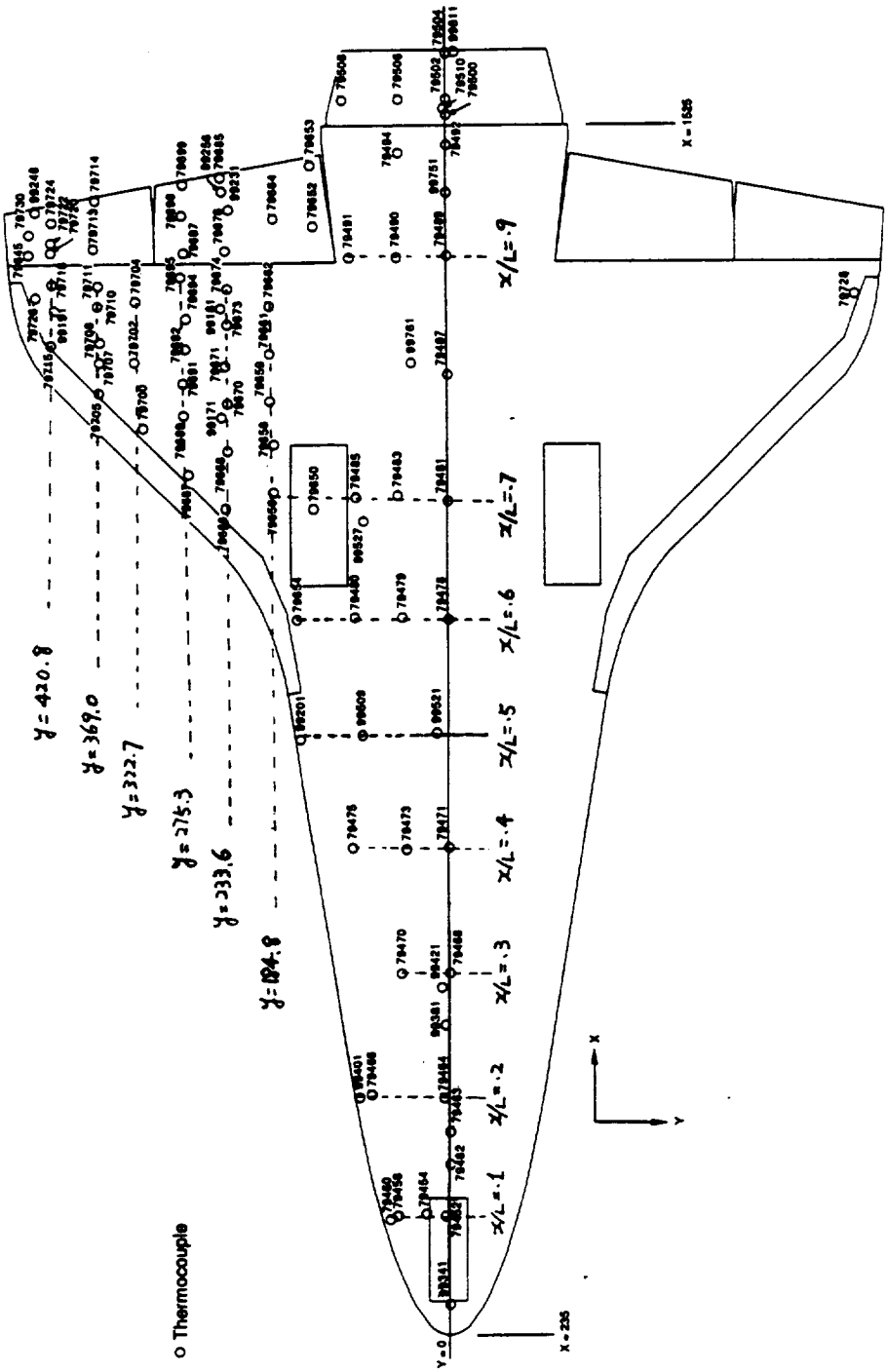

Figure 12: Orbiter windward surface thermocouple locations

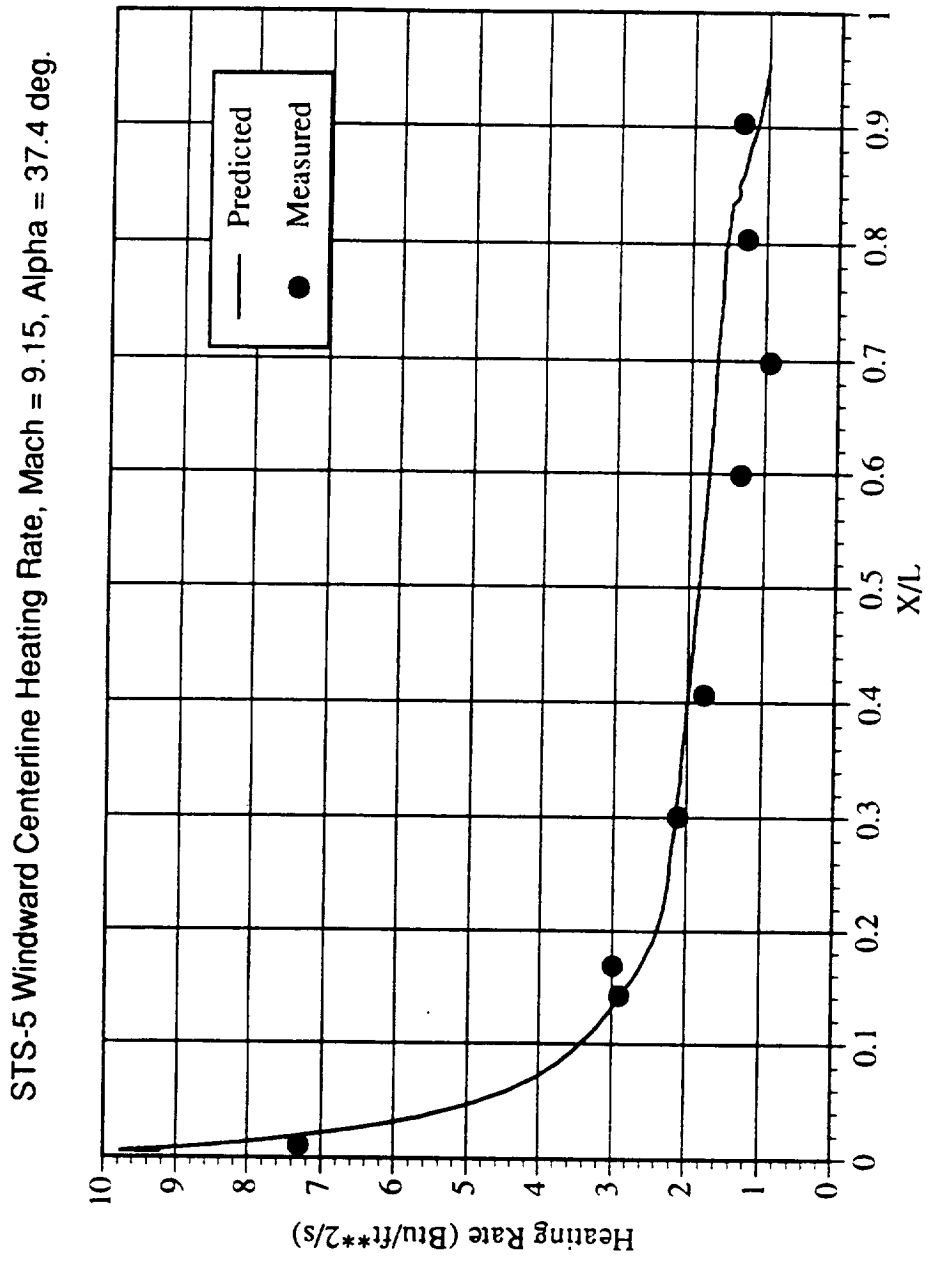

Figure 13: Windward centerline heating distribution, STS-5 case, $Mach$ = 9.15, $Alpha$ = 37.4 deg.

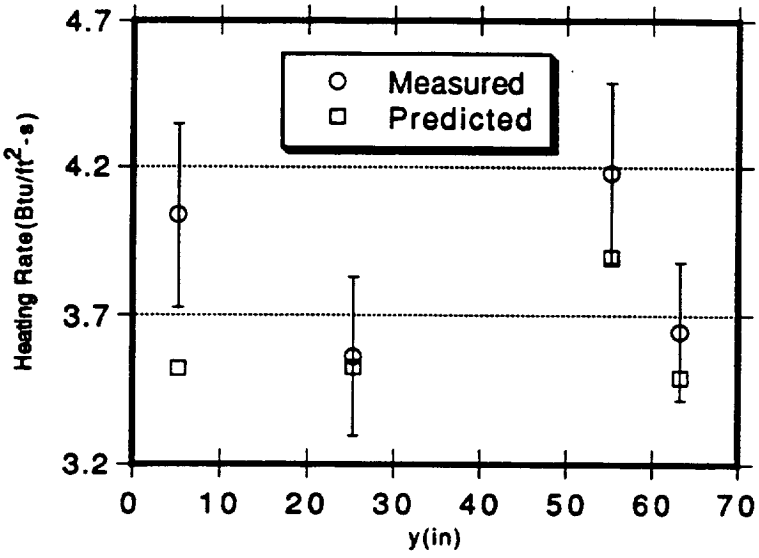

Figure 14: Circumferential heating distribution at $x/L = .1$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg*.

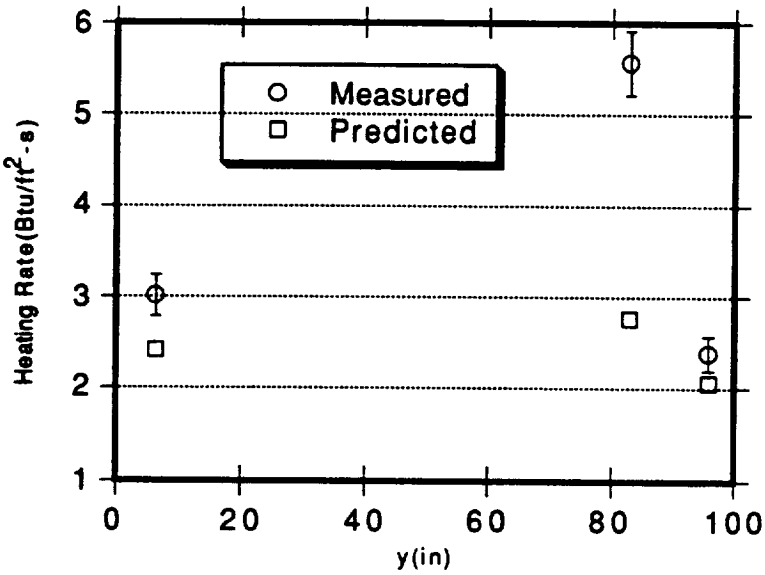

Figure 15: Circumferential heating distribution at $x/L = .2$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg*.

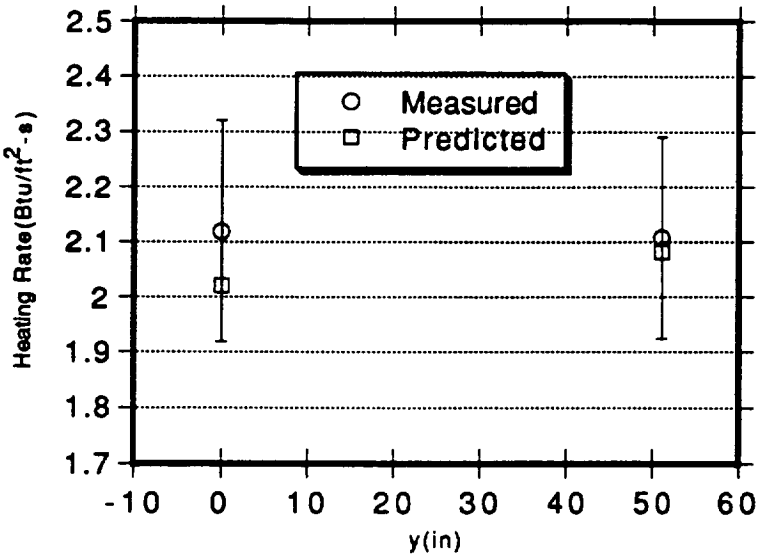

Figure 16: Circumferential heating distribution at $x/L = .3$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

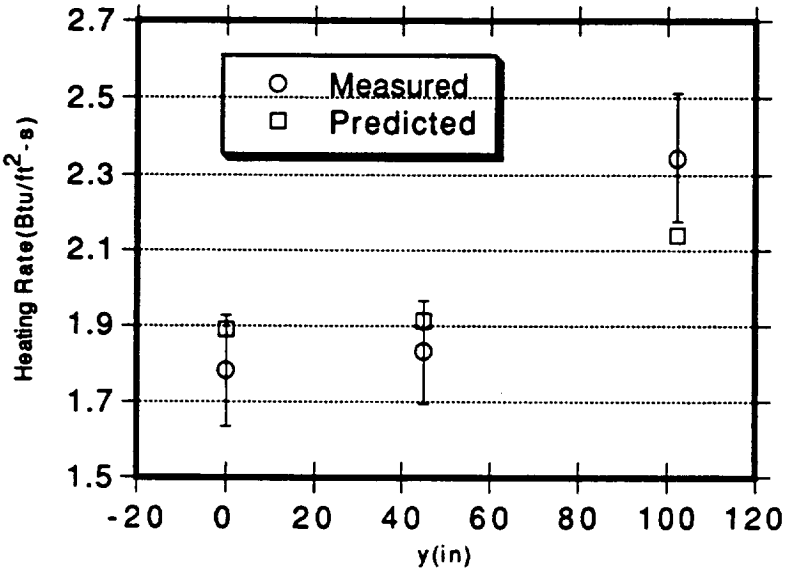

Figure 17: Circumferential heating distribution at $x/L = .4$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

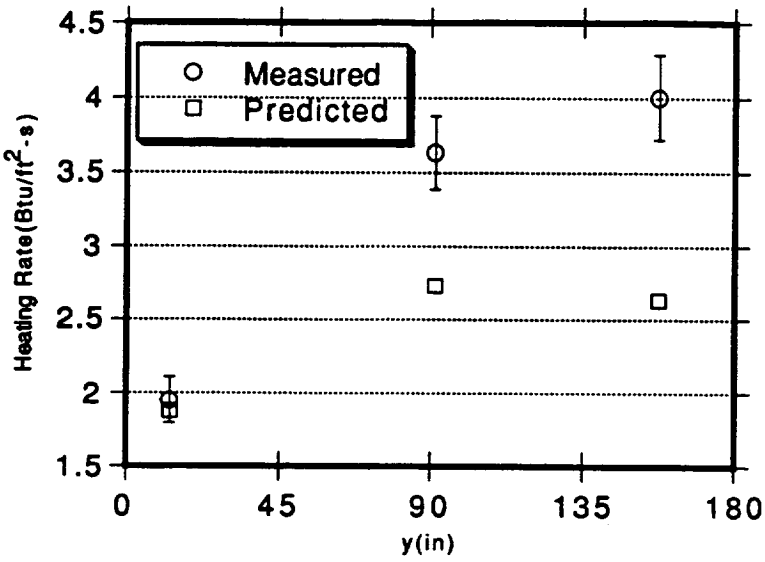

Figure 18: Circumferential heating distribution at $x/L = .5$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

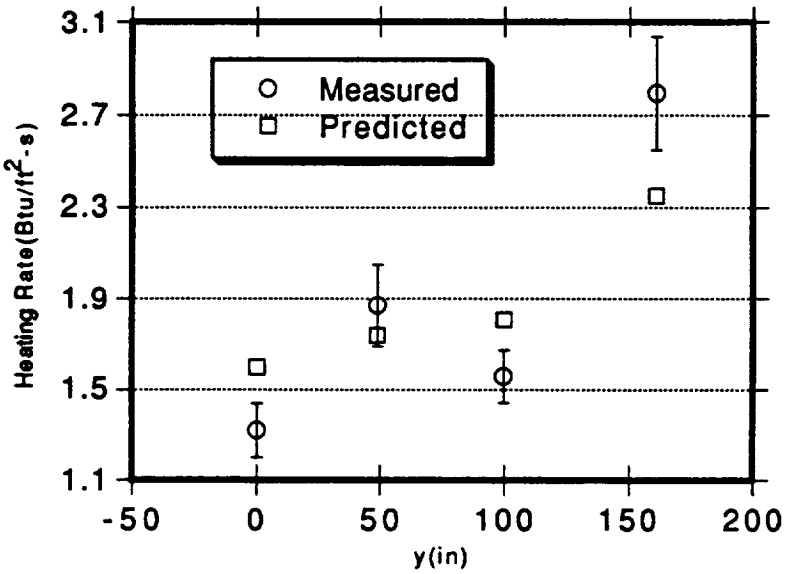

Figure 19: Circumferential heating distribution at $x/L = .6$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

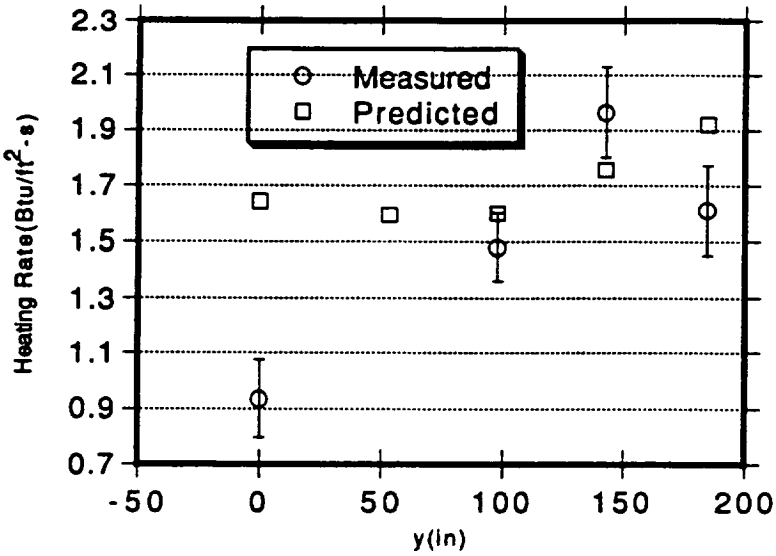

Figure 20: Circumferential heating distribution at $x/L = .7$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg*.

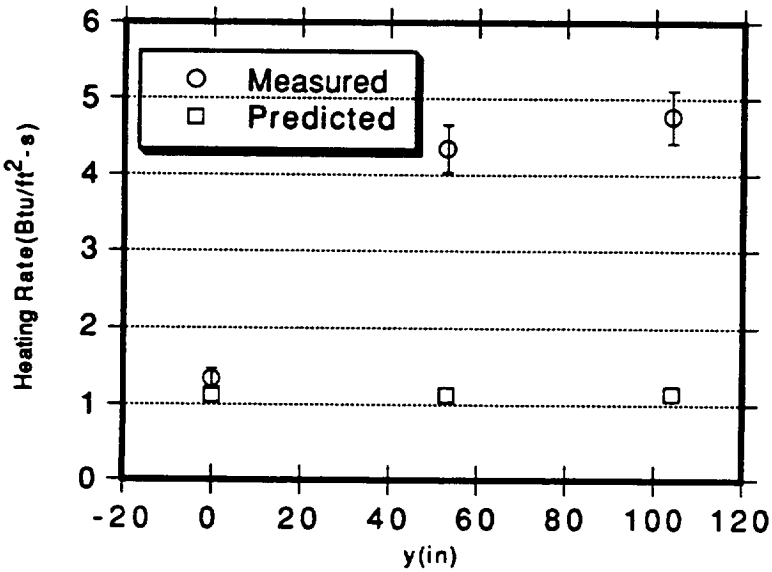

Figure 21: Circumferential heating distribution at $x/L = .9$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg*.

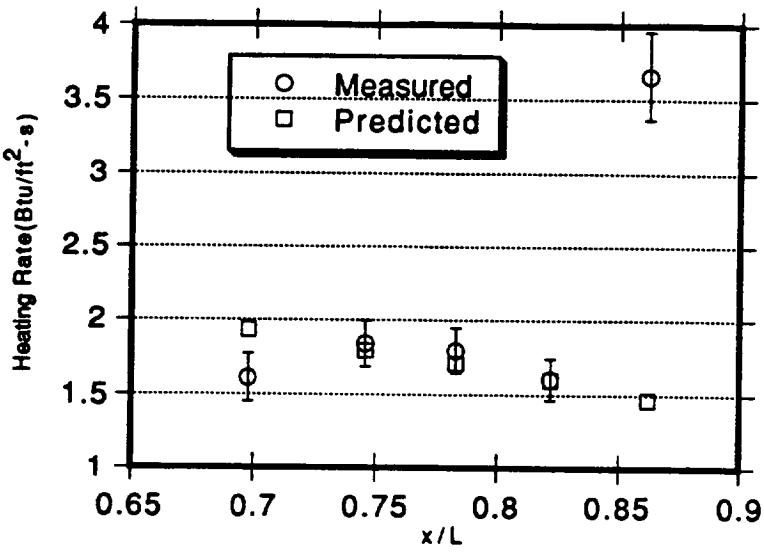

Figure 22: Heating distribution on windward surface at $y = 184.8\ in.$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

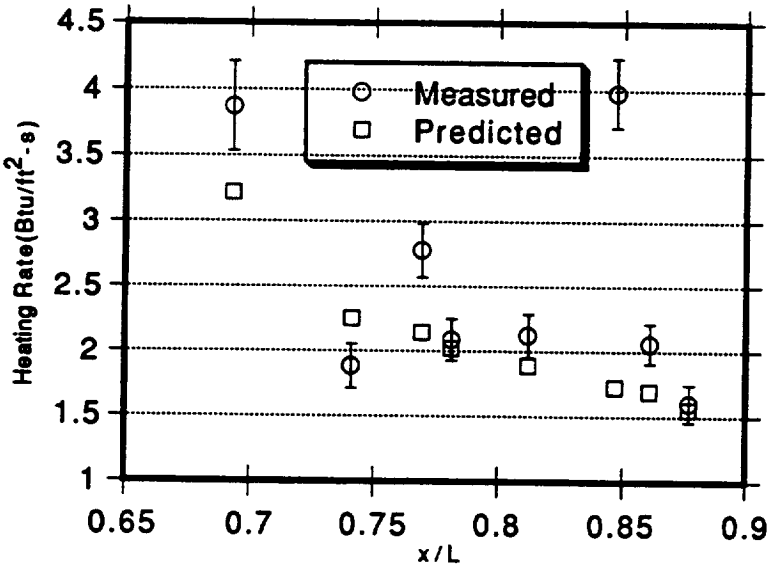

Figure 23: Heating distribution on windward surface at $y = 233.6\ in.$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

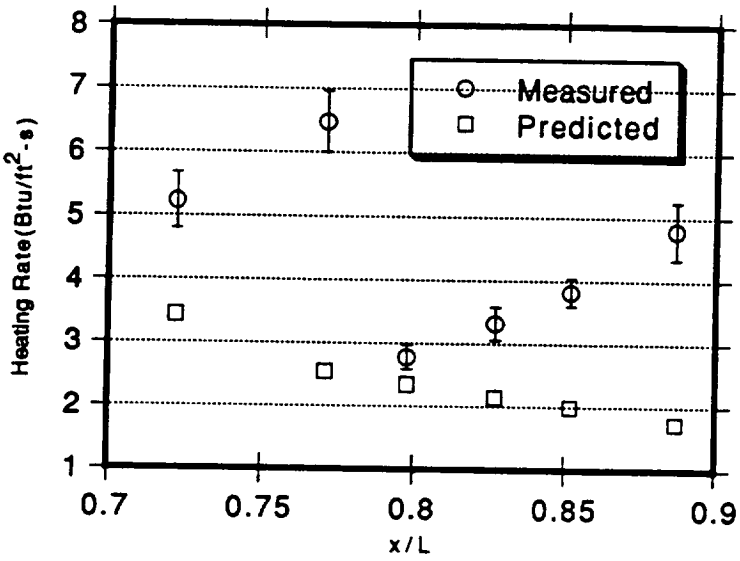

Figure 24: Heating distribution on windward surface at $y = 275.3$ *in.*, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg.*

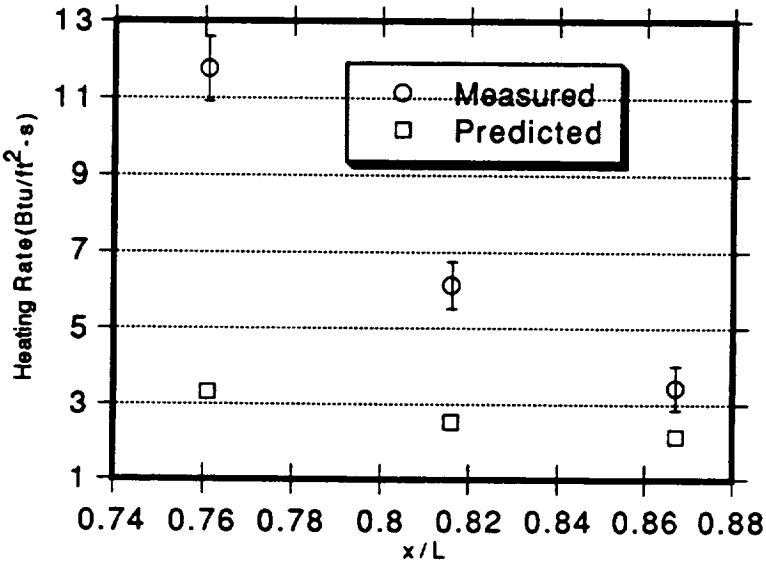

Figure 25: Heating distribution on windward surface at $y = 322.7$ *in.*, STS-5 case, $Mach = 9.15$, $Alpha = 37.4$ *deg.*

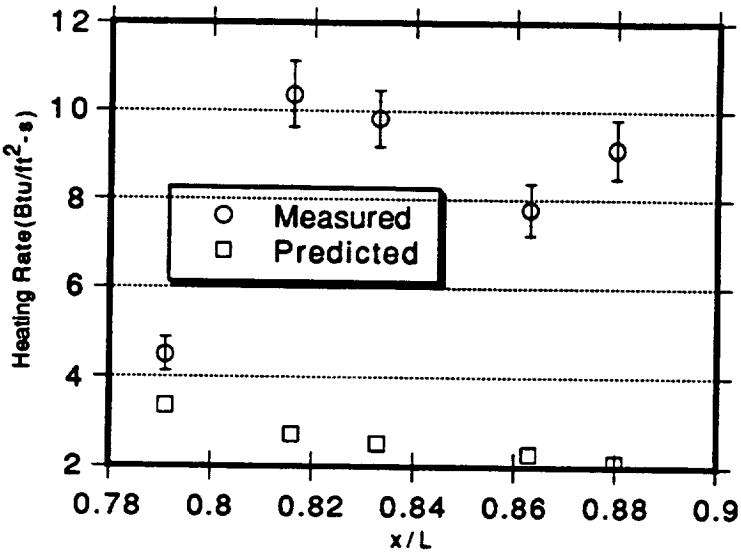

Figure 26: Heating distribution on windward surface at $y = 369.0\ in.$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

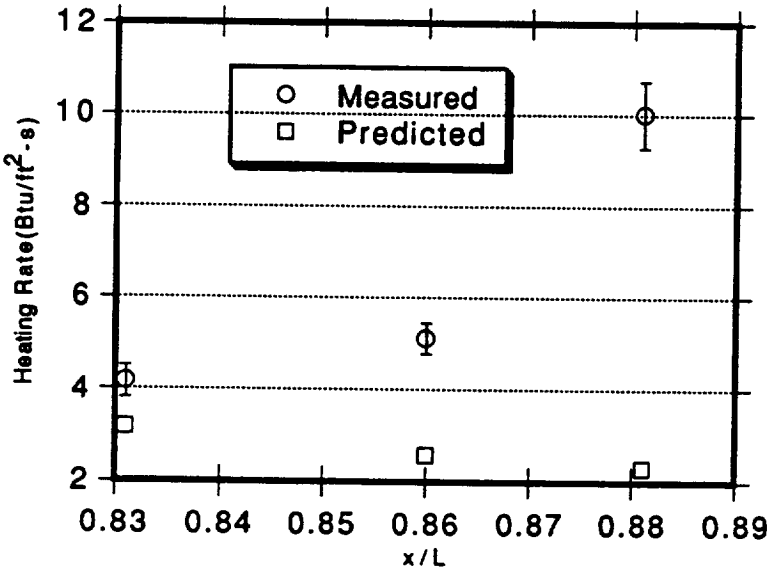

Figure 27: Heating distribution on windward surface at $y = 420.8\ in.$, STS-5 case, $Mach = 9.15$, $Alpha = 37.4\ deg.$

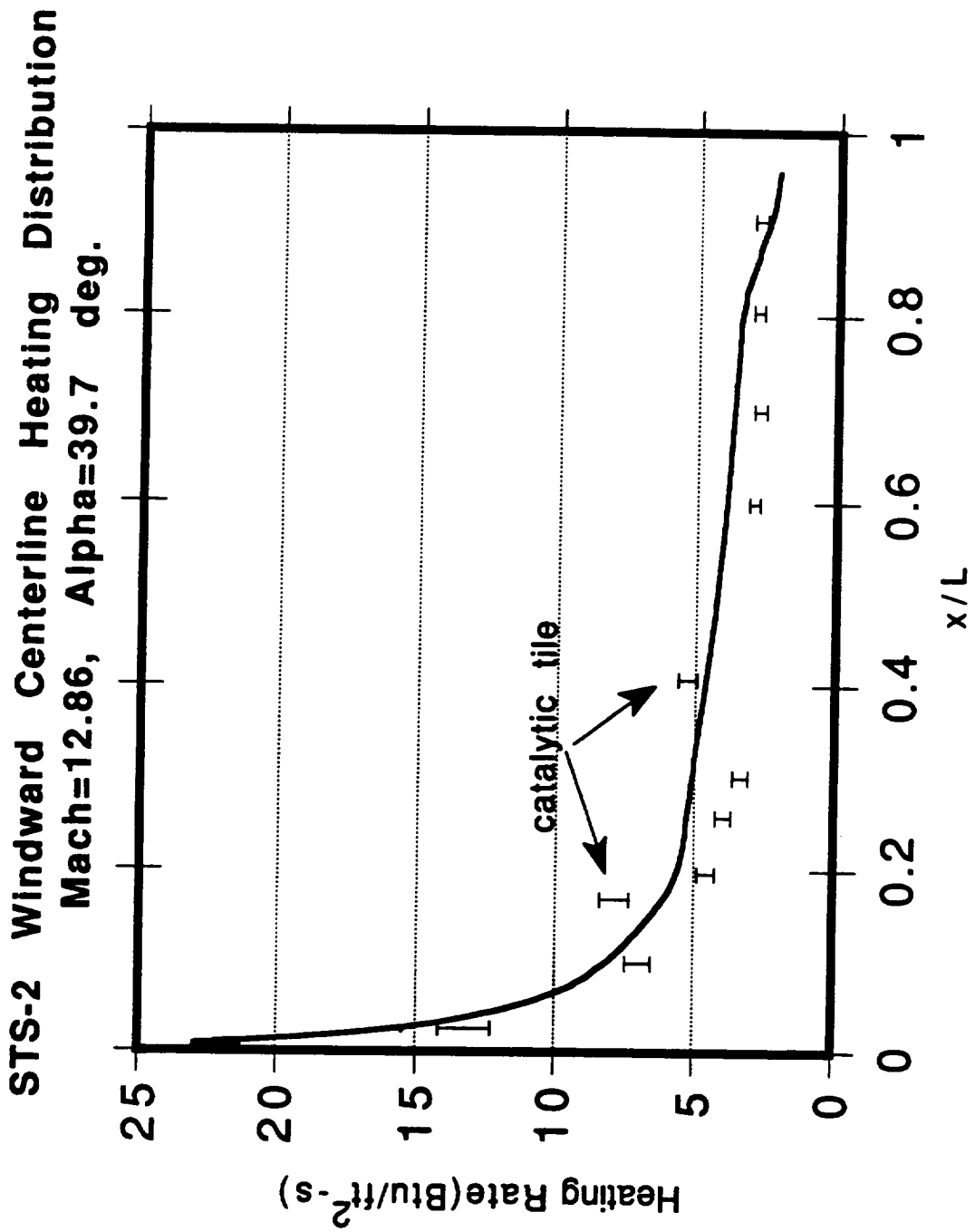

Figure 28: Heating distribution along windward centerline, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

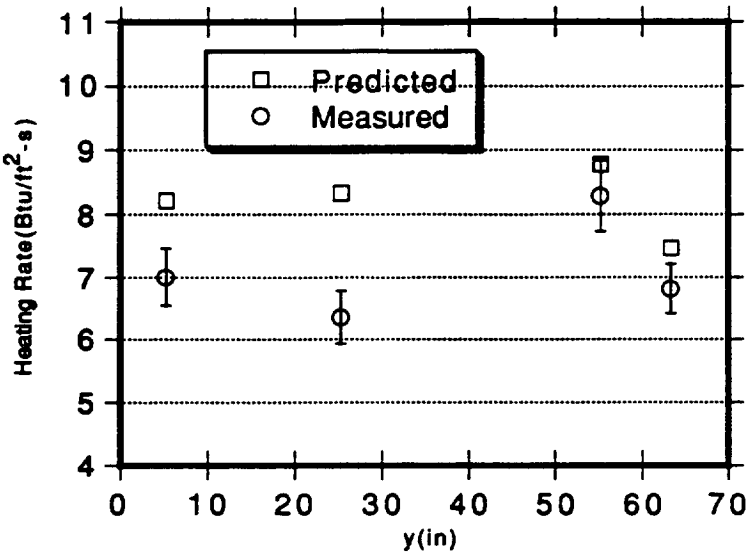

Figure 29: Circumferential heating distribution at $x/L = .1$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

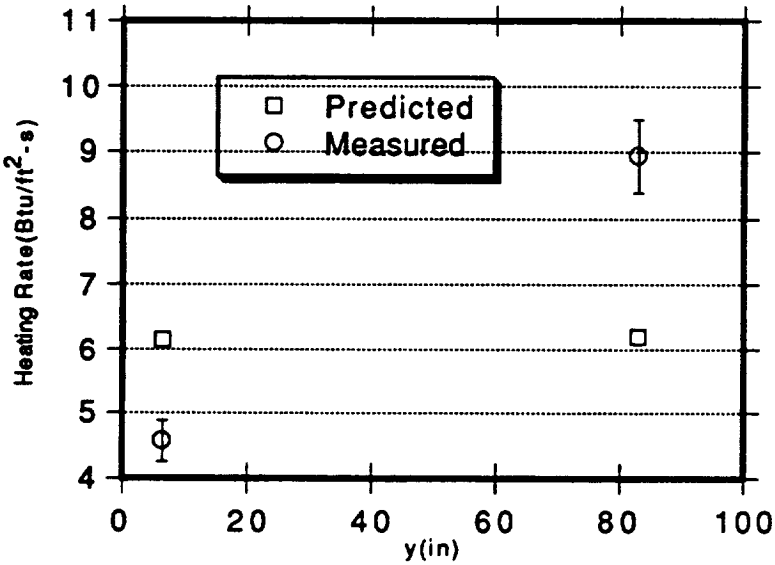

Figure 30: Circumferential heating distribution at $x/L = .2$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

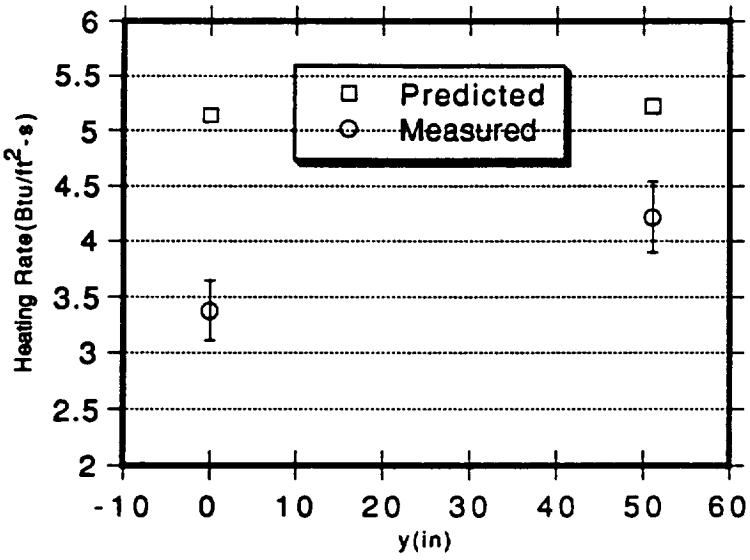

Figure 31: Circumferential heating distribution at $x/L = .3$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

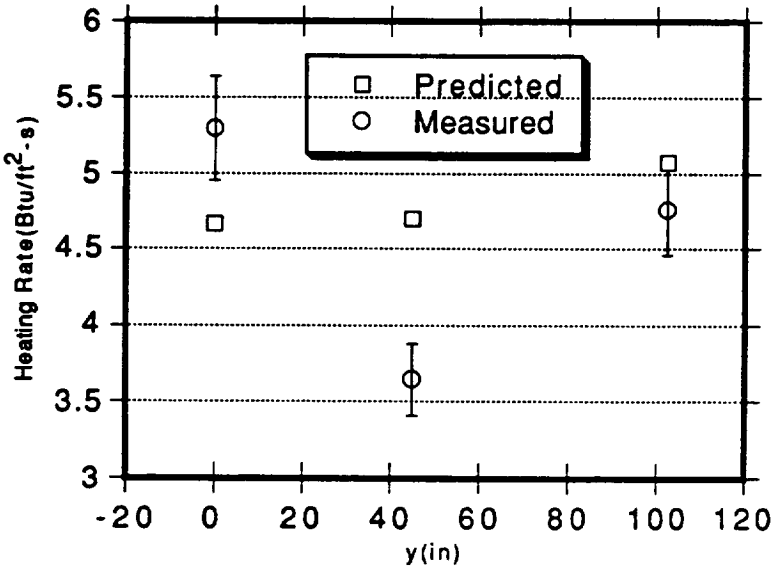

Figure 32: Circumferential heating distribution at $x/L = .4$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

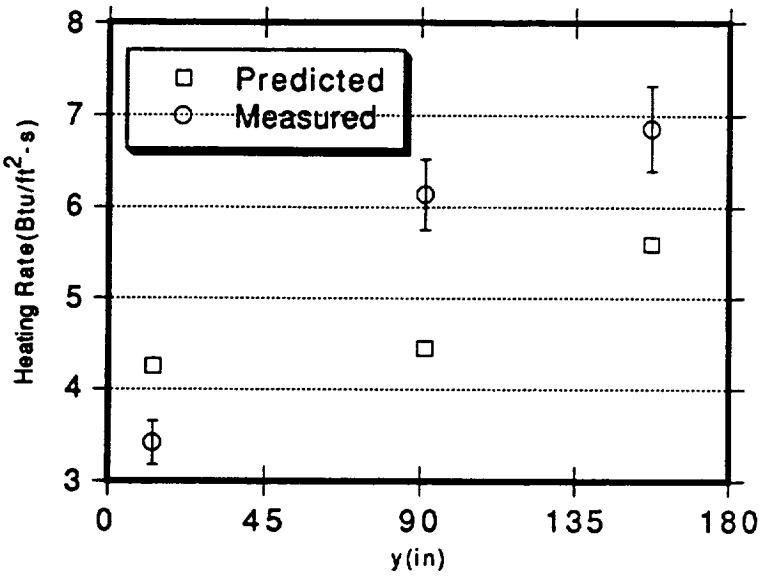

Figure 33: Circumferential heating distribution at $x/L = .5$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

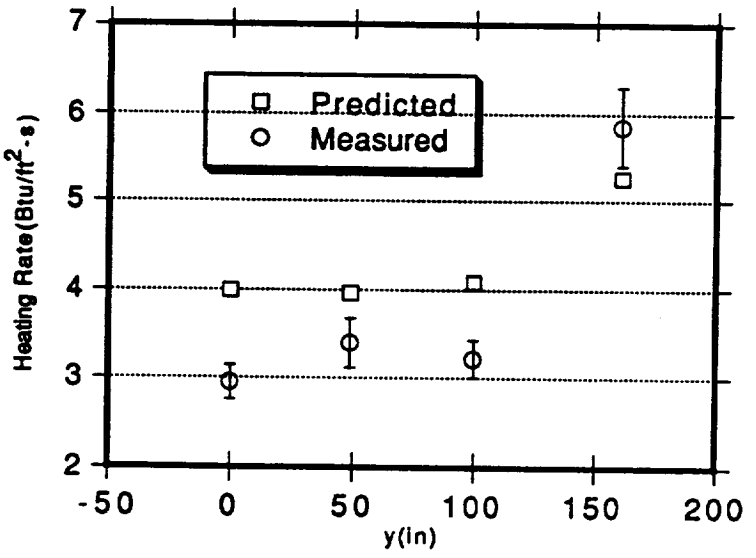

Figure 34: Circumferential heating distribution at $x/L = .6$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg*.

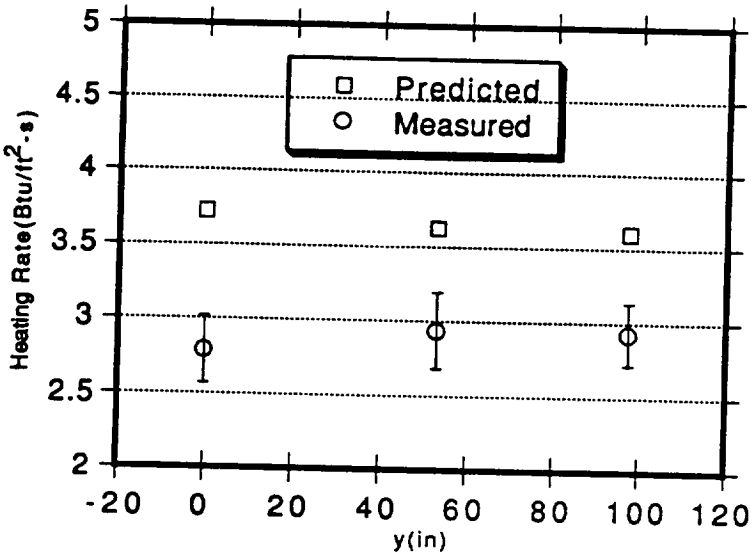

Figure 35: Circumferential heating distribution at $x/L = .7$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

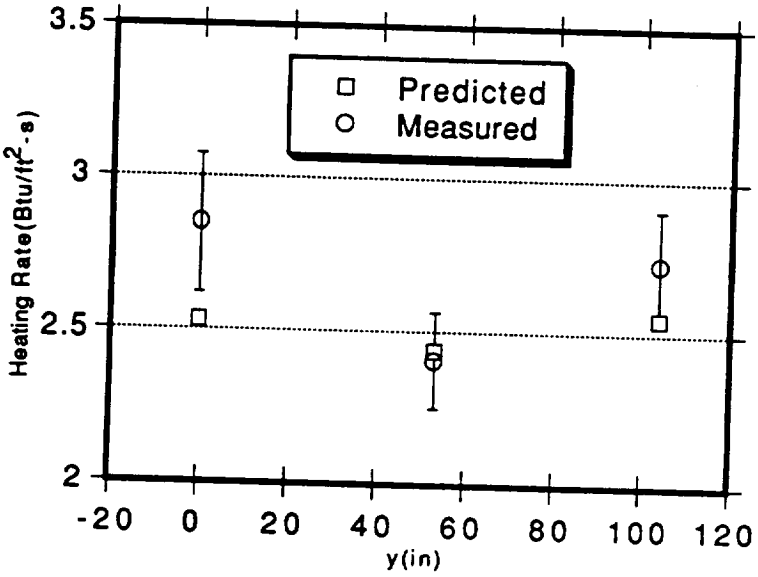

Figure 36: Circumferential heating distribution at $x/L = .9$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

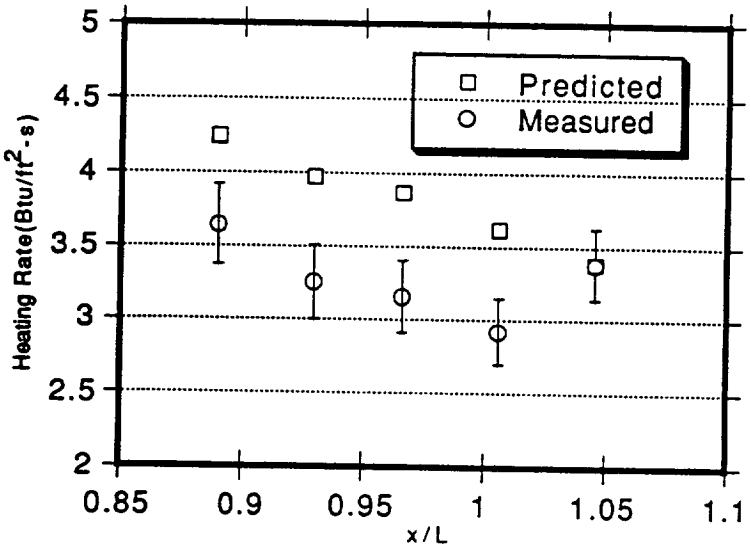

Figure 37: Heating distribution on windward surface at $y = 184.8$ *in.*, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg.*

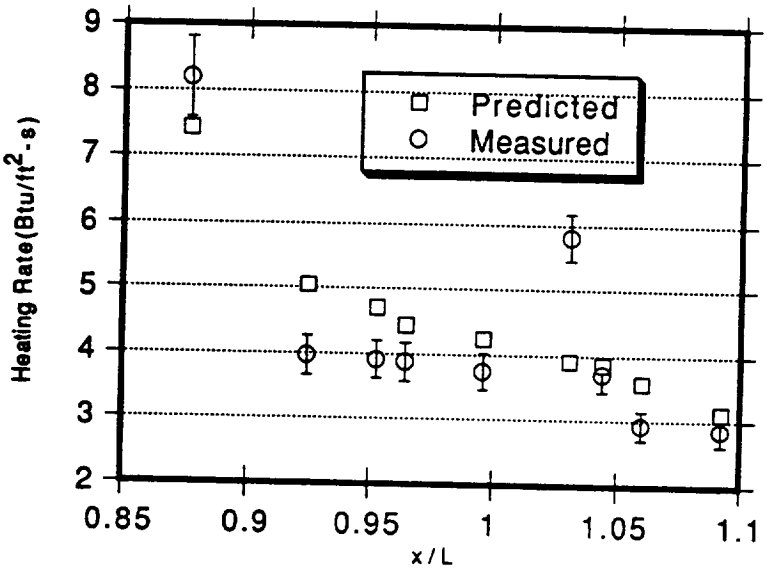

Figure 38: Heating distribution on windward surface at $y = 233.6$ *in.*, STS-2 case, $Mach = 12.86$, $Alpha = 39.7$ *deg.*

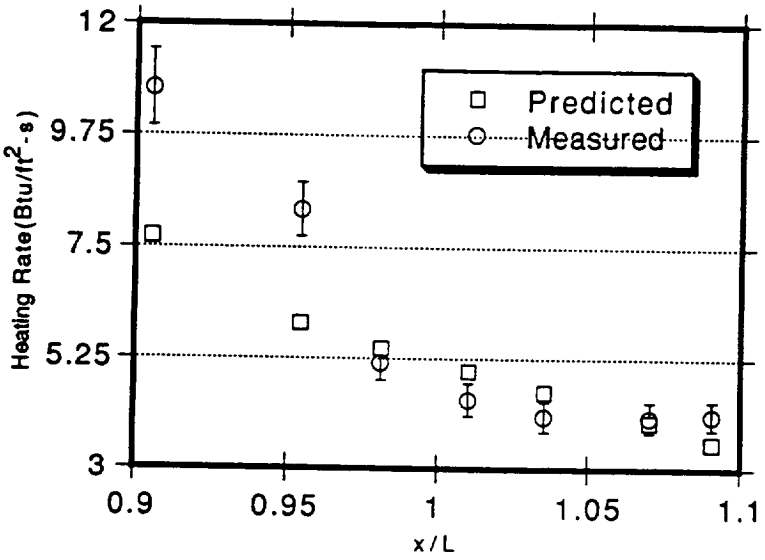

Figure 39: Heating distribution on windward surface at $y = 275.3\ in.$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

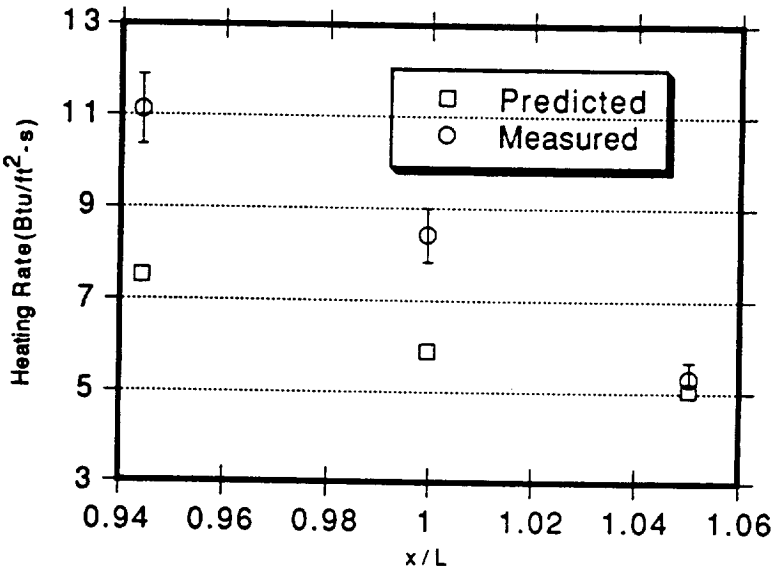

Figure 40: Heating distribution on windward surface at $y = 322.7\ in.$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

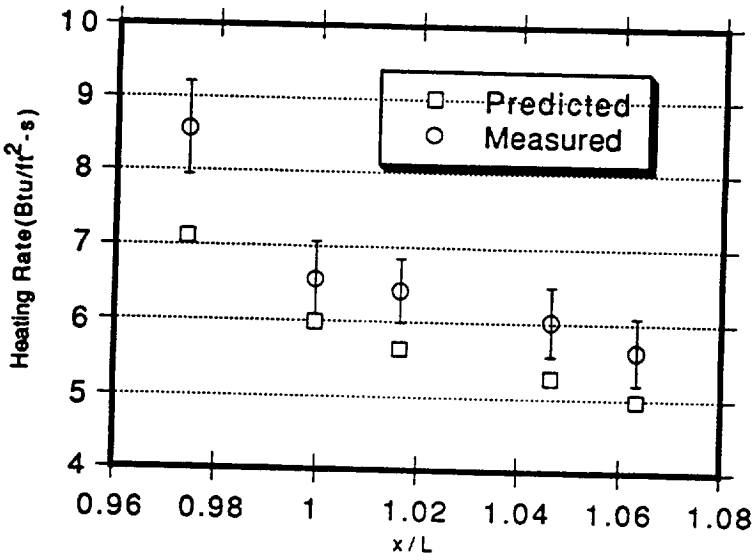

Figure 41: Heating distribution on windward surface at $y = 369.0\ in.$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

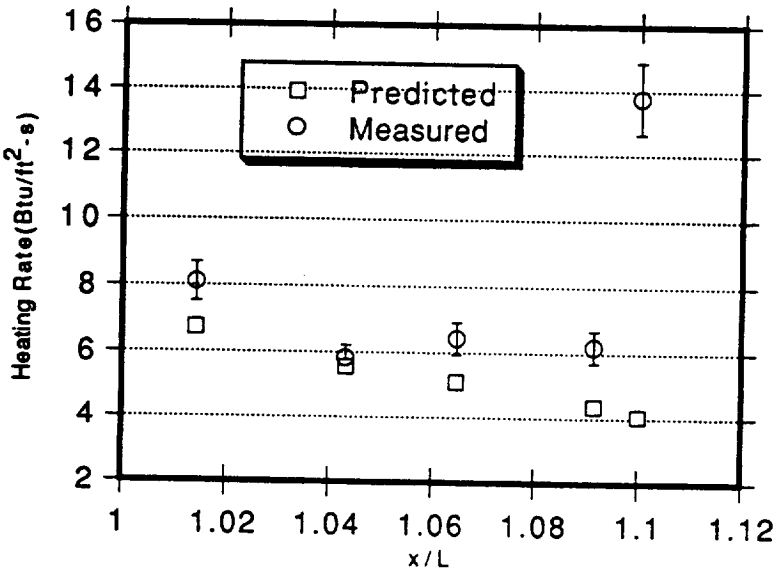

Figure 42: Heating distribution on windward surface at $y = 420.8\ in.$, STS-2 case, $Mach = 12.86$, $Alpha = 39.7\ deg.$

A Appendix A - Input Description

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

record	*description*
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=1kdim), (((z(i,j,k),i=1,idim),j=1,jdim),k=1kdim)
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)
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)
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=1kdim)
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)
6	Pressure file, must be written using FORTRAN UNFORMATTED I/O as follows: write(uint) idim, jdim, kdim write(unit) (((p(i,j,k),i=1,idim),j=1jdim),k=1,kdim)
7	streamline output file, contains following: streamline coordinates, metric coefficient, distance from stagnation point and pressure
8	streamline outfile, suitable for BLIMP input, contains distance from stagnation point, metric coefficient and pressure to stagnation pressure ratio

record	*description*
9	output surface grid file in PLOT3D format(unformatted)
10	an output PLOT3D q file(unformatted), contains surface heating rate, surface radiation equilibrium temperature, boundary layer thickness, momentum thickness, and skin friction
11	input file, contains wall temperature
12	a temporary file
13	an output file, contains location, distance from stagnation point, heating rate and skin friction along a streamline
14	output file, contains radiation equilibrium temperature along a streamline
15	output file, contains surface heating rate at grid point
16	outfile file, contains error message
17	grid scale
18	model scale
19	free stream pressure(psf)
20	free stream temperature (deg *Rankin*)
21	free stream velocity(fps)
22	free stream Mach number
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
28	initial wall temperature
29	angle of attack(deg.)

record	*description*
30	distance from stagnation point where streamline tracing starts(not used in backward streamline tracing)
31	initial step size of integration
32	maximum relative truncation error allowed in variable step Runge-Kutta integration
33	minimum step size allowed in variable step Runge-Kutta integration
34	number of integration steps to be taken(not used in backward tracing)
35	number of streamlines to be generated(not used in backward tracing)
36	initial streamline spreading angle(not used in backward tracing)
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

```
'sts2_nunf.xyz'
'sts2_nunf.q1'
'sts2_nunf.q2'
'sts2_nunf.q3'
'sts2_nunf.q4'
'sts2_nunf.p'
'sts2_ws.dat'
'sts2_wm.dat'
'sts2_basur.xyz'
'sts2_basur.q'
'twall.dat'
'temp.dat'
'htrate.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'
```

Table 1: Sample AA2LCH input file

B Appendix B - AA2LCH Listing

```
      program aa2lch
c
c This program reads in an euler flowfield in Cartesian coordinate,
c and trace the streamine and compute metric coefficient in a backward
c or forward fashion from a given starting location. Third order Runge-Kutta
c integrator is used to perform numerical integration.
c
      common /connty/ i1,j1,k1
      common /stagpt/ xstag,ystag,zstag
      common /psidps/ psi(8),dpdxi(8),dpdeta(8),dpdzta(8)
      common /nomxyz/ xn(151,110),yn(151,110),zn(151,110),xnp,ynp,znp
      common /grdxyz/ 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),xizs(151,110),
     &                etxs(151,110),etys(151,110),etzs(151,110),
     &                flgsm(151),flgsn(151)
      common /dvdwyz/ dvdy(151,110),dvdz(151,110),dwdy(151,110),
     &                dwdz(151,110)
c
c     modification of 8/16/93
c
      common /duxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
     &               dvdx(151,110),dwdx(151,110)
c     end of modification
      common /geom/ dxdy(151,110),dxdz(151,110)
      common /surfpr/ vel,surfn,up,vp,wp,surfp,dxdzp,dxdyp,dxi,det
      common /sfbond/ msp(151),msr(151),nsp(151),nsr(151)
c      common /qxyz/ rho(151,40,110),u(151,40,110),v(151,40,110),
      common /qxyz/ u(151,40,110),v(151,40,110),
     &              w(151,40,110), p(151,40,110)
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
      common /freest/ pinf,tinf,rhoinf,vinf,vninf,cpinf,igas
      common /radeq/ itwall,wallt,irad,eps
      common /turbml/ iturb
      common /tempwl/ twall(1000)
c
c     boundary and momentum thickness
c
      common /bltrn/ deltal,thetl
      dimension y(50), us(151)
      dimension xb(1500),yb(1500),zb(1500),hb(1500)
      dimension q(151,1,110,5)
      dimension ql1(151,1,110,5),ql2(151,1,110,5)
      logical blank,first,back,second,fext
      character*80 gdfile,rhofle,rhoufl,rhovfl,rhowfl,pfile,
     &     sfile,mfile,twfile,xfile,qfile,lqf1,lqf2,
     &     ilfile,outfle,rwfile,shfile,errfle
      character*30 dummy
      character*1 flaura
c
c conversion factors
c
      data cmin,fps,atm,den /2.54, 30.48, 2116.2, .01601846/
      data rtok /1.8/
c
c Stefan-Boltzman constant(w/cm**2/k)
c
      data sigma /5.67e-12/
c
c input surface grid locations
c
      read(5,*) gdfile
      read(5,*) rhofle
      read(5,*) rhoufl
```

```
      read(5,*) rhovfl
      read(5,*) rhowfl
      read(5,*) pfile
      read(5,*) sfile
      read(5,*) mfile
      read(5,*) xfile
      read(5,*) qfile
      read(5,*) twfile
      read(5,*) ilfile
      read(5,*) outfle
      read(5,*) rwfile
      read(5,*) shfile
      read(5,*) errfle
      read(5,*) dummy,scaleg
      read(5,*) dummy,scalem
      scale = scalem / scaleg
c
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
c     input turbulent flag
c
      read(5,*) dummy,iturb
c
c     input radiation equilibrium flag
c
      read(5,*) dummy,irad
c
c     input surface emissivity
c
      read(5,*) dummy,eps
c
c     input wall temperature flag
c
      read(5,*) dummy,itwall
      if(itwall .eq. 0) then
         read(5,*) dummy,wallt
      endif
c
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,xmuinf,xkinf,prinf,gaminf,
     &     arminf,iflag)
      if(ifatal .ne. 0) then
         write(6,*) ' Bad freestream condition encountered'
         stop
      endif
c
c normalization factor for velocity (cm/sec)
c
```

```
      vninf = sqrt(pinf*1.0133e6/rhoinf)
c
      open(unit=7,file=gdfile,form='unformatted',
     &status='old')
c      open(unit=8,file=rhofle,form='unformatted',
c     &status='old')
      open(unit=9,file=rhoufl,form='unformatted',
     &status='old')
      open(unit=10,file=rhovfl,form='unformatted',
     &status='old')
      open(unit=11,file=rhowfl,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=twfile,form='formatted',
     &     status='unknown')
      inquire(file=xfile,exist=fext)
      if(fext) then
        open(unit=16,file=xfile,form='unformatted',status='old')
        open(unit=17,file=qfile,form='unformatted',status='old')
        read(17) idim,jdim,kdim
        read(17) aminf,alp,re,time
        read(17) ((((q(i,j,k,nx),i=1,idim),j=1,jdim),k=1,kdim),
     &  nx=1,5)
      else
        open(unit=16,file=xfile,form='unformatted',status='new')
        open(unit=17,file=qfile,form='unformatted',status='new')
      endif
c
c     additional file
c
      open(unit=15,file=ilfile,form='unformatted',status='unknown')
      open(unit=24,file=outfle,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=errfle,form='formatted',status='unknown')
c
      idim = 0
      jdim = 0
      kdim = 0
   10 continue
      read(7) idim,jdim,kdim
      read(7) (((xyz(i,j,k,1),i=1,idim),j=1,jdim),k=1,kdim),
     &     (((xyz(i,j,k,2),i=1,idim),j=1,jdim),k=1,kdim),
     &     (((xyz(i,j,k,3),i=1,idim),j=1,jdim),k=1,kdim)
c
c     multiply by scale factor
c
      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
 20   continue
c
c     modification of 1/11/94
c
      pi = atan2(1.0,0.) * 2.
      dangle = pi / kdim
      do 30 j = 1,jdim
      do 30 k = 1,kdim
        angle = (k-1)*dangle
```

```
        xyz(1,j,k,2) = 1.e-7 * sin(angle) * scale
        xyz(1,j,k,3) = 1.e-7 * cos(angle) * scale
 30   continue
c end of modification
c
c     read rho
c
c      read(8) idim,jdim,kdim
c      read(8) (((rho(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c
c     read u
c
      read(9) idim,jdim,kdim
      read(9) (((u(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c
c     read v
c
      read(10) idim,jdim,kdim
      read(10) (((v(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c
c     read w
c
      read(11) idim,jdim,kdim
      read(11) (((w(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c
c     read p
c
      read(12) idim,jdim,kdim
      read(12) (((p(i,j,k),i=1,idim),j=1,jdim),k=1,kdim)
c
c alp is angle of attack
c
      read(5,*) alp
      pi = atan2(0.,-1.)
      dtr = pi / 180.
      alpr = alp * dtr
      cosa = cos(alpr)
      sina = sin(alpr)
      ksdim = kdim
      if(alp .lt. 0.) ksdim = 1
c
c locate stagnation point (no yaw)
c
      istag = 0
      do 160 i = 1,idim
      us(i) = u(i,1,ksdim)
 160  continue
c
c locate a pair grid points between which u changes sign.
c
      do 170 i = idim,2,-1
      im1 = i - 1
      if(us(i)*us(im1) .lt. 0.) then
        is = im1
        isp1 = is + 1
        go to 180
      else if(us(i) .eq. 0.) then
        istag = 1
        is = i
        isp1 = is + 1
        go to 180
      else if(us(im1) .eq. 0.) then
        istag = 1
        is = im1
        isp1 = is + 1
        go to 180
```

```
      endif
 170  continue
c
      write(6,*) ' Stagnation point cannot be located'
      stop
 180  continue
      write(6,*) 'is,isp1',is,isp1
c
c find coordinates of stagnation point
c
      xstag = 0.
      ystag = 0.
      zstag = 0.
      if(istag .eq. 1) then
        xstag = xyz(isp1,1,ksdim,1)
        ystag = xyz(isp1,1,ksdim,2)
        zstag = xyz(isp1,1,ksdim,3)
      else
c
c use linear interpolation to find coordinates of
c stagnation point.
c
        ds = sqrt((xyz(is,1,ksdim,1) - xyz(isp1,1,ksdim,1))**2 +
     &            (xyz(is,1,ksdim,3) - xyz(isp1,1,ksdim,3))**2)
        ds1 = - us(is) * ds / (us(isp1) - us(is))
        ds2 = ds - ds1
        write(6,*) ' ds,ds1,ds2',ds,ds1,ds2
c
        xstag = xyz(is,1,ksdim,1) + ds1/ds*
     &          (xyz(isp1,1,ksdim,1)-xyz(is,1,ksdim,1))
        zstag = xyz(is,1,ksdim,3) + ds1/ds*
     &          (xyz(isp1,1,ksdim,3)-xyz(is,1,ksdim,3))
        pstag = p(is,1,ksdim)+ds1/ds*(p(isp1,1,ksdim)-p(is,1,ksdim))
        write(6,*) ' Interpolated stagnation pressure : ',pstag
       endif
c
c compare with maximum pressure
c
      do 190 i = 1,idim
      if(p(i,1,1) .gt. pstag) pstag = p(i,1,1)
      if(p(i,1,kdim) .gt. pstag) pstag = p(i,1,kdim)
 190  continue
      write(6,*) ' Stagnation point is at', xstag,' 0.',zstag
      write(6,*) ' Stagnation pressure is ',pstag
c
c find starting point location
c
      read(5,*) dels
      if(dels .gt. ds1) then
        iss = is
        is = is - 1
        if(is .le. 0) then
          is = iss
          dels = ds1
        endif
      else if(dels .gt. ds2) then
        isp1 = isp1
        isp1 = isp1 + 1
        if(isp1 .gt. idim) then
          isp1 = isp1
          dels = ds2
        endif
      endif
      if(alp .gt. 0.) then
        xstag1 = xstag + dels/ds2 * (xyz(isp1,1,ksdim,1)-xstag)
        zstag1 = zstag + dels/ds2 * (xyz(isp1,1,ksdim,3)-zstag)
```

```
        ystag1 = 0.
        xstag2 = xstag + dels/ds1 * (xyz(is,1,ksdim,1)-xstag)
        zstag2 = zstag + dels/ds1 * (xyz(is,1,ksdim,3)-zstag)
        ystag2 = 0.
      else
        xstag1 = xstag + dels/ds1 * (xyz(is,1,ksdim,1)-xstag)
        zstag1 = zstag + dels/ds1 * (xyz(is,1,ksdim,3)-zstag)
        ystag1 = 0.
        xstag2 = xstag + dels/ds2 * (xyz(ispl,1,ksdim,1)-xstag)
        zstag2 = zstag + dels/ds2 * (xyz(ispl,1,ksdim,3)-zstag)
        ystag2 = 0.
      endif
      if(is .ne. iss) is = iss
      if(isspl .ne. ispl) ispl = isspl
c
c set up boundary values for each element
c
      msper = 0
      nsper = 0
      ms1 = 1
      ns1 = 1
      mdim = idim
      ndim = kdim
      call bounds(mdim,ndim,ms1,mdim,ns1,ndim,msper,nsper)
c
c compute surface metrics
c
      call surmet(mdim,ndim,ms1,mdim,ns1,ndim,msper,nsper)
c
c n is number of functions to be integrated
c
c      n = 5
c
c     modifiation of 8/16/93
c
       n = 6
c
c h is initial step size
c
      read(5,*) saveh
      hsave = saveh
c
      back = .false.
c
c emax is maximum relative truncation error
c
      read(5,*) emax
c
c hmin is minimum step size allowed
c
      read(5,*) hmin
c
c nstep is the number of steps of integration
c
      read(5,*) nstep
c
c ns1 is total number of streamlines to be generated
c
      read(5,*) ns1
c
c angi is initial spreading angle between streamlines
c
      read(5,*) angi
c
c read in backward streamline tracing flag
c
```

```
      read(5,*) dummy,iback
c
c     itype = 1 : multiple streamline, imin,imax,kmin,kmax required
c     itype = 0 : single streamline
c
      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
c
         open(unit=21,file=lqf1,form='unformatted',status='old')
         open(unit=22,file=lqf2,form='unformatted',status='old')
c
         read(21) ildim,jldim,kldim
         read(21) dum1,dum2,dum3,dum4
         read(21) (((ql1(i,1,k,nx),i=1,ildim),k=1,kldim),nx=1,5)
c
         read(22) ildim,jldim,kldim
         read(22) dum1,dum2,dum3,dum4
         read(22) (((ql2(i,1,k,nx),i=1,ildim),k=1,kldim),nx=1,5)
      endif
c      read(5,*) dummy,nbstep
      if(iback .eq. 1) then
        back = .true.
      else
c
c find coordinates of center
c
        xc = xstag + dels
        zc = zstag + dels * tan(alpr)
        yc = 0.
c
        sinthe = cos(alpr)
        costhe = sqrt(1.-sinthe*sinthe)
        rr = dels/cosa
c
c starting location
c
c      read(5,*,end=9999) isav,jsav,ksav,xf,yf,zf
        second = .false.
        igsav = 1
        issav = 1
        asav = 0.
        bsav = 0.
        gsav = 0.
        ifatal = 0
c
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
c
c     determin number of streamlines to be traced
c
```

```
      if(itype .ne. 0) then
         nsl = (ibmax-ibmin+1) * (kbmax-kbmin+1)
         istart = ibmin
         kstart = kbmin
      endif
      do 300 nl = 1,nsl
      second = .false.
      if(iback .eq. 1) back = .true.
      if(back) then
        if(itype .eq. 0) then
          xin = xst
          yin = yst
          zin = zst
        else
          xin = xyz(istart,1,kstart,1)
          yin = xyz(istart,1,kstart,2)
          zin = xyz(istart,1,kstart,3)
        endif
      else
        first = .false.
        if(nl .eq. 1) first = .true.
c
c initial position
c
        if(nl .eq. 1) then
          xin = xstag1
          yin = ystag1
          zin = zstag1
          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
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
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 = xstag2
            yin = ystag2
            xin = zstag2
            if(alp .ge. 0.) then
              xin = xstag1
              yin = ystag1
              zin = zstag1
            endif
          endif
        endif
      endif
c      xin = xs(isp1+1,kdim-nl+1)
c      yin = ys(isp1+1,kdim-nl+1)
c      zin = zs(isp1+1,kdim-nl+1)
```

```
 195  continue
      write(6,*) ' xin,yin,zin ',xin,yin,zin
      ms = 1
      me = idim
      ns = 1
      ne = kdim
      call fndnrp(ms,me,ns,ne,xin,yin,zin)
      istat = 0
      delxi = 0.
      delet = 0.
      call quads(delxi,delet,xin,yin,zin,istat)
      if(istat .gt. 1) then
        write(6,*) ' Search failed, istat = ',istat
        write(40,*) ' Location ',istart,kstart,' failed'
        go to 240
      endif
      i1 = isav
      k1 = ksav
c      if(nl .eq. 1) then
c        if(k1 .eq. 1 .or. k1 .eq. kdim) then
c          delet = 0.
c        endif
c      endif
      c11 = delxi * delet
      c10 = delxi - c11
      c01 = delet - c11
      c00 = 1. - delxi - delet + c11
      ip1 = msp(i1)
      kp1 = nsp(k1)
      uin = u(i1,1,k1)*c00 + u(ip1,1,k1)*c10 + u(ip1,1,kp1)*c11
     &    + u(i1,1,kp1)*c01
      vin = v(i1,1,k1)*c00 + v(ip1,1,k1)*c10 + v(ip1,1,kp1)*c11
     &    + v(i1,1,kp1)*c01
      win = w(i1,1,k1)*c00 + w(ip1,1,k1)*c10 + w(ip1,1,kp1)*c11
     &    + w(i1,1,kp1)*c01
      pin = p(i1,1,k1)*c00 + p(ip1,1,k1)*c10 + p(ip1,1,kp1)*c11
     &    + p(i1,1,kp1)*c01
      dxdyin = dxdy(i1,k1)*c00 + dxdy(ip1,k1)*c10 +
     &         dxdy(ip1,kp1)*c11 + dxdy(i1,kp1)*c01
      dxdzin = dxdz(i1,k1)*c00 + dxdz(ip1,k1)*c10 +
     &         dxdz(ip1,kp1)*c11 + dxdz(i1,kp1)*c01
c
c find norm vector
c
      xnin = xn(i1,k1)*c00 + xn(ip1,k1)*c10 +
     &       xn(ip1,kp1)*c11 + xn(i1,kp1)*c01
      ynin = yn(i1,k1)*c00 + yn(ip1,k1)*c10 +
     &       yn(ip1,kp1)*c11 + yn(i1,kp1)*c01
      znin = zn(i1,k1)*c00 + zn(ip1,k1)*c10 +
     &     zn(ip1,kp1)*c11 + zn(i1,kp1)*c01
c
c     interpolate boundary layer transition parameters
c
      if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
        repf = ql1(i1,1,k1,4)*c00 + ql1(ip1,1,k1,4)*c10 +
     &       ql1(ip1,1,kp1,4)*c11 + ql1(i1,1,kp1,4)*c01
c
        thkm = ql1(i1,1,k1,2)*c00 + ql1(ip1,1,k1,2)*c10 +
     &       ql1(ip1,1,kp1,2)*c11 + ql1(i1,1,kp1,2)*c01
c
        thkd = ql1(i1,1,k1,3)*c00 + ql1(ip1,1,k1,3)*c10 +
     &       ql1(ip1,1,kp1,3)*c11 + ql1(i1,1,kp1,3)*c01
c
        rek = ql2(i1,1,k1,3)*c00 + ql2(ip1,1,k1,3)*c10 +
     &       ql2(ip1,1,kp1,3)*c11 + ql2(i1,1,kp1,3)*c01
c
```

```
        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
c initial metric coefficient
c
      hmet = dist
c      hmet = sqrt((xin-xstag)**2+(yin-ystag)**2+(zin-zstag)**2)
c
c initial surface norm vector magnitude
c
      surfn = sqrt(1. + dxdyin**2 + dxdzin**2)
      vel = sqrt(uin**2+vin**2+win**2)
c
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 * (xnin*win/vel - znin*uin/vel)
      y(5) = hmet * (xnin*vin/vel - ynin*uin/vel)
c
c     modification of 8/16/93
c
      y(6) = hmet * (ynin*win/vel - znin*vin/vel)
c end of modification
c
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)
c      hmet1 = -y(4) / (xnin*win/vel - znin*uin/vel)
c      hmet2 = y(5) / (xnin*vin/vel - ynin*uin/vel)
      hmet=(y(5)-y(4))/(xnin*(vin+win)/vel-uin/vel*(ynin+znin))
c
c     modification of 8/16/93
c
      hmet = sqrt(y(4)**2 + y(5)**2 + y(6)**2)
c end of modification
c      hmet = surfn/(vel-uin/vel*(uin-vin*dxdyin-win*dxdzin))*
c     &       (vin*y(5) - win*y(4))
c      hmet1 = -surfn*y(4)/(win/vel+uin/vel*dxdzin)
c      hmet2 = surfn*y(5)/(vin/vel+uin/vel*dxdyin)
c      if(xf .ne. 0. .and. yf .ne. 0. .and. zf .ne. 0.) then
c        x = xf
c        y(1) = yf
c        y(2) = zf
c      endif
c
c output streamline locations
c
      write(13,*) ' Streamline No. ',nl
      write(14,*) ' Streamline No. ',nl
      write(6,*) ' Streamline No. ',nl
      write(13,9001) y(1),y(2),y(3),dist,hmet,pin
      if(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(1)-xnin,y(2)-ynin,y(3)-znin
       write(6,*) y(1),y(2),y(3),y(4),y(5),surfn,vel,hmet,pin
c
c integration loop
c
       h = hsave
       hmin = hmin
       write(6,*) ' real step size',h,hmin
       xsave = y(1)
       ysave = y(2)
       zsave = y(3)
c      do 250 i = 1,nstep
 200   continue
       if(second) then
          i = nstep
       else
          i = 1
       endif
 210   continue
       if(second) then
         y(1) = xb(i)
         y(2) = yb(i)
         y(3) = zb(i)
         if(i .eq. 2) then
           h = hb(i)
         else
           h = hb(i) - hb(i-1)
         endif
c         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
c         if(itype .eq. 1) 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
           i = i + 1
           nstep = i
c         else
c            if(i .gt. nstep) go to 220
c            i = i + 1
c         endif
         xb(i) = y(1)
         yb(i) = y(2)
         zb(i) = y(3)
         hb(i) = x
       else
          i = i - 1
       endif
c         write(6,*) ' step ',i
c         write(6,*) isptr,xb(isptr),yb(isptr),zb(isptr),hb(isptr)
       write(6,*) 'vp,y(5),wp,y(4)',vp,y(5),wp,y(4),surfn/vel
       hmet = surfn / vel * (vp*y(5) - wp*y(4))
       hmet = (vp*y(5) - wp*y(4))/(vel*xnp)
```

```
c       hmet1 = -y(4) / (xnp*wp/vel - znp*up/vel)
c       hmet2 = y(5) / (xnp*vp/vel - ynp*up/vel)
        hmet=(y(5)-y(4))/(xnp*(vp+wp)/vel-up/vel*(ynp+znp))
c
c     modification of 8/16/93
c
        hmet = sqrt(y(4)**2 + y(5)**2 + y(6)**2)
c end of modification
c       hmet = surfn/(vel-up/vel*(up-vp*dxdyp-wp*dxdzp))*
c     &       (vp*y(5) - wp*y(4))
c       hmet1 = -surfn*y(4)/(wp/vel+up/vel*dxdzp)
c       hmet2 = surfn*y(5)/(vp/vel+up/vel*dxdyp)
        dist = dist + sqrt((y(1)-xsave)**2 + (y(2)-ysave)**2
     & + (y(3)-zsave)**2)
        xsave = y(1)
        ysave = y(2)
        zsave = y(3)
        hs = h
c
c     interpolate boundary layer transition parameters
c
        if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
          i1 = isav
          k1 = ksav
          c11 = dxi * det
          c10 = dxi - c11
          c01 = det - c11
          c00 = 1. - dxi - det + c11
          ip1 = msp(i1)
          kp1 = nsp(k1)
          repf = ql1(i1,1,k1,4)*c00 + ql1(ip1,1,k1,4)*c10 +
     &       ql1(ip1,1,kp1,4)*c11 + ql1(i1,1,kp1,4)*c01
c
          thkm = ql1(i1,1,k1,2)*c00 + ql1(ip1,1,k1,2)*c10 +
     &       ql1(ip1,1,kp1,2)*c11 + ql1(i1,1,kp1,2)*c01
c
          thkd = ql1(i1,1,k1,3)*c00 + ql1(ip1,1,k1,3)*c10 +
     &       ql1(ip1,1,kp1,3)*c11 + ql1(i1,1,kp1,3)*c01
c
          rek = ql2(i1,1,k1,3)*c00 + ql2(ip1,1,k1,3)*c10 +
     &       ql2(ip1,1,kp1,3)*c11 + ql2(i1,1,kp1,3)*c01
c
          rem = repf * thkm / 12.
          red = repf * thkd / 12.
          rex = repf * dist / 12.
          write(26,9001) thkm,thkd,rem,red,rek,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
c       write(17,9010) y(1),y(2),y(3),y(1)-xnp,y(2)-ynp,y(3)-znp
        write(6,*) ' i,k = ',isav,ksav
        write(6,*) y(1),y(2),y(3),y(4),y(5),dist,hmet,surfp
        if(.not. back) then
           if(i .le. 1) go to 230
        endif
        go to 210
  220   continue
        if(back) then
          rewind 14
          rewind 13
          rewind 15
          if(flaura .eq. 'y' .or. flaura .eq. 'Y') then
             rewind 26
          endif
          dist=sqrt((y(1)-xstag)**2+(y(2)-ystag)**2+(y(3)-zstag)**2)
```

```
        write(13,*) ' Streamline No. ',nl
        write(14,*) ' Streamline No. ',nl
        write(6,*) ' Streamline No. ',nl
        back = .false.
        second = .true.
        xin = xsave
        yin = ysave
        zin = zsave
        hsave = hs
        go to 195
      endif
 230  continue
c
c     call subroutine heat to evaluate convective heating
c
      call heat(qw,qwbtu,wltemp,cf)
      rewind 13
      rewind 14
      rewind 15
      rewind 6
      rewind 24
      rewind 25
      write(30,*) istart,kstart,qw,qwbtu
      if(ifatal .ne. 0) then
         write(40,*) ' Location ',istart,kstart,' failed'
         ifatal = 0
c         go to 240
      endif
      if(itype .ne. 0) then
        q(istart,1,kstart,1) = qwbtu
        q(istart,1,kstart,2) = wltemp
        q(istart,1,kstart,3) = deltal
        q(istart,1,kstart,4) = thetl
        q(istart,1,kstart,5) = cf
      endif
 240  continue
      if(istart .eq. ibmax) then
         if(kstart .eq. kbmax) then
c
c     output plot3d file
c
            jdim3d = 1
            if(.not. fext) then
            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)
            endif
c
            rewind 17
            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)
            go to 9999
         else
            kstart = kstart + 1
         endif
      else
         kstart = kstart + 1
         if(kstart .gt. kbmax) then
            kstart = kbmin
            istart = istart + 1
         endif
      endif
      hsave = saveh
```

```
 300  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)
c
         write(17) idim,jdim3d,kdim
         write(17) aminf,alp,re,time
         write(17) ((((q(i,j,k,nx),i=1,idim),j=1,jdim3d),k=1,kdim),
     &        nx=1,5)
      endif
c
 9001 format(6(1x,e11.4,1x))
 9002 format(3(1x,e15.6,',',1x))
 9010  format(6(e12.5,1x))
 9999 continue
      stop
      end
c
c subroutine rk34
c
      subroutine rk34(n,x,y,h,e,hmin)
c
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
c
      dimension y(50), ytemp(50), yhat(50), f(50), a(50),
     &b(50),c(50),d(50)
      logical second,first,back
c
      data c0,c2,c3,ch0,ch2,ch3,ch4,a1,a2,a3,b10,b20,b21,
     & b30,b31,b32,b40,b42,b43/.1612244898, .5998345284,
     & .2389409818, .1557823129, .6205184777, .1681436539,
     & .0555555556, .2857142857, .4666666667, .9210526316,
     & .2857142857, .0855555556, .3811111111,
     & .5574792244, -1.406455023, 1.770028430, .1612244898,
     & .5998345284, .2389409818/
c
    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
      return
   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
      go to 9
   12 continue
      h = h * 0.5
      write(6,*)' h = ',h
      go to 1
      end
c
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(151,40,110,3),idim,jdim,kdim,
     &                iblank(151,40,110), blank,isubs(2),
     &                jsubs(2),ksubs(2)
      common /qxyz/ u(151,40,110),v(151,40,110),
     &              w(151,40,110), p(151,40,110)
      common /dvdwyz/ dvdy(151,110),dvdz(151,110),dwdy(151,110),
     &                dwdz(151,110)
c
c     modification of 8/16/93
c
      common /duxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
     &               dvdx(151,110),dwdx(151,110)
c     end of modification
```

```
      common /geom/ dxdy(151,110),dxdz(151,110)
      common /surfpr/ vel,surfn,up,vp,wp,surfp,dxdzp,dxdyp,dxi,det
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
      logical first,back,second
c
      dimension y(50),f(50)
c
      sx = y(1)
      sy = y(2)
      sz = y(3)
c
c surface interpolation
c
      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
c      ns = 1
c      ne = kdim
      call fndnrp(ms,me,ns,ne,sx,sy,sz)
c
      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
      i1 = isav
      k1 = ksav
c      if(first) then
c        if(k1 .eq. 1 .or. k1 .eq. kdim) then
c          delet = 0.
c        endif
c      endif
      dxi = delxi
      det = delet
      c11 = delxi * delet
      c10 = delxi - c11
      c01 = delet - c11
      c00 = 1. - delxi - delet + c11
      ip1 = msp(i1)
      kp1 = nsp(k1)
      up = u(i1,1,k1)*c00 + u(ip1,1,k1)*c10 + u(ip1,1,kp1)*c11
     &   + u(i1,1,kp1)*c01
      vp = v(i1,1,k1)*c00 + v(ip1,1,k1)*c10 + v(ip1,1,kp1)*c11
     &   + v(i1,1,kp1)*c01
      wp = w(i1,1,k1)*c00 + w(ip1,1,k1)*c10 + w(ip1,1,kp1)*c11
     &   + w(i1,1,kp1)*c01
      surfp = p(i1,1,k1)*c00 + p(ip1,1,k1)*c10 + p(ip1,1,kp1)*c11
     &   + p(i1,1,kp1)*c01
      dvdyp = dvdy(i1,k1)*c00 + dvdy(ip1,k1)*c10 +
     &        dvdy(ip1,kp1)*c11 + dvdy(i1,kp1)*c01
      dvdzp = dvdz(i1,k1)*c00 + dvdz(ip1,k1)*c10 +
     &        dvdz(ip1,kp1)*c11 + dvdz(i1,kp1)*c01
      dwdyp = dwdy(i1,k1)*c00 + dwdy(ip1,k1)*c10 +
     &        dwdy(ip1,kp1)*c11 + dwdy(i1,kp1)*c01
      dwdzp = dwdz(i1,k1)*c00 + dwdz(ip1,k1)*c10 +
     &        dwdz(ip1,kp1)*c11 + dwdz(i1,kp1)*c01
```

```
c
c     modification of 8/16/93
c
      dudxp = dudx(i1,k1)*c00 + dudx(ip1,k1)*c10 +
     &        dudx(ip1,kp1)*c11 + dudx(i1,kp1)*c01
      dudyp = dudy(i1,k1)*c00 + dudy(ip1,k1)*c10 +
     &        dudy(ip1,kp1)*c11 + dudy(i1,kp1)*c01
      dudzp = dudz(i1,k1)*c00 + dudz(ip1,k1)*c10 +
     &        dudz(ip1,kp1)*c11 + dudz(i1,kp1)*c01
      dvdxp = dvdx(i1,k1)*c00 + dvdx(ip1,k1)*c10 +
     &        dvdx(ip1,kp1)*c11 + dvdx(i1,kp1)*c01
      dwdxp = dwdx(i1,k1)*c00 + dwdx(ip1,k1)*c10 +
     &        dwdx(ip1,kp1)*c11 + dwdx(i1,kp1)*c01
c
      dxdyp = dxdy(i1,k1)*c00 + dxdy(ip1,k1)*c10 +
     &        dxdy(ip1,kp1)*c11 + dxdy(i1,kp1)*c01
      dxdzp = dxdz(i1,k1)*c00 + dxdz(ip1,k1)*c10 +
     &        dxdz(ip1,kp1)*c11 + dxdz(i1,kp1)*c01
c
c surface norm vector
c
      xnp = xn(i1,k1)*c00 + xn(ip1,k1)*c10 +
     &      xn(ip1,kp1)*c11 + xn(i1,kp1)*c01
      ynp = yn(i1,k1)*c00 + yn(ip1,k1)*c10 +
     &      yn(ip1,kp1)*c11 + yn(i1,kp1)*c01
      znp = zn(i1,k1)*c00 + zn(ip1,k1)*c10 +
     &      zn(ip1,kp1)*c11 + zn(i1,kp1)*c01
c
      vel = sqrt(up*up + vp*vp + wp*wp)
      surfn = sqrt(1. + dxdyp*dxdyp + dxdzp*dxdzp)
c
      f(1) = up
      f(2) = vp
      f(3) = wp
c      f(4) = dvdyp * y(4) + dvdzp * y(5)
c      f(5) = dwdyp * y(4) + dwdzp * y(5)
      f(4) = dvdxp * y(6) + dvdyp * y(4) + dvdzp * y(5)
      f(5) = dwdxp * y(6) + dwdyp * y(4) + dwdzp * y(5)
      f(6) = dudxp * y(6) + dudyp * y(4) + dudzp * y(5)
      if(back) then
        f(1) = -f(1)
        f(2) = -f(2)
        f(3) = -f(3)
      endif
c      write(6,*) ' f(1),f(2) = ',f(1),f(2),f(3),f(4),f(5)
      return
      end
c
      subroutine bounds ( mdim,ndim,ms,me,ns,ne,
     &                    msper,nsper)
      common /sfbond/ msp(151),msr(151),nsp(151),nsr(151)
c
c  Initialize arrays.
c
      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
c
c  Set up +/- index arrays.
c
      do 20 m = ms,me
```

```
            msp(m) = m+1
            msr(m) = m-1
   20       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
   21       continue
         if (nsper.eq.0) then
            nsp(ne) = ne
            nsr(ns) = ns
         else
            nsp(ne) = ns+1
            nsr(ns) = ne-1
         endif

         return
         end
c
         subroutine surmet ( mdim,ndim,ms,me,ns,ne,msper,nsper)
c
c  Compute metrics for defining surface grid.
c
         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),xizs(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)
c
c     modification of 8/16/93
c
         common /duxyz/ dudx(151,110),dudy(151,110),dudz(151,110),
        &               dvdx(151,110),dwdx(151,110)
c     end of modification
         common /geom/ dxdy(151,110),dxdz(151,110)
         dimension jacm(151),jacn(151)
c
          njac = 0
c
c  Initialize arrays.
c
         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.
            xizs(m,n) = 0.
            etxs(m,n) = 0.
            etys(m,n) = 0.
            etzs(m,n) = 0.
            dvdy(m,n) = 0.
            dvdz(m,n) = 0.
            dwdy(m,n) = 0.
            dwdz(m,n) = 0.
            dxdy(m,n) = 0.
            dxdz(m,n) = 0.
   12       continue
c
c  Set up weight functions.
c
         do 20 m = ms,me
            flgsm(m) = .5
   20       continue
         if (msper.eq.0) then
            flgsm(me) = 1.
            flgsm(ms) = 1.
         endif

         do 21 n = ns,ne
            flgsn(n) = .5
   21       continue
         if (nsper.eq.0) then
            flgsn(ns) = 1.
            flgsn(ne) = 1.
         endif
c
c  Compute metrics.
c
         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,1,n,1) - xyz(mr,1,n,1)) * flgsm(m)
               yxi = (xyz(mp,1,n,2) - xyz(mr,1,n,2)) * flgsm(m)
               zxi = (xyz(mp,1,n,3) - xyz(mr,1,n,3)) * flgsm(m)
               vxi = (v(mp,1,n) - v(mr,1,n)) * flgsm(m)
               wxi = (w(mp,1,n) - w(mr,1,n)) * flgsm(m)
               xet = (xyz(m,1,np,1) - xyz(m,1,nr,1)) * flgsn(n)
               yet = (xyz(m,1,np,2) - xyz(m,1,nr,2)) * flgsn(n)
               zet = (xyz(m,1,np,3) - xyz(m,1,nr,3)) * flgsn(n)
               vet = (v(m,1,np) - v(m,1,nr)) * flgsn(n)
               wet = (w(m,1,np) - w(m,1,nr)) * flgsn(n)
c
c     modification of 8/16/93
c
               uxi = (u(mp,1,n) - u(mr,1,n)) * flgsm(m)
               uet = (u(m,1,np) - u(m,1,nr)) * flgsn(n)
c     end of modification
c
c debug print
c
               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
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 .ne. ns .and. n .ne. ne)then
      x1 = xyz(m-1,1,n+1,1) - xyz(m,1,n,1)
      y1 = xyz(m-1,1,n+1,2) - xyz(m,1,n,2)
      z1 = xyz(m-1,1,n+1,3) - xyz(m,1,n,3)
      x2 = xyz(m,1,n+1,1) - xyz(m-1,1,n,1)
      y2 = xyz(m,1,n+1,2) - xyz(m-1,1,n,2)
      z2 = xyz(m,1,n+1,3) - xyz(m-1,1,n,3)
      xn1 = y1*z2 - y2*z1
      yn1 = x2*z1 - x1*z2
      zn1 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
      xn1 = xn1*sqri
      yn1 = yn1*sqri
      zn1 = zn1*sqri
c
      x1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1)
      y1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2)
      z1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3)
      x2 = xyz(m-1,1,n,1) - xyz(m,1,n-1,1)
      y2 = xyz(m-1,1,n,2) - xyz(m,1,n-1,2)
      z2 = xyz(m-1,1,n,3) - xyz(m,1,n-1,3)
      xn2 = y1*z2 - y2*z1
      yn2 = x2*z1 - x1*z2
      zn2 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
      xn2 = xn2*sqri
      yn2 = yn2*sqri
      zn2 = zn2*sqri
c
      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)
      xn1 = y1*z2 - y2*z1
      yn1 = x2*z1 - x1*z2
      zn1 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
      xn1 = xn1*sqri
      yn1 = yn1*sqri
      zn1 = zn1*sqri
c
      x1 = xyz(m+1,1,n+1,1) - xyz(m,1,n,1)
      y1 = xyz(m+1,1,n+1,2) - xyz(m,1,n,2)
      z1 = xyz(m+1,1,n+1,3) - xyz(m,1,n,3)
      x2 = xyz(m+1,1,n,1) - xyz(m,1,n+1,1)
      y2 = xyz(m+1,1,n,2) - xyz(m,1,n+1,2)
      z2 = xyz(m+1,1,n,3) - xyz(m,1,n+1,3)
      xn2 = y1*z2 - y2*z1
      yn2 = x2*z1 - x1*z2
      zn2 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
      xn2 = xn2*sqri
      yn2 = yn2*sqri
```

```
      zn2 = zn2*sqri
      xnn = 0.5*(xn1+xn2)
      ynn = 0.5*(yn1+yn2)
      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 .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)
      xn1 = y1*z2 - y2*z1
      yn1 = x2*z1 - x1*z2
      zn1 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
      xn1 = xn1*sqri
      yn1 = yn1*sqri
      zn1 = zn1*sqri
c
      x1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1)
      y1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2)
      z1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3)
      x2 = xyz(m-1,1,n,1) - xyz(m,1,n-1,1)
      y2 = xyz(m-1,1,n,2) - xyz(m,1,n-1,2)
      z2 = xyz(m-1,1,n,3) - xyz(m,1,n-1,3)
      xn2 = y1*z2 - y2*z1
      yn2 = x2*z1 - x1*z2
      zn2 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
      xn2 = xn2*sqri
      yn2 = yn2*sqri
      zn2 = zn2*sqri
      xnn = 0.5*(xn1+xn2)
      ynn = 0.5*(yn1+yn2)
      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-1,1,n,1) - xyz(m,1,n,1)
      y1 = xyz(m-1,1,n,2) - xyz(m,1,n,2)
      z1 = xyz(m-1,1,n,3) - xyz(m,1,n,3)
      x2 = xyz(m,1,n+1,1) - xyz(m,1,n,1)
      y2 = xyz(m,1,n+1,2) - xyz(m,1,n,2)
      z2 = xyz(m,1,n+1,3) - xyz(m,1,n,3)
      xn1 = y1*z2 - y2*z1
      yn1 = x2*z1 - x1*z2
      yn1 = 0.
      zn1 = x1*y2 - x2*y1
      sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
      xn(m,n) = 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-1,1,n,1) - xyz(m,1,n,1)
      y2 = xyz(m-1,1,n,2) - xyz(m,1,n,2)
```

```
            z2 = xyz(m-1,1,n,3) - xyz(m,1,n,3)
            xn1 = y1*z2 - y2*z1
            yn1 = x2*z1 - x1*z2
            yn1 = 0.
            zn1 = x1*y2 - x2*y1
            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
            x1 = xyz(m-1,1,n-1,1) - xyz(m,1,n,1)
            y1 = xyz(m-1,1,n-1,2) - xyz(m,1,n,2)
            z1 = xyz(m-1,1,n-1,3) - xyz(m,1,n,3)
            x2 = xyz(m-1,1,n,1) - xyz(m,1,n-1,1)
            y2 = xyz(m-1,1,n,2) - xyz(m,1,n-1,2)
            z2 = xyz(m-1,1,n,3) - xyz(m,1,n-1,3)
            xn1 = y1*z2 - y2*z1
            yn1 = x2*z1 - x1*z2
            zn1 = x1*y2 - x2*y1
            sqri = 1./sqrt(xn1*xn1 + yn1*yn1 + zn1*zn1)
            xn1 = xn1*sqri
            yn1 = yn1*sqri
            zn1 = zn1*sqri
c
            x1 = xyz(m-1,1,n+1,1) - xyz(m,1,n,1)
            y1 = xyz(m-1,1,n+1,2) - xyz(m,1,n,2)
            z1 = xyz(m-1,1,n+1,3) - xyz(m,1,n,3)
            x2 = xyz(m,1,n+1,1) - xyz(m-1,1,n,1)
            y2 = xyz(m,1,n+1,2) - xyz(m-1,1,n,2)
            z2 = xyz(m,1,n+1,3) - xyz(m-1,1,n,3)
            xn2 = y1*z2 - y2*z1
            yn2 = x2*z1 - x1*z2
            zn2 = x1*y2 - x2*y1
            sqri = 1./sqrt(xn2*xn2 + yn2*yn2 + zn2*zn2)
            xn2 = xn2*sqri
            yn2 = yn2*sqri
            zn2 = zn2*sqri
c
            x1 = xyz(m+1,1,n+1,1) - xyz(m,1,n,1)
            y1 = xyz(m+1,1,n+1,2) - xyz(m,1,n,2)
            z1 = xyz(m+1,1,n+1,3) - xyz(m,1,n,3)
            x2 = xyz(m+1,1,n,1) - xyz(m,1,n+1,1)
            y2 = xyz(m+1,1,n,2) - xyz(m,1,n+1,2)
            z2 = xyz(m+1,1,n,3) - xyz(m,1,n+1,3)
            xn3 = y1*z2 - y2*z1
            yn3 = x2*z1 - x1*z2
            zn3 = x1*y2 - x2*y1
            sqri = 1./sqrt(xn3*xn3 + yn3*yn3 + zn3*zn3)
            xn3 = xn3*sqri
            yn3 = yn3*sqri
            zn3 = zn3*sqri
c
            x1 = xyz(m+1,1,n-1,1) - xyz(m,1,n,1)
            y1 = xyz(m+1,1,n-1,2) - xyz(m,1,n,2)
            z1 = xyz(m+1,1,n-1,3) - xyz(m,1,n,3)
            x2 = xyz(m,1,n-1,1) - xyz(m+1,1,n,1)
            y2 = xyz(m,1,n-1,2) - xyz(m+1,1,n,2)
            z2 = xyz(m,1,n-1,3) - xyz(m+1,1,n,3)
            xn4 = y1*z2 - y2*z1
            yn4 = x2*z1 - x1*z2
            zn4 = x1*y2 - x2*y1
            sqri = 1./sqrt(xn4*xn4 + yn4*yn4 + zn4*zn4)
            xn4 = xn4*sqri
            yn4 = yn4*sqri
            zn4 = zn4*sqri
c
```

```
                xnn = 0.25*(xn1+xn2+xn3+xn4)
                ynn = 0.25*(yn1+yn2+yn3+yn4)
                znn = 0.25*(zn1+zn2+zn3+zn4)
                sqri = 1./sqrt(xnn*xnn + ynn*ynn + znn*znn)
                xn(m,n) = xnn*sqri
                yn(m,n) = ynn*sqri
                zn(m,n) = znn*sqri
              endif
c
              rxrx = xxi**2  + yxi**2  + zxi**2
              rxre = xxi*xet + yxi*yet + zxi*zet
              rere = xet**2  + yet**2  + zet**2
              rjac = ( rxrx*rere - rxre**2 )
c
c  Make sure jacobian is positive. For degenerate or collapsed points
c  a more rigorous forward or backward scheme should be used. For now,
c  just ignore these points.
c
              if (rjac.gt.0. .and. rjac .gt. 1.e-10) then
c               if (rjac .gt. 0.) then
                 rd = 1./ rjac
                 xixs(m,n) = ( rere*xxi - rxre*xet) * rd
                 xiys(m,n) = ( rere*yxi - rxre*yet) * rd
                 xizs(m,n) = ( rere*zxi - rxre*zet) * rd
                 etxs(m,n) = (-rxre*xxi + rxrx*xet) * rd
                 etys(m,n) = (-rxre*yxi + rxrx*yet) * rd
                 etzs(m,n) = (-rxre*zxi + rxrx*zet) * rd
c
c surface velocity derivatives
c
                 dvdy(m,n) = vxi*xiys(m,n) + vet*etys(m,n)
                 dvdz(m,n) = vxi*xizs(m,n) + vet*etzs(m,n)
                 dwdy(m,n) = wxi*xiys(m,n) + wet*etys(m,n)
                 dwdz(m,n) = wxi*xizs(m,n) + wet*etzs(m,n)
c
c     modification of 8/16/93
c
                 dudx(m,n) = uxi*xixs(m,n) + uet*etxs(m,n)
                 dudy(m,n) = uxi*xiys(m,n) + uet*etys(m,n)
                 dudz(m,n) = uxi*xizs(m,n) + uet*etzs(m,n)
                 dvdx(m,n) = vxi*xixs(m,n) + vet*etxs(m,n)
                 dwdx(m,n) = wxi*xixs(m,n) + wet*etzs(m,n)
c end of modification
c
c surface derivatives
c
                 dxdy(m,n) = xxi*xiys(m,n) + xet*etys(m,n)
                 dxdz(m,n) = xxi*xizs(m,n) + xet*etzs(m,n)
c                 if(n .eq. ns .or. n .eq. ne) then
c                   write(50,*) m,n,dvdy(m,n),dvdz(m,n)
c                   write(51,*) m,n,dwdy(m,n),dwdz(m,n)
c                   write(52,*) m,n,dxdy(m,n),dxdz(m,n)
c                 endif
              else
                 njac = njac + 1
                 jacm(njac) = m
                 jacn(njac) = n
                 xixs(m,n) = 0.
                 xiys(m,n) = 0.
                 xizs(m,n) = 0.
                 etxs(m,n) = 0.
                 etys(m,n) = 0.
                 etzs(m,n) = 0.
                 dvdy(m,n) = 0.
                 dvdz(m,n) = 0.
                 dwdy(m,n) = 0.
```

```
               dwdz(m,n) = 0.
               dxdy(m,n) = 0.
               dxdz(m,n) = 0.
            endif

   30       continue
c
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)
  40         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,msave,jsav,nsave,asav,bsav,gsav,
     &     first,back,second,ifatal
      logical blank, first, back, second
c
c  Find point on surface subset that is closest to (xp,yp,zp).
c
       dsave = 1.0e8
       do 10 n = ns,ne
       do 10 m = ms,me
          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
 10       continue
c       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,ksmax,
     &                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),flgsn(151)
      common /sfbond/ jsp(151),jsr(151),ksp(151),ksr(151)
      logical first,back,second
       ifail = 0
c
c  find quadrant in which the point (xp,yp,zp) lies.
c
```

```
      jstart = js
      kstart = ks

    5 continue

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

      do 10 iter = 1,iterm

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

```
     1           + etzs(js,kp) * c01 + etzs(jp,kp) * c11
c          endif
             ddelxi = xix * (xp - xt)
     1               + xiy * (yp - yt)
     2               + xiz * (zp - zt)
             ddelet = etx * (xp - xt)
     1               + ety * (yp - yt)
     2               + etz * (zp - zt)
             delxi  = delxi + ddelxi
             delet  = delet + ddelet
c
c  Test for being far off.
c
             if (max(abs(delxi-0.5),abs(delet-0.5)).gt.3.5) goto 12
c
c  Close enough?
c
             error = ddelxi**2 + ddelet**2
c             error = sqrt(error)
             if (error.le.1.e-3 ) goto 12

   11        continue

   12     continue
c
c  Converged to the right cell?
c
          if (max(abs(delxi-0.5),abs(delet-0.5)).le.0.5+toler) goto 20
c
c  Update cell number.
c
          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
c
c  If (js,ks) didn't change, we hit the grid boundary. Limit delxi,delet
c  to the range [0,1].
c
          if (js.eq.jso.and.ks.eq.kso) then
             delxi = min(max(delxi,0.),1.)
             delet = min(max(delet,0.),1.)
             istat = 1
             write(6,*) ' delxi,delet',delxi,delet
             goto 20
          endif

   10     continue
```

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

   20  continue
c       write(6,*) 'js,ks',js,ks
c       write(6,*) 'delxi delet',delxi,delet
       return
       end



      subroutine heat(qw,qwbtu,wltemp,cf)
c
c This program reads in streamline paths and metric coefficients, uses
c Zoby's approximate heating methodology and used Gupta's curve fit to
c compute equilibrium air thermodynamics and transport properties to
c evaluate convective heating along the streamline.
c
      common /connty/ i1,j1,k1
      common /stagpt/ xstag,ystag,zstag
      common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzta(8)
      common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
     &                iblank(151,40,110), blank,isubs(2),
     &                jsubs(2),ksubs(2)
      common /qxyz/ u(151,40,110),v(151,40,110),
     &              w(151,40,110), p(151,40,110)
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
      common /freest/ pinf,tinf,rhoinf,vinf,vninf,cpinf,igas
      common /turbml/ iturb
      common /radeq/ itwall,wallt,irad,eps
      common /tempwl/ twall(1000)
c
c     boundary and momentum thickness
c
      common /bltrn/ deltal,thetl
c
      dimension xs(1000),ys(1000),zs(1000),xn(1000),yn(1000),
     &          zn(1000),vels(1000),ps(1000),hmet(1000),
     &     isaves(1000),ksaves(1000)
c
      logical blank,back,first,second
c
c conversion factors
c
      data cmin,fps,atm,den /2.54, 30.48, 2116.2, .01601846/
      data rtok /1.8/
c
c Stefan-Boltzman constant(w/cm**2/k)
c
      data sigma /5.67e-12/
c
      rewind 15
      is = 1
```

```
 80    continue
       read(15,end=90) xs(is),ys(is),zs(is),xn(is),yn(is),zn(is),
      &                 vels(is),ps(is),hmet(is),isaves(is),ksaves(is)
       hmet(is) = hmet(is)*cmin
       is = is + 1
       go to 80
 90    continue
       it = 1
 92    continue
       if(itwall .eq. 1) then
         read(18,*,end=95) twall(it)
         twall(it) = twall(it) / rtok
         it = it + 1
         go to 92
       endif
 95    continue
       is = is -1
       if(itwall .eq. 0) then
         do 97 iu = 1,is
         twall(iu) = wallt/rtok
 97      continue
       endif
       iwit = 0
 99    continue
       isubs(1) = 1
       isubs(2) = idim
       jsubs(1) = 1
       jsubs(2) = jdim
       ksubs(1) = 1
       ksubs(2) = kdim
       blank = .false.
c
       igsav = 1
       issav = 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
c
c initial edge velocity
c
       vele = vels(iter) * vninf
       velet = vele
c       vele = vels(iter) * 30.48
c
c edge pressure
c
       pe = ps(iter) * pinf
c
c stagnation enthalpy (cal/g)
c
       hinf = cpinf*tinf + 0.5*vinf**2*2.3901e-8
       write(6,*) ' hinf(computed) = ',hinf
c
c initial search cell
c
       isav = isaves(iter)
       jsav = 1
```

```
      ksav = ksaves(iter)
 100  continue
c
c edge enthalpy (cal/g)
c
c      he = hinf - 0.5*vele**2*2.3901e-8
      he = hinf - 0.5*velet**2*2.3901e-8
c
c get edge property
c
      he = 1.0e-3*he
      write(6,*) ' he = ',he
      iflag = 1
      if(igas .eq. 0) then
        iflag = 0
        te = he /.24e-3
      else if(he .lt .12) then
         te = he / .24e-3
      else
         call enthalpy(pe,he,te,iflag)
         if(ifatal .ne. 0) return
      endif
      call eqprop(pe,te,cpe,rhoe,xmue,xke,pre,gammae,airme,iflag)
      if(ifatal .ne. 0) return
c
c get wall properties(tw is input value)
c
      iflag = 2
      if(igas .eq. 0) iflag = 0
      tw = twall(iter)
      call enthalpy(pe,hw,tw,iflag)
      if(ifatal .ne. 0) return
      call eqprop(pe,tw,cpw,rhow,xmuw,xkw,prw,gammaw,airmw,iflag)
      if(ifatal .ne. 0) return
c
c compute Eckert's reference enthalpy
c
c      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
c get reference properties
c
      iflag = 1
      if(igas .eq. 0) then
        tstar = hstar / .24e-3
        iflag = 0
      else
         call enthalpy(pe,hstar,tstar,iflag)
         if(ifatal .ne. 0) return
      endif
      call eqprop(pe,tstar,cpstar,rhostr,xmustr,xkstar,prstar,gamstr,
     &     armstr,iflag)
         if(ifatal .ne. 0) return
c
c compute momentum boundary layer thickness
c
      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
c        sum = 0.5*f2*ds
c        sum = 0.25*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
      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*velet*hmet(iter)**c3
      if(iter .eq. 1) then
c         sum = 0.25*f2*ds
         sum = 0.5*f2*ds
      else
         sum = 0.5 * (f1 + f2) * ds + sum0
      endif
c      thetl = (c2*sum)**c4/(rhoe*vele*hmet(iter)) * .93
c      thetl = (c2*sum)**c4/(rhoe*vele*hmet(iter))
      thetl = (c2*sum)**c4/(rhoe*velet*hmet(iter))
      write(6,*) ' thetlo, thetl = ',thetlo,thetl
c      reth = rhoe * vele * thetl / xmue
      reth = rhoe * velet * thetl / xmue
      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.)*
     &          (rncap+2.)))**rem
           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
      endif
c      f2 = rhostr*xmustr*vele*hmet(iter)**2
c      thetl = .664*sqrt(sum)/(rhoe*vele*hmet(iter))*.93
c      thetl = .664*sqrt(sum)/(rhoe*vele*hmet(iter))
      write(6,*) ' Momentum thickness = ',thetl
c
c compute boundary layer thickness
c
      if(iturb .eq. 0) then
         deltal = 5.55*thetl
```

```
      else
c         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.)*hw/(rncap*haw)+1.)*
c     &         (1.+1.29*prw**.333*.5*vele**2*2.3901e-11/he))
     &         (1.+1.29*prw**.333*.5*velet**2*2.3901e-11/he))
          deltal = thetl * thedd
      endif
c      deltal = 5.55*thetl*1.18
      write(6,*) ' Boundary layer thickness = ',deltal
c      deltal = 0.
      if(icon .eq.1) go to 150
c
      xble = xs(iter) + xn(iter)*deltal/cmin
      yble = ys(iter) + yn(iter)*deltal/cmin
      zble = zs(iter) + zn(iter)*deltal/cmin
c
c convert back to inch for interpolation
c
c      xble = xble/cmin
c      yble = yble/cmin
c      zble = zble/cmin
c
c find cell for interpolation
c
      isrch = 0
      igrid = 1
      nsubs = 1
      call close3(isrch,igrid,nsubs,i,j,k,a,b,g,xble,yble,zble,isub,
     & istat)
      if(istat .ne. 0) then
        write(6,*) ' close3 search  failed'
c        stop
        i = isaves(iter)
        j = jj
c        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
c
c interpolation of edge properties
c
      i1 = i
      j1 = j
      k1 = k
      call find(xble,yble,zble)
      if(ifatal .ne. 0) return
c      velb = vele
      velb = velet
      ip1 = i1 + 1
      jp1 = j1 + 1
      kp1 = k1 + 1
      ue = u(i1,j1,k1)*psi(1)+u(ip1,j1,k1)*psi(2)+u(ip1,j1,kp1)*psi(3)
     &   + u(i1,j1,kp1)*psi(4)+u(i1,jp1,k1)*psi(5)+u(ip1,jp1,k1)*psi(6)
     &   + u(ip1,jp1,kp1)*psi(7)+u(i1,jp1,kp1)*psi(8)
c
      ve = v(i1,j1,k1)*psi(1)+v(ip1,j1,k1)*psi(2)+v(ip1,j1,kp1)*psi(3)
     &   + v(i1,j1,kp1)*psi(4)+v(i1,jp1,k1)*psi(5)+v(ip1,jp1,k1)*psi(6)
     &   + v(ip1,jp1,kp1)*psi(7)+v(i1,jp1,kp1)*psi(8)
c
      we = w(i1,j1,k1)*psi(1)+w(ip1,j1,k1)*psi(2)+w(ip1,j1,kp1)*psi(3)
```

```
     &    + w(i1,j1,kp1)*psi(4)+w(i1,jp1,k1)*psi(5)+w(ip1,jp1,k1)*psi(6)
     &    + w(ip1,jp1,kp1)*psi(7)+w(i1,jp1,kp1)*psi(8)
c
      pe = p(i1,j1,k1)*psi(1)+p(ip1,j1,k1)*psi(2)+p(ip1,j1,kp1)*psi(3)
     &    + p(i1,j1,kp1)*psi(4)+p(i1,jp1,k1)*psi(5)+p(ip1,jp1,k1)*psi(6)
     &    + p(ip1,jp1,kp1)*psi(7)+p(i1,jp1,kp1)*psi(8)
c
      pe = pe*pinf
c
      vele = sqrt(ue*ue + ve*ve + we*we)
c
c     use tangential component
c
      velen = ue*xn(iter) + ve*yn(iter) + we*zn(iter)
      velet = sqrt(vele*vele - velen*velen)
      velet = velet*vninf
      vele = vele*vninf
c      vele = vele*30.48
c
      write(6,*) ' vele,velb = ',vele,velb
c     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
c          write(6,*) ' edge failed'
          icon = 1
        endif
      endif
      go to 100
c
c compute laminar heating
c
 150  continue
c
c momentum thickness Reynolds number
c
c      reth = rhoe*vele*thetl/xmue
      write(6,*) ' reth = ',reth
c
c     compute internal energy per unit mass (m**2/s**2)
c
      ein = he*4.184e10 - pe*1.01336e6/rhoe
      eins = ein * 1.e-4
      rhoes = rhoe * 1.e3
c
c     use tannelhill's curve fit to evaluate entropy
c
      call tgas2(eins,rhoes,entrpy)
c
c adiabatic wall enthalpy (kcal/g)
c
c      haw = he +0.5*sqrt(prw)*vele**2*2.3901e-11
      haw = he +0.5*sqrt(prw)*velet**2*2.3901e-11
      if(iturb .eq. 0) then
         cf = .44 / reth*(rhostr/rhoe)*(xmustr/xmue)
         if(xs(iter) .le. 128.8) then
            cf = 1.534*(rhoe*xmue)**.43*(xmuw*rhow)**.07
         endif
c         cf = .44 / reth
c         qw = 0.22*rhostr*xmustr*rhoe*vele/(rhoe*xmue)*
         qw = 0.22*rhostr*xmustr*rhoe*velet/(rhoe*xmue)*
     &         (haw - hw)/(reth*prw**.6)
      else
```

```
         cf = 2.*c1/(reth**rem)*(rhostr/rhoe)*(xmustr/xmue)**rem
c         cf = 2.*c1/(reth**rem)
c         qw = c1*(rhostr/rhoe)*(xmustr/xmue)**rem*rhoe*vele*
         qw = c1*(rhostr/rhoe)*(xmustr/xmue)**rem*rhoe*velet*
     &         (haw-hw)/(prw**.4*reth**rem)
      endif
c
c convert kcal/cm**2/s to watt/cm**2
c
      qw = qw*4.18392e3
c
c compute radiation equilibrium wall temperature
c
      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
c
c convert to btu/ft**2/s
c
      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)
c      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
 200  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
c
 9001 format(5(1x,e14.7))
 9002 format(4(1x,e14.7))
 9999 continue
      return
      end
c
      SUBROUTINE TGAS2(E,R,S)
C
C     INPUTS FOR SUBROUTINE:
C     E=INTERNAL ENERGY IN (M/SEC)**2
C     R=DENSITY IN KG/M**3
C
```

```
C     OUTPUT:
C     S=ENTROPY IN (M/SEC)**2/K
C
C
      DATA E0,R0,GASCON/78408.4E00,1.292E00,287.06E00/
      RRATIO=R/R0
      ERATIO=E/E0
      Y=ALOG10(RRATIO)
      Z=ALOG10(ERATIO)
      IF(ABS(Y+4.5E00).LT.2.5E-02) GO TO 10
      IF(ABS(Y+0.5E00).LT.0.5E-02) GO TO 40
      IFLAG=-1
      GO TO 80
10    IFLAG=0
      RSAVE=R
      YM=Y
      Y=-4.5E00+2.5E-02
      YHIGH=Y
      R=(10.**Y)*R0
      JFLAG=-1
      GO TO 80
20    SHIGH=S
      Y=-4.5E00-2.5E-02
      YLOW=Y
      R=(10.**Y)*R0
      JFLAG=0
      GO TO 80
30    SLOW=S
      GO TO 70
40    IFLAG=1
      RSAVE=R
      YM=Y
      Y=-0.5E00+0.5E-02
      YHIGH=Y
      R=(10.**Y)*R0
      JFLAG=-1
      GO TO 80
50    SHIGH=S
      Y=-0.5E00-0.5E-02
      YLOW=Y
      R=(10.**Y)*R0
      JFLAG=0
      GO TO 80
60    SLOW=S
70    S=SLOW+(SHIGH-SLOW)/(YHIGH-YLOW)*(YM-YLOW)
      R=RSAVE
      RETURN
80    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   DELTZ=Z-0.4E00
      DELTS=(2.5E00*DELTZ-Y)*GASCON*2.302585E00
      S=6779.2004E00+DELTS
      GO TO 130
120   SNON=GAS1+GAS2+GAS3+GAS4
      S=GASCON*SNON
130   IF(IFLAG) 160,140,150
140   IF(JFLAG) 20,30,160
150   IF(JFLAG) 50,60,160
160   CONTINUE
1000  FORMAT(/20X,48HWARNING]  OUTSIDE VALIDITY RANGE OF CURVE FIT
     *,/,20X,5HRHO =,1PE15.8,5X,3HE =,1PE15.8,/)
      RETURN
      END
c
      subroutine enthalpy(pinp,hinp,t,iflag)
c
c This subroutine takes input pressure and enthalpy, uses Gupta's
c equilibrium air thermodynamic and transport properties curve fit
c 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), eh(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, 4, 3/
      data ipmax /7/
      data tmax /2250., 4250., 6750., 10750., 17750., 25000.,
     &           2250., 4250., 6750., 11750., 18750., 28000.,
     &           2750., 5250., 9750., 17750., 30000., 30001.,
     &           3250., 6250., 15250., 30000., 30001., 30001.,
     &           3750., 8250., 17750., 30000., 30001., 30001.,
     &           4250., 9250., 18750., 30000., 30001., 30001.,
     &           6250., 12750., 30000., 30001., 30001., 30001./
c
      data ah /.128180e1, .125380e2, .426138e2, .885088e1, .151569e2,
     &         .101759e2,
     &         .902850, .237222e2, .880011e2, -.333238e2, .196866e2,
     &         .446849e2,
     &         .653358, .431122e1, -.126229e1, .209845e2, .268647e2, 0.,
     &         .363885, -.865884e1, -.164319e2, -.207249e2, 0., 0.,
     &         .209284, -.171560e2, -.134978e2, -.564265e1, 0., 0.,
     &         .124937, -.120314e2, -.913636e1, .639208e1, 0., 0.,
     &        -.755123e-2, -.117469e1, -.245329e1, 0., 0., 0./
c
      data bh /.121182e2, .720107e2, .123001e3, -.207380e2, -.713138e1,
     &         -.161956e2,
     &         .839944e1, .118014e3, .213329e3, -.316397e2, -.201771e2,
     &         -.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./
```

```
c
      data ch /.424907e2, .148949e3, .121801e3, -.134604e2, -.172524,
     &         -.336892e1,
     &         .289458e2, .214780e3, .181623e3, -.401000e1, .635249e1,
     &         .159412e3,
     &         .201689e2, .541203e2, .109117e2, -.399635, .145439e3,
     &         0.,
     &         .110641e2, -.132700e2, .447878e1, -.678713e2, 0., 0.,
     &         .622153e1, -.332532e2, .192371e1, -.396119e2, 0., 0.,
     &         .355163e1, -.129249e2, -.259796e1, .882252e1, 0., 0.,
     &         .366590, -.214181e1, -.288683, 0., 0., 0./
c
      data dh /.665524e2, .133853e3, .509305e2, .166408e1, .643645,
     &         .161274e2,
     &         .448640e2, .171168e3, .661367e2, .379639e1, -.174347,
     &         -.738595e2,
     &         .309518e2, .462077e2, .400303e1, .387388e1, -.846045e2,
     &         0.,
     &         .173605e2, .242899e1, .196275e1, .312942e2, 0., 0.,
     &         .101561e2, -.747816e1, .930272, .251297e2, 0., 0.,
     &         .617946e1, .262066, .114665e1, .258596e1, 0., 0.,
     &         .210603e1, .251111e1, .421200, 0., 0., 0./
c
      data eh /.385195e2, .451550e2, .995964e1, .356570e1, .356353e1,
     &         -.201068e1,
     &         .256452e2, .513939e2, .110476e2, .325469e1, .354258e1,
     &         .155141e2,
     &         .174843e2, .152182e2, .284253e1, .283981e1, .212051e2,
     &         0.,
     &         .999025e1, .417259e1, .256061e1, -.158288e1, 0., 0.,
     &         .603650e1, .178858e1, .244209e1, -.207198e1, 0., 0.,
     &         .386028e1, .235363e1, .236890e1, .107086e1, 0., 0.,
     &         .195195e1, .212013e1, .239842e1, 0., 0., 0./
c
      pinps = 0.
      if(iflag .eq. 1) write(6,*) 'pinp, hinp = ',pinp,hinp
      if(iflag .eq. 0) then
        hinp = .24e-3*t
        return
      endif
c
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
c find the index which brace the input pressure
c
      do 10 i = 1,7
      if(pinp .le. ptb(i)) then
        if(abs(pinp-ptb(i)) .le. 1.e-4) then
```

```
          intp = 0
        else
          intp = 1
        endif
        ip = i
        ipm1 = ip - 1
        go to 20
      endif
 10   continue
      write(6,*) ' Pressure is out of range'
 20   continue
c      write(6,*) ' ip, ipm1 ',ip,ipm1
      if(iflag .eq. 2) then
        xi = alog(t*.0001)
        if(intp .eq. 0) go to 50
        do 30 i = 1,ntmax(ipm1)
        if(t .gt. tmax(i,ipm1)) go to 30
        it = i
        go to 40
 30   continue
        write(6,*) ' Temperature ',t,' is outside the range of',
     &' available data'
        ifatal = 1
        return
 40     continue
        if(t .lt. 500.) then
           hinp = .24e-3*t
          if(pinps .ne. 0.) pinp = pinps
          return
        endif
        h1 = ah(it,ipm1)*xi**4 + bh(it,ipm1)*xi**3 + ch(it,ipm1)*
     &  xi**2 + dh(it,ipm1)*xi + eh(it,ipm1)
c
 50     continue
        if(t .lt. 500.) then
           hinp = .24e-3*t
          if(pinps .ne. 0.) pinp = pinps
          return
        endif
        do 60 i = 1,ntmax(ip)
        if(t .gt. tmax(i,ip)) go to 60
        it = i
        go to 70
 60   continue
        write(6,*) ' Temperature ',t,' is outside the range of',
     &' available data'
        ifatal = 1
        return
 70     continue
        h2 = ah(it,ip)*xi**4 + bh(it,ip)*xi**3 + ch(it,ip)*xi**2
     &      + dh(it,ip)*xi + eh(it,ip)
c
        if(intp .eq. 0) then
          alogh = h2
        else
          alogh = (h2 - h1) * (alog(pinp) - alog(ptb(ipm1)))
     &        / (alog(ptb(ip)) - alog(ptb(ipm1))) + h1
        endif
c
        hinp = exp(alogh)
        go to 200
      endif
c
c compute maximum enthalpy for each temperature range for each pressure
c
      if(ipass .eq. 0) then
```

```
        ipass = 1
        do 80 j = 1,ipmax
        do 80 i = 1,ntmax(j)
        xi = alog(tmax(i,j)*.0001)
        hmax(i,j) = exp(ah(i,j)*xi**4 + bh(i,j)*xi**3 +
     &                  ch(i,j)*xi**2 + dh(i,j)*xi + eh(i,j))
c        write(6,*) ' hmax ',j,i,hmax(i,j)
 80     continue
       endif
       if(intp .eq. 0) go to 100
c
c find correct temperature range and curve to solve for temperature
c
       ntip = ntmax(ipm1)
       do 90 i = 1,ntip
       if(hinp .lt. hmax(i,ipm1)) then
         coef1 = ah(i,ipm1)
         coef2 = bh(i,ipm1)
         coef3 = ch(i,ipm1)
         coef4 = dh(i,ipm1)
         coef5 = eh(i,ipm1)
         tmin = 500.
         hmin = .24e-3*tmin
         if(i .gt. 1) then
           tmin = tmax(i-1,ipm1)
           hmin = hmax(i-1,ipm1)
         endif
c        ti = (tmax(i,ipm1)+tmin)*.5
         ti=tmin+(alog(hinp)-alog(hmin))*(tmax(i,ipm1)-tmin)
     &          /(alog(hmax(i,ipm1))-alog(hmin))
         call newton(ti,hinp,t1)
         go to 100
       else if(hinp .eq. hmax(i,ipm1)) then
         t1 = tmax(i,ipm1)
         go to 100
       endif
 90    continue
c
       write(6,*) ' Enthalpy is out of range', hinp
       ifatal = 1
       return
 100   continue
c
       ntip = ntmax(ip)
       do 110 i = 1,ntip
       if(hinp .lt. hmax(i,ip)) then
         coef1 = ah(i,ip)
         coef2 = bh(i,ip)
         coef3 = ch(i,ip)
         coef4 = dh(i,ip)
         coef5 = eh(i,ip)
         tmin = 500.
         hmin = .24e-3*tmin
         if(i .gt. 1) then
           tmin = tmax(i-1,ip)
           hmin = hmax(i-1,ip)
         endif
         ti = tmin + (alog(hinp)-alog(hmin))*(tmax(i,ip)-tmin)
     &             / (alog(hmax(i,ip))-alog(hmin))
         call newton(ti,hinp,t2)
         if(ifatal .ne. 0) return
         go to 120
       else if(hinp .eq. hmax(i,ip)) then
         t2 = tmax(i,ip)
         go to 120
       endif
```

```
 110  continue
c
      write(6,*) ' Enthalpy is out of range', hinp
      ifatal = 1
      return
 120  continue
      if(intp .eq. 0) then
        t = t2
        go to 200
      endif
c
c use logrithm interpolation to find temperature
c
      alogt = (alog(t2) - alog(t1)) * (alog(pinp) - alog(ptb(ipm1)))
     &        / (alog(ptb(ip)) - alog(ptb(ipm1))) + alog(t1)
c
      t = exp(alogt)
c
 200  continue
c
      if(pinps .ne. 0.) pinp = pinps
      return
      end
c
      subroutine newton(ti,hinp,t)
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
c
c use newton's method to solve for polynomial
c
      common /coef/ a, b, c, d, e
c
      xi = alog(ti*.0001)
      iter = 0
 10   continue
      res1 = exp(a*xi**4 + b*xi**3 + c*xi**2 + d*xi + e)
      res = res1 - hinp
      if(abs(res) .lt. 1.0e-4) go to 100
c
c compute derivative
c
      fp = res1 * (4.*a*xi**3 + 3.*b*xi**2 + 2.*c*xi + d)
      dxi = -res/fp
      xi = xi + dxi
      iter = iter + 1
      if(iter .gt. 50) then
        write(6,*) ' Newton method failed to converge'
        ifatal = 1
        return
      endif
      go to 10
 100  continue
      t = 10000. * exp(xi)
      return
      end
      subroutine eqprop(pinp,tinp,cp,rho,xmu,xk,pr,gamma,airm,iflag)
c
c This routine uses Gupta's curve fit equation to obtain thermodynamics
c and transport properties using logarithm interpolation.
c Inputs are pressure (pinp) and temperature (tinp)
c
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
c
      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),
```

```
     &           amu(4,7), bmu(4,7), cmu(4,7), dmu(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(5,7), tmumax(4,7),
     &           tkmax(7,7), tprmax(8,7), ncpmax(7), nzmax(7),
     &           nmumax(7), nkmax(7), nprmax(7)
c
c     runiv is in cal/g-mole-k
c
      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, 5, 4, 3/
      data nmumax /4, 4, 4, 4, 3, 3, 2/
      data nkmax /7, 7, 7, 6, 6, 5, 4/
      data nprmax /8, 8, 8, 7, 7, 5, 5/
      data ipmax /7/
c
      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.,5750.,9250.,13750.,22500.,30000.,30001.,30001.,
     & 1750.,3750.,6750.,10750.,17750.,30000.,30001.,30001.,30001./
c
      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./
c
      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./
c
      data tkmax /
     & 1750., 2750., 4750., 6250., 10250., 17750., 25000.,
     & 1750., 2750., 4750., 6250., 11250., 18250., 28000.,
     & 2250., 3250., 5750., 7750., 12750., 18750., 30000.,
     & 2250., 4250., 6750., 9250., 16750., 30000., 30001.,
     & 2250., 4250., 7750., 10750., 19250., 30000., 30001.,
     & 3250., 5250., 8750., 13750., 30000., 30001., 30001.,
     & 3750., 6250., 10750., 30000., 30001., 30001.,30001./
c
      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./
c
      data acp /
     & .349023, .152264e2, -.159675e2, -.108293e3, -.116264e4,
     & -.238707e2, -.209557e2, .762671e3, -.789820e3,
```

```
     & .199532, .345376e1, -.369572e2, -.146237e3, -.758521e3,
     & -.330240e2, -.618098e2, .125063e3, 0.,
     & .669436, -.453138e2, -.151035e3, .539167e3, .217707e2,
     & -.122810e3, .162348e3, 0., 0.,
     & .291577, -.662937e1, .128388e3, -.296048e2, -.308894e3,
     & .104767e3, -.188079e3, .232697e3, 0.,
     & .164992, -.830572e1, .848335e2, -.945467e1, -.153176e3,
     & .975058e2, -.473648e2, 0., 0.,
     & .111751, .252675, .450386e2, .231376e2, -.799940e2,
     & .491689e2, -.253231e3, 0., 0.,
     & .986591e-1, .974261e-1, .210207e2, .143729e2, -.347606e2,
     & .450529e2, 0., 0., 0./
c
      data bcp /
     & .344158e1, .129277e3, -.136508e3, -.515276e3, -.266973e4,
     & -.104336e3, .253228e2, -.167407e4, .263864e4,
     & .192597e1, .315624e2, -.128366e3, -.581296e3, -.139794e4,
     & -.866157e2, .103127e3, -.298121e3, 0.,
     & .644478e1, -.292666e3, -.591051e3, .126894e4, -.450370e2,
     & .240030e3, -.497482e3, 0., 0.,
     & .278787e1, -.382984e2, .596922e3, -.133243e3, -.267701e3,
     & -.105447e3, .472158e3, -.869061e3, 0.,
     & .156336e1, -.483112e2, .361629e3, -.640807e2, -.476111e2,
     & -.158721e3, .818135e2, 0., 0.,
     & .105018e1, .341131e1, .167261e3, -.104484e1, .170114e2,
     & -.116351e3, .955890e3, 0., 0.,
     & .923581, .146776e1, .677318e2, -.128820e2, .320177e2,
     & -.143364e3, 0., 0., 0./
c
      data ccp /
     & .126715e2, .411057e3, -.411657e3, -.882748e3, -.221802e4,
     & -.890658e2, .212355e2, .130713e4, -.326378e4,
     & .694347e1, .107177e3, -.129698e3, -.848597e3, -.900003e3,
     & -.489572e2, -.262275e2, .210795e3, 0.,
     & .230631e2, -.699603e3, -.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, .131529e2, .224425e3, -.271807e2, .187072e2,
     & .889977e2, -.132457e4, 0., 0.,
     & .323392e1, .575473e1, .778089e2, -.173603e2, .148249e1,
     & .161302e3, 0., 0., 0./
c
      data dcp /
     & .208154e2, .580300e3, -.525250e3, -.642505e3, -.791376e3,
     & -.182697e2, -.128857e2, -.422349e3, .176381e4,
     & .112521e2, .160585e3, -.169299e2, -.532403e3, -.238528e3,
     & -.182071e1, -.850086e1, -.295269e2, 0.,
     & .365225e2, -.730849e3, -.502696e3, .370582e3, .517928e1,
     & .225676e2, -.216688e3, 0., 0.,
     & .157475e2, -.627915e2, .757047e3, -.100614e3, .326629e1,
     & .595868e1, .141182e3, -.682883e3, 0.,
     & .879873e1, -.897230e2, .376565e3, -.403342e2, .314736e1,
     & -.936668e1, -.836769e2, 0., 0.,
     & .591074e1, .203259e2, .128924e3, -.102436e2, -.350311e1,
     & -.242638e2, .798459e3, 0., 0.,
     & .519284e1, .896935e1, .381171e2, -.137585e1, -.510951e1,
     & -.752038e2, 0., 0.,  0./
c
      data ecp /
     & .116592e2, .305728e3, -.241298e3, -.166628e3, -.102433e3,
     & .138792e1, .135712e1, .482128e2, -.348874e3,
     & .570825e1, .884544e2, .207647e2, -.119389e3, -.216169e2,
     & .229104e1, .253250e1, -.792067e1, 0.,
```

```
     & .203928e2, -.281133e3, -.107793e3, .457650e2, .180195e1,
     & .100733e1, .277132e2, 0., 0.,
     & .820277e1, -.154364e2, .205793e3, -.168003e2, .838365,
     & .821623, -.156018e2, .143351e3, 0.,
     & .412806e1, -.280651e2, .915792e2, -.458728e1, .922570e-1,
     & .987515, .339060e2, 0., 0.,
     & .244269e1, .100197e2, .263694e2, -.333185e-1, .184168,
     & .263659e1, -.175990e3, 0., 0.,
     & .202191e1, .384233e1, .628850e1, .743313, .877002,
     & .129598e2, 0., 0., 0./
c
      data az /
     & .710750, -.614415e1, -.632086e2, -.467833e2, .556705e2,
     & .824286, .746758e1, -.385889e2, -.455262e2, .809623e2,
     & .873086, -.195828e1, -.417508e2, -.431463e2, .208036e3,
     & .904213, .124751e1, -.325326e2, -.428667e2, .217096e3,
     & .102671e1, .387376e2, -.161621e2, -.255245e2, -.784807e2,
     & .970875, -.10200e1, -.993188e1, .398457e-1, 0.,
     & .103304e1, -.555015e1, .202955e2, 0., 0./
c
      data bz /
     & .107229e1, .861656e1, .370722e2, .139011e2, -.135009e2,
     & .625098, -.460729e1, .209649e2, .121138e2, -.162146e2,
     & .434929, .324383e1, .199010e2, .101757e2, -.342626e2,
     & .311295, .485004, .137742e2, .888031e1, -.309522e2,
     & -.465274e-1, -.204439e2, .637080e1, .419968e1, .129796e2,
     & .869030e-1, .186655e1, .353080e1, -.612253, 0.,
     & -.585872e-1, .157079e1, -.323532e1, 0., 0./
c
      data cz /
     & -.125673e1, -.370256e1, -.776456e1, -.138693e1, .118386e1,
     & -.689867, .109594e1, -.398276e1, -.108251e1, .120105e1,
     & -.454400, -.123210e1, -.334091e1, -.804882, .210825e1,
     & -.302086, -.321087, -.203163e1, -.620696, .165245e1,
     & .972123e-2, .404607e1, -.827695, -.208573, -.758996,
     & -.737745e-1, -.540958, -.389667, .997321e-1, 0.,
     & .237877e-1, -.115055, .203092, 0., 0./
c
      data dz /
     & .564944, .681208, .706484, .592861e-1, -.427210e-1,
     & .286982, -.898428e-1, .327436, .415356e-1, -.375039e-1,
     & .176448, .198816, .243749, .274096e-1, -.563525e-1,
     & .107468, .632573e-1, .130377, .188157e-1, -.384201e-1,
     & .417402e-2, -.344141, .466769e-1, .395832e-2, .194343e-1,
     & .218303e-1, .639254e-1, .187431e-1, -.411847e-2, 0.,
     & -.281715e-2, .324023e-2, -.525620e-2, 0., 0./
c
      data ez /
     & -.822333e-1, -.443045e-1, -.233636e-1, -.903887e-3, .551468e-3,
     & -.376727e-1, .162238e-2, -.970559e-2, -.569596e-3, .424122e-3,
     & -.212727e-1, -.110471e-1, -.644569e-2, -.334336e-3, .555405e-3,
     & -.116924e-1, -.364522e-2, -.302863e-2, -.206237e-3, .330019e-3,
     & -.536830e-3, .107287e-1, -.941988e-3, -.175392e-4, -.182292e-3,
     & -.179762e-2, -.255478e-2, -.320898e-3, .542207e-4, 0.,
     & .168221e-3, -.18832e-4, .489857e-4, 0., 0./
c
      data amu /
     & -.1160076e-4, -.9105422, .1463029e-1, -.2140374e-2,
     & .2397194e-4, -.5784272, .1658118e-1, .6903134e-2,
     & .5085043e-4, -.341487, .2450600e-1, -.3561146e-1,
     & .6394112e-4, -.2376368, .6309492e-3, -.1622687e1,
     & .5781887e-4, -.1844238, .2606784e-1, 0.,
     & .7256455e-4, -.9524274e-1, .5037513e-1, 0.,
     & .7609039e-4, -.7868582e-1, 0., 0./
c
      data bmu /
```

```
     & .6656010e-3, .4949794, -.5019958e-2, .6529285e-3,
     & .5564725e-3, .2816531, -.5027652e-2, -.1345295e-2,
     & .4774840e-3, .1473594, -.6697224e-2, .7255623e-2,
     & .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./
c
      data cmu /
     & -.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./
c
      data dmu /
     & .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, .9370344e-3, .9381977e-5, .8445985e-3,
     & .1688754e-4, .4609058e-3, -.1018512e-4, 0.,
     & .1114437e-4, .1227975e-3, -.1682098e-4, 0.,
     & .8791566e-5, .6257766e-4, 0., 0./
c
      data emu /
     & -.6456605e-5, -.5896774e-3, .1623374e-5, -.9908122e-7,
     & -.2690853e-5, -.2352317e-3, .1135683e-5, .8514686e-7,
     & -.8014978e-6, -.8216415e-4, .1134307e-5, -.4590857e-6,
     & -.5014106e-6, -.3279124e-4, -.2960969e-6, -.1570909e-4,
     & -.8622324e-6, -.1377229e-4, .1576999e-6, 0.,
     & -.5020411e-6, -.2772500e-5, .2731352e-6, 0.,
     & -.4216496e-6, -.1234998e-5, 0., 0./
c
      data fmu /
     & .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, .4529650e-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./
c
      data ak /
     & .395299e1, .119879e2, -.832682e2, -.103603e4, .261125e2,
     & .246095e2, -.571805e2,
     & .199665e1, -.831120e2, -.110139e3, .299875e3, .434485e2,
     & .895136e1, -.422029e2,
     & .198558e1, .595832e2, -.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./
c
      data bk /
     & .386816e2, .412181e2, -.419438e3, -.242470e4, .411940e1,
     & -.507490e2, .168628e3,
     & .194822e2, -.560438e3, -.481050e3, .923042e3, .464790e1,
     & -.322183e2, .144838e3,
     & .189164e2, .288748e3, -.206207e3, -.873106e3, -.115449e2,
     & .801073e1, -.960227e1,
```

```
     & .100924e2, .490653e3, -.324274e2, -.306306e3, -.330961e2,
     & -.541310e2, 0.,
     & .328202e1, .387106e2, -.147438e2, -.828711e2, -.381078e2,
     & -.358117e2, 0.,
     & .383393e1, .308927e3, -.133907e2, .143656e2, .146502e2,
     & 0., 0.,
     & .192122e1, .123284e3, .336369, -.216552e2, 0., 0., 0./
c
      data ck /
     & .140687e3, .717156e1, -.751764e3, -.206135e4, -.186054e2,
     & .369131e2, -.181577e3,
     & .706404e2, -.140314e4, -.757873e3, .992814e3, -.155778e2,
     & .350726e2, -.182586e3,
     & .668384e2, .500756e3, -.324643e3, -.445088e3, -.113653e2,
     & .146420e2, .196818e1,
     & .356709e2, .868620e3, -.942568e2, -.782124e2, -.146607e1,
     & .693640e2, 0.,
     & .119939e2, .387282e2, -.530293e2, .998789e1, .117041e2,
     & .553921e2, 0.,
     & .131885e2, .423174e3, -.337860e2, .135247e2, -.187337e2,
     & 0., 0.,
     & .658813e1, .144224e3, -.142705e2, .204578e2, 0., 0., 0./
c
      data dk /
     & .226110e3, -.911924e2, -.566912e3, -.757541e3, -.645054e1,
     & -.897288e1, .859388e2,
     & .113538e3, -.154128e4, -.505860e3, .442621e3, -.220224e1,
     & -.129798e2, .101698e3,
     & .104546e3, .363789e3, -.204871e3, -.927269e2, .135799e1,
     & -.117248e2, .643952e1,
     & .561818e2, .666792e3, -.647282e2, -.466313e1, .306898e1,
     & -.366810e2, 0.,
     & .200944e2, .548304e1, -.299886e2, .227739e1, .122011e1,
     & -.353787e2, 0.,
     & .207305e2, .250668e3, -.122339e2, -.233991e1, .107119e2,
     & 0., 0.,
     & .107630e2, .728083e2, -.134534e1, -.597164e1, 0., 0., 0./
c
      data ek /
     & .127138e3, -.810415e2, -.157470e3, -.108281e3, -.621476e1,
     & -.623025e1, -.213335e2,
     & .599079e2, -.632398e3, -.125800e3, .634709e2, -.558790e1,
     & -.498154e1, -.270417e2,
     & .527822e2, .844428e2, -.494556e2, -.128529e2, -.542822e1,
     & -.389761e1, -.896353e1,
     & .249670e2, .180596e3, -.180857e2, -.585083e1, -.562490e1,
     & .115271e1, 0.,
     & .462882e1, -.120106e2, -.961485e1, -.581069e1, -.578171e1,
     & .274595e1, 0.,
     & .427728e1, .475889e2, -.610064e1, -.556444e1, -.717162e1,
     & 0., 0.,
     & -.127699e1, .684807e1, -.498832e1, -.485454e1, 0., 0., 0./
c
      data apr /
     & .7695318, .4081926e2, -.5060907e3, .3172744e2, -.7624622e4,
     & .1034825e3, -.3414530e3, .1398841e2,
     & .7483071, .4355608e2, .6877578e3, -.7024989e3, -.3107328e3,
     & .9153227e2, .2304245e4, -.1424413e2,
     & .7406012, .3206825e2, .4409105e3, .2474049e2, -.9639789e3,
     & .9436717e2, -.7123564e2, -.7846718e2,
     & .7548377, .6190687e2, -.2641868e3, .7451996e2, .1034771e3,
     & -.1252168e3, -.5022975e4, 0.,
     & .8037721, .9173178e2, .1874806e2, -.1974460e3, .5713095e2,
     & .1619271e3, -.1396134e3, 0.,
     & .8167207, -.1857130e2, -.4287169e2, .6447596e2, .7384318e2,
     & 0., 0., 0.,
```

```
     & .814777, -.6650e1, -.2736096e2, .4889586e2, -.5525036e2,
     & 0., 0., 0./
c
      data bpr /
     & -.2487156, -.6945482e2, .5765441e3, -.1923593e2, .3908646e4,
     & -.3861297e2, .1193445e3, -.2209182e1,
     & -.1659050, -.6751147e2, -.5984367e3, .4515788e3, .1863582e3,
     & -.3041980e2, -.6078459e3, .3715427e1,
     & -.1629686, -.4531855e2, -.3447078e3, .2323747e1, .3960004e3,
     & -.2761116e2, .2178196e2, .1520447e2,
     & -.2611566, -.8052889e2, .2214640e3, -.2929220e2, -.2138868e2,
     & .2959573e2, .9203587e3, 0.,
     & -.5531739, -.1294114e3, -.1571672e2, .9815976e2, -.2693925e2,
     & -.3657252e2, .2786187e2, 0.,
     & -.618678, .2610712e2, .3082544e2, -.2113068e2, -.1345184e2,
     & 0., 0., 0.,
     & -.6129922, .9087650e1, .1729215e2, -.1351549e2, .1123889e2,
     & 0., 0., 0./
c
      data cpr /
     & -.3371200e-2, .4642510e2, -.2594581e3, .6031312e1, -.7963915e3,
     & .5742599e1, -.1639670e2, .1254939,
     & -.3548542e-1, .4101816e2, .2059622e3, -.1153420e3, -.3984823e2,
     & .4035104e1, .6397478e2, -.3586053,
     & .8014385e-1, .2519178e2, .1064873e3, -.4089151e1, -.6388451e2,
     & .3229296e1, -.2554952e1, -.1165686e1,
     & .3168480, .4117452e2, -.7330522e2, .4227110e1, .1519602e1,
     & -.2775372e1, -.6737887e2, 0.,
     & .9090234, .7264736e2, .5656501e1, -.1915726e2, .4625149e1,
     & .3311637e1, -.2213081e1, 0.,
     & .1015263e1, -.1366805e2, -.8483934e1, .2725852e1, .9862360,
     & 0., 0., 0.,
     & .1017670e1, -.4283159e1, -.4132288e1, .1493734e1, -.8807574,
     & 0., 0., 0./
c
      data dpr /
     & .5237761, -.1500926e2, .5768434e2, -.1191714e1, .8069517e2,
     & -.4251452, .1109347e1, -.2782377e-2,
     & .3843113, -.1204527e2, -.3502408e2, .1460894e2, .4000552e1,
     & -.2667702, -.3358410e1, .1651461e-1,
     & .1480977, -.6755087e1, -.1623127e2, .8375614, .5082427e1,
     & -.1885223, .1451363, .4430242e-1,
     & -.1019887, -.1025306e2, .1198777e2, -.2590555, -.3418909e-1,
     & .1290359, .2463703e1, 0.,
     & -.6243393, -.2015462e2, -.1040290e1, .1841274e1, -.3686820,
     & -.1501584, .8742029e-1, 0.,
     & -.6989158, .3448930e1, .1132069e1, -.1705427, -.3633532e-1,
     & 0., 0., 0.,
     & -.7111724, .9588901, .4767165, -.8122753e-1, .3371865e-1,
     & 0., 0., 0./
c
      data epr /
     & -.4272376, .2350767e1, -.6333573e1, .1301576, -.4069244e1,
     & .1565525e-1, -.3701973e-1, .6736752e-5,
     & -.2847560, .1712608e1, .2943975e1, -.9148849, -.1930037,
     & .8781471e-2, .8795057e-1, -.3685522e-3,
     & -.1294933, .8752933, .1221500e1, -.6543669e-1, -.2000122,
     & .5488061e-2, -.4016437e-2, -.8360113e-3,
     & -.1326718e-1, .1246034e1, -.9677752, .5423023e-2, -.5284942e-3,
     & -.2973274e-2, -.4499464e-1, 0.,
     & .1903222, .2763213e1, .9598832e-1, -.8709938e-1, .1391085e-1,
     & .3407133e-2, -.1716422e-2, 0.,
     & .2151091, -.4205156, -.7326105e-1, .5177594e-2, .6720911e-3,
     & 0., 0., 0.,
     & .2215290, -.1026406, -.2655841e-1, .2173037e-2, -.6355462e-3,
     & 0., 0., 0./
```

```
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-5,
     & .2544039e-1, -.4395200e-1,-.3631877e-1,.1793128e-2,.3121600e-2,
     & -.6365539e-4,.4349632e-4,.6273942e-5,
     & .6541023e-2, -.5926933e-1,.3085867e-1,.1836850e-4,.2331795e-4,
     & .2716070e-4, .3283492e-3, 0.,
     & -.2182155e-1,-.1497413,-.3499282e-2,.1622326e-2,-.2015436e-3,
     & -.3092820e-4, .1339615e-4, 0.,
     & -.2479595e-1,.1987815e-1,.1844057e-2,-.6127611e-4,-.4987500e-5,
     & 0., 0., 0.,
     & -.2557178e-1,.4226421e-2,.5732041e-3,-.2296542e-4,.4740705e-5,
     & 0., 0., 0./
c
c if temperature is less than 500K
c
      if(tinp .lt. 500. .or. iflag .eq. 0) then
        cp = .24
        z = 1.
        xmu = 1.4584e-5*tinp**1.5/(tinp+110.33)
        xk = 5.9776e-6*tinp**1.5/(tinp+194.4)
        pr = .24*xmu/xk
        rho = pinp*airm0/(runiv*tinp)*2.4218e-2
        airm = airm0
        gamma = 1./(1.-runiv/airm/cp)
        return
      endif
c
c Find pressure range for interpolation
c
      if(pinp .lt. 1.e-4 .and. pinp .gt. 100.) then
        write(6,*) ' Input pressure of ',pinp,' is outside range of',
     &             ' available curve fit'
        ifatal = 1
        return
      endif
c
c Find pressures which bound the input pressure
c
      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
        ipm1 = ip -1
        go to 20
      endif
 10   continue
c
      write(6,*) ' Pressure is outside the range of available data'
      ifatal = 1
      return
 20   continue
c
      xi = alog(tinp*.0001)
c
c Specific heat
c
      do 30 i = 1,ncpmax(ip)
      if(tinp .gt. tcpmax(i,ip)) go to 30
```

```
      it = i
      go to 40
 30   continue
c
      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)
c
      if(intp  .eq. 0 .or. ipm1 .le. 0) then
        cp = exp(cpm2)
        go to 65
      endif
      do 50 i = 1,ncpmax(ipm1)
      if(tinp .gt. tcpmax(i,ipm1)) go to 50
      it = i
      go to 60
 50   continue
c
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - cp low'
      ifatal = 1
      return
 60   continue
      cpm1 = acp(it,ipm1)*xi**4+bcp(it,ipm1)*xi**3+ccp(it,ipm1)*xi**2
     &     + dcp(it,ipm1)*xi + ecp(it,ipm1)
c
c interpolation for specific heat
c
      alogcp = (cpm2 - cpm1)/(alog(ptb(ip))-alog(ptb(ipm1)))*
     &         (alog(pinp) - alog(ptb(ipm1))) + cpm1
      cp = exp(alogcp)
c
c Compressibility factor
c
 65   continue
      xi = tinp*.001
      do 70 i = 1,nzmax(ip)
      if(tinp .gt. tzmax(i,ip)) go to 70
      it = i
      go to 80
 70   continue
c
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - z high'
      ifatal = 1
      return
 80   continue
      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. ipm1 .le. 0) then
        z = z2
        go to 105
      endif

c
      do 90 i = 1,nzmax(ipm1)
      if(tinp .gt. tzmax(i,ipm1)) 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,ipm1) + bz(it,ipm1)*xi + cz(it,ipm1)*xi**2
     &    + dz(it,ipm1)*xi**3 + ez(it,ipm1)*xi**4
c
c interpolation of compressibility factor
c
      z = (z2 - z1)/(ptb(ip) - ptb(ipm1)) * (pinp - ptb(ipm1)) + z1
c
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. ipm1 .le. 0) then
        xmu = xmu2
        go to 145
      endif
c
      do 130 i = 1,nmumax(ipm1)
      if(tinp .gt. tmumax(i,ipm1)) go to 130
      it = i
      go to 140
 130  continue
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - mu low'
      ifatal = 1
      return
 140  continue
      xmu1 = amu(it,ipm1) + bmu(it,ipm1)*xi + cmu(it,ipm1)*xi**2
     &    + dmu(it,ipm1)*xi**3 + emu(it,ipm1)*xi**4 + fmu(it,ipm1)*xi**5
c
c interpolation of viscosity
c
      xmu = (xmu2 - xmu1)/(ptb(ip) - ptb(ipm1)) * (pinp - ptb(ipm1))
     &    + xmu1
c
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)*
```

```
     &       xi**2 + dk(it,ip)*xi + ek(it,ip)
      if(intp .eq. 0 .or. ipm1 .le. 0) then
        xk = exp(xk2)
        go to 185
      endif
c
      do 170 i = 1,nkmax(ipm1)
      if(tinp .gt. tkmax(i,ipm1)) go to 170
      it = i
      go to 180
 170  continue
c
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - K low'
      ifatal = 1
      return
 180  continue
      xk1 = ak(it,ipm1)*xi**4 + bk(it,ipm1)*xi**3 + ck(it,ipm1)*xi**2
     &     + dk(it,ipm1)*xi + ek(it,ipm1)
c
c interpolation for thermal conductivity
c
      alogk = (xk2 - xk1)/(alog(ptb(ip))-alog(ptb(ipm1)))*
     &          (alog(pinp) - alog(ptb(ipm1))) + xk1
      xk = exp(alogk)
c
c Prandtl number
c
 185  continue
      xi = tinp *.001
      do 190 i = 1,nprmax(ip)
      if(tinp .gt. tprmax(i,ip)) go to 190
      it = i
      go to 200
 190  continue
c
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - Pr high'
      ifatal = 1
      return
 200  continue
      pr2 = apr(it,ip) + bpr(it,ip)*xi + cpr(it,ip)*xi**2
     &   + dpr(it,ip)*xi**3 + epr(it,ip)*xi**4 + fpr(it,ip)*xi**5
      if(intp .eq. 0 .or. ipm1 .le. 0) then
        pr = pr2
        go to 300
      endif
c
      do 210 i = 1,nprmax(ipm1)
      if(tinp .gt. tprmax(i,ipm1)) go to 210
      it = i
      go to 220
 210  continue
      write(6,*) ' Temperature ',tinp, 'is outside the range of',
     &' available data - Pr low'
      ifatal = 1
      return
 220  continue
      pr1 = apr(it,ipm1) + bpr(it,ipm1)*xi + cpr(it,ipm1)*xi**2
     &   + dpr(it,ipm1)*xi**3 + epr(it,ipm1)*xi**4 + fpr(it,ipm1)*xi**5
c
c interpolation of Prandtl number
c
      pr = (pr2 - pr1)/(ptb(ip) - ptb(ipm1)) * (pinp - ptb(ipm1))
     &    + pr1
c
```

```
 300  continue
c
c compute density
c
      rho0 = pinp*airm0/(runiv*tinp)*2.4218e-2
      if(z .lt. 1.) z = 1.
      rho = rho0/z
c
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
c
      return
      end
c
      subroutine find(x0,y0,z0)
c
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), dpdxi(8), dpdeta(8), dpdzta(8)
      common /connty/ i1,j1,k1
      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,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
      logical blank,first,second,back
      dimension x(8), y(8), z(8), r(3), fj(3,3), dr(3)
c
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.
      ip1 = i1 + 1
      jp1 = j1 + 1
      kp1 = k1 + 1
c
      x(1) = xyz(i1,j1,k1,1)
      y(1) = xyz(i1,j1,k1,2)
      z(1) = xyz(i1,j1,k1,3)
c
      x(2) = xyz(ip1,j1,k1,1)
      y(2) = xyz(ip1,j1,k1,2)
      z(2) = xyz(ip1,j1,k1,3)
c
      x(3) = xyz(ip1,j1,kp1,1)
      y(3) = xyz(ip1,j1,kp1,2)
      z(3) = xyz(ip1,j1,kp1,3)
c
      x(4) = xyz(i1,j1,kp1,1)
      y(4) = xyz(i1,j1,kp1,2)
      z(4) = xyz(i1,j1,kp1,3)
c
```

```
      x(5) = xyz(i1,jp1,k1,1)
      y(5) = xyz(i1,jp1,k1,2)
      z(5) = xyz(i1,jp1,k1,3)
c
      x(6) = xyz(ip1,jp1,k1,1)
      y(6) = xyz(ip1,jp1,k1,2)
      z(6) = xyz(ip1,jp1,k1,3)
c
      x(7) = xyz(ip1,jp1,kp1,1)
      y(7) = xyz(ip1,jp1,kp1,2)
      z(7) = xyz(ip1,jp1,kp1,3)
c
      x(8) = xyz(i1,jp1,kp1,1)
      y(8) = xyz(i1,jp1,kp1,2)
      z(8) = xyz(i1,jp1,kp1,3)
c
      iter = 0
   10 continue
c
c compute shape function and it's derivatives
c
      call shape
c
c initialize jacobian matrix
c
      do 20 j = 1,3
      do 20 i=1,3
      fj(i,j) = 0.
   20 continue
c
      r(1) = x0
      r(2) = y0
      r(3) = z0
c
c get jacobian matrix
c
      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)*dpdzta(i)
      fj(2,3) = fj(2,3) + y(i)*dpdzta(i)
      fj(3,3) = fj(3,3) + z(i)*dpdzta(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)
c      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
c      write(6,*) dr
      sdr = abs(dr(1)) + abs(dr(2)) + abs(dr(3))
      xi = xi + dr(1)
      eta = 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
c
      subroutine shape
c
c This routine evaluate 3-D linear shape functions and their
c derivatives.
c
      common /pos/ xi,eta,zeta
      common /psidps/ psi(8), dpdxi(8), dpdeta(8), dpdzta(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 * opxize
      dpdeta(7) = -dpdeta(6)
      dpdeta(8) = -dpdeta(5)
c
      dpdzta(1) = -.125 * omxiet
      dpdzta(2) = -.125 * opxime
      dpdzta(3) = -.125 * opxiet
      dpdzta(4) = -.125 * omxipe
      dpdzta(5) = -dpdzta(1)
      dpdzta(6) = -dpdzta(2)
      dpdzta(7) = -dpdzta(3)
      dpdzta(8) = -dpdzta(4)
c
      return
      end
c
      SUBROUTINE GAUSS(FJAC,C,X,IDIM)
      common /save/ igsav,issav,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
C
C GAUSS ELIMNATION 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                  I,J,K,A,B,G,X,Y,Z,AMAT,BMAT,STATUS)
*****************************************************************************
C
C   Find the point (x,y,z) in the grid XYZ and return its (i,j,k) cell
```

```
C   number. We ASSUME that the given (i,j,k), (a,b,g), and subset are
C   valid.  These can be checked with STRTxx.
C
C      STATUS=1 - Unable to find the point without going out of the
C                 computational domain or active subet.  The computat-
C                 ional point returned indicates the direction to look..
C
      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
C
      PARAMETER (MITER=5)
      LOGICAL DIALOG
      INTEGER DI,DJ,DK
C
      DATA ISAV1,JSAV1,KSAV1/3*0/,ISAV2,JSAV2,KSAV2/3*0/
      DATA DIALOG/.FALSE./
c      DATA DIALOG/.true./
C
      STATUS= 0
C
C   Reset saved points used for checking if we're stuck on the same
C   point (or ping-ponging back and forth) when searching for the right
C   cell.  This would happen while following a point outside the
C   computational domain.
C
      ISAV1= 0
      ISAV2= 0
      DI= 1
      DJ= 1
      DK= 1
C
C   Maximum number of steps before we'll give up is max(IDIM,JDIM,KDIM).
C
      MSTEP= MAX(IDIM,JDIM,KDIM)
      NSTEP= 0
C
   20 CONTINUE
      NSTEP= NSTEP+1
C
      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
C
      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)
C
      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)
C
      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)
C
      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
C
      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
C
      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
C
      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)
     C                                              +XABG*(A*(B*G))
         YH = Y0+YA*A+YB*B+YG*G+YAB*(A*B)+YAG*(A*G)+YBG*(B*G)
     C                                              +YABG*(A*(B*G))
         ZH = Z0+ZA*A+ZB*B+ZG*G+ZAB*(A*B)+ZAG*(A*G)+ZBG*(B*G)
     C                                              +ZABG*(A*(B*G))
C
         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)
C
         AMAT(1,2)= XB + XAB*A + XBG*G + XABG*(A*G)
         AMAT(2,2)= YB + YAB*A + YBG*G + YABG*(A*G)
         AMAT(3,2)= ZB + ZAB*A + ZBG*G + ZABG*(A*G)
C
         AMAT(1,3)= XG + XAG*A + XBG*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)
C
         CALL INV3X3(AMAT,BMAT,ISTAT)
         IF (ISTAT.NE.0) THEN
            IF (DIALOG) WRITE(6,*) 'DEGENERATE VOLUME AT INDEX ',I,J,K
```

```
C
C   See if we're at the edge of the cell.  If so, move away from the
C   (possibly degenerate) edge and recompute the matrix.
C
            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
C
C   We're inside a cell and the transformation matrix is singular.  Move
C   to the next cell and try again.
C
            ELSE
               A  = DI+.5
               B  = DJ+.5
               G  = DK+.5
               GOTO 40
            ENDIF
         ENDIF
C
         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
C
C   If we're WAY off, don't bother with the error test.  In fact, go
C   ahead and try another cell.
C
         IF (ABS(A-.5).GT.3. .OR. ABS(B-.5).GT.3.
     C                       .OR. ABS(G-.5).GT.3.) THEN
            GOTO 40
C
C   Check iteration error and branch out if it's small enough.
C
         ELSE
            ERR2= DA*DA + DB*DB + DG*DG
            IF (ERR2/3.LE.1.E-4) GOTO 40
         ENDIF
         IF (ITER.LT.MITER) GOTO 30
C
   40 CONTINUE
C
C   The point is in this cell.
C
       IF (ABS(A-.5).LE..50005 .AND. ABS(B-.5).LE..50005
     C                         .AND. ABS(G-.5).LE..50005) THEN
         IF (DIALOG) WRITE(6,*) 'MATCH'
C
C   We've taken more steps then we're willing to wait...
C
       ELSE IF (NSTEP.GT.MSTEP) THEN
         STATUS= 1
         IF (DIALOG) WRITE(6,*) 'MORE THAN ',MSTEP,' STEPS'
C
```

```
C   Update our (i,j,k) guess, keeping it inbounds.
C
      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)
c         IF (g.LT.0.) JN  = MAX(JN-1,JS  )
c         IF (g.GT.1.) JN  = MIN(JN+1,JE-1)
c         IF (b.LT.0.) KN  = min(KN+1,KS  )
c         IF (b.GT.1.) KN  = max(KN-1,KE-1)
         IF (DIALOG) WRITE(*,*) 'TRY CELL INDEX ',IN,JN,KN
C
C   Check IBLANK for this cell.
C
         IF (BLANK) THEN
            IN1 = IN+1
            JN1 = JN+1
            KN1 = KN+1
            IF (  IBLANK(IN,JN,KN)   .EQ.0 .OR. IBLANK(IN1,JN,KN)   .EQ.0
     C       .OR. IBLANK(IN,JN1,KN) .EQ.0 .OR. IBLANK(IN1,JN1,KN) .EQ.0
     C       .OR. IBLANK(IN,JN,KN1) .EQ.0 .OR. IBLANK(IN1,JN,KN1) .EQ.0
     C       .OR. IBLANK(IN,JN1,KN1).EQ.0 .OR. IBLANK(IN1,JN1,KN1).EQ.0)
     C                                                                THEN
               STATUS= 1
               IF (DIALOG) WRITE(6,*) 'EXTRAPOLATE'
               GOTO 50
            ENDIF
         ENDIF
C
C   Not repeating a previous point.  Use the new (i,j,k) and try again.
C
         IF (      (IN.NE.ISAV1 .OR. JN.NE.JSAV1 .OR. KN.NE.KSAV1)
     C        .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
C
C   We've been here before...
C
         ELSE
C
C   It seems to be outside the domain.  We would have to extrapolate to
C   find it.
C
            STATUS= 1
            IF (DIALOG) WRITE(6,*) 'EXTRAPOLATE'
         ENDIF
      ENDIF
C
   50 CONTINUE
```

```
      RETURN
      END
*** End of Subroutine Cell3



*************************************************************************
      SUBROUTINE CLOSE3(ISRCH,IGRID,NSUBS,I,J,K,A,B,G,
     C                  X,Y,Z,ISUB,STATUS)
*************************************************************************
C
C   Find the cell containing the given (x,y,z) values and return its
C   (i,j,k) indices and subset number.  We will use several different
C   strategies to try to do this efficiently, based on the value of
C   ISRCH:
C
C      ISRCH= 0 - General search.
C             1 - Search from previous point.
C            -n - Coming from grid n.
C
C      STATUS=1 - (x,y,z) point could not be found.
C
      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,isav,jsav,ksav,asav,bsav,gsav,
     &     first,back,second,ifatal
      LOGICAL BLANK,first,second,back
      INTEGER STATUS
C
      STATUS= 0
C
C   Search from previous point.
C
      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)
            ie = isubs(2)
            js = jsubs(1)
            je = jsubs(2)
            ks = ksubs(1)
            ke = ksubs(2)
            CALL PSRCH3(is,ie,js,je,ks,ke,I,J,K,A,B,G,X,Y,Z,ISTAT)
            IF (ISTAT.EQ.0) GOTO 10
         ENDIF
C
C   Try edges next.
C
         CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
         IF (ISTAT.EQ.0) GOTO 10
C
C   Try faces next.
C
         CALL FSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
         IF (ISTAT.EQ.0) GOTO 10
C
C   Try hole boundaries next.
C
```

```
          CALL HSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
C
C   Coming from grid n...
C
        ELSE IF (ISRCH.LT.0) THEN
C
C   Try edges first.
C
          CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
C
C   If that fails, try closest point with IBLANK = -ISRCH.
C
          CALL NSRCH3(-ISRCH,I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
C
C   General search.
C
        ELSE
C
C   Try edges.
C
          CALL ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
C
C   Try faces next.
C
          CALL FSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
C
C   Try hole boundaries next.
C
          CALL HSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,ISTAT)
          IF (ISTAT.EQ.0) GOTO 10
        ENDIF
C
C   Nothing worked.  Sorry.
C
        STATUS= 1
        IGSAV = 0
C
   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.
```

```
C
      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
C
C   No previous points.  Establish a minimum distance.
C
      IF (NPTS.EQ.0) D2MIN= 1.E35
      DO 10 K= KS,KE
      DO 10 J= JS,JE
      DO 10 I= IS,IE
         IF (.NOT.BLANK .OR. IBLANK(I,J,K).NE.0) THEN
            D2= (X-XYZ(I,J,K,1))**2+(Y-XYZ(I,J,K,2))**2
     C                             +(Z-XYZ(I,J,K,3))**2
C
C   Got a closer point.  This replaces all previous points.
C
            IF (D2.LT.D2MIN) THEN
               D2MIN= D2
               NPTS = 1
               IPTS(1,NPTS)= I
               IPTS(2,NPTS)= J
               IPTS(3,NPTS)= K
C
C   Tie for closest point.  Add this to the list.
C
            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
   10    CONTINUE
      RETURN
      END
*** End of Subroutine DMin3



**************************************************************************
      SUBROUTINE ESRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,STATUS)
**************************************************************************
C
C   Search from closest point on any edge.
C
C      STATUS=1 - Couldn't find the point.
C
      common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
     &                iblank(151,40,110), blank,isubs(2),
     &                jsubs(2),ksubs(2)
      LOGICAL BLANK
      INTEGER STATUS
C
      PARAMETER (MPTS=5)
      DIMENSION IPTS(3,MPTS)
C
      STATUS= 0
C
C   Loop through the subsets.
C
```

```
      IS= ISUBS(1)
      IE= ISUBS(2)
      JS= JSUBS(1)
      JE= JSUBS(2)
      KS= KSUBS(1)
      KE= KSUBS(2)
C
C   Find the closest point(s) on any edge.  Be careful of degenerate
C   subsets.
C
      NPTS = 0
      D2MIN= 1.E35
C
      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
C
      CALL DMIN3(IS,IS,JS+1,JE-1,KS,KS,X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      IF (IS.NE.IE) THEN
         CALL DMIN3(IE,IE,JS+1,JE-1,KS,KS,X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      ENDIF
      IF (KS.NE.KE) THEN
         CALL DMIN3(IS,IS,JS+1,JE-1,KE,KE,X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
         IF (IS.NE.IE) THEN
            CALL DMIN3(IE,IE,JS+1,JE-1,KE,KE,
     C                 X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
         ENDIF
      ENDIF
C
      CALL DMIN3(IS,IS,JS,JS,KS+1,KE-1,
     C           X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      IF (IS.NE.IE) THEN
         CALL DMIN3(IE,IE,JS,JS,KS+1,KE-1,
     C              X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
      ENDIF
      IF (JS.NE.JE) THEN
         CALL DMIN3(IS,IS,JE,JE,KS+1,KE-1,
     C              X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
         IF (IS.NE.IE) THEN
            CALL DMIN3(IE,IE,JE,JE,KS+1,KE-1,
     C                 X,Y,Z,MPTS,NPTS,IPTS,D2MIN)
         ENDIF
      ENDIF
C
C   Search from each of these points.
C
      DO 10 II= 1,NPTS
c     ISUB  = IISUB
         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,I,J,K,A,B,G,X,Y,Z,ISTAT)
         IF (ISTAT.EQ.0) GOTO 30
   10    CONTINUE
C
```

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

```
C
         DO 10 II= 1,NPTS
c      ISUB  = IISUB
               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,
     C                     I,J,K,A,B,G,X,Y,Z,ISTAT)
               IF (ISTAT.EQ.0) GOTO 30
   10          CONTINUE
C
C    Failed all searches.
C
         STATUS= 1
C
   30 CONTINUE
      RETURN
      END
*** End of Subroutine FSrch3



************************************************************************
      SUBROUTINE HSRCH3(I,J,K,A,B,G,X,Y,Z,ISUB,STATUS)
************************************************************************
C
C    Search from closest point on a hole boundary.
C
C       STATUS=1 - Couldn't find the point.
C
       common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
     &                 iblank(151,40,110), blank,isubs(2),
     &                 jsubs(2),ksubs(2)
       LOGICAL BLANK
       INTEGER STATUS
C
       PARAMETER (MPTS=5)
       DIMENSION IPTS(3,MPTS)
C
       STATUS= 0
C
C    Loop through the subsets.
C
       if(blank) then
               IS= ISUBS(1)
               IE= ISUBS(2)
               JS= JSUBS(1)
               JE= JSUBS(2)
               KS= KSUBS(1)
               KE= KSUBS(2)
C
C    Search for the closest point(s) on the boundary of any hole.
C
               NPTS = 0
               D2MIN= 1.E35
               DO 10 KK= KS,KE
               DO 10 JJ= JS,JE
               DO 10 II= IS,IE
                  IF (IBLANK(II,JJ,KK).NE.0 .AND.
     C             (      (II.NE.1    .AND. IBLANK(II-1,JJ,KK).EQ.0)
     C              .OR. (II.NE.IDIM .AND. IBLANK(II+1,JJ,KK).EQ.0)
```

```
     C            .OR. (JJ.NE.1    .AND. IBLANK(II,JJ-1,KK).EQ.0)
     C            .OR. (JJ.NE.JDIM .AND. IBLANK(II,JJ+1,KK).EQ.0)
     C            .OR. (KK.NE.1    .AND. IBLANK(II,JJ,KK-1).EQ.0)
     C            .OR. (KK.NE.KDIM .AND. IBLANK(II,JJ,KK+1).EQ.0))) THEN
                   D2=  (X-XYZ(II,JJ,KK,1))**2
     C                 +(Y-XYZ(II,JJ,KK,2))**2
     C                 +(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
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
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,
     C                      I,J,K,A,B,G,X,Y,Z,ISTAT)
                IF (ISTAT.EQ.0) GOTO 30
   20           CONTINUE
C
      ENDIF
C
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
C    Invert the 3x3 matrix A.  If A is singular, do our best to find the
C    pseudo-inverse.
C
C       STATUS=1 - A has one dependent column.
C       STATUS=2 - A has two dependent columns.
C       STATUS=3 - A is zero.
```

```
C
      DIMENSION A(3,3),AINV(3,3)
      INTEGER STATUS
C
      DIMENSION TMP(3,3),WORK(3),S(3),E(3),U(3,3),V(3,3),SIU(3,3)
C
      STATUS= 0
C
      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)
C
      AINV(2,1)= A(3,1)*A(2,3) - A(2,1)*A(3,3)
      AINV(2,2)= A(1,1)*A(3,3) - A(3,1)*A(1,3)
      AINV(2,3)= A(2,1)*A(1,3) - A(1,1)*A(2,3)
C
      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)
C
      DET= A(1,1)*AINV(1,1) + A(2,1)*AINV(1,2) + A(3,1)*AINV(1,3)
C
C   Matrix is nonsingular.  Finish up AINV.
C
      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
C
C   Matrix is singular.  Do a singular value decomposition to construct
C   the pseudo-inverse.  Use LINPACK routine SSVDC.
C
      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
C
C              -1
C   Compute V S  U.
C
         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
C
C   Start out assuming S is a diagonal matrix.
```

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



*************************************************************************
      SUBROUTINE NSRCH3(N,I,J,K,A,B,G,X,Y,Z,ISUB,STATUS)
*************************************************************************
C
C   Search from closest point coming from grid N.
C
C      STATUS=1 - Couldn't find the point.
C
      common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
     &                iblank(151,40,110), blank,isubs(2),
     &                jsubs(2),ksubs(2)
      LOGICAL BLANK
```

```
      INTEGER STATUS
C
      PARAMETER (MPTS=5)
      DIMENSION IPTS(3,MPTS)
C
      STATUS= 0
C
C   Loop through the subsets.
C
      IF (BLANK) THEN
            IS= ISUBS(1)
            IE= ISUBS(2)
            JS= JSUBS(1)
            JE= JSUBS(2)
            KS= KSUBS(1)
            KE= KSUBS(2)
C
C   Search for the closest point(s) on the boundary of any hole.
C
            NPTS = 0
            D2MIN= 1.E35
            DO 10 KK= KS,KE
            DO 10 JJ= JS,JE
            DO 10 II= IS,IE
               IF (IBLANK(II,JJ,KK).EQ.-N) THEN
                  D2=  (X-XYZ(II,JJ,KK,1))**2
     C                +(Y-XYZ(II,JJ,KK,2))**2
     C                +(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
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
   10          CONTINUE
C
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,
     C                     I,J,K,A,B,G,X,Y,Z,ISTAT)
               IF (ISTAT.EQ.0) GOTO 30
   20          CONTINUE
C
      ENDIF
C
```

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



************************************************************************
      SUBROUTINE PSRCH3(IS,IE,JS,JE,KS,KE,I,J,K,A,B,G,X,Y,Z,STATUS)
************************************************************************
C
C   Search for a point using CELL3.
C
C      STATUS=1 - Couldn't find the point.
C
      common /grdxyz/ xyz(151,40,110,3),idim,jdim,kdim,
     &                iblank(151,40,110), blank,isubs(2),
     &                jsubs(2),ksubs(2)
      LOGICAL BLANK
      INTEGER STATUS
C
      DIMENSION AMAT(3,3),BMAT(3,3)
C
      STATUS= 0
C
      CALL CELL3 (IS,IE,JS,JE,KS,KE,
     C            I,J,K,A,B,G,X,Y,Z,AMAT,BMAT,ISTAT)
      IF (ISTAT.NE.0) THEN
         STATUS= 1
         GOTO 10
      ENDIF
C
   10 CONTINUE
      RETURN
      END
*** End of Subroutine PSrch3



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



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

```
**********************************************************************
      integer ldx,n,p,ldu,ldv,job,info
      real x(ldx,1),s(1),e(1),u(ldu,1),v(ldv,1),work(1)
c
Caveat receptor.  (Jack) dongarra@anl-mcs, (Eric Grosse) research!ehg
Compliments of netlib   Fri Dec 12 16:40:31 CST 1986
c
c     ssvdc is a subroutine to reduce a real nxp matrix x by
c     orthogonal transformations u and v to diagonal form.  the
c     diagonal elements s(i) are the singular values of x.  the
c     columns of u are the corresponding left singular vectors,
c     and the columns of v the right singular vectors.
c
c     on entry
c
c         x         real(ldx,p), where ldx.ge.n.
c                   x contains the matrix whose singular value
c                   decomposition is to be computed.  x is
c                   destroyed by ssvdc.
c
c         ldx       integer.
c                   ldx is the leading dimension of the array x.
c
c         n         integer.
c                   n is the number of rows of the matrix x.
c
c         p         integer.
c                   p is the number of columns of the matrix x.
c
c         ldu       integer.
c                   ldu is the leading dimension of the array u.
c                   (see below).
c
c         ldv       integer.
c                   ldv is the leading dimension of the array v.
c                   (see below).
c
c         work      real(n).
c                   work is a scratch array.
c
c         job       integer.
c                   job controls the computation of the singular
c                   vectors.  it has the decimal expansion ab
c                   with the following meaning
c
c                        a.eq.0    do not compute the left singular
c                                  vectors.
c                        a.eq.1    return the n left singular vectors
c                                  in u.
c                        a.ge.2    return the first min(n,p) singular
c                                  vectors in u.
c                        b.eq.0    do not compute the right singular
c                                  vectors.
c                        b.eq.1    return the right singular vectors
c                                  in v.
c
c     on return
c
c         s         real(mm), where mm=min(n+1,p).
c                   the first min(n,p) entries of s contain the
c                   singular values of x arranged in descending
c                   order of magnitude.
c
c         e         real(p).
c                   e ordinarily contains zeros.  however see the
c                   discussion of info for exceptions.
```

```
c
c         u         real(ldu,k), where ldu.ge.n.  if joba.eq.1 then
c                                 k.eq.n, if joba.ge.2 then
c                                 k.eq.min(n,p).
c                   u contains the matrix of left singular vectors.
c                   u is not referenced if joba.eq.0.  if n.le.p
c                   or if joba.eq.2, then u may be identified with x
c                   in the subroutine call.
c
c         v         real(ldv,p), where ldv.ge.p.
c                   v contains the matrix of right singular vectors.
c                   v is not referenced if job.eq.0.  if p.le.n,
c                   then v may be identified with x in the
c                   subroutine call.
c
c         info      integer.
c                   the singular values (and their corresponding
c                   singular vectors) s(info+1),s(info+2),...,s(m)
c                   are correct (here m=min(n,p)).  thus if
c                   info.eq.0, all the singular values and their
c                   vectors are correct.  in any event, the matrix
c                   b = trans(u)*x*v is the bidiagonal matrix
c                   with the elements of s on its diagonal and the
c                   elements of e on its super-diagonal (trans(u)
c                   is the transpose of u).  thus the singular
c                   values of x and b are the same.
c
c     linpack. this version dated 03/19/79 .
c              correction to shift calculation made 2/85.
c     g.w. stewart, university of maryland, argonne national lab.
c
c     ***** uses the following functions and subprograms.
c
c     external srot
c     blas saxpy,sdot,sscal,sswap,snrm2,srotg
c     fortran abs,amax1,max0,min0,mod,sqrt
c
c     internal variables
c
      integer i,iter,j,jobu,k,kase,kk,l,ll,lls,lm1,lp1,ls,lu,m,maxit,
     *        mm,mm1,mp1,nct,nctp1,ncu,nrt,nrtp1
      real sdot,t,r
      real b,c,cs,el,emm1,f,g,snrm2,scale,shift,sl,sm,sn,smm1,t1,test,
     *     ztest
      logical wantu,wantv
c
c
c     set the maximum number of iterations.
c
      maxit = 30
c
c     determine what is to be computed.
c
      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.
c
c     reduce x to bidiagonal form, storing the diagonal elements
c     in s and the super-diagonal elements in e.
c
      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
c
c           compute the transformation for the l-th column and
c           place the l-th diagonal in s(l).
c
            s(l) = snrm2(n-l+1,x(l,l),1)
            if (s(l) .eq. 0.0e0) go to 10
               if (x(l,l) .ne. 0.0e0) s(l) = sign(s(l),x(l,l))
               call sscal(n-l+1,1.0e0/s(l),x(l,l),1)
               x(l,l) = 1.0e0 + x(l,l)
   10       continue
            s(l) = -s(l)
   20    continue
         if (p .lt. lp1) go to 50
         do 40 j = lp1, p
            if (l .gt. nct) go to 30
            if (s(l) .eq. 0.0e0) go to 30
c
c              apply the transformation.
c
               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       continue
c
c           place the l-th row of x into  e for the
c           subsequent calculation of the row transformation.
c
            e(j) = x(l,j)
   40    continue
   50    continue
         if (.not.wantu .or. l .gt. nct) go to 70
c
c           place the transformation in u for subsequent back
c           multiplication.
c
            do 60 i = l, n
               u(i,l) = x(i,l)
   60       continue
   70    continue
         if (l .gt. nrt) go to 150
c
c           compute the l-th row transformation and place the
c           l-th super-diagonal in e(l).
c
            e(l) = snrm2(p-l,e(lp1),1)
            if (e(l) .eq. 0.0e0) go to 80
               if (e(lp1) .ne. 0.0e0) e(l) = sign(e(l),e(lp1))
               call sscal(p-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
c
c              apply the transformation.
c
               do 90 i = lp1, n
                  work(i) = 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 = lp1, p
                  call saxpy(n-l,-e(j)/e(lp1),work(lp1),1,x(lp1,j),1)
  110          continue
  120       continue
            if (.not.wantv) go to 140
c
c              place the transformation in v for subsequent
c              back multiplication.
c
               do 130 i = lp1, p
                  v(i,l) = e(i)
  130          continue
  140       continue
  150    continue
  160 continue
  170 continue
c
c     set up the final bidiagonal matrix or order m.
c
      m = min0(p,n+1)
      nctp1 = nct + 1
      nrtp1 = nrt + 1
      if (nct .lt. p) s(nctp1) = x(nctp1,nctp1)
      if (n .lt. m) s(m) = 0.0e0
      if (nrtp1 .lt. m) e(nrtp1) = x(nrtp1,m)
      e(m) = 0.0e0
c
c     if required, generate u.
c
      if (.not.wantu) go to 300
         if (ncu .lt. nctp1) go to 200
         do 190 j = nctp1, 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+1,u(l,l),1,u(l,j),1)/u(l,l)
                  call saxpy(n-l+1,t,u(l,l),1,u(l,j),1)
  210          continue
  220          continue
               call sscal(n-l+1,-1.0e0,u(l,l),1)
               u(l,l) = 1.0e0 + u(l,l)
               lm1 = l - 1
               if (lm1 .lt. 1) go to 240
               do 230 i = 1, lm1
                  u(i,l) = 0.0e0
  230          continue
  240          continue
            go to 270
  250       continue
               do 260 i = 1, n
                  u(i,l) = 0.0e0
  260          continue
               u(l,l) = 1.0e0
  270       continue
  280    continue
  290    continue
```

```
  300 continue
c
c     if it is required, generate v.
c
      if (.not.wantv) go to 350
         do 340 ll = 1, p
            l = p - ll + 1
            lp1 = l + 1
            if (l .gt. nrt) go to 320
            if (e(l) .eq. 0.0e0) go to 320
               do 310 j = lp1, p
                  t = -sdot(p-l,v(lp1,l),1,v(lp1,j),1)/v(lp1,l)
                  call saxpy(p-l,t,v(lp1,l),1,v(lp1,j),1)
  310          continue
  320       continue
            do 330 i = 1, p
               v(i,l) = 0.0e0
  330       continue
            v(l,l) = 1.0e0
  340    continue
  350 continue
c
c     main iteration loop for the singular values.
c
      mm = m
      iter = 0
  360 continue
c
c        quit if all the singular values have been found.
c
c     ...exit
         if (m .eq. 0) go to 620
c
c        if too many iterations have been performed, set
c        flag and return.
c
         if (iter .lt. maxit) go to 370
            info = m
c     ......exit
            go to 620
  370    continue
c
c        this section of the program inspects for
c        negligible elements in the s and e arrays.  on
c        completion the variables kase and l are set as follows.
c
c           kase = 1     if s(m) and e(l-1) are negligible and l.lt.m
c           kase = 2     if s(l) is negligible and l.lt.m
c           kase = 3     if e(l-1) is negligible, l.lt.m, and
c                        s(l), ..., s(m) are not negligible (qr step).
c           kase = 4     if e(m-1) is negligible (convergence).
c
         do 390 ll = 1, m
            l = m - ll
c        ...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
c        ......exit
               go to 400
  380       continue
  390    continue
  400    continue
         if (l .ne. m - 1) go to 410
```

```
            kase = 4
         go to 480
  410    continue
            lp1 = l + 1
            mp1 = m + 1
            do 430 lls = lp1, mp1
               ls = m - lls + lp1
c           ...exit
               if (ls .eq. l) go to 440
               test = 0.0e0
               if (ls .ne. m) test = test + abs(e(ls))
               if (ls .ne. l + 1) test = test + abs(e(ls-1))
               ztest = test + abs(s(ls))
               if (ztest .ne. test) go to 420
                  s(ls) = 0.0e0
c           ......exit
                  go to 440
  420          continue
  430       continue
  440       continue
            if (ls .ne. l) go to 450
               kase = 3
            go to 470
  450       continue
            if (ls .ne. m) go to 460
               kase = 1
            go to 470
  460       continue
               kase = 2
               l = ls
  470       continue
  480    continue
         l = l + 1
c
c        perform the task indicated by kase.
c
         go to (490,520,540,570), kase
c
c        deflate negligible s(m).
c
  490    continue
            mm1 = m - 1
            f = e(m-1)
            e(m-1) = 0.0e0
            do 510 kk = 1, mm1
               k = mm1 - kk + 1
               t1 = s(k)
               call srotg(t1,f,cs,sn)
               s(k) = t1
               if (k .eq. l) go to 500
                  f = -sn*e(k-1)
                  e(k-1) = cs*e(k-1)
  500          continue
               if (wantv) call srot(p,v(1,k),1,v(1,m),1,cs,sn)
  510       continue
         go to 610
c
c        split at negligible s(l).
c
  520    continue
            f = e(l-1)
            e(l-1) = 0.0e0
            do 530 k = l, m
               t1 = s(k)
               call srotg(t1,f,cs,sn)
               s(k) = t1
```

```
               f = -sn*e(k)
               e(k) = cs*e(k)
               if (wantu) call srot(n,u(1,k),1,u(1,l-1),1,cs,sn)
  530       continue
         go to 610
c
c        perform one qr step.
c
  540    continue
c
c           calculate the shift.
c
            scale = amax1(abs(s(m)),abs(s(m-1)),abs(e(m-1)),abs(s(l)),
     *                    abs(e(l)))
            sm = s(m)/scale
            smm1 = s(m-1)/scale
            emm1 = e(m-1)/scale
            sl = s(l)/scale
            el = e(l)/scale
            b = ((smm1 + sm)*(smm1 - sm) + emm1**2)/2.0e0
            c = (sm*emm1)**2
            shift = 0.0e0
            if (b .eq. 0.0e0 .and. c .eq. 0.0e0) go to 550
               shift = sqrt(b**2+c)
               if (b .lt. 0.0e0) shift = -shift
               shift = c/(b + shift)
  550       continue
            f = (sl + sm)*(sl - sm) + shift
            g = sl*el
c
c           chase zeros.
c
            mm1 = m - 1
            do 560 k = l, mm1
               call srotg(f,g,cs,sn)
               if (k .ne. l) e(k-1) = f
               f = cs*s(k) + sn*e(k)
               e(k) = cs*e(k) - sn*s(k)
               g = sn*s(k+1)
               s(k+1) = cs*s(k+1)
               if (wantv) call srot(p,v(1,k),1,v(1,k+1),1,cs,sn)
               call srotg(f,g,cs,sn)
               s(k) = f
               f = cs*e(k) + sn*s(k+1)
               s(k+1) = -sn*e(k) + cs*s(k+1)
               g = sn*e(k+1)
               e(k+1) = cs*e(k+1)
               if (wantu .and. k .lt. n)
     *            call srot(n,u(1,k),1,u(1,k+1),1,cs,sn)
  560       continue
            e(m-1) = f
            iter = iter + 1
         go to 610
c
c        convergence.
c
  570    continue
c
c           make the singular value  positive.
c
            if (s(l) .ge. 0.0e0) go to 580
               s(l) = -s(l)
               if (wantv) call sscal(p,-1.0e0,v(1,l),1)
  580       continue
c
c           order the singular value.
```

```
c
  590       if (l .eq. mm) go to 600
c           ...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. l .lt. p)
     *            call sswap(p,v(1,l),1,v(1,l+1),1)
               if (wantu .and. l .lt. n)
     *            call sswap(n,u(1,l),1,u(1,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 sswap (n,sx,incx,sy,incy)
**********************************************************************
c
c     interchanges two vectors.
c     uses unrolled loops for increments equal to 1.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),stemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      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 not equal
c         to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        stemp = sx(ix)
        sx(ix) = sy(iy)
        sy(iy) = stemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
c
c       clean-up loop
c
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        stemp = sx(i)
        sx(i) = sy(i)
```

```
        sy(i) = stemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,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,incx,sy,incy)
*****************************************************************************
c
c     constant times a vector plus a vector.
c     uses unrolled loop for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),sa
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (sa .eq. 0.0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        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
        sy(iy) = sy(iy) + sa*sx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
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 mp1 = m + 1
      do 50 i = mp1,n,4
        sy(i) = sy(i) + sa*sx(i)
        sy(i + 1) = sy(i + 1) + sa*sx(i + 1)
```

```
        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,incx,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,mp1,n
c
      stemp = 0.0e0
      sdot = 0.0e0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
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)
        ix = ix + incx
        iy = iy + incy
   10 continue
      sdot = stemp
      return
c
c        code for both increments equal to 1
c
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
   40 mp1 = m + 1
      do 50 i = mp1,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)
**********************************************************************
      integer          next
      real   sx(1),  cutlo, cuthi, hitest, sum, xmax, zero, one
      data   zero, one /0.0e0, 1.0e0/
c
c     euclidean norm of the n-vector stored in sx() with storage
c     increment incx .
c     if    n .le. 0 return with result = 0.
c     if n .ge. 1 then incx must be .ge. 1
c
c           c.l.lawson, 1978 jan 08
c
c     four phase method     using two built-in constants that are
c     hopefully applicable to all machines.
c         cutlo = maximum of  sqrt(u/eps)  over all known machines.
c         cuthi = minimum of  sqrt(v)      over all known machines.
c     where
c         eps = smallest no. such that eps + 1. .gt. 1.
c         u   = smallest positive no.   (underflow limit)
c         v   = largest  no.            (overflow  limit)
c
c     brief outline of algorithm..
c
c     phase 1    scans zero components.
c     move to phase 2 when a component is nonzero and .le. cutlo
c     move to phase 3 when a component is .gt. cutlo
c     move to phase 4 when a component is .ge. cuthi/m
c     where m = n for x() real and m = 2*n for complex.
c
c     values for cutlo and cuthi..
c     from the environmental parameters listed in the imsl converter
c     document the limiting values are as follows..
c     cutlo, s.p.   u/eps = 2**(-102) for  honeywell.  close seconds are
c                   univac and dec at 2**(-103)
c                   thus cutlo = 2**(-51) = 4.44089e-16
c     cuthi, s.p.   v = 2**127 for univac, honeywell, and dec.
c                   thus cuthi = 2**(63.5) = 1.30438e19
c     cutlo, d.p.   u/eps = 2**(-67) for honeywell and dec.
c                   thus cutlo = 2**(-33.5) = 8.23181d-11
c     cuthi, d.p.   same as s.p.  cuthi = 1.30438d19
c     data cutlo, cuthi / 8.232d-11,  1.304d19 /
c     data cutlo, cuthi / 4.441e-16,  1.304e19 /
      data cutlo, cuthi / 4.441e-16,  1.304e19 /
c
      if(n .gt. 0) go to 10
         snrm2  = zero
         go to 300
c
   10 assign 30 to next
      sum = zero
      nn = n * incx
c                                                 begin main loop
      i = 1
   20    go to next,(30, 50, 70, 110)
   30 if( abs(sx(i)) .gt. cutlo) go to 85
      assign 50 to next
      xmax = zero
c
c                       phase 1.  sum is zero
c
   50 if( sx(i) .eq. zero) go to 200
      if( abs(sx(i)) .gt. cutlo) go to 85
c
c                               prepare for phase 2.
      assign 70 to next
```

```
      go to 105
c
c                                 prepare for phase 4.
c
  100 i = j
      assign 110 to next
      sum = (sum / sx(i)) / sx(i)
  105 xmax = abs(sx(i))
      go to 115
c
c                     phase 2.  sum is small.
c                               scale to avoid destructive underflow.
c
   70 if( abs(sx(i)) .gt. cutlo ) go to 75
c
c                       common code for phases 2 and 4.
c                       in phase 4 sum is large.  scale to avoid overflow.
c
  110 if( abs(sx(i)) .le. xmax ) go to 115
         sum = one + sum * (xmax / sx(i))**2
         xmax = abs(sx(i))
         go to 200
c
  115 sum = sum + (sx(i)/xmax)**2
      go to 200
c
c
c                    prepare for phase 3.
c
   75 sum = (sum * xmax) * xmax
c
c
c     for real or d.p. set hitest = cuthi/n
c     for complex      set hitest = cuthi/(2*n)
c
   85 hitest = cuthi/float( n )
c
c                     phase 3.  sum is mid-range.  no scaling.
c
      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
c
  200 continue
      i = i + incx
      if ( i .le. nn ) go to 20
c
c               end of main loop.
c
c               compute square root and adjust for scaling.
c
      snrm2 = xmax * sqrt(sum)
  300 continue
      return
      end
*** End of Function SNRM2



**********************************************************************
      subroutine srot (n,sx,incx,sy,incy,c,s)
**********************************************************************
c
```

```
c     applies a plane rotation.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),stemp,c,s
      integer i,incx,incy,ix,iy,n
c
      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 not equal
c         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 = c*sx(ix) + s*sy(iy)
        sy(iy) = c*sy(iy) - s*sx(ix)
        sx(ix) = stemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
   20 do 30 i = 1,n
        stemp = c*sx(i) + s*sy(i)
        sy(i) = c*sy(i) - s*sx(i)
        sx(i) = stemp
   30 continue
      return
      end
*** End of Subroutine SRot



**********************************************************************
      subroutine srotg(sa,sb,c,s)
**********************************************************************
c
c     construct givens plane rotation.
c     jack dongarra, linpack, 3/11/78.
c
      real sa,sb,c,s,roe,scale,r,z
c
      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 sscal(n,sa,sx,incx)
*****************************************************************************
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to 1.
c     jack dongarra, linpack, 3/11/78.
c
      real sa,sx(1)
      integer i,incx,m,mp1,n,nincx
c
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        sx(i) = sa*sx(i)
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sx(i) = sa*sx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        sx(i) = sa*sx(i)
        sx(i + 1) = sa*sx(i + 1)
        sx(i + 2) = sa*sx(i + 2)
        sx(i + 3) = sa*sx(i + 3)
        sx(i + 4) = sa*sx(i + 4)
   50 continue
      return
      end
*** End of Subroutine SSCal



*****************************************************************************
      SUBROUTINE ZERO(LEN,ARRAY)
*****************************************************************************
C
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   have been removed.
C
      CHARACTER*(*) STRING
C
      CHARACTER NULL,TAB
C
C   Initialize the null and tab characters.
C
      NULL= CHAR(0)
      TAB = CHAR(9)
C
C   Loop backwards through the character string and find the last
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                          .AND. STRING(L:L).NE.TAB) THEN
            LSTRIN= L
            GOTO 20
         ENDIF
   10    CONTINUE
C
C   ALL blank or null or tabs!
C
      LSTRIN= 0
C
   20 CONTINUE
      RETURN
      END
*** End of Subroutine Trim



*************************************************************************
      SUBROUTINE UPCASE(STRING)
*************************************************************************
C
C   Convert this character string to upper case.
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
```