SANDIA REPORT

SAND95-2547 • UC-905

Unlimited Release

Printed November 1995

RECEIVED
DEC 14 1995
OSTI

CUERVO – A Finite Element Computer Program for Nonlinear Scalar Transport Problems

Melinda B. Sirman, David K. Gartling

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.

MASTER

SF2900Q(8-81)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

TM

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A06
Microfiche copy: A01

SAND95-2547
Unlimited Release
Printed November 1995

**CUERVO -
A FINITE ELEMENT COMPUTER PROGRAM
FOR NONLINEAR SCALAR TRANSPORT
PROBLEMS**

Melinda B. Sirman and David K. Gartling
Engineering Sciences Center
Sandia National Laboratories
Albuquerque, New Mexico 87185

ABSTRACT

CUERVO is a finite element code that is designed for the solution of multi-dimensional field problems described by a general nonlinear, advection-diffusion equation. The code is also applicable to field problems described by diffusion, Poisson or Laplace equations. The finite element formulation and the associated numerical methods used in CUERVO are outlined here; detailed instructions for use of the code are also presented. Example problems are provided to illustrate the use of the code.

Contents

1	Introduction	1
2	Computational Method	4
2.1	Finite Element Formulation	4
2.2	Computational Details	6
3	Program Overview	8
3.1	Program Features	8
3.2	Program Organization	8
4	Input Guide	11
4.1	Input Syntax	12
4.2	Title Data Block	14
4.3	Material Data Block	15
4.4	Problem Definition Data Block	20
4.5	Solution Data Block	28
4.6	Post-processing Data Block	35
4.7	Time Function Data Block	37
4.8	Variable Function Data Block	38
4.9	User Constants Data Block	39
4.10	Termination Data	40
4.11	User Supplied Subroutines	41
4.11.1	Material Property Subroutines	41
4.11.2	Volumetric Sources	45
4.11.3	Material Velocity	46

4.11.4	Boundary Conditions	47
4.12	Initial Conditions and Restarts	52
4.12.1	Initial Condition Subroutine	52
4.13	Error Messages	54
5	Code Installation and Access	55
5.1	FORTTRAN Coding and System Dependencies	55
5.2	File Formats	56
5.3	File Usage	57
5.4	Access to the Code	57
6	Example Problems	59
6.1	Problem 1 - Electrostatic Field Problem	59
6.2	Problem 2 - Time-Independent Thermal Diffusion	64
6.3	Problem 3 - Time-Dependent Advection-Diffusion	69
7	References	75
	Appendix A - Summary of Input Commands	77
	Appendix B - Common Block and Array Storage	83

1 Introduction

The CUERVO computer code is a general purpose program designed for the solution of field problems described by a general, nonlinear, time-dependent advection-diffusion equation ¹. The code is also applicable to the diffusion equation and related boundary value problems, such as Poisson's equation and Laplace's equation. In addition, solutions of the hyperbolic, advection equation are also possible with this formulation. The solution method is based on the Galerkin form of the finite element method (FEM). The present version of CUERVO is based directly on the code COYOTE II [1-2], which is specifically intended for the solution of heat conduction problems. In developing CUERVO those aspects of COYOTE II that are specific to thermal analysis (*e.g.*, radiation, chemistry, kinematics, *etc.*) have been removed, resulting in a more generic type of field equation solver.

The class of problems treated by CUERVO are basically those described by the following advection-diffusion equation:

$$\beta \frac{D\phi}{Dt} = \beta \left(\frac{\partial \phi}{\partial t} + u_j \frac{\partial \phi}{\partial x_j} \right) = \frac{\partial}{\partial x_i} \left[\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right] + Q \quad (1)$$

which is defined for the bounded domain Ω . In equation (1), $\phi(x_i, t)$ is the unknown scalar field variable, t is the time, x_i are the spatial coordinates, β is a scalar coefficient, κ_{ij} is a diffusion (conductivity) tensor and Q is the volumetric source. The equation is written for a fixed, Cartesian reference frame with the i, j indices running between 1 and 3, and the usual summation conventions in effect. The material derivative D/Dt is written in terms of the convective velocity vector $u_j(x_i, t)$, which is assumed to be given; the CUERVO code does not include any capability for computing material velocities. Note that (1) is written in nonconservation form, which implies that the specified velocity field must be compatible with the velocity constraint $\partial u_i / \partial x_i = 0$. The parameters β and κ_{ij} , as well as the volume source Q , may be functions of the spatial coordinates, time and the dependent variable, ϕ .

Boundary conditions for the above equation are of the usual type and include Dirichlet, Neumann, and/or mixed conditions as given by

$$\phi = f(s_i, t) \quad \text{on } \Gamma_D \quad (2)$$

$$\left(\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right) n_i = g(s_i, t) \quad \text{on } \Gamma_N \quad (3)$$

$$\left(\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right) n_i = \lambda(s_i, t)\phi + \gamma(s_i, t) \quad \text{on } \Gamma_M \quad (4)$$

¹The term advection is used here interchangeably with the word convection. Advection seems to be more popular in describing the equation given in (1), while convection is normally used to denote a type of transport.

where (4) is recognized as a generalization of (3). In equations (2)-(4), n_i is the outward unit normal to the boundary, s_i are coordinates on the boundary and f, g, λ and γ are functions that may depend on the spatial coordinates, time and/or the dependent variable. For the time dependent case, an initial value of ϕ throughout the region is required.

The above initial, boundary value problem is defined on the domain Ω with boundary $\Gamma = \Gamma_D \cup \Gamma_N \cup \Gamma_M$. Domains with multiple regions may also be considered, in which case the following continuity conditions are assumed:

$$\phi|_{\Gamma_m^+} = \phi|_{\Gamma_m^-} \quad (5)$$

$$\left(\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right) n_i \Big|_{\Gamma_m^+} = \left(\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right) n_i \Big|_{\Gamma_m^-} \quad (6)$$

where the superscript $+, -$ notation indicates properties or variables evaluated on either side of the region interface Γ_m . The coefficients and source term in (1) may be discontinuous across region boundaries.

The problem specified in (1)-(6) is quite general and a number of simpler field problems may be easily derived from the basic equation. When there is no convective transport, $u_i = 0$ and (1) reduces to the general diffusion equation

$$\beta \frac{\partial \phi}{\partial t} = \frac{\partial}{\partial x_i} \left[\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right] + Q \quad (7)$$

In many cases, the diffusion tensor is a scalar ($\kappa_{ij} = \kappa \delta_{ij}$) which produces a further simplified diffusion equation. If $\beta = 0$ or the problem has no dependence on time, then (7) becomes

$$\frac{\partial}{\partial x_i} \left[\kappa_{ij} \frac{\partial \phi}{\partial x_j} \right] = Q \quad (8)$$

which for a constant, isotropic κ reduces to the Poisson equation

$$\frac{\partial}{\partial x_i} \left[\frac{\partial \phi}{\partial x_i} \right] = \frac{\partial^2 \phi}{\partial x_i^2} = \nabla^2 \phi = Q. \quad (9)$$

When the volume source is absent, then (9) reduces to the simple Laplacian for the scalar field variable ϕ

$$\frac{\partial}{\partial x_i} \left[\frac{\partial \phi}{\partial x_i} \right] = \frac{\partial^2 \phi}{\partial x_i^2} = \nabla^2 \phi = 0. \quad (10)$$

Boundary conditions for these reduced equations are still as shown in equations (2)-(4).

Another problem related to (1) is the case where diffusion is absent, $\kappa_{ij} = 0$, and the hyperbolic advection equation

$$\beta \frac{D\phi}{Dt} = \beta \left(\frac{\partial \phi}{\partial t} + u_j \frac{\partial \phi}{\partial x_j} \right) = 0 \quad (11)$$

is an appropriate description of the transport process. Due to the reduction in order of this equation compared to (1), only the Dirichlet boundary condition in (2) can be employed with this type of problem. Furthermore, this boundary condition is restricted to the "inflow" ($u_i n_i < 0$) portions of the boundary.

Physical applications for all of these equations are numerous throughout science and engineering. A partial list of such application areas is given below:

- a) Convective/diffusive transport, energy, mass or a passive scalar
- b) Convective transport, energy, mass or a passive scalar
- c) Diffusive transport, energy, mass or a passive scalar
- d) Saturated or partially saturated flow in porous media
- e) Potential fluid flow
- f) Gravitational fields
- g) Electrostatic fields
- h) Electric conduction
- i) Lubrication flows

Many of these types of problems may be conveniently solved via CUERVO in either their steady-state or transient form.

The present document is intended to provide a detailed description of the input data necessary to use CUERVO in the solution of the initial, boundary value problems listed above. The theoretical basis for the code and some of the numerical procedures used in the program are only outlined here because they are described in detail in the COYOTE II documents [1,2]. The present manual follows closely the style and content of references [1,2], which reflects the common heritage of the two codes.

In the next section, a brief description of the finite element formulation is provided. Program capabilities and organization of the code are then described, followed by the main section that details the input to the code. The penultimate section provides a brief discussion of programming and installation details. Several short example problems are included in the last section to illustrate typical input data files.

2 Computational Method

In this section the basic computational method used in CUERVO is outlined and some of the specific numerical techniques are described. A more complete discussion of these topics is given in [1,2].

2.1 Finite Element Formulation

The spatial discretization of the boundary value problem outlined in Section 1 utilizes the method of weighted residuals and the Galerkin form of the finite element method. Let the region of interest, Ω , be divided into a number of simply shaped regions called finite elements. Within each element, a set of nodal points are established at which the dependent variable (*i.e.*, ϕ) is evaluated. The spatial variation of the dependent variable within each element is approximated by an expansion of the form

$$\phi(x_i, t) = \sum_{n=1}^{N_e} \Theta_e^n(x_i) \varphi_e^n(t) \quad (12)$$

where Θ_e represents the N_e interpolation functions and φ_e are the N_e nodal point values of the dependent variable in the element. The ability to define simple, local approximations to the dependent variable is a primary feature of the finite element method. Writing (12) in a more convenient matrix notation

$$\phi(x_i, t) = \Theta^T(x_i) \varphi(t) \quad (13)$$

where Θ is a vector of basis or interpolation functions, φ is a vector of nodal point unknowns and superscript T denotes a vector transpose. Substitution of equation (13) into the partial differential equation (1) yields a set of residual equations, due to the approximate nature of equation (13). In functional form then

$$\mathcal{F}(\Theta, \varphi) = \mathcal{R}. \quad (14)$$

The Galerkin method guarantees the orthogonality of the residual vector to the space spanned by the interpolation functions. This orthogonality is expressed by the inner product,

$$\langle \Theta, \mathcal{F} \rangle = \langle \Theta, \mathcal{R} \rangle = 0 \quad (15)$$

where $\langle a, b \rangle$ denotes the inner product defined by

$$\langle a, b \rangle = \int_{\Omega} a \cdot b \, d\Omega \quad (16)$$

Carrying out the above operations explicitly for the advection-diffusion equation (1) yields the following,

$$\int_{\Omega} \beta \Theta \Theta^T \frac{\partial \varphi}{\partial t} \, d\Omega + \int_{\Omega} \beta \Theta u_j \frac{\partial \Theta^T}{\partial x_j} \varphi \, d\Omega - \int_{\Omega} \Theta \frac{\partial}{\partial x_i} \left(\kappa_{ij} \frac{\partial \Theta^T}{\partial x_j} \varphi \right) \, d\Omega - \int_{\Omega} \Theta Q \, d\Omega = 0. \quad (17)$$

As is standard practice [3,4], the second-order diffusion term in (17) may be rewritten using the divergence theorem to produce a first-order term plus a boundary integral.

$$\begin{aligned} \int_{\Omega} \beta \Theta \Theta^T \frac{\partial \varphi}{\partial t} d\Omega + \int_{\Omega} \beta \Theta u_j \frac{\partial \Theta^T}{\partial x_j} \varphi d\Omega + \int_{\Omega} \frac{\partial \Theta}{\partial x_i} \left(\kappa_{ij} \frac{\partial \Theta^T}{\partial x_j} \varphi \right) d\Omega = \\ \int_{\Omega} \Theta Q d\Omega + \int_{\Gamma} \Theta \left(\kappa_{ij} \frac{\partial \Theta^T}{\partial x_j} \varphi \right) n_i d\Gamma. \end{aligned} \quad (18)$$

Recognizing the boundary integral in (18) as part of the boundary condition specification in (3) or (4) allows this term to be reconfigured as

$$\int_{\Omega} \beta \Theta \Theta^T \frac{\partial \varphi}{\partial t} d\Omega + \int_{\Omega} \beta \Theta u_j \frac{\partial \Theta^T}{\partial x_j} \varphi d\Omega + \int_{\Omega} \frac{\partial \Theta}{\partial x_i} \left(\kappa_{ij} \frac{\partial \Theta^T}{\partial x_j} \varphi \right) d\Omega = \int_{\Omega} \Theta Q d\Omega + \int_{\Gamma} \Theta g d\Gamma \quad (19)$$

where the simpler Neumann boundary condition in (3) has been used in the right-hand-side boundary integral. The mixed boundary condition from equation (4) may also be used in the boundary integral when appropriate. Noting that the nodal point unknowns are independent of the spatial integration and may be moved outside the integrals allows (19) to be written in the following matrix form

$$\mathbf{M}(\varphi) \dot{\varphi} + \mathbf{C}(\mathbf{u}, \varphi) \varphi + \mathbf{K}(\varphi) \varphi = \mathbf{F}_Q(\varphi) + \mathbf{F}(\varphi) \quad (20)$$

where the superposed dot indicates a time derivative and the possible dependencies on the dependent variable have been indicated. The individual matrices and vectors are defined by

$$\begin{aligned} \mathbf{M}(\varphi) &= \int_{\Omega} \beta \Theta \Theta^T d\Omega \\ \mathbf{C}(\mathbf{u}, \varphi) &= \int_{\Omega} \beta \Theta u_j \frac{\partial \Theta^T}{\partial x_j} d\Omega \\ \mathbf{K}(\varphi) &= \int_{\Omega} \frac{\partial \Theta}{\partial x_i} \left(\kappa_{ij} \frac{\partial \Theta^T}{\partial x_j} \right) d\Omega \\ \mathbf{F}_Q(\varphi) &= \int_{\Omega} \Theta Q d\Omega \\ \mathbf{F}(\varphi) &= \int_{\Gamma} \Theta g d\Gamma \end{aligned} \quad (21)$$

The matrix equation given by (20) represents the discrete analogue of the conservation equation for an individual finite element. Note that the $\mathbf{M}$ matrix represents the storage or capacitance term, the unsymmetric $\mathbf{C}$ matrix represents advective (convective) transport and the $\mathbf{K}$ matrix represents the diffusion operator. The $\mathbf{F}_Q$ and $\mathbf{F}$ vectors provide the forcing functions for the equation in terms of volume and surface fluxes, respectively.

The above derivation has been concerned with a single finite element and the limited portion of the continuum that it represents. The discrete representation of the entire

domain of interest is obtained through an assemblage of elements such that interelement continuity of the dependent variable is enforced. This continuity requirement is met through the appropriate summation of equations for nodes common to adjacent elements (the so-called "direct stiffness" approach) [3,4]. The result of such an assembly process is a system of matrix equations of the form given by (20).

2.2 Computational Details

The details of the construction and solution of the discretized equation shown in (20) are omitted here as they are extensively discussed in other references [1-4]. Only a brief outline of the essential numerical techniques employed in CUERVO will be given.

An essential part of a finite element code is the finite element library. The element library in CUERVO is illustrated in Figure 1 and includes isoparametric quadrilaterals and triangles for two-dimensional applications and isoparametric tetrahedrons, wedges and hexahedrons for three-dimensional simulations. Within each of these elements, the dependent variable, ϕ , is approximated using either linear or quadratic basis functions. For each of these element types the evaluation of the element matrices in (21) is carried out using numerical quadrature such as product Gauss rules and specialized area/volume integration rules on simplexes. Isoparametric mapping [3,4] is used to define elements with curved edges or surfaces. The assembly of element equations into a global matrix system is carried out element-by-element using the direct stiffness method.

The solution methods for the equation system given in (20) vary according to the time-dependence of the problem. For the analysis of time-independent problems, CUERVO provides a choice of several variants of a Picard iteration scheme. Transient problems are analyzed using either of two implicit integration schemes – a first-order Euler method or a second-order trapezoid rule. Both integrators may be used in a predictor/corrector mode with a fixed timestep or a dynamic timestep algorithm. A first-order, explicit integration method is also available for use with similar types of timestep control. Details of these methods and their implementation can be found in [1].

For steady state problems or implicit time integration methods, the solution procedures in CUERVO result in a large, sparse matrix problem that must be solved at each iteration or time step. The matrix solution methods used in CUERVO are based on a preconditioned conjugate gradient (PCG) algorithm. The matrix solution technique is embedded in an external library package that supports both symmetric and unsymmetric matrices. A variety of preconditioners are available for each of the basic iterative solution techniques. To accommodate the storage requirements of the global matrix, a standard sparse matrix format is used that only records the nonzero entries of the matrix. Complete details on the iterative solvers are available in [5].

Triangle

Quadrilateral

Hexahedron

Prism

Tetrahedron

Figure 1: Finite elements in the CUERVO library; the linear and quadratic version of each element is available in the code.

3 Program Overview

The present description is intended only as an overview of the major capabilities and features of the program. A brief section is also included to show the organizational components of the program.

3.1 Program Features

CUERVO is intended for the analysis of the advection-diffusion equation or any of its simplified forms. Geometrically, this version of the program is appropriate for treating two-dimensional, plane or axially symmetric problems and fully three-dimensional analyses. The theoretical formulation assumes isotropic or orthotropic behavior for all material regions. Further, the regions of interest may be heterogeneous with properties that vary in a quite general manner with spatial location and time. The volumetric source term may also be defined and allowed to vary in a general manner. The problems may be either independent of time or fully time-dependent in nature. Conjugate problems, in which different equations are solved in various regions of the domain, are also within the capabilities of CUERVO.

Boundary condition specification is very general and allows all of the standard types of boundary conditions to be imposed on a problem. All boundary conditions may be general functions of spatial location, time and/or the dependent variable, ϕ . Data may be input in tabular form or programmed with a user-supplied subroutine.

CUERVO relies on specialized external codes for mesh generation and graphical processing of solution data. Mesh data and output for graphical processing are read and written by CUERVO in a standard, well-documented format that allows easy coupling with other programs. The data analysis portions of the code allow local gradients and fluxes to be computed. A facility also exists for reporting flux results that are integrated over element surfaces to predict the total transport through a boundary of the domain.

3.2 Program Organization

CUERVO has been organized in a modular form in an effort to make the operation of the code understandable and to ease the task of code modification. As shown in the schematic of Figure 2, the code consists of a main program plus a number of major, task-oriented subroutines. Each of the major subroutines may access one or more subordinate routines plus the utilities attached to the main program.

Input to the program is organized into six distinct blocks that correspond to major tasks within the code. Within each block, keyword oriented data is specified in a free-field format; data within the block may be ordered arbitrarily. A list of the six major

Figure 2: Organization of the CUERVO code.

data blocks, their associated keywords, and a brief description of their function is given below.

- Title Data Block (**TITLe**) - specifies problem title and comments
- Material Data Block (**MATerial**) - specifies material property data
- Problem Definition Data Block (**PROBLEM DEFinition**) - specifies problem type, element block properties, boundary conditions and simulation options
- Solution Data Block (**SOLution**) - specifies solution method and options
- Post-processing Data Block (**POST**) - specifies post-processing output and options
- Function Data Block (**TIME FUNction, VARiable FUNction, USER CON-
stants**) - specifies function data and user-defined constants

The large amount of data needed in a typical finite element analysis is handled in CUERVO by a combination of disk files and in-core memory. Data that is repeatedly required by several of the major subroutines (*e.g.*, nodal point coordinates, nodal point connectivity and current solution vectors) are stored in main memory. The in-core storage scheme makes use of a dynamic memory algorithm that allocates needed array space during program execution. Less frequently needed data (*e.g.*, input data file) and extremely large blocks of data (*e.g.*, previous solution vectors) may be stored on various disk files. The utilization of the files shown in Figure 2 is discussed in Section 5.3.

4 Input Guide

The structure of an input data file for the CUERVO program directly reflects the steps required to formulate, solve and analyze a finite element model for a transport problem. Input to the code consists of four types of cards ²: a) Commentary cards, b) Data Block cards, c) Data cards and d) Termination cards. Input on any given card is in a free-field format style with the first entry usually being a keyword. Comment lines may appear at any point within the input file; this type of data is echoed to the output file but is otherwise ignored by the code. Data block cards and termination cards occur in pairs and bracket a series of data cards. The order in which data block cards appear in the input file is arbitrary. Data block cards may appear as many times as necessary within an input file but must always be closed by a termination card. Data cards within a data block may be ordered arbitrarily except for function data, which must appear in a logical, ascending order.

The input data file that is provided to CUERVO is previewed by the code in an effort to uncover obvious errors. All keywords are reviewed for syntax, proper termination of each data block is checked and the presence of all mandatory data blocks is verified. This data scan also serves to set certain critical sizing parameters needed for proper program execution.

The data block cards recognized by CUERVO are listed below in the order in which they are discussed in the subsequent sections. This is also the order in which they would normally appear in a typical input file. A summary showing the form and content of the indicated data block cards and associated data cards is provided in Appendix A.

- Title Data Block
- Material Data Block
- Problem Definition Data Block
- Solution Data Block
- Post-processing Data Block
- Time and Variable Function Data Blocks
- User Constants Data Block
- Termination Data

²Notice that throughout this manual input data is referred to in terms of "cards"; this is a convenient term that in actuality refers to a record on an input device.

4.1 Input Syntax

The CUERVO input cards generally follow the standard format shown below:

KEYWORD=parameter1, parameter2, ...

In some cases a keyword does not require any additional data, in which case the parameter list is omitted.

In describing the input data, the following conventions have been adopted for the following sections:

- (a) Bold face words indicate keywords, *e.g.*, **MATerial**. Most keywords can be abbreviated to the first 3-5 characters of each word. Upper case characters indicate the shortest abbreviation that is recognized by the code.
- (b) Keyword data that is required for the proper specification of a problem and is mandatory for all input files is flagged with a □ symbol at the right hand margin.
- (c) Upper case words indicate a required alphanumeric input value, *e.g.*, **USER**. Most alphanumeric input can be abbreviated to the first 3-5 characters of each word. Upper case characters indicate the shortest abbreviation that is recognized by the code.
- (d) Lower case words and symbols imply that an alphanumeric or numerical value for the specified variable is expected, *e.g.*, **itermax**.
- (e) All input values are specified in a free field format with successive variables separated by commas, equal signs or blanks. A convenient convention that is used in this manual uses blanks within keywords, equal signs to separate keywords from variable lists and commas for separating variables.
- (f) The total number of input values specified on any single input record (including all continuation cards) is presently limited to 50.
- (g) All alphanumeric input data is limited to twenty characters under the free-field format.
- (h) The \$ character may be used to end an input line and use the remaining space on the line for comments. A \$ character at the beginning of a line indicates a full comment line that is ignored by the code.
- (i) The * character may be used to continue an input line onto a subsequent data card. The continuation character should follow the last delimiter (comma or equal sign) on the card to be continued.

Figure 3: Coordinate system definition.

- (j) *Italics* indicate optional parameters which may be omitted by using successive delimiters (commas or equal signs) in the input line. If the omitted parameter is not followed by any required parameters, no additional delimiters need be specified.
- (k) () indicates the data type for a particular input variable, i.e., alphanumeric or character (C), real (R) or integer (I).
- (l) < > indicates the default value for an optional parameter.
- (m) The contents of each input line are indicated by underlining.
- (n) All quantities associated with a coordinate direction are expressed in terms of the cartesian x, y or x, y, z coordinate system. The corresponding quantities for axisymmetric problems are obtained by the association of the radial coordinate, r , with x and the axial coordinate, z , with y . The global coordinate system used by the code is right-handed as indicated in the sketch shown in Figure 3.
- (o) Explanatory notes for various input options and data are numbered consecutively within a section and are located at the end of each Data Block.

4.2 Title Data Block

The title data block allows the specification of an appropriate problem title and any additional commentary that is useful in documenting the simulation. This data block is optional. The title data block has the following form:

```
TITle  
THIS IS AN EXAMPLE OF A PROBLEM TITLE  
.  
.  
THESE ARE EXAMPLES OF COMMENT LINES  
UP TO 10 LINES OF PROBLEM DESCRIPTION MAY  
BE INCLUDED ON THESE CARDS  
.  
.  
.  
ENDtitle
```

The first line following the **TITle** keyword is taken as the problem title and is written to various output files to assist in the identification of a particular code execution. Following the title card a number of comment cards may be included to provide a description of the particular problem, modeling assumptions, material models, *etc.*. Up to 10 comment lines may be specified. The comment lines are reproduced at the beginning of the printed output file as problem documentation. The title data block is terminated with an **END** card.

4.3 Material Data Block

The input of material properties to CUERVO is accomplished through the material data block and its associated set of data cards. A separate material data block is required for each material used in a CUERVO simulation. Input of material data for each block is terminated by an **END** card. The material data block is of the following form:

MATerial, material name, *material model*

□

Material Property Data

ENDmaterial

□

where

material name (C) : is a required material name (see Note 1).

material model (C) <ISOTropic> : is the type of material model used to describe the κ tensor. When this parameter is omitted or set to ISOTropic, the material is modeled as isotropic with $\kappa_{ij} = \kappa\delta_{ij}$; a parameter value of ORTHotropic indicates a material with a full tensor κ_{ij} (see Note 2).

Material Property Data

Physical property data for each material is specified by a series of data cards within the data block. Each property data card begins with a property keyword and is followed by the numerical value of the material property or an option that directs the code to evaluate the property through a function or a user subroutine. CUERVO does not contain any dimensional constants and, therefore, the units for the material properties are free to be chosen by the user.

The individual material data cards recognized by CUERVO are listed below along with the various options available for each property.

BETA= β

BETA=USER

BETA=VFUNction, idvar

β (R) <0.0> : specifies the value of the coefficient beta.

USER (C) : specifies that beta will be evaluated by a user-supplied subroutine, USRBET.

VFUNCTION (C) : specifies that beta will be evaluated via a variable function (see Note 3).

idvar (I) : sets the variable function identification number (see Note 3).

KAPpa= κ_{11} , κ_{22} , κ_{33} □

KAPpa=USER □

KAPpa=VFUNCTION, idvar1, idvar2, idvar3 □

κ_{11} , κ_{22} , κ_{33} (R) : specify the values of the principle components of the tensor κ . If the material is isotropic only the first value, κ_{11} , is required; orthotropic materials in two-dimensional problems only require the first two components of the tensor κ_{11} and κ_{22} , to be specified (see Note 2).

USER (C) : specifies that the principle components of the κ tensor will be evaluated by a user-supplied subroutine, USRKAP (see Note 2).

VFUNCTION (C) : specifies that the principle components of the κ tensor will be evaluated via one or more variable functions (see Notes 2 and 3).

idvar1, idvar2, idvar3 (I) : set the variable function identification numbers for each of the principle components of κ . If the material is isotropic only the first function idvar is required; orthotropic materials in two-dimensional problems only require the first two function idvars to be specified. Components with the same functional behavior may share a single function specification (see Note 3).

TENsor ROTation= $x\hat{x}$, $y\hat{x}$, $z\hat{x}$, $x\hat{y}$, $y\hat{y}$, $z\hat{y}$

TENsor ROTation=USER

$x\hat{x}$, $y\hat{x}$, $z\hat{x}$, $x\hat{y}$, $y\hat{y}$, $z\hat{y}$ (I or R) <1,0,0,0,1,0> : represent the orientation of the principle material axes with respect to the global coordinate system. The six components specify the projection of the $\hat{x}$ and $\hat{y}$ principle material axes for the κ tensor onto the global coordinate axes (see Note 2).

USER (C) : specifies that the local orientation of κ will be evaluated by a user subroutine, USRRROT (see Note 2).

VOLume SOURce= Q

VOLume SOURce=USER

VOLume SOURce=TFUNction, idtim

VOLume SOURce=VFUNction, idvar

Q (R) <0.0> : specifies the value of the volumetric source.

USER (C) : specifies that the volumetric source will be evaluated by a user-supplied subroutine, USRVS.

TFUNction (C) : specifies that the volumetric source will be evaluated via a time function (see Note 3).

idtim (I) : sets the time function identification number (see Note 3).

VFUNction (C) : specifies that the volumetric source will be evaluated via a variable function (see Note 3).

idvar (I) : sets the variable function identification number (see Note 3).

INITial VALue= ϕ_{init}

INITial VALue=USER

ϕ_{init} (R) <0.0> : specifies an initial value of ϕ for the material.

USER (C) : specifies that the initial condition for the material will be evaluated by a user-supplied subroutine, USRIC (see Note 4).

Notes:

- 1) The material name assigned to each material data block must be unique. This name is also used in the problem definition section (see Section 4.4) to allow material properties to be associated with individual blocks of elements. One and only one set of material property data cards should be supplied for each material name. No abbreviations of the name are allowed; the name must be less than 20 characters.
- 2) The material models allowed in CUERVO include materials with either an isotropic or orthotropic tensor. For isotropic materials, $\kappa_{ij} = \kappa$, and the κ tensor is specified completely by setting the value of the scalar κ . Dependence of the isotropic coefficient κ on the dependent variable can be implemented via a variable function or a user-supplied subroutine. For orthotropic materials, $\kappa_{ij} = \delta_{ij}\kappa_{ij} = \kappa_{ii}$, and κ is determined by specifying the components of the tensor with respect to the principle material axes. Dependence of each of the principle components can be specified independently through the variable functions or a user subroutine. If the principle material axes are not aligned with the global coordinate axes then the

Figure 4: Notation for orthotropic conductivity tensor, κ .

TENSOR ROTATION data card must be used to supply the appropriate coordinate transformation (rotation). Referring to Figure 4, the $\hat{x}$, $\hat{y}$ and $\hat{z}$ axes indicate the principle material axes while x , y , and z denote the global reference frame. The parameters $x\hat{x}$, $y\hat{x}$, $z\hat{x}$ on the **TENSOR ROTATION** data card refer respectively to the x , y , z projections of a vector in the positive $\hat{x}$ direction. Similarly, the parameters $x\hat{y}$, $y\hat{y}$, $z\hat{y}$ refer to the x , y , z projections of a vector in the positive $\hat{y}$ direction. Note that these vectors ($\hat{x}$, $\hat{y}$) need not be unit vectors. For two-dimensional applications only the $x\hat{x}$, $y\hat{x}$ projections need to be specified for a vector in the positive $\hat{x}$ direction. To illustrate the definition of a tensor rotation, a 45° rotation about the y axis is specified by $\langle x\hat{x}, y\hat{x}, z\hat{x} \rangle = \langle 1, 0, 1 \rangle$ and $\langle x\hat{y}, y\hat{y}, z\hat{y} \rangle = \langle 0, 1, 0 \rangle$. Rotations defined by parameters on the **TENSOR ROTATION** card are applied uniformly to all elements with the given material name. If the tensor rotation is required to vary with spatial location and must be computed on an element-by-element basis, then the user subroutine, **USRROT**, must be employed (see Section 4.11.1).

- 3) Material data that has a simple functional dependence will usually be specified via a variable function (**VFUNction**) or a time function (**TFUNction**). The correspondence between each material property and the appropriate function is determined from the integer identification (**idvar** or **idtim**) number. Within each category of function, the **id** must be unique. Function definitions are described in Sections 4.7 and 4.8. For material data that has a complex functional form, a user subroutine is recommended.
- 4) If the initial condition for a material is set to **USER**, then a subroutine, **USRIC**, must be attached to **CUERVO**. All materials with a **USER** setting are evaluated

with the same subroutine; different materials are identified by their material name. The format for this subroutine is described in Section 4.12.

4.4 Problem Definition Data Block

The problem definition data block is used to define the general attributes of the problem that is to be solved. Normally, only one such data block is required. Input of problem definition parameters is terminated by an **END** card. The problem definition data block is of the following form:

```
PROBLEM DEFINITION, format           □  
.  
.  
Problem Definition Data  
.  
.  
.  
ENDproblem                           □
```

where

format (C) <EXODus> : is the name of the format used to provide mesh data to CUERVO. Currently the only supported data format is the EXODUS II format [6].

Problem Definition Data

The setting of various program and modeling options is set through a series of data cards within the problem definition data block. Each option begins with a keyword and may be followed by one or more parameters that define the option. The individual problem definition data cards recognized by CUERVO are listed below with the various parameter choices.

GEOMETRY=type □

type (C) : specifies the type of geometry used in the problem. The type parameter should be set to 2D for planar, two-dimensional models, AXISY for axisymmetric, two-dimensional models or 3D for fully three-dimensional models.

ELEMENT BLOCK= block id, material name, *integration rule* □

block id (I) : specifies the previously assigned identifier for a block of elements in the mesh. The elements within this block are expected to all be the same material and be the same type of finite element (see Note 1).

material name (C) : specifies the material name for the element block. This identifier should correspond to one of the names used on the **MATerial** data block card.

integration rule (I) : specifies the integration (quadrature) rule used to construct the finite element coefficient matrices (see Note 2).

BCtype=bcname, nodeset id, bc specification, *mfactor*

□

bcname (C) : specifies the name for the type of boundary condition applied to a node set. The permissible node set boundary condition is **DIRichlet** (see Note 3).

nodeset id (I) : specifies the previously assigned node set identifier for a group of boundary conditions (see Note 3).

bc specification : specifies the method for evaluating the boundary condition. This parameter may be replaced with any of the following:

bcvalue (R) : the constant value of the boundary condition.

USER (C) : the boundary condition will be evaluated by a user-supplied subroutine, **USRDIR**.

TFUNction (C), idtim (I) : the boundary condition will be evaluated via a time function with time function identification number, idtim (see Note 5).

mfactor (R) <1.0> : is a multiplying factor that is applied to the value of the boundary condition. This factor is not used with the user subroutine option.

BCtype=bcname, sideset id, bc specification, *mfactor*

□

bcname (C) : specifies the name for the type of boundary condition applied to a side set. The permissible side set boundary condition for this format is **NEUMann** (see Note 4).

sideset id (I) : specifies the previously assigned side set identifier for a group of boundary conditions (see Note 4).

bc specification : specifies the method for evaluating the boundary condition. This parameter may be replaced with any of the following:

bcvalue (R) : the constant value of the boundary condition.

USER (C) : the boundary condition will be evaluated by a user-supplied subroutine, **USRNEU**.

TFUNction (C), idtim (I) : the boundary condition will be evaluated via a time function with time function identification number, idtim (see Note 5).

VFUNction (C), idvar (I) : the boundary condition will be evaluated via a variable function with variable function identification number, idvar (see Note 5).

mfactor (R) <1.0> : is a multiplying factor that is applied to the value of the boundary condition. This factor is not used with the user subroutine option.

BCtype=bcname, sideset id, LAMbda = coef specification, *mfactor*,

GAMma = rhs specification, *mfactor* □

bcname (C) : specifies the name for the type of boundary condition applied to a side set. The permissible side set boundary condition for this format is MIXed (see Note 4).

sideset id (I) : specifies the previously assigned side set identifier for a group of boundary conditions (see Note 4).

coef specification : specifies the method for evaluating the coefficient λ . This parameter may be replaced with any of the following:

cvalue (R) : the constant value of the coefficient.

USER (C) : the coefficient will be evaluated by user-supplied subroutine, USRLAM.

TFUNction (C), idtim (I) : the coefficient will be evaluated via a time function with time function identification number, idtim (see Note 5).

VFUNction (C), idvar (I) : the coefficient will be evaluated via a variable function with variable function identification number, idvar (see Note 5).

mfactor (R) <1.0> : is a multiplying factor that is applied to the value of the coefficient. This factor is not used with the user subroutine option.

rhs specification : specifies the method for evaluating the right hand side function, γ . This parameter may be replaced with any of the following:

tvalue (R) : the constant value of the right hand side function.

USER (C) : the right hand side function will be evaluated by user-supplied subroutine, USRGAM.

TFUNction (C), idtim (I) : the right hand side function will be evaluated via a time function with time function identification number, idtim (see Note 5).

VFUNction (C), idvar (I) : the right hand side function will be evaluated via a variable function with variable function identification number, idvar (see Note 5).

mfactor (R) <1.0> : is a multiplying factor that is applied to the value of γ . This factor is not used with the user subroutine option.

VELocity=type, block id, velocity specification

type (C) : specifies the type of reference frame used to describe the motion of the material. This parameter must currently be set to EULERian. The material in the element block is assumed to move continuously through a fixed mesh with the velocity field specified on this card.

block id (I) : specifies the previously assigned block identifier for the block of elements for which material motion is allowed.

velocity specification : specifies the method for evaluating the velocity of the material. This parameter may be replaced with any of the following:

U_x, U_y, U_z (R) : the constant values of the x, y and z components of material velocity.

USER (C) : the material velocity will be evaluated by a user-supplied subroutine, USRVEL.

TFUNCTION (C), idtim1, idtim2, idtim3 (I) : the x, y and z velocity components for the material will be evaluated via time functions with separate time function identification numbers (see Note 5).

PRINTed OUTput=type

type (C) <SUMmary> : specifies the extent of printed output that is written to the output file for the problem setup portion of the run. The permissible values for type include SUMmary, EXTended and DEBug.

OUTput LOCations=e1, e2, e3, ... , en TO em, ...

e1, e2, e3, ... , en TO em, ... (I) : is a list of element numbers for which output data will be printed during the solution process. Element numbers may be listed individually or with the syntax en TO em, which indicates an inclusive string of elements. A maximum of forty data entries may be listed on a single card; multiple data cards are permitted.

FLUX=sideset id1,sideset id2, ... sideset idn

sideset idn (I) : specifies the previously assigned side set identifiers for which integrated normal flux computations are to be performed. If the optional sideset id's are omitted, only nodal values of the flux components will be computed (see Note 6).

POTential FUNCTION= $\mathcal{P}_0$

$\mathcal{P}_0$ (R) <0.0> : specifies the value of the potential function at the first node in the first element processed (see Note 6).

MAXimum CONnectivity=maxcon

maxcon (I) : the maximum value of the element connectivity (see Note 7).

Notes:

- 1) Element blocks form a basic grouping for data within CUERVO and also within the input and output EXODUS II files. An element block consists of a group of finite elements that must share the same geometry (shape and number of nodes), material and number of element attributes. The definition of an element block occurs during the mesh generation process and is at the discretion of the user as long as the previously defined restrictions are employed. Note that an element block is not necessarily an inclusive grouping; multiple element blocks with the same element type, material and attributes may be defined if this is convenient or required for the particular application. Associated with each element block is a unique integer identifier (block id) that is assigned by the user at the time of block creation. Data assignments for an element block are always accomplished within CUERVO through reference to the element block id. See reference [6] for further details regarding element blocks.
- 2) The quadrature rules used to integrate the finite element matrices can be varied from their default definitions through specification of the optional integration rule number. Such changes are not generally recommended but in some circumstances (*e.g.*, the occurrence of highly distorted elements or the need to reduce the cost of element construction) may be warranted. Listed in Table 1 are the quadrature rule numbers available for each element type, the number of integration points in each formula and an indication of the default rule. Quadrature rules are defined for individual element blocks; the same rule should be defined for all elements of the same type. If multiple rules are specified for one element type, the last rule input will be used.
- 3) Boundary conditions for individual nodal points in a mesh are assigned in CUERVO by association of the name (type) of the boundary condition with each node set identifier (nodeset id). The nodeset id is a unique integer that is assigned by the user to one or more nodes during the mesh generation process. Note that spatial variations in the nodal boundary condition can be controlled through a distribution factor that is also specified during mesh generation. The nodeset ids and distribution factors are passed to CUERVO through the EXODUS II file. See reference [6] for further details.
- 4) Boundary conditions for element surfaces or edges are assigned in CUERVO by association of the name (type) of the boundary condition with each side set identifier

Element Type	Rule Number	Number of Integration Points	Default Rule for Each Element	Rule for Flux Evaluation
Triangle	1	1	Linear	*
	2	3		
	3	4	Quadratic	
	4	7		
Quad	1	1	Linear	*
	2	4 (2 × 2)		
	3	9 (3 × 3)	Quadratic	
Tetrahedron	1	1	Linear	*
	2	4		
	3	5	Quadratic	
Prism	1	1	Linear	*
	2	6		
	3	12	Quadratic	
	4	21		
Hexahedron	1	1	Linear	*
	2	6		
	3	8 (2 × 2 × 2)		
	4	14		
	5	27 (3 × 3 × 3)		

Table 1: Quadrature rules for elements in CUERVO

(sideset id). The sideset id is a unique integer that is assigned by the user to one or more element surfaces (edges) during the mesh generation process. Note that spatial variations in the boundary condition over the element surface can be controlled through a set of distribution factors that are also specified during mesh generation. The sideset ids and distribution factors are passed to CUERVO through the EXODUS II file.

- 5) Simple variations in boundary conditions assigned to element nodes or surfaces (edges) may be conveniently specified by assigning a functional variation to the boundary condition. Time functions and variable functions (*i.e.*, functions of ϕ) are described in Sections 4.7 and 4.8. Each function is assigned an integer id which is then used in the boundary condition specification. Boundary condition node sets and side sets may share functions; the *mfactor* can be used to scale individual functions to obtain the correct boundary values. Spatial variations in a boundary condition are controlled through the use of distribution factors (see Notes 3 and 4) or a user subroutine.
- 6) The specification of the basic **FLUX** option causes the x, y and z components of the diffusive flux vector to be computed for each time step or iteration. The fluxes are evaluated at a fixed number of (integration) points within each element. These discontinuous flux values are subsequently extrapolated to the corner nodes of the element and averaged between adjoining elements to produce a continuous nodal point flux field. For higher-order elements, midedge (midside) values of the fluxes are generated by interpolation. The flux components should be written to the post-processing file as no printed output of these quantities is available. Note that the integration points used to define flux locations may not be the same (quadrature) points used to evaluate element matrices. The data in Table I shows the integration points used for the flux evaluation.

When *sideset id's* are specified, some additional processing of the flux data occurs. For each element face (side) in a given sideset, the flux normal to the surface is computed and integrated over the area (length) of the element face (side). These integrated flux contributions are summed to produce the total transfer for the surface defined by a sideset; these values are reported in the printed output for each time step or iteration.

The **POTential FUNction** option is only available for two-dimensional geometries and causes a scalar potential to be constructed that is useful in the visualization of transport through a domain. The definition of the potential function and its numerical construction is outlined in [2,7]. Because the potential function requires nodal values of the flux, this option automatically invokes the basic **FLUX** computation, if it has not been previously specified. Also, as the potential function is constructed incrementally, on an element-by-element basis, the sequencing of elements must be continuous or the process will fail. The potential function should be written to the post-processing file as no printed output of this quantity is available.

- 7) The sparse matrix format used by CUERVO is a very efficient method for storing large sparse systems. However, to properly allocate memory for this scheme an estimate of the maximum element connectivity is required. This connectivity is easily computed for structured mesh topologies but is rather expensive and difficult to find for unstructured discretizations. By definition, the maximum connectivity is the largest number of nodes any given node is connected to through adjoining elements. For a structured, two-dimensional mesh of four node elements, the maximum connectivity is 9 while for an eight node element the connectivity is 21. The maximum connectivity for a structured, three-dimensional mesh of eight node elements is 27. The maximum connectivity parameter should generally be set for any mesh that is unstructured; a reasonable connectivity estimate is 130% to 150% of the connectivity for a structured mesh with the same basic element type.

4.5 Solution Data Block

The control of the solution process for the steady or time-dependent transport problem is accomplished through the solution data block and its associated data cards. A separate solution block is required for each change in solution algorithm or change in any of the associated solution parameters. Input of data for each solution block is terminated by an **END** card. The solution data block is of the following form:

<u>SOLution</u> , solution block number, time dependence	□
.	
.	
Solution Data	
.	
.	
.	
<u>ENDsolution</u>	□

where

solution block number (I) : is the number of the solution block. Solution blocks are executed in numerical order and should be numbered consecutively starting with 1 (see Note 1).

time dependence (C) : indicates if the solution block should be treated as a transient or steady state problem. This parameter should be set to TIME INDependent for a steady solution and set to TIME DEPEndent for a transient solution sequence.

Solution Data

The parameters that control the steady or transient solution process within each solution block are specified by a series of data cards. Each solution option data card begins with a keyword and is followed by one or more parameters. Data supplied within a given solution block applies only within that block; parameter settings are not maintained across solution blocks and are reset (or defaulted) by subsequent solution blocks.

The individual solution data cards recognized by CUERVO are listed below along with the various options available for each parameter.

REStart TIME= $t_{restart}$

$t_{restart}$ (R) <0.0> : specifies the timeplane (time) on the restart file that is to be used for initializing the solution field (see Note 2).

REStart STEP=step number

step number (I) <last solution on the file> : specifies the timeplane (number) on the restart file that is to be used for initializing the solution field (see Note 2). The default setting selects the last solution available on the restart file.

ITERative METHod=scheme

scheme (C) <PICard> : specifies the type of iterative method employed for non-linear, time-independent problems. The permissible value for this parameter is PICard.

INTEgration METHod=scheme, predictor option

scheme (C) <TRAPEzoid> : specifies the type of time integration method employed for time-dependent problems. The permissible values for this parameter are EULER, TRAPEzoid and EXPLICIT (see Note 3).

predictor option (C) <PREDict> : specifies if the explicit predictor is to be used in conjunction with an implicit integration method. This parameter may be set to PREDict or left blank, if the predictor step is to be employed; a parameter setting of NOPredict eliminates the predictor step (see Note 3).

CAPacitance MATRix=type

type (C) <CONSistent> : specifies the treatment of the capacitance matrix in time-dependent problems. The permissible values for this parameter are CONSistent and LUMPed (see Note 3).

TIME STEP OPTion=type

type (C) <AUTOstep> : specifies the type of time step control used in time-dependent problems. The permissible values of this parameter are FIXed or AUTOstep (see Note 3).

INTEgration TOLerance=integ tol

integ tol (R) <0.001> : specifies the integration tolerance for the time step control under the AUTOstep option. This parameter is only applicable to the implicit integration methods (see Note 3).

NORM VALue= ϕ_{norm}

ϕ_{norm} (R) < maximum ϕ in solution > : specifies an appropriate (maximum) value of the dependent variable to be used in defining an RMS norm for the change in the variable over a time step or iteration. The default scale for the norm is the maximum value of the dependent variable found in the solution vector at each step or iteration. If the maximum value is not constant for the simulation, this parameter should be specified to ensure a consistent definition of the norm. The definition of the RMS norm is given in [1].

TIME STEP FACTor= $\beta_{explicit}$

$\beta_{explicit}$ (R) <0.9> : specifies the scale factor to be applied to the maximum stable explicit time step. This parameter is only applicable to the explicit integration method (see Note 3).

TIME STEP= Δt

□

Δt (R) : specifies the time step for a time-dependent simulation. For the FIXed time step option this time step is used throughout the solution block. When the AUTOSTep option is selected this is the initial time step for the integration process.

INITial TIME= t_{init}

t_{init} (R) <0.0> : specifies the initial time for a time-dependent simulation. Note that this should be the initial time for the current solution block.

FINAl TIME= t_{final}

□

t_{final} (R) : specifies the final time for a time-dependent simulation (see Note 4).

NUMber TIME STEPs= $nstep$

□

$nstep$ (I) <1000> : specifies the total number of time steps allowed in a time-dependent simulation (see Note 4).

ABSolute LIMit= ϕ_{lim}

ϕ_{lim} (R) < 1.0×10^{20} > : specifies the allowable upper limit on the solution field. If ϕ at any node exceeds ϕ_{lim} then the simulation is terminated (see Note 4).

MINimum TIME STEP= Δt_{min}

$\Delta t_{min} (R) < \Delta t_{init} \times 0.0001 >$: specifies the minimum time step allowed during a time-dependent simulation.

MAXimum TIME STEP= Δt_{max}

$\Delta t_{max} (R) < \Delta t_{init} \times 10000.0 > (R)$: specifies the maximum time step allowed during a time-dependent simulation.

MAXimum VARIABLE STEP= $\Delta \phi_{max}$

$\Delta \phi_{max} (R)$: specifies the maximum change in the dependent variable that is allowed during any time step in a time-dependent simulation. This parameter is only valid when the AUTOstep time step option has been selected (see Note 3).

CONVergence TOLerance=tol

tol (R) $< 1.0 \times 10^{-5} >$: specifies the convergence tolerance for a nonlinear, steady state problem or the steady state tolerance for a time dependent simulation (see Note 4).

RELaxation FACTor= α

$\alpha (R) < 0.0 >$: specifies the relaxation factor to be used in the iterative solution of a nonlinear, steady state problem.

MAXimum ITERations=itermax

itermax (I) $< 25 >$: specifies the maximum number of iterations allowed in the iterative solution of a nonlinear, steady state problem (see Note 4).

PRINTed OUTput=option, frequency

option (C) $< \text{SUMmary} >$: specifies the type of printed output provided during the solution process. The allowable options are NONE, SUMmary, EXTended and DEBug.

frequency (I) $< 1 >$: specifies how often data is printed during the solution process; data is printed every n iterations or time steps where $n = \text{frequency}$.

MATRix SOLver=method, krylov subspace

method (C) < CG > : specifies the type of method used to solve the linear algebra problem at each iteration or time step. The parameter setting CG specifies that a conjugate gradient method will be used for the matrix problem. Other permissible parameter settings include CGS, for the conjugate gradient squared method, GMRES for the generalized minimum residual method and QMR for the quasi-minimum residual method. A final parameter setting of DIRECT allows access to a direct (non-iterative) matrix solution method (see Note 5).

krylov subspace (I) <10> : indicates the dimension of the Krylov subspace used with the GMRES iterative solution method.

PREConditioner TYPE=option, polynomial order

option (C) <JACobi> : specifies the type of preconditioner to be used in conjunction with the selected iterative matrix solver. The permissible values for this parameter are NONE, JACobi, POLYnomial, ILU and IC (see Note 5).

polynomial order (I) < 1 > : specifies the order of the polynomial used in the POLYnomial preconditioner option.

L2 NORM= L_2

L_2 (R) < 1.0×10^{-6} > : specifies the value of the L_2 norm for terminating the preconditioned conjugate gradient solution of the matrix problem during any given iteration or time step (see Note 5).

RESidual NORM=resid

resid (R) < 0.1 > : specifies the value of the residual norm for terminating the preconditioned conjugate gradient solution of the matrix problem during any given iteration or time step (see Note 5).

MAXimum MATRix ITERations=matrix iter

matrix iter (I) < numnod > : specifies the maximum number of iterations allowed for the solution of the matrix problem during any given iteration or time step (see Note 5).

Notes:

- 1) Each solution block should be constructed such that all the parameters required for that portion of the solution process are defined. The only data that is carried across solution blocks is the last computed temperature field, the current time and a global step number. Any parameters that are defined in one block but not defined in subsequent blocks revert to their default values. Though the input order of solution blocks is arbitrary, execution of the blocks is carried out according to the solution block number, which must form a consecutive sequence. Also, note that changes in the solution algorithms across solution blocks are generally arbitrary.
- 2) Restarts of a solution procedure from a previously computed solution require that a file in an EXODUS II format be attached to CUERVO with a specified file name (see Section 5.3). The particular solution field used to initialize the new solution process is specified by providing either the time or the step (iteration) number for the field located on the external file. Further details on the restart process and restrictions are provided in Section 4.12.
- 3) All of the time integration methods available in CUERVO may be employed with either a fixed time step or an adaptive time step control. For the implicit integrators (Euler and Trapezoid methods) the adaptive time step is based on an accuracy criteria that is specified by the **INTEGRATION TOLERANCE** parameter. Adaptive time step control for these methods also requires that the *predictor option* be activated; if the predictor is deactivated, the time step option is automatically set to a fixed time step procedure. The adaptive time step control for the explicit integrator is based on stability considerations; the **TIME STEP FACTOR** allows the maximum stable explicit time step to be scaled by the user. The capacitance matrix may be lumped or left in a consistent form with the implicit integration methods. Capacitance lumping for the explicit integrator is automatically invoked by the code. The use of the lumping option with higher-order (quadratic interpolation) elements is not recommended.
- 4) The solution processes in CUERVO may be terminated by any of several criteria. For time-independent iterative methods the solution process may be terminated after a certain number of iterations (**MAXIMUM ITERATIONS**), when the change in the solution (norm) between iterations has reached a nominal value (**CONVERGENCE TOLERANCE**) or when ϕ exceeds a specified value (**ABSOLUTE LIMIT**). The time integration methods are terminated based on the total number of time steps taken (**NUMBER TIME STEPS**), the end of the integration interval (**FINAL TIME**), the attainment of a steady state (**CONVERGENCE TOLERANCE**) or when ϕ exceeds a specified value (**ABSOLUTE LIMIT**). These stopping parameters are generally set to values that should not prematurely terminate the analysis, though the (**CONVERGENCE TOLERANCE**) parameter may need to be reset for problems that evolve slowly at early times.

- 5) The primary matrix solution methods available in CUERVO consist of iterative methods based on the Krylov or conjugate direction algorithms. To use these methods effectively also requires the selection of an acceptable preconditioning technique. The solution package called by CUERVO was developed by Schunk and Shadid [5] and contains a wide spectrum of symmetric and unsymmetric matrix methods with a variety of preconditioners. For most standard diffusion (and related) problems the selection of a conjugate gradient solver (CG) with JACobi preconditioning is adequate. Unsymmetric systems arising from the advection-diffusion equation can be solved with a generalized minimum residual method (GMRES) and the ILU preconditioner. Further details on these solvers and their overall performance on different types of problems can be found in [5]. Though a direct solution method is also available in this package, it is not recommended except for modest-sized, two-dimensional simulations. The matrix solver parameters **L2 NORM** and **RESidual NORM** do not usually require adjustment; the parameter **MAXimum MATRix ITERations** is set to the number of unknowns in the problem, which is an upper bound on the number of iterations needed for convergence. The precise definition and function of these parameters is given in [5].

4.6 Post-processing Data Block

The control of the output of data to the post-processing file is accomplished through the post-processing data block and its associated data cards. This data block is optional. Input of post-processing data is terminated by an **END** card. The post-processing data block is of the following form:

POST

.

.

Post-processing Data

.

.

.

ENDpost

Post-processing Data

The output of various solution variables to the post-processing EXODUS file is managed through a series of data cards with the post-processing data block. Each data card begins with a keyword and is followed by one or more parameters that define the various options. The individual post-processing data cards recognized by CUERVO are listed below.

NODal DATA=name1, name2,....

name1, name2,.. (C) : specify the names of the nodal point variables that are to be written to the post-processing output file. The following variable names are allowed: PHI, PDOT (time rate of change of ϕ), XFLUX, YFLUX, ZFLUX (components of the flux vector), PFUNction (two-dimensional potential function) and VELX, VELY, VELZ (components of the material velocity vector).

ELEMent DATA=name1, name2,....

name1, name2,.. (C) : specify the names of the element variables that are to be written to the post-processing output file. There are no element variables currently available in CUERVO.

GLOBAL DATA=name1, name2,....

name1, name2,.. (C) : specify the names of the global variables that are to be written to the post-processing output file. The following variable names are recognized: TIMESTEP (the integration time step), CGITER (number of iterations

for matrix solver) and QN1, QN2, ... QN n , ... QN10 (integrated fluxes on element side sets). The n in the QN name refers to the sequence number of the side sets specified on the FLUX option (see Section 4.4).

OUTput FREQuency=nsteps

nsteps (I) $< 1 >$: specifies the frequency at which data is written to the post-processing file. All requested data is written every nsteps time steps or iteration cycles.

OUTput TIMEs= $t_1, t_2, \dots, t_n$

$t_1, t_2, \dots, t_n$ (R) : specify the simulation times at which data is written to the post-processing file. All requested data is written whenever the simulation time exceeds one of the specified output times. A maximum of fifty specific times may be specified; the times must be ordered in a strictly increasing sequence (see Note 1).

OUTput TIME STEP= Δt_{out}

Δt_{out} (R) : specifies the time interval at which data is written to the post-processing file. All requested data is written whenever the simulation time exceeds one of the times given by the sequence $t_{out} = t_{init} + n\Delta t_{out}$ where $n = 1, 2, 3, \dots$ (see Note 1).

Notes:

- 1) When requesting output at specified times or time intervals it is important to coordinate these requested times with the time step being used in the solution of the finite element equations. In general, output time intervals must be larger than solution time steps. If successive output times fall within a single computational time step, the test for output of a solution will fail and no further output will be generated. The output processor always checks to see if the next requested output time has been passed by the solution time; once the output time falls behind the solution time the output check will always fail.

4.7 Time Function Data Block

The specification of time functions for use in boundary conditions or material models is controlled by the time function data block and its associated data cards. A separate time function block is required for each time function used in a CUERVO simulation. Input to each time function is terminated by an **END** card. This data block is optional. The time function data block is of the following form:

```
TIME FUNction=function id  
t1, f(t1)  
t2, f(t2)  
t3, f(t3)  
.  
.  
.  
tn, f(tn)  
.  
.  
ENDtime
```

where

function id (I) : is a unique, integer identifier for the time function.

$t_n, f(t_n)$ (R) ; is a list of times and function values used to define the time function (see Note 1).

Notes:

- 1) The sequence of times for each function must form an increasing sequence. Also, no extrapolation or continuation of data beyond the last data point is allowed by CUERVO. If the solution time exceeds the time for the function the code will terminate execution with an error message. The function data is used with a linear interpolation scheme to provide variable values at intermediate times.

4.8 Variable Function Data Block

The specification of variable functions for use in boundary conditions or material models is controlled by the variable function data block and its associated data cards. A separate variable function block is required for each variable function used in a CUERVO simulation. Input to each variable function is terminated by an **END** card. This data block is optional. The variable function data block is of the following form:

VARiable FUNction=function id

$\phi_1, f(\phi_1)$

$\phi_2, f(\phi_2)$

$\phi_3, f(\phi_3)$

.

.

.

$\phi_n, f(\phi_n)$

.

.

ENDvariable

where

function id (I) : is a unique, integer identifier for the variable function.

$\phi_n, f(\phi_n)$ (R) ; is a list of variable and function values used to define the variable function.

In the current version of CUERVO, ϕ is the only variable allowed in the definition of a variable function (see Note 1).

Notes:

- 1) The sequence of ϕ for each function must form an increasing sequence. Also, no extrapolation or continuation of data beyond the last data point is allowed by CUERVO. If the current ϕ falls above or below the range of the function the code will terminate execution with an error message. The function data is used with a linear interpolation scheme to provide function values at intermediate values of ϕ .

4.9 User Constants Data Block

The specification of real or integer constants that may be useful within user supplied boundary condition or material property subroutines is controlled by the user constant data block and its associated data cards. The data supplied in this block is recorded in two vectors (real and integer) and passed to all user-supplied subroutines through their parameter lists. Subroutine evaluations that make use of these constants may therefore be altered through the input file, without modification and recompilation of the subroutines. This data block is optional. The user constants data block is of the following form:

USER CONstants

.

.

USER REAL=n, rvalue

.

.

.

USER INTeger=m, ivalue

.

.

ENDconstant

where

n,m (I) : are unique, integer indices that locate each constant within its appropriate vector (see Note 1).

rvalue, ivalue (R and I) ; are the real and integer values of the constants.

Notes:

- 1) Real and integer user defined constants are stored in two vectors called RCONST and ICONST, respectively. The indices specified on the data cards allow the user to precisely determine where in each vector the constant will be stored. Note that the RCONST and ICONST vectors are dynamically allocated and the length of each vector corresponds to the largest n and m values encountered in the input.

4.10 Termination Data

The termination of data within a particular data block is signaled with an **END** command and has been described in the previous sections. The termination of all data input to CUERVO is signaled by the data cards **EXIT** or **STOP**. This command must be the last data entry in the input file to ensure proper program termination; any data cards following the **EXIT** or **STOP** card are ignored.

4.11 User Supplied Subroutines

There are several instances which require the user to supply FORTRAN coded subroutines to CUERVO. The occurrence of variable material properties or source terms, and the specification of certain types of variable boundary or initial conditions result in the need for one or more user-supplied subroutines. The required formats for these subroutines will be described in the following sections; the user subroutine for initial conditions is described in Section 4.12. Skeleton versions of all user subroutines are supplied with the CUERVO code and may be used as templates for specific applications.

4.11.1 Material Property Subroutines

Variations in material properties due to variations in ϕ can normally be accommodated with function type data. However, when property variations are complex or depend on variables other than ϕ , then a subroutine evaluation may be more convenient. When any of the material property data cards **BETA**, **KAPPA** or **TENSOR ROTATION** are set to **USER** for any material in the problem (see Section 4.3), a user subroutine describing that material property must be supplied to CUERVO. Each user subroutine is used to evaluate the appropriate material property for *all* materials labeled as **USER** on the material data card; the user assigned material name allows specific materials to be identified within each subroutine.

Beta

The coefficient β for each material is evaluated with a subroutine of the following form:

```

      SUBROUTINE USRBET (BETA, PHI, XIP, YIP, ZIP, NAME, NUMIPT,
*                      TIME, KSTEP, RCONST, ICONST)
C
C      *****
C
C      CHARACTER*20 NAME
C
C      DIMENSION BETA(*), PHI(*)
C      DIMENSION XIP(*), YIP(*), ZIP(*)
C      DIMENSION RCONST(*), ICONST(*)
C
C      *****
C
C      FORTRAN coding to evaluate beta at the element
C      integration points for each material
```

C

```
RETURN
END
```

where the variables in the subroutine parameter list are

BETA(NUMIPT) : an array containing integration point values of β for the material (output).

PHI(NUMIPT) : an array containing integration point values of the dependent variable ϕ (input).

XIP(NUMIPT), YIP(NUMIPT), ZIP(NUMIPT) : arrays containing integration point values of the x, y and z coordinates (input).

NAME : a character variable specifying the material name (input).

NUMIPT : an integer specifying the number of integration points in the element (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

Kappa

The diffusion or conductivity tensor κ for each material is evaluated with a subroutine of the following form:

```
      SUBROUTINE USRKAP (COND11, COND22, COND33, PHI, XIP, YIP, ZIP,
C      *                  NAME, NUMIPT, TIME, KSTEP, RCONST, ICONST)
C
C      *****
C
C      CHARACTER*20 NAME
C
C      DIMENSION COND11(*), COND22(*), COND33(*)
C      DIMENSION PHI(*)
C      DIMENSION XIP(*), YIP(*), ZIP(*)
C      DIMENSION RCONST(*), ICONST(*)
```

```

C
C *****
C
C   FORTRAN coding to evaluate kappa at the element
C   integration points for each material
C
C   RETURN
C   END

```

where the variables in the subroutine parameter list are

COND11(NUMIPT), COND22(NUMIPT), COND33(NUMIPT) : arrays containing integration point values of the principle components of κ (κ_{11} , κ_{22} and κ_{33}) for the material (output). For isotropic materials set COND22=COND33=COND11; for two-dimensional problems COND33 need not be evaluated.

PHI(NUMIPT) : an array containing integration point values of the dependent variable ϕ (input).

XIP(NUMIPT), YIP(NUMIPT), ZIP(NUMIPT) : arrays containing integration point values of the x, y and z coordinates (input).

NAME : a character variable specifying the material name (input).

NUMIPT : an integer specifying the number of integration points in the element (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

Tensor Rotation

The local orientation of the diffusion tensor for each anisotropic material is evaluated with a subroutine of the following form:

```

      SUBROUTINE USRROT (XXHAT, YXHAT, ZXHAT, XYHAT, YYHAT, ZYHAT, XIP,
*                      YIP, ZIP, NAME, NUMIPT, TIME, KSTEP, RCONST,
*                      ICONST)
C

```

```

C *****
C
C CHARACTER*20 NAME
C
C DIMENSION XXHAT(*), YXHAT(*), ZXHAT(*)
C DIMENSION XYHAT(*), YYHAT(*), ZYHAT(*)
C DIMENSION XIP(*), YIP(*), ZIP(*)
C DIMENSION RCONST(*), ICONST(*)
C
C *****
C
C FORTRAN coding to evaluate the orientation of the principle material
C axes with respect to the global coordinate frame at the element
C integration points for each material
C
C RETURN
C END

```

where the variables in the subroutine parameter list are

XXHAT(NUMIPT), YXHAT(NUMIPT), ZXHAT(NUMIPT) : arrays containing integration point values of the x, y, z projection of a vector in the positive $\hat{x}$ direction. The x, y, z axes denote the global reference frame and the $\hat{x}, \hat{y}, \hat{z}$ axes are the principle material axes. For two-dimensional problems only the XXHAT and YXHAT variables need to be evaluated. See Section 4.3 for a further discussion of the reference frames.

XYHAT(NUMIPT), YYHAT(NUMIPT), ZYHAT(NUMIPT) : arrays containing integration point values of the x, y, z projection of a vector in the positive $\hat{y}$ direction. The x, y, z axes denote the global reference frame and the $\hat{x}, \hat{y}, \hat{z}$ axes are the principle material axes. For two-dimensional problems these variables need not be evaluated. See Section 4.3 for a further discussion of the reference frames.

XIP(NUMIPT), YIP(NUMIPT), ZIP(NUMIPT) : arrays containing integration point values of the x, y and z coordinates (input).

NAME : a character variable specifying the material name (input).

NUMIPT : an integer specifying the number of integration points in the element (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

4.11.2 Volumetric Sources

When the volumetric source parameter, **VOLume SOURce** (see Section 4.3) for one or more materials is set to **USER**, then CUERVO expects a source subroutine (USRVS) to be supplied by the user. This subroutines must have the following form:

```

      SUBROUTINE USRVS (VOLSR, PHI, XIP, YIP, ZIP, NAME, NUMIPT,
*                      TIME, KSTEP, RCONST, ICONST)
C
C      *****
C
C      CHARACTER*20 NAME
C
C      DIMENSION VOLSR(*), PHI(*)
C      DIMENSION XIP(*), YIP(*), ZIP(*)
C      DIMENSION RCONST(*), ICONST(*)
C
C      *****
C
C      FORTRAN coding to evaluate the volumetric source at the
C      element integration points for each material
C
      RETURN
      END
```

where the variables in the subroutine parameter list are

VOLSR(NUMIPT) : an array containing integration point values of the volumetric source (output).

PHI(NUMIPT) : an array containing integration point values of the dependent variable ϕ (input).

XIP(NUMIPT), YIP(NUMIPT), ZIP(NUMIPT) : arrays containing integration point values of the x, y and z coordinates (input).

NAME : a character variable specifying the material name (input).

NUMIPT : an integer specifying the number of integration points in the element (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

4.11.3 Material Velocity

Simple velocity fields for one or more blocks of elements can often be supplied directly to CUERVO through the **VELOCITY** data card that is defined in Section 4.4. More complex velocity fields require the use of a user-supplied subroutine. This option is invoked by setting the velocity specification parameter on the **VELOCITY** data card to **USER** and supplying CUERVO with a subroutine of the following form:

```
      SUBROUTINE USRVEL (IDBLK, NNODES, XNODE, YNODE, ZNODE, UVEL, VVEL,
*                      WVEL, TIME, RCONST, ICONST)
C
C      *****
C
      DIMENSION XNODE(*), YNODE(*), ZNODE(*)
      DIMENSION UVEL(*), VVEL(*), WVEL(*)
      DIMENSION RCONST(*), ICONST(*)
C
C      *****
C
C      FORTRAN coding to evaluate the components of the velocity vector
C      for each node in an element
C
      RETURN
      END
```

where the variables in the subroutine parameter list are

IDBLK : an integer specifying the id for the current element block (input).

NNODES : an integer specifying the number of nodes in the element (input).

XNODE(NNODES), YNODE(NNODES), ZNODE(NNODES) : arrays containing node values of the x, y and z coordinates (input).

UVEL(NNODES), VVEL(NNODES), WVEL(NNODES) : arrays containing node values of the x, y and z velocity components (output).

TIME : the value of the current time (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

4.11.4 Boundary Conditions

Boundary conditions that vary as a simple function of time or the dependent variable, ϕ , can generally be specified with function data. Complex boundary conditions may require the use of subroutines that describe the appropriate functional behavior of the boundary values or required coefficients. These subroutines may also be used to modify boundary conditions as a function of the iteration cycle and thereby improve convergence of the solution process. When any of the boundary condition data cards include the USER specification, a user subroutine of the appropriate type must be supplied to CUERVO. The correspondence between a particular boundary condition and its variation is established through the node set or side set parameter, which appears on the BCtype data card (Section 4.4). All boundary conditions of the same type are evaluated via a single subroutine; the node set or side set identification number allows specific boundary conditions to be identified within the subroutine.

Dirichlet Boundary Condition

A specified value of the dependent variable (bcname=DIRichlet) for a group of nodal points is evaluated by setting the bc specification parameter (Section 4.4) equal to USER and supplying a subroutine of the following form:

```
      SUBROUTINE USRDIR (PHIBC, XNODE, YNODE, ZNODE, IDNSET, TIME,
*                               KSTEP, RCONST, ICONST)
C
C *****
C
C   DIMENSION RCONST(*), ICONST(*)
C
C *****
C
C   FORTRAN coding to evaluate the dependent variable at a node
C
C   RETURN
C   END
```

where the variables in the subroutine parameter list are

PHIBC : the value of the dependent variable at the node (output).

XNODE, YNODE, ZNODE : nodal point values of the x, y and z coordinates (input).

IDNSET : an integer specifying the current node set id (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

Neumann Boundary Condition

An applied flux (bcname=NEUMann) for an element edge or element surface is evaluated by setting the bc specification parameter (Section 4.4) equal to USER and supplying a subroutine of the following form:

```
      SUBROUTINE USRNEU (FLUX, PHIS, XS, YS, ZS, NNODES, IDSSET, TIME,
      *                  KSTEP, RCONST, ICONST)
C
C      *****
C
      DIMENSION FLUX(*), PHIS(*)
      DIMENSION XS(*), YS(*), ZS(*)
      DIMENSION RCONST(*), ICONST(*)
C
C      *****
C
C      FORTRAN coding to evaluate the applied flux at the element
C      edge or surface nodes for each side set
C
      RETURN
      END
```

where the variables in the subroutine parameter list are

FLUX(NNODES) : an array containing nodal point values of the applied flux (output).

PHIS(NNODES) : an array containing nodal point values of the dependent variable ϕ (input).

XS(NNODES), YS(NNODES), ZS(NNODES) : arrays containing nodal point values of the x, y and z coordinates (input).

NNODES : an integer specifying the number of nodes on the element edge or surface (input).

IDSSET : an integer specifying the current side set id (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

Mixed Boundary Condition

The use of a mixed boundary condition (bcname=MIXed) of the form $\kappa_{ij} \frac{\partial \phi}{\partial x_j} n_i = \lambda \phi + \gamma$ may require that the variation of the coefficient λ and/or the right hand side function, γ , be specified through a user subroutine. The λ coefficient for an element edge or surface is evaluated by setting LAMBda=USER (Section 4.4) and supplying a subroutine of the following form:

```
      SUBROUTINE USRLAM (COEFL, PHIS, COEFG, XS, YS, ZS, NNODES, IDSSET,
*                        TIME, KSTEP, RCONST(*), ICONST(*))
C
C      *****
C
      DIMENSION COEFL(*), PHIS(*), COEFG(*)
      DIMENSION XS(*), YS(*), ZS(*)
      DIMENSION RCONST(*), ICONST(*)
C
C      *****
C
C      FORTRAN coding to evaluate the coefficient at the
C      element edge or surface nodes for each side set
C
      RETURN
      END
```

where the variables in the subroutine parameter list are

COEFL(NNODES) : an array containing nodal point values of the coefficient λ (output).

PHIS(NNODES) : an array containing nodal point values of the dependent variable ϕ (input).

COEFG(NNODES) : an array containing nodal point values of the right hand side function γ (input).

XS(NNODES), YS(NNODES), ZS(NNODES) : arrays containing nodal point values of the x, y and z coordinates (input).

NNODES : an integer specifying the number of nodes on the element edge or surface (input).

IDSSET : an integer specifying the current side set id (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCNST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

The right hand side function, γ , for a mixed boundary condition on an element edge or surface is evaluated by setting GAMMA=USER (Section 4.4) and supplying a subroutine of the following form:

```
      SUBROUTINE USRGAM (COEG, PHIS, XS, YS, ZS, NNODES, IDSSET, TIME,
*                      KSTEP, RCNST, ICONST)
C
C *****
C
C      DIMENSION COEFG(*), PHIS(*)
C      DIMENSION XS(*), YS(*), ZS(*)
C      DIMENSION RCNST(*), ICONST(*)
C
C *****
C
C      FORTRAN coding to evaluate the right hand side function, gamma,
C      at the element edge or surface nodes for each side set
C
C      RETURN
C      END
```

where the variables in the subroutine parameter list are

COEFG(NNODES) : an array containing nodal point values of the right hand side function γ (output).

PHIS(NNODES) : an array containing nodal point values of the dependent variable ϕ (input).

XS(NNODES), YS(NNODES), ZS(NNODES) : arrays containing nodal point values of the x, y and z coordinates (input).

NNODES : an integer specifying the number of nodes on the element edge or surface (input).

IDSSET : an integer specifying the current side set id (input).

TIME : the value of the current time (input).

KSTEP : an integer specifying the number of the current iteration cycle or time step (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

4.12 Initial Conditions and Restarts

The analysis of a transient problem requires the specification of a set of initial conditions for the dependent variables. For cases where the dependent variable may be assumed uniform over each material, the initial conditions may be set through the input parameter available in the material data block (see Section 4.3). For problems in which the initial fields are more complex, CUERVO allows the initial conditions to be input from a user-supplied subroutine or an external file. To be compatible with CUERVO, the initial conditions specified from a file must be written as a standard EXODUS II [6] file. The mechanics of attaching a restart file to a CUERVO execution are described in Section 5.3.

The output file containing the solutions generated by CUERVO has the same EXODUS II format as described above. Therefore, solutions obtained from CUERVO may generally be used directly as initial conditions for subsequent problems.

Because the input and output files for the solution share a common format, the code can be easily restarted from a particular point in a previous solution. Such a procedure consists of attaching a previous solution file, re-executing the input file and specifying within the solution data block (see Section 4.5) the iteration/timeplane number or time at which the solution is to be restarted. The input file must be re-executed because there is no explicit provision made for storing all of the data from a previous run. Typically, the computation time for problem setup and initialization is small enough that the storage of this information for a possible restart operation is not warranted. Note that when restarts are carried out using the above procedure, the resulting solution file contains the time history of the problem only from the restart timeplane.

4.12.1 Initial Condition Subroutine

As noted above and in Section 4.3, the initial conditions for one or more materials can be specified through a user-supplied subroutine. When the **INITIAL VALUE** material data card is set to **USER** for any material in the problem (see Section 4.3), then a user supplied subroutine describing the initial distribution of the dependent variable must be supplied to CUERVO. This subroutine is used to evaluate the initial conditions for *all* materials with the **USER** designation; the user-assigned material name allows specific materials to be identified within the subroutine.

The initial condition subroutine has the following form:

```
SUBROUTINE USRIC (PNIC, XN, YN, ZN, NAME, NNODES, RCONST, ICONST)
C
C *****
C
```

```

      CHARACTER*20 NAME
C
      DIMENSION PNIC(*)
      DIMENSION XN(*), YN(*), ZN(*)
      DIMENSION RCONST(*), ICONST(*)
C
C *****
C
C   FORTRAN coding to evaluate the initial conditions for an element
C
      RETURN
      END

```

where the variables in the subroutine parameter list are

PNIC(NNODES) : an array containing the initial nodal point values of the dependent variable ϕ (output).

XN(NNODES), YN(NNODES), ZN(NNODES) : arrays containing nodal point values of the x, y and z coordinates (input).

NAME : a character variable specifying the material name (input).

NNODES : an integer specifying the number of nodes in the mesh (input).

RCONST(*) : an array containing (real) user-defined constants.

ICONST(*) : an array containing (integer) user-defined constants.

4.13 Error Messages

CUERVO has been constructed with a number of error checks and tests for bad or inconsistent data, storage problems, *etc.* When an error is encountered, an error message and other pertinent data are printed and the program is terminated with a STOP ERROR. The error citations usually contain sufficient information to allow the problem to be diagnosed and corrected. For each listed error condition, the subroutine encountering the error is given along with an explanatory message. Also, any relevant integers and/or character data that help to explain the problem are output by the code.

The general format for error messages takes the following form,

ERROR FOUND IN - 'SUBROUTINE NAME'

DESCRIPTION - 'ERROR MESSAGE'

RELEVANT PARAMETERS -

MESSAGE 1 = 'INTEGER DATA'

MESSAGE 2 = 'INTEGER DATA'

MESSAGE 3 = 'FLOATING POINT DATA'

MESSAGE 4 = 'FLOATING POINT DATA'

MESSAGE 5

MESSAGE 6 = 'CHARACTER DATA'

5 Code Installation and Access

The use of a program such as CUERVO generally requires some knowledge of how the program is installed and operates on a specific computer system. In particular, details regarding the use of system and utility libraries, file usage, file formats, FORTRAN coding standards and access to the program are important to the successful and efficient use of the code. These topics become especially critical when the code is used for the analysis of very complex problems, when the code is modified for a new application or when the program is transported to a new computer system. In the following sections, all of the above topics are discussed in some detail. The main emphasis will be on the operation of CUERVO on the Sandia National Laboratories computer systems; the operation of the code on other computers should involve only minor alterations to the code and operating procedures.

5.1 FORTRAN Coding and System Dependencies

The major part of the CUERVO program is written in ANSI standard FORTRAN 77 and should therefore be usable on any computer system that supports such a compiler. The iterative matrix solver used in CUERVO resides in a general software package developed by Schunk and Shadid [5] and is written in standard C. The compilation and linking of the various CUERVO modules is most conveniently handled through a standard UNIX makefile which is available with the source distribution of the code.

As indicated in Section 3.2, CUERVO is a modular program that consists of a main driver routine and a number of subordinate, task-oriented subroutines and utilities. Communication between the main program and the various subroutines is accomplished primarily through named common blocks and subroutine parameter lists. Named common blocks generally contain pointers, option flags, sizing parameters and data that is invariant with respect to problem specification. Data that is stored in vectors and arrays are normally passed through subroutine parameter lists due to the use of a dynamic memory allocation scheme that is described below. A description of all of the major common blocks and arrays used in CUERVO is included in Appendix B. This information would normally be of value only if the code is to be modified.

A dynamic dimensioning procedure is used in CUERVO to allocate and release computer memory during program execution. Under this algorithm the individual vectors and arrays needed by the program are stored in (noncontiguous) blocks of memory under a single vector name. Pointers indicating the location of individual arrays within memory are maintained by the memory manager as is the allocation of needed storage space. All memory allocation is performed in the main program. Further details on the memory manager and its operation are given in [8]. Some computer systems may not permit execution-time changes in memory allocation; modifications to the code and memory manager to handle this case are outlined in [8].

CUERVO also makes use of a few system-dependent utilities. To increase portability of the code all of the system dependencies have been isolated in a few subroutines that are located in a utility library that accompanies the main code. The utilities available in the SUPES library [8] include date and time stamping, the memory manager and parsing of input for use in the free-field input routines. These library routines are available for a number of commonly used computer systems and are heavily commented to ease the task of converting these utilities to other, nonsupported computers.

The pre- and postprocessing files currently utilized by CUERVO are in the EXODUS II format [6], which is a random access file structure based on NetCDF [9]. Access to these files is accomplished through a FORTRAN interface to a library of C subroutines that read and write data in the specified format. All of the mesh generation programs and graphics packages that are to be used in conjunction with CUERVO must provide data in the EXODUS II format. Detailed documentation for the contents of EXODUS II files is given in [6].

The standard version of CUERVO was developed for use on a spectrum of computer systems ranging from large vector supercomputers to engineering workstations. The code is also suitable for smaller computer systems, provided that the memory is sufficient to handle the required problem size. There are relatively few built-in limits in CUERVO with respect to the size or complexity of the model. The code is designed to handle problems with an arbitrary number of elements, materials, boundary conditions, *etc.* up to the capacity of the computer memory. The largest portion of memory is used for the storage of the assembled global diffusion matrix; this array is stored in a sparse matrix format [5,10] suitable for use with the iterative matrix solver.

5.2 File Formats

The disk files used internally by CUERVO are generally sequential access, unformatted files. The specific form of these files is not usually of concern to the user except for the pre- and postprocessing files and the restart (or solution) file (see Section 4.12).

As noted in the previous section, the pre- and postprocessing files follow the EXODUS II format. Within CUERVO II the pre- and postprocessing files are maintained separately to allow for reuse, though they share a substantial data set. The pre-processing file contains information from the mesh generator regarding element coordinates, connectivity, element types and boundary condition flags. The postprocessing file contains a copy of the pre-processing data in addition to the computed solution variables. The specific contents of these files are given in [6]; instructions for accessing EXODUS II data through the subroutine interface is also available in [6].

5.3 File Usage

CUERVO makes use of a series of disk files for the storage of large blocks of input and element data, storage of solution vectors and input and output functions. All of the files used by the program are listed in Table 2 with a brief description of their function, their internal FORTRAN identification and format type. Specific formats for these files are not given as that information is readily available from a code listing. Previous sections have commented on those file formats that would normally be of concern to the user.

When files are to be attached to a CUERVO job or saved after processing, the unit numbers used in the job control statements must conform to those indicated in the table. File names are arbitrary but must conform to the syntax rules for the particular operating system. The proper relation between file name and unit number must also be established in the job control statements. A utility program from the SUPES library [8] obtains needed file names from the operating system for use in opening files during program execution. As described in the next section, a script is available for CUERVO that simplifies the association of local file names with unit numbers. Note that within CUERVO all files are opened in the OPNFIL subroutine.

Unit Number	FORTTRAN Name/ID	File Usage	File Access	File Format
5	NIN	Input file	Sequential	Coded
6	NOUT	Output file	Sequential	Coded
10	NTP0	EXODUS II input data	Random Access	Binary
11	NTP1	Free field input	Sequential	Binary
12	NTP2	EXODUS II output data	Random Access	Binary
13	NTP3	Unused		Binary
14	NTP4	EXODUS II restart data	Random Access	Binary
15	NTP5	Unused		Binary

Table 2: Files used in CUERVO

5.4 Access to the Code

A source version of CUERVO is maintained on the Network Storage System (NSS) of the Central Computer Facility at Sandia National Laboratories and on the Local Area Networks (LANs) operated by the Engineering Sciences Center, 9100. The current location of the program source code for any particular computer system may be obtained from the authors.

The utility libraries needed for the execution of CUERVO are available for several computer systems. Access to these routines varies depending on the particular operating

system. Instructions for accessing and loading these routines are available in [5,6,8]. The required libraries include the EXODUS II file access routines, the iterative matrix solution package and the SUPES utilities. When user-supplied subroutines are employed with CUERVO they must be compiled and then made available to the loader as a library to be searched for unsatisfied external references.

In order to assist users in the running of CUERVO, a UNIX script is also available on the 9100 LAN's and through the SEACAS system [11]. This script has a standard UNIX, single-line, input style and allows execution of the code in either a batch or interactive mode on the host computer.

6 Example Problems

A series of three problems have been included in this section to demonstrate the use of the previously described data cards and some of the capabilities of the CUERVO program. Though the examples were designed to illustrate the basic features of the code, all possible options and subtleties of the program could not be covered.

6.1 Problem 1 - Electrostatic Field Problem

The first example concerns the electrostatic analysis of a porcelain insulator with two embedded electrodes. This problem is taken from the example problems for the electromagnetics code, MSC/EMAS [12] and was also used to demonstrate the EM code TORO II [13]. Figure 5 shows the geometry of interest; dimensions may be obtained from reference [12]. The boundary value problem in this case corresponds to a reduced form of equation (1)

$$\frac{\partial}{\partial x_i} \left[\kappa \frac{\partial \phi}{\partial x_i} \right] = 0$$

where κ is now interpreted as the (constant) electric permittivity of the material and ϕ is the electric potential. The above equation is to be solved over the various material regions including the porcelain insulator and the surrounding air space. In the present model the field is also evaluated in the electrodes, though they are held at a uniform potential.

Figure 5: Schematic of porcelain insulator and electrodes.

The finite element mesh for this axisymmetric geometry was developed using the FASTQ [14] mesh generator. The boundary for the air region was moved sufficiently far from the region of interest that a constant potential boundary condition could be applied and have a minimal influence on the solution near the electrodes. The axisymmetric mesh of four-node quadrilateral elements is shown in Figure 6. A three-dimensional mesh was also generated to allow CUERVO results to be obtained and checked in the higher dimensionality. The eight-node hexahedral element mesh is shown in Figure 6 and was produced by sweeping the axisymmetric mesh through 180° using the GEN3D [15] utility.

The input file to CUERVO for these simulations is shown in Figure 7. The only differences in the axisymmetric and three-dimensional input data are in the GEOM card and the POST definition of the nodal variables to be output to the post-processing file. This is a symmetric, linear problem so that the solution only requires a single iteration of the Picard scheme; the default matrix solver was used.

Shown in Figure 8 are contour plots of the (electric) potential and magnitude of the vector flux (electric field) fields. The magnitude of the flux was generated by use of the ALGEBRA [16] utility. Note that individual components of the flux could also have been contoured as well as vector plots of the flux. The plots were obtained from the BLOT [17] graphics package. The results shown in Figure 8 compare well with the results from the EMAS code [12] and are exactly reproduced by the related code TORO II [13]. Also, as expected, the three-dimensional results are reproduced by the axisymmetric model with the two-dimensional solution executing approximately 30 times faster; the axisymmetric model is an order of magnitude smaller (number of elements) than the three-dimensional model.

Figure 6: Axisymmetric and three-dimensional meshes for insulator, electrodes and surrounding air space.

```

TITLE
PORCELAIN INSULATOR
EXAMPLE PROBLEM FOR CUERVO
AXISYMMETRIC AND THREE-DIMENSIONAL ELECTRIC POTENTIAL PROBLEM
END
MATERIAL,PORCELAIN
KAPPA=4.4
END
MATERIAL,AIR
KAPPA=1.0
END
MATERIAL,COND1
KAPPA=25.0
END
MATERIAL,COND2
KAPPA=25.0
END
PROBLEM DEFINITION
GEOMETRY=AXISYMMETRIC          $ GEOMETRY=3D
ELEMENT BLOCK=10,PORCELAIN
ELEMENT BLOCK=20,AIR
ELEMENT BLOCK=30,COND1
ELEMENT BLOCK=40,COND2
BC=DIRICHLET,10,0.0
BC,DIRICHLET,20,10000.
BC,DIRICHLET,30,1000.
MAXIMUM CONNECTIVITY=40
FLUX
END
SOLUTION=1,TIME INDEPENDENT
ITERATIVE METHOD=PICARD
NORM VALUE=1.0
MAXIMUM ITERATIONS=1
PRINT OUT=SUMMARY,1
END
POST
NODAL DATA=PHI,XFLUX,YFLUX    $ NODAL DATA=PHI,XFLUX,YFLUX,ZFLUX
OUTPUT FREQUENCY=1
END
STOP

```

Figure 7: Input file for CUERVO simulation of electrostatic field problem.

Electric Potential

Electric Field (Magnitude)

Figure 8: Contour plots of the potential and magnitude of the flux.

6.2 Problem 2 - Time-Independent Thermal Diffusion

A time-independent diffusion problem is considered in the second example and corresponds to a thermal problem originally analyzed by Ting and Peddicord [18]. A three-dimensional model of their experimental configuration is shown in Figure 9. The apparatus consists of a volumetrically heated cylinder, eccentrically located within a filled cylindrical tube. The ends of the device are insulated and a temperature is measured (specified) circumferentially on the outside wall of the outer tube. Due to the large length to diameter ratio of the geometry and the insulated end condition, this configuration can be analyzed with a two-dimensional cross-section of the cylinder that is further simplified by symmetry. The modeled region and finite element mesh of four-node quadrilateral elements generated by FASTQ [14] is shown in Figure 10.

The boundary value problem corresponds to equation (7) with an isotropic, scalar conductivity

$$\frac{\partial}{\partial x_i} \left[\kappa \frac{\partial \phi}{\partial x_i} \right] + Q = 0$$

where κ is now the thermal conductivity, Q is the volume heating and ϕ is the temperature. The thermal properties corresponding to the three materials are given in [18] and can be found in the CUERVO input file in Figure 11. The temperature specified on the outside tube wall was given in [18] as a function of angular position around the tube; this boundary condition was provided to CUERVO through the user subroutine shown in Figure 12.

Figure 9: Schematic of the eccentric annulus problem of [18].

Figure 10: Finite element mesh for the two-dimensional cross section of the eccentric annulus problem.

A contour plot of the temperature field is shown in Figure 13. The power level corresponds to the case considered by Ting and Peddicord and the results are in substantial agreement with the measured and computed fields reported in [18]. Also shown in Figure 14 is a contour plot of the magnitude of the conductive heat flux in the composite cylinder. These plots were produced by post-processing the CUERVO output through the BLOT program [17] with data manipulation handled in the ALGEBRA utility [16].

```

TITLE
ECCENTRIC ANNULUS - 2D
THERMAL ANALYSIS OF A COMPOSITE CYLINDRICAL REGION
END
MATERIAL,HEATER
KAPPA=.00265
VOLUME SOURCE=4.8423E-2
END
MATERIAL,ANNULUS
KAPPA=.00479
END
MATERIAL,TUBE
KAPPA=.00186
END
PROBLEM DEFINITION
GEOMETRY=2D
ELEMENT BLOCK=10,HEATER
ELEMENT BLOCK=20,ANNULUS
ELEMENT BLOCK=30,TUBE
BC=DIRICHLET,10,USER
MAXIMUM CONNECTIVITY=20
FLUX
END
SOLUTION=1,TIME INDEPENDENT
ITERATIVE METHOD=PICARD
NORM VALUE=303.0
MAXIMUM ITERATIONS=1
PRINT OUT=SUMMARY,1
END
POST
NODAL DATA=PHI,XFLUX,YFLUX
OUTPUT FREQUENCY=1
END
STOP

```

Figure 11: Input file for CUERVO simulation of eccentric annulus problem.

```

SUBROUTINE USRDIR (PHIBC,XNODE,YNODE,ZNODE,IDNSET,TIME,KSTEP,
*                RCONST,ICONST)
C
C *****
C
C DIMENSION RCONST(*), ICONST(*)
C
C *****
C
C USER SUPPLIED FORTRAN CODE TO EVALUATE THE DEPENDENT VARIABLE
C AT A BOUNDARY NODE
C
C THE DIRICHLET BOUNDARY CONDITION IS
C
C PHIBC=303.667 + 3.0*COS(THETA)+0.33*COS(2*THETA)
C
C WHICH IS SPECIFIED ON THE OUTSIDE BOUNDARY OF A CIRCULAR DOMAIN
C
C
C FIND ANGULAR LOCATION OF NODE
C
C RADIUS=SQRT(XNODE**2+YNODE**2)
C THETA=ASIN(YNODE/RADIUS)
C
C EVALUATE FUNCTION
C
C PHIBC=303.667 + 3.0*COS(THETA)+0.33*COS(2*THETA)
C
C RETURN
C END

```

Figure 12: Boundary condition subroutine for CUERVO simulation of the eccentric annulus problem.

Figure 13: Contour plot of the temperature field for the eccentric annulus problem.

Figure 14: Contour plot of the heat flux magnitude for the eccentric annulus problem.

Figure 15: Schematic of rotating “cosine hill” problem, initial condition.

6.3 Problem 3 - Time-Dependent Advection-Diffusion

The last example is a standard test for numerical methods and time integration methods that utilizes the advection-diffusion equation. As outlined by Baker [19] and Gresho, *et al.* [20], the problem considers the initial condition of an isolated “cosine hill”, that is subjected to a velocity field in the form of a solid body rotation about a point in a plane. A schematic of the problem is shown in Figure 15. Many numerical methods have difficulty with this problem and are unable to propagate the initial field without artificial dissipation or phase error. The problem is especially difficult when advection is significantly larger than diffusion, *i.e.*, the Péclet number, $Pe = UL/\alpha$ ($\alpha = \kappa/\beta$), is large in the nondimensional form of equation (1)

$$\left(\frac{\partial \phi}{\partial t} + u_j \frac{\partial \phi}{\partial x_j} \right) = Pe^{-1} \frac{\partial}{\partial x_i} \frac{\partial \phi}{\partial x_i}.$$

In the above definition U is a characteristic velocity for the transport problem and L is a characteristic length for the domain.

The CUERVO test was initially run on a uniform 32×32 four-node element grid in the x, y plane. The mesh was produced by FASTQ [14]. The initial condition was a uniform $\phi = 0$ field with a cosine function $\phi = a + b \cos(\pi r / 2r_{hill})$ ($a = 0, b = 2$ and $r_{hill} = 4$) centered at the point $x = 10, y = 0$. This initial condition was specified through the user subroutine USRIC as indicated in the CUERVO input file, Figure 16, and the subroutine listing in Figure 17. User defined constants were employed to parameterize the geometry of the cosine distribution and allow changes to be made without recompiling the

user subroutine. The imposed velocity field was $u(x, y) = r\dot{\omega}$ where r is measured from the centerpoint of the mesh and $\dot{\omega} = 0.10$. The velocity field was supplied to CUERVO through the user subroutine listed in Figure 18. Note that a Dirichlet boundary condition was specified over the “inflow” portions of the boundary (half the nodes on each side of the domain) while a natural boundary condition was imposed on the outflow sections. The solution was generated using a fixed time step trapezoid rule and was run for a full revolution of the cosine hill about the centerpoint. The GMRES matrix solver was used because the advection-diffusion system is nonsymmetric.

Shown in Figure 19 are a series of contour plots showing the progression of the hill for the case of an infinite Peclet number, *i.e.*, $\beta = 0$ and there is no diffusion in the problem. The exact solution for this case is the rotation of the initial profile about the centerpoint without alteration. The computed result shown in Figure 19 is for a Courant number ($C = U\Delta t/h$ where Δt is the time step and h is the mesh size) of $C = 0.1$. The shape of the initial profile is well maintained throughout the rotation and numerical diffusion (reduction in the amplitude of the hill) is negligible. The contour bands that appear after one quarter of a revolution and trail the cosine profile are an indication of the dispersion error in the solution. The discrete modes that make up the cosine profile are advected at slightly different speeds leading to the formation of the wake; the magnitude of the wake is approximately 5%. However, the overall phase behavior of the solution is very good with the centerpoint of the hill arriving at the quarter rotation points at the correct time. Use of other (dissipative) time integration methods, the use of mass lumping or the selection of significantly higher Courant numbers can lead to significant decreases in the accuracy of the solution. Simulations with finite Peclet numbers may also be computed with the problem becoming less difficult as diffusion effects are increased.


```

TITLE
ROTATING COSINE HILL
CONE RADIUS=4, AMPLITUDE=2
ROTATION RADIUS=10, ROTATION RATE=0.1
MESH 32 X 32, FOUR-NODE QUADS
END
MATERIAL,MAT1
BETA=1.0
KAPPA=0.0
INITIAL VALUE=USER
END
PROBLEM DEFINITION
GEOMETRY=2D
ELEMENT BLOCK=10,MAT1
BC=DIRICHLET,10,0.0
VELOCITY=EULERIAN,10,USER
END
SOLUTION=1,TIME DEPENDENT
INTEGRATION METHOD=TRAPEZOID,NOPREDICT
TIME STEP OPTION=FIXED
NORM VALUE=1.0
TIME STEP=0.1
INITIAL TIME=0.0
FINAL TIME=65.
PRINT OUT=SUMMARY,1
MATRIX SOLVER=GMRES
PRECONDITIONER TYPE=ILU
END
POST
NODAL DATA=PHI,VELX,VELY
OUTPUT FREQUENCY=5
END
USER CONSTANTS
USER REAL=1,0.10      $ ROTATION RATE
USER REAL=2,10.0      $ RADIUS OF ROTATION
USER REAL=3,4.1        $ COSINE HILL RADIUS
USER REAL=4,10.0      $ X COORDINATE - CENTERPOINT OF COSINE HILL
USER REAL=5,0.0        $ Y COORDINATE - CENTERPOINT OF COSINE HILL
USER REAL=6,2.0        $ AMPLITUDE OF COSINE HILL
END
STOP

```

Figure 16: Input file for CUERVO simulation of rotating "cosine hill".

```

SUBROUTINE USRIC (PNIC,XN,YN,ZN,NAME,NNODES,RCONST,ICONST)
C
C *****
C
C CHARACTER*20 NAME
C
C DIMENSION PNIC(*)
C DIMENSION XN(*), YN(*), ZN(*)
C DIMENSION RCONST(*), ICONST(*)
C
C PARAMETER (PI=3.14159)
C
C *****
C
C USER SUPPLIED FORTRAN CODE TO EVALUATE THE INITIAL CONDITION
C
C RADIUS OF ROTATION      - ROTRAD=RCONST(2)
C RADIUS OF COSINE HILL   - HILRAD=RCONST(3)
C CENTER POINT OF HILL    - XHILL=RCONST(4)
C                          - YHILL=RCONST(5)
C AMPLITUDE OF HILL       - AMP=RCONST(6)
C
C ROTRAD=RCONST(2)
C HILRAD=RCONST(3)
C XHILL=RCONST(4)
C YHILL=RCONST(5)
C AMP=RCONST(6)
C
C CHECK LOCATION OF NODES, IF WITHIN HILL SET TO COSINE FUNCTION
C                          IF OUTSIDE HILL SET TO 0.0
C
C DO 10 I=1,NNODES
C RAD=SQRT((XN(I)-XHILL)**2+(YN(I)-YHILL)**2)
C IF (RAD.LE.HILRAD) THEN
C   PNIC(I)=AMP*COS((PI*RAD)/(2.0*HILRAD))
C ELSE
C   PNIC(I)=0.0
C END IF
10 CONTINUE
C
C RETURN
C END

```

Figure 17: User subroutine for initial conditions.

```

SUBROUTINE USRVEL (IDBLK,NNODES,XNODE,YNODE,ZNODE,UVEL,VVEL,WVEL,
*
TIME,RCONST,ICONST)
C
C *****
C
DIMENSION XNODE(*), YNODE(*), ZNODE(*)
DIMENSION UVEL(*), VVEL(*), WVEL(*)
DIMENSION RCONST(*), ICONST(*)
DIMENSION UTAN(8), RAD(8), THETA(8)
C
C *****
C
C USER SUPPLIED FORTRAN CODE TO EVALUATE THE COMPONENTS OF THE
C VELOCITY VECTORS, UVEL, VVEL, WVEL
C
C ANGULAR VELOCITY, OMEGA=RCONST(1)
C
OMEGA=RCONST(1)
C
C COMPUTE TANGENTIAL VELOCITIES, POLAR ANGLES AND RADII
C
DO 10 I=1,NNODES
RAD(I)=SQRT(XNODE(I)**2+YNODE(I)**2)
IF (RAD(I).LT.1.0) THEN
UTAN(I)=0.0
THETA(I)=0.0
ELSE
UTAN(I)=RAD(I)*OMEGA
THETA(I)=ATAN2(YNODE(I),XNODE(I))
END IF
10 CONTINUE
C
C COMPUTE VELOCITY COMPONENTS
C
DO 20 I=1,NNODES
UVEL(I)=-UTAN(I)*SIN(THETA(I))
VVEL(I)= UTAN(I)*COS(THETA(I))
20 CONTINUE
C
RETURN
END

```

Figure 18: User subroutine for velocity field.

Figure 19: Contour plots for the “cosine hill” problem, top view at various times.

7 References

1. D. K. Gartling and R. E. Hogan, "COYOTE II - A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part I - Theoretical Background," SAND94-1173, Sandia National Laboratories, Albuquerque, NM (1994)
2. D. K. Gartling and R. E. Hogan, "COYOTE II - A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part II - User's Manual," SAND94-1180, Sandia National Laboratories, Albuquerque, NM (1994)
3. E. B. Becker, G. F. Carey and J. T. Oden, "*Finite Elements, An Introduction, Volume I*," Prentice-Hall, NJ (1981)
4. J. N. Reddy and D. K. Gartling, "*The Finite Element Method in Heat Transfer and Fluid Dynamics*," CRC Press, Boca Raton, FL (1994)
5. P. R. Schunk and J. N. Shadid, "Iterative Solvers in Implicit Finite Element Codes," SAND92-1158, Sandia National Laboratories, Albuquerque, NM (1992)
6. L. A. Schoof and V. R. Yarberr, "EXODUS II - A Finite Element Data Model," SAND92-2137, Sandia National Laboratories, Albuquerque, NM (1994)
7. S. Kimura and A. Bejan, "The 'Heatline' Visualization of Convective Heat Transfer," *J. Heat Trans.*, **105**, 916-919 (1983)
8. J. R. Red-Horse, W. C. Mills-Curran, and D. P. Flanagan, "SUPES Version 2.1 - A Software Utilities Package for the Engineering Sciences," SAND90-0247, Sandia National Laboratories, Albuquerque, NM (1990)
9. "NetCDF User's Guide; An Interface for Data Access," Version 1.11, Unidata Program Center (1991)
10. Y. Saad, "SPARSKIT: A Basic Tool for Sparse Matrix Computations," Tech. Report, Research Institute for Advanced Computer Science, NASA Ames, Moffitt Field, CA (1990)
11. G. D. Sjaardema, "Overview of the Sandia National Laboratories Engineering Analysis Code Access System," SAND92-2292, Sandia National Laboratories, Albuquerque, NM (1993)
12. "Semi-Guided Exercises for MSC/EMAS," MSC/EMAS Version 2, MacNeal/Schwendler Corp. (1992)
13. D. K. Gartling, "TORO II - A Finite Element Program for Nonlinear Quasi-Static Problems in Electromagnetics, Part II - User's Manual," SAND95-xxxx, Sandia National Laboratories, Albuquerque, NM (1995)
14. T. D. Blacker, "FASTQ Users Manual, Version 1.2," SAND88-1326, Sandia National Laboratories, Albuquerque, NM (1988)

15. G. D. Sjaardema, "GEN3D: A GENESIS Database 2D to 3D Transformation Program," SAND89-0485, Sandia National Laboratories, Albuquerque, NM (1989)
16. A. P. Gilkey, "ALGEBRA - A Program That Algebraically Manipulates the Output of a Finite Element Analysis," SAND86-0881, Sandia National Laboratories, Albuquerque, NM (1986)
17. A. P. Gilkey and J. H. Glick, "BLOT - A Mesh and Curve Plot Program for the Output of a Finite Element Analysis," SAND88-1432, Sandia National Laboratories, Albuquerque, NM (1989)
18. Y. P. Ting and K. L. Peddicord, "Heat Conduction in Eccentric Annuli," *Int. J. Heat Mass Trans.*, **26**, 11-21 (1983)
19. A. J. Baker, *Finite Element Computational Fluid Mechanics*, McGraw-Hill, New York (1983)
20. P. M. Gresho, R. L. Lee and R. L. Sani, "Advection-Dominated Flows, with Emphasis on the Consequences of Mass Lumping," in *Finite Elements in Fluids, Volume 3*, R. H. Gallagher, O. C. Zienkiewicz, J. T. Oden, M. Cecchi and C. Taylor, Eds., John Wiley, New York (1978)

Appendix A - Summary of Input Commands

In this section all of the data block and associated data cards recognized by CUERVO are summarized. No attempt is made to define the parameters on each input line since these descriptions are available in the main text.

Title Data Block :

TITle

THIS IS AN EXAMPLE OF A PROBLEM TITLE

THESE ARE EXAMPLES OF COMMENT LINES
UP TO 10 LINES OF PROBLEM DESCRIPTION MAY
BE INCLUDED ON THESE CARDS

ENDtitle

Material Data Block :

MATerial, material name, material model

☐

BETa= β

BETa=USER

BETa=VFUNction, idvar

KAPpa= $\kappa_{11}, \kappa_{22}, \kappa_{33}$

☐

KAPpa=USER

☐

KAPpa=VFUNction, idvar1, idvar2, idvar3

☐

TENsor ROTation= $x\hat{x}, y\hat{x}, z\hat{x}, x\hat{y}, y\hat{y}, z\hat{y}$

TENsor ROTation=USER

VOLume SOURce= Q

VOLume SOURce=USER

VOLume SOURce=TFUNction, idtim

VOLume SOURce=VFUNction, idvar

INITial VALue= ϕ_{init}
INITial VALue=USER

ENDmaterial

Problem Definition Data Block :

PROBLEM DEFinition, *format*

GEOMetry=type

ELEMent BLOCk= block id, material name, *integration rule*

BCtype=bcname, nodeset id, bc specification, *mfactor*

BCtype=bcname, sideset id, bc specification, *mfactor*

BCtype=bcname, sideset id, LAMBda = coef specification, *mfactor*,
GAMma = rhs specification, *mfactor*

VELocity=type, block id, velocity specification

PRINTed OUTput=type

OUTput LOCations=e1, e2, e3, ... , en TO em, ...

FLUX=*sideset id1, sideset id2, ... sideset idn*

POTential FUNCtion= $\mathcal{P}_0$

MAXimum CONnectivity=maxcon

ENDproblem

Solution Data Block :

SOLution, solution block number, time dependence

□

.

.

REStart TIME= $t_{restart}$

.

.

REStart STEP=*step number*

.

.

ITERative METHod=scheme

.

.

INTEgration METHod=scheme, *predictor option*

.

.

CAPacitance MATRix=type

.

.

TIME STEP OPTion=type

.

.

INTEgration TOLerance=integ tol

.

.

NORM VALue= ϕ_{norm}

.

.

TIME STEP FACTor= $\beta_{explicit}$

.

.

TIME STEP= Δt

□

.

.

INITial TIME= t_{init}

.

.

FINAl TIME= t_{final}

□

.

.

NUMber TIME STEPs=nstep

□

.

.

ABSolute LIMit= ϕ_{lim}

MINimum TIME STEP= Δt_{min}

MAXimum TIME STEP= Δt_{max}

MAXimum VARIABLE STEP= $\Delta \phi_{max}$

CONvergence TOLerance=tol

RELaxation FACTor= α

MAXimum ITERations=itermax

PRINTed OUTput=option, frequency

MATRIX SOLver=method, *krylov subspace*

PREConditioner TYPE=option, *polynomial order*

L2 NORM= L_2

RESidual NORM=resid

MAXimum MATRIX ITERations=matrix iter

ENDsolution

□

Post-processing Data Block :

POST

.
NODal DATA=name1, name2,....

.
ELEMent DATA=name1, name2,....

.
GLOBAL DATA=name1, name2,....

.
OUTput FREQuency=nsteps

.
OUTput TIMES= $t_1, t_2, \dots t_n$

.
OUTput TIME STEP= Δt_{out}

.
ENDpost

Time Function Data Block :

TIME FUNction=function id

$t_1, f(t_1)$

$t_2, f(t_2)$

$t_3, f(t_3)$

.
 $t_n, f(t_n)$

.
ENDtime

Variable Function Data Block :

VARIABLE FUNction=function id

$\phi_1, f(\phi_1)$

$\phi_2, f(\phi_2)$

$\phi_3, f(\phi_3)$

.
 $\phi_n, f(\phi_n)$

.
.
ENDvariable

User Constants Data Block :

USER CONstants

.
.
USER REAL=n, rvalue

.
.
.
USER INTeger=m, ivalue

.
.
ENDconstant

Termination Data :

EXIT or STOP

Appendix B - Common Block and Array Storage

The labeled common blocks and major arrays used in CUERVO are defined in the following two sections. In most cases, sizing and dimensioning parameters, option flags, pointers and problem invariant data (*e.g.*, quadrature rules) are stored in named common blocks. The common block name should give some indication of the function of the data within the block. The large data arrays used by CUERVO are allocated by the dynamic memory manager within the main program and are contained within two vectors labeled A and CA; the CA vector is for character data. Integer pointers partition A and CA into specific arrays that are passed through subroutine parameter lists to other parts of the code. When these arrays are used within the various subroutines they have a descriptive name that is cataloged in a subsequent section. Comment lines within the source code also help to define the use of various arrays and vectors.

Common Blocks

The common blocks used in CUERVO are grouped below according to function.

File Data

```
COMMON /TAPES/  NIN,NOUT,NTP0,NTP1,NTP2,NTP3,NTP4,NTP5
COMMON /NTPDAT/ IFILES(8)
COMMON /FILDAT/ INFILE
```

<u>Variable</u>	<u>Description</u>
NIN,NOUT,NTP0..	Disk file names and ids
IFILES	Disk status for each file, 1=open, 0=closed
INFILE	Type of mesh input file, 1=EXODUS, 2=Other

Header Data

```
COMMON /HEADER/ PRBHED,MSHHED,CMMNT(10)
COMMON /RUNDAT/ CODNAM,VERSN,RDATE,RTIME,HRDWRE,SFTWRE,VERSNX
```

<u>Variable</u>	<u>Description</u>
-----------------	--------------------

PRBHED	Input title (CHARACTER*80)
MSHHED	Input title from mesh file (CHARACTER*80)
CMMNT	Input comment lines (CHARACTER*80)
CODNAM	Code name (CHARACTER*8)
VERSN	Code version number (CHARACTER*8)
RDATE	Run date (CHARACTER*8)
RTIME	Run time (CHARACTER*8)
HRDWRE	Computer hardware (CHARACTER*8)
SFTWRE	Computer operating system (CHARACTER*8)
VERSNX	Version number of EXODUS II used in CUERVO

Problem Data

COMMON /PRBDAT/ IGEOM,ITMDEP,IEXTFL,IFLUX,IPOST

<u>Variable</u>	<u>Description</u>
IGEOM	Problem geometry flag, 1=2D, 2=axisymmetric, 3=3D
ITMDEP	Time dependence flag, 0=steady, 1=transient
IEXTFL	External variable file flag, 0=omitted, 1=included
IFLUX	Flux computation flag, 0=omitted, 1=included
IPOST	Post-processing file flag, 0=omitted, 1=included

Sizing Data

COMMON /MATDAT/ NUMMAT
COMMON /MSHDAT/ NUMDIM,NUMEL,NUMNOD,NUMVAR,NUMDOF
COMMON /SZDAT/ MXELBK,MXNDEL,MXDFEL,MXATEL
COMMON /BLKDAT/ NUMBLK
COMMON /SLNDAT/ NUMSLN

<u>Variable</u>	<u>Description</u>
NUMMAT	Number of materials
NUMDIM	Number of coordinate dimensions
NUMEL	Number of elements
NUMNOD	Number of nodal points
NUMVAR	Number of variables per node

NUMDOF	Number of degrees of freedom (total)
MXELBK	Maximum number of elements per element block
MXNDEL	Maximum number of nodes per element
MXDFEL	Maximum number of degrees of freedom per element
MXATEL	Maximum number of element attributes per element
NUMBLK	Number of element blocks
NUMSLN	Number of solution blocks

Boundary Condition Data

```
COMMON /BCDAT/ NUMNS,LNSNL,LNSDF,NUMSS,LSEL,LSSDF,NUMNBC,NUMSBC,
*              MXNSNL,MXNSDF,MXSEL,MXSSDF
```

<u>Variable</u>	<u>Description</u>
NUMNS	Number of nodal point sets
LNSNL	Length of concatenated node set node list
LNSDF	Length of concatenated node set distribution factor list
NUMSS	Number of element side sets
LSEL	Length of concatenated side set element list
LSSDF	Length of concatenated side set distribution factor list
NUMNBC	Number of node boundary conditions
NUMSBC	Number of side boundary conditions
MXNSNL	Maximum number of nodes in any node set list
MXNSDF	Maximum number of distribution factors in any node set list
MXSEL	Maximum number of elements in any side set list
MXSSDF	Maximum number of distribution factors in any side set list

Element Data

```
COMMON /ELMDAT/ NNELM(20),NNCOR(20),NFACES(20),NNFACE(20,6),
*              NCFACE(20,6),NDFACE(20,6,8)
```

<u>Variable</u>	<u>Description</u>
NNELM	Number of nodes for each element type
NNCOR	Number of corner nodes for each element type
NFACES	Number of faces/edges for each element type

NNFACE	Number of nodes on each face/edge for each element type
NCFACE	Number of corner nodes on each face/edge for each element type
NDFACE	Local node numbers on each face/edge for each element type

Function and Constant Data

```
COMMON /FNCDAT/ NTIME,MXPTTF,NPTSTF,NVARF,MXPTVF,NPTSVF
COMMON /USRDAT/ NUSRR,NUSRI
```

<u>Variable</u>	<u>Description</u>
NTIME	Number of time functions
MXPTTF	Maximum number of points in any time function
NPTSTF	Number of time function points
NVARF	Number of variable functions
MXPTVF	Maximum number of points in any variable function
NPTSVF	Number of variable function points
NUSRR	Number of user defined (real) constants
NUSRI	Number of user defined (integer) constants

Flux Data

```
COMMON /FLXDAT/ NUMFLX,IDFLUX(20)
```

<u>Variable</u>	<u>Description</u>
NUMFLX	Number of side sets for integrated flux computations
IDFLUX	Side set ids for integrated flux computations coordinates

Input/Output Data

```
COMMON /IODAT/ IEDIT,IPRNTD,IPRNTS,NPRNTS
```


<u>Variable</u>	<u>Description</u>
IEDIT	Solution output editing flag, 0=print all elements, 1=print selected elements
IPRNTD	Print level for input data, 1=summary, 2=extended, 3=debug
IPRNTS	Print level for solution data, 0=none, 1=summary, 2=extended, 3=debug
NPRNTS	Print frequency for solution data

Solution Algorithm Data

```
COMMON /ALGDAT/ IALGOR,IAUTO,ILUMP,IPRDCT
COMMON /RSTDAT/ IRSTRT,NSTEPS,RDTIME
COMMON /PCGDAT/ NDOFND,MXCON,NNZERO
```

<u>Variable</u>	<u>Description</u>
IALGOR	Solution algorithm flag, steady - 1=Picard, 2=Newton, transient - 1=Euler, 2=trapezoid, 3=explicit
IAUTO	Automatic time step selection flag, 1=omitted, 2=included
ILUMP	Lumped capacitance matrix flag, 1=omitted, 2=included
IPRDCT	Explicit predictor flag, 1=omitted, 2=included
IRSTRT	Restart flag, 0=new solution, 1=restart from old solution
NSTEPS	Time step number at which to restart
RDTIME	Time at which to restart
NDOFND	Multiple degrees of freedom per node flag, 1=1 dof, 0=n dof
MXCON	Maximum number of nodes connected to any node
NNZERO	Number of nonzero entries in global matrix

Mesh Velocity

```
COMMON /VELDAT/ IEULRN
```

<u>Variable</u>	<u>Description</u>
IEULRN	Eulerian velocity flag, 0=omitted, 1=included

Norm Data

```
COMMON /MAXVAR/ PMAX,IFNDMX
COMMON /NORMS/  ANRMP,BNRMP,PDIFF
```

<u>Variable</u>	<u>Description</u>
PMAX	Maximum dependent variable in the current solution vector
IFNDMX	Flag for computation/input of norm value
ANRMP	Norm on dependent variable change between iterations/time steps
BNRMP	Norm on dependent variable integration error
PDIFF	Maximum dependent variable change between steps or iterations

Postprocessing Data

```
COMMON /EXODAT/ NVARNP,NVAREL,NVARGL
COMMON /EXONAM/ NAMECO(3),NAMEEL(20),NAMENV(10),NAMEEV(5),NAMEGV(15)
COMMON /EXOUT/  IEXOPT,NWFREQ,DELTEX,NTIMEX,TIMEX(50)
```

<u>Variable</u>	<u>Description</u>
NVARNP	Number of nodal variables to be written on EXODUS II file
NVAREL	Number of element variables to be written on EXODUS II file
NVARGL	Number of global variables to be written on EXODUS II file
NAMECO	Names for coordintes (CHARACTER*8)
NAMEEL	Names for elements (CHARACTER*8)
NAMENV	Names for nodal variables (CHARACTER*8)
NAMEEV	Names for element variables (CHARACTER*8)
NAMEGV	Names for global variables (CHARACTER*8)
IEXOPT	EXODUS II output option flag, 1=write output at every NWFREQ steps, 2=write output at every DELTEX time interval, 3=write output at every time specified in TIMEX
NWFREQ	Frequency for writing to EXODUS II output file
DELTEX	Time step for writing EXODUS II output file
NTIMEX	Number of output times stored in TIMEX
TIMEX	Specified times for writing EXODUS II output file

Shape Function Data (2D)

```

COMMON /TRI3/   TR3(3,7),TR3G(3,2,7),TR3F(3,3),TR3FG(3,2,3),TR3WT(7)
COMMON /TRI6/   TR6(6,7),TR6G(6,2,7),TR6F(6,3),TR6FG(6,2,3),TR6WT(7)
COMMON /QUAD4/  QD4(4,9),QD4G(4,2,9),QD4F(4,4),QD4FG(4,2,4),QD4WT(9)
COMMON /QUAD8/  QD8(8,9),QD8G(8,2,9),QD8F(8,4),QD8FG(8,2,4),QD8WT(9)
COMMON /EDGE2/  ED2(2,3),ED2G(2,3),ED2WT(3)
COMMON /EDGE3/  ED3(3,3),ED3G(3,3),ED3WT(3)

```

<u>Variable</u>	<u>Description</u>
TR3,TR6 QD4,QD8	Two-dimensional, triangular and quadrilateral, element shape functions evaluated at the quadrature points. The first index is the shape function (node) number; the second index is the quadrature point number.
TR3G,TR6G QD4G,QD8G	Two-dimensional, triangular and quadrilateral, element shape function derivatives evaluated at the quadrature points. The first index is the shape function (node) number, the second index is the local spatial direction for the derivative and the third index is the quadrature point number.
TR3F,TR6F QD4F,QD8F	Two-dimensional, triangular and quadrilateral, element shape functions evaluated at the flux evaluation points. The first index is the shape function (node) number; the second index is the flux evaluation point number.
TR3FG,TR6FG QD4FG,QD8FG	Two-dimensional, triangular and quadrilateral, element shape function derivatives evaluated at the flux evaluation points. The first index is the shape function (node) number, the second index is the local spatial direction for the derivative and the third index is the flux evaluation point number.
TR3WT,TR6WT QD4WT,QD8WT ED2,ED3	Quadrature weights for each element at each integration point. One-dimensional, element edge shape functions evaluated at the quadrature points. The first index is the shape function (node) number; the second index is the quadrature point number.
ED2G,ED3G	One-dimensional, element edge shape function derivatives evaluated at the quadrature points. The first index is the shape function (node) number and the second index is the quadrature point number.
ED2WT,ED3WT	Quadrature weights for each element edge at each integration point.

Shape Function Data (3D)

```

COMMON /TETR4/  TT4(4,5),TT4G(4,3,5),TT4F(4,4),TT4FG(4,3,4),TT4WT(5)
COMMON /TETR10/ TT10(10,5),TT10G(10,3,5),TT10F(10,4),TT10FG(10,3,4),
*              TT10WT(5)
COMMON /WEDG6/  WD6(6,21),WD6G(6,3,21),WD6F(6,6),WD6FG(6,3,6),WD6WT(21)
COMMON /WEDG15/ WD15(15,21),WD15G(15,3,21),WD15F(15,6),WD15FG(15,3,6),
*              WD15WT(21)
COMMON /HEX8/   HX8(8,27),HX8G(8,3,27),HX8F(8,8),HX8FG(8,3,8),HX8WT(27)
COMMON /HEX20/  HX20(20,27),HX20G(20,3,27),HX20F(20,8),HX20FG(20,3,8),
*              HX20WT(27)
COMMON /FACE3/  FC3(3,7),FC3G(3,2,7),FC3WT(7)
COMMON /FACE6/  FC6(6,7),FC6G(6,2,7),FC6WT(7)
COMMON /FACE4/  FC4(4,9),FC4G(4,2,9),FC4WT(9)
COMMON /FACE8/  FC8(8,9),FC8G(8,2,9),FC8WT(9)

```

<u>Variable</u>	<u>Description</u>
TT4,TT10 WD6,WD15 HX8,HX20	Three-dimensional, tetrahedral, wedge and hexahedral, element shape functions evaluated at the quadrature points. The first index is the shape function (node) number; the second index is the quadrature point number.
TT4G,TT10G WD6G,WD15G HX8G,HX20G	Three-dimensional, tetrahedral, wedge and hexahedral, element shape function derivatives evaluated at the quadrature points. The first index is the shape function (node) number, the second index is the local spatial direction for the derivative and the third index is the quadrature point number.
TT4F,TT10F WD6F,WD15F HX8F,HX20F	Three-dimensional, tetrahedral, wedge and hexahedral, element shape functions evaluated at the flux evaluation points. The first index is the shape function (node) number; the second index is the flux evaluation point number.
TT4FG,TT10FG WD6FG,WD15FG HX8FG,HX20FG	Three-dimensional, tetrahedral, wedge and hexahedral, element shape function derivatives evaluated at the flux evaluation points. The first index is the shape function (node) number, the second index is the local spatial direction for the derivative and the third index is the flux evaluation point number.
TT4WT,TT10W WD6WT,WD15WT HX8WT,HX20WT	Quadrature weights for each element at each integration point.
FC3,FC6 FC4,FC8	Two-dimensional, triangular and quadrilateral element face shape functions evaluated at the quadrature points. The

	first index is the shape function (node) number; the second index is the quadrature point number.
FC3G,FC6G FC4G,FC8G	Two-dimensional, triangular and quadrilateral element face shape function derivatives evaluated at the quadrature points. The first index is the shape function (node) number, the second index is the local (surface) spatial direction for the derivative and the third index is the quadrature point number.
FC3WT,FC6WT FC4WT,FC8WT	Quadrature weights for each element face at each integration point.

Quadrature Data

```

COMMON /INTDAT/ NIRULE(20),NFRULE(20)
COMMON /GAUSS1/ GSPT1(3,1,3),GSWT1(3,3),NGSPT1(3)
COMMON /GAUSS2/ GSPT2(9,2,3),GSWT2(9,3),NGSPT2(3)
COMMON /GAUSS3/ GSPT3(27,3,5),GSWT3(27,5),NGSPT3(5)
COMMON /GAUSS4/ GSPT4(7,3,4),GSWT4(7,4),NGSPT4(4)
COMMON /GAUSS5/ GSPT5(5,4,3),GSWT5(5,3),NGSPT5(3)
COMMON /GAUSS6/ GSPT6(21,4,4),GSWT6(21,4),NGSPT6(4)
COMMON /CENTER/ STRPT(20,3)
COMMON /TRNSFM/ EXTTR(3,3),EXTQD(4,4),EXTTT(4,4),
*                EXTWD(6,6),EXTHX(8,8)

```

<u>Variable</u>	<u>Description</u>
NIRULE	Integer indicating integration rule for each element type
NFRULE	Integer indicating flux computation rule for each element type
GSPTn	Quadrature points for each quadrature rule and each element type. The first index is the number of the quadrature point, the second index is the number of the local coordinate direction and the third index is the integration rule. The integer n in the array name separates the data according to element geometry. Arrays with n=1 contain quadrature data for one-dimensional, line elements, n=2 is for two-dimensional quadrilateral regions, n=3 is for three-dimensional hexahedrons, n=4 is for two-dimensional triangular areas, n=5 is for three-dimensional tetrahedrons and n=6 is for three-dimensional wedges.
GSWTn	Quadrature weights for each integration point and each quadrature rule. The first index specifies the number of the quadrature point and the second index is the integration rule.
NGSPTn	Number of quadrature points for each integration rule.

STRPT	Local element coordinates for element centroid.
EXTmmm	Flux point to node point linear extrapolation operators for each element type.

Free Field Input Data

```
COMMON /INDATR/ RDATA(50)
COMMON /INDATI/ IDATA(50)
COMMON /INDATC/ CDATA(50)
COMMON /INDATK/ KDATA(50)
```

<u>Variable</u>	<u>Description</u>
RDATA	Real data from the free field reader
IDATA	Integer data from the free field reader
CDATA	Character data from the free field reader (CHARACTER*20)
KDATA	Data type flags from the free field reader

Array Storage

The major arrays used in CUERVO are grouped below according to function.

Mesh Data

```
DIMENSION IDBLK(NUMBLK)
DIMENSION X(NUMNOD), Y(NUMNOD), Z(NUMNOD)
DIMENSION ICON(MXNDEL, NUMEL)
DIMENSION LSTKND(NUMEL), LSTOUT(NUMEL)
DIMENSION ELATRB(MXATEL, NUMEL)
DIMENSION DATBLK(10, NUMBLK), IPTBLK(20, NUMBLK)
DIMENSION ICONBK(MXNDEL*MXELBK)
DIMENSION BLKMAT(NUMBLK)
```

<u>Array</u>	<u>Description</u>
IDBLK	Block id numbers
X, Y, Z	Nodal coordinates
ICON	Element connectivity
LSTKND	Encoded list of element type and material (1000*mat id + element type)
LSTOUT	Output flags for elements, 0=no print, 1=print
ELATRB	Element attributes
DATBLK	Element block data DATBLK(1, NUMBLK) - UX velocity for block DATBLK(2, NUMBLK) - UY velocity for block DATBLK(3, NUMBLK) - UZ velocity for block DATBLK(4, NUMBLK) - Unused DATBLK(5, NUMBLK) - Unused DATBLK(6, NUMBLK) - Unused DATBLK(7, NUMBLK) - Unused DATBLK(8, NUMBLK) - Unused DATBLK(9, NUMBLK) - Unused DATBLK(10, NUMBLK) - Unused
IPTBLK	Element block pointers and flags IPTBLK(1, NUMBLK) - Number of elements in block IPTBLK(2, NUMBLK) - Element type IPTBLK(3, NUMBLK) - Number of element attributes IPTBLK(4, NUMBLK) - Number of integration points IPTBLK(5, NUMBLK) - Material number IPTBLK(6, NUMBLK) - First element in block

	IPTBLK(7,NUMBLK) - Last element in block
	IPTBLK(8,NUMBLK) - Unused
	IPTBLK(9,NUMBLK) - Unused
	IPTBLK(10,NUMBLK) - Unused
	IPTBLK(11,NUMBLK) - Unused
	IPTBLK(12,NUMBLK) - Unused
	IPTBLK(13,NUMBLK) - Unused
	IPTBLK(14,NUMBLK) - Number of flux points
	IPTBLK(15,NUMBLK) - Unused
	IPTBLK(16,NUMBLK) - Velocity type (0=None, 1=Eulerian)
	IPTBLK(17,NUMBLK) - Velocity user subroutine flag
	IPTBLK(18,NUMBLK) - Function id for UX
	IPTBLK(19,NUMBLK) - Function id for UY
	IPTBLK(20,NUMBLK) - Function id for UZ
ICONBK	Element connectivity for a block
BLKMAT	Material name for element block (CHARACTER*20)

Boundary Condition Data

```

DIMENSION IDNS(NUMNS), NNNS(NUMNS), NDFNS(NUMNS)
DIMENSION IPNNS(NUMNS), IPDFNS(NUMNS)
DIMENSION LSNNS(LNSNL), DFNS(LNSDF)
DIMENSION IDSS(NUMSS), NESS(NUMSS), NDFSS(NUMSS)
DIMENSION IPESS(NUMSS), IPDFSS(NUMSS)
DIMENSION LSESS(LSSEL), LSSSS(LSSEL), DFSS(LSSDF)
DIMENSION DATNS(2,NUMNS), KPTNS(5,NUMNS)
DIMENSION DATSS(6,NUMSS), KPTSS(10,NUMSS), KPTELM(NUMEL)

```

<u>Array</u>	<u>Description</u>
IDNS	Node set ids
NNNS	Node set node counts
NDFNS	Node set distribution factor counts
IPNNS	Node set node pointers
IPDFNS	Node set distribution factor pointers
LSNNS	Node set node list
DFNS	Node set distribution factor list
IDSS	Side set ids
NESS	Side set element counts
NDFSS	Side set distribution factor counts
IPESS	Side set element pointers

IPDFSS	Side set distribution factor pointers
LSESS	Side set element list
LSSSS	Side set side list
DFSS	Side set distribution factor list
DATNS	Node set boundary data DATNS(1,NUMNS) - Value of boundary condition DATNS(2,NUMNS) - Multiplying factor for boundary value
KPTNS	Node set boundary pointers KPTNS(1,NUMNS) - Node set id KPTNS(2,NUMNS) - Type of bc, 1=dirichlet KPTNS(3,NUMNS) - User subroutine flag KPTNS(4,NUMNS) - Function id for bc, +id=variable function, -id=time function KPTNS(5,NUMNS) - Constant bc flag
DATSS	Side set boundary data DATSS(1,NUMSS) - Value 1 of boundary condition DATSS(2,NUMSS) - Value 2 of boundary condition DATSS(3,NUMSS) - Unused DATSS(4,NUMSS) - Unused DATSS(5,NUMSS) - Multiplying factor for value 1 DATSS(6,NUMSS) - Multiplying factor for value 2
KPTSS	Side set boundary pointers KPTSS(1,NUMSS) - Side set id KPTSS(2,NUMSS) - Type of bc, 1=neumann, 2=mixed KPTSS(3,NUMSS) - User subroutine flag for value 1 KPTSS(4,NUMSS) - User subroutine flag for value 2 KPTSS(5,NUMSS) - Function id for value 1, +id=variable function, -id=time function KPTSS(6,NUMSS) - Function id for value 2, +id=variable function, -id=time function KPTSS(7,NUMSS) - Constant bc flag, value 1 KPTSS(8,NUMSS) - Constant bc flag, value 2 KPTSS(9,NUMSS) - Unused KPTSS(10,NUMSS) - Unused
KPTELM	Encoded side set list for each element

Material Data

```

DIMENSION DATMAT(21,NUMMAT)
DIMENSION PTMAT(21,NUMMAT)
DIMENSION IPTMAT(21,NUMMAT)
DIMENSION MATNAM(NUMMAT)

```

<u>Array</u>	<u>Description</u>
DATMAT	<p>Material property values</p> <p>DATMAT(1,NUMMAT) - β coefficient</p> <p>DATMAT(2,NUMMAT) - Unused</p> <p>DATMAT(3,NUMMAT) - Conductivity (xx component)</p> <p>DATMAT(4,NUMMAT) - Conductivity (yy component)</p> <p>DATMAT(5,NUMMAT) - Conductivity (zz component)</p> <p>DATMAT(6,NUMMAT) - Volume source</p> <p>DATMAT(7,NUMMAT) - Unused</p> <p>DATMAT(8,NUMMAT) - Conductivity tensor orientation ($x\hat{x}$)</p> <p>DATMAT(9,NUMMAT) - Conductivity tensor orientation ($y\hat{x}$)</p> <p>DATMAT(10,NUMMAT) - Conductivity tensor orientation ($z\hat{x}$)</p> <p>DATMAT(11,NUMMAT) - Conductivity tensor orientation ($x\hat{y}$)</p> <p>DATMAT(12,NUMMAT) - Conductivity tensor orientation ($y\hat{y}$)</p> <p>DATMAT(13,NUMMAT) - Conductivity tensor orientation ($z\hat{y}$)</p> <p>DATMAT(14,NUMMAT) - Unused</p> <p>DATMAT(15,NUMMAT) - Unused</p> <p>DATMAT(16,NUMMAT) - Unused</p> <p>DATMAT(17,NUMMAT) - Unused</p> <p>DATMAT(18,NUMMAT) - Unused</p> <p>DATMAT(19,NUMMAT) - Unused</p> <p>DATMAT(20,NUMMAT) - Initial value, dependent variable</p> <p>DATMAT(21,NUMMAT) - Unused</p>
PTMAT	<p>Material property pointers (CHARACTER*1)</p> <p>PTMAT(1,NUMMAT) - Subroutine and function flags for property 'V'=variable function, 'T'=time function 'C'=constant value, 'U'=user subroutine</p> <p>PTMAT(16,NUMMAT) - Subroutine and function flag for property</p> <p>PTMAT(17,NUMMAT) - Unused</p> <p>PTMAT(18,NUMMAT) - Unused</p> <p>PTMAT(19,NUMMAT) - Unused</p> <p>PTMAT(20,NUMMAT) - Subroutine flag for ic</p> <p>PTMAT(21,NUMMAT) - Material model, 'I'= isotropic, 'O'=orthotropic</p>
IPTMAT	<p>Material property function flags</p> <p>IPTMAT(1,NUMMAT) - Function id for property</p> <p>IPTMAT(16,NUMMAT) - Function id for property</p> <p>IPTMAT(17,NUMMAT) - Unused</p> <p>IPTMAT(18,NUMMAT) - Unused</p>

IPTMAT(19,NUMMAT) - Unused
 IPTMAT(20,NUMMAT) - Unused
 IPTMAT(21,NUMMAT) - Unused
 MATNAM Material property names (CHARACTER*20)

Function and Constant Data

DIMENSION DATTFN(2,NPTSTF), IPTTFN(3,NTIME), TFNVAL(NTIME)
 DIMENSION DATVFN(2,NPTSVF), IPTVFN(3,NVARF)
 DIMENSION RCONST(NUSRR), ICONST(NUSRI)

<u>Array</u>	<u>Description</u>
DATTFN	Time function data DATTFN(1,NPTSTF) - Time DATTFN(2,NPTSTF) - Function value
IPTTFN	Time function pointers IPTTFN(1,NTIME) - Function id IPTTFN(2,NTIME) - Number of points in time function IPTTFN(3,NTIME) - Pointer to first function point in DATTFN
TFNVAL	Time function values at each time step
DATVFN	Variable function data DATVFN(1,NPTSVF) - Dependent variable value (temperature) DATVFN(2,NPTSVF) - Function value
IPTVFN	Variable function pointers IPTVFN(1,NVARF) - Function id IPTVFN(2,NVARF) - Number of points in time function IPTVFN(3,NVARF) - Pointer to first function point in DATVFN
RCONST	User defined constants (real)
ICONST	User defined constants (integer)

Solution Control Data

DIMENSION DATSLN(20,NUMSLN)
 DIMENSION IPTSLN(20,NUMSLN)

<u>Array</u>	<u>Description</u>
--------------	--------------------

DATSLN

Solution control data

- DATSLN(1,NUMSLN) - Initial solution time
- DATSLN(2,NUMSLN) - Final solution time
- DATSLN(3,NUMSLN) - Initial time step
- DATSLN(4,NUMSLN) - Minimum time step
- DATSLN(5,NUMSLN) - Maximum time step
- DATSLN(6,NUMSLN) - Maximum dependent variable change
per time step
- DATSLN(7,NUMSLN) - Convergence tolerance on dependent variable
- DATSLN(8,NUMSLN) - Integration tolerance
- DATSLN(9,NUMSLN) - Relaxation factor
- DATSLN(10,NUMSLN) - Unused
- DATSLN(11,NUMSLN) - Unused
- DATSLN(12,NUMSLN) - Unused
- DATSLN(13,NUMSLN) - Explicit time step scale factor
- DATSLN(14,NUMSLN) - L_2 norm for iterative solver
- DATSLN(15,NUMSLN) - Residual norm for iterative solver
- DATSLN(16,NUMSLN) - Absolute dependent variable limit
- DATSLN(17,NUMSLN) - Dependent variable value for norm
- DATSLN(18,NUMSLN) - Unused
- DATSLN(19,NUMSLN) - Unused
- DATSLN(20,NUMSLN) - Unused

IPTSLN

Solution control pointers and flags

- IPTSLN(1,NUMSLN) - Time dependence flag,
1=steady, 2=transient
- IPTSLN(2,NUMSLN) - Iterative method flag,
1=Picard, 2=Newton
- IPTSLN(3,NUMSLN) - Maximum number of iterations
- IPTSLN(4,NUMSLN) - Time integration method flag,
1=Euler, 2=trapezoid, 3=explicit
- IPTSLN(5,NUMSLN) - Capacitance matrix flag,
1=consistent, 2=lumped
- IPTSLN(6,NUMSLN) - Time step option flag, 1=fixed step,
2=autostep
- IPTSLN(7,NUMSLN) - Maximum number of time steps
- IPTSLN(8,NUMSLN) - Predictor flag, 1=no predictor ,
2=predictor
- IPTSLN(9,NUMSLN) - Unused
- IPTSLN(10,NUMSLN) - Unused
- IPTSLN(11,NUMSLN) - Unused
- IPTSLN(12,NUMSLN) - Unused
- IPTSLN(13,NUMSLN) - Unused
- IPTSLN(14,NUMSLN) - Type of matrix solver, 0=CG, 1=GMRES,
2=CGS, 3=QMR, 5=Direct

IPTSLN(15,NUMSLN) - Krylov subspace dimension
 IPTSLN(16,NUMSLN) - Type of preconditioner, 0=None,
 1=Jacobi, 3=polynomial, 4=ILU, 9=IC
 IPTSLN(17,NUMSLN) - Order of polynomial preconditioner
 IPTSLN(18,NUMSLN) - Maximum number of solver iterations
 IPTSLN(19,NUMSLN) - Print flag, 1=summary, 2=detailed, 3=debug
 IPTSLN(20,NUMSLN) - Print frequency

Presolution and Solution Data

```

DIMENSION NELPND(NUMNOD), NDLST(NUMNOD), NELST(NUMEL)
DIMENSION NDFNOD(NUMNOD), NFIRST(NUMNOD), NDFELM(NUMEL)
DIMENSION NCHECK(NUMNOD)
DIMENSION NK(MXDFEL), INK(MXDFEL)
DIMENSION ITMP(MXCON), IKTMP(MXCON,NUMDOF), IJK(MXCON*NUMNOD+1)
DIMENSION TN(NUMNOD), TPNP1(NUMNOD), TDOT(NUMNOD), SCRATCH(NUMNOD)
DIMENSION BNDVAL(MXNNPS)
DIMENSION BIGK(NNZERO), BIGF(NUMNOD)
DIMENSION VELX(NUMNOD), VELY(NUMNOD), VELZ(NUMNOD)
DIMENSION QX(NUMNOD), QY(NUMNOD), QZ(NUMNOD), NODFLX(NUMNOD)
  
```

<u>Array</u>	<u>Description</u>
NELPND	Number of elements per node
NDLST	Temporary node list (presolution)
NELST	Temporary element list (presolution)
NDFNOD	Number of degrees of freedom per node
NFIRST	Number of first degree of freedom at each node
NDFELM	Number of degrees of freedom per element
NCHECK	Scratch check array
NK	Element pointers for sparse matrix storage
INK	Element pointers for sparse matrix storage
ITMP	Temporary pointers for sparse matrix storage
IKTMP	Temporary pointers for sparse matrix storage
IJK	Pointers for sparse matrix storage
TN	Solution vector
TPNP1	Predicted solution vector
TDOT	Rate vector
SCRATCH	Scratch solution vector
BNDVAL	Boundary conditions for nodal points
BIGK	Global matrix (sparse matrix format)

BIGF	Global force vector
VELX	Nodal point velocities, x component
VELY	Nodal point velocities, y component
VELZ	Nodal point velocities, z component
QX	Nodal point flux component
QY	Nodal point flux component
QZ	Nodal point flux component
NODFLX	Number of fluxes per node

Postprocessing Data

DIMENSION GBLV(NVARGL)
 DIMENSION IPPNOD(10), IPPELM(5), IPPGLB(15)

<u>Array</u>	<u>Description</u>
GBLV	Global variable vector
IPPNOD	Output flags for nodal variables (0=omitted, 1=output) IPPNOD(1) - Flag for dependent variable IPPNOD(2) - Flag for dependent variable rate IPPNOD(3) - Flag for flux, x component IPPNOD(4) - Flag for flux, y component IPPNOD(5) - Flag for flux, z component IPPNOD(6) - Flag for potential function (2D) IPPNOD(7) - Unused IPPNOD(8) - Unused IPPNOD(9) - Unused IPPNOD(10) - Unused
IPPELM	Output flags for element variables (0=omitted, 1=output) IPPELM(1,NUMBLK) - Unused IPPELM(2,NUMBLK) - Unused IPPELM(2,NUMBLK) - Unused IPPELM(2,NUMBLK) - Unused IPPELM(2,NUMBLK) - Unused
IPPGLB	Output flags for global variables (0=omitted, 1=output) IPPGLB(1) - Flag for timestep IPPGLB(2) - Flag for CG iterations IPPGLB(3) - Unused IPPGLB(4) - Unused IPPGLB(5) - Unused IPPGLB(6) - Flag for integrated flux surface 1

IPPGLB(7) - Flag for integrated flux surface 2

IPPGLB(8) - Flag for integrated flux surface 3

.

.

.

IPPGLB(13) - Flag for integrated flux surface 8

IPPGLB(14) - Flag for integrated flux surface 9

IPPGLB(15) - Flag for integrated flux surface 10

Distribution:

G. F. Carey
University of Texas
Department of Aerospace Engineering
and Engineering Mechanics
Austin, Texas 78712

R. I. Tanner
University of Sydney
Department of Mechanical Engineering
Sydney, New South Wales 2006
Australia

P. J. Roache
Ecodynamics Research Associates, Inc.
P.O. Box 9229
Albuquerque, New Mexico 87119

J. N. Reddy
Texas A & M University
Department of Mechanical Engineering
College Station, Texas 77843

M. S. Ingber
Department of Mechanical Engineering
University of New Mexico
Albuquerque, New Mexico 87131

Sandia Internal:

MS0841	9100	P. J. Hommert
MS0828	9102	R. D. Skocypec
MS0833	9103	J. H. Biffle
Route to: 9112, 9113, 9116		
MS0828	9104	E. D. Gorham
Route to: 9111, 9114, 9115		
MS0826	9111	W. Hermina
MS0826	9111	D. K. Gartling (10)
MS0826	9111	R. C. Givler
MS0826	9111	M. W. Glass
MS0826	9111	C. E. Hickox
MS0826	9111	S. N. Kempka
MS0826	9111	P. A. Sackinger
MS0826	9111	P. R. Schunk
MS0826	9111	J. A. Schutt
MS0826	9111	M. B. Sirman (10)
MS0834	9112	A. C. Ratzel
MS0834	9112	P. L. Hopkins
MS0834	9112	M. J. Martinez
MS0835	9113	R. E. Hogan
MS0443	9117	H. S. Morgan
Route to: 9118		
MS0899	4414	Technical Library (5)
MS0100	7613-2	Document Processing for DOE/OSTI (2)
MS0619	12615	Print Media
MS9018	8523-2	Central Technical Files