LA-10613-MS-REV. 1

UC-32
Issued: January 1990

CAVEAT: A Computer Code for Fluid Dynamics Problems with Large Distortion and Internal Slip

(Revised edition, 1990)

Frank L. Addessio
John R. Baumgardner
John K. Dukowicz
Norman L. Johnson
Bryan A. Kashiwa
Rick M. Rauenzahn
Charles Zemach

Los Alamos

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

PREFACE

When the first report on CAVEAT was written in 1985 [Addessio et al., 1985], the CAVEAT code was being developed and regularly being modified. In contrast, presently the basic hydrodynamics in CAVEAT is rarely modified, and the majority of the work is in adding additional physics or non-standard capabilities to the basic hydrodynamics. For this reason, we decided now is an appropriate time to document and release the core of the CAVEAT code, the hydrodynamics, for general use. (The source code is available through the Argonne Code Center, Argonne National Laboratory, Argonne, IL 60439.) Extensions to the code will be documented and released separately.

CAVEAT has a long history of collective effort and would not exist in its current form without the contributions of a wide number of people. The following highlights the major developers of CAVEAT, but recognition is also given to the many users that played an essential role in the development.

The code that would eventually be named CAVEAT was started by Hans Ruppel and Francis Harlow in about 1981. Their goal was to investigate various numerical methods that could calculate multimaterial dynamics with strong contortions and large slip between and within materials in both two and three dimensions. At the same time they wished to combine the best features of Lagrangian and Eulerian methods, thereby avoiding, if possible, the classical disadvantages of each. At an early stage they settled on the use of cell-centered velocities (following the much earlier Particle-in-cell method), a Lagrangian treatment for motion normal to material interfaces, and rezone in the tangential direction along material interface, and an arbitrarily specified rezoning-remapping within each material.

The earliest versions of CAVEAT used simple interpolation procedure for the calculation of cell-edge velocities and pressure; despite the relatively poor coupling between pressure and velocity fields, the results were highly encouraging. Significant improvement was later achieved switching to a Godunov method with an approximate Riemann solver that was usable with any reasonable equation of state.

Daniel Carrol* worked in parallel during this early development, implementing the data and logic structure that was to prove essential for the efficient utilization of fluid-dynamic algorithms. Carrol extended the initial research code that Ruppel and Harlow had developed into a more production-oriented code and commenced an elaborate set of tests, of which an especially useful series examined the transport of a shock around and through a dense, compressible cube. Carrol also added major physics packages, including more general equations of state and a high explosive treatment, and a mesh generation scheme for complex geometries. About this time Daniel Butler applied the code to its first significant programmatic application. Soon the use of CAVEAT expanded to include a variety of additional extensions and applications, many of them accomplished by the authors of the original CAVEAT report.

The original report on CAVEAT documented version 5C. Concurrent with the development of the two-dimensional (2D) version, a three-dimensional (3D) version was developed by John Baumgardner. Shortly thereafter a new 2D version of the code was created from the 3D version because of the significant improvement in vectorization. This became a new version 1 of CAVEAT. Changes in the code from version 1 to the version documented in this report, version 8, became directed more towards extending the capabilities and directed less towards modifying the basic hydrodynamics. The last revision of the code, from version 7 to 8, also represents a substantial simplification of the coding of the basic algorithms in anticipation of its broader use.

The following list provides the contributors to the various sections of this report. In addition to those listed below, Frank Harlow and John Hopson continue to maintain considerable association with CAVEAT, both regarding its evolutionary trends and its currently broad spectrum of applications. Concerning the later, the contributions of Donald Sandoval, Bart Daly, and Martin Torrey have been enormous value to the CAVEAT development.

Frank L. Addessio	4.2, 4.5
John R. Baumgardner	3.2.2, Introduction to 4, 4.3.2
John K. Dukowicz	2, 3.1, 3.2.1, 4.4.3, 4.4.4, D
Norman L. Johnson	1, 4.1.3a, 4.1.3d, 4.2.5, 4.4.1, 4.4.2, 5, 7, A, B
Bryan A. Kashiwa	3.2.4, 3.2.5, 4.3.1, 4.4.5, C
Rick M. Rauenzahn	3.2.3, 4.1.4, 4.3.3
Charles Zemach	4.1.1, 4.1.2, 4.1.3b, 4.1.3c, 6

We thank Adrienne T. Rosen for her dedicated typing and producing many of the figures. The funding for the development of CAVEAT is from U. S. Department of Energy.

Norman L. Johnson
January 1990

---

*Currently at Sandia National Laboratory, Albuquerque, NM.

TABLE OF CONTENTS

NOTATION

The notation used in this report is given in this section. A brief definition follows the symbol. A more detailed description can be found in the section where the symbol is first used, identified by § plus the section number. The units, if any, are given in l=length, t=time, m=mass, T=temperature.

Variables

A	area	l^2	§2.1
a	sound speed	l/t	§2.2
A,B,C,...	constants in equations of state	-	§2.2
A_S	strong shock parameter	-	§3.2.1a
E	mass specific total energy	l^2/t^2	§2.1
e	mass specific internal energy	l^2/t^2	§2.1
F	body force per unit mass	l/t^2	§2.1
h	heat flux vector	m/l·t	§2.1
i, **j**	unit vector along mesh directions 1 and 2		§3.2.4a
J	Jacobian	l^2	§3.2.2a
C	arc length along a line	l	§3.1.2a
L_L	Lagrangian cell boundary in one dimension	l	§3.1.1a
m	cell mass	m	§3.2.1a
n, (n_1, n_2, n_3)	outward unit vector, normal to surface	-	§2.1
Π	deviatoric stress tensor	$m/l \cdot t^2$	§2.1
Q	energy-release rate per volume	$m/l^2 \cdot t$	§2.1
Q	any extensive conserved quantity	-	§2.1
q	any spatial density such as ρ, ρ**u**, or ρE	-	§2.1
R	pseudoradius	none or l	§3.1.1a
r	radial coordinate in cylindrical geometry	l	§3.1a,§4.1.1
r, (x,y,z), (x_1, x_2, x_3)	position vector	l	§2.1
S	surface	l^2	§2.1
T	temperature	T	§2.1
t	time	t	§2.1
t	tangent unit vector	-	§3.2.4a

U	specified material velocity	l/t	§3.2.4
u; (u_1, u_2, u_3)	material velocity	l/t	§2.1
V	volume	l^3	§2.1
w	weight function	-	§3.2.2
w	normal component of velocity of a cell face	l/t	§3.2.1a
α	limiting coefficient for gradients		§3.1.2b
β	unit vector in the radial direction	l or none	§3.1.1a
ξ_i (i=1,2,3)	physical coordinates for a specified coordinate system, in three dimensions	-	§4.1.1
Δt	time step	t	§3.1.1b
(η,ξ)	natural coordinates in mesh index space	-	§3.2.2a
ϕ	angle in spherical coordinates	-	§4.1.1
γ	ratio of the specific heats	-	§2.2
λ	arc length along a line	l	§3.1.1
θ	angle in cylindrical coordinates	-	§4.1.1
ρ	mass density	m/l^3	§2.1

Subscripts

a	associated with an arbitrary control volume	§2.1
H	quantity on the Hugoniot	§2.2
i,j,k,l	indices referring to cell in the mesh	§2.1, §4.1.3b
L	associated with the Lagrangian control volume	§2.1
L	associated with the "left" side of face	§3.2.4a
n	component of a vector normal to interface	§3.2.4a
R	associated with the "right" side of face	§3.2.4a
R	quantity relative to the Lagrangian frame	§2.1
S	associated with a shock	§2.2
t	component of a vector tangent to interface	§3.2.4a
α	index over all faces of a cell	§3.1.1a
0	quantity at a reference state	§2.2

Superscripts

*	Riemann quantities as cell surfaces	
n	index for time discretization ($t = n \Delta t$)	§3.1.1b
n	intermediate step in an iterating method	§3.2.2b
L	variable at the end of the Lagrangian phase	§3.2.3, §4.3.3

Special Symbols

overline	cell-averaged quantities	§3.1.1b
< >	area average	§3.1.1a

Math Symbols

∇	vector differential operator or del operator	l^{-1}	§2.1
∂	partial derivative		§2.1
d	total derivative		§2.1
·	scalar product of two vectors		

Symbols Used in Description of Code

I, J, K	index to cell, face, or vertex values; also for general reference to an array value	§4.1.3b
IB	index to boundary conditions	§4.1.3c
IBLK	index to blocks	§4.1.2
IMAT	index to material number (1-30)	§4.4.3
IP	index to a part in a block, referenced linearly	§4.1.2
IPRT	index to part in a block along the 1-direction, referenced by pairs of numbers (IPRT, JPRT)	§4.1.2
JPRT	index to part in a block along the 2-direction, referenced by pairs of numbers	§4.1.2
KPRT	taken to be IPRT if M=1, JPRT if M=2	§4.1.2
M	mesh direction (1 or 2) for vectors	§4.1.3b

CAVEAT: A Computer Code for Fluid Dynamics Problems with Large Distortion and Internal Slip †
(revised edition, 1990)

by

Frank L. Addessio, John R. Baumgardner, John K. Dukowicz,
Norman L. Johnson, Bryan A. Kashiwa,
Rick M. Rauenzahn, Charles Zemach

A B S T R A C T

This report is a description of the two-dimensional version of CAVEAT, a computer code which solves numerically the equations of transient, multimaterial, compressible fluid dynamics. CAVEAT is written to treat a wide variety of problems. It has the ability, for example, to describe material interfaces and the large slip along interfaces, to describe complex geometries without sacrificing vector processing, and to apply tabular equations of state (SESAME library). Its numerical methods were chosen to minimize numerical diffusion, achieve a high degree of vectorization, and facilitate extension to three dimensions.

CAVEAT uses an explicit time-marching, conservative finite-volume numerical technique in which all state variables, including velocity, are cell centered; values at vertices and cell faces are derived. The technique is a variation of the Godunov method that uses an approximate Riemann solver and accommodates arbitrary equations of state. Spatial differencing may either be first order (constant across the cell) or second order (linear variation across the cell) with a choice of limiters of the gradient in an attempt to preserve monotonicity. The formulation is spatially two-dimensional with options for Cartesian and curvilinear geometries. Discretization is achieved with a mesh of arbitrary quadrilateral cells whose vertices can move with time. Arbitrary mesh motion is supported by allowing transport of material between cells according to the Arbitrary Lagrangian-Eulerian (ALE) technique. The computation is performed in two phases: a Lagrangian phase and a remapping phase in which conserved variables are transferred from the Lagrangian mesh to an arbitrary specified mesh. The dynamic mesh capability generally smooths distortions in the mesh and can also result in higher resolution around features of interest, such as a shock discontinuity.

This report is a second edition of an earlier report on CAVEAT [Addessio et al., 1985]. Because of the maturity and expanded use of CAVEAT, this report includes new sections directed toward first-time users. We have chosen to document in this report only the hydrodynamics part of the much larger version of CAVEAT. The previously documented extensions to the hydrodynamics (high explosives, material strength, turbulence, slide-line treatment) and new additions (interface tracking, turbulent mixing, predictor-corrector numerical technique, a three-dimensional version) will be documented separately.

The main improvements and changes to the hydrodynamics since the last report are the addition of interacting blocks of mesh, greatly improved vectorization, directional splitting in the remap phase, a faster global rezone method, and a new tangential rezone method.

---

† Direct inquires about CAVEAT to Norman L. Johnson, MS B216, Los Alamos National Laboratory, Los Alamos, New Mexico 87545 or by electronic mail to NLJ@LANL.GOV on INTERNET.

SECTION 1

INTRODUCTION

CAVEAT is a general purpose computer code for compressible hydrodynamics that solves classes of problems not previously amenable to numerical study. In particular, the CAVEAT code is applicable to the computation of high-speed dynamics during large contortion of several interacting materials in the presence of significant interfacial slip. In addition many of the techniques in CAVEAT may be applied to codes that are based on alternative numerical schemes.

Perspective

Traditionally, numerical solutions of problems in fluid dynamics have used either a Lagrangian or Eulerian approach. The typical Lagrangian approach uses a finite-difference or finite-element mesh with fixed logical connections among the cells. In the finite-difference Lagrangian approach the velocities are typically located at mesh vertices, and pressures, density, internal energy are located at the cell centers. The Lagrangian methods have the advantages of well-resolved material interfaces and no advective diffusion, but applications with a large shearing distortion lead to highly distorted cells and, inevitably, to mesh tangling. The slide line treatment, an extension to Lagrangian codes, greatly extends their ability to describe interfacial slip, but not without problems of stability and difficulty of extension to three dimensions.

In contrast, Eulerian approaches do not suffer the Lagrangian limitations of mesh distortions but do suffer from advective diffusion and inaccuracies inherent in the treatment of multiple materials within a single cell (a mixed cell treatment). Furthermore, the restriction to a fixed, orthogonal mesh limits the ability of Eulerian codes to describe problems with large changes in characteristic lengths over time.

Advances beyond these traditional approaches have resulted in new combined methods. One such method, the Particle-in-Cell (PIC) technique, overlays Lagrangian marker particles on an Eulerian computational mesh, thereby allowing both extreme distortion of material interfaces and interfacial slip. The difficulties of the classical PIC method are similar to those of the mixed cell methods, with the addition of higher computational cost. Recent reformulations of the PIC method, in combination with an adaptive mesh, have eliminated many of the earlier difficulties and rekindled new interest in this older technique.

The more recent arbitrary-connectivity methods (including free-Lagrangian and smooth particle hydrodynamics) allow time-varying connections between mesh points. These have the advantages of both Lagrangian and Eulerian methods (resolved material interfaces, minimal advective diffusion, and

ability to treat interfacial slip) but suffer either from inaccurate treatment of material interfaces or inherently higher complexity, and maybe difficult to apply to three dimensions.

In recent years it has become customary to split the hydrodynamic time-step into a Lagrangian phase and a rezone-remap phase, in which *rezoning* is the specification of the new mesh and *remapping* is the transfer of the state variables from one mesh to another. This approach is the *Arbitrary Lagrangian-Eulerian* (ALE) formulation. An opportunity arises once the two phases are separated. After the Lagrangian step is taken, a new mesh can be specified in a manner desirable for the accuracy or stability of the calculation. This rezoning can follow the fluid motion (pure Lagrangian, with no advection relative to the mesh), return the mesh to the same initial configuration each cycle (pure Eulerian, with considerable advection), or recast the mesh in various other ways, for example, to follow features of interest moving through the mesh such as a shock wave.

Goals in Development

The CAVEAT code draws on these and other numerical techniques to produce a general-purpose code. The objective was to produce:

- A flexible and robust code that can treat a variety of problems with large shears and resolved material interfaces,
- A code with minimal diffusion so added physical processes are not masked by the numerical diffusion,
- An efficient code that can treat large, complex problems,
- A code that uses methods that are extendible to three-dimensions, and
- A code that is easily modifiable and applicable to the development of new physical models.

When alternatives arise in the development of CAVEAT, some balance among these objectives is necessary. Typical questions were: *Is the improved accuracy worth the additional computational cost or complexity? , Can the method be extended to three dimensions?*, and *Is the method robust?* . Many alternatives were rejected as a consequence, and the resulting computer code, CAVEAT, is a compromise among many competing methods. In some situations no alternatives were clearly superior so several treatments were implemented (as in the calculation of the fluxing volume).

Characteristic Features of CAVEAT

CAVEAT calculates the material dynamics using a finite-volume, finite-difference formulation. Some of its characteristic features are the following:

- All primary variables, including velocities, are cell centered, which enables slip between regions of fluid,
- Mesh lines along material interfaces are *Lagrangian*, i.e., they remain on the material interface,

- Mesh rezoning and variable remapping allow for a range of prescribed mesh motions within each material (ALE formulation),
- A Godunov technique allows close coupling between cell-centered pressure and velocity fields and minimal numerical diffusion, and
- The problem domain is a set of interacting, logically-rectangular meshes with computational cells with four sides that results in a code that is highly vectorized and extendible to three dimensions.

The use of cell-centered velocities is a point of major departure from most finite-difference codes. There are four principal advantages: (1) the facilitation of the calculation of material slip; (2) consistency among control volumes for the advection of mass, momentum, and energy; (3) consistency in the control volume for internal and kinetic energy; and (4) the ability to use the Godunov approach to obtain the cell-edge velocities and pressure. These are each discussed briefly below.

The benefits of cell-centered velocities in accommodating slip are illustrated by the example in Figs. 1-1 and 1-2. Fig. 1-1a shows the configuration of computational cells in the vicinity of an interface between two materials at the beginning of a computational cycle for a cell-centered velocity formulation. For numerical methods that use cell-centered velocities as in CAVEAT, the acceleration and work are computed based on the cell-centered velocities. This treatment yields the mesh configuration at the end of the Lagrangian phase shown in Fig. 1-1b. (This intermediate configuration cannot be realized in the calculation because typically multiple vertices do not exist along the interface.)

Cell-centered velocities

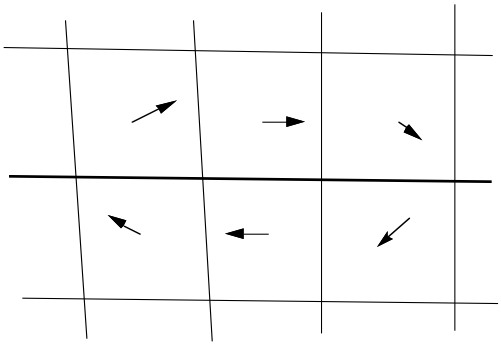


Fig. 1-2b. Computational cells after the Lagrangian phase of the cycle.

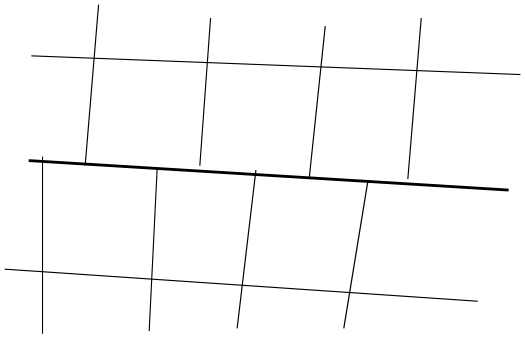


Fig. 1-2b. Computational cells after the Lagrangian phase of the cycle.

Vertex-centered velocities

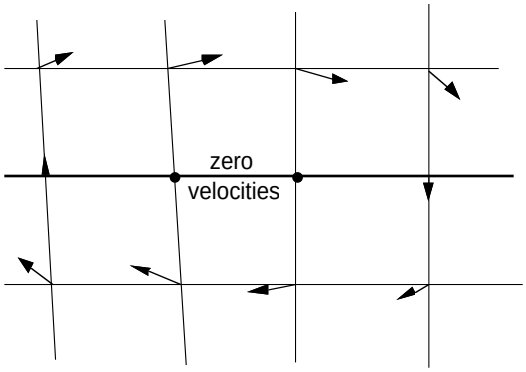


Fig. 1-2a. Computational cells and velocity vectors at the beginning of the cycle along a material interface.

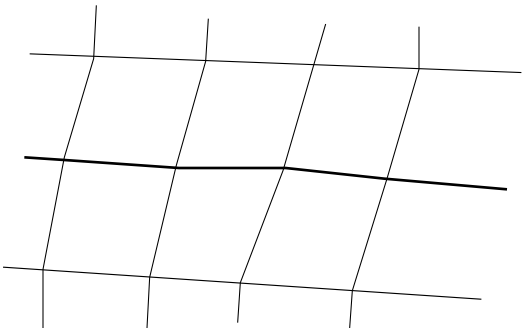


Fig. 1-2b. Computational cells after the Lagrangian phase of the cycle.

In Fig. 1-2 velocities and computational cells analogous to Fig. 1.1 are shown for a formulation with vertex-centered velocities. The velocities at the vertices that lie on the interface must have a tangential component that is some average of the velocities across the interface; in the example in Fig. 1-2, these tangential velocities would be close to zero while the tangential velocities of the materials are non-zero. This establishes a fictitious boundary layer and, hence, a source of numerical viscosity, in addition to any physical viscosity in the problem, which is a function of the cell size. It should be noted that the formulations with cell-centered velocities sacrifice second-order time differencing (corner coupling) in the Lagrangian phase in the place of a better description of interfacial slip (§7).

The advantage of consistency among control volumes for advection of mass, momentum and energy can be illustrated by considering the inconsistency that occurs with staggered control volumes, for example, with vertex-centered velocities and cell-centered masses. Momentum advected into a vertex-centered control volume must yield a new velocity for that vertex, calculated by the quotient of new total momentum divided by the new mass associated with the vertex. Mass advection, however, takes place relative to cell-centered control volumes, so the appropriate new mass at the vertex must be calculated as some weighted average of four surrounding cell masses. Any inconsistency produces an erroneous velocity from the momentum. Experience with vertex-velocity techniques shows a nagging propensity for anomalous velocities arising from such unavoidable inconsistencies.

The coincident location of kinetic and internal energies is a distinct advantage in computer codes that are total-energy based, as is CAVEAT. Because the internal energy is calculated from the difference between the total energy and the kinetic energy, inconsistencies in the determination of the velocity and mass, as described above, can result in anomalous internal energies, and therefore

erroneous pressures. For internal-energy based codes, this appears not to be a problem, but such formulations are not rigorously conservative of total energy.

The benefits of a Godunov approach are of considerable importance. Staggered-mesh techniques result in a close coupling between pressure and velocity fields but usually require artificial dissipation for numerical stability and/or the achievement of proper entropy changes. Such artificial dissipation may be explicit (artificial viscosity) or hidden (donor-cell advection), but such methods typically fail to overcome mesh-drifting tendencies and often are excessively diffusive. With the Godunov approach, cell-edge pressures and velocities required for acceleration and work terms are obtained by solving the Riemann problem each cycle from the cell-centered states on the two sides of each cell edge. The result is again close coupling between pressure and velocity fields, usually with just the optimal local level of dissipation. Because the minimal amount of dissipation is added by the Riemann solution, the Godunov method results in lower overall diffusion, and this does not mask the effect of the realistic sources of diffusion from physical processes. An alternative to the Godunov procedure with cell-centered velocities is to use simple weighted average of adjacent pressures and velocities at each cell edge, but such an approach can be demonstrated to produce adverse decoupling between pressure and velocity fields. Furthermore, the Godunov method results in realistic behavior at material interfaces, where density discontinuities may be large.

The example in Fig. 1-1 can be used to illustrate the meaning of the ALE technique. The rezoning and remapping phase is applied to the mesh in Fig. 1-1b. By using the ALE formulation, we are free to specify a new mesh (rezoning) within the constraints of the problem, and remap the state variables onto the new mesh. One constraint on the specification of the mesh is that the mesh must move with the fluid velocity in the direction normal to the material interface in order to preserve the Lagrangian nature of the material interface; i.e., no material can flux through the interface. Another constraint is that in order to retain the quadrilateral topology of the mesh in the tangential direction, a common point must be chosen for each split-vertex pair along the interface to restore the logical connection between the meshes. (Otherwise, a slide-line treatment is required.) But these common points can be chosen arbitrarily so long as the shape of the Lagrangian interface is preserved. Furthermore, away from material interfaces, the specification of the new mesh is somewhat arbitrary. Therefore within the constraints mentioned above, a new mesh can be chosen with desirable properties such as improved smoothness or higher resolution near interesting features such as shock waves. The ALE formulation, consequently, can extend the duration of a typical Lagrangian calculation by preventing mesh tangling, and it can use the mesh more efficiently by refining the mesh in regions of interest at the expense of regions that require less resolution. There is a limit to the distortion that can be accommodated by CAVEAT, such as a vortex in the vicinity of a material interface, because the mesh must have a constant connectivity. In situations where these limits are exceeded, the mixed-cell extensions to CAVEAT must be used [Johnson et al., 1990].

The choice of the quadrilateral cell topology is largely motivated by its high computational efficiency, easy extension to three dimensions, and ease of modification. Because of this choice, vector loops are constructed to go over the entire mesh, the range of the vector loops is specified once at initialization, in contrast to arbitrary-connectivity codes that require recalculation of the connectivity, and data is stored regularly in memory. The disadvantage of a quadrilateral topology is in the description of problems with large changes in characteristic length, because a fine mesh as needed by the small length scales must extend unnecessarily throughout the problem (§7). This disadvantage is significantly offset by the capability of CAVEAT to connect logically rectangular blocks of mesh. The multi-block capability also facilitates multi-tasking, use of multiple time steps, and application of the CRAY's solid-state disk [Johnson et al., 1990].

Some indication of the degree of vectorization achieved in CAVEAT is that the main hydrodynamics routines run between 80 and 100 MFLOPS (10^6 floating point operations per second) on the CRAY XMP, which is near optimal. A typical grind time (computer time per cell per cycle) for the Lagrangian phase using the second-order treatment and an analytical equation of state is 15 μs, and 30 μs for a calculation including the remapping phase. Because CAVEAT allows for multiple Lagrangian steps (subcycling of the Lagrangian phase) for each remapping calculation, the larger grind time for the remapping step need not degrade the average grind time for calculations in which restrictions on the time step are dominated by the sound speed and not advection.

Applicability to Other Codes

Most of the techniques used in CAVEAT are applicable to other codes. In particular, the ALE formulation with adaptive capabilities can be used in many Lagrangian codes. The concept of interconnection of blocks of mesh is useful to Eulerian codes where, as in CAVEAT, the limitation of the quadrilateral topology leads to non-optimal use of the mesh in problems with large changes in characteristic length. Finally, the Godunov method with its second-order implementation is of interest to those who wish to minimize the numerical diffusion in their codes. A potential disadvantage of the Godunov method that should be stressed is that the addition of new physics in a Godunov code may require modifying the Riemann solution; this is not generally a well-understood procedure (§7), but it has been done in CAVEAT for high explosive, elastic-plastic, and turbulence treatments [Johnson et al., 1990].

Differences Since the Last Report

This report is the second documentation of the CAVEAT code. Although the fundamental approach has remained unchanged, the structure and implementation of the code have undergone significant modification. The history of the current version of CAVEAT has been outlined in the preface. In summary, almost the entire code has been rewritten in a form that is highly vectorized (a

factor of three to five times faster). Important extensions and changes that are documented in this report include: (1) the capability for division of the fluid region into separate blocks of mesh, (2) spatial splitting of the remapping phase to achieve corner coupling in the advection, (3) a significantly faster global rezoner that is extendable to three dimensions, and (4) a new tangential rezoner for material interfaces. These differences are documented in §4.1.4, §3.2.3c, §3.2.2, and §3.2.2d, respectively.

Overview of the Report

This report serves two purposes: to document the current implementation of CAVEAT and to provide a users manual for the computer code. The organization of the report is directed toward serving the needs of the those who wish to modify the source code, developers of other codes, and those who wish simply to apply the code. For those interested in the fundamentals of CAVEAT, §2 describes the basic equations and physics on which the code is based, and §3 presents the numerical techniques. For those who are interested only in applying the computer code, the description of the computer program in §4, the example problems in §5, and the introduction for first users in §6 constitute a user's manual. For those familiar with the code, Appendix A is a useful quick reference to the input variables and files. Entries in Appendix A are cross referenced to the text to provide further information if required.

We have also chosen in the current report to document only the basic hydrodynamics, because we perceive a growing need for such a code and cannot do justice to the documentation of the extensions, many of which are still being developed. The CAVEAT code forms the basis for numerous extensions, including the features documented in the previous report (high explosives, elastic-plastic material strength, turbulence, and slide-line treatment) and new extensions not reported previously (interface tracking, turbulence mixing, ductile failure model, predictor-corrector method instead of the Godunov method, and a three-dimensional version). These will be documented in separately but are briefly described in companion report [Johnson et al., 1990].

The source code for the basic hydrodynamics described in this report is available through the Argonne Code Center (see preface for address). The code has been written for CRAY computers using the CRAY Fortran CFT77 compiler. Certain system calls may present difficulties when CAVEAT is implemented at CRAY facilities other than at Los Alamos National Laboratory or on other types of computers; these non-standard system calls and alternatives are described in Appendix C. For users located at Los Alamos National Laboratory, Appendix B describes how to obtain the source code and create an executable file.

SECTION 2

EQUATIONS AND PHYSICS

§2.1 HYDRODYNAMICS

CAVEAT is a multi-material, single-phase code. The calculational domain is a set of multiple, distinct, but interacting regions (cells), each of which contains a single material with a single velocity field. Each of these regions is described by the following single-material conservation equations written in Eulerian form:

Mass Conservation

$$\frac{\partial \rho}{\partial t} + \nabla\cdot (\rho \mathbf{u}) = 0 \ , \tag{2.1-1}$$

Momentum Conservation

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla\cdot (\rho \mathbf{u}\mathbf{u}) = - \nabla p + \nabla\cdot \Pi + \rho \mathbf{F} \ , \tag{2.1-2}$$

Energy Conservation

$$\frac{\partial \rho E}{\partial t} + \nabla\cdot (\rho E \mathbf{u}) = - \nabla\cdot p\mathbf{u} + \nabla\cdot \Pi\cdot \mathbf{u} + \rho \mathbf{F}\cdot \mathbf{u} - \nabla\cdot \mathbf{h} + Q \, . \tag{2.1-3}$$

In these equations ρ is the mass density, $\mathbf{u}$ is the material velocity, $E = e + (1/2)\mathbf{u}\cdot\mathbf{u}$ is the mass specific total energy, and e is the mass specific internal energy (exclusive of chemical energy). The pressure, p, is given as a function of ρ and e (or temperature, T) by the equation of state relations for the material, described in §2.2. The deviatoric stress tensor Π may describe simple fluid viscosity, turbulence, or elastic-plastic material deformation. Depending on the problem, there may exist a body force per unit mass $\mathbf{F}$, a heat-flux vector $\mathbf{h}$, or energy-release rate per unit volume Q.

It is convenient, for the subsequent discretization, to recast these equations in the more fundamental control-volume formulation [Thompson, 1982] which holds for an arbitrary moving control volume:

$$\frac{d}{dt}\int_{V_a(t)} \rho \, dV + \int_{S_a(t)} \rho(\mathbf{u} - \mathbf{u}_a)\cdot\mathbf{n} \, dS = 0 \ , \tag{2.1-4}$$

$$\frac{d}{dt}\int_{V_a(t)} \rho \mathbf{u}\, dV + \int_{S_a(t)} \rho \mathbf{u}(\mathbf{u} - \mathbf{u}_a)\cdot\mathbf{n}\, dS = -\int_{S_a(t)} p\mathbf{n}\, dS + \int_{S_a(t)} \mathbf{n}\cdot\Pi\, dS + \int_{V_a(t)} \rho \mathbf{F}\, dV, \qquad (2.1\text{-}5)$$

$$\frac{d}{dt}\int_{V_a(t)} \rho E\, dV + \int_{S_a(t)} \rho E(\mathbf{u} - \mathbf{u}_a)\cdot\mathbf{n}\, dS = -\int_{S_a(t)} p\mathbf{u}\cdot\mathbf{n}\, dS + \int_{S_a(t)} \mathbf{n}\cdot\Pi\cdot\mathbf{u}\, dS$$

$$-\int_{S_a(t)} \mathbf{h}\cdot\mathbf{n}\, dS + \int_{V_a(t)} \rho \mathbf{F}\cdot\mathbf{u}\, dV + \int_{V_a(t)} Q\, dV\,, \qquad (2.1\text{-}6)$$

Here, the surface $S_a(t)$ of the control volume $V_a(t)$ is assumed to move with an arbitrary local velocity $\mathbf{u}_a$. The time rate of change operator d/dt is used to indicate that we are following the motion of the control volume $V_a(t)$. The surface integrals involve **n**, the outward unit normal to the surface. Implicit in the use of these equations is also the conservation of volume:

$$\frac{dV_a(t)}{dt} = \int_{S_a(t)} \mathbf{u}_a\cdot\mathbf{n}\, dS\,, \qquad (2.1\text{-}7)$$

which is equivalent to the local kinematic equations

$$\frac{d\mathbf{x}_a}{dt} = \mathbf{u}_a\,, \qquad (2.1\text{-}8)$$

where $\mathbf{x}_a$ and $\mathbf{u}_a$ are the position and velocity of a point on the control-volume surface.

The control-volume velocity $\mathbf{u}_a$ has two important limiting values. In the Eulerian limit, $\mathbf{u}_a = \mathbf{0}$, and the control volumes are fixed. In the Lagrangian limit, $\mathbf{u}_a = \mathbf{u}$ and the control volumes are coincident with material volumes. The various advantages and disadvantages of these two choices are well known. In CAVEAT we use the ALE (Arbitrary-Lagrangian-Eulerian) approach to permit complete flexibility in the choice of the mesh or control-volume velocity [Hirt, et al., 1974]. In this approach, we recognize that the conservation Eqs. (2.1-4) through (2.1-6) are a combination of the fluid dynamics (a Lagrangian phase) and a coordinate transformation due to the mesh motion (a remapping phase), and we separate the two effects. The computation is decomposed into two phases, we transform Eqs. (2.1-4)-(2.1-6) to a coordinate system moving with the material velocity by the setting $\mathbf{u}_a = \mathbf{u}$:

$$\frac{d}{dt}\int_{V_L} \rho\, dV = 0\,, \qquad (2.1\text{-}9)$$

$$\frac{d}{dt}\int_{V_L} \rho\mathbf{u}\, dV = -\int_{S_L} p\mathbf{n}\, dS + \int_{S_L} \mathbf{n}\cdot\Pi\, dS + \int_{V_L} \rho\mathbf{F}\, dV, \tag{2.1-10}$$

$$\frac{d}{dt}\int_{V_L} \rho E\, dV = -\int_{S_L} p\mathbf{u}\cdot\mathbf{n}\, dS + \int_{S_L} \mathbf{n}\cdot\Pi\cdot\mathbf{u}\, dS - \int_{S_L} \mathbf{h}\cdot\mathbf{n}\, dS + \int_{V_L} \rho\mathbf{F}\cdot\mathbf{u}\, dV + \int_{V_L} Q\, dV\,, \tag{2.1-11}$$

where V_L is the Lagrangian (material) control volume, S_L is its surface, and the time rate of change refers to quantities associated with this volume. Following the Lagrangian phase, we have a distorted mesh, which we may wish to change. This is accomplished in the remapping phase which is defined by

$$\left.\begin{aligned} &\frac{d}{dt}\int_{V_k} \rho\, dV - \int_{S_k} \rho\mathbf{u}_R\cdot\mathbf{n}\, dS = 0\,, \\ &\frac{d}{dt}\int_{V_k} \rho\mathbf{u}\, dV - \int_{S_k} \rho\mathbf{u}\mathbf{u}_R\cdot\mathbf{n}\, dS = 0, \\ &\frac{d}{dt}\int_{V_k} \rho E\, dV - \int_{S_k} \rho E\mathbf{u}_R\cdot\mathbf{n}\, dS = 0, \end{aligned}\right\} \{V_k : V_L \rightarrow V_k^*\}, \tag{2.1-12,13,14}$$

where V_k is the control volume defined by the mesh velocity $\mathbf{u}_a$, S_k is its surface, $\mathbf{u}_R = \mathbf{u}_a - \mathbf{u}$ is the mesh velocity relative to the Lagrangian frame, and V_k^* is the new mesh-cell volume (control-volume), or the volume at the termination of the integration of these equations. The control volume V_k changes from its initial value V_L to its final value V_k^* in this process. The remapping equations are all of the form

$$\frac{d}{dt}\int_{V_k} q\, dV - \int_{S_k} q\mathbf{u}_R\cdot\mathbf{n}\, dS = 0\,, \tag{2.1-15}$$

where q represents the spatial density of the remapped quantity, such as ρ, ρ**u**, or ρE. We assume that q has been determined in the Lagrangian phase and does not change (is frozen) during the remapping phase. It is clear from the preceding equations that we are primarily interested in quantities integrated over the cell volume (or the control volume). Therefore, we could as well integrate directly over the known density distribution to obtain

$$Q_k = \int_{V_k^*} q \, dV \, . \qquad (2.1\text{-}16)$$

Thus, we have two options for the remapping phase: the use of Eq. (2.1-15), which is known as a continuous remapping, or the use of Eq. (2.1-16), which may be referred to as integral remapping [Dukowicz, 1984]. Only the former is available in CAVEAT while the latter has been used experimentally and may be incorporated as an option in later versions.

§2.2 EQUATIONS OF STATE

CAVEAT can utilize five types of equations of state (EOS). Four of these are analytical: linear, quadratic (Osborne), Gamma Law (ideal gas), Stiffened gas, and HOM (Gruneisen). The fifth equation of state is the SESAME tabular form.

The user associates an equation of state with a material by specifying an equation of state type when the material is defined. The parameters of the equation of state are also set in this step, which is explained in detail in §4.4.3. This section will describe the equation of state forms and their parameters.

Linear EOS

The form of this equation of state is

$$p = A + B\,E \qquad (2.2\text{-}1)$$

where

$$E = \rho_0 \, e \, ,$$

$$A = \alpha \, (c_1 + c_2 \, |\alpha|) \, ,$$

$$B = c_3 + \alpha \, c_4 \, ,$$

$$\alpha = \Gamma - 1 \, , \quad \Gamma = \rho/\rho_0 \, .$$

In the above equations, p is the pressure, ρ_0 is the initial density, e is the specific internal energy, ρ is the density for the cell under consideration, and $c_1 \ldots c_4$ are the constants.

The sound speed a, for all equations of state, is:

$$a^2 = \left(\frac{\partial p}{\partial \rho}\right)_e + \frac{p}{\rho^2}\left(\frac{\partial p}{\partial e}\right)_\rho \qquad (2.2\text{-}2)$$

For the Linear equation of state,

$$\left(\frac{\partial p}{\partial \rho}\right)_e = \frac{1}{\rho_0}(c_1 + 2c_2|\alpha| + c_4 E) \ ,$$

$$\left(\frac{\partial p}{\partial e}\right)_\rho = B\,\rho_0 \ .$$

A useful choice of the Linear EOS is for a constant sound-speed fluid: $c_1 = \rho_0 c^2$, $c_2=c_3=c_4=0$. Then the sound speed is c.

Quadratic (Osborne) EOS

The form of this equation of state is

$$p = \frac{A + E(B + C E)}{E + E_0} \ , \qquad (2.2\text{-}3)$$

where

$$E = \rho_0 e \ ,$$

$$A = \alpha (c_1 + c_2 |\alpha|) \ ,$$

$$B = c_3 + \alpha (c_4 + c_5 \alpha) \ ,$$

$$C = c_6 + c_7 \alpha \ ,$$

$$\alpha = \Gamma - 1 \ , \quad \Gamma = \rho/\rho_0 \ .$$

In the above equations, p is the pressure, ρ_0 is the initial density, e is the specific internal energy, ρ is the density for the cell under consideration, and $c_1 ... c_7$ and E_0 are the constants.

The derivatives of the pressure with respect to density and internal energy are given by

$$\left(\frac{\partial p}{\partial \rho}\right)_e = \frac{c_1 + 2c_2|\alpha| + E\,(c_4 + 2\alpha c_5 + Ec_7)}{\rho_0(E + E_0)} ,$$

$$\left(\frac{\partial p}{\partial e}\right)_\rho = \frac{\rho_0(B + 2CE - p)}{E + E_0} .$$

Gamma Law (Ideal Gas) EOS

The form of this equation of state is

$$p = (\gamma - 1)\,\rho\, e , \qquad (2.2\text{-}4)$$

where p is the pressure, e is the specific internal energy, ρ is the density for the cell under consideration, and γ is the ratio of the specific heats. The derivatives of the pressure are

$$\left(\frac{\partial p}{\partial \rho}\right)_e = (\gamma - 1)\, e ,$$

$$\left(\frac{\partial p}{\partial e}\right)_\rho = (\gamma - 1)\,\rho .$$

The sound speed is calculated using Eq. 2.2-2.

Stiffened Gas EOS

The form of this equation of state is

$$p = c^2\,(\rho - \rho_0) \qquad (2.2\text{-}5)$$

where p is the pressure, ρ_0 is the initial density, e and ρ are the specific internal energy and the density for the cell under consideration, and c is the reference sound speed. The derivatives of the pressure are

$$\left(\frac{\partial p}{\partial \rho}\right)_e = c^2 ,$$

$$\left(\frac{\partial p}{\partial e}\right)_\rho = 0 \ .$$

The sound speed is calculated using Eq. 2.2-2.

HOM (Gruneisen) EOS

This is the Gruneisen equation of state. The pressure is given by

$$p = \frac{\Gamma}{V}(e - e_H) + p_H \ , \tag{2.2-6}$$

and the temperature is given by

$$T = (e - e_H)/C_V \ + T_H \ , \tag{2.2-7}$$

where p_H is the pressure on the Hugoniot, given by

$$p_H = \left[\frac{C}{V_0 - S(V_0 - V)}\right]^2 (V_0 - V) \ ,$$

and e_H is the energy on the Hugoniot, given by

$$e_H = (1/2)\, p_H\, (V_0 - V)$$

In these expressions, V is the specific volume ($V = 1/\rho$) and V_0 is the initial specific volume ($V_0 = 1/\rho_0$). The Hugoniot temperature is

$$T_H = \exp \{ F + G (\ln V) + H (\ln V)^2 + I (\ln V)^3 + J (\ln V)^4\} \ .$$

In the above equations, C and S are constants obtained from the relationship between shock speed U_S and particle speed U_p behind the shock, i.e.,

$$U_S = C + S\, U_p \ .$$

The value of C is also the speed of sound in the solid. There are two possible values for C and S. The usual values for these quantities are denoted as just C and S. If $V < V_{sw}$, then we use the parameters C_1 and S_1. For specific volumes greater than V_0, the Gruneisen expression is used with a standard pressure curve. So, for $V > V_0$, we use

$$p = \frac{\Gamma}{V}\left[e - \frac{C_V}{3\alpha}\left(\frac{V}{V_0} - 1\right)\right] + \frac{p_0 V_0}{V} \ , \tag{2.2-8}$$

$$T = T_0 + e / C_V \ .$$

The derivatives of the pressure and temperature are given by

$$\left(\frac{\partial p}{\partial e}\right)_V = \frac{\Gamma}{V} ,$$

$$\left(\frac{\partial p}{\partial V}\right)_e = - \frac{\Gamma}{V^2}\left(e - e_H + V \frac{de_H}{dV}\right) + \frac{dp_H}{dV} ,$$

$$\left(\frac{\partial T}{\partial e}\right)_V = \frac{1}{C_V} ,$$

$$\left(\frac{\partial T}{\partial V}\right)_e = \frac{\partial T_H}{\partial V} - \frac{\partial T}{\partial e}\frac{\partial e_H}{\partial V} .$$

The sound speed is calculated using Eq. 2.2-2.

SESAME

Reference [Holian, 1984] defines the SESAME equation of state library as "... a standardized, computer-based library which contains tables of thermodynamic properties for a wide range of materials over a wide range of physical regions (from ambient to astrophysical conditions)." The SESAME equation of state is further mentioned in Sections 4.4.3, 4.4.4, and Appendix D. The numerical details of the SESAME implementation are available form other sources [Cranfill, 1983; Kerley, 1977; Slattery and Spangenberg, 1982]. In this section we document the two additional features that have been added to the standard SESAME capability: density scaling and low density ramp equation of state.

SESAME Density Scaling

Density scaling allows the use of the SESAME tables for mixtures of materials which are not specifically included in the tables. This feature is commonly used for gases. When the option is activated, the density is multiplied by the scaling factor before the tables are accessed. The scaling factor is the ratio of the atomic weight (At.Wt.) of the table material to the atomic weight of the desired material. For example, a scaling factor of 20/22 can be used to obtain a Ne^{22} equation of state from a tabular equation of state of Ne^{20}, and a factor of 20/21 can be used with the Ne^{22} equation of state for a 50%-50% mixture of Ne^{20}-Ne^{22} gas. The factor is computed from the ratio

$$\frac{\text{At.Wt. } Ne^{20}}{50\% \text{ At.Wt. } Ne^{20} + 50\% \text{ At.Wt. } Ne^{22}} = \frac{20.}{0.5 \cdot 20. + 0.5 \cdot 22.} .$$

SESAME Ramp EOS

The ramp equation of state is used to model the behavior of low density foams or other types of porous materials that exhibit one response during a *crush* phase and another response when the voids have been eliminated. The latter behavior is described by the SESAME tables. Thus, another functional form is needed to describe the crush phase of the compression. The method employed in CAVEAT follows the recommended form [Abdallah, et al., 1980].

The ramp equation of state has two parts. For pressures below the transition pressure, p_t, the pressure is given by

$$p = A_1 (\rho/\rho_0 - 1) \ ,$$

where A_1 is the bulk modulus, $\rho_0 c^2$, and c is the reference sound speed. When the pressure is above a transition pressure, p_t, the pressure is described by the *crush* curve,

$$p = A_2 (\rho/\rho_0 - A_3) \ .$$

In obtaining these constants, the slope of the curve is determined by A_3, and the value of A_2 is obtained from:

$$A_2 = p_t A_1 / [p_t + A_1 (1 - A_3)] \ .$$

As noted in reference [Abdallah, et al., 1980], Section III.C, the crush-curve formula can also be used to model phase transitions in metals. The ramp model also includes an energy-shift factor that is added to the internal energy before the tables are accessed and an option that selects whether the ramp is irreversible or not. If the ramp is irreversible, once the SESAME table is used, it is always used; if the ramp is reversible, then the ramp pressure will be used whenever the ramp pressure exceeds the pressure obtained from the SESAME table.

SECTION 3

NUMERICAL TECHNIQUES

This section describes the numerical techniques used in CAVEAT. The objectives of the numerical methods in CAVEAT are to treat compressible, multimaterial problems in an accurate and general manner, and which can be implemented efficiently in multiple dimensions.

The first section introduces the formulations and techniques that are common to later sections on the hydrodynamics.

§3.1 FUNDAMENTAL TECHNIQUES

§3.1.1 Differencing of Equations

This report describes the two-dimensional version of CAVEAT. The computational mesh employed in CAVEAT is a regular (nonoverlapping) union of topologically rectangular submeshes, called blocks, each composed of arbitrary quadrilateral cells. A typical cell and its associated spatial-index notation, based on the logical integer indices (i,j), is illustrated in Fig. 3.1.1-1. There are three types of quantities defined on the mesh: those associated with cells, cell faces, and vertices. Quantities associated with the cells are the primary extensive quantities such as mass, momentum, and energy, as well as the average intensive quantities derived from them such as density, velocity, internal energy, pressure, temperature, etc. There are also secondary cell-quantities, such as various gradients, which will be described presently. Cell-face quantities are those which are required to evaluate the surface integrals, such as certain pressures, velocities, and stress tensor components. Typically, cell-face quantities are not retained but are derived as needed. Vertex quantities include the coordinates and velocities which are required to determine the mesh. Additionally, certain intermediate quantities, which are required in our numerical expressions, are associated with the vertices. The cells are assumed to be composed of a single material, and material interfaces are assumed to coincide with cell faces.

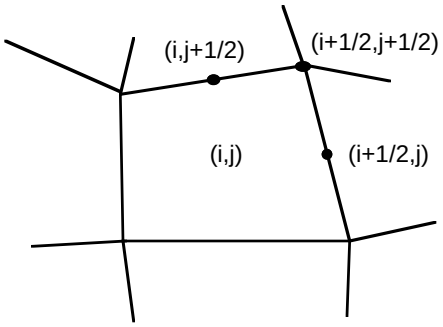


Fig. 3.1.1-1. Spatial index notation for a typical cell (i,j).

CAVEAT is an ALE code, as explained in a previous section. This means that the computations are carried out in two distinct phases, the Lagrangian phase and the remapping phase, as opposed to other codes which combine these two phases. The advantages of using the ALE method include its

clear physical motivation, based on separating the evolution of physical processes from the mesh interpolation procedure, and its flexibility, which allows the code to use arbitrarily moving meshes, ranging from the Lagrangian (moving with the fluid) mesh to the Eulerian (stationary) mesh.

The equations solved by CAVEAT are conservation equations, and the equations given in §2.1 are in conservation form in the absence of volumetric force and heat-release terms. CAVEAT, therefore, enforces conservation in the difference equations as a matter of principle. Internal and interface boundary conditions assume the continuity of pressure, normal velocity and stress components, and the normal component of heat flux (i.e., if the stress tensor and heat flux are implemented). These conditions are sufficient to guarantee the conservation properties of both the discrete and continuum forms of the Lagrangian equations (Eqs. 2.1-9 through 2.1-11), while the remapping equations (Eq. 2.1-15 or Eq. 2.1-16) are made conservative by construction. The only exception is in the case of cylindrical coordinates where the equations themselves are not in conservation form.

a. Spatial Differencing

CAVEAT uses a set of generalized orthogonal coordinates (x_1,x_2) which allow one to specify various geometrical options utilizing either Cartesian (x,y) or cylindrically symmetric (r,z) coordinates. These options will be discussed further below. In the cylindrically symmetric case, the equations of §2.1 contain a common factor of 2π due to the integration in the azimuthal direction. Customarily we drop this factor of 2π and consider all integrated quantities to be defined *per unit radian*. The equations describing these two cases can be combined by introducing pseudo-Cartesian coordinates (x_1,x_2), a pseudoradius $R(\mathbf{r})$, and a unit vector in the radial direction β. There are three cases (see the definition of IRADIAL in §4.4.1) shown in Table 3.1.1-1.

Table 3.1.1-1

Identification of coordinates for different geometries.

Coordinate System	x_1	x_2	$R(\mathbf{r})$	$\beta = \nabla R$
Cartesian	x	y	1	(0,0)
Cylindrical	r	z	r	(1,0)
Cylindrical	z	r	r	(0,1)

The Lagrangian equations of §2.1, in this combined form [Ramshaw and Dukowicz, 1979], may be written as

$$\frac{d}{dt}\int_{A_L} \rho R\, dA = 0, \tag{3.1.1-1}$$

$$\frac{d}{dt}\int_{A_L} \rho \mathbf{u} R\, dA = -\int_{A_L} \nabla p\, RdA + \int_{L_L} \mathbf{n}\bullet\Pi\, Rd\lambda - \int_{A_L} \Pi_0 \nabla R\, dA + \int_{A_L} \rho \mathbf{F}\, RdA ,$$

$$\frac{d}{dt}\int_{A_L} \rho E R\, dA = -\int_{L_L} p\mathbf{n}\bullet\mathbf{u}\, Rd\lambda + \int_{L_L} \mathbf{n}\bullet\Pi\bullet\mathbf{u}\, Rd\lambda - \int_{L_L} \mathbf{n}\bullet\mathbf{h}\, Rd\lambda + \int_{A_L} (\rho \mathbf{F}\bullet\mathbf{u} + Q)\, RdA ,$$

where $dA = dx_1 dx_2$ is the area element in the pseudo-Cartesian coordinates, A_L is the Lagrangian cell area, $d\lambda$ is the line element around the cell perimeter (corresponding to dS), L_L is the Lagrangian cell boundary, and Π_0 is the hoop-stress component. Note the altered appearance of the pressure gradient term and the appearance of the hoop-stress term in the cylindrical case. These two terms are differenced as follows

$$\int_{A_L} \nabla p\, RdA \cong \langle R\rangle \int_{A_L} \nabla p\, dA = \langle R\rangle \int_{L_L} p\mathbf{n}\, d\lambda ,$$

$$\int_{A_L} \Pi_0 \nabla R\, dA = \beta \int_{A_L} \Pi_0\, dA = \beta\, A_L \langle \Pi_0 \rangle ,$$

where

$$\langle R\rangle = V_L/A_L , \quad V_L = \int_{A_L} R\, dA , \quad \langle \Pi_0 \rangle = \frac{1}{A_L}\int_{A_L} \Pi_0\, dA ,$$

and β is the unit vector in the radial direction, defined in Table 3.1.1-1. Note that in cylindrical coordinates these terms contribute volumetric source terms so that the equations are no longer in pure conservation form.

The remapping (advection) equations (Eq. 2.1-15) become

$$\frac{d}{dt}\int_{A_k} q\, RdA = \int_{L_k} q\, \mathbf{n}\bullet\mathbf{u}_R\, Rd\lambda . \tag{3.1.1-2}$$

In all the above equations we note that the primary quantities of interest in CAVEAT are the extensive quantities associated with each cell, such as mass, momentum, and total energy, which may be conveniently expressed by

$$Q_k = \int_{A_k} q\, R dA \quad ,$$

where q represents the respective intensive quantities ρ, $\rho\mathbf{u}$, and ρE, and A_k is the area of a representative cell k. The intensive quantities all appear explicitly on the right-hand sides of the equations and must, in some way, be defined in terms of the extensive quantities, as will be described later. Most of the terms on the right-hand side of Eq. 3.1.1-1 are line integrals. These line integrals are expressed as sums over cell faces (or subfaces). Thus, the computations in CAVEAT are largely *cell-face oriented*. As an illustration, consider the generic example:

$$\int_{L_k} \mathbf{n}\cdot\mathbf{q}\, R d\lambda = \sum_{\alpha} \mathbf{n}_\alpha\cdot\mathbf{q}_\alpha R_\alpha \lambda_\alpha \, ,$$

where α is an index over all the faces (or subfaces) of cell k, λ_α is the length of face α, and $\mathbf{n}_\alpha$ is the unit outward normal of face α (which is a constant since each face is planar). Taking the right-hand face of the cell in Fig. 3.1.1-1 as typical (α = i+1/2,j), we can write

$$R_\alpha = \frac{1}{2}(R_{i+1/2,j+1/2} + R_{i+1/2,j-1/2}),$$

$$n_{1,\alpha}\lambda_\alpha = x_{2,i+1/2,j+1/2} - x_{2,i+1/2,j-1/2} \, ,$$

$$n_{2,\alpha}\lambda_\alpha = - x_{1,i+1/2,j+1/2} + x_{1,i+1/2,j-1/2} \, ,$$

where R is the pseudoradius defined previously, n_1, n_2 are the components of the unit vector **n**, λ is the length of the side, and x_1, x_2 are the respective vertex coordinates. The total contribution from this face, assuming that the (generic) variable vector **q** is defined at the center of the face, is

$$R_\alpha(q_{1,\alpha}n_{1,\alpha}\lambda_\alpha + q_{2,\alpha}n_{2,\alpha}\lambda_\alpha) \, ,$$

where q_1, q_2 are the components of the vector **q**. Other surface terms follow this general pattern.

b. Temporal Differencing

Time differencing in CAVEAT is explicit. The solution proceeds with respect to a sequence of discrete times t^n, where n is the cycle number (n = 0, 1 ,2,···). The time step $\Delta t^n = t^{n+1} - t^n$ varies from cycle to cycle. Quantities are labeled with a superscript to denote a time level or cycle number, e.g., Q^n

denotes a value at the current time t^n, that is, at cycle n. Absence of such a superscript implies the current time t^n, i.e., $Q \Leftrightarrow Q^n$.

The time differencing is explicit because the time advanced value of any extensive quantity Q is given by

$$Q_{i,j}^{n+1} = Q_{i,j}^{n} + \Delta t^n \left[dQ/dt \right]_{i,j}^{n},$$

where $\left[dQ/dt \right]_{i,j}^{n}$ is the right-hand-side of any of the conservation equations, evaluated with currently known quantities. This is the case for all quantities except for the volume V whose final value V^{n+1} is computed geometrically.

The above differencing is nominally first order accurate in time. The spatial differencing, described previously, amounts to central differencing on a regular mesh because of the cell-centering of the variables. It might be expected that such a combination would be unstable. However, it is known that inclusion of terms that are second-order accurate in time, which are essentially diffusive in nature, provides the necessary stability [Dukowicz and Ramshaw, 1979]. In CAVEAT, as in any Godunov-type method, the fluxes used to evaluate the right-hand-side of any of the conservation equations are essentially advanced time quantities, providing sufficient implicit diffusion to ensure the stability of the method.

§3.1.2 Gradients and Limiting

We have pointed out that CAVEAT solves equations for the conserved extensive quantities Q_k associated with each cell k, while the terms on the right-hand-sides of these equations involve the intensive quantities q_k. The accuracy of the computation, as well as the spatial order of accuracy, largely depends on the assumptions that are made regarding the spatial variation of $q_k(\mathbf{r})$ within each cell.

The extensive quantities provide us with cell-average intensive quantities:

$$\bar{q}_k = Q_k/V_k \ .$$

Other cell-average intensive quantities are generated from these, such as the cell velocity

$$\bar{\mathbf{u}}_k = \left(\overline{\rho \mathbf{u}}\right)_k / \bar{\rho}_k \ ,$$

or the pressure

$$\bar{p}_k = p\left(\bar{\rho}_k, \bar{e}_k\right),$$

which is obtained from the equation of state in terms of cell-averaged quantities. We will extend the definition of the cell-average intensive quantity $\bar{q}_k$ to include all these secondary quantities.

Let us now introduce a Taylor series-like expansion of $q_k(\mathbf{r})$ about some point $\mathbf{r}_k$ contained within cell k:

$$q_k(\mathbf{r}) = \bar{q}_k + \vec{\nabla} q_k \bullet (\mathbf{r} - \mathbf{r}_k) + \text{Higher Order Terms} , \qquad (3.1.2\text{-}1)$$

This will indeed be a valid, spatially second-order accurate Taylor series expansion if ∇q_k is at least a first-order ($O(\mathbf{r} - \mathbf{r}_k)$, $\mathbf{r}_k \in V_k$) approximation to the gradient, and $\bar{q}_k = q_k(\mathbf{r}_k)$, at least to second order. As a matter of terminology we denote our method as *first-order* if it truncates Eq. (3.1.2-1) to

$$q_k(\mathbf{r}) = \bar{q}_k ,$$

that is, if it assumes all variables to be *constant* within each cell, and *second-order* if it uses the truncation

$$q_k(\mathbf{r}) = \bar{q}_k + \nabla q_k \bullet (\mathbf{r} - \mathbf{r}_k) , \qquad (3.1.2\text{-}2)$$

that is, if it assumes a *linear* variation of q_k within the cell. *Note that this does not necessarily imply the corresponding formal order of accuracy.*

CAVEAT currently defines $\mathbf{r}_k$ to be the *cell center,* calculated as the average of the positions of the four vertices of cell k. We note that if the point $\mathbf{r}_k$ coincides with the centroid of cell k then the assumed distribution (Eq. 3.1.2-2) satisfies a constraint which must be true for the primary conserved quantities, namely, that

$$\int_{V_k} q_k(\mathbf{r})\, dV = \bar{q}_k V_k .$$

This constraint is automatically satisfied in first order. The existence of such a constraint is not necessary for nonconserved quantities such as **u** or p, but it is essential in order to maintain conservation when these distributions are used for remapping conserved quantities, such as ρ, ρ**u**, or ρE, when using Eq. (2.1-16).

a. Gradient Calculations

A convenient and generally useful approximation to the gradient ∇q_k is obtained if we define a secondary mesh whose nodes are the centers of the cells of the primary mesh. The gradient in cell k is taken to be the *area average gradient* :

$$\langle \nabla q_k \rangle = \frac{1}{A_j} \int_{A_j} \nabla q \, dA \quad , \qquad (3.1.2\text{-}3)$$

averaged over an area A_j (in the pseudo-Cartesian plane) defined by all the centers of the cells surrounding the given cell k. The numerical evaluation of Eq. (3.1.2-3) is facilitated by use of the divergence theorem to obtain

$$\langle \nabla q_k \rangle = \frac{1}{A_j} \int_{C_j} q\mathbf{n}\, d\lambda \quad , \tag{3.1.2-4}$$

where C_j is the contour of integration around the area A_j and **n** is the unit outward normal to the integration contour. This computation gives a first-order accurate approximation to the gradient if we assume a piecewise linear variation of q along the perimeter, because Eq. (3.1.2-4) then gives the exact result if q(**r**) is a linear function of the coordinates. Differencing is analogous to that described for surface integrals in §3.1.1.

In actual practice the gradient is evaluated in two steps. In the first step, intermediate gradients, called *trial gradients*, associated with cell vertices, are calculated using Eq. (3.1.2-4) where A_j is the area defined by the four cell centers surrounding a given cell vertex. These trial gradients are themselves useful in monotone limiting, described below. The cell gradients are then obtained in the second step as the area-weighted averages of the four trial gradients associated with the vertices of the given cell.

b. Limiting the Gradients

The above gradient approximation usually cannot be used directly in Eq. (3.1.2-2) because it can introduce severe overshoots (or undershoots) when compared to neighboring data, particularly in the vicinity of steep gradients. This destroys the desirable property of monotonicity of the local data distribution. To preserve monotonicity, we limit (reduce) the magnitude of the gradient. Of course, when the gradient is changed from the value given by Eq. (3.1.2-3), or Eq. (3.1.2-4), then the order of accuracy of the distribution given by Eq. (3.1.2-2) is locally reduced from second to first order. That is, we sacrifice the order of accuracy for the sake of monotonicity. Such limiting was first introduced in the one-dimensional case by van Leer [van Leer, 1979]. We employ two methods of limiting, which are described next.

Van Leer Limiting

This is a multidimensional extension of van Leer's original one-dimensional method (sometimes called the MUSCL scheme). We introduce a limiting coefficient α_k ($0 \le \alpha_k \le 1$) for each cell k such that

$$\nabla q_k = \alpha_k \langle \nabla q_k \rangle,$$

and α_k is determined by enforcing a local monotonicity criterion requiring that the quantity q in cell k does not lie outside the range of the average quantities q in the neighboring cells. This determines α_k to be

$$\alpha_k = \min\{1,\ \alpha_{max},\ \alpha_{min}\},$$

where

$$\alpha_{max} = \max\left\{0,\ (\bar{q}_{max} - \bar{q}_k)/(q_{kmax} - \bar{q}_k)\right\},$$

$$\alpha_{min} = \max\left\{0,\ (\bar{q}_{min} - \bar{q}_k)/(q_{kmin} - \bar{q}_k)\right\},$$

and $\bar{q}_{max}, \bar{q}_{min}$ are the maximum and minimum values of q in the neighboring cells, q_{kmax}, q_{kmin} are the maximum and minimum values of q in cell k, obtained from Eq. (3.1.2-2) with $\nabla_k q = \langle \nabla_k q \rangle$. The procedure described above applies to scalar quantities q. For vector quantities, such as velocity or momentum density, the procedure is modified somewhat. A single limiting coefficient α_k is used for both components of the vector, and α_k is calculated as above except that the quantities q are the scalar products of the vector in question with a reference vector. The reference vector currently used is the velocity or momentum density of the cell k for which the gradient is being calculated.

This limiter can nevertheless produce a *sawtooth* type of distribution of q at cell boundaries on a nonuniform mesh (i.e., it can be locally nonmonotone). Some oscillatory behavior is thus possible. We can eliminate this behavior by the use of an alternative limiter, described next.

Monotone Limiting

Recalling that we have previously introduced trial gradients $\langle \nabla_{ik} q \rangle$, which are calculated according to Eq. (3.1.2-4) and are associated with each vertex i of cell k, we find the maximum and minimum among these gradients:

$$(\nabla_k q)_{max} = \max\left\{\langle \nabla_{ik} q \rangle,\ i = 1, 2, \cdots, 4\right\},$$

$$(\nabla_k q)_{min} = \min\left\{\langle \nabla_{ik} q \rangle,\ i = 1, 2, \cdots, 4\right\}.$$

The limiting is then given by

$$\nabla_k q = \begin{cases} (\nabla_k q)_{min}, & (\nabla_k q)_{min} > 0; \\ (\nabla_k q)_{max}, & (\nabla_k q)_{max} < 0; \\ 0, & \text{otherwise}. \end{cases}$$

This limiting is, of course, done separately for each component.

In general, the van Leer limiting permits the steepest possible gradient without undue oscillations and therefore offers the least amount of diffusion or smearing. We have found in practice that the van Leer limiter is satisfactory in almost all cases. The monotone limiter is more conservative and therefore more diffusive. It is an option for those cases in which the van Leer limiter produces undesired local oscillations.

§3.2 HYDRODYNAMICS

§3.2.1 Lagrangian Phase

a. Godunov Method

The CAVEAT hydrodynamics algorithm in the Lagrangian phase is based on the Godunov method. There are many variants of the Godunov method, particularly in multidimensions. The common element in all these variants is the use of a solution to a Riemann problem, either exact or approximate. Advantages of the Godunov method include the absence of an explicit artificial shock viscosity and particularly good behavior at material interfaces, where there is an equation of state change and where the density discontinuity may be large. It is this last property that is particularly advantageous in CAVEAT.

The Godunov method as used in CAVEAT is defined with respect to the inviscid fluid equations (the Euler equations), which in the Lagrangian frame may be written as:

$$\frac{dm}{dt} = 0 \ ,$$

$$\frac{d}{dt}(m\bar{\mathbf{u}}) = -\langle R\rangle \int_{L_L} p^{*}\mathbf{n}\, d\lambda \ ,$$

$$\frac{d}{dt}(m\bar{E}) = -\int_{L_L} p^{*}\mathbf{u}^{*}\cdot\mathbf{n}\, R d\lambda \ ,$$

where $m = \int_{A_L} \rho R dA$ is the cell mass, $\bar{\mathbf{u}} = \frac{1}{m}\int_{A_L} \rho \mathbf{u} R\, dA$ is the cell-average velocity, $\bar{E} = \frac{1}{m}\int_{A_L} \rho E R\, dA$ is the cell-averaged total energy, and $\bar{e} = \bar{E} - \frac{1}{2}\bar{\mathbf{u}}\cdot\bar{\mathbf{u}}$ is the cell-averaged internal energy.

The quantities p* and **u***·**n** are the cell-face pressure and face-normal velocity, respectively. The pressure p* and the normal velocity **u***·**n** are obtained from the solution of a Riemann problem at the cell face. This use of a Riemann problem makes this a Godunov-type method, as stated above.

A local Riemann problem is defined by the discontinuity of the state variables ρ,e,w (w = **u**·**n**) on either side of a cell face. The discontinuity is resolved by a wave system which consists, in general, of a contact discontinuity and a shock or a rarefaction wave propagating to either side of, and away from the contact surface. We arbitrarily choose a left and right side of the cell face and define the normal velocity w to be positive toward the right. This gives us the two states (ρ_L, e_L, w_L) and (ρ_R, e_R, w_R) on either side of the discontinuity, which are assumed to be uniform away from the discontinuity. The solution of the Riemann problem provides the unique pressure p* and normal velocity w* of the contact surface, which are the required quantities associated with the cell face for use in the Euler equations.

The general (exact) solution of a Riemann problem for arbitrary materials is a very difficult nonlinear problem. We find, fortunately, that it is sufficient to solve the problem approximately, while preserving its essential features. The approximate method used in CAVEAT, applicable to arbitrary materials, is described in Ref. [Dukowicz, 1985]. Here we will only summarize some essential results. The approximate Riemann solver is based on the following approximate form of the shock Hugoniot:

$$p^* - p_s = \rho_s \left[a_s + A_s \left| w^* - w_s \right| \right] (w^* - w_s), \qquad (3.2.1\text{-}1)$$

where the subscript s refers to the left or the right state, and a_s and A_s are two *material-dependent* parameters. The general definition of these parameters is

$$a_s = \sqrt{\left(\frac{\partial p}{\partial \rho} + \frac{p}{\rho^2} \frac{\partial p}{\partial e} \right)_s} \quad = \text{the local isentropic speed of sound,}$$

$$A_s = \lim_{\text{strong shock}} \left[\frac{\rho_2/\rho_1}{\rho_2/\rho_1 - 1} \right]_s .$$

Thus, the strong shock parameter A_s is given in terms of the density ratio across a shock, in the strong shock limit. Typically this parameter does not depart greatly from unity, because for all materials with p = p(ρ) it can be shown that $A_s = 1$, while for polytropic (ideal gas) equations of state, $A_s = {}^1/_2(\gamma + 1)$. Thus, $1 \le A_s \le {}^4/_3$ for the physically realistic range of γ ($1 \le \gamma \le {}^5/_3$). More details are given in §4.4.3. The shock Hugoniot, in the form given in Eq. (3.2.1-1), is also useful for certain boundary conditions, for example, when the boundary pressure or velocity is specified (§3.2.4).

As in §3.1.2, we distinguish between a *first-order* and a *second-order* method. In the first-order method, the left and right states are given by the respective cell-centered values. In other words, it is assumed that all variables are uniform and constant within each cell. This most closely corresponds to Godunov's original description of this method. In the second-order method, we define the left and right

states to be the interpolations of the linear distributions described in §3.1.2 between the cell center and the center of each cell face, with the Courant number being the interpolation parameter. This most closely corresponds to a second-order discretization in time. However, numerical diffusion may be reduced by biasing the interpolation towards the cell-face center, and we provide a code parameter which accomplishes this (§4.3.1). With reference to Eq. (3.2.1-1), this is done for the pressure, density, and velocity, while the parameters a_s and A_s are assumed constant at their cell-centered values. Of course, the Riemann problem is time varying in such a case, but we use the instantaneous values of p* and w* immediately following the breakdown of such a discontinuity, and these values are the same as the values for the corresponding spatially uniform case.

We find that the first-order method is very robust, but also very diffusive, as might be expected from its close relationship to the well-known donor-cell differencing method. The second-order method is much more accurate and less diffusive, and therefore it is recommended in general. ln addition, there are two options when calculating gradients in the second-order method, as described in §3.1.2. The van Leer option frequently gives the best results but sometimes may cause local oscillatory behavior, in which case the monotone gradient option can be used to avoid these undesired oscillations.

b. Lagrangian Vertex Velocities

Vertex flow velocities are needed in CAVEAT to propagate the mesh in a Lagrangian calculation. They may also be needed to determine the velocity of the mesh relative to the flow for computing advection (the fluxing volume) in an ALE calculation (§3.2.3). Since these velocities are not directly available, they must be deduced from the Riemann face-normal velocities, w*.

Consider a typical internal vertex, illustrated in Fig. 3.2.1-1. There are many possible algorithms for determining the vertex velocity. CAVEAT uses a relatively simple and robust least-squares algorithm. We would like the normal projection of the vertex velocity on any of the connecting faces (typically four in the interior) to be equal to the corresponding Riemann velocity; this gives us the equations:

$$\mathbf{u}\cdot\mathbf{n}_\alpha = w^*_\alpha \ ; \qquad \alpha = 1, 2, \cdots, k \ ,$$

where α is the face index and k is the number of faces. Since we have only two unknowns (the two velocity components) and k equations (typically four), the system is usually overdetermined. We solve it using least-squares. That is, we minimize the weighted sum of the squares of the residuals with respect to the unknown parameters:

$$\frac{\partial}{\partial u_i} \sum_\alpha W_\alpha (\mathbf{u}\cdot\mathbf{n}_\alpha - w^*_\alpha)^2 = 0 \ ,$$

where $\mathbf{u} = (u_1,u_2)$ is the unknown vertex velocity, and W_α is a weight, associated with face α, which is equal to the sum of the material densities on either side of the face. This produces a 2x2 system of equations,

$$\sum_\alpha n_{i,\alpha} W_\alpha (\mathbf{u}\cdot\mathbf{n}_\alpha - w^*_\alpha) = 0 \quad ; \quad i = 1,\ 2\ ,$$

which is easily solved for the unknowns u_1,u_2.

This algorithm has some disadvantages in terms of accuracy. For example, because it mixes the two velocity components, it may produce spurious orthogonal velocities from a unidirectional velocity field on a nonuniform mesh. Nevertheless, the algorithm is difficult to improve upon due to its simplicity, robustness, and flexibility, and since it is equally applicable in one, two, or three dimensions to vertices with a variable number of connecting faces (as at interfaces and corners of mesh blocks).

§3.2.2 Rezone Phase and Adaptivity - Determination of the New Mesh

a. Winslow's Variable Diffusion Method

An algorithm for adaptive mesh generation proposed by Winslow [Winslow, 1981] is the basis for the rezoning procedure implemented in the CAVEAT code. Winslow's method minimizes the functional

$$I = \int_D \frac{1}{w}\left\{(\nabla\xi)^2 + (\nabla\eta)^2\right\} dxdy \qquad (3.2.2\text{-}1)$$

on the domain D, where ξ and η are the natural coordinates in mesh index space and $w = w(x,y)$ is a positive function on D. The function w controls the local mesh spacing and may be used to provide fine mesh in the vicinity of a feature of interest, such as a shock front. The reciprocal of w is much like a diffusivity, and this fact contributes to the naming of the method. When Eq. (3.2.2-1) is integrated by parts with Dirichlet boundary conditions, the following Euler equations are obtained that minimize the functional

$$\nabla\cdot\left(\frac{1}{w}\nabla\xi\right) = 0 \qquad (3.2.2\text{-}2a)$$

$$\nabla\cdot\left(\frac{1}{w}\nabla\eta\right) = 0\ . \qquad (3.2.2\text{-}2b)$$

This pair of equations may be manipulated through some tedious algebra to obtain the following generator equation for the method

$$\alpha \underline{x}_{\xi\xi} - 2\beta \underline{x}_{\xi\eta} + \gamma \underline{x}_{\eta\eta} = -2\ J\ (\nabla w)\ /\ w \qquad (3.2.2\text{-}3)$$

where $J = x_\xi y_\eta - y_\xi x_\eta$, $\alpha = (x_\eta^2 + y_\eta^2)/J$, $\beta = (x_\xi x_\eta + y_\xi y_\eta)/J$, $\gamma = (x_\xi^2 + y_\eta^2)/J$ and $\underline{x} = [x\ y]^T$. Subscripts ξ and η imply differentiation with respect to the natural coordinates ξ and η. More details on the derivation of the generator equation from the Euler equations can be found elsewhere [Thompson et al., 1974; Thompson et al., 1985].

b. Jacobi Iteration Solver

The generator Eq. (3.2.2-3) represents an elliptic system of equations that can be solved by a variety of methods. Although incomplete Cholesky decomposition-conjugate gradient (ICCG) and multigrid solvers offer more rapid convergence and could be used, we have chosen for reasons of simplicity and reduced memory requirements to use Jacobi iteration to solve the generator equation.

Suppose we desire to solve the linear system $Ax = b$ in an iterative manner. Iterative methods in general compute $x^{n+1} = x^n + \tilde{A}^{-1}(b - A\,x^n)$, where $\tilde{A}^{-1}$ represents an approximate inverse of the linear operator A. The method in CAVEAT used to solve the generator equation simply takes the inverse D^{-1} of the diagonal of A as the approximate inverse. This is the Jacobi method.

By applying this method to the system of equations of interest, one obtains from Eq. (3.2.2-3)

$$\underline{x}^{n+1} = \underline{x}^n + [2J\,(\nabla w)\,/\,w + \alpha\,\underline{x}_{\xi\xi} - 2\beta\,\underline{x}_{\xi\eta} + \gamma\,\underline{x}_{\eta\eta}]\,/\,(2\alpha + 2\gamma)\,. \qquad (3.2.2\text{-}4)$$

Here the approximate inverse is $-1/(2\alpha + 2\gamma)$. The factor of (-2) comes from the diagonal terms in the $x_{\xi\xi}$ and $y_{\eta\eta}$ second derivatives. The only control parameter required for this solver is the number of iterations. For most applications three iterations are sufficient.

c. Weight Function Generation

The primary control one has in the CAVEAT rezoner is in the specification of the weight function w. Generally speaking, in local regions where the logarithmic gradient $(\nabla w/w)$ of w is large, the mesh will be relatively finer than in neighboring regions where the logarithmic gradient is smaller. In problems for which the mesh is approximately rectangular and equal area cells are desired, a constant value of w provides excellent performance. However, in problems that use a polar mesh, in which one block boundary is collapsed to a point, it is necessary to use a weight function that increases with radius to prevent the mesh from collapsing onto the singular point.

If one desires the mesh to be finer about some local feature of interest, say a point or a line, one can achieve this simply by assigning the weight function w a large value in the neighborhood of the feature. Because it is helpful for w to be smooth, a few iterations of a simple smoothing filter are normally applied to w to remove any high frequency variations that may be present. In summary, good performance on a large class of problems is possible with w constant and no special attention given to the weight function by the user. However, to exploit the powerful adaptive capabilities of the rezoner in

problems where higher resolution in certain regions is critical, a user will usually need to design a weight function specifically suited to his application.

d. Material Interface Treatment

Applications frequently include more than one material type in the problem domain, and the interfaces between different materials need to be treated as Lagrangian surfaces. In particular the Lagrangian character of these interfaces must be preserved during the rezoning process. The approach taken in CAVEAT is first to compute the rezone motion of points on a material interface neglecting that the interface is Lagrangian and then to subtract away the portion of that rezone motion that is not tangent to the interface (Fig. 3.2.2-1). This procedure is followed during each iteration of Eq. (3.2.2-4). The critical issue in this approach is to calculate the interface tangent direction in a manner that is both robust and preserves the areas on either side of the interface as points slide along it

The strategy selected for defining the tangent direction at a given mesh point on the interface involves a linear combination of the results of two methods (Fig. 3.2.2-2). The first method takes the tangent direction of the interface in the vicinity of a given vertex to be parallel to the line segment connecting the nearest right and left neighboring vertices of the given vertex. This method is area-preserving but not stable. The second method treats the interface itself as a sequence of connected straight line segments. If the rezoner motion has the sense of moving a given vertex to the right of its present position, then the interface tangent corresponds to the line segment connecting the given vertex with its right-hand neighbor. If the motion is to the left, then the interface tangent is the line segment connecting the given vertex with its left-hand neighbor. This method is robust but not area-preserving. For most applications it has been found that using 98% of the first method and 2% of the second method is sufficient to obtain a result that is both robust and very nearly preserving of area. The user, however, can input whatever relative contributions from the two methods he may desire.

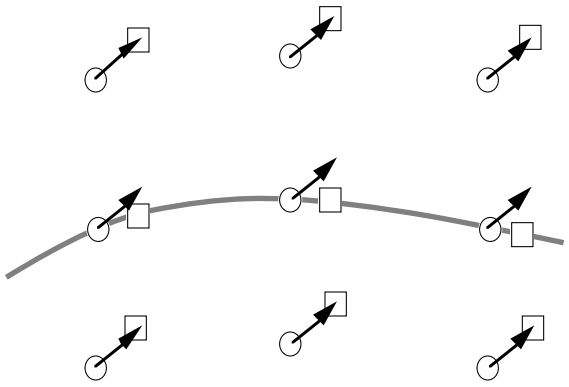


Fig. 3.2.2-1. Technique for rezoning material interfaces. In this example the rezone displacements are identical. On the material interface, the portion of the rezone displacement normal to the interface is subtracted from the initial displacement. Here circles denote original vertex positions, arrows the rezone displacement, and squares the position of the rezone vertices.

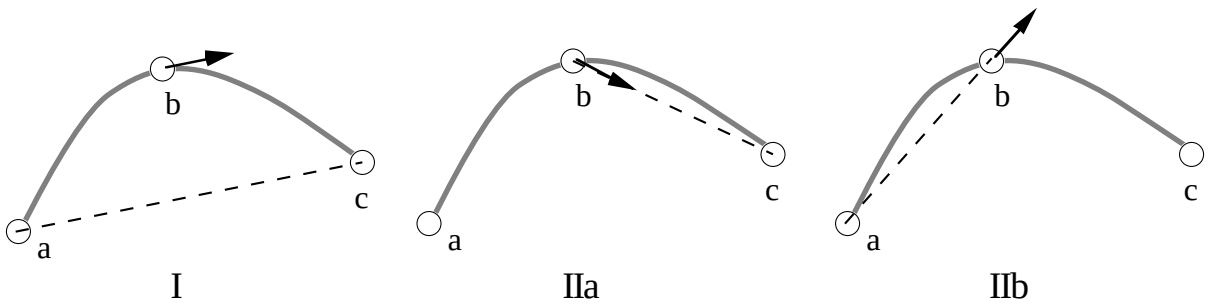


Fig. 3.2.2-2. Two methods for computing the local tangential direction on a material interface that passes through vertices a, b, and c. Method I uses as the tangential direction in the neighborhood of b the direction parallel to line segment $\overline{ac}$. Method II selects as the local tangent the direction parallel to the line segment $\overline{bc}$ in the region to the right of $\overline{ab}$ (IIa) and direction parallel to the line segment $\overline{ab}$ in the region to the left of b (IIb).

e. Boundary Treatments

Rezoning of the points along problem boundaries is usually desirable, and the method just described for rezoning material interfaces may be applied to problem boundaries as well. To achieve this it is necessary to update the positions of the ghost vertices surrounding the mesh block after each iteration of Eq. (3.2.2-4). This method for rezoning problem boundaries is adequate for a wide class of applications and is the default method in CAVEAT.

However, for some applications it is desirable to force a more uniform spacing of points along a boundary than can easily be provided by the Winslow generator. One example is when there is strong jetting along a boundary. For these situations a separate boundary rezoning treatment can be invoked that overrides the rezone motions computed by the Winslow method. This special boundary rezoner first finds the total arc length of the boundary and then calculates the lengths of each segment between points on the boundary. The segments can be of equal length or they can be adjusted according to the local curvature of the boundary, with shorter segments in regions of smaller radius of curvature.

Again it is desirable to preserve area inside the boundary and also have a computational stable procedure. The same approach described above of taking a linear combination of an area-preserving method and a robust method for calculating the local normal direction to the boundary is used. When material interfaces intersect the block boundary to which this special treatment is applied, then each section of boundary associated with a given material is rezoned separately.

§3.2.3 Remap Phase - Advection of Conserved Quantities

After rezoning, the state variables (mass, momentum, and total energy) at the end of the Lagrangian phase must be remapped to the new mesh. Remapping in this form makes use of the

conservative advection terms in the conservation laws and is therefore called continuous or advection remapping. Stability considerations limit zoning changes to advection volumes that are small relative to a cell volume during a single time step (§3.2.5). We focus in this discussion, therefore, on the discretization of the advection remapping given by (§3.1.1 and Eq. (3.1.1-2))

$$\frac{d}{dt}\int_{A_k} q\, R dA = \int_{L_k} q\, \mathbf{n}\cdot\mathbf{u}_R\, R d\lambda \ ,$$

where q is any of the state vector quantities per unit volume (ρ, $\rho\mathbf{u}$, or ρE) and $\mathbf{u}_R$ is the velocity of the fluid relative to the mesh ($\mathbf{u}_R = \mathbf{u}_V - \mathbf{u} = (\mathbf{x}^{n+1} - \mathbf{x}^L)/\Delta t$) (§2.1). In this expression and hereafter in this section, superscript L denotes state quantities that have been updated at the end of the Lagrangian step (§3.2.1).

a. Fluxing Algorithm

Using the superscript L to denote quantities present on the mesh at the end of the Lagrangian phase, the previous equation can be written in discrete form for each cell as

$$(Vq)^{n+1} = (Vq)^L - \sum_\alpha \langle q^L_\alpha \rangle \left(\mathbf{u}^L_{R,\alpha}\cdot\mathbf{n}_\alpha\, R_\alpha \lambda_\alpha\, \Delta t\right) \ ,$$

where V is the cell volume and Δt is the time step. In the summation, $\langle q^L_\alpha \rangle$ is the mean value of the state variable q in the region of space defined by the fluxing volume, $\left(\mathbf{u}^L_{R,\alpha}\cdot\mathbf{n}_\alpha\, R_\alpha \lambda_\alpha\, \Delta t\right)$, swept out by face α of the computational cell. Similarly, the cell volume is updated as follows:

$$V^{n+1} = V^L - \sum_\alpha \left(\mathbf{u}^L_{R,\alpha}\cdot\mathbf{n}_\alpha\, R_\alpha \lambda_\alpha\, \Delta t\right) \ .$$

The mean value $\langle q^L_\alpha \rangle$ is determined in one of three ways, according to user preference. In the first option, $\langle q^L_\alpha \rangle$ is set equal to the cell-centered value of q^L on the upstream side of cell face α (donor cell). In the second option, $\langle q^L_\alpha \rangle$ is obtained from a weighted average of the cell-centered values of q^L on either side of face α (pseudo-interpolated donor cell). In the third option, the mean value is computed by a truncated Taylor expansion in space about the value of q^L defined in either the upstream cell (as in donor cell) or by a weighting of two Taylor expansions from both cells adjoining the face.

All of the foregoing specifications of $\langle q^L_\alpha \rangle$ result in a conservative formulation. The first method, donor cell, is first order in space and therefore quite diffusive. The pseudo-interpolated donor cell method is higher order, in the sense that less numerical diffusion is introduced into the computation. This scheme approaches second-order accuracy as a user-specified coefficient that controls the degree of upwinding nears zero (becoming a standard centered-difference scheme), but it can be unstable and introduces oscillations. The Taylor expansion is also second order in space in

regions where gradient limiting does not occur (§3.1.2) and is stable provided that the Taylor expansion is carried only to the midpoint of the fluxing volume:

$$\langle q^L_\alpha \rangle = q^L_{\alpha,u} + \left[1 - (1/V)\,\mathbf{u}^L_{R,\alpha}\cdot\mathbf{n}_\alpha\, R_\alpha \lambda_\alpha\, \Delta t\right]\left[\nabla q^L_{\alpha,u}\cdot(\mathbf{x}_\alpha - \mathbf{x}_c)\right] \quad ,$$

where V is the cell volume, $\mathbf{x}_c$ is the upstream cell center coordinate, and the combined subscripts α,u denote a quantity centered in the cell upstream of face α. If a weighted average value from two cells is used, giving a second-order interpolated donor cell technique, this Taylor expansion is performed starting from each cell center near the face. The value of $\langle q^L_\alpha \rangle$ obtained through expansion to the cell face center is used only in the limit of vanishingly small advection volume, and if the fluxing volume matches the entire cell volume, the scheme reduces to either simple donor cell or interpolated donor cell.

b. Fluxing Volume Determination

The amount of a state quantity fluxed also depends on the magnitude of the fluxing volume, denoted above by $\left(\mathbf{u}^L_{R,\alpha}\cdot\mathbf{n}_\alpha\, R_\alpha \lambda_\alpha\, \Delta t\right)$. CAVEAT has two options available for calculation of the fluxing volume, each with its own advantages and drawbacks. Since the Lagrangian mesh point positions are available, it seems natural to merely calculate the volume swept out as the mesh points are moved to their post-rezoning locations. In fact, this option is the default in CAVEAT, giving, for example, a fluxing volume on the left-hand face of a cell given by the volume enclosed in the shaded box in Fig. 3.2.3-1. This volume depends only on the Lagrangian vertex positions and their locations after rezoning. As previously discussed in §3.2.1, however, the vertex velocities are computed on the basis of four surrounding face velocities. In Eulerian mesh calculations with large slip or shearing velocities, the Lagrangian mesh velocities near the slip plane will recognize the shearing and give, therefore, unwanted advection volumes in quiescent regions of the flow. Near the Lagrangian limit, this method of computing the fluxing volume should be quite accurate.

As an alternative, we also provide the means to calculate another advection volume based on the Riemann face velocity (§3.2.1), the n-time mesh point locations, and the rezoned mesh positions (see §4.3.3 for a detailed description of the method). Here, the fluid velocity is just that computed in the Lagrangian hydrodynamics phase, and the mesh velocity is taken to be $(\mathbf{x}^{n+1} - \mathbf{x}^n)/\Delta t$, where $\mathbf{x}^{n+1}$ is the vector of mesh positions after rezoning. In the Eulerian limit, because $\mathbf{x}^{n+1} = \mathbf{x}^n$, this method should yield accurate results, especially because the Riemann velocities will support a large shearing slip without inducing an artificial shear viscosity. However, in nearly Lagrangian calculations, the cell volume change computed by Riemann velocities will not coincide with the actual volume change resulting from Lagrangian mesh point movement. We suspect that this difference can cause a systematic error during advection, because the cell volume determined at the end of the time step (from vertex positions) will not be consistent with this fluxing scheme.

c. Split Directional Operator

It can be shown that temporal accuracy of advection is enhanced if directional splitting is applied to the operator described above. This is accomplished numerically by replacing the second equation in this section with two similar equations:

$$(Vq)^{*} = (Vq)^{L} - \sum_{\alpha} \langle q_{\alpha}^{L} \rangle \left(\mathbf{u}_{R,\alpha}^{L} \cdot \mathbf{i}_{\alpha} \, R_{\alpha} \lambda_{\alpha} \, \Delta t \right) = L_x(q^L) \quad ,$$

$$(Vq)^{n+1} = (Vq)^{*} - \sum_{\alpha} \langle q_{\alpha}^{*} \rangle \left(\mathbf{u}_{R,\alpha}^{*} \cdot \mathbf{j}_{\alpha} \, R_{\alpha} \lambda_{\alpha} \, \Delta t \right) = L_y(q^*) \quad ,$$

where $\mathbf{i}_{\alpha}$ and $\mathbf{j}_{\alpha}$ are the unit vectors in the two coordinate directions. Notice that the fluxed quantity q has been updated before the second step. To achieve a symmetric treatment, normally one would apply the operator four times per time step alternating directions as follows:

$$(Vq)^{n+1} = L_x L_y L_y L_x(q^L) \quad ,$$

with each directional operator using only half of a time step. In CAVEAT, however, we use only two passes each time step as shown above and alternate the first direction.

Note also that the calculation of the fluxing volume is affected by the split operator method as well. The mesh points are moved from their Lagrangian positions in the specified direction to an intermediate state, which gives the fluxing volume for the first pass. For the second step, the fluxing volume calculation relies on these intermediate values for two of the four points on the fluxing volume quadrilateral.

§3.2.4 Boundary Conditions

This section contains a discussion of the manner in which boundary conditions are specified or

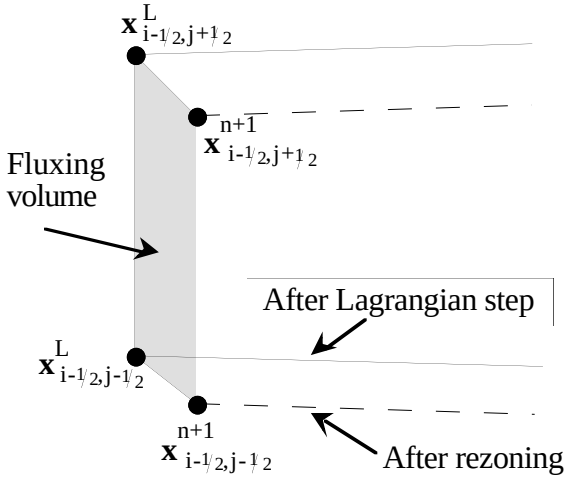


Figure 3.2.3-1. Illustration of the determination of the fluxing volume.

calculated in CAVEAT, for cell sides that form both exterior and interior boundaries in a problem.

a. External Boundaries

Boundaries that form the exterior surface of the computational domain are called *external* boundaries. There are two classes of external boundaries available in CAVEAT: Lagrangian and Eulerian. Lagrangian boundaries are those whose position evolves with the problem solution; Eulerian boundaries are those whose boundaries that are stationary in space.

Different boundary types are available in each class. The Lagrangian boundary types available are: specified velocity and specified pressure. The available Eulerian boundary types are: specified inflow, zero-gradient outflow, reflective (symmetry) wall, and curved-free-slip wall.

All boundary types of both boundary classes require determination of the Riemann pressure p* and velocity w*. In addition to p* and w*, the incoming and outgoing density, fluid velocity, total energy, and pressure must be specified in a set of *fictitious* (ghost) cells outside the computational domain adjacent to inflow and outflow boundaries. The remaining paragraphs of this section describe the manner in which the boundary values are determined.

Specified Velocity

The specified velocity boundary, in its simplest form, is represented by a piston moving in a shock tube; fluid particles adjacent to the piston move with the velocity of the piston, and there is no flux of fluid through the piston face (the boundary). In CAVEAT, this boundary condition is implemented by the specification of the boundary-vertex velocities **U**. (In general, **U** can be time-varying, but currently in CAVEAT **U** is taken to be constant.) From the velocity **U** the boundary-normal velocity is computed. For example, if the specified velocity boundary coincides with the right side of cell (i,j), then we have

$$w^*_{i+1/2,j} = \frac{1}{2}\,\mathbf{U} \cdot \mathbf{n}_{i+1/2,j} \ , \qquad (3.2.4\text{-}1)$$

where **n** is the outward-facing unit surface normal vector of the cell side. The pressure p* corresponding to the velocity of the contact surface w* is given by [Dukowicz, 1985]:

$$p^*_{i+1/2,j} - p^*_L = -\rho_L A_L \left| w^* - w^*_{max} \right| \left(w^* - w^*_{max} \right) \ , \qquad (3.2.4\text{-}2)$$

where

$$p_L^* = p_L - \frac{1}{4}\frac{\rho_L a_L^2}{A_L} ,$$

$$w^* = \min\left(w^*_{i+1/2,j},\ w^*_{max}\right) ,$$

$$w^*_{max} = w_L + \frac{1}{2}\frac{a_L}{A_L} ,$$

$$w_L = \mathbf{u}_{i,j} \cdot \mathbf{n}_{i+1/2,j} ,$$

in which the subscript L refers to the "left" side quantities in the Riemann problem, and ρ, A, a, and **u** are the cell-centered density, shock parameter, sound speed, and velocity, respectively. (In CAVEAT, the "left" side of the Riemann problem at the domain boundaries is always the side that lies to the *inside* of the problem domain.) Hence, once the Riemann velocity w* is specified, the Riemann pressure p* follows directly and depends on cell-centered quantities on the interior of the mesh. In second order calculations, the cell-centered density and velocity are replaced with linear interpolations in ρ, p, and **u** about the cell-centered value (§3.1.2).

Specified Pressure

The specified pressure-boundary condition is simply the inverse of the specified velocity boundary. For example, if the right side of a cell coincides with a specified pressure boundary, we have

$$p^*_{i+1/2,\,j} = P_0 ,$$

where P_0 is the applied pressure. (In general, P_0 can vary in space and time, but in the current version of CAVEAT, P_0 is taken to be constant in both space and time during the course of the calculation.) With p* given, Eq. (3.2.4-2) is inverted to give the boundary velocity w*

$$w^*_{i+1/2,j} = \begin{cases} w_t^* ; & (P_0 - P_L)/(w_t^* - w_L) < 0 \\ -w_t^* ; & (P_0 - P_L)/(w_t^* - w_L) > 0 \end{cases} \qquad (3.2.4\text{-}3)$$

where

$$w_t^* = \left|\frac{P_0 - P_L}{\rho_L A_L}\right|^{1/2} + w^*_{max} ,$$

and all the other quantities are as defined for the specified-velocity boundary condition.

Note that a free surface is simply a special case of a specified-pressure boundary in which $P_0 = 0$.

Specified Inflow

At the inflow boundary, the present version of CAVEAT places user-specified values of the density, velocity, and pressure at the cell-center values of the ghost cells just outside the physical computational mesh (the internal energy, e, is computed from the user-specified pressure, density, and equation of state). The specified values of pressure *and* the normal part of the velocity are also used as the Riemann pressure and velocity. For example, if the right side of cell (i,j) coincides with a specified inflow boundary, we have

$$\rho_{i+1,j} = \rho_0 \quad ,$$

$$\mathbf{u}_{i+1,j} = \mathbf{u}_0 \quad ,$$

$$E_{i+1,j} = e_0 + \tfrac{1}{2}\mathbf{u}_0 \cdot \mathbf{u}_0 \quad ,$$

$$p^*_{i+1/2,j} = p_{i,j} \quad ,$$

$$w^*_{i+1/2,j} = \mathbf{u}_0 \cdot \mathbf{n}_{i+1/2,j} \quad .$$

where ρ_0, $\mathbf{u}_0$, and e_0 are the user-specified density, velocity (a vector), and internal energy (as determined from the specified pressure), $\mathbf{n}$ is the outward-facing unit surface normal of the real cell side coinciding with the domain boundary, and $p^*_{i+1/2,j}$ is given by Eq. (3.2.4-2)· Note that the specified-inflow boundary condition can result in flow out of the mesh with the proper choices of input variables.

Zero-Gradient Outflow

The zero-gradient outflow boundary condition is implemented in CAVEAT by use of ghost cells, as with the specified inflow boundary. The term *zero gradient* simply means that there is no spatial gradient in the state vector at the outflow boundary. If, for example, the right side of a real cell (i,j) coincides with an outflow boundary, the zero-gradient specification is given by

$$\rho_{i+1,j} = \rho_{i,j} \quad ,$$

$$\mathbf{u}_{i+1,j} = \mathbf{u}_{i,j} \quad ,$$

$$E_{i+1,j} = E_{i,j} \quad ,$$

$$p_{i+1,j} = p_{i,j} \quad ,$$

$$w^*_{i+1/2,j} = \mathbf{u}_{i,j} \cdot \mathbf{n}_{i+1/2,j} \quad ,$$

$$p^{*}_{i+1/2,j} = p_{i,j} \quad .$$

Note that under certain conditions the outflow boundary can result in flow of material *into* the mesh; this can be useful is setting up problems where it is desired that material identical to the material in the interior of the mesh flow into the boundary.

Reflective (Free-Slip) Wall

A reflective-wall boundary is a symmetry boundary on which the normal-velocity component is zero and the tangential-velocity component is the same as the component of the adjacent cell-centered velocity in a direction tangent to the cell side coincident with the boundary. Hence, if the ride side of the real cell (i,j) coincides with a reflective wall, we have

$$u_n = 0$$

and

$$u_t = \mathbf{u}_{i,j} \cdot \mathbf{t}_{i+1/2,j} \quad ,$$

where u_n and u_t are the normal and tangential velocity at the wall, and

$$\mathbf{t}_{i+1/2,j} = \frac{1}{L}\left[(x_{i+1/2,j+1/2} - x_{i+1/2,j})\mathbf{i} + (y_{i+1/2,j+1/2} - y_{i+1/2,j})\mathbf{j}\right] \quad ,$$

where

$$L = \left[(x_{i+1/2,j+1/2} - x_{i+1/2,j})^2 + (y_{i+1/2,j+1/2} - y_{i+1/2,j})^2\right]^{1/2} \quad ,$$

and, **i** and **j** are the unit vectors along the mesh directions 1 and 2. Also,

$$w^{*}_{i+1/2,j} = 0 \quad ,$$

and, the pressure $p^{*}_{i+1/2,j}$ is given by Eq. (3.2.4-2), just as for the specified-velocity boundary condition.

b. Internal Boundaries

This section describes the way in which boundaries that lie within the computational domain, called *internal* boundaries are treated in CAVEAT. In this report, all internal boundaries are treated as Lagrangian normal to the orientation of the boundary (§3.2.2). This means that the internal boundary location evolves with the problem solution.

In general, when a cell side coincides with a material interface, no special treatment is necessary. This is because the input to the Riemann problem for the material interface accounts for the separate material properties on each side for the interface. The resulting Riemann pressure and normal velocity give the (one-dimensional) behavior of the material interface (contact surface). This treatment neglects any mixing that may physically occur close to the material interface. This is one advantage of the

Godunov scheme for multi-material problems in high speed flows: material interface boundaries are handled in precisely the same manner as all other internal cell sides.

§3.2.5 Timestep Control

The criterion for the selection of the timestep Δt in CAVEAT is a multi-dimensional, ALE generalization of the well-known Courant-Friedrichs-Levy (CFL) condition. For one-dimensional, explicit, Eulerian hydrodynamics the CFL condition is

$$\max\left[\frac{(|\mathbf{u}|+c)\,\Delta t}{\Delta x}\right] < 1\,,$$

where **u** is the fluid velocity, c is the soundspeed and Δx is the mesh spacing. The maximum is taken over the entire problem domain. In simple terms the CFL condition represents the restriction that a signal carried by the largest characteristic speed in the problem cannot travel more than a single mesh spacing in a single timestep. The CAVEAT multi-dimensional generalization of this is expressed in terms of the volume

$$\max\left[\frac{\delta V}{V}\right] < \xi\,,$$

where V is the cell volume, δV is the change in the volume, and $0<\xi<1$. We define the volume swept by the largest characteristic speed

$$\delta V = \frac{\Delta t}{2}\sum_{\alpha}\vartheta_{\alpha}A_{\alpha}\quad,$$

where the summation is over the sides of the computational cell, A_α is the area of side α, and the cell-face characteristic speed ϑ_α is defined

$$\vartheta_{\alpha} = \max(c_{+}, c_{-})_{\alpha} + \left|\mathbf{w}\cdot\mathbf{n}\right|_{\alpha} + \begin{cases} 0. & \text{, Lagrangian} \\ \left|\mathbf{u}\cdot\mathbf{n}\right|_{\alpha} & \text{, non-Lagrangian} \end{cases}\quad.$$

Here, c_+ and c_- are the respectively the soundspeeds on each side of face α, **w** is the fluid velocity on one side relative to the fluid velocity on the other side of face α, and **u** is the average fluid velocity at the cell face (one-half the sum of the two fluid velocities across the face). This choice of ϑ_α leads to a conservative timestep, because it assumes the cell face is stationary. The factor of 1/2 is used in the definition of δV so that for problems in which the flow is aligned with the mesh, the one-dimensional result for the timestep is recovered. The relative velocity is included in the characteristic speed to ensure stability for calculations involving infinite-strength shocks, in which c may be zero everywhere initially, but **w** will be non-zero.

SECTION 4

COMPUTER CODE

Several objectives underlie the coding style and structure of CAVEAT. One objective was speed on a vector computer, like the Cray. This motivated the use of blocks of logically rectangular mesh which could be adjoined to form more complex geometries. Logically rectangular mesh has the advantage that offsets for neighboring cells and vertices are identical over an entire block. Therefore the cost for obtaining neighbor data from memory is minimal, and the coding is trivial. Furthermore, the code employs single loops that cover both spatial directions to enhance speed as well as simplify coding. This format led to the convention that mesh-wide arrays for cell-based quantities, vertex-based quantities, and face-based quantities all have the same dimensions even though a small number of storage locations are not utilized. Because of the need for boundary communication between blocks sharing a common boundary, *ghost* locations on all four logical sides of each mesh-wide array are also provided. Use of ghost cells also allows boundary conditions to be handled efficiently.

The reasoning just outlined led to a code with the following features:

- block structure for treating complex geometries
- logically rectangular data structures
- loops that cover both spatial directions
- arrays of equal length for cell-, vertex-, and face-data
- ghost elements for all mesh-wide arrays

In addition a modular programming style was adopted. The code is organized into numerous short subroutines, each ideally with a single obvious specific function. Common variables are split into relatively short common blocks according to their function and use. Only those commons with variables needed in a given subroutine are inserted in that subroutine. This was done to keep each subroutine as simple and short and as readable as possible. The Fortran extension of pointers is used extensively throughout the code to reduce the number of variables passed as subroutine arguments and to facilitate the treatment of multiple mesh blocks. Considerable effort was expended to modular structure, logical organization, and tho the naming of variables and subroutines to make the coding easy to read and comprehend, despite the complexity of the numerical methods that are used.

§4.1 FUNDAMENTAL CONCEPTS

This section provides a description of the fundamental concepts that form the basis for the computer code. These include the treatment of 3-dimensional space in a 2-dimensional code (§4.1.1), the division of space into discretized divisions of mesh (§4.1.2), the data structure and the conventions

for referencing data (§4.1.3), the technique used for communication between blocks of mesh (§4.1.4), and the definitions and use of the flags in the code (§4.1.5). Despite our concern for organizational simplicity and readability, the priority of computational efficiency resulted in some unavoidable complexity. Probably the main impediment to understanding the computer code is the extensive use of pointers and predefined arrays for addressing information. The use of these predefined arrays permit minimal recalculation of repeatedly used information at the expense of some readability of the code. This section will present the basis for these techniques and is a useful guide to the control variables used in most subroutines.

§4.1.1 Geometries

CAVEAT as described in this document is a 2-dimensional code. This section describes the methods used when the material motion in 3-dimensional space, because of the existence of symmetry, depends on no more than two degrees of freedom.

To describe a problem in 3-dimensional space we adopt a system of 3-dimensional coordinates ξ_1, ξ_2, ξ_3 in which initial and boundary conditions are most easily expressed. These are the "physical coordinates." The two-dimensional version of CAVEAT is applicable if the data are independent of one of these coordinates, say ξ_3. The remaining coordinates ξ_1, ξ_2 are the "independent variables" of the code. Depending on the application the "independent variables" may denote a Cartesian, radial, polar, or azimuthal coordinates.

The various two-dimensional systems are implemented by a combination of an initial shape of the mesh and the use of Cartesian or cylindrical differencing of the conservation equations (as discussed in §3.1.1a). Table 4.1.1-1 lists the various systems that are available. Close examination of Table 4.1.1-1 should eliminate a common confusion in the use and modification of CAVEAT: the interpretation of the coordinate systems (computational variables) used in CAVEAT. Internally, the computer code uses either Cartesian or cylindrical (r-θ) coordinates; these are not necessarily the same as the independent variables in the 2-D system selected, although they are related by simple trigonometric expressions.

Table 4.1.1-1.

List of the 2-dimensional systems that are available*.

2-D System	ξ_1, ξ_2, ξ_3	Independent Variables in 2-D	Computational Variables	Internal Coordinates	Type of mesh
rectangular	x, y, z	x, y	x, y	Cartesian	rectangular
r-z cylindrical	r, θ, z	r, z	r, z	cylindrical	rectangular
r-θ cylindrical	r, θ, z	r, θ	x, y	Cartesian	polar
r'-ϕ spherical	r', θ', ϕ	r', ϕ	r, z	cylindrical	polar

* The identification of the computational variables and the internal coordinates is redundant information. Note that the meaning of the variables θ and r in cylindrical coordinates is not the same as θ' and r' in spherical coordinates (see Figure 4.1.1-1). The multiblock meshes are not listed here; see §4.4.1.

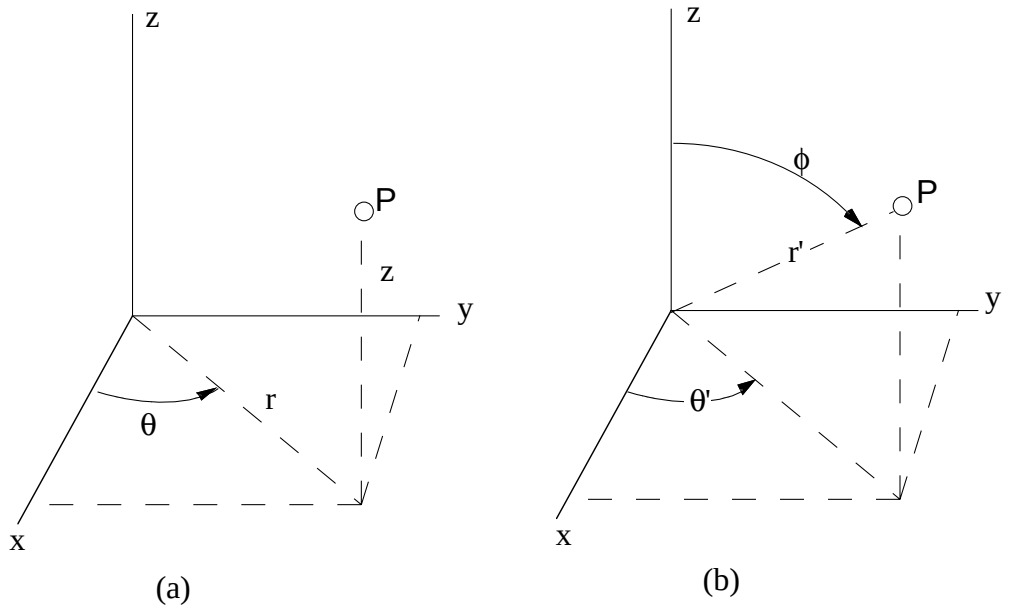


Figure 4.1.1-1. Choices of physical coordinates. (a) Cylindrical coordinates with $0 \le r \le \infty$, $0 \le \theta \le 2\pi$, $-\infty \le z \le \infty$. (b) Spherical coordinates with $0 \le r' \le \infty$, $0 \le \phi \le \pi$, $0 \le \theta' \le 2\pi$.

§4.1.2 Blocks, Parts, Cells, and Ghost Cells

The computational region is divided into subunits: *blocks*, *parts*, and *cells*, each of which is logically rectangular portions of mesh (each unit has four sides). A set of cells make up a part; a set of

parts make up a block. Blocks and cells are the main computational units in the computer code; the concept of parts is only used during the initialization of the problem. This section describes in detail the definitions and use of blocks, parts, and cells.
(Note that in the computer code, parameters are used for NB, NP, and NS, which are the maximum number of blocks, of parts, and of parts along a mesh direction for a given problem.)

Subdivision of the Computational Region into Blocks

The computational region is divided into one or more blocks. A block is a *logically rectangular* portion of the mesh, having *left* and *right boundaries* corresponding to minimum and maximum values of ξ_1 (see §4.1.1 for the definition of ξ_i). and *bottom* and *top boundaries* corresponding to minimum and maximum values of ξ_2. A block boundary may be part of the true boundary of the system, or may be an interface adjacent to another block. In certain 2-D systems, a boundary may degenerate into a single point, e.g., the center of a spherical coordinate system. If the computational region contains more than one block, the interblock boundary treatment is used (§4.1.4) to relate the data in adjacent blocks.

The relation of the ξ coordinates to the computational region is defined at initialization by the user in the input file (§4.4.1), and is determined at future times by the dynamics. In all cases, the ranges of ξ_1, ξ_2 for each block are defined independently of each other; this is a constraint on the choice of 3D reference frame. For connectivity in a multiblock description of space, blocks do not overlap and have common boundaries that span the same range of ξ_1, ξ_2. More general initial configurations can be defined by modification of the HYDINIT subroutine.

Subdivision of Blocks into Cells and Ghost Cells

CAVEAT discretizes block number IBLK by dividing the ranges of ξ_1, ξ_2, into N1(IBLK) and N2(IBLK) intervals, respectively, thus dividing the block into *cells*. The number of cells (more precisely, the number of *real cells*, excluding *ghost cells*) is N1∗N2. The cells are *logically rectangular*, that is, each cell has four *vertices* and four *faces*. Each interior face is common to two cells, and each interior vertex is common to four cells. The spatial computation of the cell vertices is stored in the mesh-wide array XV. The computation of areas, volumes, gradients, etc. is carried out in terms of XV and mesh-wide arrays representing data values assigned either to cells, faces, or vertices.

Although cell vertices are specified by two spatial variables, the cells still represent 3-dimensional volumes. When the computational variables are Cartesian, the cells have a nominal depth of $\Delta z = 1$ length units. When the computational variables are cylindrical, the cells have a nominal depth of $\Delta\theta' = 1$ radian. Moreover, the computation of cell volumes and face areas must take into consideration that one of the computational variables represents a radial variable, not a Cartesian variable (§3.1.1).

Along each boundary of a block is a layer of *ghost cells* (see Figure 4.1.2-1). The states in the ghost cells that lie on the boundary of the computational region are assigned values so as to implement various boundary conditions for the material motion (§3.2.4 and §4.4.5). Ghost cells on a boundary adjacent to another block are assigned data values appropriate to the latter block so that subroutines which implement the equations of motion may operate on one block at a time (§4.1.4).

Note that extra rows of ghost cells (see Figure 4.1.2-1), are required at the right and top sides of each block in order to provide a storage for the vertices of the first row of ghost cells on the right and bottom sides; the cell-centered values for these ghost cells are never required. Also note that the vertices at the right of these ghost cells are actually the vertices at the bottom of the ghost cells at the bottom of the block (the vertices at top of the top row of ghost cells are shared with the bottom of the bottom row of ghost cells in the next block of mesh); hence, the volume of these extra ghost cells are not meaningful.

Subdivision of Blocks into Parts

At the start of a calculation, the set of ξ_i-intervals may consist of a subset of intervals of common interval size, followed by another subset of intervals of a another common interval size, and so on. The set of cells defined by two such subsets of intervals along the two directions constitutes a *part* of a block. By providing for parts, CAVEAT allows unequal cell sizes in a single block, changes of material types, and initial conditions, and different boundary conditions for each part. Once the mesh is initialized, the information associated with the parts and the coordinates ξ_i is no longer used by the code. As a consequence, the specification of many parts in the initializations has no influence on the vectorization of the code.

The input parameters NPRTS(1,IBLK), NPRTS(2,IBLK) define the numbers of subsets in each direction (§4.4.1). Do-loops over parts may be indexed either by a pair of indices IPRT,JPRT or by a single index IP,

$$\mathrm{IP} = \mathrm{IPRT} + \mathrm{NPRTS}(1,\mathrm{IBLK})*(\mathrm{JPRT}-1),$$
$$1 \le \mathrm{IP} \le \mathrm{NPRTS}(1,\mathrm{IBLK})*\mathrm{NPRTS}(2,\mathrm{IBLK}).$$

Here, NPRTS(1,IBLK)*NPRTS(2,IBLK) is the number of parts in block IBLK. A calculation is initialized from the input file with material data which is constant over a part, but otherwise variable over the block. In principle, there may be as many parts in a block as there are cells.

Example Initial Configurations

Figure 4.1.2-2 shows several mesh configurations which are supported by CAVEAT without modification of its data structure or mesh generation subroutines.

In Figure 4.1.2-2a, a mesh on a single block is shown, along with its ghost cells. The mesh has six parts, 35 real cells, and a total of 63 cells, including ghost cells.

Figure 4.1.2-2b displays a 1-block mesh on a circle with two parts defined by unequal interval lengths in the ξ_1 direction, and with ghost cells on the outer ξ_1 boundary. Ghost cells are not shown for the ξ_1=0 boundary, which is a degenerate case, or for the ξ_2=0 and ξ_2=2π boundaries, which coincide in space.

Figs. 4.1.2-2c and 4.1.2-2d show (leaving out ghost cells) examples of mesh arrangements for several blocks in a region. Mesh lines must be continuous across the boundary from one block to an adjacent block, as required by the code. Fig. 4.1.2-2e, which shows mesh lines terminating at a boundary shared by two blocks, is not allowed in the current version of CAVEAT (§7.1.3, §7.4.1).

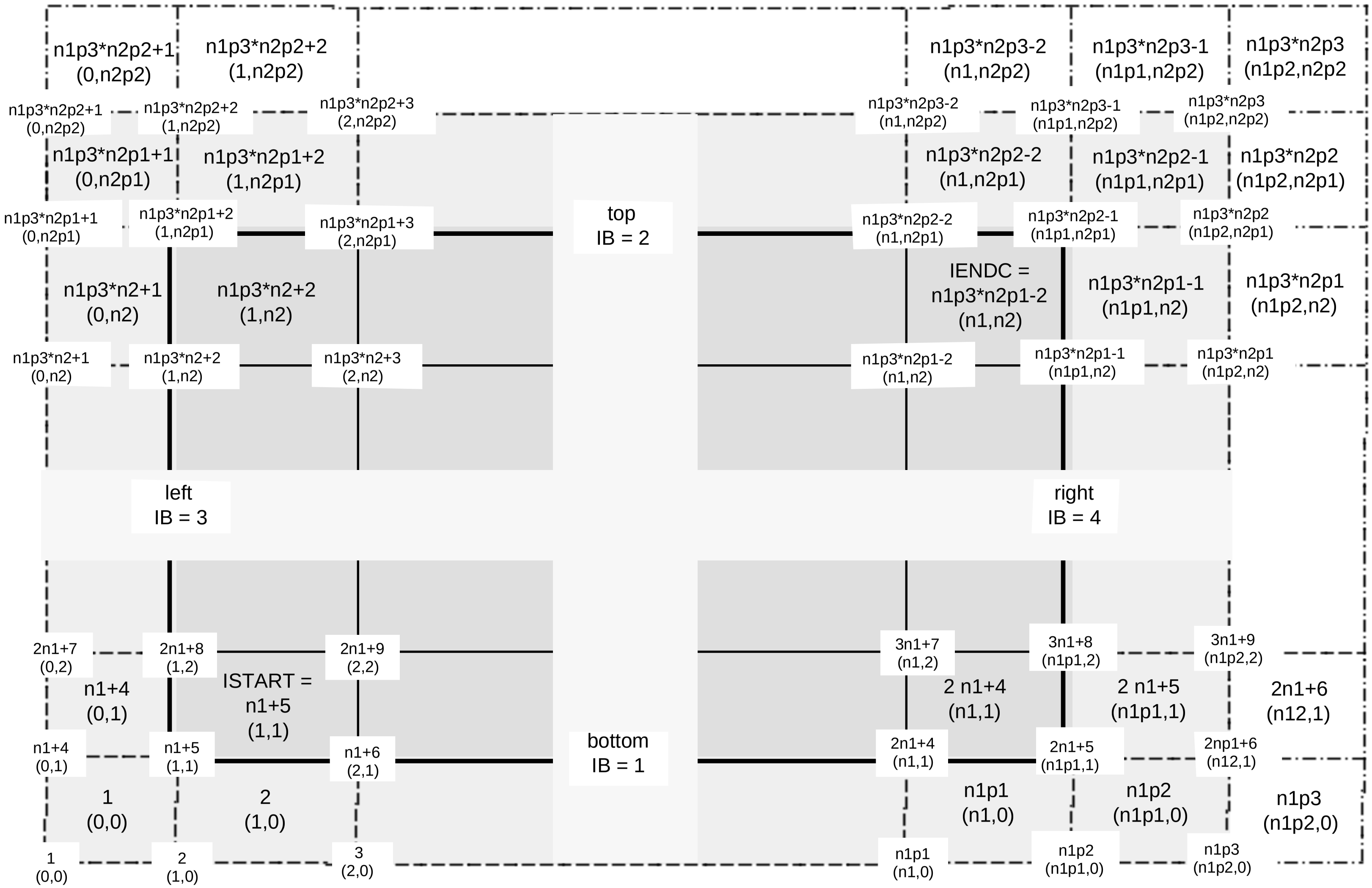


Fig. 4.1.2-1. Index notation of cells, vertices, and boundaries. Two numbers are shown for each location: the upper number is the single index notation, the lower number is the double index notations (§4.1.3b). The numbers within the cells are for the cell centers; the numbers at the intersection of the cells (in white) are for the vertices. The following abreviations are used: n1p1 = n1+1, n1p2 = n1+2, n1p3 = n1+3, n2p1 = n2+1, n2p2 = n2+2, and n2p3 = n2+3.

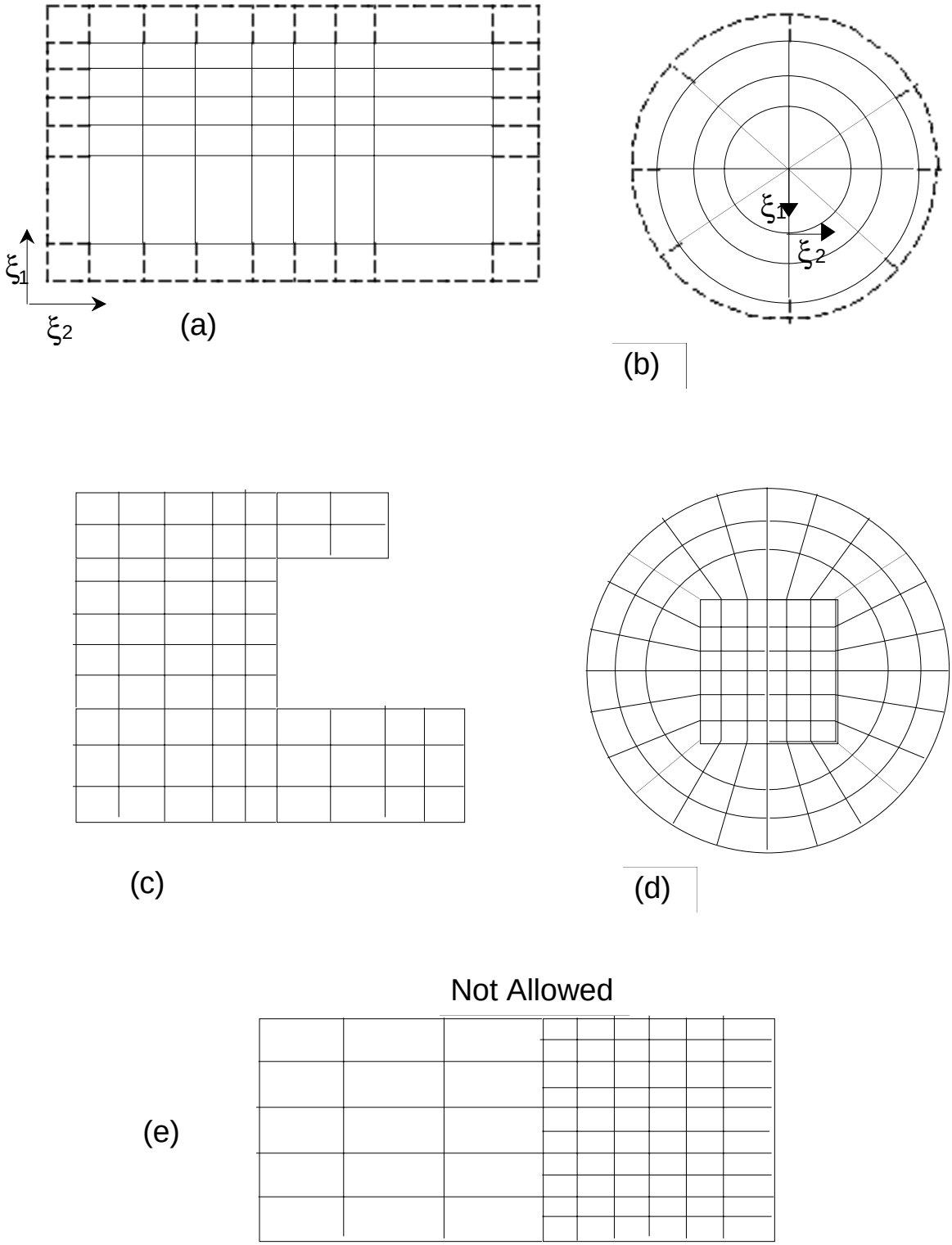


Figure 4.1.2-2. Examples of initial mesh configurations.

§4.1.3 Data Structure and Data References

This section presents the data structure and its use in CAVEAT. In particular the method that is used to reference the mesh data and determine limits of do-loops is presented; this is essential in order to interpret the inner workings of most of the subroutines. The last section describes the feature of CAVEAT that allows easy addition of mesh-wide arrays; typically this is useful for development of alternative numerical schemes or new physical models.

a. Memory Allocation and Block References (Pointers)

The executable version of CAVEAT determines the computer memory allocation at initialization for a particular problem. The memory requirements depend on the number of cells in each block and the number of blocks (§4.4.1) and the choice of equation of state (§4.4.3). Thereby, the code does not need to be recompiled to run problems of different sizes, and the memory utilized to run a problem is minimized. The one exception is if the number of blocks, the number of parts, or the number of parts along a mesh direction in the initial setup of a problem exceeds the values of the parameters NB, NP, or NS, respectively. Consequently, the appropriate parameter must be increased and the code recompiled; error messages will alert the users if this setup error occurs (§4.5f).

CAVEAT requires access to the SESAME tables for each SESAME equation of state. The SESAME tables are loaded into memory above the computer program and common blocks (see Fig. 4.1.3a-1) at execution time. The subroutine SESSET extends the run-time memory of CAVEAT by the size stored in the variable NWSESTB and loads the SESAME tables into the increased memory.

Also, during the initialization of CAVEAT, the subroutine PNTRSET is called once to calculate the needed memory for the mesh variables, to allocate the additional memory by a system call MEMADJ (see §C.3), and to define a pointer for each mesh-wide variable (see Fig. 4.1.3a-1). A pointer is a location in memory that FORTRAN recognizes and is used in CAVEAT to locate the beginning of a mesh-wide array. Table 4.1.3a-1 contains the names for these *absolute* pointers, the corresponding variables, and sizes. The modifier *absolute* indicates that once calculated at initialization they are never changed (compare with *block* pointers discussed below). Note that the mesh variables are grouped in memory by blocks and then by component if they are a vector, as indicated in Fig. 4.1.3a-1 for the mesh variable XV.

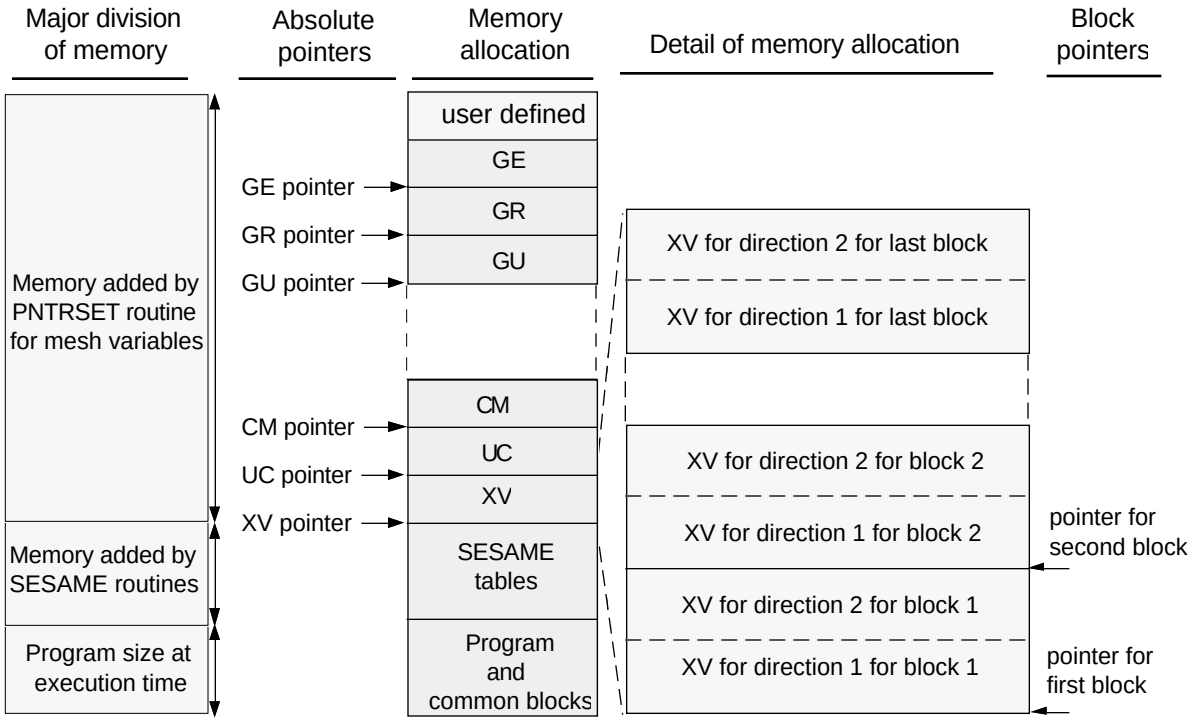


Figure 4.1.3a-1. The allocation of memory in CAVEAT. The location of the *absolute* pointers is also shown. The *block* pointers point to the beginning of the variable for each block, as illustrated at the right, and are reassigned as each block is processed.

Table 4.1.3a-1

Absolute and block pointers and their corresponding variables and sizes.

Absolute Pointer	Block Pointer	Variable Name	Size*	Absolute Pointer	Block Pointer	Variable Name	Size*
MCM	ICM	CM	1	MRH	IRH	RHO	1
MFA	IFA	FA	2	MRV	IRV	RAV	1
MFN	IFN	FN	4	MSI	ISI	SIE	1
MFX	IFX	FLUX	2	MSS	ISS	SS	1
MGE	IGE	GE	2	MTE	ITE	TE	1
MGP	IGP	GP	2	MTL	ITL	TEL	1
MGR	IGR	GR	2	MUC	IUC	UC	2
MGU	IGU	GU	4	MUF	IUF	UF	2
MIE	IIE	IEOS	1	MUL	IUL	UCL	2
MIF	IIF	IFLG	1	MUM	IUM	UM	2
MPF	IPF	PF	2	MVO	IVO	VOL	1
MPR	IPR	PR	1	MXC	IXC	XC	2
MRA	IRA	RA	1	MXV	IXV	XV	2

* The size of the variables is given in terms of the multiple of the mesh size.

During the execution of the program, the *driver* subroutines that call other subroutines generally contain loops over the blocks in the problem (e.g., HYDROCYC). The first subroutine called in these loops over blocks, PNTRBLK, calculates the the pointers for each mesh-wide variable for the block being processed (see Fig. 4.1.3a-1); these *block* pointers are passed to the rest of the subroutines in the loop through the common block PNTR2 and are reassigned for each block. The use of PNTRBLK greatly simplifies the subroutines and eliminates the need to pass the mesh variables for each block in the calling arguments.

b. Mesh Indexing

CAVEAT uses both *double indexing* and *single indexing* for the enumeration of cells, vertices, and faces. The former is more convenient for user input and for start-up routines that define the initial configurations. The later is more efficient for maximizing the lengths of vectorized do-loops and for the pointer method of referencing large arrays.

Double Indexing

The cells of a single block are indexed as (I,J) or as (I1,I2). The index ranges are $0 \le I1 \le N1+1$, $0 \le I2 \le N2+1$, where $I1=0$ and $I1=N1+1$ refer to ghost cells on the left boundary and right boundary, respectively, of the block, and $I2=0$ and $I2 = N2+1$ refer to ghost cells on the bottom boundary and top boundary, respectively. In Figure 4.1.2-1 the double index for cells and vertices is given for an arbitrary block. In some subroutines $I1=1$ is taken to be the first row of ghost cells so that the index ranges are $1 \le I1 \le N1+2$, $1 \le I2 \le N2+2$. The subroutines that use this convention generally are not the initialization routines, but the highly vectorized, hydrodynamics routines; see the next section. *Care must be taken to identify which convention is used within a subroutine.*

The vertices are also indexed by (I,J) or (I1,I2). Vertex (I1,I2) is defined as the vertex at the bottom, left corner of cell (I1,I2). The index ranges for vertices are $0 \le I1 \le N1+2$, $0 \le I2 \le N2+2$. The vertex ranges are one unit longer than the cell ranges in order to account for the right vertices of the right-border ghost cells and the top vertices of the top-border ghost cells.

Finally, the face which serves as the bottom face of cell (I1,I2) is labeled (I1,I2,1) and the face which serves as the left face of cell (I1,I2) is labeled (I1,I2,2). The ranges of face indices I1,I2 are the same as the ranges of vertex indices for similar reasons.

Single Indexing

For block IBLK, let the double array (I1,I2) with ranges $1 \le I1 \le N1(IBLK)+3$, $1 \le I1 \le N2(IBLK)+3$ be ordered as a single sequence. See Figure 4.1.2-1 for a comparison of the two notations for an

arbitrary block. Note that we use the alternative definition of (I1,I2) that includes the ghost cells (see the previous section); single indexing is commonly used in highly vectorized hydrodynamics subroutines which include the ghost cells. The parameter MSZ(IBLK) = (N1(IBLK)+3)*(N2(IBLK)+3) gives the total number of elements in the sequence. These elements can be labeled by a single index I or IJ in the range $1 \le I \le MSZ(IBLK)$. The relation between the single index and the double indices is:

I = I1 + I2 * (N1+3).

Each cell and vertex of block IBLK can now be identified with a unique single index I as well as with an index double (I1,I2). The bottom face of a cell can be identified uniquely as (I,1), and the left face can be identified uniquely as (I,2). Not all values of I in the allowed range necessarily denote a physical or ghost cell, however, because the entire range of I includes cells that are not used.

The sequences of length MSZ(IBLK) for the different blocks can now be lined up in a single master sequence (see §4.1.3a and Figure 4.1.2-1) for the entire region. Define parameter MOB by:

MOB(1) = 0,

MOB(IBLK) = sum over all MSZ(I) for I < IBLK.

Then MOB(IBLK) serves as the *offset* or *starting location* in the computer memory of scalar variables defined on cells or on vertices of the mesh for block IBLK. The length of the master sequence is MOB(NBLKS) + MSZ(NBLKS).

Both double indexing and single indexing are laid out in some detail in Figure 4.1.2-1, especially for the four corner areas of the mesh on a block. This figure, and the code as well, use the abbreviations

N1P1 = N1(IBLK) + 1, N2P1 = N2(IBLK) + 1,

N1P2 = N1(IBLK) + 2, N2P2 = N2(IBLK) + 2,

N1P3 = N1(IBLK) + 3, N2P3 = N2(IBLK) + 3.

c. Do-Loop Ranges and Cell/Vertex/Face Offsets

To systemize the specification of the many do-loops in CAVEAT over cells, vertices, and faces, a set of initial and final index arrays are defined by

IFIRST(IBLK)= N1P3 + 2,

LASTV(IBLK) = N1P3*N2P3 -1,

LASTC(IBLK) = N1P3*N2P1 -2,

MSIZV(IBLK) = N1P3*N2P1 -2,

MSZ(IBLK) = N1P3*N2P3.

Then with cells and vertices indexed by I in the single index scheme, do-loops over cells and vertices have the ranges (compare with Figure 4.1.2-1)

IFIRST ≤ I ≤ LASTC	for real cells,
1 ≤ I ≤ LASTV	for all cells (real and ghost),
IFIRST ≤ I ≤ LASTV	for real vertices,
1 ≤ I ≤ MSZ	for all vertices (real and ghost).

The do-loop ranges for block elements do include some ghost elements. The extra work by the code is acceptable in order that the largest possible do-loops for vectorization are obtained, and this does not conflict with use of ghost data when it is needed.

Do-loops are also needed for calculations on block elements along block boundaries. The boundaries are indexed by IB = 1,2,3,4 for bottom, top, left, and right. The arrays I1BC(IB,IBLK), I2BC(IB,IBLK), I3BC(IB,IBLK) provide, respectively, initial and final index values, and strides for do-loops along boundary IB. Depending on whether the loop is over cells, vertices, or faces, some offset to I1BC or I2BC may be required, but does not require definition of additional arrays.

Another necessary bookkeeping chore for mesh-wide arrays is to locate block elements neighboring on a cell or vertex at , say, a base index I. Indices of such elements are expressible as I plus some offset. This motivates these array definitions:

ICV(1,IBLK) = 0,	IFV(1,IBLK) = 1,
ICV(2,IBLK) = 1,	IFV(2,IBLK) = N1P3,
ICV(3,IBLK) = N1P3 + 1,	ICN(1,IBLK) = - N1P3,
ICV(4,IBLK) = N1P3,	ICN(2,IBLK) = 1.

The ICV(K,IBLK) are the offsets from cell I to the four vertices of that cell. Their negatives are the offsets from vertex I to its four cells of that vertex. The IFV(M,IBLK) are the offsets from face I,M to the adjoining vertex of that face (to the right for M=1, to the top for M=2). The ICN(M,IBLK) are the offsets from cell I to the cell on its bottom boundary for M=1, or on its left boundary for M=2.

Additional do-loop ranges and offsets appear in the CAVEAT code. With the above as a guide and with reference to Fig. 4.1.2-1, the interpretations of these ranges and offsets should not be difficult.

d. User-Defined Work Arrays

NWK(I) [defaults: 0]. Some modifications to the code by users require additional memory for variables. CAVEAT provides up to 20 additional arrays by specification of the input parameter NWK(I). Each array has an absolute pointer MWK(I) and a block pointer IWK(I). If NWK(I) is a positive integer, then the number of real numbers allocated in memory is NWK(I). If NWK(I) is a negative integer, then the

number of real numbers allocated in memory is the total mesh size times (-NWK(I)); this is convenient for mesh-wide variables. As a further convenience, when NWK(I) is negative, the routine PNTRBLK computes the pointer IWK(I), the beginning address in block IBLK for the array in a manner that is similar to other mesh-wide variables (see PNTRBLK in §4.1.3a).

These work arrays can be made available in any subroutine by including the common that has the pointer definitions, PNTR1 or PNTR2, and a pointer statement that defines a variable name to be associated with the pointer.

For example, if NWK(1)=-1 and NWK(2)=-2 are included in the input file, then a subroutine that (a) has IBLK included in the argument list, (b) includes the commons MESH1 and PNTR2, and (c) includes the statements

POINTER (IWK(1), *RWK*(1))
POINTER (IWK(2), *SWK*(MSZ(IBLK),2))

can use variable *RWK*(IJ) in a manner similar to the mesh-wide RHO(IJ) and variable *SWK*(IJ,M) in a manner similar to UC(IJ,M).

§4.1.4 Interblock Communication

The computational domain in CAVEAT can be divided into logically rectangular blocks of mesh. For many problems, one block will suffice. For more complicated geometries, several blocks can be linked together in logical space by boundaries that are completely transparent to the flow (see Fig. 4.1.2-2 for various examples). If multiple blocks are used in a problem, the interblock boundary condition can be thought of as being similar to usual reflective or inflow boundary conditions (see, for example, §4.4.5), except that the adjoining block provides information to the ghost cells.

The interblock boundary conditions are implemented by loading the ghost cells with the required information, as is done for all other boundary conditions. Because neighboring blocks adjoin in space, there will be overlap of ghost cells in logical space (see Fig. 4.1.4-1 and §4.1.2). By applying boundary conditions through the ghost cells that overlap at the common sides of the blocks, we insure that block boundaries that were originally in contact will remain so.

Blocks that are adjacent in space are required to communicate information or else continuity will be destroyed. The user must indicate in the input file how the blocks are to be arranged in logical space. One necessary input array for multiblock problems is **NBC**, which is dimensioned as NBC(2,4,NB) where NB is greater than or equal to NBLKS; the other necessary input variable is **IBC** (§4.4.4). The NBC array contains two types of information. A nonzero value of NBC(1,IB,IBLK) indicates that boundary IB of block IBLK is in communication with another block and is equal to that block number. The boundary number in block NBC(1,IB,IBLK) that IB is communicating with is provided by the user in location NBC(2,IB,IBLK). Thus, there is a reciprocity that must be satisfied within the NBC array. That is, if we denote NBC(1,IB,IBLK) by JBLK and NBC(2,IB,IBLK) by JB, then NBC(1,JB,JBLK) must equal IBLK and NBC(2,JB,JBLK) must equal IB. If NBC is not specified consistently, an error message is reported at initialization (§4.5.6).

Successful block communication relies on the sharing of three types of information during program execution: the face array UF, the vertex array XV, and several cell-centered arrays (such as pressure and density). CAVEAT utilizes the existing ghost-cell structure in order to share these types of information; hence it is only through the boundaries that one block knows that another is adjacent. The current implementation imposes the restriction that the number of cells and initial cell spacing on both sides of the communicating boundary must be the same. For each communicating boundary, offsets and strides in the global region-wide arrays are defined in subroutine BCOMSET so that when block communication is necessary, the array locations that mark the cells that must be processed are immediately available and do not need to be recalculated every time step. For example, the array MOC(IB,IBLK) contains the starting location in *global* array space of the cell-centered arrays along

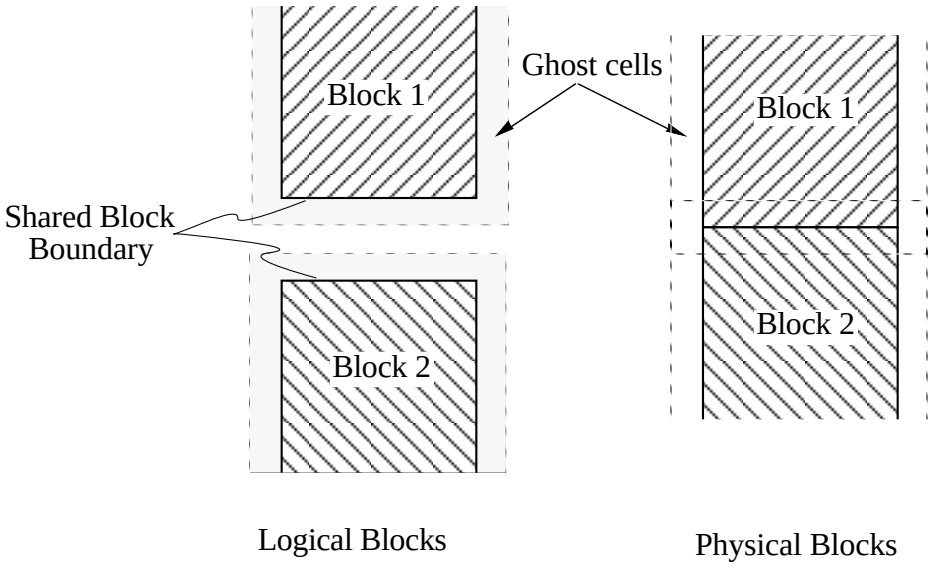


Fig. 4.1.4-1. Two Blocks with a interblock boundary. The illustration of the physical blocks is as the blocks would appear in space. The illustration of the logical blocks shows the blocks separated, illustrating the ghost cells that surround the blocks.

boundary IB and MS1(IB,IBLK) contains the mesh stride for that boundary (see Fig. 4.1.4-2). Thus for boundary 1 of block 1, MOC(1,1) = 0 and MS1(1,1) = 1 while MOC(3,1) = 1- (N1(1) + 3) and MS1(3,1) = N1(1) + 3. This convention allows the ghost cells of block 1 to be filled by simply starting at the initial location (MOC) and successively adding the mesh stride (MS1) until all ghost cells on boundary IB have been accessed. The locations to be filled, called acceptor cells in the acceptor block, for boundary 4 of block 1 span from MOC(4,1) + MS1(4,1) to MOC(4,1) + (N2(IBLK) + 2) * MS1(4,1).

Offsets for other blocks must be augmented to leap past the storage locations for previous blocks. An array MOB(IBLK) is defined such that

$$\mathrm{MOB(IBLK)} = \sum_{IBLK'=1}^{\mathrm{IBLK-1}} \mathrm{MSZ}(IBLK') \quad .$$

Therefore, MOB(1) = 0 and MOB(3) = MSZ(1) + MSZ(2). Furthermore, MOC(1,3) = MOB(3) and MOC(4,3) = MOB(3) - 1.

Starting locations for face (MOF) and vertex (MOV) quantities are not necessarily the same as those for cell-centered arrays but strides for stepping along the boundary are the same. This occurs because, for instance, the relevant vertices along the top and right boundaries are on the right-hand side of the ghost cell (see Fig. 4.1.4-2). Thus, MOV(4,1), the offset for vertex quantities on boundary 4 of block 1 is 0 while MOC(4,1) = -1. Note, however, that MOV(4,3) will equal 2 x MOB(3) because vertex positions are double-wide arrays. Likewise, because cell-centered velocities also have two components, we add another MOB(IBLK) to MOC(IB,IBLK) when evaluating the starting acceptor location.

Likewise, donor starting locations and strides, which are primarily real cells (see Fig. 4.1.4-3), are also determined at initialization in similar fashion. The indices IB and IBLK in LOC(IB,IBLK) (the starting donor location in global space) and LS1(IB,IBLK) (the stride on the donor boundary of the donor block) always refer to the *acceptor* block, not to the donor block itself. Thus, if NBC(1,1,1) = 2 and NBC(2,1,1) = 3, then LOC(1,1) = MOB(2) - (N1(3) + 3) + 2 and LS1(1,1) = N1(3) + 3 (see Fig.

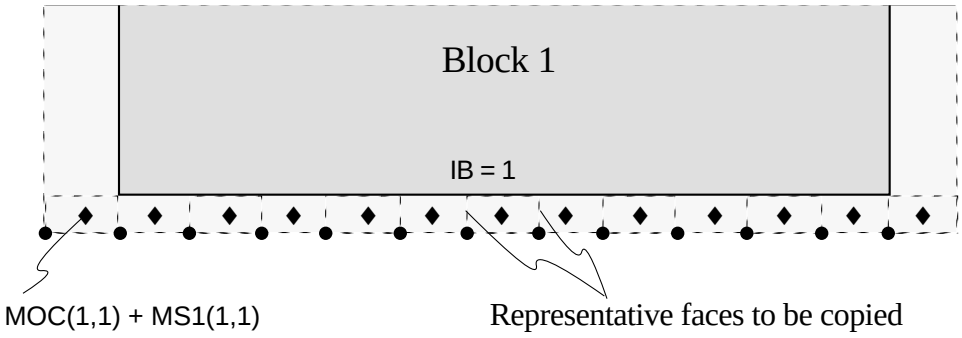


Fig 4.1.4-2. Schematic showing logical locations of information to be copied. The symbol ♦ locates the cell-centered values that are copied. The symbol • locates the vertices to be copied. All the faces, similar to the representative faces, are also copied.

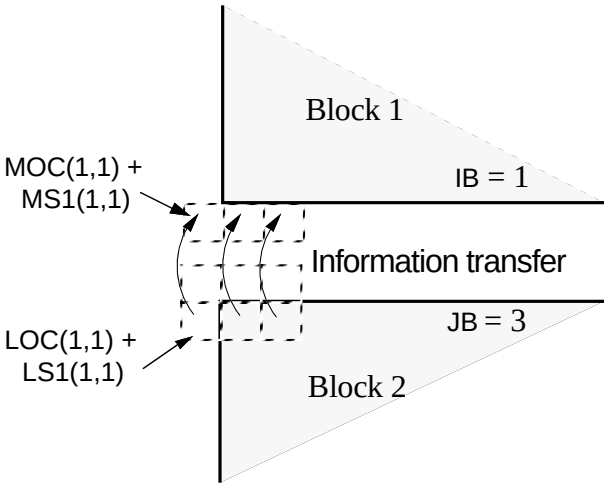


Figure 4.1.4-3. An example showing representative cells in the donor block (Block 2) and acceptor block (Block 1) with schematic of information transfer between real cells in the donor and ghost cells in the acceptor.

4.1.4-3).

Boundary communication must be completed during certain phases of the calculation. After the determination of face velocities in the Lagrangian phase (§4.3.1), the face values normal to the communicating boundary must be copied from the neighboring block (see Fig. 4.1.4-2). Before remapping, updated cell-centered values are required. Modifications in XV by mesh movement during Lagrangian, remap, and rezone phase must also be recognized by the acceptor block. In each of these circumstances, other boundary conditions are completed for each block individually, then the appropriate boundary communication routine is called (BCOMHYD, BCOMUF, BCOMXV) for each block, making sure that all storage locations of the variable to be communicated are available. Then, in a loop over boundaries, the ghost cells for all boundaries that are communicating for each block are loaded with the appropriate donor region values.

CAVEAT allows four types of block communication (Fig. 4.1.4-4): (a) simple, side-by-side boundary contact, (b) four blocks meeting at a point, (c) three blocks coming together in an L-shaped configuration, thereby leaving a void, and (d) three blocks meeting at a point. Each of the last three cases requires some special consideration.

For example, in case (b), care must be taken so that the ghost cells for each block that coincide with the real cells in blocks diagonally opposite in logical space are properly loaded. This is accomplished by testing whether the flag I4BK(IB,IBLK) is equal to one for boundary IB. In order to determine whether this flag should be set, rotate the block until IB is on the logical bottom of the block; if the left-hand edge of IB *in this orientation* contains a ghost cell to be processed in a four-block meeting point, then I4BK(IB,IBLK) equals 1. Special acceptor (MAC(IB,IBLK), MAV(IB,IBLK)) and donor (MDC(IB,IBLK), MDV(IB,IBLK)) are loaded in BCOMSET to quickly access the acceptor ghost cell and the diagonally opposite real cell in the donor block (IDONR(IB,IBLK)).

As another example, we will examine case (c) (Fig. 4.1.4-5). Here three blocks meet and leave a void, similar to case (b) but with one of the four blocks absent. Here, in addition to treating the cell-centered and vertex arrays for the diagonally opposite blocks as above, the face velocities and vertex positions noted in the figure must also be copied. The array I3VD(IB,IBLK) will equal one if the left-hand edge of IB, referenced as above, requires special treatment. If a fourth region were present, the usual copying between blocks would have handled this situation with no extra effort. Again, special acceptor and

donor arrays for vertex positions (MAVB, MDVB) and face velocities (MAF, MDF) have been established in BCOMSET to easily access these special faces and vertices.

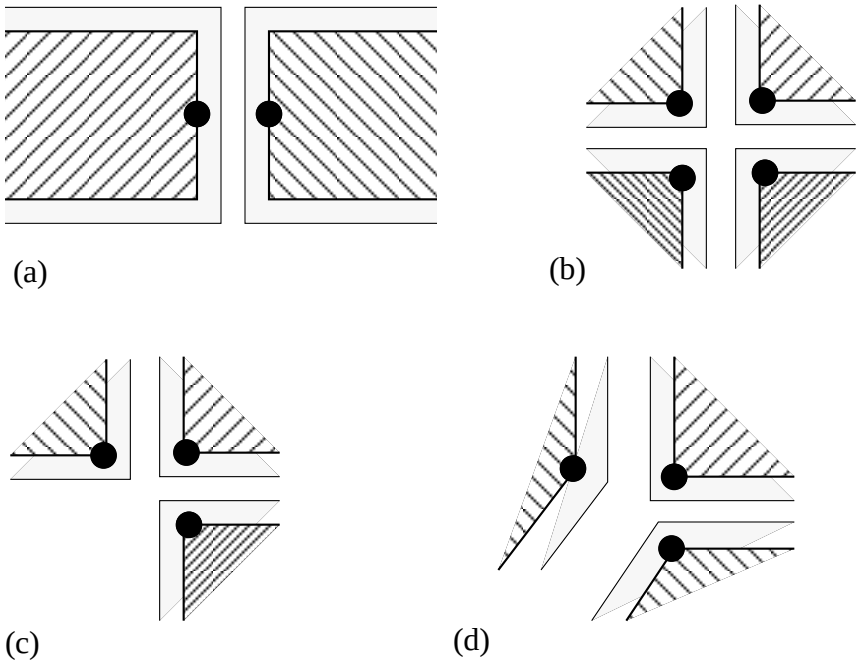


Fig. 4.1.4-4. Four different types of interblock communication supported by CAVEAT. The symbol ● indicates the vertices in each block that are spatially coincident. The objects shaded with lines represent corners of blocks; the objects uniformly shaded are the ghost cells that surround the blocks.

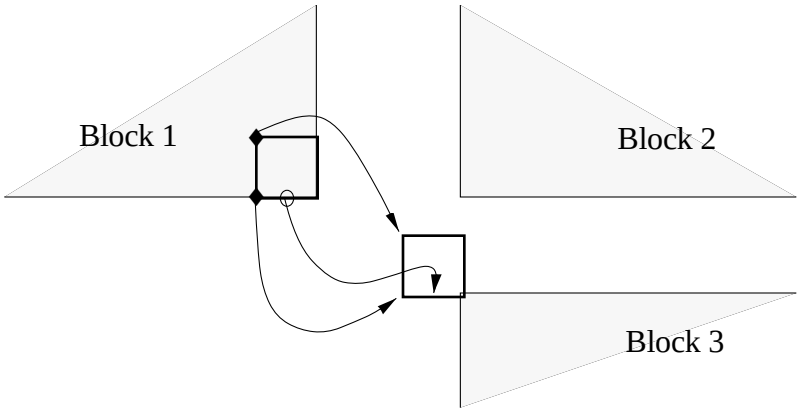


Fig. 4.1.4-5. Three regions meeting with a void space. Copying procedure is shown for the extra vertices (♦) and faces (○) from block 1, the donor, to block 3, the acceptor.

§4.1.5 Flags - Cell Information in a Compacted Form

CAVEAT uses a mesh-wide array, IFLG, as flag storage. The flags consist of patterns of bits in the IFLG word, whose presence or absence indicates certain properties belonging to the cell, cell faces, or cell vertices. The use of flags is a convenient technique to reduce the amount of memory required to store binary or limited information about each cell in the mesh.

The flags for each cell are set upon initialization in the subroutine FLAGSET. Throughout the execution of CAVEAT, the flags are used in conjunction with predefined masks as illustrated below. These masks are available in the common block MASKS, and begin with the letters MSK, and are initialized in the subroutine FLAGSET. In Table 4.1.5-1 the name of each mask, its bit location(s), and its use are given. Figure 4.1.5-1 indicates the locations in the cell that the masks are defined.

The information contained in IFLG for a given cell is extracted with logical operators. Logical functions AND, OR, and NOT are used in conjunction with the masks in Table 4.4.4-1 to test or modify bits of interest. The following examples illustrate the techniques.

To test the contents of a flag

The results of a test of a flag is usually used in a FORTRAN IF test (or a CRAY vector merge test (Appendix C.1)). For a single bit flag, the argument of the IF might have the form:

AND(IFLG(IJ) , *MSKXXX*) .NE. 0 .

The logical AND extracts the correct bit corresponding to *MSKXXX*, and if it were one or *on*, then the test would return TRUE. The material number stored in IFLG is extracted in the same manner but is used differently. To obtain the material number associated with cell IJ, use a statement of the form:

MATNUM0 = AND(IFLG(IJ) , MSKMAT)

where *MATNUM0* is the desired material number.

To change a flag

To set a flag in the IFLG array, one simply uses the logical OR to set the appropriate bit:

IFLG(IJ) = OR(IFLG(IJ) , *MSKXXX*)

This sets the bit associated with the mask *MSKXXX* in IFLG(IJ). To zero out a flag in the IFLG array, the mask

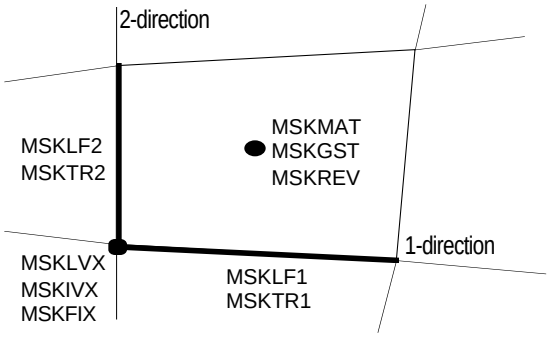


Fig. 4.1.5-1. Illustration of the locations within a cell of the information contained in the various flags.

is changed to a reversed mask using a logical NOT operation and a logical AND is used to *turn off* the specific bit:

IFLG(IJ) = AND(IFLG(IJ) , .NOT. *MSKXXX*)

The examples above are combined to change a material number in IFLG:

```
IFLG(IJ) = AND( IFLG(IJ) , .NOT. MSKMAT ) +
           AND( MATNUM0 , MSKMAT )
```

where the first logical AND zeros out the old material number in IFLG(IJ) and the second use of AND insures the new material number, *MATNUM0*, does not make any changes in IFLG(IJ) except in the bits reserved for the material number.

Table 4.1.5-1

List of the masks, their bit location, and use*.

Name	Bits	Use
MSKMAT	1-6	Material number for cell (§4.4.3); possible values are 0-30.
MSKGST	7	Indicates a ghost cell (§4.1.1b).
MSKLVX	8	Indicates the vertex at the bottom-left corner of the cell is a Lagrangian point; i.e., it moves with the fluid velocity (§3.2.2). Typically these points are intersections of material interfaces. Note MSKLVX and MSKIVX are never both *on*.
MSKLF1	9	Indicates the bottom face (along mesh direction 1) of the cell lies along a material interface (§3.2.2 and §4.3.2), and consequently no fluxing of material can occur through this face.
MSKLF2	10	Same as MSKLF1 except for the left cell face (along mesh direction 2).
MSKIVX	11	Indicates the bottom-left vertex of the cell lies along a material interface and can be tangentially rezoned (§3.2.2 and §4.3.2); how it is rezoned is determined by MSKTR1 and MSKTR2. Note MSKLVX and MSKIVX are never both *on*.
MSKTR1	12	Indicates the bottom-left vertex of the cell lies along a material interface and can be tangentially rezoned along the 1-mesh direction (§3.2.2 and §4.3.2).
MSKTR2	13	Same as MSKTR1 except for the 2-mesh direction.
MSKFIX	14	Indicates the bottom-left vertex of the cell remains fixed in space; used in the tangential rezone treatment for the vertex at the intersection of certain boundaries (§4.3.2f).
MSKREV	15	Indicates that the cell is no longer in the *ramp* region (§2.2) and is in the tabular portion of the equation of state (§2.2 and §4.4.3b).

* The bit location is taken from the lowest position in the word (right side). In all cases, except for MSKMAT, the masks are a single bit; if the bit is set, then the situation under its use applies; if the bit is not set, then the situation does not apply. This information, as represented by the masks, is set for every cell in the mesh and stored in the mesh-wide array, IFLG.

§4.2 SUBROUTINE ORGANIZATION AND FLOW OF INFORMATION

The CAVEAT computer program is divided into two groups of subroutines. The first group of subroutines contains the fundamental algorithms inherent to the arbitrary Lagrangian-Eulerian, Godunov formulation. The organization and naming conventions of subroutines included in this section of the computer code will be addressed herein. The graphics subroutines are contained in the second group. A description of the graphics and systems routines will not be provided because they are computer system dependent and a familiarity of this group is not necessary for an understanding of the CAVEAT computer code.

Subroutines are ordered alphabetically in the CAVEAT code. Subroutine naming conventions indicate the functions of each subroutine. Furthermore, the first three or four characters of each name provide references to common subroutine functions. Consequently, subroutines associated with related functions are grouped together in the listing. They include:

ADV	-	remapping (advection) related algorithms
BCOM	-	general inter-block communication
BND	-	routines that load boundary data into the computational arrays
CRNR	-	inter-block communication for computational cells which lie at the intersection of three (3) blocks
DUMP	-	routines that access the dump files
FACE	-	computation of cell face (side) parameters
GRAD	-	calculation of cell-centered gradients
LAG	-	calculation of material properties for the Lagrangian step of the algorithm
NEW	-	setup of a new problem
PNTR	-	setup of the pointer arrays for a block
REZ	-	mesh rezone algorithms
RIEM	-	routines that calculate cell face properties from the Riemann solution
SES	-	SESAME equation-of-state subroutines.

The CAVEAT subroutines may be grouped into six global functions. This division is somewhat arbitrary and is used in an attempt to provide clarity. The subroutines of the first group access the *Input* files and *Initialize* the computational data arrays. The next four groups of subroutines are required to advance a calculation one complete time cycle. They include the *Lagrangian, Rezone, Remap,* and *Update* steps. In the Lagrangian phase of the cycle, the variables are advanced assuming the computational mesh moves with the material velocity. A new mesh is determined in the Rezone phase and the variables are mapped onto this mesh in the Remap phase. Finally, cell intensive and geometrical properties are computed in the *Update* phase thereby completing a calculation cycle. Subroutines that provide *Output* compose the last group.

Flow diagrams that detail the calling sequence of the CAVEAT subroutines are provided in Figs. 4.2.4-1 through 4.2.4-8. Utility, systems, and graphics subroutines have not been included in

these figures. Major loops or logical branches are indicated only in high level or driver subroutines. These simplifications have been made in an effort to provide a clear representation of the computational logic.

In Figs. 4.2.4-1 through 4.2.4-8, subroutines that are detailed in later figures are indicated by dashed boxes. Loops over the block data structure also are indicated in the figures. In these loops, the subroutine PNTRBLK first is called to update the variable pointers then the appropriate subroutines are accessed to modify the block information (§4.1.3a).

The calling sequence of subroutines in the main program (CAVEAT) are provided in Fig. 4.2.4-1. The association of subroutines with the six groups discussed above also are provided in Fig. 4.2.4-1. The fundamental loop that advances the problem state one computational cycle is indicated. In Figs. 4.2.4-1 and 4.2.4-4, references are made to the utility GAS. This graphics utility is discussed in §4.5.5.

The flow diagrams for the initialization subroutines NEWPROB and RESTART are provided in Figs. 4.2.4-2 and 4.2.4-3. NEWPROB is accessed when input data is available only from the input file IN2D (§4.4). If input data also is available from a dump provided by a previous CAVEAT calculation, the file RS2D (§4.4) also must be read. The subroutine RESTART is utilized for this later situation.

Values for the variables located on the faces of each cell are obtained by solving a Riemann problem. The subroutine RIEMDRV accesses the routines that are necessary for obtaining Riemann solutions. The flow diagram for RIEMDRV is provided in Fig. 4.2.4-4.

A new mesh is obtained in the subroutine REZONE. Values of the cell-centered variables are then remapped (advected) onto this new mesh. Control of the rezoning and remapping algorithms is provided by the subroutine REMAP. Flow diagrams for the subroutines REMAP and REZONE are provided in Figs. 4.2.4-5 and 4.2.4-6, respectively. Two loops shown in Fig. 4.2.4-5, one on the number of passes and one on direction, require further explanation (§3.2.3 for a detailed discussion). The advection algorithm involves an operator splitting technique to achieve a second order treatment in time (corner coupling across cells). Consequently, there is a loop on the advection direction in Fig. 4.2.4-5 resulting in an advection operator of the form

$$L(\Delta t) = L_x(\Delta t)\, L_y(\Delta t)$$

The next time step the directions are reversed in order to obtain a symmetric algorithm. If the user requires four or more Lagrangian steps for every remap, then two additional passes are made through the advection algorithms in order to obtain a symmetric algorithm. The resulting advection operator then takes the form

$$L(\Delta t) = L_x(\Delta t/2)\, L_y(\Delta t/2)\, L_y(\Delta t/2)\, L_x(\Delta t/2)$$

Following the completion of the Lagrangian, rezone, and remap steps of the calculation, the geometry variables are updated, the equation-of-states are accessed to update the pressure and sound speed, boundary conditions are updated, and a new time step size is determined. Control of the

subroutines that update these remaining variables is provided by the routine NEWCYC. A flow diagram of NEWCYC is provided in Fig. 4.2.4-7.

Control of the CAVEAT output is provided by the subroutine HYDROOUT, which is outlines in Fig. 4.2.4-8. Six media for output are indicated in Fig. 4.2.4-8. They include output to the terminal, a print file, two graphics files, a movie file, and a dump file. Currently, the movie file is not implemented. Further discussion of the CAVEAT output files is provided in §4.5.

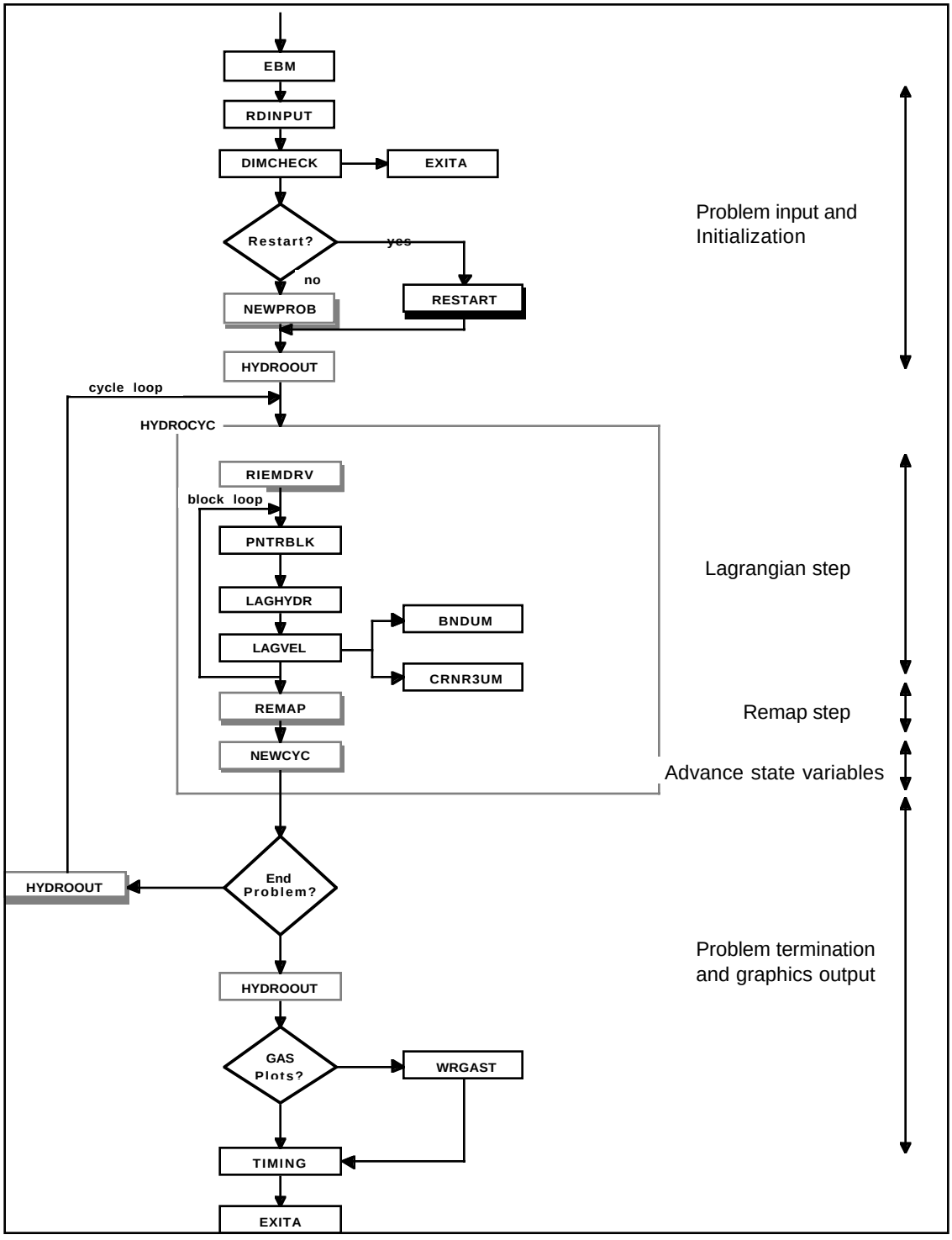


Figure 4.2.4-1. Calling sequence for CAVEAT. Subroutines that are highlighted are detailed in a separate figure that follows.

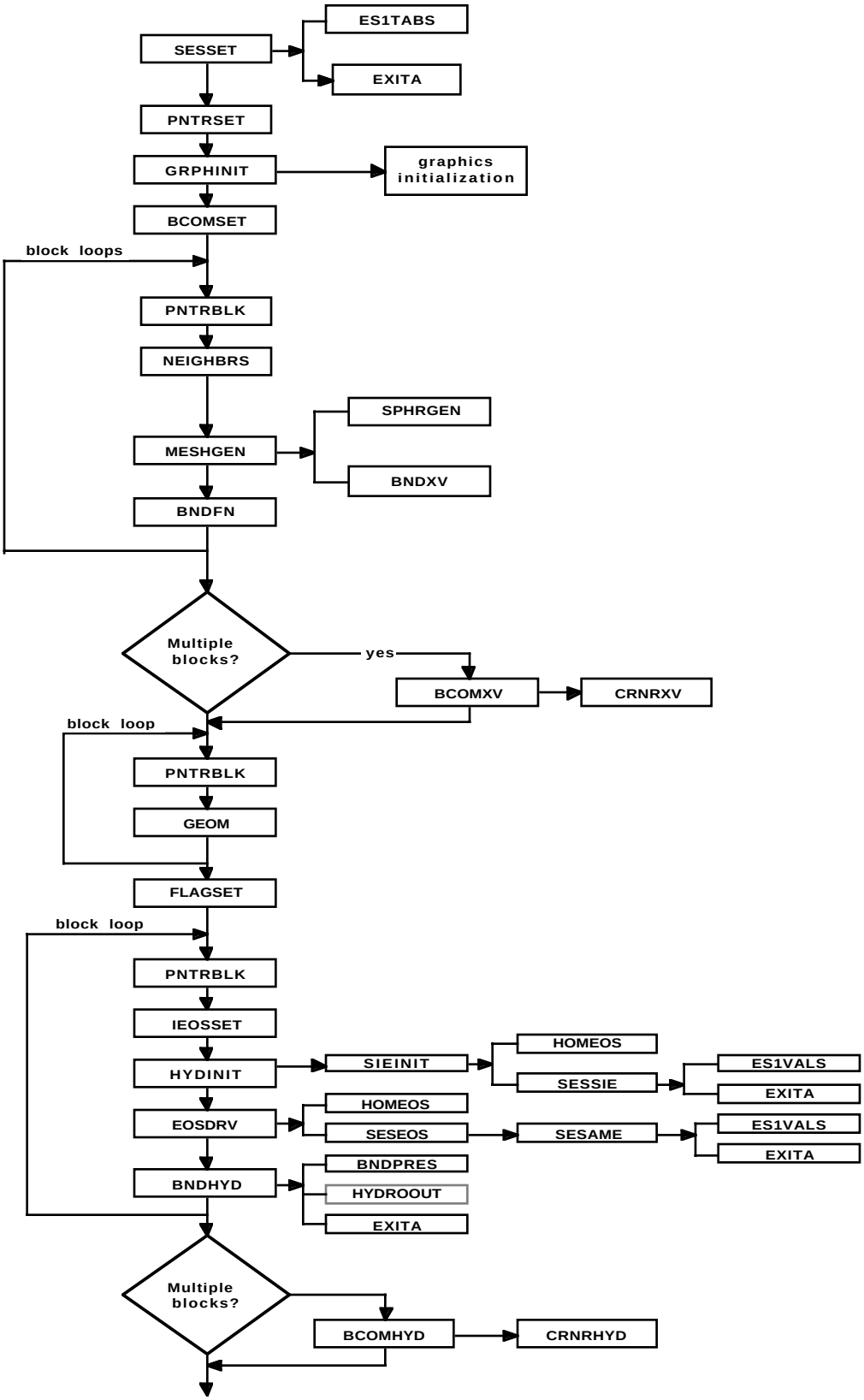


Figure 4.2.4-2. Calling sequence for subroutine NEWPROB.

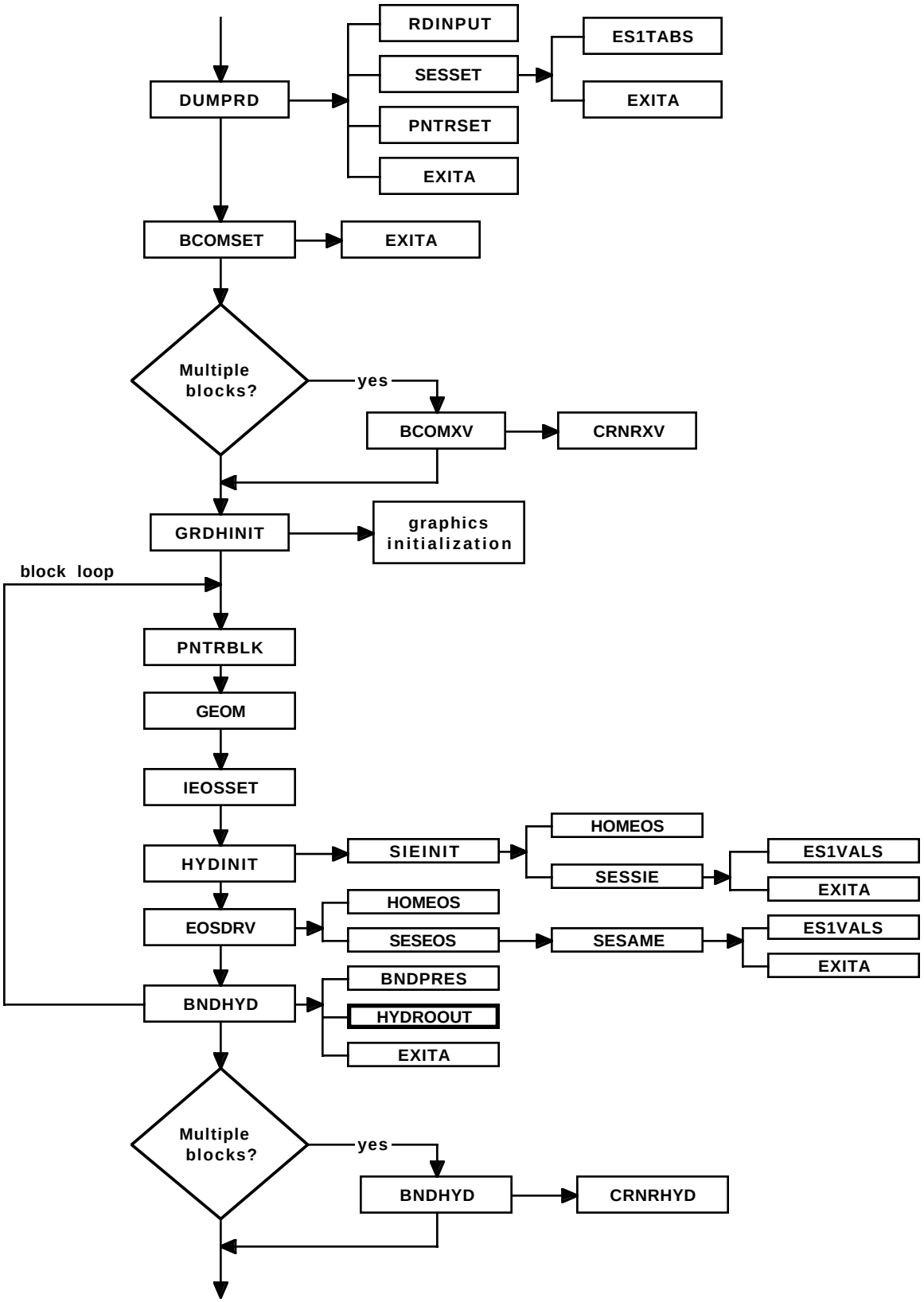


Figure 4.2.4-3. Calling sequence for subroutine RESTART.

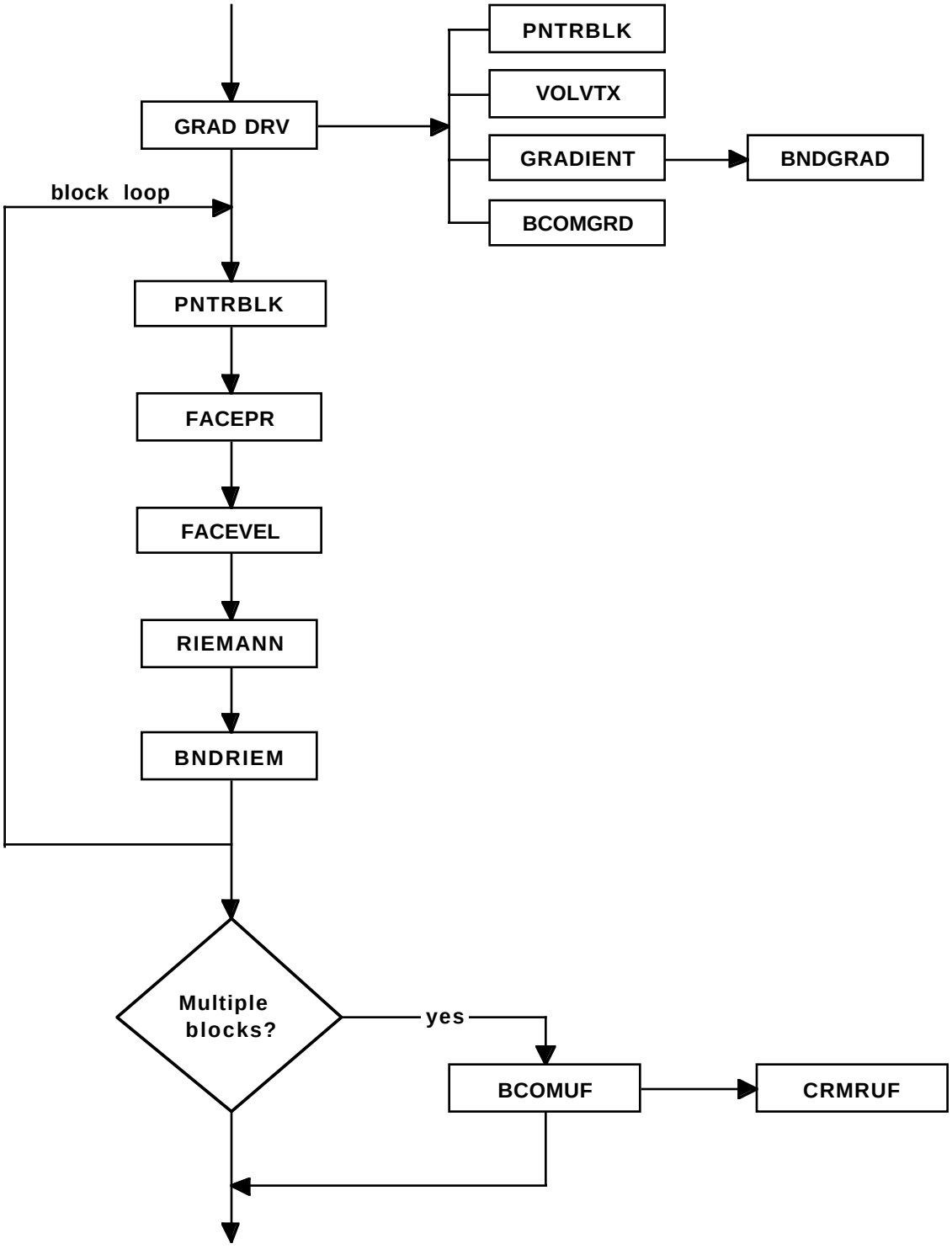


Figure 4.2.4-4. Calling sequence for subroutine RIEMDRV.

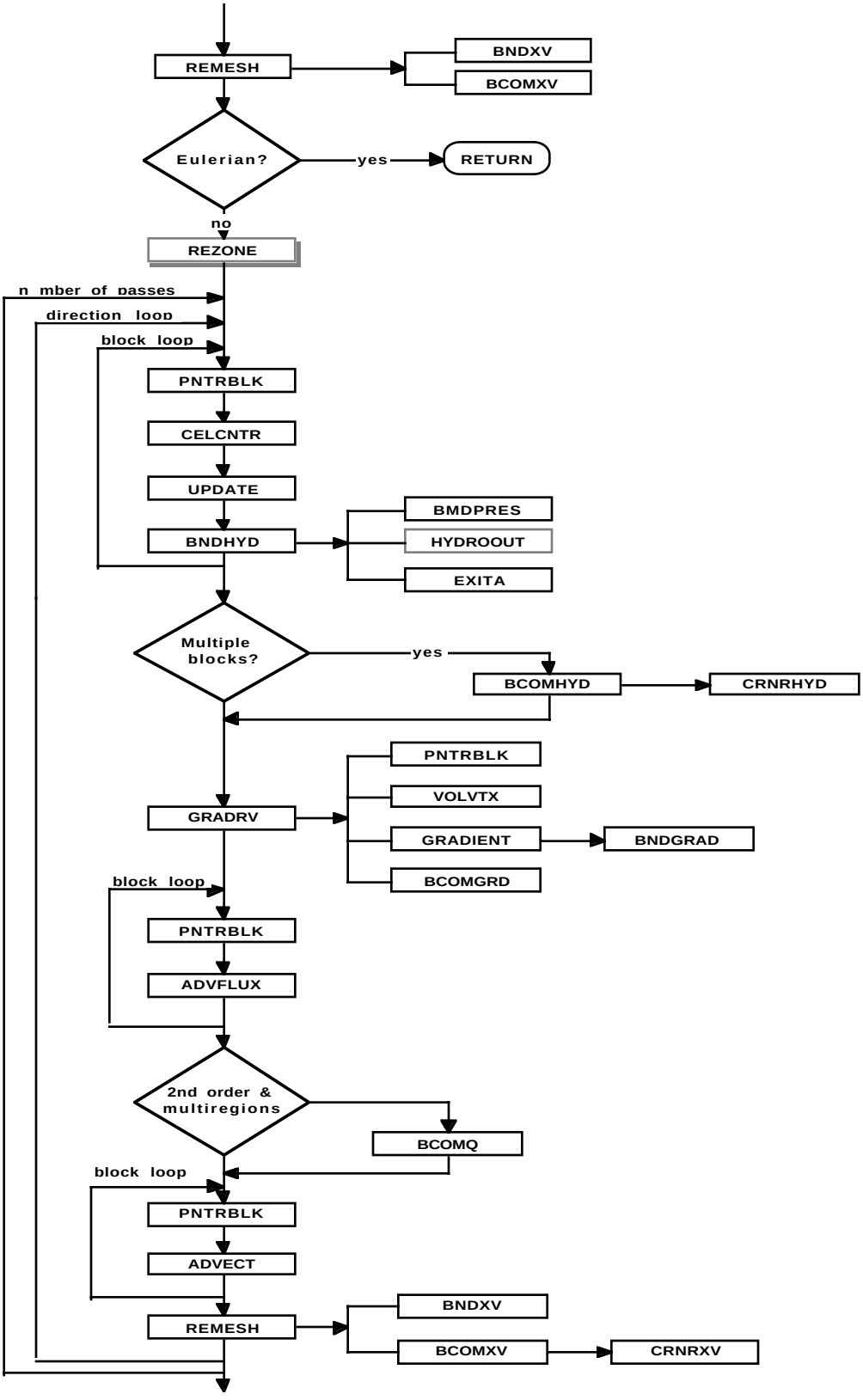


Figure 4.2.4-5. Calling sequence for subroutine REMAP.

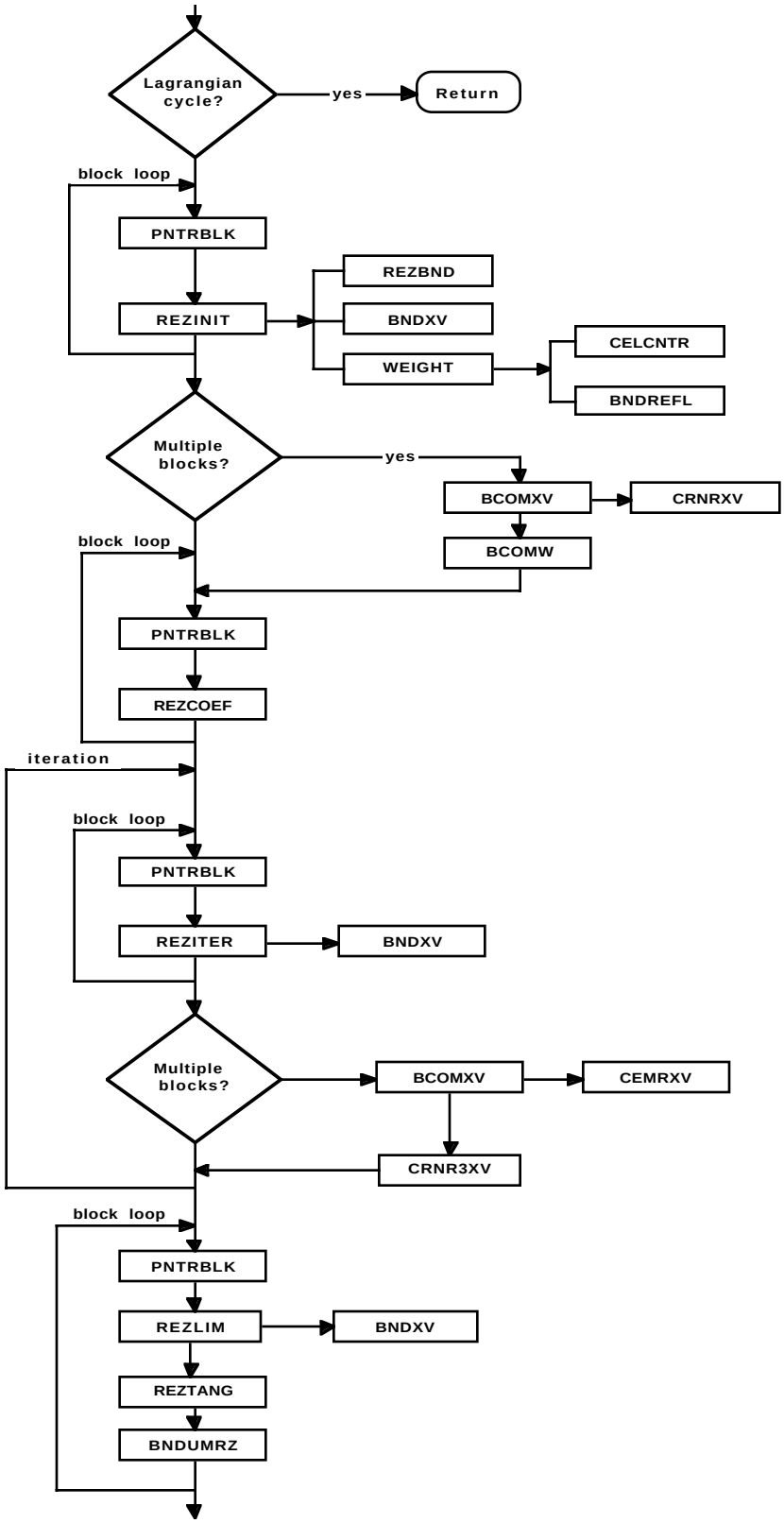


Figure 4.2.4-6. Calling sequence for subroutine REZONE.

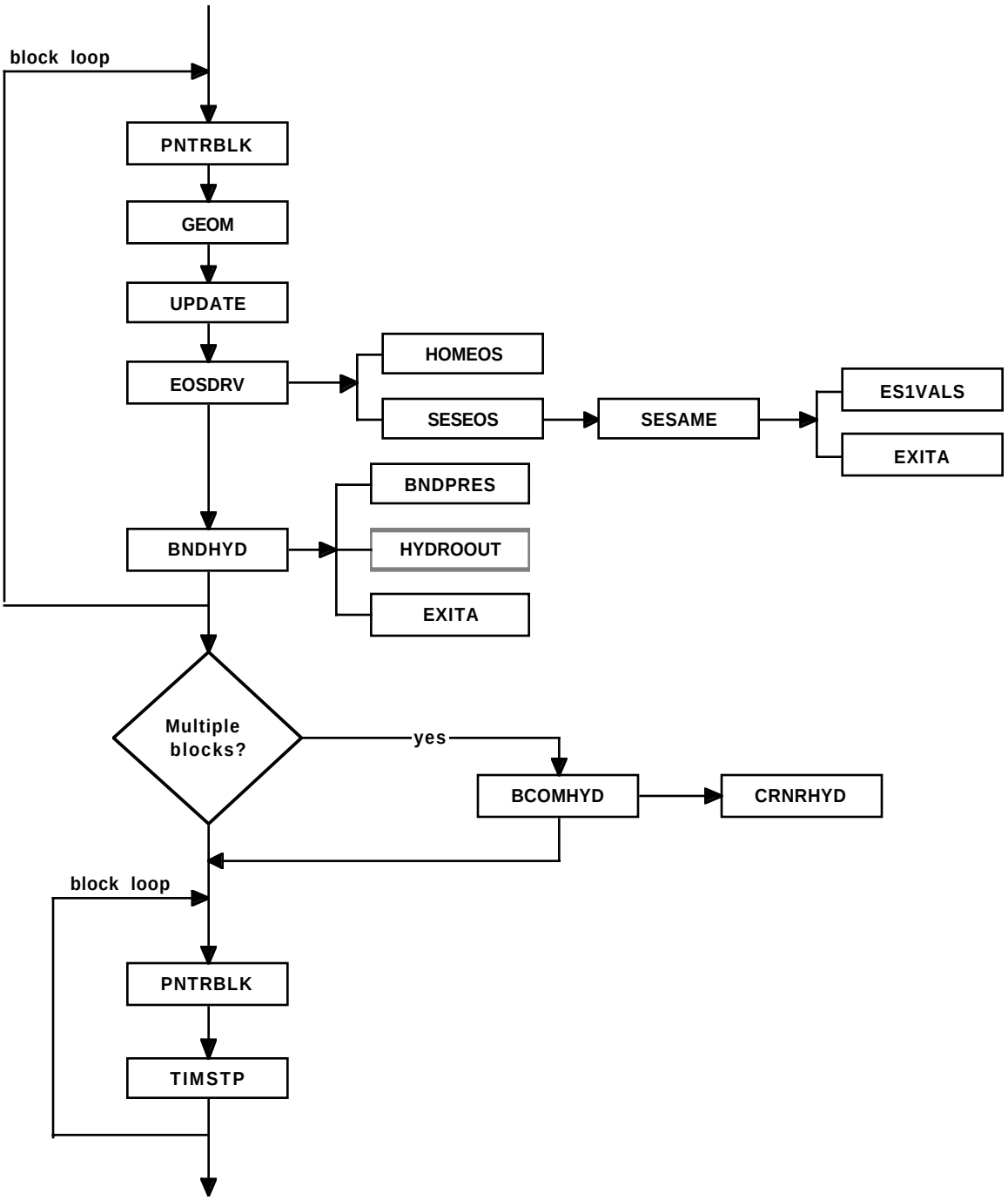


Figure 4.2.4-7. Calling sequence for subroutine NEWCYC.

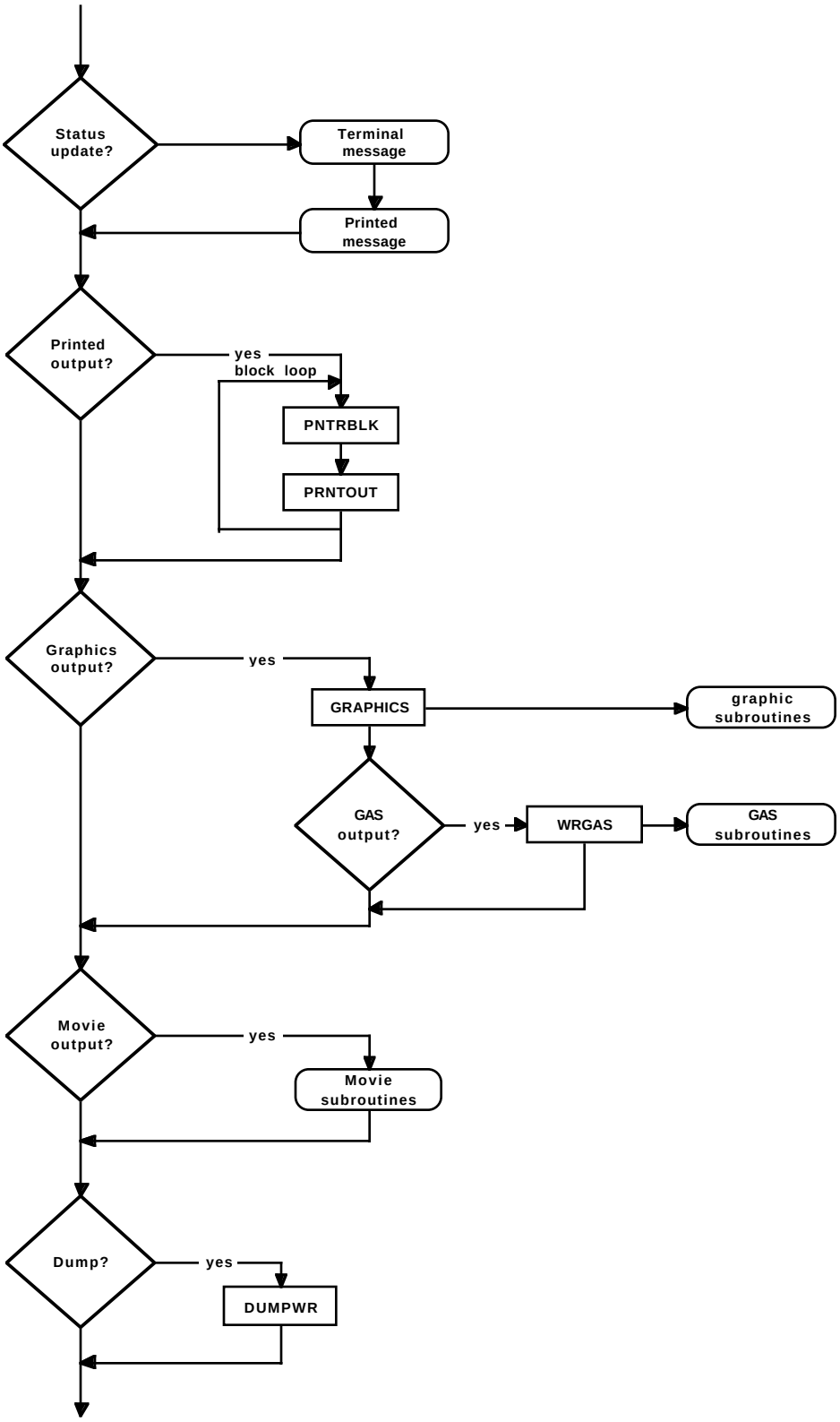


Figure 4.2.4-8. Calling sequence for subroutine HYDROOUT.

§4.3 HYDRODYNAMICS

§4.3.1 Lagrangian Phase

a. Overview of Approach

The Lagrangian phase of the CAVEAT computational cycle is simply the acceleration from a pressure differential and heating from a compression (or expansion) of a fixed mass of material in each cell of the problem domain (the solution to Eqs. 2.1-9 through 2.1-11). This function is controlled from the driver subroutine HYDROCYC, which delegates the work among the routines RIEMDRV, LAGHYDRO, and LAGVEL. The subroutine RIEMDRV controls the calculation of the cell face pressure, p*, and the face normal velocity, u*, resulting form the solution of the approximate Riemann problem (§3.2.1a). LAGHYDRO uses p* and u* to accelerate and perform pdV work on the constant mass in each cell of the mesh. Finally LAGVEL uses the density across the cell faces and u* of the cell faces that join at the vertex to estimate the Lagrangian vertex velocity (§3.2.1b). At the end of the Lagrangian phase, all cell quantities are updated before the remap phase begins (actually done in subroutine UPDATE called from REMAP).

b. Data, Variables, and Input

Upon entry to HYDROCYC the n-time state is defined by the mass, velocity, total energy per unit mass, and the cell vertex position, respectively denoted in the code by CM, UC, TE, and XV. From this state and the material type are derived the auxiliary quantities of volume, density, pressure, sound speed, strong shock parameter, and cell-center position, respectively denoted VOL, RHO, PR, SS, RA, and XC. The geometric quantities FN, FA, and RAV contain the components of each face normal, face area, and the average radial position for each cell.

On exit from RIEMDRV, the variables UF and PF contain the Riemann quantities u* and p*. On exit from LAGHYDRO the variables VOL, UCL, and TEL contain the n+1 time variables of compressed (or expanded) volume, accelerated velocity, and corresponding total energy per unit mass. These are call the *Lagrangian* quantities at n+1 time. (In subroutine REMAP, UC and TE are set equal to UCL and TEL.) On exit from LAGVEL, the array UM contains an estimate of the fluid velocity at cell vertices which are called the Lagrangian vertex velocity.

IORDER [default: 1]; **LIMGRAD** [default: 2]. Two of user-specified input variables that are used in the Lagrangian phase are the order of the spatial differencing (IORDER) and the type of limiting (LIMGRAD) of the gradient used in the second order method. For IORDER equal to 1 or 2 the spatial differencing is first or second order (§3.1.1). For LIMGRAD equal to 0, 1, or 2, the limiting methods are van Leer for all variables except velocity which has no limiting, monotonic, and van Leer (§3.1.2). Note

that the choices of IORDER and LIMGRAD also effect the calculations in the remap phase (§3.2.3 and §4.3.3).

ANTIDIF [default: 0.]. This user specified parameter controls the amount of anti-diffusion introduced in the Lagrangian phase in the second-order calculations. In second-order calculations the states on each side of a cell face, as needed in the Riemann problem, are given by a linear interpolation between the cell-face-center values and the cell-center values. Interpolating a fraction of the distance equal to the Courant number corresponds most closely to the second-order differencing in time and the default (ANTIDIF = 0.). Interpolating closer to the cell-face-center (0. < ANTIDIF < 1.) reduces the numerical diffusion (i.e., introduces anti-diffusion). The choice ANTIDIF = 1. places the interpolation point at the cell-face-center and should be used with caution, because this may be marginally unstable in some circumstances.

NADVSKP [default: 1]. In some simulations in which the time step is controlled by the soundspeed in the problem rather than the advection (§3.2.5), it is possible to take several Lagrangian steps for each remap step without sacrificing accuracy. This can greatly improve the computational efficiency without sacrificing the advantages of remapping. The input parameter NADVSKP is the number of Lagrangian steps taken for every remap step in an ALE hydrodynamics calculation. This parameter must be greater than four to be operative.

c. Boundary Routines

Boundary conditions are applied in the subroutine BNDRIEM, called from RIEMDRV in the computation of p* and u*. Before the call to RIEMANN, the ghost cells of each block of the problem are loaded in such a way that the appropriate u* and p* are calculated in routine RIEMANN (also called from RIEMDRV) for boundary type 3 (outflow) and 5 (interblock boundary). After the call to RIEMANN, BNDRIEM calculates u* and p* for the boundary types 1 (symmetry), 2 (specified pressure), 4 (inflow), 8 (curved-free-slip wall), and 9 (specified velocity) according to the prescription given in §3.2.4. See §4.4.4 for the user-specified input variables for the various boundary conditions.

§4.3.2 Rezone Phase

Nine subroutines comprise the portion of the CAVEAT code concerned specifically with the rezoning phase of the hydrodynamics cycle, that is, the specification of the new mesh. These subroutines and a summary of their function are as follows:

REZONE	- serves as the driver
REZINIT	- performs certain initialization tasks
WEIGHT	- computes the weight function
REZCOEF	- computes the coefficients of the generator equation
REZITER	- executes one iteration of the Jacobi solver on the generator equation
REZLIM	- limits the local amplitude of the rezone mesh motion

REZTANG	- provides an alternate rezone treatment for problem boundaries
REZBND	- loads initial normal velocities on fluxing boundaries
BNDUMRZ	- imposes velocity boundary conditions on rezone velocities

a. Driver Subroutine

The routine REZONE serves as the driver subroutine to direct the various aspects of the rezone task. If the calculation is strictly Eulerian, REZONE reverses the sign of the Lagrangian vertex velocities stored in array UM, to restore the mesh to its original state and returns. If the calculation is not strictly Eulerian, it calls initialization routines REZINIT and REZCOEF for each of the mesh blocks and performs interblock communication if appropriate on the weight function array WT and on the rezone vertex coordinate array XN. It then performs ITREZN iterations of the Jacobi solver by calling routine REZITER. If more than one mesh block is present, interblock communication is performed on the array XN after each iteration. Next, routines REZLIM, REZTANG, and BNDUMRZ are called for each mesh block to limit the local mesh displacement, apply an alternative rezone treatment on boundaries if desired, and to impose boundary conditions on the rezone velocities. The last step is to load the mesh velocity array UM with the final rezone velocities.

b. Initialization

The initialization routine REZINIT accomplishes several tasks. The first is to load the array IVX with the indices of all interface vertices. This array is used for purposes of indirect addressing when processing interfaces and boundaries. The second task is to compute the initial guess for the coordinates of the rezoned mesh vertices, array XN. This initial guess consists of the current vertex positions, XV, with two components subtracted. The first component is the fraction (1. - ALECOEF) of the Lagrangian mesh displacement. Since at this stage in the calculation the array XV includes the full displacement from the Lagrangian phase, subtracting this component leaves array XN with only ALECOEF times the Lagrangian displacement. On Lagrangian interfaces this component must be constrained to lie along the interface. For strictly Lagrangian vertices this component is zero. The second component, computed by routine REZBND, represents normal displacements of fluxing boundaries. Subtracting it implies that there is zero net normal vertex motion on such boundaries. The final task performed by this routine is to call routine WEIGHT to generate the rezone weight function.

c. Weight Function

Subroutine WEIGHT generates the weight function array WT based on the values of several input parameters. When WTAMPL is equal to 1, the routine creates a weight function everywhere equal to unity and returns. For many applications such a weight function is entirely adequate.

One situation, however, for which a constant weight function is inappropriate is a polar mesh. If a constant weight function is used for a mesh with a center of convergence, the Winslow generator will cause the mesh to collapse onto the center point. For polar meshes it is assumed that mesh direction two is the radial direction, and the weight function is initially given the value of the cell index I2 that represents the distance in cells from the origin. When properly scaled, this weighting has been found to give suitable performance for polar meshes. Values of 5.0-10.0 for parameter WTAMPL (see below) typically provides a proper scaling in such problems.

The routine includes options for creating weight functions that cause the mesh to adapt to regions of large pressure gradient and/or specific material type. When parameter WFGR is given a nonzero value, the routine computes a cell-centered pressure or density gradient and adds the magnitude of WFGR times the maximum of WFLM or sum of the squares of its components to the weight function. Similarly, when the array WFMT has a positive value for any material index IMAT, the weight function for all cells with this material type is increased by the amount WFMT(IMAT). Such adaptive capability can prove of great value in a variety of applications. An experienced user of CAVEAT may find it worthwhile to modify this routine to provide still additional options for fashioning the weight function for specific applications.

Routine WEIGHT also has options for smoothing and scaling the weight function. By setting the parameter NSMOOTH to a value greater than zero, the user causes the weight function to be smoothed by NSMOOTH iterations over the local neighborhood. No smoothing is performed, however, across material interfaces or other Lagrangian surfaces. In addition, the routine, as a final operation, scales the weight function such that its mean value is unity and the ratio of maximum value to minimum value is equal to the scaling parameter WTAMPL.

Some difficulties can occur when starting with a mesh created by another code or when using the adaptive feature of CAVEAT. When a mesh is setup from a computer code other than CAVEAT, the initial mesh may not be smoothed in the manner that results from the rezoner in CAVEAT. Under these conditions the mesh may move very quickly during the early part of the simulation. Similarly the adaptive options may be enabled that would result in the a rapid movement in the mesh during the first few time steps in order to better resolve some feature of interest, such as a density gradient. Consequently, an option to relax the initial mesh before the start of the calculation is provided. INITRZN is the number of rezone iterations to be applied to the mesh during problem initialization. Also see §4.41.

d. Generator Equation Coefficients

The coefficients for the generator Eqs. 3.2.2-3 are calculated in routine REZCOEF. In terms of the notation of §3.2.2, code variables X1, X2, Y1, and Y2 correspond to the quantities x_ξ, x_η, y_ξ, and y_η, respectively. Variables W1 and W2 are derivatives of the weight function with respect to mesh indices.

These weight function derivatives are set to zero across Lagrangian surfaces. The variable DINV represents the approximate inverse used in the Jacobi solver. DINV is set to zero for vertices that are not to be rezoned. Arrays A and B contain the sets of coefficients needed to iterate Eq. 3.2.2-4. Array A contains the quantities α, $\beta/2$, and γ, all multiplied by the approximate inverse, and array B contains the two components of the term involving the gradient of the weight function, also multiplied by the approximate inverse. These coefficients are computed once per hydrodynamics cycle and used for all iterations of the Jacobi solver during that cycle.

e. Jacobi Iteration

Subroutine REZITER calculates one iteration of the Jacobi solver Eq. (3.2.2-4) to obtain an improved solution of the Winslow generator equation for the mesh vertex positions in array XN. The procedure is to compute the vertex displacement DX for all vertices using Eq. (3.2.2-4) and then to subtract away the normal component of the displacement for vertices lying on Lagrangian interfaces.

Two methods for determining the direction of the interface through a given vertex are used. The first assumes the interface is parallel to the line segment connecting the neighboring vertices on either side. The second assumes the interface itself consists of line segments connecting successive vertices. The first method has the advantage that it is area-preserving; the second has the advantage that it is robust. The strategy of taking a linear combination of both methods is followed to realize the advantages of both. Therefore interface directions from both methods are calculated and dot products with both these unit vectors and the vertex displacement DX are computed to give two displacement magnitudes. A linear combination of displacements along the two interface directions is used to overwrite the original displacement DX. The input parameter for specifying the fraction of the second method is TANREZN. Its default value is 0.02.

The final step in this routine is to update the rezone vertex coordinate array XN with the displacements stored in array DX and to call routine BNDXV to calculate ghost vertex locations for the array XN.

f. Limiting of Vertex Movement

To assure that the rezone displacements do not result in a tangled mesh, a test is performed in routine REZLIM to limit the magnitude of the vertex movement to be no greater than 0.8 times the distance to the nearest neighboring vertex based on the old vertex positions in array XV. Because the ghost vertices in XV lie almost on the boundaries, in order to allow vertices on the boundaries to move, it is necessary to generate new ghost vertex locations that are a cell width beyond the boundaries. To accomplish this the array XV is copied into scratch array XW, which in turn is processed by routine BNDXV. Array XW is then used in place of XV in the limiting procedure. Routine REZLIM converts the

vertex position information stored in arrays XN and XV into rezone mesh velocities, applies the limiting, sets these velocities to zero for any vertex whose flag indicates it is not to be rezoned, and stores the results in array UMR.

g. Special Boundary Rezoner

Some applications produce behavior along problem boundaries that is ill-suited to the rezoning properties of the Winslow algorithm. One common example is jetting along a boundary. In such situations the Winslow method tends not to move mesh points on the boundary fast enough to track the jet. To overcome these limitations a special boundary rezoning treatment is included in subroutine REZTANG. The method used in this special procedure first involves calculating the total arc length of the given boundary, or if material interfaces intersect the boundary, the arc length of segments of the boundary corresponding to a given material.

If the boundary is straight, the arc length is divided into equal intervals corresponding to the number of cells along the boundary segment, and the resulting vertex displacements are used to obtain rezone velocities for the affected vertices.

If the boundary is curved, the option exists to space the vertices along the arc length according to the local curvature of the segment. The parameter CRVWT controls the spacing. When CRVWT is zero the spacing is uniform. Values for CRVWT of 1.0-2.0 tend to yield acceptable results when finer spacing in regions of greater curvature is desired.

Also for the case of a curved boundary, the issue arises as to the method for computing the local tangent to the boundary. The same strategy is adopted here as in routine REZITER which involves taking a linear combination of two methods. The first method computes the tangent direction near a given vertex as parallel to the line segment connecting the right and left nearest neighboring vertices. The second method treats the boundary as line segments connecting successive vertices and selects the tangent direction depending on which line segment the point of interest lies. The first method is area-preserving while the second method provides robustness. The input parameter BNDCOEF specifies the fraction of the second method to be used.

The user specifies the boundary IB and block IBLK to be rezoned with this special treatment by setting the array IBCREZN(IB,IBLK) equal to one. The default values for this array are zero, which implies the Winslow method is to be applied on all rezonable boundaries unless the user specifies otherwise.

h. Summary of Rezone Input Parameters

BNDCOEF [default: 0.5]. The input parameter BNDCOEF specifies the fraction of the second method used in determining the tangent direction when sliding vertices along a curved boundary in the special boundary treatment (see section g. above). Increasing the value of BNDCOEF will improve the

robustness of the method and tend to minimize ripples on the boundary but in general will decrease the ability of the procedure to preserve the Lagrangian position of the boundary. The recommended range of values is 0.3-1.0.

CRVWT [default: 1.0]. The input parameter CRVWT controls the effect of local curvature on the spacing of vertices on all boundaries rezoned with the special treatment described in section g. above. A value of zero gives uniform spacing with no influence of the local curvature. Increasing positive values give stronger effects of curvature on spacing, with closer spacing of vertices in sections of boundary with small radius of curvature. The recommended range of values is 1.0-2.5.

IBCREZN(IB,IBLK) [default: 0]. The input array IBCREZN serves as a flag to specify the special boundary rezone treatment described in section g. above. A value of one for boundary IB and block IBLK implies the special treatment. The default value of zero gives the standard Winslow treatment used for interior portions of the mesh.

INITRZN [default: 0]. The input parameter ITREZN specifies the number of rezone iterations applied to the mesh during problem initialization. If greater than zero, this parameter allows mesh to relax toward the solution of the Winslow generator equations prior to starting the hydrodynamic calculations.

ITREZN [default: 3]. The input parameter ITREZN specifies the number of iterations for Jacobi solver of rezoner generator equation.

NSMOOTH [default: 0]. The input parameter NSMOOTH specifies the number of iterations of smoothing of the weight function to be performed.

TANREZN [default: 0.02]. The input parameter TANREZN specifies the fraction of the second method (see section e. above) used in determining the interface tangent direction when removing the normal component of rezone displacement on material interfaces and block boundaries. Increasing the value of TANREZN will improve the robustness of the method and tend to reduce ripples on interfaces and boundaries but in general will decrease the ability of the rezone procedure to preserve the Lagrangian position of the interface or boundary. The recommended range of values is 0.02-0.10.

WFGR [default: 0.0]. The sign of WFGR selects between adapting the mesh to high gradients of the pressure or density field for positive and negative values. The magnitude of WFGR specifies the degree of the effect of the gradient on the weight function. If the user is constructing a weight function with contributions from any other sources, such as volume, he will need an idea of the expected amplitude of the the gradients to choose an appropriate value for this parameter. Also see WFLM.

WFLM [default: 1.e100]. This input parameter WFLM is the limiting value of the square of the gradient of density or pressure, above which the gradient contribution saturates. Small values allow regions with smaller gradients to retain their effect on the weight function. Also see WFGR.

WFMT(IMAT) [default: 0.0]. The input array WFMT provides a means to give different weights to different materials in a problem such that the fineness of the mesh depends on material type. This

input serves to add the value WFMT to the weight function for all cells with material number IMAT. Because the weight function is not diffused through material interfaces, this option is primarily used in conjunction with the mixed cell extension of CAVEAT [Johnson et al., 1990].

WFVO [default: 1.0]. The input parameter WFVO provides a means for setting the background value for the weight function to specify the relative importance of cell volume weighting when constructing a weight function with contributions from other sources.

WTAMPL [default: 1.0]. The input parameter WTAMPL controls the amplitude range of the weight function by specifying the ratio of its maximum value to its minimum value. A value of one results in a weight function that has a constant amplitude of unity which is adequate for a wide range of applications. For this value, the input variables controlling the weight function, WFVO, WFGR, WFMT, are ignored. A value of 5.0-10.0 is usually required for problems with polar geometry.

§4.3.3 Remap Phase

a. Computation of Fluxing Volume with a Split Operator

Advection completes the hydrodynamics portion of the CAVEAT time step. In subroutine REMAP, after rezone velocities are computed and stored in array UM, gradients of the state variables RHO, RHO*UC, and RHO*TE are computed (§3.1.2) if the calculation is second order. Then the fluxing volume is computed in ADVFLUX for the directionally split operator used in CAVEAT (see §3.2.3 for details).

Suppose that, for example, we are using the vertex-based fluxing algorithm (the first method described in §3.2.3) and that the x-direction is the first direction of movement for this time step. Then, the fluxing volume FLUX on the left side of cell (i,j) for this pass is (also see Fig. 4.3.3-1)

$$\begin{aligned} \text{FLUX} = {} & r_1(x^{n+1}_{i-1/2,j+1/2} - x^{L}_{i-1/2,j+1/2}) \cdot (y^{L}_{i-1/2,j-1/2} - y^{L}_{i-1/2,j+1/2}) + \\ & r_2(x^{L}_{i-1/2,j-1/2} - x^{n+1}_{i-1/2,j-1/2}) \cdot (y^{L}_{i-1/2,j+1/2} - y^{L}_{i-1/2,j-1/2}) , \end{aligned} \tag{4.3.3-1}$$

where the n+1 superscript indicates a position after the rezone phase and, if the x-direction is also the radial direction,

$$r_1 = (1/6)\,(x^{L}_{i-1/2,j-1/2} + x^{L}_{i-1/2,j+1/2} + x^{n+1}_{i-1/2,j+1/2})$$

$$r_2 = (1/6)\,(x^{L}_{i-1/2,j-1/2} + x^{n+1}_{i-1/2,j-1/2} + x^{n+1}_{i-1/2,j+1/2}) .$$

If the problem is in plane coordinates, r_1 and r_2 are both equal to 0.5. The fluxing volumes for the other cases (such as y-direction first) are easily obtained by variations of this example.

For the Riemann velocity-based fluxing volume (the second method described in §3.2.3), we again take the scalar product of the face velocity with each of the coordinate directions as described in Figure 4.3.3-2 (x-direction first, say), to give

$$\text{FLUX} = \text{UF}*\textit{Face Area}*(\textit{Face normal in x-direction})^2 * \Delta t - \textit{rezone volume} \quad (4.3.3\text{-}2)$$

where the *rezone volume* is computed from the vertex locations at times n and n+1 (again for x-direction on the first pass):

$$\textit{rezone volume} = r_1 (x^n_{i-1/2,j+1/2} - x^{n+1}_{i-1/2,j+1/2}) \cdot (y^{n+1}_{i-1/2,j-1/2} - y^{n+1}_{i-1/2,j+1/2}) +$$

$$r_2 (x^{n+1}_{i-1/2,j-1/2} - x^n_{i-1/2,j-1/2}) \cdot (y^{n+1}_{i-1/2,j+1/2} - y^{n+1}_{i-1/2,j-1/2})$$

where r_1 and r_2 computed in a manner similar to above if the problem is in cylindrical coordinates:

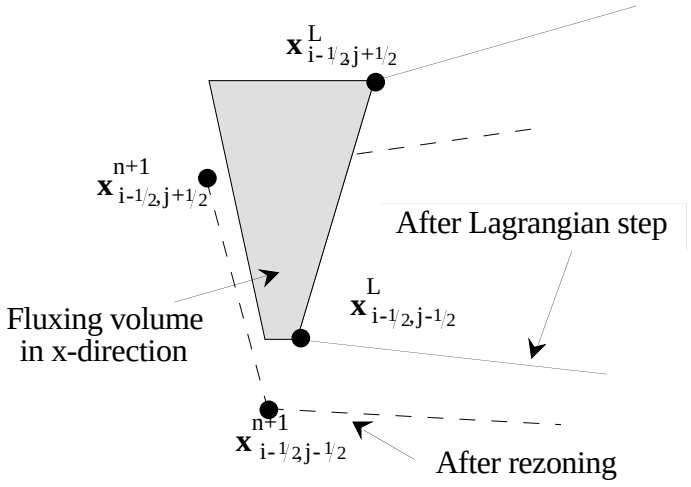


Figure 4.3.3-1. Determination of the fluxing volume based on vertex motion.

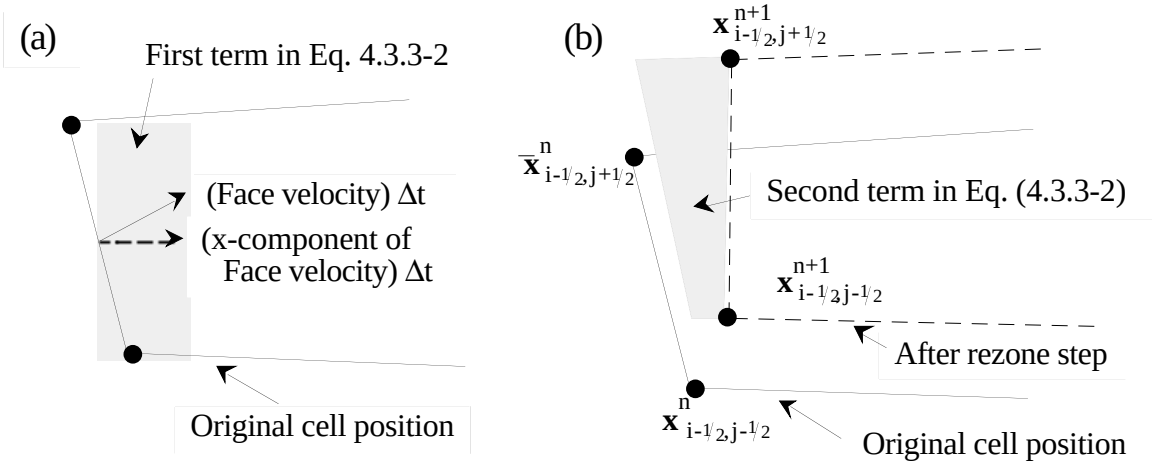


Figure 4.3.3-2. Determination of the fluxing volume based on face motion. The figures in (a) and (b) represent the first and second terms in Eq. (4.3.3-2).

$$r_1 = (1/6)\,(x^{n+1}_{i-1/2,j-1/2} + x^{n+1}_{i-1/2,j+1/2} + x^{n}_{i-1/2,j+1/2}) \ ,$$

$$r_2 = (1/6)\,(x^{n}_{i-1/2,j-1/2} + x^{n+1}_{i-1/2,j-1/2} + x^{n}_{i-1/2,j+1/2}) \ .$$

If the problem is in plane coordinates, r_1 and r_2 are both equal to 0.5.

b. Main Advection Subroutine

After the fluxing volume is known, ADVECT is called to remap the state vector of mass, momentum, and total energy. If first-order advection is desired, **IORDER**=1, (§3.2.3), then velocity and energy are converted first to momentum and total energy per unit volume. Determination of what average value to assign to the fluxed quantity in the advection volume is controlled by **FDONOR**. If FDONOR equals 1.0 (the default), then the momentum and energy contained in the fluxing volume is set to that of the cell-center upstream of the face. Otherwise, if FLUX > 0, the value of the total energy that is taken to be in the fluxing volume is calculated as follows:

upstream weighted volume = .5*FLUX * (1. + FDONOR)

downstream weighted volume = FLUX - *upstream weighted volume*

total energy in FLUX = *upstream weighted volume* * $(\text{RHO*TE})_{\text{upstream}}$ +

downstream weighted volume * $(\text{RHO*TE})_{\text{downstream}}$,

giving an interpolated donor-cell technique.

For second-order advection, **IORDER**=2, a similar scheme is used except that the upstream and downstream RHO*UC's are obtained by Taylor expansions from the respective cell centers using the gradients computed in subroutine GRADIENT. Then,

$$(\text{RHO*TE})_{\text{upstream}} = (\text{RHO*TE})_{\text{center}} + \text{FAC} * (\text{GE}_x * (X_{\text{side}} - X_{\text{center}}) + (\text{GE}_y * (Y_{\text{side}} - Y_{\text{center}})) \ .$$

Here, subscript 'side' refers to the midpoint of the cell face to which FLUX refers, and FAC is

FAC = 1. - (*upstream weighted volume*) / (*upstream cell volume*)

to give better temporal accuracy and retain overall stability of the technique. **LIMGRAD** has the same meaning as in §4.3.1b.

After one pass through ADVECT, the mesh point locations are updated only in the direction that was taken in ADVFLUX when computing FLUX. Then the procedure is repeated, now for the other

direction, using updated state variables to compute gradients and starting mass, momentum, and total energy values.

c. General Advection Subroutines

CAVEAT also provides advection subroutines to advect a general scalar (ADVSCL) and a general symmetric tensor with three distinct components (ADVTNS). The computational philosophy used in these routines is similar to that in the main advection routine. ADVSCL handles advection of either volume-specific or mass-specific scalars. If the input scalar field is mass-specific (e.g., energy per unit mass), then the parameter ISPC is set to 1 by the calling routine and the scalar must be multiplied by the material density *before* entering ADVSCL. Volume-specific scalars need no density weighting and require that ISPC equal 0. ADVTNS currently treats only tensors that are mass-specific. Also, ADVSCL and ADVTNS require gradients for second order calculations (§3.1.2). All other aspects of these subroutines follow directly from the main advection subroutine. The input parameters IORDER and LIMGRAD also apply to these subroutines (§4.3.1b).

§4.4 INITIALIZATION OF THE CALCULATION

These sections cover the necessary information to set up a problem using CAVEAT. This includes the generation of the mesh, the initialization of the materials, the specification of the equations of state, the boundary conditions, and control parameters for the numerical options. Covered in the following sections are the input variables for accomplishing these tasks. Some of the input variables are covered in detail in other sections; these are described briefly in this section, and references are given to the more complete descriptions in other sections. Most input variables have default values, but a few (NCELL, DX, PR0, RHO0, TWFIN, and DT0) have no default values and must be included in the input file. CAVEAT does error checking on the required input variables, and when an error occurs, a message is sent to the screen and the output file (OUT2D).

§4.4.1 Mesh Generation

CAVEAT has a limited mesh generation capability for simple problem configurations. Additional meshing capability can be achieved by using the inter-block communication (§4.2.2). But meshes which require a nonuniform mesh spacing, as described below, must be generated with another code and read into CAVEAT on execution. This section does not apply for problems that are begun from a restart tape (§4.5c).

The different types of mesh that can be generated by CAVEAT are summarized in §4.1.2 and in Fig. 4.1.2-1. Before proceeding with this section, the concept of parts and blocks of mesh should be reviewed in §4.1.2, because the input variables that specify the initial mesh are based on these concepts.

Briefly, the computational region is broken up into blocks of mesh, which may or may not be adjoining. Each block can contain one or more parts, which must be adjoining and have the same number of cells along the common edges. Parts are used only for the setup of the mesh and for initialization of the materials (see next section); once the initial problem is specified, the concept of parts is no longer used.

Unlike some codes, CAVEAT is not vectorized along one mesh direction but over the entire mesh. For example, a one-dimensional problem (one cell wide and many cells long) will be just as vectorized if the problem extends along the 1 or 2 direction. Therefore, it is not necessary to consider the orientation of the mesh, if it were arbitrary, in problems with large differences in the number of cells in the two mesh directions.

NPRTS(M,IBLK)[default: 1,1; one part in each direction]**, NCELL**(KPRT,M,IBLK) [no default]**, DX**(KPRT,M,IBLK) [no default]. The input for the mesh generation requires the number of parts in a block NPRTS(M,IBLK) for each mesh direction M and for each block IBLK, the number of cells in each part NCELL(KPRT,M,IBLK), and the spacing of each cell DX(KPRT,M,IBLK), as in the example in Fig. 4.4.1-1; see IGEOM below for the meaning of DX for the different geometries. The input variables NCELL and DX *must* be provided in the input file by the user. Because the mesh generation is part oriented, the mesh is uniform within each part. Note that any units for DX can be chosen as long as the units for the other input quantities are chosen consistently; the exception to this is noted in §4.4.3. Note that NPRTS cannot exceed the code parameter **NP** [default:64] otherwise an error will occur at execution.(§4.5f), and the computer code must be recompiled with a larger value for NP. A similar limitation exists for KPRT, the index for the number of parts along a given mesh direction; KPRT cannot exceed the parameter **NS** [default:8] without resulting in an error at execution (§4.5f).

NBLKS [default: 1]. NBLKS defines the number of mesh blocks in the problem. The use of more than one mesh block to describe a problem can greatly extend the meshing capability (§4.1.4). Typically multiple blocks are connected by the interblock boundary condition described in §4.1.4, but blocks can be totally noninteracting, except that all blocks have a common time step. Note that NBLKS cannot exceed the code parameter **NB** [default:4] otherwise an error will occur at execution.(§4.5f), and the computer code must be recompiled with a larger value for NB.

IGEOM [default: 1]. IGEOM defines the type of mesh that is constructed for all blocks and the coordinate system in which the equations are solved (see Tables 4.1.1-1 and 4.4.1-1). The specified value of IGEOM applies to all blocks. The choice of IGEOM=5 and 6 generates a multiblock mesh for curvilinear geometries, and input variables take on different meanings with this option (see Table 4.4.1-1, Fig. 4.4.1-2 and Fig. 4.1.2-1).

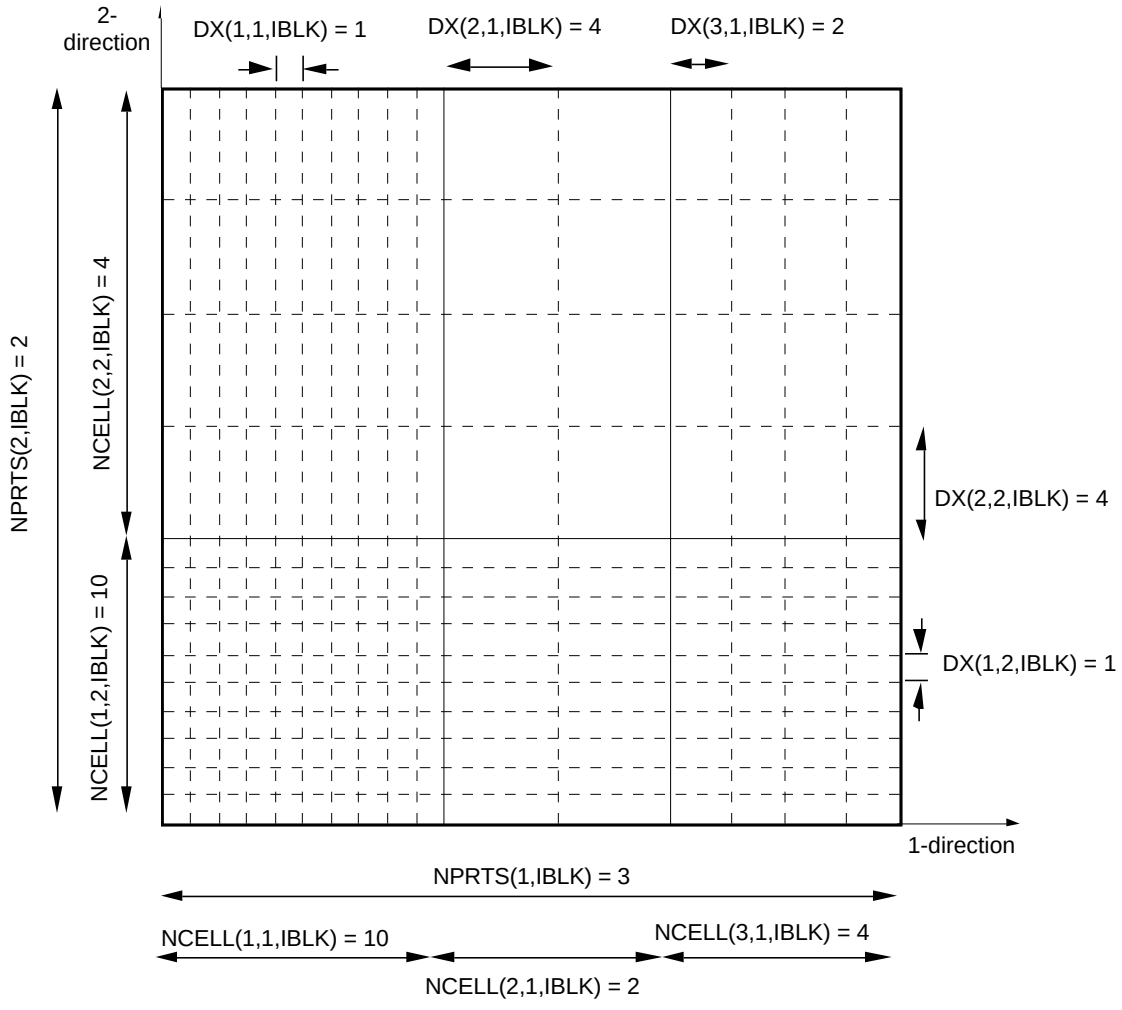


Fig. 4.4.1-1. An example problem illustrating the definitions of the mesh-generation variables. In this example IGEOM is either 1 or 2 and the block number is IBLK. The lower left corner is located at (X0(1,IBLK), X0(2,IBLK)).

X0(M,KPRT) [default: (0.,0.)]. X0 defines the spatial location of the lower left corner of the mesh in Cartesian coordinates (see Figs. 4.4.1-1 and 4.4.2-1 for examples). For IGEOM equal to 2 or 4, because the polar mesh is generated in a clockwise direction, the lower left corner of the mesh is located along the surface with the smaller r value. For IGEOM equal to 5 or 6, X0 takes on the the dimensions shown in Fig. 4.4.1-2. (Also see Fig. 4.4.2-1 for the meaning of X0 in the different geometries.)

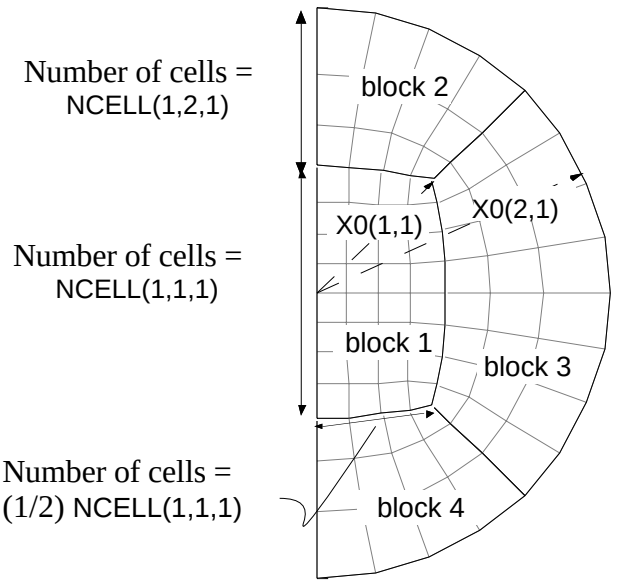


Figure 4.4.1-2. Mesh generation for IGEOM=5 and 6, *polar* mesh using multiblocks. In this example NCELL(1,1,1)=8 and NCELL(1,2,1)=3. Note that the input variables for X0 for this choice of IGEOM refer to the radius of different features of the mesh, relative to the origin.

IRADIAL [default: 1]. IRADIAL controls the orientation of the cylindrical axis in the IGEOM option of 2, 4, or 6. For IRADIAL = 1, the cylindrical axis corresponds to the coordinate direction two and the radial direction corresponds to the coordinate direction one. For IRADIAL = 2, the correspondence is reversed. If IGEOM is equal to one or three, IRADIAL is not meaningful, and if it is specified in the input file, it is ignored and set equal to zero. Note that in Table 3.1.1-1 the three lines correspond to values of IRADIAL of 0, 1, and 2.

Table 4.4.1-1.

Association between IGEOM, the mesh setup, internal coordinates, and DX*.

IGEOM	System	Type of mesh	Internal Coordinates	DX(KPRT,1,IBLK)	DX(KPRT,2,IBLK)
1	rectangular	rectangular	Cartesian	x increment	y increment
2	r-z cylindrical	rectangular	cylindrical	r increment	z increment
3	r-θ cylindrical	polar	Cartesian	θ increment	r increment
4	r-φ spherical	polar	cylindrical	φ increment	r increment
5	r-θ cylindrical	multiblock†	Cartesian	ignored	ignored
6	r-φ spherical	multiblock†	cylindrical	ignored	ignored

* *System* means the type of coordinate system and mesh that is desired; *type of mesh* indicates how the mesh is generated in space; *internal coordinates* indicate how the difference equations are written (§3.1.1). Note that the input parameter, IRADIAL, allows the user to reverse the association of the coordinate direction and the radial direction.
† For the multiblock treatment, see §4.1.4.

INITRZN [default: 0] When a mesh is setup from a computer code other than CAVEAT, the initial mesh may not be smoothed in the manner that results from the rezoner in CAVEAT. Under these conditions the mesh may move very quickly during the early part of the simulation. Similarly the adaptive options (§4.3.2c) may be enabled that would result in the a rapid movement in the mesh during the first few time steps in order to better resolve some feature of interest, such as a density gradient. Consequently, an option to relax the initial mesh before the start of the calculation is provided. INITRZN is the number of rezone iterations to be applied to the mesh during problem initialization.

§4.4.2 Material or Part Initialization

Once the mesh is generated, each cell is loaded with initial conditions that are supplied in the input file or by default values. The initial conditions are the type of material, the density, the internal energy, and the velocity. The assignment of the initial conditions is done by parts. Within each part the initial conditions are uniform. This section does not apply to problems that are begun from a restart tape (see §4.5.2).

All of the input parameters described in this section reference the part number by a single number (see the definition of IP in §4.1.2), in contrast to a pair of numbers as in the last section. Figure 4.4.2-1 illustrates the numbering convention for parts.

MATNUM(IP,IBLK) [default: 1]. The input variable MATNUM(IP,IBLK) defines the *material number* for the part IP in the block IBLK. The material number is a number from one to thirty that designates a specific material type. Each unique material number in MATNUM must have a corresponding property array (see input variable PROPRTY in §4.4.3).

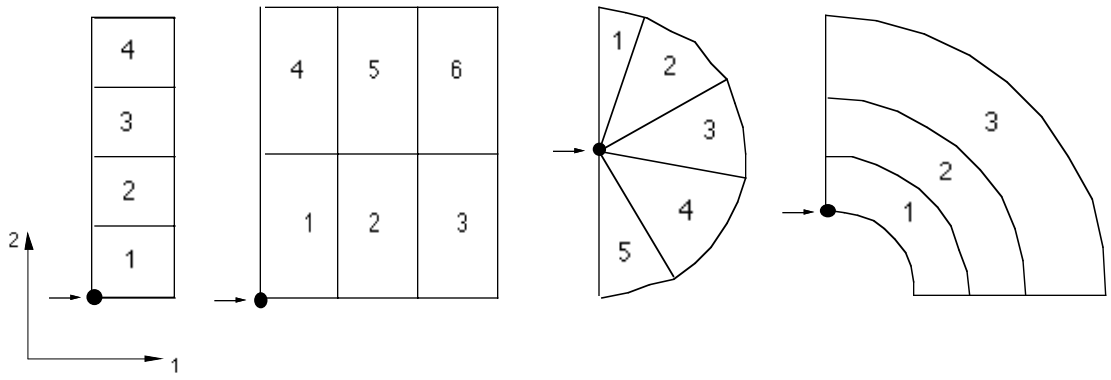


Figure 4.4.2-1. Examples illustrating the numbering of parts. The first two examples on the left are with IGEOM equal to 1 or 2. The two examples on the right are with IGEOM equal to 3 or 4. The '→●' indicates the location of X0 for each mesh.

PR0(IP,IBLK) [no default]. The input variable PR0(IP,IBLK) defines the initial pressure for the part IP in the block IBLK. See the comment below under RHO0 on the initial calculation of internal energy.

RHO0(IP,IBLK) [no default]. The input variable RHO0(IP,IBLK) defines the initial density for the part IP in the block IBLK. RHO0 must be specified by the user for each part in the problem. CAVEAT uses PR0 and RHO0 for each part along with the equation of state for the part (as defined by MATNUM) to calculate the initial internal energy and sound speed of each part. Hence the internal energy of a part cannot be specified directly.

UC0(M,IP,IBLK) [default: (0.,0.)]. The input variable UC0(M,IP,IBLK) defines the initial velocity component along the Mth Cartesian direction for the part IP in the block IBLK. Also see UR0 below.

UR0(IP,IBLK) [default: (0.,0.)]. The input variable UR0(IP,IBLK) defines the initial radial velocity component in spherical coordinates for the part IP in the block IBLK. The direction of UR0 is taken to be directed outward from the origin. It can be specified in addition to UC0 for any choice of IGEOM.

§4.4.3 Material Specification

CAVEAT allows the user to specify up to 30 different materials. Each material is assigned an identification number ranging from 1 to 30. This is the so-called *material number*. The material number may be assigned arbitrarily in the range from 1 to 30 when there are fewer than 30 materials, but it must be unique to a given material.

Material specification is accomplished by means of an array, PROPRTY, which is read in from the input file IN2D using NAMELIST format (see §A.2). If this array is not specified then a single material is assumed by default, a gamma-law gas with $\gamma = 5/3$.

The array PROPRTY(100,30) specifies up to 100 *parameters* for each of the up to 30 materials. The first four parameters have a common meaning for all materials; the meaning of the remaining parameters depends on the *material type*, the first of the four parameters. For a given material number (IMAT), these first four parameters may be described as follows:

PROPRTY(1,IMAT) [default: 1.]. This is a number, ranging between 1 and 8, describing the *material type* or *EOS type*. Currently, there are five material types available. There are four analytic EOS types, described in §2.2, and a tabular EOS type using the Los Alamos SESAME library (§D). The available EOS types are

PROPERTY(1,IMAT)	EOS Type (§2.2)
1.0	Gamma-law (or Ideal gas)
2.0	Linear (or Stiffened gas)
3.0	Quadratic (or Osbourne)
5.0	HOM
8.0	SESAME

Depending on the material type, the units used by CAVEAT may be different. Data for the analytic EOS types may be specified in any consistent set of units. SESAME, however, internally assumes units of Mg/m^3, GPa, and MJ/kg for density, pressure, and energy, respectively. These units may be changed for use in CAVEAT by specifying the corresponding conversion factors **CONVD**, **CONVP**, and **CONVE,** in the input file IN2D. The default values are 1.0, 0.01, 0.01, which convert the SESAME output values to units of g/cm^3, MBar, and MBar-cm^3/g for density, pressure, and energy, respectively.

PROPRTY(2,IMAT) [default: 1.]. This parameter specifies the reference density (ρ_0) for the material. It is relevant only for the linear, quadratic, and SESAME EOS types.

PROPRTY(3,IMAT) [default: 0.]. This parameter specifies the pressure floor. Pressures obtained from the EOS are limited to lie on or above the pressure floor.

PROPRTY(4,IMAT) [default: 4./3.]. This parameter specifies the strong shock parameter (A_S) used in the approximate Riemann solver and described in §3.3.1. This is a nondimensional thermodynamic parameter specifying the asymptotic slope of the shock Hugoniot in U_S-U_p form, where U_S is the shock speed and U_p is the particle speed. For many materials the shock Hugoniot can be well fitted by the linear expression

$$U_p = a_S + A_S U_S,$$

where a_S is the speed of sound and the slope A_S is the desired parameter. [Note that this parameter is the same as the HOM EOS parameter S (i.e., $S \equiv A_S$).]

The parameter A_S may be obtained or estimated as follows:

a) For an ideal gas (gamma-law type), $A_S = {}^1/_2 (\gamma + 1)$,

b) For a material whose pressure depends on density only, $A_S = 1$,

c) For a material whose experimental shock Hugoniot is available (e.g., [Marsh, 1980]), A_S is obtained from the slope of the Hugoniot in the above-mentioned form. Alternatively, Hugoniot fits may be obtained from other sources (e.g., [Holian, 1984]),

d) As a last resort, the shock Hugoniot may be computed using analytical or SESAME equation of state and A_S obtained from the slope of the Hugoniot:

$$A_S = \lim_{U_p \to \infty} \frac{dU_S}{dU_p} \quad .$$

PROPRTY(I,IMAT; I ≥ 5) [default: 5./3. for I = 5; no default for I > 5]. These are additional parameters that are required to specify the equation of state. They are described for the various equation of state types in §2.2 and are listed in Table 4.4.3-1.

Table 4.4.3-1

Equation of state parameters specified in the input array PROPRTY(I,IMAT) for I ≥ 5.

EOS #:	1	2	3	5	8
EOS name:	Gamma-law	Linear	Quadratic	HOM	SESAME
5	γ	c_1	c_1	C	SESAME #
6		c_2	c_2	S	RSCALE
7		c_3	c_3	V_{SW}	ESHIFT
8		c_4	c_4	C_1	A_1
9			c_5	S_1	A_2
10			c_6	F	A_3
11			c_7	G	IREVFLG
12			c_8	H	
13				I	
14				J	
15				Γ	
16				C_V	
17				V_0	
18				α	
19				T_0	
20				p_0	
21				v_{max}	
22				v_{min}	

§4.4.4 SESAME Interpolating Option

ISESOPT [default: 3]. This parameter, whose value may be 1, 2, or 3, specifies the method to be used for interpolating the SESAME data tables. The available interpolating methods [Kerley, 1977; Slattery and Spangenberg, 1982] are:

ISESOPT	Interpolation Method
1	4 point bi-linear function
2	6 point bi-quadratic function
3	12 point bi-rational function.

These options are listed in order of increasing accuracy and reliability, but also in order of increasing complexity and computational cost. Note that the same option is selected for *all* SESAME materials.

§4.4.5 Boundary Conditions

Boundary types are set in CAVEAT by specifying values of the input array IBC (a quick reference is in the code under the subroutine GLOSSARY and in §A) and associated input variables. Different input variables are required for each of the boundary conditions. Each block of the mesh has four sides, numbered one through four, in the order bottom, top, left, and right, in which the block is oriented so that the lower left corner of the block has the lowest storage location and the upper-right corner has the highest storage location; or alternatively, the bottom lies along the side in the one-direction and has the lowest mesh coordinate in the two-direction within the mesh.

Information for the input variables for the boundary conditions is given in Table 4.4.5-1. The convention for the dimensions of the input arrays in Table 4.4.5-1 is: IBC(IB,IBLK), PBC(KPRT,IB,IBLK), RHOBC(KPRT,IB,IBLK), MATBC(KPRT,IB,IBLK), UBC(M,KPRT,IB,IBLK), and URBC(M,KPRT,IB,IBLK); the index KPRT is IPRT when IB is 1 or 2, and KPRT is JPRT when IB is 3 or 4. For the meanings of the variables see the definitions of PR0, RHO0, UC0, UR0, and MATNUM in §4.4.2; these variables have identical meanings as PBC, RHOBC, UBC, URBC, and MATBC, except they are applied to parts, instead of boundaries. All of these input variables for boundary conditions are defaulted to zero, except IBC and MATBC, which are defaulted to 1.

§4.4.6 Control Parameters

This section discusses the input parameters that control aspects of the execution of CAVEAT that are not covered elsewhere.

ALECOEF [default: 0.9]. ALECOEF is the input parameter that controls the application of the remap phase. When ALECOEF is set equal to zero, the mesh is returned to its initial position at the end of the Lagrangian phase; the calculation is Eulerian. When ALECOEF is set equal to one, the mesh is moved with the Lagrangian velocity, and no fluxing occurs between the cells; the calculation is Lagrangian. For values of ALECOEF between zero and one, the calculation is arbitrary Lagrangian-Eulerian (ALE). For values of ALECOEF close to 0.0, the Lagrangian velocity fluid has little effect on the movement of the mesh. For values of ALECOEF close to 1.0, the Lagrangian velocity fluid has the dominant effect on the movement of the mesh. For more about the use of ALECOEF, see §5 and the test problems in §6.

DT0 [no default]. DT0 is the initial time step for the simulation.

DTMIN [default: 1.e-3*DT0]. DTMIN is the timestep below which the calculation will terminate.

DTMAX [default: 1.e4*DT0]. DTMAX is the largest possible time step; values of the time step above DTMAX will be reset to DTMAX .

TWFIN [no default]. TWFIN is the time for the simulation to terminate.

NCYCMAX [default: 10,000]. NCYCMAX is maximum number of time step for a simulation at which the program will gracefully terminate.

CUSHION [default: 0.0]. CUSHION is the time in seconds of the job time limit at which CAVEAT will begin a soft termination. This option enables the program to complete all final output before running out of alloted time.

Table 4.4.5-1

Input variables for boundary conditions.

IBC	Boundary Type	Fixed in Space?	Required Input Variables
1	Reflective or symmetry (default)	yes	none
2	Specified pressure	no	PBC
3	Outflow or zero gradient	yes	none
4	Inflow	yes	PBC, RHOBC, MATBC, UBC, URBC
5	Boundary between blocks	no	see §4.1.4
8	Curved, rigid, free-slip	yes	none
9	Specified velocity	no	UBC or URBC

§4.5 CALCULATION OUTPUT

CAVEAT provides information to six files or devices. They include the user terminal, printed output, a dump file, two graphics files, and a movie file. Currently, the movie file is not implemented. A brief description of these files including the information contained on them is provided in this section. In general, the problem time at which data are provided to the files is specified by the user (§4.4).

a. Terminal Output - Status and Timing Information

Information concerning the termination or the status of a CAVEAT run is provided to the user terminal (TTY, tape9). Termination may be the result of an error encountered during the setup or computation of the problem or following the completion of a successful calculation. A list of the information available to the terminal is provided in this section under the heading of Output Messages (§4.5f).

b. Print File (OUT2D) - Initialization, History, and Timing Information

The print file (OUT2D, tape6) contains a copy of the input variables, a detailed summary of the values of the cell-centered variables at times specified by the input parameter **TWPRNT** as well as information about the status of a CAVEAT run. Messages are supplied when a CAVEAT run terminates as a result of an error encountered during a computation or following the completion of a successful calculation. A list of the information available to the print file is provided in this section under the heading of Output Messages (§4.5f).

c. Dump File (DP2D and RS2D) - Restart Option

The dump file DP2D contains all of the computational information necessary to uniquely define the state of the calculation at a specific CAVEAT cycle. This provides the ability to restart a calculation in a consistent fashion by renaming the dumpfile to RS2D and setting the dump number (parameter **NRSTART** in file IN2D) to an appropriate value corresponding to the desired dump as reported in OUT2D. The computational times at which data are supplied to the dump file are controlled by the input parameter **TWDUMP** (§A.3d). The dump times are printed to the OUT2D file (§4.5b) along with the corresponding dump number.

d. Graphics Plot (PLOT)

Graphical output is provided to the CAVEAT file PLOT. Information on this file is generated by subroutines that are part of the Common Graphics System (CGS) library. This system is supported by

the Computer Graphics Group of the Computing and Communications Division (C-Division, Group C-6) at the Los Alamos National Laboratory. The computational times at which information is provided to the file PLOT is controlled by the user-supplied parameter **TWFILM** (§A.3d).

ILINE(M,1 through 10) [default: -1], **I2DPLOT** [default: 0]. One-dimensional, contour, and velocity vector plots are available. The one-dimensional plots include pressure, density, specific internal energy, and the magnitude of the velocity along a constant i-line or j-line. Control of abscissa (the constant logical line) is provided by the user-supplied parameter ILINE. Plots along 10 different logical lines may be obtained. For example ILINE(2,1) = 10 results in one dimensional plots along the 2 direction for the 10th row of cells in the 1 direction. Contour and velocity vector plots are provided if the user sets the parameter I2DPLOT equal to 1. The contour plots include the pressure, density, and specific internal energy.

e. GAS File - Graphics output file

GAS (Graphical Analysis System) is a versatile graphics utility, which is supported by the Computer Graphics Group of the Computing and Communications Division C-Division, Group C-6 at the Los Alamos National Laboratory. High quality plots and movies may be created using this utility. Graphics are obtained interactively or from a batch mode once an input file has been created. The GAS utility requires the CAVEAT supplied dump file GASDUMP (for large problems, the files are provided sequentially as GASDUMP, GASDUMQ, ...) as input. The GASDUMP is then read and manipulated by the GAS utility to provided the desired graphical results.

IGAS [default: 0]. GASDUMP file is obtained by setting the input variable IGAS equal to 1. The computational times at which output is written to the GASDUMP file is controlled by the user-supplied parameter TWFILM (§A.3d). The identification number of the variables in the GAS output file are 1: material number, 2: coordinate along direction one, 3: coordinate along direction two, 4: mesh velocity along direction one, 5: mesh velocity along direction two, 6: material velocity along direction one, 7: material velocity along direction two, 8: pressure, 9: internal energy per mass, 10: density, 11: total energy, and 12: sound speed.

GASSCL(1 through 12) [default: 1.]. The variables sent to the gas file can be scaled by the input variable array: GASSCL. The input variable GASSCL(1) scales the time; the rest of the values scale the variables listed above; for example a value of GASSCL(8) of 0.1 would multiply the value of the pressure sent to the GAS file by 0.1.

f. Output Messages

Messages are supplied to the user terminal (TTY) and the print file (OUT2D). A list of the available messages is provided herein. The list is divided into two categories. First the error messages

and then information regarding the status of a CAVEAT calculation. The CAVEAT parameters that are supplied with each message are underlined. The output media that the message is issued to is supplied in brackets. A brief translation of the CAVEAT message follows. (The words in the following descriptions that are set in italics represent the numbers supplied during the calculation.)

Error messages are supplied to the output files when an error is encountered during a CAVEAT calculation. CAVEAT will terminate following the transmission of an error message. The list of CAVEAT supplied error messages includes:

- "Job ended -- time step fell below minimum of Δt. Final time = t and total cycles = *NCYC*. Minimum time step = Δt_{min} occurred in cell *IJ* of block *IBLK*." [TTY, OUT2D]. The CAVEAT calculation terminated because $\Delta t < \Delta t_{min}$ (DTMIN). For more information see §5.3.

- "BCOMSET: mesh sizes are different on edge *I* of block IBLK. Neighbor edge is *IBN* of block *IBLKN*." There is an error in the input array NBC(2, *IB*, *IBLK*). Consequently, the number of cells along one side of a block does not equal the number of cells on the corresponding side of the neighboring block. For more information see §4.1.4.

- "Job aborted -- *NBLKS* blocks exceeds the dimensioning limit of *NB*." [TTY, OUT2D]. The CAVEAT calculation terminated because the input parameter NBLKS was larger than the dimension NB in the CAVEAT parameter statement. For more information see §4.1.3a and §4.4.1.

- "Job aborted -- *NPRTS* segments in block *IBLK* exceeds dimensioning limit of *NS*." [TTY, OUT2D]. The CAVEAT calculation terminated because the input parameter NPRTS(2, iblk) was larger than the dimensions NS in the CAVEAT parameter statement. For more information see §4.1.3a and §4.4.1.

- "Job aborted -- *NSEG* parts in block *IBLK* exceeds dimensioning limit of *NP*." [TTY, OUT2D]. The CAVEAT calculation terminated because the value of NPRTS(1, IBLK) * NPRTS(2, IBLK) exceeded the dimension NP in the CAVEAT parameter statement. For more information see §4.1.3a and §4.4.1.

- "Wrong dump number: dump *NRESTART* not found." [TTY, OUT2D]. The dump number specified by the input parameter NRESTART could not be located on the restart file RS2D. For more information see §4.5e.

- “Job aborted -- invalid EOS type.” [TTY, OUT2D]. The equation-of-state identifier PROPRTY (1, IMAT) was not a valid value. Correct values are:PROPRTY (1,IMAT) =

  1 : gamma law gas
  2 : linear equation-of-state
  3 : quadratic equation-of-state
  5 : HOM equation-of-state
  8 : SESAME tables.

  For more information see §4.4.3.

- “error in sesame evaluation, ecold, number” *IERR* OUT2D]. The CAVEAT calculation terminated because an error was encountered while computing the cold-curve energy from the SESAME equation-of-state. The parameter *IERR* is a SESAME error number [Cranfill, 1983].

- “error in sesame evaluation, p(rho,e), number” *IERR* "OUT2D].The CAVEAT calculation terminated because an error was encountered while computing the pressure from the SESAME equation-of-state. The parameter *IERR* is a SESAME error number [Cranfill, 1983].

- “error in sesame setup, table, mat, index, number *I M IERR* “ [TTY, OUT2D]. The CAVEAT calculation terminated because an of error encountered while initializing the SESAME equation-of-state tables. The printed variables *I, M, IERR* are provided by the SESAME routines and refer to a SESAME table, SESAME material number, and SESAME error number [Cranfill, 1983].

- “routine sessie failed to converge: iseq = *ISEQ”* [TTY, OUT2D]. The CAVEAT calculation terminated because the SESAME equation-of-state failed to converge on a specific internal energy. Typically this error occurs when an initial state is outside of the SESAME tables. If this is not the case, try a different interpolation method (§4.4.3).

- “error in sesame evaluation, pr(ρ,e), number *IERR*” [TTY, OUT2D]. The CAVEAT calculation terminated because the SESAME equation-of-state failed to compute a value for the pressure. The parameter *IERR* is a SESAME error number [Cranfill, 1983].

- "error in sesame evaluation, e-cold, number ierr” [TTY, OUT2D]. The CAVEAT calculation terminated because the SESAME equation-of-state failed to compute a cold-curve energy. The parameter *IERR* is a SESAME error number [Cranfill, 1983].

- "error in BCOMSET: Parameter *NBLKS* must be greater than one when array NBC has nonzero values". [TTY, OUT2D]. The CAVEAT calculation terminated in the subroutine BCOMSET. The input variable NBLKS must be greater than one when the input array NBC (2, IB, IBLK) indicates there is more than one computational block. For more information see §4.1.4.

- "error in BCOMSET: Inconsistency exists in NBC array that involves side *IB* of block *IBLK*." [TTY, OUT2D]. The CAVEAT calculation terminated in the subroutine BCOMSET. There was an error in the input variable NBC (2, IB, IBLK) for the specified boundary and block. For more information see §5.3.

- "error in BNDHYD: Negative density in cell *IMIN* of block *IBLK*." [TTY, OUT2D]. The CAVEAT calculation terminated in subroutine BNDHYD because of a negative density in the specified computational cell and block.

- "Input error, sees last lines in file OUT2D" [TTY]. CAVEAT checks to insure that all of the required input quantities have been specified. If an error occurs, this message is supplied to the terminal [TTY]. The message "The input variable *VAR* has not been defined or is zero in IN2D" is supplied to OUT2D where *VAR* is the undefined variable, which includes the density, pressure, number of cells, grid size, time for the completion of the calculation, or the initial time step size (§5.2 and §A).

Information concerned with the status of a CAVEAT calculation or detailed data at user specified time intervals also are provided to the output file. Additionally, a copy of the input variables found in the namelist INPUT are provided to the print file (OUT2D). A list of the CAVEAT status messages includes:

- "dump *NDUMP* at t = *T* cycle = *NCYC*" [TTY, OUT2D]. A CAVEAT dump was written to file DP2D at the indicated problem time and cycle number. The dump number *NDUMP* can be used for the parameter NRESTART to restart the problem.

- "Restarting from td *NRESTART* t = *T* cycle = *NCYC*"' [TTY, OUT2D]. The dump number NRESTART was located on the file RS2D. CAVEAT will restart a calculation at the indicated time and cycle number.

- "CPU SECONDS USED AND MEGAFLOPS" [TTY,OUT2D]. Following the completion of a CAVEAT run, a list of grindtimes (microsecond/cell /cycle) and total number of operations (megaflops) are provided for 14 of the CAVEAT subroutines.

- "HYDRO REQUIRED *TGRIND* MICROSECONDS/CELL/CYCLE" [TTY,OUT2D]. Following the completion of a CAVEAT run, the total grind time (microseconds/cell/ cycle) for the entire run is provided.

- "ncyc = *NCYC* t = *T* dt = *DT* limited in block iblk i = *I* j = *J*" [TTY,OUT2D]. The status of a CAVEAT run is provided at every 100 cycles or following the completion of a CAVEAT run. Provided are the cycle number (NCYC), the problem time (T), time step size (DT), as well as the block number (IBLK) and cell number (I,J) responsible for the minimum time step size.

- "REGION *IBLK*, time = *T*, i2 = *J*, i1, x, y, density, energy, pressure, u, v" [TTY,OUT2D]. A times specified by the input parameter TWPRNT, results from the CAVEAT run are provided to the print file. This information is provided as a table of values for an I-line (I1). Given are the cell-centered values of cell position (x,y), density (ρ), specific internal energy (e), pressure (p), and velocity (u,v). At table is provided for each block (IBLK) and j-line (I2).

- 'The gas file *GASDUMx* was truncated to *IADT* words." [TTY,OUT2D]. This message is issued to the print file following the dump of information to the last GAS file. It indicates the name of this last file (*GASDUMx*) and its size (*IADT*).

SECTION 5

PRIMER FOR FIRST USERS

This section provides a short introduction to the use of CAVEAT for first users. It is assumed that the reader has access to an executable version of the code. For those users at LANL, Appendix B outlines how to create an executable version of the code; for users elsewhere, Appendix B can serve as an outline of the necessary steps to create an executable code. Throughout this section, references will be made to other sections that can be consulted for more detailed information; these references have the form (§5.1) and refer to a section number in this manual.

§5.1 INTRODUCTION TO USING CAVEAT

CAVEAT contains a simple mesh generator (§4.4.1) and graphical output (§4.5.4). Therefore, simple problems can be generated, calculated, and analyzed with a single executable code. (For users at LANL, the GAS (Graphical Analysis System) is also available to analyze the output of CAVEAT (§4.5.5)). All of these capabilities are controlled by the input variables in the input file (IN2D). Furthermore, CAVEAT does not require recompilation for different problems. CAVEAT is written so that the executable memory is increased as needed to treat the requested problem (§4.2.3).

In setting up a problem, CAVEAT does error checking on the input values and if an error is encountered will output an error message to the screen and terminate gracefully (§4.4, §4.5.6). One example is a check of the size of the defaulted parameters (NB, NS, NP) which alerts the user to recompile CAVEAT if the initial setup of a problem is sufficiently complex.

The input file contains the necessary information for CAVEAT to generate the mesh, initialize the materials, determine the choice of the numerical method, and select the type of output information. The input file can be very short, because most of the input parameters for CAVEAT have defaults. Refer to Appendix A.2 for a description of the input variables and the section in which they are described.

§5.2 A SAMPLE PROBLEM AND DEFAULT INPUT VALUES

In Table 5.2-1 is the input file for a simple shock tube problem (see §6.1 for a description of the output). Figure 5.2-1 illustrates the problem. This input file includes the minimum input variables required by CAVEAT. All other input variables have default values. Default values are chosen to be *best*

values for a variety of problems, as in the choices for the rezoner, or conservative values that increase robustness of the code at the expense of accuracy, as in the choice of the first-order numerical option. The following lists the defaults for the main types of input variables:

Mesh: The problem uses a rectangular mesh and a single block, solved in Cartesian space (IGEOM=1).

Boundary conditions: All boundary conditions on the single block are reflective (all IBC are 1) except for the inflow boundary. Inflow conditions for the inflow boundary are set in the input file.

Material type: The material is a single material with a material number one (MATNUM=1) with a gamma law equation of state (PROPRTY(1,1)=1) with a isentropic exponent of 5/3 (PROPRTY(5,1)=5/3). The material has a reference density of 1.0 (PROPRTY(2,1)=1.0), and a strong shock parameter of 4/3 (PROPRTY(4,1) = 4/3).

Initial state: The initial state of the material in all of the parts is zero pressure (PR0(IPRT)=0.0) and velocity (UC0(IPRT,1) = 0.0, UC0(IPRT,2) = 0.0).

Time step controls: The stability factor (DTFAC) is 0.5. The initial time step (DT0) is set by the user. The minimum time step (DTMIN) for the calculation to terminate is set equal to 0.0001*DT0 if it is not included in IN2D. The maximum time step (DTMAX) is 1000*DT0 if it is not included in IN2D.

Numerical method : The calculation is first order (IORDER=1) for the Lagrangian and remap phase.

Rezoning: The calculation uses the arbitrary Lagrangian-Eulerian option (ALECOEF = 0.9). Because this problem has a single material and rigid boundaries, the rezone capability in CAVEAT produces a uniform mesh for small values of ALECOEF; if the mesh is initially uniform, it remains uniform. Consequently, identical results would be obtained for a calculation with the ALECOEF small and for the Eulerian option (ALECOEF = 0.0).

Graphics: The internal graphics option is selected, but no graphics are produced unless TWFILM is defined, as in the input file. Output of the GAS graphics utility is enabled by setting IGAS = 1.

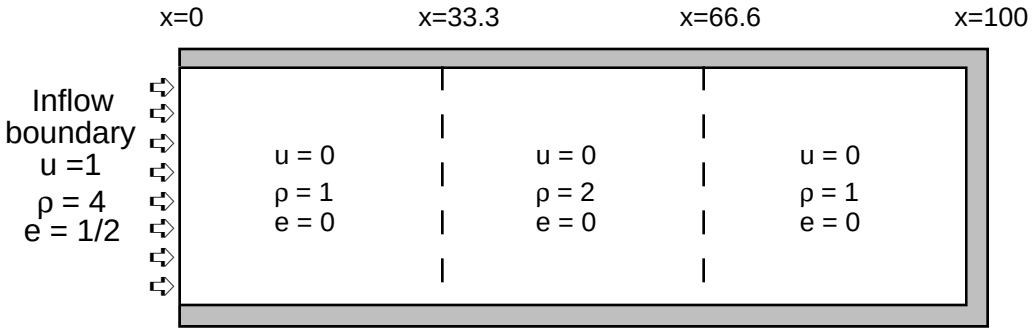
To learn more about the options in CAVEAT, try modifying the input file in Table 5.2-1 as in the first test problem in §6.1, or try some of the additional problems in §6.

Table 5.2-1.

A sample input file (IN2D) that contains the minimum required input variables.

```
Test Problem 1.minimum
$input

 nprts=3,
 ncell(1,1)=33, 33, 33,
 ncell(1,2)=1,
 dx(1,1)=3*1.0101010101,
 dx(1,2)=1.0,

 rho0  (1)  = 1., 2., 1.,
 pr0   (1)  = 3*0.,
 matnum(1)  =  1,  1,  1,

 proprty(1,1)=1., 1., 0., 1.333333, 1.66666666,

 ibc(3) = 4,
 ubc(1,1,3) = 1.,     pbc(1,3) = 1.333333333,
 rhobc(1,3) = 4.0,  matbc(1,3) = 1,


 dt0=0.1,
 twfin=80., twfilm=0.0, 80., 5., igas=1,$

 (Comments can be included here.)
 This test problem is for a one-dimensional shock tube.
 It uses the minimum number of input parameters for the
 problem.
```

* Note that the block index has been omitted in the input parameters that require IBLK, the specification of IBLK is optional because this problem has only one block.



Fluid: Gamma–law gas with $\gamma = 5/3$

Figure 5.2-1. Description of the sample shock tube problem as specified by the input file in Table 5.2-1.

§5.3 GUIDELINES FOR TROUBLESHOOTING

The difficulties with using CAVEAT generally fall into three categories: the setup of a problem of interest, the choice of numerical options in the simulation of the problem, and what to do if problems are encountered in the simulation. The solution to all of these problems is complicated by the large number of options available to the user in CAVEAT. Usually the best approach is to consult an experienced user of the code. The following guidelines might be helpful if an experienced user is not available.

Mesh Generation and Initialization

CAVEAT automatically checks for the most common, fatal errors in specifying the input variables, such as zero values for DX and RHO0. Beyond these, a common difficulty encountered by a first user is gaining a working understanding of the concepts of parts, blocks, and multiple blocks. A review of the sections that cover these concepts (§4.2 and §4.4) or consulting an experienced user is helpful. Briefly, the computational region is broken up into blocks of mesh, which may or may not be adjoining. Each block can contain one or more parts, which must be adjoining and have the same number of cells along the common edges. Parts are used only for the setup of the mesh and for initialization of the materials; once the initial problem is specified, the concept of parts is no longer used.

The many options in setting up a problem can lead to a variety of different meshing strategies for a given problem, with no obvious choice as to which is better. As a general rule, the following considerations usually make one meshing strategy more desirable than another:

- as a problem evolves, some choices of the mesh may be less susceptible to mesh distortion than others (for example, a problem that has severe shear would require a different mesh than a problem that has pure extensional flow);
- if boundary conditions change type along a boundary (such as reflective to outflow), then multiple blocks must be used; otherwise multiple parts in a single block will suffice (note that a single type of boundary condition can change properties, such as applied pressure or inflow conditions, for each part);
- because vectorization is over blocks, division of a problem into parts will result in faster execution than if the problem were divided into blocks; and
- use of multiple blocks for circular meshes (IGEOM=5 and 6), which eliminates the point of convergence in a typical circular mesh (IGEOM=3 or 4), will result in a significantly larger time step, and will more accurately calculate signals passing along the axis.

Choice of Numerical Options

The many specialized options of CAVEAT make it appear rather unwieldy to a first time user. As with mesh generation, probably the main difficulty encountered in choosing the numerical options is the large number of choices available. As before, experience is the best guide, but typically the choice of the numerical options is a balance between accuracy and robustness.

For example, a variety of numerical treatments of a given problem can be realized by varying the input parameter ALECOEF. In the absence of material interfaces and deformable boundaries, the pure Eulerian option (ALECOEF=0.0) can be used to keep initial mesh spacing fixed during the calculation. Note that the Eulerian option *cannot* be used in the presence of material interfaces and deformable boundaries. Also, a tradeoff exists between the increased likelihood of mesh distortion in Lagrangian (ALECOEF=1.0) or near Lagrangian calculations (ALECOEF almost equal to unity) and the decreased accuracy in highly rezoned calculations (ALECOEF almost equal to zero) due to the additional diffusion when advecting. But, the judicious use of adaptivity to features of interest can significantly improve the accuracy of highly rezoned calculations. Hence, a sophisticated application of the ALE capability with adaptivity can result in accuracy similar to that available from Lagrangian calculations, but without the limitations of a Lagrangian calculation. Because of the many input parameters that control the adaptivity option (§4.3.2), the addition of adaptivity to a problem should be considered *after* a problem is understood in the absence of adaptivity. Finally ALECOEF affects the efficiency of the calculation (the simulated time to computer time); in order of least execution time for the same size problem, the list is Lagrangian (ALECOEF=1.0), Eulerian (ALECOEF=0.0), and ALE (0<ALECOEF<1.0),

The choice between the first and second order numerical option is also a tradeoff between accuracy and robustness. A list of choices from most to least robust (and similarly from least to most accurate) is: first order (IORDER=1), second order with monotone limiting (IORDER=2, LIMGRAD=1), second order with van Leer limiting (IORDER=2, LIMGRAD=2), second order with van Leer limiting on all variables except velocity (IORDER=2, LIMGRAD=0). The stability factor (DTFAC) may be changed from the default value if difficulties are encountered, indicating a smaller value is needed, or if a larger time step is required (see next discussion for more guidance).

Further information on the advantages of the different numerical options can be found in the next section.

Common Causes for Early or Fatal Termination

There are two common indications that a calculation is in trouble: the time step becomes very small or a floating point error occurs or is about to occur but was caught by CAVEAT and the calculation terminated. These will be treated separately, although they can be related.

If the time step decreases below the input or default value of DTMIN, CAVEAT will terminate the calculation. In some cases, a decreasing time step during the course of a calculation may be unavoidable, as in the case of problems with converging geometries which result in smaller cells or with increasing energies which result in larger sound speeds and therefore, smaller time steps. In these situations, DTMIN must be decreased and the calculation continued.

One possibility that must be considered if the time step drops precipitously is instabilities, overshoots, or undershoots in the state variables in the calculation. Typically, this is not a problem when using first order numerical method. To examine this possibility, see the discussion above on numerical options and robustness. One class of problems that is susceptible to difficulties of this type, independent of the order of the calculation, consists of systems with a rapid expansion caused by a moving surface. In the region of the expansion the internal energy can undershoot because of too large a time step and become large and negative; for most equations of state, this results in a large sound speed and, therefore, a small time step. Usually, a smaller value of DTFAC or different initial conditions that do not begin with a velocity discontinuity can correct the difficulty.

Another cause for a decreasing time step is that the local distortion of the mesh may result in a highly distorted cell, which results in a small length scale and, therefore, a small time step. One solution to this problem is to consider a different initial mesh that is less susceptible to the distortion of the mesh caused by the flow. Another solution is to try different numerical options. Obviously, if the calculation is Lagrangian, then using the ALE capability probably will significantly correct the problem. For ALE calculations, distorted cells almost always lie along a material interface, and improvements mainly include changing the input parameters that control the remeshing. Some of the recommended changes are to increase the number of iterations in the remeshing (ITREZN), to try different values for the weighting of the curvature along material interfaces (CRVWT), and to decrease the contribution of Lagrangian movement in the remeshing (decrease ALECOEF). In some situations where a material interface is sheared in a direction normal to its orientation, such as near a vortex, there is little that can be done to continue the calculation. In these situations the interface tracking extension to CAVEAT [Johnson et al., 1990] must be used.

CAVEAT checks, after updating the cell variables (subroutine BNDHYD), if the density has become negative (§4.5f). Without this check and graceful termination, CAVEAT would inevitably experience an abnormal termination. Two situations can lead to a negative density. The most common is an extreme deformation of a cell that leads to a negative volume; in ALE calculations this generally occurs along material interfaces. Diagnosis of this problem can be accomplished by using a debugger to determine if a negative volume has occurred or by obtaining pictures of the mesh just before the time when the error condition occurred and looking for cells which are highly deformed. Another cause for a negative density is a negative mass at the end of the advection phase. This generally occurs during the first time step when DT0 is too large. The solution is to decrease DT0 in the input file.

Note that floating point errors also occur if a cell has zero volume, but because CAVEAT does error checking on DX (§4.5f), this can occur only if the user is creating a mesh by modifying the code or by reading in a mesh created by another code.

§5.4 UNITS USED IN CAVEAT

There are no set units used in CAVEAT, with the one exception noted below. The units set in the PROPRTY array (§4.4.3) determine the units that are required for the material initialization and numerical parameters such as the time step. As long as the units are used consistently, any set of units can be used.

The one exception to this is when the SESAME library is used (see §4.4.3 and §D). Because the SESAME tables assume a consistent set of units, when this equation of state is used in a problem, all units specified elsewhere in the problem must correspond to this set of units. The user can redefine this set by the input variables, CONVD, CONVP, and CONVE (§4.4.3). The defaulted values of these input variables result in the units of g, cm, and μs for mass, length, and time.

SECTION 6

EXAMPLE PROBLEMS

This section contains example problems for CAVEAT. The test problems were chosen to illustrate how to exercise the various options available in CAVEAT within the context of simple problems. Hence, none of the test problems fully communicate the complexity of problems that can be addressed with the code; more complex problems have been published elsewhere (Sandoval, 1987; Cagliostro, et al., 1988) and will be documented in future publications. Because of the limitations on the number of figures, the graphical results of the test problems that are included with the text are minimal, but because the input files are given, the problems can be executed and examined in detail by users of the code.

§6.1 SHOCK TUBE PROBLEM

This example problem, a one-dimensional shock tube with a density step, is one of the simplest problems one might examine with CAVEAT. The description of the problem is given in Fig. 6.1-1 and is also the problem used as an example in §5.

The objective of this example problem is to illustrate the large number of alternatives available in CAVEAT and their effect on the accuracy and computational efficiency of the solution. In Fig. 6.1-2, the time history of the density for the problem is given. The results were obtained using the input file in Table 6.1-1. Because the calculation uses a very fine mesh, the solution is considered to be identical to an analytical solution to the problem and will be the basis for comparison for accuracy of the calculations. As one observes from Fig. 6.1-2, the result of this "simple" test problem are quite complex

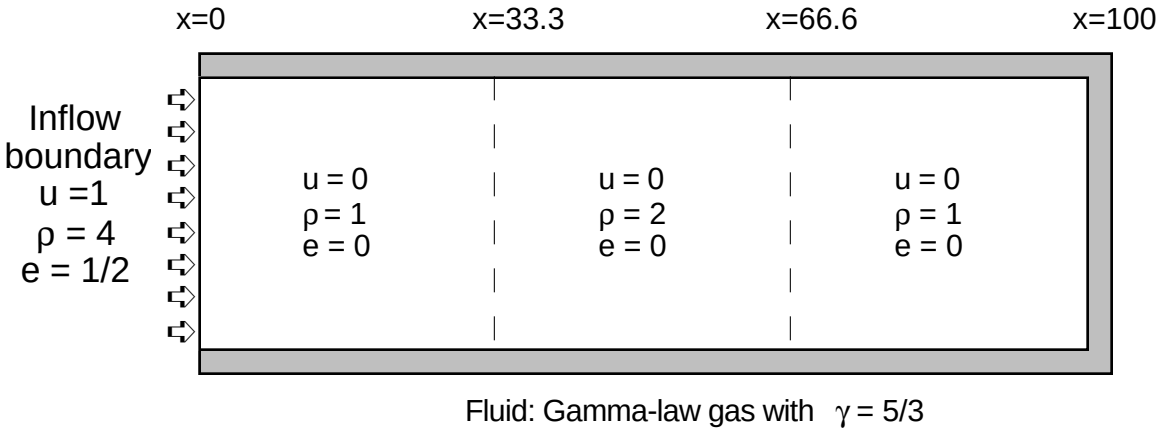


Figure 6.1-1. Description of a shock tube problem with a step in density.

at later times as the many rarefactions interact. We shall see that many of the discontinuities (both shocks and rarefactions) that are resolved in the fine mesh are approximated to different degrees on a coarser mesh and with different various options.

The input file and results for the reference problem that will be the basis for trying various options in CAVEAT are given in Table 6.1-2 and Fig. 6.1-3. The numerical choices in this test problem are described in detail in §5. The differences between this simulation and the simulation presented in Fig. 6.1-2 are the coarser mesh (a factor of ten larger) and first order treatment of the spatial differencing. As can be seen in Fig. 6.1-3, the changes result in a poorer description of the discontinuities in the problem, although densities away from the discontinuities are mostly reproduced. Note that the mesh movement from the ALE treatment results in a rounding of the initial density drop at a time of 45.0; even though there is no movement of the material ahead of the shock, the mesh is being adjusted throughout the problem and consequently the density discontinuity is being diffused.

In Fig. 6.1-4, the reference simulation is redone with the input parameter ALECOEF = 0.0 added to the input file in Table 6.1-2. Two major observations can be made. The initial density discontinuity is not disturbed until the arrival of the shock because there is no movement of the mesh. The discontinuities at later times are smeared out even more than the ALE calculation (the reference problem in Fig. 6.1-3); this is a consequence of the additional movement of the material through the mesh. In the ALE calculation the large default value of ALECOEF (0.9) results in the mesh being moved with the shock and consequently the mesh resolution is finer in the region of interest at later times.

Table 6.1-1.

Input file for the shock tube problem with a fine mesh.

```
Test Problem 1.fine
$input

 nprts=3,
 ncell(1,1)=330, 330, 330,
 ncell(1,2)=1,
 dx(1,1)=3*0.10101010101,
 dx(1,2)=1.0,

 rho0   = 1., 2., 1.,
 pr0    = 3*0.,
 matnum =  1,  1,  1,

 proprty(1,1)=1., 1., 0., 1.333333, 1.66666666,

 ibc(3) = 4,
 ubc(1,1,3) = 1.,     pbc(1,3) = 1.333333333,
 rhobc(1,3) = 4.0,  matbc(1,3) = 1,
 iorder=2, limgrad=1,
 dt0=0.01,
 twfin=80., twfilm=20., 80., 05., igas=1,$
```

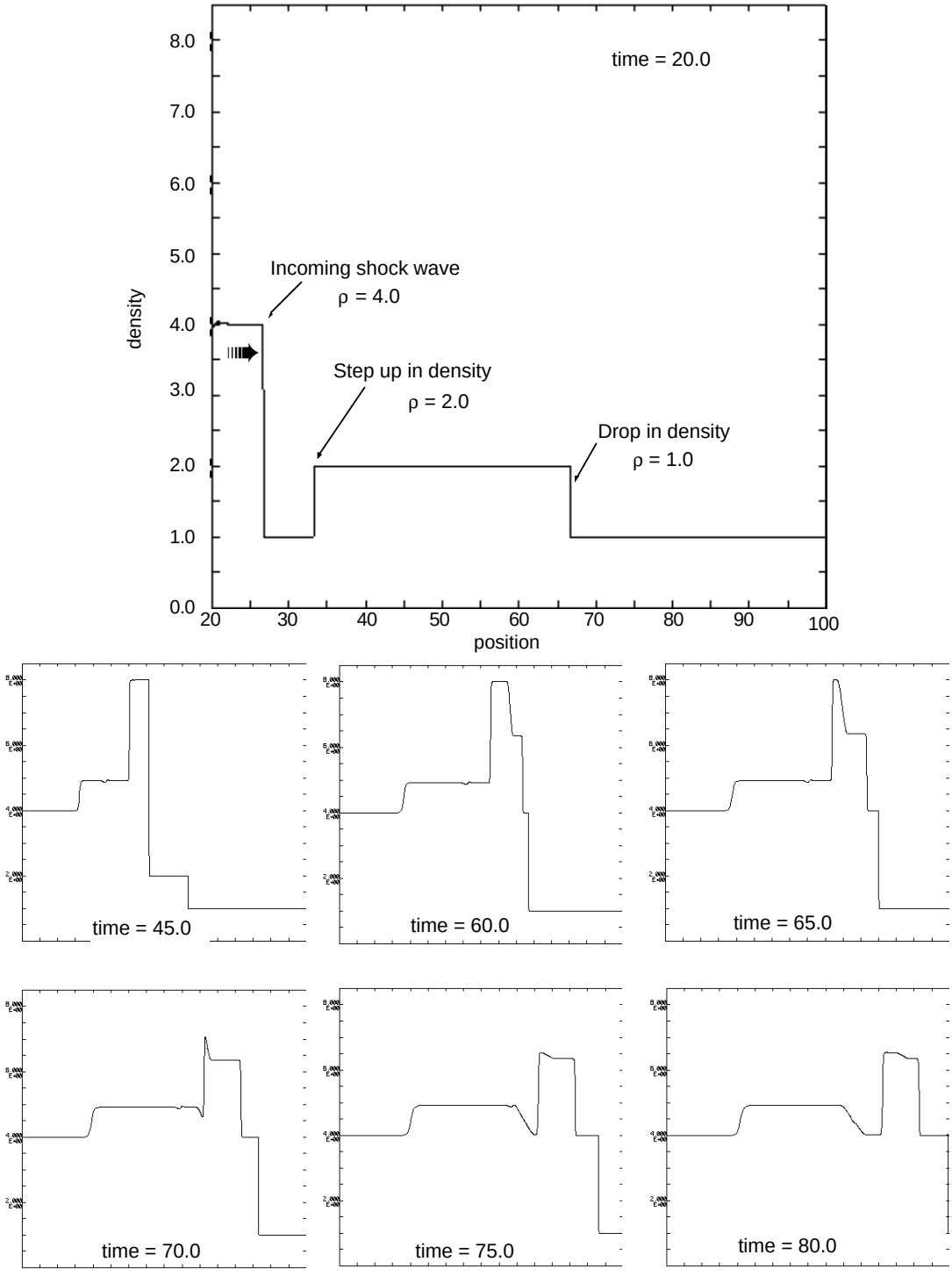


Figure 6.1-2. Density plots versus position for the shock tube problem for the input file in Table 6.1-1. The top figure shows the axes in detail; the identical axes are used in all of the figures above. Any consistent set of units can be chosen for the variables (§5.4).

Table 6.1-2.
Input file for reference simulation for the shock tube problem.*

```
 1  Test Problem 1.reference
 2  $input
 3
 4   nprts=3,
 5   ncell(1,1)=33, 33, 33,
 6   ncell(1,2)=1,
 7   dx(1,1)=3*1.0101010101,
 8   dx(1,2)=1.0,
 9
10   rho0  (1)  = 1., 2., 1.,
11   pr0   (1)  = 3*0.,
12   matnum(1)  =  1,  1,  1,
13
14   proprty(1,1)=1., 1., 0., 1.333333, 1.66666666,
15
16   ibc(3) = 4,
17   ubc(1,1,3) = 1.,     pbc(1,3) = 1.333333333,
18   rhobc(1,3) = 4.0,  matbc(1,3) = 1,
19
20
21   dt0=0.1,
22   twfin=80., twfilm=0.0, 80., 5., igas=1,$
```

* Details of the numerical choices for this simulation are described in §5.1. The line numbers are not part of the input file but are used for reference in later simulations.

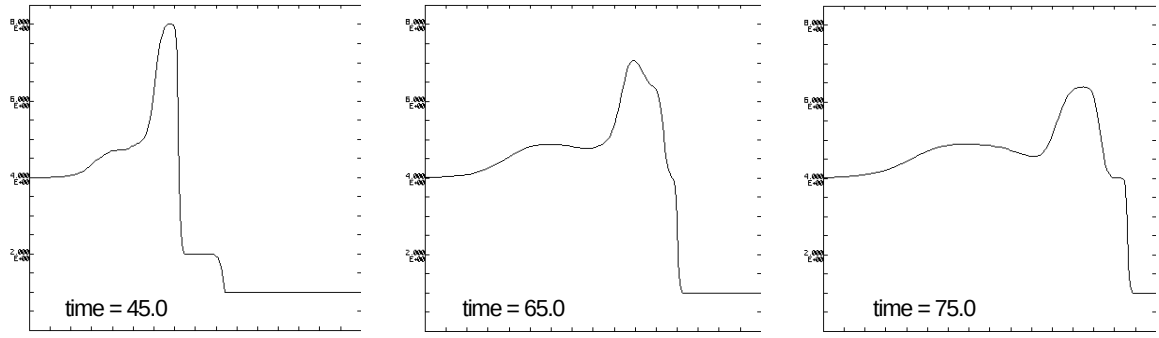


Figure 6.1-3. Density versus position plots for the reference simulation using the input file in Table 6.1-2. Note that the axes are identical to Fig. 6.1-2

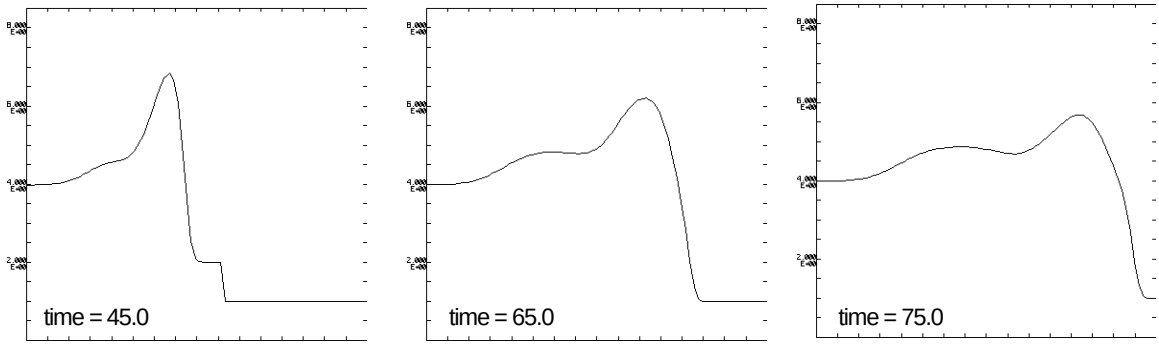


Figure 6.1-4. Density versus position plots for the reference simulation with ALECOEF = 0.0 added. Note that the axes are identical to Fig. 6.1-2.

For completeness, the simulation results are shown for ALECOEF = 1.0 (Lagrangian treatment) in Fig. 6.1-5. Because the left, inflow boundary moves with the fluid, the resulting problem at later times is not equivalent to the previous simulations, because rarefactions interact with the left boundary. This simulation is also identical to specifying the left boundary as a rigid piston moving with a constant velocity (replace lines 16-18 in Table 6.1-2 with IBC(3) = 9, UBC(1,1,3) = 1.0).

All of the previous simulations used the first order treatment of the spatial differencing (IORDER = 1, the default value). In Fig. 6.1-6, the results are presented for the reference problem with the addition IORDER = 2 added to the input file in Table 6.1-2. This selects second-order spatial differencing and Van Leer limiting (LIMGRAD = 2, the default value). In Figs. 6.1-7 and 6.1-8, the results are presented for the other two types of limiting with the second order treatment, LIMGRAD = 1 (monotone limiting) and LIMGRAD = 0 (no limiting). Overall the second-order treatment results in significantly reduced diffusion over the first order treatment. A comparison of the different limiting of the second order treatment shows that there is a tradeoff between reduced diffusion and reduced overshoots at discontinuities; no limiting having the least diffusion, and monotone limiting having the least overshoots.

The adaptive capability of CAVEAT can be illustrated with this example problem. For many shock problems the option to adapt the mesh to pressure or density gradients, a standard option of CAVEAT (§4.3.2c), can greatly enhance the efficiency of the calculation. In this problem, the mesh is refined in regions of large density gradient (WFGR=-2.0). Furthermore, the option to limit effect of the magnitude of the gradient on the weight function (WFLM=0.05) is used in order to adapt to the rarefactions as well as the shocks.

Two simulations are presented using the same settings for the adaptive options (WTAMPL=4.0, WFGR=-2.0, WFLM=0.05, NSMOOTH=15, ALECOEF=0.01, ITREZN=9). Fig. 6.1-9 displays the results for the first-order calculation and Fig. 6.1-10 for the second-order spatial differencing with van Leer

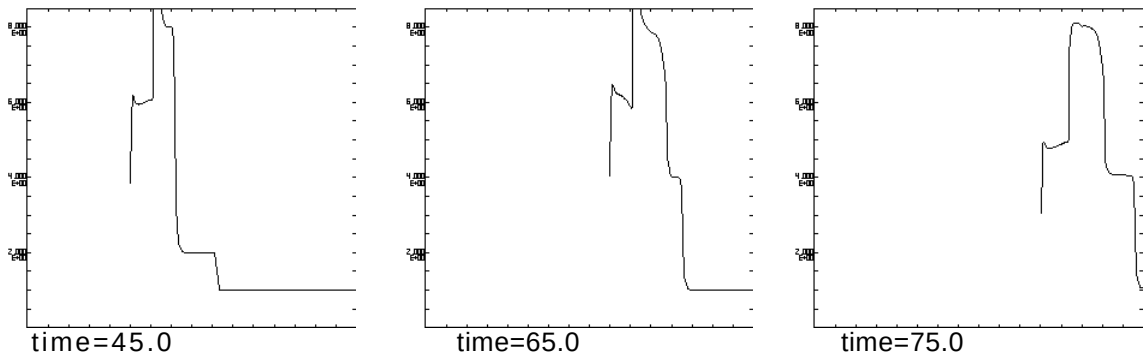


Figure 6.1-5. Density versus position plots for the reference simulation with ALECOEF = 1.0 (Lagrangian) added and with a rigid piston boundary condition (see text for changes to the input file). Note that the axes are identical to Fig. 6.1-2.

limiting (IORDER=2, LIMGRAD=2). The dynamic refinement of the mesh to the density gradients can be observed in the mesh spacing given at the bottom of the figures. Comparisons to the respective figures without adaptivity illustrate marked improvements that results from the finer mesh near density gradients. In particular, the second-order calculation is approaching the accuracy in simulation with the very fine mesh in Fig. 6.1-2 but taking 22 times less computational time (see Table 6.1-3). Further improvements in running times and minimizing diffusion can be achieved by using the option to subcycle on the Lagrangian phase (see NADVSKP in §4.3.1b).

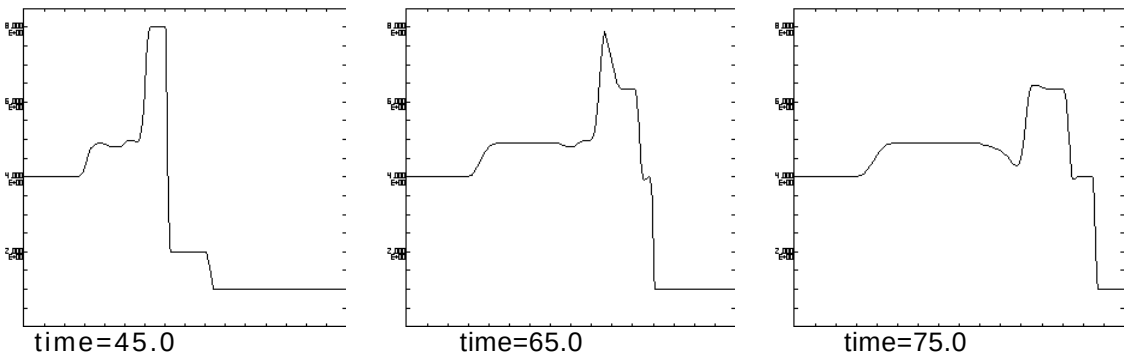


Figure 6.1-6. Density versus position plots for the reference simulation with IORDER = 2 added and van Leer limiting (LIMGRAD = 2, the default). Note that the axes are identical to Fig. 6.1-2.

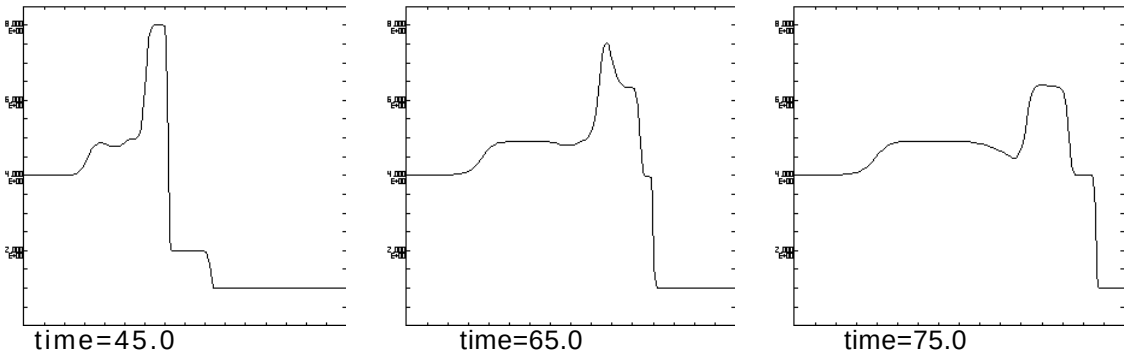


Figure 6.1-7. Density versus position plots for the reference simulation with IORDER = 2 and LIMGRAD = 1 (monotone limiting) added. Note that the axes are identical to Fig. 6.1-2.

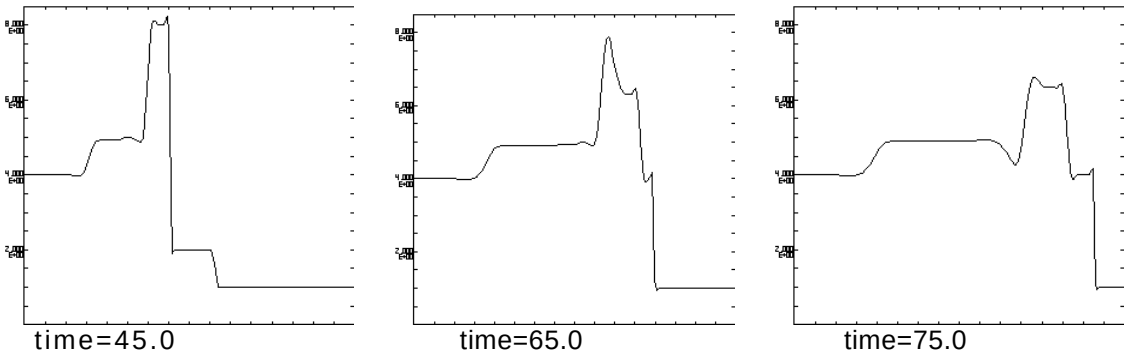


Figure 6.1-8. Density versus position plots for the reference simulation with IORDER = 2 and LIMGRAD = 0 (no limiting). Note that the axes are identical to Fig. 6.1-2.

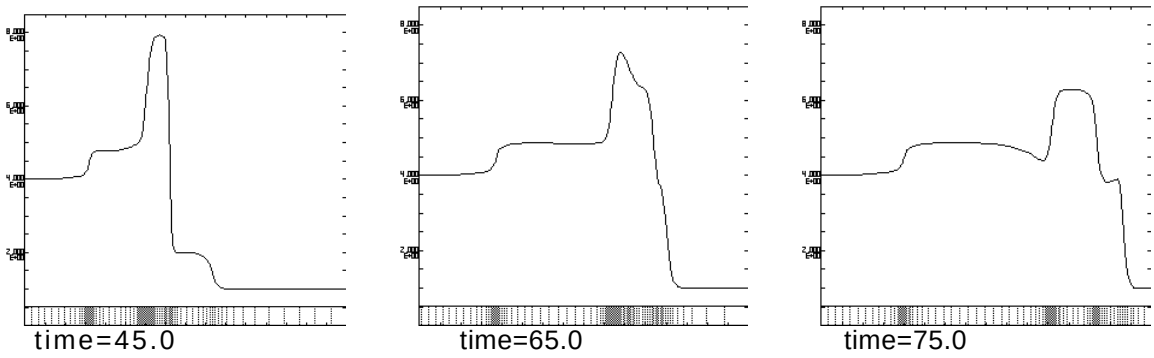


Figure 6.1-9. Density versus position plots for the reference simulation with adaptivity to density gradient (see text for parameter settings). This figure should be compared to Fig. 6.1-3. The spacing for the one-dimensional mesh is given at the bottom of the graph. Note that the axes are identical to Fig. 6.1-2.

A final example illustrates the effect of changing the single material problem to two materials. The reference input file in Table 6.1-1 is modified to add an additional material definition and to make the density step a different material. The line

```
property(1,2)=1., 1., 0., 1.333333, 1.66666666,
```

is added to the input file, and line 12 is replaced by

```
matnum(1)  =  1,  2,  1,  .
```

In addition, the following input parameters were added to the reference input file:

```
iorder=2, limgrad=1, alecoef=0.01,   .
```

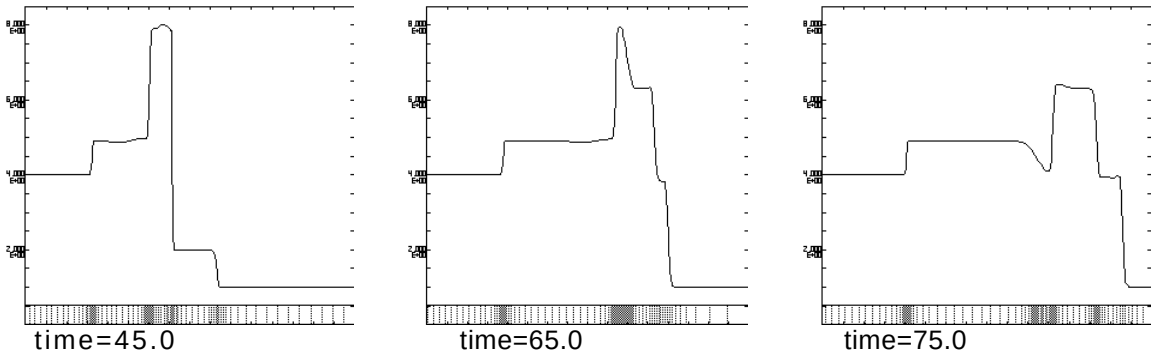


Figure 6.1-10. Density versus position plots for the reference simulation with adaptivity to density gradient (see text for parameter settings) and IORDER=2 (second order with van Leer limiting). This figure should be compared to Fig. 6.1-6. The spacing for the one-dimensional mesh is given at the bottom of the graph. Note that the axes are identical to Fig. 6.1-2.

The results of the two material simulation are given in Fig. 6.1-11. The primary difference in this calculation in comparison to all previous calculations is that the material interfaces at the right and left of the density step prevent the mesh from being moved through these interface locations, as can be seen in the mesh spacing at the bottom of the figures. This has the advantage that no diffusion of the density pulse can occur before the arrival of the shock. But it has also the disadvantage that the mesh spacing across the interface can be very abrupt and result in inaccuracies near the interface.

The timing of the all the simulations presented for this test problem are summarized in Table 6.1-3. Many observations can be made on the effect of the various options on the grind time (CPU/cell-cycle) and on total CPU, coupled with the earlier conclusions on the relative accuracy. For example, even though the Lagrangian simulation has the lowest grind time, the total CPU time is not the lowest because the smaller cell sizes result in smaller time steps. Another observation is that the second-order spatial treatment is computationally more efficient than reducing the cell size by a factor of two and using the first-order treatment. As observed earlier, the most sophisticated use of CAVEAT, the second-

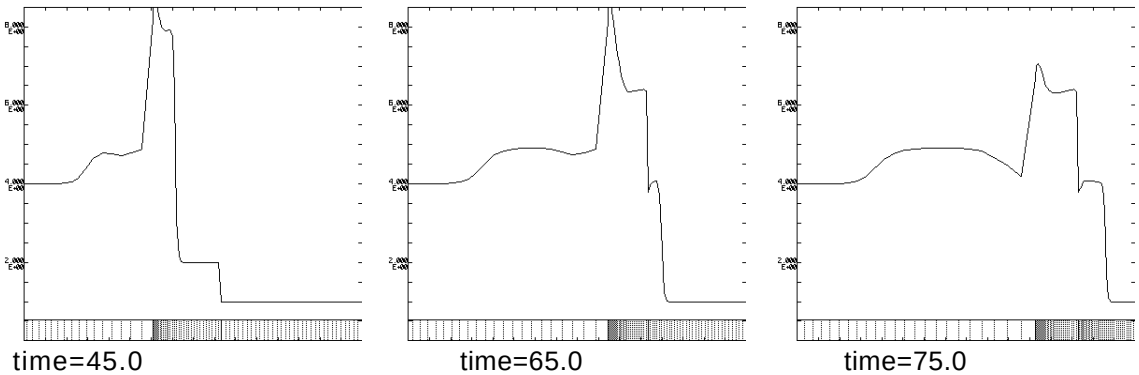


Figure 6.1-11. Density versus position plots for the reference simulation with two materials (see the text for modifications of the input file in Table 6.1-2). The spacing for the one-dimensional mesh is given at the bottom of the graph. Note that the axes are identical to Fig. 6.1-2.

Table 6.1-3

Comparison of the execution time (total CPU excluding graphics output in μseconds) and the grind time (CPU/cell-cycle in seconds) on a CRAY YMP.

Option	CPU/cell-cycle	Total CPU	Figure
Fine mesh	43	362	6.1-2
Reference problem 0.0<ALECOEF<1.0 *	36	1.44	6.1-3
ALECOEF=0.0	25	0.99	6.1-4
ALECOEF=1.0	13	1.89	6.1-5
IORDER=2, LIMGRAD=2	59	4.22	6.1-6
IORDER=2, LIMGRAD=1	57	3.97	6.1-7
IORDER=2, LIMGRAD=0	57	4.11	6.1-8
Adaptive, first order	52	11.2	6.1-9
Adaptive, second order	79	15.9	6.1-10
Two material problem	57	9.88	6.1-11

* Total CPU is given for the reference problem; the grind time is correct for the range of ALECOEF shown.

order and adaptive calculations, result in the best compromise between accuracy and computational efficiency. Finally the addition of two materials to the test problem does not change the grind time over the single material problem, because the code is written so that any cell face may be an interface between materials. The only additional computational time required in multimaterial problems is the tangential rezoning along material interfaces. As mentioned earlier, the total CPU time increases in the two material problem as a consequence of the smaller cell sizes near the material interface.

All of the options presented in this subsection also apply to multidimensional problems. Similar tradeoffs occur in run times and amount of diffusion experienced by the different options. The test problems in the rest of this section address two-dimensional examples and are presented in less detail, but interested users can modify the input files that are provided to examine the effect of the various options discussed in this subsection.

§6.2 SMEAR BOX ADVECTION PROBLEM

In this problem, we test the ability of CAVEAT to handle a pure advection situation. The problem merely involves advecting a box of high density material (see the problem description in Fig. 6.2-1) at 45 degrees through the mesh. The problem is pure kinematics with no hydrodynamics. Because the box is initially in pressure equilibrium with the surrounding fluid, the velocity is uniform everywhere, and all boundaries are outflow (§3.2.4a), the pressure field remains unchanged and the velocity field retains its initial value of 0.01 in both the vertical and horizontal directions throughout the course of the calculation. The exact solution to this problem transports the high density box through space without diffusing the initial discontinuity in the density. Therefore, this problem is a test of the accuracy of the transport of gradients during advection.

This problem was computed with both a fixed pure Eulerian mesh and a mesh that is adapted to density gradients. The input file for the pure Eulerian problem is given in Table 6.2-1. Density contour plots at the initial time and when the box reaches the upper left corner are given in Figs. 6.2-2 and 6.2-3. A comparison of the two figures shows that noticeable diffusion has occurred. (A similar calculation with first-order spatial differencing would result in an almost circular density contours at the same late time.)

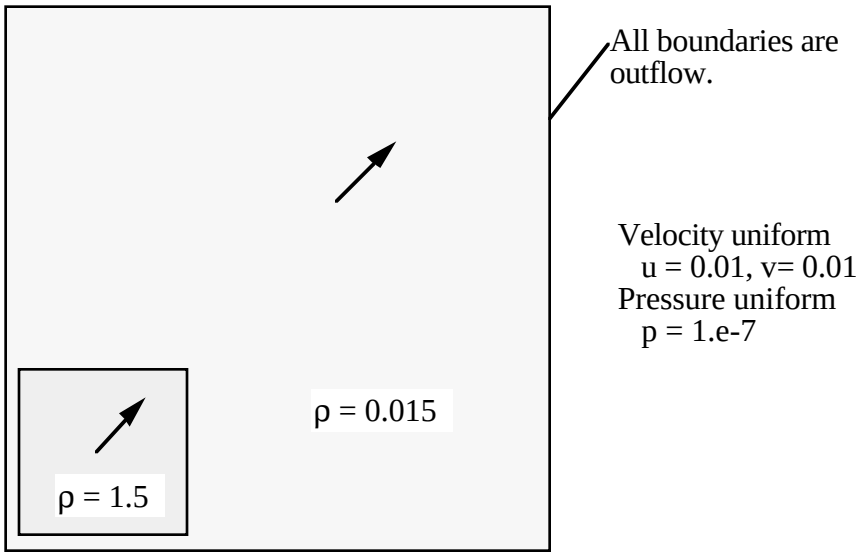


Fig. 6.2-1. Initial setup for the smear box problem. Note that an "outflow" boundary can also allow material to flow into the problem; the zero-gradient "outflow" boundary (§3.2.4a) at the lower and left boundary results in flow of material into the problem that is identical to the material adjacent to these boundaries.

Table 6.2-1

The input file for the smear-box problem using an Eulerian mesh.

```
SMEARBOX TRAVELING DIAGONALLY
$input
 nprts=3, 3,
 ncell(1,1)=2, 20, 50,
 ncell(1,2)=2, 20, 50,
 dx(1,1)=1.0, 1.0, 1.0,
 dx(1,2)=1.0, 1.0, 1.0,
 igeom=1,

 rho0=4*0.015, 1.5, 4*0.015,
 pr0= 9*1.e-7,
 uc0= 18*0.01,
 matnum=9*1,

 ibc=3,3,3,3,

 iorder=2, limgrad=2, alecoef=0.00,

 dtfac=0.90, dtmin=1.0e-5, dtmax=100., dt0=10.,
 twfin=4.5e3, twfilm=0.0, 4.5e3, 0.5e03, $
```

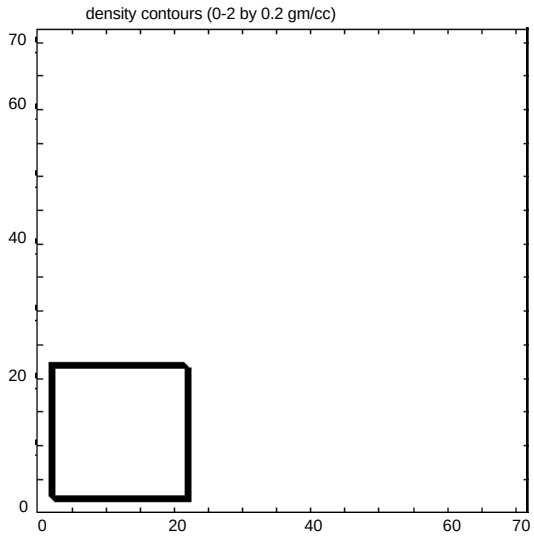


Fig. 6.2-2. Density contours at initial time.

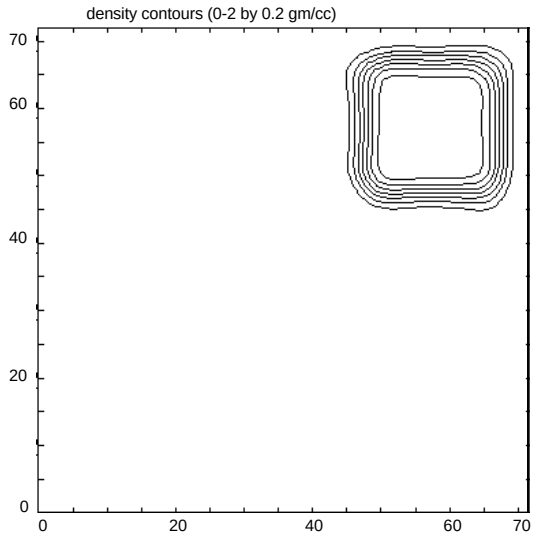


Fig. 6.2-3. Density contours at a late time.

We can greatly improve the results by introducing adaptivity to the problem. The following changes are made to the input file in Table 6.2-1. The value of 0.0 for the ALECOEF is changed to 0.005. And the following line is added to the input file:

```
nsmooth=3, wfgr=-8.0, wtampl=5., wflm=0.005,
```

at any location. The parameter ALECOEF was changed to retain a nearly Eulerian mesh and to reduce the effect of the bulk fluid motion on the adaptivity. The rezone weight function was specified to respond to density gradients (WFGR=-8.0), but the maximum gradient to be considered in the rezone was set to a low value (WFLM=0.005). This permitted regions with low gradients, particularly nearer to the corners of the box, to participate in the adaptivity. Several passes through the mesh were taken to smooth the resulting weight function distribution (NSMOOTH=3) before the ratio of maximum to minimum weight values was restricted (WTAMPL=5.).

The density contour and the mesh plot for this run are shown in Figs. 6.2-4 and 6.2-5. A comparison of the density plots in for the Eulerian and adaptive calculations shows the improved accuracy of the adaptive calculation. The mesh plot illustrates the power of adaptivity to pull cells in from the surrounding fluid to regions of high density, thereby reducing the amount of diffusion of the density gradients.

§6.3 BLAST WAVE PROBLEM

Consider a uniform medium in ν dimensions (ν = 1,2,3), with zero initial pressure. At time t=0,

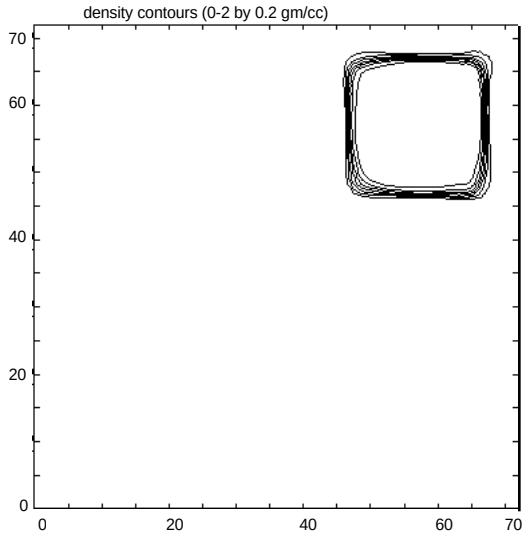


Fig. 6.2-4. Density contours at late time for adaptive mesh calculation.

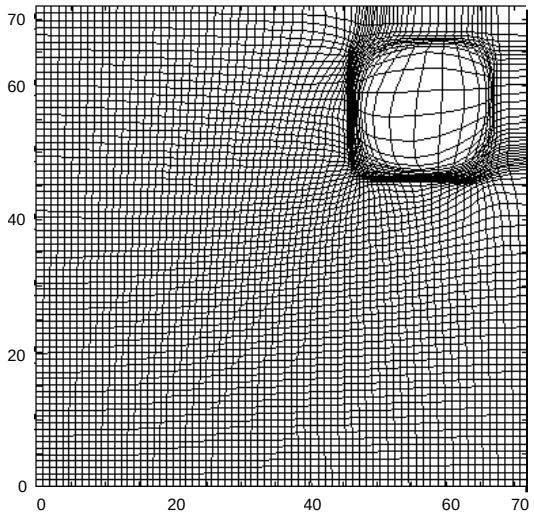


Fig. 6.2-5. Mesh at late time for adaptive mesh calculation.

a fixed amount of energy E is deposited at the origin, r=0. As time increases, a blast wave is expands away from the origin. Because the initial pressure is zero, the shock associated with the blast wave is infinite in strength, and a similarity solution for the post shock profile can be obtained. This solution was first found by Sedov [Sedov, 1959] for a gamma-law gas (§2.2) and is particularly useful for examining the accuracy of multidimensional computational schemes. Here we run a two-dimensional calculation and compare to the analytical solution.

The post-shock conditions are [Sedov, 1959]

$$v_2 = \frac{2}{\gamma+1} c$$
$$\rho_2 = \frac{\gamma+1}{\gamma-1} \rho_1$$
$$p_2 = \frac{2}{\gamma+1} \rho_1 c^2$$

where for cylindrical symmetry ($\nu = 2$ and the energy is deposited along a line)

$$r_2 = \left(\frac{E}{\rho_1}\right)^{1/4} t^{1/2} \quad ,$$

and

$$c = \frac{1}{2}\left(\frac{E}{\rho_1}\right)^{1/4} t^{-1/2} \quad .$$

Here, ρ_1 is the initially uniform density of the gas; v_2, ρ_2, and p_2 are the post shock velocity, density, and pressure at time t; and r_2 and c are the radial shock position and shock speed at time t.

An illustrative choice for a computational mesh is one in which the plane coordinates are $\Delta x = \Delta y = 2/N$ where N is an odd integer. A single unit of energy is deposited in the in the central cell of the mesh. With $\rho_1 = 1$, we expect $r_2 = 1$ at $t = 1$. Fig. 6.3-1 shows the mesh at $t \approx 1$ for the CAVEAT in the Lagrangian (no rezone) mode with N=21. The calculation is run second order with monotone limiting. The input file is given in Table 6.3-1. The corresponding density curve, plotted through the center of the row of cells, is shown in Fig. 6.3-2. Note that the symmetry is preserved. Figs. 6.3-3 is the contour plot of the density.

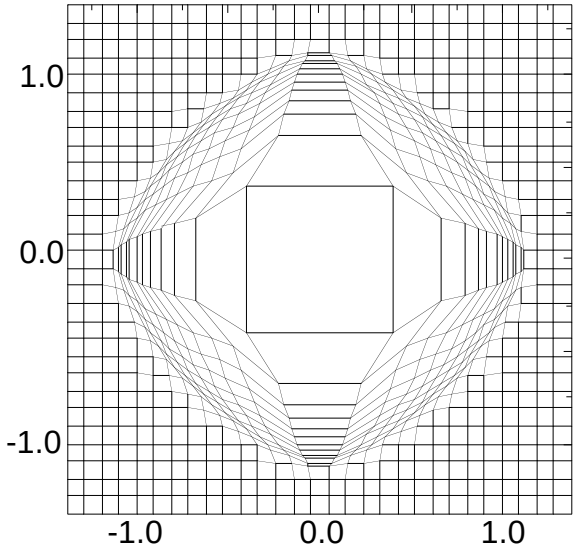


Fig. 6.3-1. Mesh plot at t=1.018.

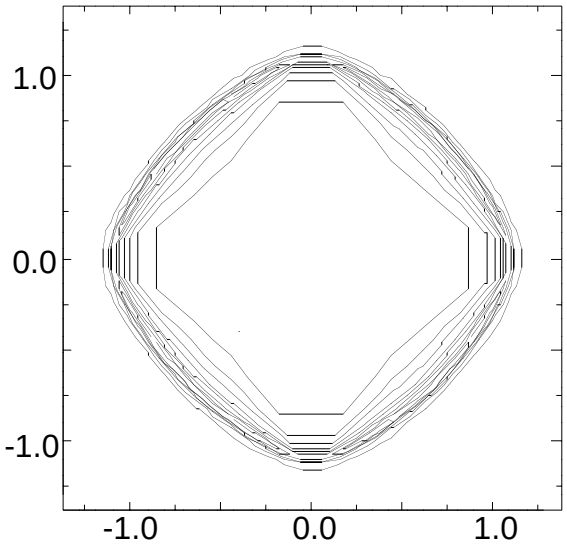


Fig. 6.3-2. Density contours from 0.03 to 4.33 by 0.43.

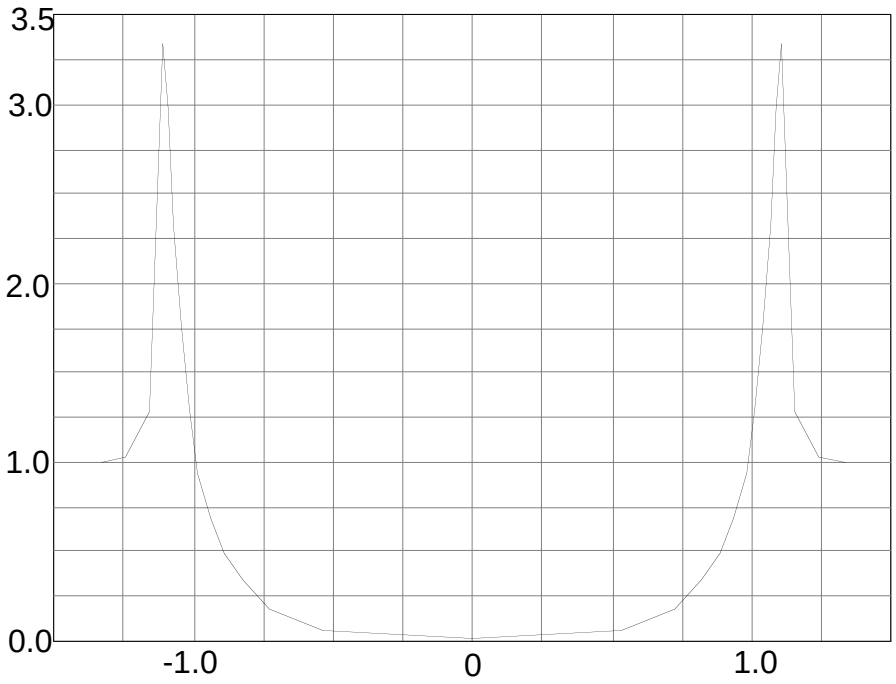


Fig. 6.3-2. Density along the center row of cells.

The blast wave should be spherical at all times. The origin of the flatness of the blast wave in Fig. 6.3-1 along the diagonals is due to the generation of spurious vorticity for shocks moving oblique to the mesh orientation. This difficulty is a consequence of the current implementation of the Godunov method in a skewed mesh or when shocks move obliquely to the mesh. This is discussed in more detail in §7.2. The alternative numerics either in the full version of CAVEAT, the TVD scheme [Johnson et al., 1990], or the global Godunov method (§7.2) exhibit much less, if any, flattening of the blast wave along the diagonals.

In this sample calculation $\gamma = 1.4$; so that the peak density should be 6, and the error is assessed relative to this value. We define the error E

$$E = \frac{|\rho_{max}-6|}{6}$$

and plot E versus N in Fig. 6.3-4. The slope of the line in Fig. 6.3-4 is -1, corresponding to first-order spatial accuracy. Thus, the CAVEAT scheme exhibits first-order accuracy, which is the best that can be expected of a monotone-preserving scheme in two dimensions [Goodman and Le Veque, 1985].

Table 6.3-1

The input file for the blast wave problem in cylindrical coordinates.

```
Blast wave problem
 $input
      nprts(1,1)=3, nprts(2,1)=3,
      ncell(1,1,1)=14,1,14,
      ncell(1,2,1)=14,1,14,
      x0(1,1)=-1.3810,-1.3810,
      dx(1,1)=3*0.0952, dx(1,2)=3*0.0952, igeom=1,
      proprty=1.,1.,0.,1.2,1.4,
      rho0(1,1)=9*1.000, pr0(1,1)=0.,0.,0.,0.,044.10,0.,0.,0.,0.,
      alecoef=1.0, iorder=2, i2dplot=1, iline=1,15,
      limgrad=1, dt0=1.e-3, dtfac=0.90,
      twfin=1.0, twfilm=1.0,1.0,0.5  $
```

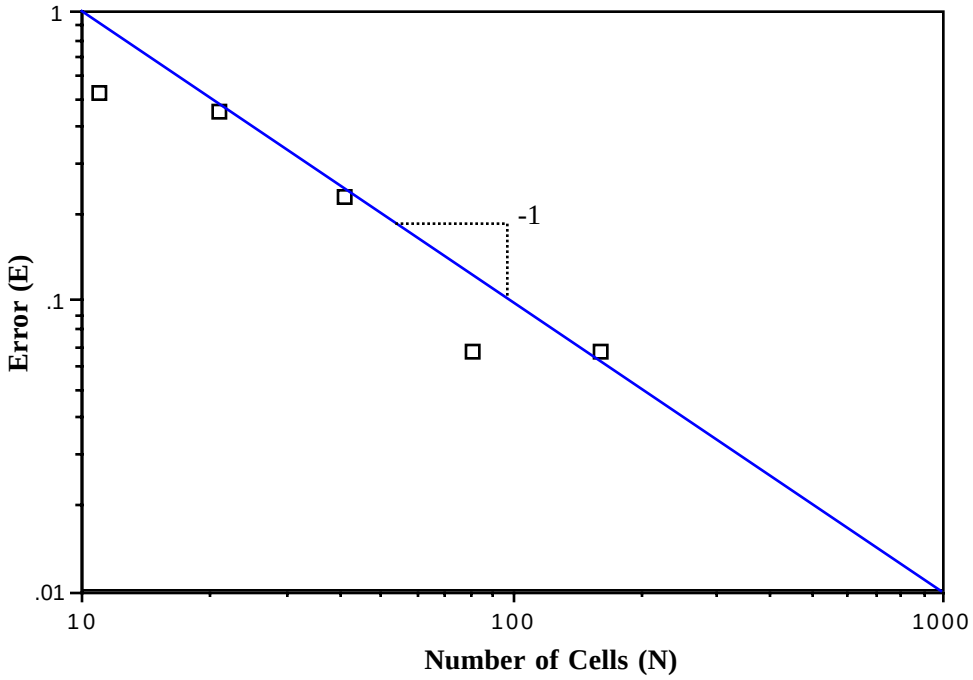


Fig. 6.3-4. Plot of the relative error in the maximum density versus the number of cells.

§6.4 SHOCKWEDGE INTERACTION PROBLEM

This problem presents a comparison of experimental data with a CAVEAT calculation. A Mach 3.47 shock interacts with a wedge inclined at 30 degrees to the flow and forms a Mach reflection. Fig. 6.4-1 shows the problem setup using CAVEAT. A figure which represents the experimental interferogram at a late time is given in Fig. 6.4-2 and is taken from a report detailing many similar comparisons and detailed hydrodynamic calculations using CAVEAT (Sandoval, 1987). The geometry and materials, as well as other details of the problem, can be found in the original report.

An update is given in Table 6.4-1 which modifies the source code to produce a mesh with the desired wedge above the 40th j-line.

The input file, given in Table 6.4-2, specifies the calculation is Eulerian with secord-order spatial differencing and Vanleer limited gradients and the inflow conditions are those for a fully supported Mach 3.47 shock into a gamma-law gas (§2.2).

In Fig. 6.4-3 the density contour plots are shown at a time of 0.56 μs. This plot compares very well to the experimental results in Fig. 6.4-2. Similar agreement will be obtained by using lower-order differencing options, though the gradients will not be as steep.

Table 6.4-1

The update* to the source code which produced a mesh with a wedge of 30°.

```
*d,meshgen.66,69
      x1 = 0.
      do 30 i=i1,icell(iprt+1,1,iblk)
         if(j.eq.40)yst=xv(1,j,2)
         if(j.gt.40)then
              xv(i,j,2) = x2
              xv(i,j,1) = x1 + (x2-yst)*tan(.523598776)
         else
              xv(i,j,1) = x1
              xv(i,j,2) = x2
         endif
      x1 = x1 + (1.7-xv(1,j,1))/52
```

* This update replaces lines 66 through 69 of the subroutine MESHGEN.

Table 6.4-2

The input file for the shock-wedge problem.

```
 shock/wedge interaction ang=30 M=3.47
$input
  alecoef = .00, iorder=2, limgrad=2,
  igeom = 1, ncell(1,1,1) = 52, ncell(1,2,1) = 198,
  dx(1,1,1) = .03269231, dx(1,2,1) = .0126262626,
  matnum(1,1) = 1, proprty(1,1) = 1.,1.,0.,1.2,1.4,
  rho0(1,1) = 1.,  pr0(1,1) = 0.7142857,
  rhobc(1,1,1)=4.42002, ubc(2,1,1)=2.89385,
  ibc(1,1) = 4,1,1,1, pbc(1,1,1)=11.53729,
  dtmin = 1.e-8, dtmax = .1, dt0 = 1.e-5, ncycmax = 10000,
  twfin = .60, twfilm = 0.,.66,.04, igas=1,  i2dplot=0,  $
```

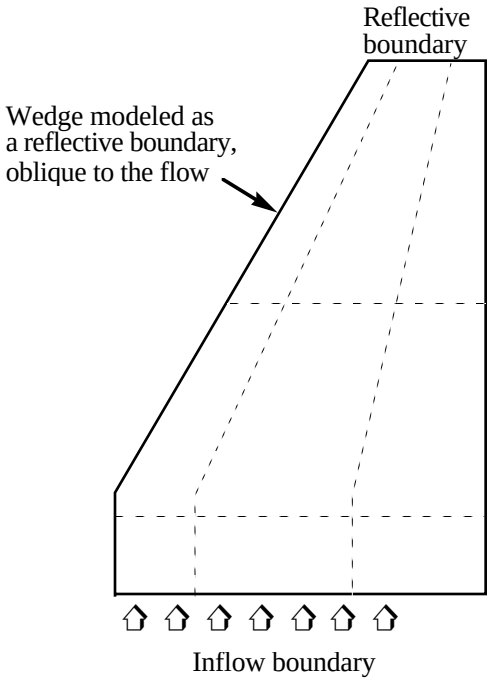


Fig. 6.4-1. Problem setup for the wedge problem. The dashed lines are two representative mesh lines for each of the two mesh directions.

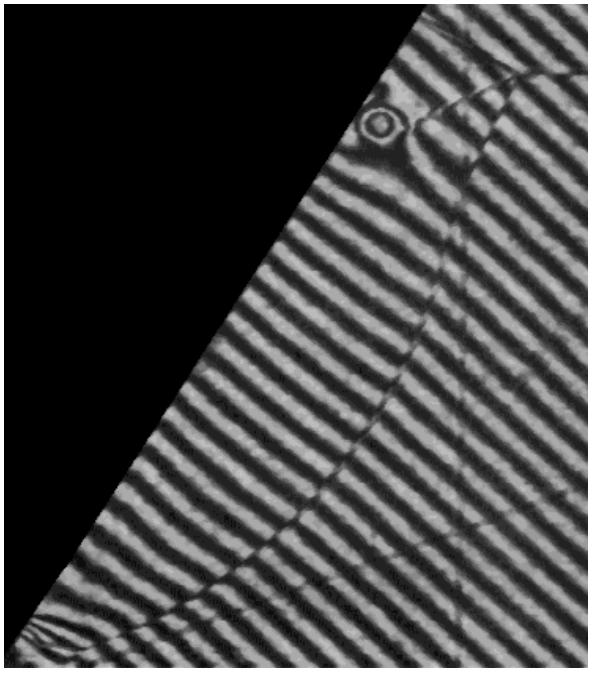


Fig. 6.4-2. Detail of an experimental interferogram of a Mach reflection of a shock off of a wedge. Abrupt changes in the interference locate steep gradients in density.

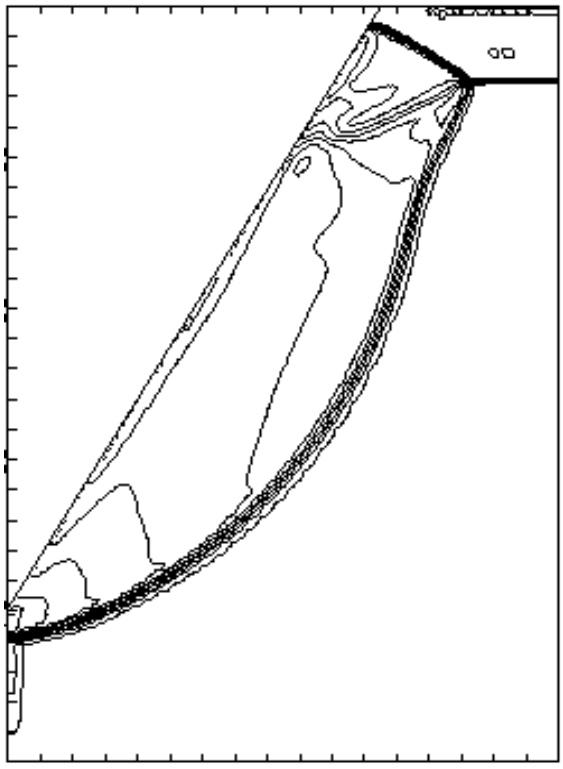


Fig. 6.4-3. Density contour plot from the CAVEAT simulation at a time of 0.5601 μs. The contours are from 0 to 10 by 0.1 gm/cc.

§6.5 ELLIPSE PROBLEM

This example problem illustrates the ability of continuous rezoning in CAVEAT to accommodate very large deformations in the problem domain. In addition choices are made in the numerical options to minimize the diffusion of steep gradients. Figure 6.5-1a displays the initial polar mesh configuration. An ellipsoid of beryllium with a ratio of major axis to minor axis of 1.5 is compressed with a pressure that ramps from zero to 1000 Mbar in 0.05 μs and then remains constant. Table 6.5-1 contains the updates required to provide the ramped pressure and to produce the ellipsoidal mesh.

Table 6.5-2 contains the input file. In the input note the specification of the polar mesh (IGEOM=4), the use of a SESAME equation of state for beryllium (PROPRTY(1,1)=8), and that the problem is run second-order (IORDER=2) with van Leer limiting (LIMGRAD=2). To minimize numerical diffusion associated with advection, the calculation is performed very near to the Lagrangian limit with ALECOEF=0.99 and NADVSKP=10. The latter specification implies that rezoning and advection are performed only at every tenth time step (§4.3.1b). At other time steps the calculation is strictly Lagrangian with no adaptive modifications of the mesh and no advection. This choice reduces the diffusion from advection and reduces the total computational time. To reduce even further the numerical diffusion, we use ANTIDIF=1.0 (§4.3.1b).

The specifications for the adaptive rezoning include only one iteration of the adaptive algorithm (ITREZN=1), no smoothing of the weight function (NSMOOTH=0), use of the special rezoning option on the boundaries (IBCREZN=0,1,1,1), moderate weighting of boundary curvature to provide closer spacing of mesh points in intervals of high curvature (CRVWT=2.5), and a maximum amplitude for the weight function (WTAMPL=8.0) that yields a relatively uniform initial mesh in polar geometry.

Figure 6.5-1 shows the time development of jetting along the z-axis as a consequence of the asymmetrical collapse. The use of the special rezoning option for the boundaries (IBCREZN=1) is essential to keeping a moderate density of mesh in the jet. This calculation demonstrates the power of the ALE method in CAVEAT to maintain a reasonably uniform mesh in the presence of extreme deformation.

Table 6.5-1

Update file for ellipse problem*.

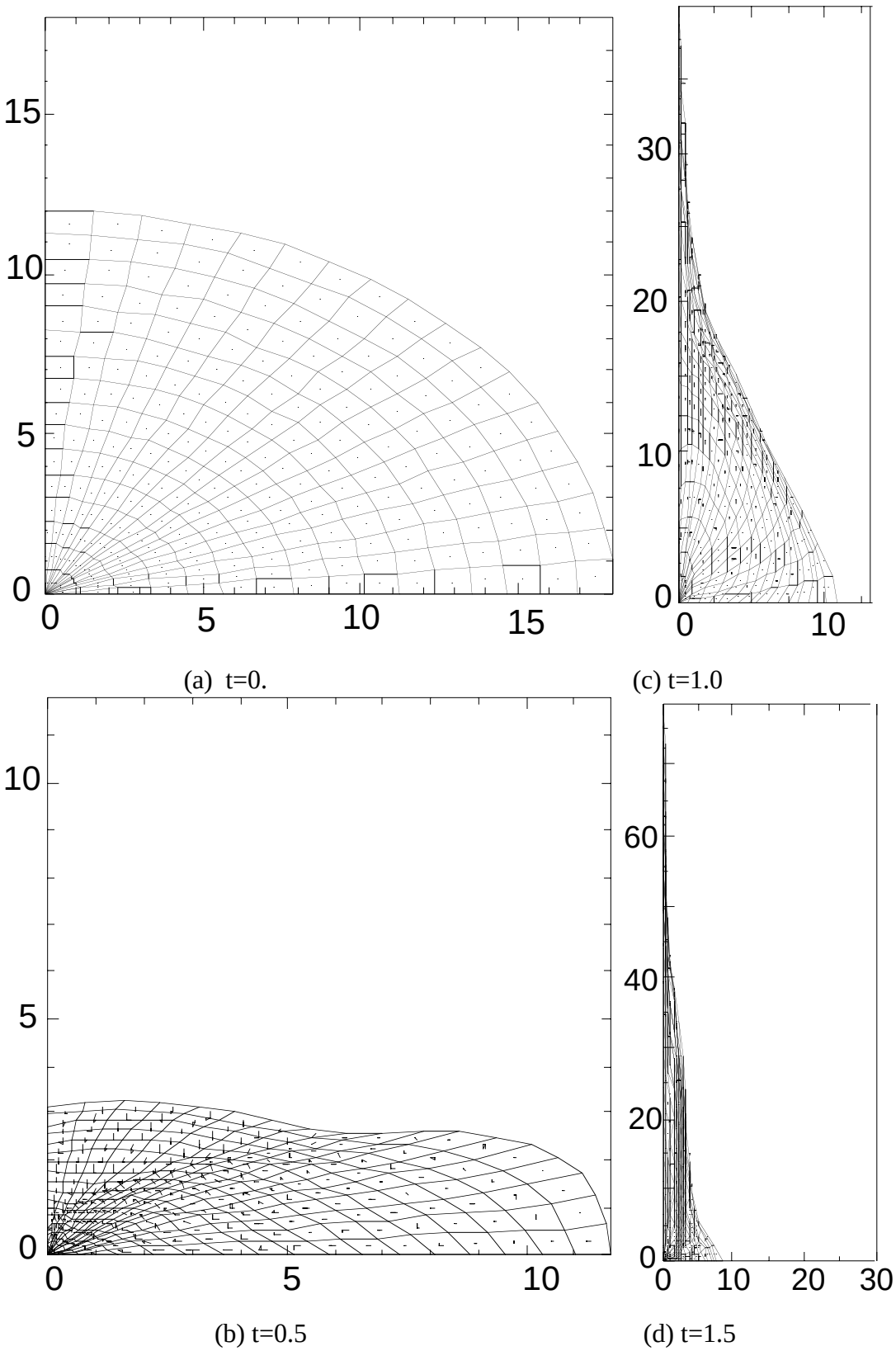
```
*i,bndpres.39
      pb = pbc(1,ib,iblk)
      if(t.lt.0.05) pb = 20.*t*pb

      do 30 i=i1bc(ib,iblk),i2bc(ib,iblk),i3bc(ib,iblk)
      pr(i) = pb
 30 continue

*d,meshgen.78
      xv(i,j,1) = x2*sin(a1*x1)*1.5
```

* This update inserts coding for the time-dependent applied pressure boundary in subroutine BNDPRES and modifies the mesh generation in MESHGEN to produce the ellipsoidal shape.

Table 6.5-2

Input file for ellipse problem.

```
ellipse test problem
 $input
igeom=4,
ncell(1,1)=18, ncell(1,2)=16,
dx(1,1)=5.,    dx   (1,2)=0.75,
rho0=1.845, pr0=1.e-10,
proprty(1,1)=8., 1.845, 0., 1.13, 2020.,

ibc(2)=2, pbc(1,2)=1000.,

iorder=2, limgrad=2,
alecoef=0.99, nadvskp=10, antidif=1.,
itrezn=1, nsmooth=0, wtampl=8.0,
ibcrezn=0,1,1,1, crvwt=2.5,
dtfac=0.9, dt0=1.e-4,
twfin=1.5, twfilm=0.0, 1.5, 0.5,   $end
```



(a) t=0.

(c) t=1.0

(b) t=0.5

(d) t=1.5

Fig. 6.5-1 Time history of ellipse test problem.

SECTION 7

DISCUSSION OF LIMITATIONS

This section highlights the aspects of the approach documented in this report that are considered to be a possible source of problems. These include inherent short-comings that are a direct consequence of the numerical choices, such as the determination of the vertex velocities in a cell-centered code, to limitations that are common to many numerical methods, such as the difficulty of mesh generation of complex geometries. If solutions to these limitations are known or if research is being undertaken, then a brief discussion is included.

§7.1 ADDITION OF MATERIAL MODELS

The choice of the Godunov method has the advantage that it naturally handles material discontinuities and minimizes the required numerical diffusion for treating shock discontinuities (§3.2.1a). To have these desirable characteristics, the solution of the velocity and stress at a cell face requires a sufficiently accurate physical description of the dynamics of the cell face. For pure hydrodynamic problems, the approximate Riemann solution accurately and efficiently accomplishes this [Dukowicz, 1985].

With the addition of a new material model, the best temporal accuracy results if the Riemann solution is modified in order to account for the effect of the additional physics in the surface integrals (Eqs. 3.1.1-1). Although desirable, including the addition physics into the Riemann solution can become complex and usually cannot be accomplished efficiently without some approximate treatment. Hence, not only does a new material model need to be added to the routines that advance the cell-centered variables, but the Riemann solution must be modified in a manner that is accurate *and* efficient. Competitive numerical methods, such as the predictor-corrector method [Johnson et al., 1990] or the finite element method, do not require this additional step in the implementation. But in fairness, these methods can suffer other difficulties that are not encountered in the numerical choices made in CAVEAT.

One such example of this approach is the addition of an elastic-plastic material model to CAVEAT [Johnson et al., 1990]. Here, the Riemann solution is modified by the addition of a perfectly elastic contribution.

If some compromise in temporal accuracy is acceptable, then for some material models the Riemann solution can be left unmodified and the contribution to the surface integrals by the material

model can be determined separately from the hydrodynamics. One example of this approach is the addition of the high explosive material model. At the end of the Lagrangian phase, the energy increase due to the burning of the high explosive is simply added to the cell-centered internal energy. In the next Lagrangian phase, this additional energy, and hence increased pressure and sound speed, is then accounted for in the Riemann solution by the change in the cell-centered quantities. Therefore, a temporal delay occurs in the effect of the high explosive on the hydrodynamics. This approach for high explosives has been successful and is believed not to compromise the performance of the material model. But in some Godunov codes, other researchers have chosen to modify the Riemann solution by including a source term corresponding to the release of high explosive energy [Bourgeade, 1988].

A related issue is the need to modify the boundary conditions in CAVEAT when adding a material model. Because the boundary conditions are based on a simplification of the Riemann solution (§3.2.4a), a similar difficulty is encountered as described above: how to include the additional physics into the boundary conditions. Again the two approaches, discussed above, also apply here. Typically whatever treatment is chosen for the interior of the mesh is also applied to the boundary conditions. For the elastic-plastic implementation, the boundary conditions are modified to include an elastic component. But for the high explosive implementation, the boundary conditions are unmodified.

§7.2 LAGRANGIAN VERTEX VELOCITIES

As mentioned in the introduction, the advantages of having all state variables stored in the cell center are offset by the loss of corner coupling (a second-order effect in time) between cells. This is a consequence of using Lagrangian face velocities in the surface integrals over the cell sides, instead of using vertex velocities. Consequently, the Lagrangian vertex velocities that are derived from the Riemann velocities (§3.2.1b) do not necessarily reproduce the volume change as calculated from the face velocities. Formulations exist which include this corner coupling within the Godunov method, but the ability to describe problems with interfacial slip is partially sacrificed. Hence the current formulation used in CAVEAT is considered a compromise between temporal accuracy and the ability to minimize numerical diffusion in the presence of interfacial slip.

Another difficulty arises in the current calculation of the Lagrangian vertex velocities. Ideally the propagation of a shock obliquely through a mesh (i.e., the shock is not aligned with either mesh direction) should not introduce Lagrangian motion of the mesh normal to the direction of propagation of the shock; i.e., distortions in the mesh should not introduce additional vorticity into the flow. The current method for the calculation of the Lagrangian vertex velocities *does not* have this ideal behavior, and as a consequence, certain types of flows in skewed meshes can result in artificial secondary flows normal to the propagation of the shock or, alternatively, can result in an artificial generation of vorticity. The magnitude of this error in CAVEAT is problem dependent and is usually not observable when the ALE formulation is used.

This error is common to most numerical formulations and has often been accepted as a unavoidable error. But research [Dukowicz and Meltz, 1990] is being completed within the context of the CAVEAT that minimizes this error, if not entirely eliminates it. The method involves two aspects: (1) an improved Riemann solution that includes the relative orientation of the cell face and the orientation of the shock and (2) the replacement of the current, localized method for the calculation of the Lagrangian vertex velocities with a global (meshwide) solution of the Lagrangian vertex velocities that satisfies the boundary conditions and does not introduce any new vorticity into the problem. The shortcoming of this method is that the global solution of the vertex velocities is computationally intensive and is not a realistic option for three dimensions. This alternative method for the calculation of the Lagrangian vertex velocities, while it may not be applicable to all problems, does offer a reference by which more simple methods can be evaluated.

§7.3 MESH GENERATION FOR COMPLEX PROBLEMS

As mentioned in the introduction, the choice of using blocks of logically rectangular mesh results in code simplicity and optimal vectorization. But the choice also complicates the generation of initial mesh for complex problems. Although multiple blocks extends the flexibility of a logically rectangular mesh, the restriction that mesh lines must be continuous through the blocks is a severe restriction for problems with large variations in characteristic length. This can result in large demands on the capabilities of a code for mesh initialization, especially in three dimensions. An experience user can often use the flexibility of the present version of CAVEAT to create a usable initial mesh for complex problems.

Many of these difficulties can be avoided by using the mixed cell option and slide line option [Johnson et al., 1990]. Furthermore, possible extensions to CAVEAT, such as the non-logical connectivity of blocks [Johnson et al., 1990], also can relax the restrictions of the logically rectangular blocks of mesh.

An additional complication that occurs in the setup of complex meshes is a result of the incompatibility of the initial mesh and the mesh that results from the global rezoner. Ideally when the simulation of a problem begins, the initial mesh should be close to mesh that is desired by the global rezoner. Any difference between these to meshes will result in large mesh movement in the beginning of the problem. While redefining the mesh by the global rezoner does not lead to any disastrous results, the large amount of mesh movement can result in advective diffusion that may smear features of interest, such as a density discontinuity. This problem can be circumvented by relaxing the initial mesh using the global rezoner as used in CAVEAT with the parameters that will be used in the simulation; this is an option in the initialization.

§7.4 HIGHER-ORDER METHODS AND CONSERVATION EQUATIONS

This section addresses a difficulty that is common to many codes but appears not to be widely appreciated. It occurs when multiple conservation equations are solved in parallel in a simulation code and then derived variables are defined and used that are combinations of the conserved quantities. Even though the conserved variables may be identically conserved in the simulation, the derived variables can have almost arbitrary values under some circumstances.

An example of this difficulty occurs with the internal energy in CAVEAT during the remap phase. When the second-order advection is used, the conserved quantities (mass, momentum, and total energy) in the fluxing volume are determined from limited gradients across the cell (§3.1.2). Because the degree limiting on each of these conserved quantities can differ largely, the value of the internal energy in the fluxing volume, as derived from the kinetic energy and total energy, can have extrema that do not occur within the cell. Under some circumstances, this can lead to anomalous internal energies at the end of the remap phase and, hence, anomalous pressures. It can easily be shown that this is not a difficulty when using the first-order method.

A possible solution of this difficulty, and one that is commonly chosen for other reasons as discussed at the end of this section, is to forfeit the conservation of total energy and advect internal energy. Then the internal energy will not contain new extrema after advection because of the requirements imposed by the limiting method. The derived quantity, the total energy, in this approach is not typically used in later calculations, and therefore variations in it do not cause difficulties. Unfortunately, a related problem arises in that the pressure, now derived from the density and internal energy through the equation of state, can have new extrema. A specific solution to this difficulty has been tested successfully by McGlaun [McGlaun and Thompson, 1989] which couples the fluxed quantities of density and internal energy in the fluxing volume by requiring that these fall on the line connecting the two states in density-internal energy space of the two cells across the fluxing volume.

The difficulty discussed above is a problem that is expected to appear in any code that solves multiple conservation equations. A general solution of this difficulty is under investigation and will be of interest beyond the current application of hydrodynamics codes.

Another difficulty that arises in codes based on total energy is commonly recognized: the potential of unrealistic heating of materials during advection. This occurs because of the drop of kinetic energy due to mixing of gradients in the velocity field during advection that is not reflected in the total energy. Hence the calculation of the internal energy from the difference of the total energy and kinetic energy results in a higher internal energy. A common solution to this problem is to solve an internal energy equation and forfeit the conservation of total energy. The difficulty first discussed in this section has not been commonly observed largely because simulation codes are often based on internal energy. For some applications, a version of CAVEAT that is internal energy based may be necessary.

REFERENCES

J. Abdallah, Jr., et al., "HYDSES: A Subroutine Package for Using SESAME in Hydrodynamic Codes," Los Alamos Scientific Laboratory, LA-8209 (1980).

A. Bourgeade, "Second-Order Scheme for Reacting Flows: Application to Detonation," in the *Proceedings of the Conference on Numerical Methods in High Temperature Physics*, Compiled by R. E. Alcouffe, D. D. Holme, and P. J. O'Rourke, Los Alamos National Laboratory, LA-11342-C (1988).

D. J. Cagliostro, R. H. Warnes, N. L. Johnson and R. K. Fujita, "Spall Measurements in Shock Loaded Hemispherical Shells from Free-Surface Velocity Histories," *Shock Waves in Condensed Matter 1987*, Editors: S. C. Schmidt, N. C. Holmes, 367-370.

C. W. Cranfill, "EOSPAC: A Subroutine Package for Accessing the Los Alamos Sesame EOS Data Library", Los Alamos National Laboratory, LA-9728-M (1983).

J. K. Dukowicz and J. D. Ramshaw, "Tensor Viscosity Method for Convection in Numerical Fluid Dynamics," *J. Comput. Phys.*, **32**, 71-79 (1979).

J. K. Dukowicz, "Conservative Rezoning (Remapping) for General Quadrilateral Meshes," *J. Comput. Phys.*, **54**, 411-424 (1984).

J. K. Dukowicz, "A General, Non-Iterative Riemann Solver for Godunov's Method," *J. Comput. Phys.*, **61**, 119-137 (1985).

J. K. Dukowicz and J. Kodis, "Accurate Conservative Remapping (Rezoning) for Arbitrary Lagrangian-Eulerian Computations," *SIAM J. Sci. Stat. Comput.*, **8**, 305-321 (1987).

J. K. Dukowicz and B. J. A. Meltz, "Vorticity Errors in Multidimensional Lagrangian Codes," To be published in *J. Comput. Phys*. in 1991.

J. M. McGlaun and S. L. Thompson, "CTH: A Three-Dimensional Shock Wave Physics Code," *Hyper-Velocity Impact Symposium 1989*, proceedings to be published.

J. B. Goodman and R. J. LeVeque, Math. Comput., **8**, 15 (1985).

K. S. Holian, Ed., *T-4 Handbook of Material Properties Data Bases, Vol. 1c: Equations of State*, Los Alamos National Laboratory, LA-10160-MS (1984),

K. S. Holian and B. L. Holian, "Hydrodynamic Simulations of Hypervelocity Impact'" *Int. J. Impact Engng.* (1989, to be published).

N. L. Johnson, F. L. Addessio, J. R. Baumgardner, B. A. Kashiwa, and R. M. Rauenzahn, "CAVEAT: Current Extensions and Future Capabilities," Los Alamos National Laboratory report, to be published in 1991.

G. I. Kerley, "Rational Function Method for Interpolation," Los Alamos National Laboratory, LA-6903-MS (1977).

B. van Leer, "Towards the Ultimate Conservative Difference Scheme," *J Comput. Phys.*, **32**, 101-136 (1979).

S. P. Marsh, *LASL Shock Hugoniot Data*, Univ. of California Press, Berkeley/Los Alamos (1980),

J. D. Ramshaw and J. K. Dukowicz, "APACHE: A Generalized-Mesh Eulerian Computer Code for Multicomponent Chemically Reactive Fluid Flow", Los Alamos Scientific Laboratory report LA-7427 (1979).

D. L. Sandoval, *CAVEAT Calculations of Shock Interactions,* LA-11001-MS (1987).

L. I. Sedov, *Similarity and Dimensional Methods in Mechanics,* Academic Press, New York 1959.

W. L. Slattery and W. H. Spangenberg, "A Fast Algorithm for Two-Dimensional Data Table Use in Hydrodynamic and Radiation Transfer Codes," Los Alamos National Laboratory, LA-UR-82-656 (1982).

J. F. Thompson, F. C. Thames, and C. W. Mastin, "??Title??", *J. Comput. Phys.*, **15**, 299 (1974).

J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin, *Numerical Grid Generation*, North-Holland, NY (1985).

P. A. Thompson, *Compressible-Fluid Dynamics*, McGraw-Hill, New York (1982).

A. M. Winslow, *Adaptive Mesh Zoning by the Equipotential Method*, UCID-19062, Lawrence Livermore National Laboratory (1981).

APPENDIX A

VARIABLES AND SUBROUTINES

A.1 INPUT AND OUTPUT FILES

IN2D	Input file that controls the calculation initiation, simulation parameters, and output.
OUT2D	Output file of input variables, history of calculation, and timing information (§4.5b).
RS2D	Restart file for continuing a calculation for a dump file, DP2D (§4.5c).
DP2D	Dump file produced during a calculation that can be used for restarting a calculation (§4.5c).
GASDUMP	Graphics output file that contains the state of the calculation at prescribed times and can be examined with the GAS utility (§4.5e).
PLOT	Graphics output file that contains one and two dimensional plots of the state of the calculation at prescribed times and can be examined with the PSCAN utility (§4.5d).

A.2 INPUT VARIABLES

All the input variables, in alphabetical order, are listed that are used in CAVEAT, along with brief definitions. Note that this information is also contained in the listing of CAVEAT in the subroutine GLOSSARY. The meaning of the arguments for the variables, if required, are:

I, J, K	index to cell, face, or vertex values; also for general reference to an array value
IB	index to boundary conditions
IBLK	index to blocks
IMAT	index to material number (1-30)
IP	index to a part in a block, referenced linearly
IPRT	index to part in a block along the 1-direction, referenced by pairs of numbers (IPRT, JPRT)
JPRT	index to part in a block along the 2-direction, referenced by pairs of numbers
KPRT	taken to be IPRT if M=1, JPRT if M=2
M	mesh direction (1 or 2) for vectors

Input parameters with the symbol '@' following the variable have no default parameter. Following the definition in parentheses, the common block that contains the variable, the default value, and the section in this report that discusses the variable are given. Note that the Fortran parameter variables NB, NP, and NS are also listed even though they are not input variables but can be changed by the user at compilation.

ALECOEF	Parameter specifying the extent of remapping. A value of zero implies calculation is in the Eulerian limit, while a value of one implies the Lagrangian limit. Intermediate values result in both rezoning of the mesh and fluxing across cell faces. (CNTRL, 0.9; §4.3.2h, §4.4.6, §5.3)
ANTIDIF	Factor for controlling the amount of 'anti-diffusion' applied in the second order Godunov treatment. A value of zero (default) applies no anti-diffusion, while a value of one applies the maximum allowable amount. A value of one typically provides sharper resolution of shocks with no discernable negative consequences. In some situations, however, it is marginally unstable and leads to obvious spurious pressure oscillations. (CNTRL, 0.0; §4.3.1b)
BNDCOEF	Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on boundaries in routine REZTANG. (REZN1, 0.5; §4.3.2h)
CONVD	Conversion factor for density units from SESAME tables. (EOS02, 1.0 implies units of g/cm^3; §4.4.4)
CONVE	Conversion factor for energy units from SESAME tables. (EOS02, 0.01 implies units of $Mbar\text{-}cm^3/g$; §4.4.4)
CONVP	Conversion factor for pressure units from SESAME tables. (EOS02, 0.01 implies units of Mbar; §4.4.4)
CRVWT	Parameter for weighting the effect of curvature in the tangential rezone of block boundaries. Increasing the value of CRVWT produces finer zoning in blocks of high curvature. (REZN1, 1; §4.3.2h)
CUSHION	Time increment in seconds before job time limit is reached that soft abort of job is initiated. (TIME1, 0; §4.4.6)
DT0@	Initial time step for the hydro cycle. (TIME2, no default; §4.4.6, §5.2)
DTFAC	Fraction of the Courant time step limit allowed for hydro time step. (TIME2, 0.5; §4.4.6, §5)
DTMAX	User specified maximum for hydro time step. (TIME2, 1000 DT0; §4.4.6, §5.2)
DTMIN	User specified minimum for hydro time step. (TIME2, 0.0001 DT0; §4.4.6, §4.5f, §5.2)
DX(KPRT,M,IBLK)@	Initial mesh spacing in each part, mesh direction, and block. (MESH2, no default; §4.4.1)
FDONOR	Extent of donor-cell treatment in the advection calculation. A value of one results in full donor-cell treatment. (CNTRL, 1.0; §4.3.3b)
GASSCL(30)	Multiplicative scale factors for GAS variables. (GRAF1, 1.0; §4.5e)
I2DAXIS	Flag indicating whether or not coordinate axes with tic marks are to be drawn on 2-D plots. I2DAXIS= 0 implies no; I2DAXIS= 1 implies yes. (GRAF1, 1; §4.5d)
I2DPLOT	Flag indicating whether or not 2-D plots of pressure, density, internal energy, and velocity are desired. I2DPLOT = 0 implies no; I2DPLOT = 1 implies yes. (GRAF1, 0; §4.5d)
IBC(IB,IBLK)	Indices specifying the boundary types for the four boundaries of block IBLK as follows: IBC = 1 (non-curved) reflective boundary IBC = 2 applied pressure IBC = 3 outflow IBC = 4 inflow

IBC = 5 boundary between blocks--use ghost cells for inter-block communication
IBC = 8 curved, undeformable free-slip boundary
IBC = 9 specified velocity (piston) boundary

The order for the four boundaries is bottom, top, left, and right. (BNDR1, 1; §4.4.5)

IBCREZN(IB,IBLK) Parameter specifying application of a special tangential rezone treatment on boundary IB of block IBLK. When set to one this parameter provides uniform spacing of vertices along straight boundaries and concentrated spacing in blocks of high curvature. This treatment is appropriate, for example, when jetting is expected. (REZN1, 0; §4.3.2h)

IFLUX Flag to specify method for computing fluxing volumes in routine ADVFLUX. A value of one specifies a method based of vertex velocities. Any other value specifies a method based on the face velocities. (CNTRL, 1.0; §4.3.3)

IGAS Flag indicating whether or not GAS dumps are desired. (GRAF1, 0; §4.5e)

IGEOM Parameter specifying the geometry type (MESH3, 1; §4.4.1):

IGEOM = 1 generates a 'x-y' rectangular mesh
IGEOM = 2 generates an 'r-z' cylindrical mesh
IGEOM = 3 generates a polar 'x-y' cylindrical mesh
IGEOM = 4 generates an 'r-z' spherical mesh
IGEOM = 5 generates a polar 'x-y' cylindrical mesh composed of four mesh blocks
IGEOM = 6 generates an 'r-z' spherical mesh composed of four mesh blocks

ILINE(M,10) Array specifying lines along which graphical output is desired. Elements of first index specify respectively the mesh direction of the line and the indices identifying the line in the other mesh direction. (GRAF1, -1; §4.5d)

INITRZN Number of rezone iterations to be applied to the mesh during problem initialization. If greater than zero, this parameter allows mesh to relax toward the solution of the Winslow generator equations prior to starting the hydrodynamic calculations. (REZN1, 0; §4.3.2h, §4.4.1)

IORDER Flag to indicate first or second order spatial treatment in the advection and Riemann routines. IORDER=1 implies first order, while IORDER=2 implies second order. (CNTRL, 1; §4.3.1b, §4.3.3b, §5.3)

IRADIAL Parameter that controls the orientation of the cylindrical axis. For IRADIAL = 1, the cylindrical axis coincides with coordinate direction two and the radial direction corresponds with coordinate direction one and the I1=1 mesh line is the cylindrical axis. When IRADIAL = 2, the radial direction corresponds to coordinate direction two and the I2=1 mesh line is the cylindrical axis. When IRADIAL = 0, there is no cylindrical axis in the plane of the mesh. Selecting IGEOM = 1, 3, or 5 automatically sets IRADIAL = 0. (MESH3, 1; §4.4.1)

ISESOPT Parameter specifying interpolation method for SESAME tables. Choices are: biline for 4-point bilinear; biquad for 6-point biquadratic; biratf for 12-point bi-rational-function. (EOS02, 6hBIRATF; §4.4.4)

ITREZN Number of iterations for Winslow rezone procedure. (REZN1, 3; §4.3.2h)

LIMGRAD Flag to indicate type of limiting used in second-order treatment. Set LIMGRAD=1 for monotonic limiting, LIMGRAD=2 for van Leer limiting, and LIMGRAD=0 for for van Leer limiting on all variables except velocity. (CNTRL, 2; §4.3.1b, §5.3)

LOGO — Flag specifying whether LANL logo is to be included on on plots. LOGO = 0 implies no; LOGO = 1 implies yes. (GRAF2, 0; §4.5d)

MATBC(KPRT,IB,IBLK) — Material numbers outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1; §4.4.5)

MATNUM(IP,IBLK) — Material number by part and block. This number ranges between 1 and 30 and identifies which material of those specified by array proprty resides in a given part and block. (EOS01, 1; §4.4.2)

N1(IBLK) — Number of cells in mesh direction one in block IBLK. (MESH1, 1; §4.4.1)

N2(IBLK) — Number of cells in mesh direction two in block IBLK. (MESH1, 1; §4.4.1)

NADVSKP — Number of Lagrangian steps for every advection step in ALE hydrodynamic calculation. This parameter must be four or greater for any advection steps to be skipped. (CNTRL, 1; §4.3.1b, §5.3b)

NB — Parameter for maximum number of blocks. (parameter variable, 4; §4.4.1, §4.1.4, §5.3)

NBC(2,IB,IBLK) — Input array specifying the block communication setup. NBC(1,IB,IBLK) is set equal to the block with which side IB communicates. NBC(2,IB,IBLK) is set equal to the side of block NBC(1,IB,IBLK) with which side IB communicates. (BCOM1, 0; §4.1.4)

NBLKS — Number of logically rectangular blocks comprising the problem domain. Complex geometries can be generated by joining blocks of logically rectangular mesh which communicate across their common boundaries. (MESH1, 1; §4.4.1)

NCELL(KPRT,M,IBLK)@ — Number of cells associated with each part, mesh direction, and block. Dimensions allow up to NS different parts in each mesh direction. (MESH2, no default; §4.4.1)

NCYCMAX — Problem hydro cycle limit. (TIME1, 10000; §4.4.6)

NP — Parameter for maximum number of part in any block. (parameter variable, 40; §4.4.1, §4.5f, §5.3)

NPRTS(M,IBLK) — Number of parts or subdivisions in each mesh direction within block IBLK. This feature allows for multiple initial states, materials and/or initial mesh spacings to be specified within a single block. (MESH2, 1; §4.4.1)

NRESTART — Dump number on file RS2D for problem restart. (TIME1, -1; §4.5c)

NS — Parameter for maximum number of parts divisions along a mesh direction. (parameter variable, 8; §4.4.1, §4.5f, §5.3)

NSMOOTH — Number of smoothing iterations in the calculation of the rezone weight function. (REZN2, 0; §4.3.2h)

NWK(I) — Size of additional work arrays as needed for development of new material models or numerical methods. See §4.1.3d for a detailed description and use. (pointer variable, 0; §4.1.3d)

PBC(KPRT,IB,IBLK) — Pressures outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)

PR0(IP,IBLK)@ — Initial pressure associated with each part and block. (INIT1, no default; §4.4.1)

PROPRTY(100,30) — Material property array.location (EOS01, see §4.4.3 for defaults)

1: EOS type (Values 1-8 imply gamma-law, linear, quadratic, blank, HOM, blank, blank, and SESAME, respectively.)

2: Reference density for EOS

3: Pressure floor

4: Riemann strong shock parameter
5-100: EOS model parameters

RHO0(IP,IBLK)@ Initial density associated with each part and block. (INIT1, no default; §4.4.1)

RHOBC(KPRT,IB,IBLK) Material densities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)

SCLARRW Scale factor for arrows in 2-D velocity plots. (GRAF2, 1; §4.5d)

TANREZN Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on interfaces. (REZN1, 0.02; §4.3.2h)

TITL(5) Hollerith string containing problem title. This is the first line of the input file and is reproduced in the graphics (GRAF1, blank)

TPULSE Width of the applied pressure pulse for cylindrical phased implosion problems. Not implemented. (PRES1, 0.0)

TWDUMP(10) Same as twprnt, except restart dump on file DP2D. (TIME1, 1.E50; §4.5c)

TWFILM(10) Same as twprnt, except for graphics output. (TIME1, 1.E50; §4.5d)

TWFIN @ Problem time limit. (TIME1, no default; §4.4.6)

TWMOVI(10) Same as twprnt, except for movie output. (TIME1, 1.E50)

TWPRNT(10) Times at which printer output is written to TAPE6. If TWPRNT(3) is less than TWPRNT(2), TWPRNT(3) is interpreted as the time increment between output times. (TIME1, 1.E50; §4.5b)

UBC(M,KPRT,IB,IBLK) Material velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0); §4.4.5)

UC0(M,IP,IBLK) Initial Cartesian velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)

UR0(IP,IBLK) Initial radial velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)

URBC(KPRT,IB,IBLK) Radial velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0,); §4.4.5)

VANLEER Factor for reducing the magnitude of the van Leer limited gradients to maintain stability. (CNTRL, 1.0)

VPHASE Phase velocity in the y-direction for cylindrical phased implosion problems. Not implemented. (PRES1, 0.0)

WFMT(30) Weighting of individual materials. (REZN2, 0.0; §4.3.2h)

WFGR Weighting of spatial gradients of various scalar fields: zero implies none; positive values specify the pressure; negative the density field. (REZN2, 0.0; §4.3.2h)

WFLM Limiting value for the gradient, above which the gradient contribution saturates. Small values allow regions with smaller gradients to find expression in the weighting. (REZN2, 1.e100; §4.3.2h)

WFVO Weighting of equal volume. (REZN2, 1.0; §4.3.2h)

WTAMPL Ratio of maximum to minimum values for the rezone weight function. The default value of one gives a constant function corresponding to uniform volume weighting. Values greater than one should be used when a polar geometry is used (recommended range is 5.0-10.0) or if any of the other weight function options are exercised. (REZN2, 1.0; §4.3.2h)

X0(M,IBLK) Origin offset relative to the physical origin for block IBLK. (MESH2, (0.,0.); §4.4.1)

A.3 INPUT VARIABLES ORGANIZED BY FUNCTION

In this section the input variables are listed are listed by their function, along with brief definitions. Note that this information is also contained in the listing of CAVEAT in the subroutine GLOSSARY. The meaning of the arguments for the variables, if required, are given in the Notes on Nomenclature section. Input parameters with the symbol '@' following the variable have no default parameter. Following the definition, the common block that contains the variable, the default value if the variable is an input parameter, and the section in this report that discusses the variable are given in parentheses.

a. Boundary Conditions and Interface Treatment

BNDCOEF — Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on boundaries in routine REZBND. (REZN1, 0.5; §4.3.2h)

CRVWT — Parameter for weighting the effect of curvature in the tangential rezone of block boundaries. Increasing the value of CRVWT produces finer zoning in blocks of high curvature. (REZN1, 1; §4.3.2h)

IBC(IB,IBLK) — Indices specifying the boundary types for the four boundaries of block IBLK as follows:

- IBC = 1 (non-curved) reflective boundary
- IBC = 2 applied pressure
- IBC = 3 outflow
- IBC = 4 inflow
- IBC = 5 boundary between blocks--use ghost cells for inter-block communication
- IBC = 8 curved, undeformable free-slip boundary
- IBC = 9 specified velocity (piston) boundary

The order for the four boundaries is bottom, top, left, and right. (BNDR1, 1; §4.4.5)

IBCREZN(IB,IBLK) — Parameter specifying application of a special tangential rezone treatment on boundary IB of block IBLK. When set to one this parameter provides uniform spacing of vertices along straight boundaries and concentrated spacing in blocks of high curvature. This treatment is appropriate, for example, when jetting is expected. (REZN1, 0; §4.3.2h)

ITREZN — Number of iterations for Winslow rezone procedure. (REZN1, 3; §4.3.2h)

MATBC(KPRT,IB,IBLK) — Material numbers outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1; §4.4.5)

NBC(2,IB,IBLK) — Input array specifying the block communication setup. NBC(1,IB,IBLK) is set equal to the block with which side IB communicates. NBC(2,IB,IBLK) is set equal to the side of block NBC(1,IB,IBLK) with which side IB communicates. (BCOM1, 0; §4.1.4)

PBC(KPRT,IB,IBLK) — Pressures outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)

RHOBC(KPRT,IB,IBLK) Material densities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)

TANREZN Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on interfaces. (REZN1, 0.02; §4.3.2h)

UBC(M,KPRT,IB,IBLK) Material velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0); §4.4.5)

URBC(KPRT,IB,IBLK) Radial velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0,); §4.4.5)

b. Material and Mesh Intialization

CONVD Conversion factor for density units from SESAME tables. (EOS02, 1.0 implies units of g/cm^3; §4.4.4)

CONVE Conversion factor for energy units from SESAME tables. (EOS02, 0.01 implies units of Mbar-cm^3/g; §4.4.4)

CONVP Conversion factor for pressure units from SESAME tables. (EOS02, 0.01 implies units of Mbar; §4.4.4)

DX(KPRT,M,IBLK)@ Initial mesh spacing in each part, mesh direction, and block. (MESH2, no default; §4.4.1)

IGEOM Parameter specifying the geometry type (MESH3, 1; §4.4.1):
IGEOM = 1 generates a 'x-y' rectangular mesh
IGEOM = 2 generates an 'r-z' cylindrical mesh
IGEOM = 3 generates a polar 'x-y' cylindrical mesh
IGEOM = 4 generates an 'r-z' spherical mesh
IGEOM = 5 generates a polar 'x-y' cylindrical mesh composed of four mesh blocks
IGEOM = 6 generates an 'r-z' spherical mesh composed of four mesh blocks

INITRZN Number of rezone iterations to be applied to the mesh during problem initialization. If greater than zero, this parameter allows mesh to relax toward the solution of the Winslow generator equations prior to starting the hydrodynamic calculations. (REZN1, 0; §4.3.2h, §4.4.1)

IRADIAL Parameter that controls the orientation of the cylindrical axis. For IRADIAL = 1, the cylindrical axis coincides with coordinate direction two and the radial direction corresponds with coordinate direction one and the I1=1 mesh line is the cylindrical axis. When IRADIAL = 2, the radial direction corresponds to coordinate direction two and the I2=1 mesh line is the cylindrical axis. When IRADIAL = 0, there is no cylindrical axis in the plane of the mesh. Selecting IGEOM = 1, 3, or 5 automatically sets IRADIAL = 0. (MESH3, 1; §4.4.1)

ISESOPT Parameter specifying interpolation method for SESAME tables. Choices are: 6hBILINE for 4-point bilinear; 6hBIQUAD for 6-point biquadratic; 6hBIRATF for 12-point bi-rational-function. (EOS02, 6hBIRATF; §4.4.4)

MATNUM(IP,IBLK) Material number by part and block. This number ranges between 1 and 30 and identifies which material of those specified by array proprty resides in a given part and block. (EOS01, 1; §4.4.2)

N1(IBLK) Number of cells in mesh direction one in block IBLK. (MESH1, 1; §4.4.1)

N2(IBLK) Number of cells in mesh direction two in block IBLK. (MESH1, 1; §4.4.1)

NBC(2,IB,IBLK) Input array specifying the block communication setup. NBC(1,IB,IBLK) is set equal to the block with which side IB communicates. NBC(2,IB,IBLK) is set equal to the side of block NBC(1,IB,IBLK) with which side IB communicates. (BCOM1, 0; §4.1.4)

NBLKS Number of logically rectangular blocks comprising the problem domain. Complex geometries can be generated by joining blocks of logically rectangular mesh which communicate across their common boundaries. (MESH1, 1; §4.4.1)

NCELL(KPRT,M,IBLK)@ Number of cells associated with each part, mesh direction, and block. Dimensions allow up to NS different parts in each mesh direction. (MESH2, no default; §4.4.1)

NCYCMAX Problem hydro cycle limit. (TIME1, 10000; §4.4.6)

NPRTS(M,IBLK) Number of parts or subdivisions in each mesh direction within block IBLK. This feature allows for multiple initial states, materials and/or initial mesh spacings to be specified within a single block. (MESH2, 1; §4.4.1)

PR0(IP,IBLK)@ Initial pressure associated with each part and block. (INIT1, no default; §4.4.1)

PROPRTY(100,30) Material property array.location (EOS01, see §4.4.3 for defaults)
1: EOS type (Values 1-8 imply gamma-law, linear, quadratic, blank, HOM, blank, blank, and SESAME, respectively.)
2: Reference density for EOS
3: Pressure floor
4: Riemann strong shock parameter
5-100: EOS model parameters

RHO0(IP,IBLK)@ Initial density associated with each part and block. (INIT1, no default; §4.4.1)

UC0(M,IP,IBLK) Initial Cartesian velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)

UR0(IP,IBLK) Initial radial velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)

X0(M,IBLK) Origin offset relative to the physical origin for block IBLK. (MESH2, (0.,0.); §4.4.1)

c. Numerical Options and Control Parameters

ALECOEF Parameter specifying the extent of remapping. A value of zero implies calculation is in the Eulerian limit, while a value of one implies the Lagrangian limit. Intermediate values result in both rezoning of the mesh and fluxing across cell faces. (CNTRL, 0.9; §4.3.2h, §4.4.6, §5.3)

ANTIDIF Factor for controlling the amount of ‘anti-diffusion’ applied in the second order Godunov treatment. A value of zero (default) applies no anti-diffusion, while a value of one applies the maximum allowable amount. A value of one typically provides sharper resolution of shocks with no discernable negative consequences. In some situations, howerever, it is marginally unstable and leads to obvious spurious pressure oscillations. (CNTRL, 0.0; §4.3.1b)

CUSHION Time increment in seconds before job time limit is reached that soft abort of job is initiated. (TIME1, 0; §4.4.6)

DT0 @ Initial time step for the hydro cycle. (TIME2, no default; §4.4.6, §5.2)

DTFAC Fraction of the Courant time step limit allowed for hydro time step. (TIME2, 0.5; §4.4.6, §5)

DTMAX	User specified maximum for hydro time step. (TIME2, 1000 DT0; §4.4.6, §5.2)
DTMIN	User specified minimum for hydro time step. (TIME2, 0.0001 DT0; §4.4.6, §4.5f, §5.2)
IFLUX	Flag to specify method for computing fluxing volumes in routine ADVFLUX. A value of one specifies a method based of vertex velocities. Any other value specifies a method based on the face velocities. (CNTRL, 1.0; §4.3.3)
IORDER	Flag to indicate first or second order spatial treatment in the advection and Riemann routines. IORDER=1 implies first order, while IORDER=2 implies second order. (CNTRL, 1; §4.3.1b, §4.3.3b, §5.3)
ISESOPT	Parameter specifying interpolation method for SESAME tables. Choices are: 6hBILINE for 4-point bilinear; 6hBIQUAD for 6-point biquadratic; 6hBIRATF for 12-point bi-rational-function. (EOS02, 6hBIRATF; §4.4.4)
ITREZN	Number of iterations for Winslow rezone procedure. (REZN1, 3; §4.3.2h)
LIMGRAD	Flag to indicate type of limiting used in second-order treatment. Set LIMGRAD=1 for monotonic limiting, LIMGRAD=2 for van Leer limiting, and LIMGRAD=0 for for van Leer limiting on all variables except velocity. (CNTRL, 2; §4.3.1b, §5.3)
NADVSKP	Number of Lagrangian steps for every advection step in ALE hydrodynamic calculation. This parameter must be four or greater for any advection steps to be skipped. (CNTRL, 1; §4.3.1b, §5.3b)
NCYCMAX	Problem hydro cycle limit. (TIME1, 10000; §4.4.6)
NRESTART	Dump number on file RS2D for problem restart. (TIME1, -1; §4.5c)
NSMOOTH	Number of smoothing iterations in the calculation of the rezone weight function. (REZN2, 0; §4.3.2h)
NWK(I)	Size of additional work arrays as needed for development of new material models or numerical methods. See §4.1.3d for a detailed description and use. (pointer variable, 0; §4.1.3d)
TANREZN	Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on interfaces. (REZN1, 0.02; §4.3.2h)
TWFIN	Problem time limit. (TIME1, no default; §4.4.6)
VANLEER	Factor for reducing the magnitude of the van Leer limited gradients to maintain stability. (CNTRL, 1.0)
WFMT(30)	Weighting of individual materials. (REZN2, 0.0; §4.3.2h)
WFGR	Weighting of spatial gradients of various scalar fields: zero implies none; positive values specify the pressure; negative the density field. (REZN2, 0.0; §4.3.2h)
WFLM	Limiting value for the gradient, above which the gradient contribution saturates. Small values allow regions with smaller gradients to find expression in the weighting. (REZN2, 1.e100; §4.3.2h)
WFVO	Weighting of equal volume. (REZN2, 1.0; §4.3.2h)
WTAMPL	Ratio of maximum to minimum values for the rezone weight function. The default value of one gives a constant function corresponding to uniform volume weighting. Values greater than one should be used when a polar geometry is used (recommended range is 5.0-10.0) or if any of the other weight function options are exercised. (REZN2, 1.0; §4.3.2h)

d. Output Options

CUSHION	Time increment in seconds before job time limit is reached that soft abort of job is initiated. (TIME1, 0; §4.4.6)
GASSCL(30)	Multiplicative scale factors for GAS variables. (GRAF1, 1.0; §4.5e)
I2DAXIS	Flag indicating whether or not coordinate axes with tic marks are to be drawn on 2-D plots. I2DAXIS= 0 implies no; I2DAXIS= 1 implies yes. (GRAF1, 1; §4.5d)
I2DPLOT	Flag indicating whether or not 2-D plots of pressure, density, internal energy, and velocity are desired. I2DPLOT = 0 implies no; I2DPLOT = 1 implies yes. (GRAF1, 0; §4.5d)
IGAS	Flag indicating whether or not GAS dumps are desired. (GRAF1, 0; §4.5e)
ILINE(M,10)	Array specifying lines along which graphical output is desired. Elements of first index specify respectively the mesh direction of the line and the indices identifying the line in the other mesh direction. (GRAF1, -1; §4.5d)
LOGO	Flag specifying whether LANL logo is to be included on on plots. LOGO = 0 implies no; LOGO = 1 implies yes. (GRAF2, 0; §4.5d)
SCLARRW	Scale factor for arrows in 2-D velocity plots. (GRAF2, 1; §4.5d)
TITL(5)	Hollerith string containing problem title. (GRAF1, blank)
TWDUMP(10)	Same as TWPRNT, except restart dump on file DP2D. (TIME1, 1.E50; §4.5c)
TWFILM(10)	Same as TWPRNT, except for graphics output. (TIME1, 1.E50; §4.5d)
TWMOVI(10)	Same as TWPRNT, except for movie output. (TIME1, 1.E50; not implemented)
TWPRNT(10)	Times at which printer output is written to OUT2D. If TWPRNT(3) is less than TWPRNT(2), TWPRNT(3) is interpreted as the time increment between output times. (TIME1, 1.E50; §4.5b)

A.3 SUBROUTINES

In this appendix, the purpose of each subroutine in CAVEAT is given. In the code listing the subroutines that call and are called by the subroutine are listed in the header for each subroutine.

ADVECT	This routine performs the advection calculation.
ADVFLUX	This routine computes fluxing volumes across cell faces.
ASSGN	Fill the array Q with the value Q0.
BCOMGRD	This routine performs inter-block communication for cell-centered gradient array GR of a variable with NCMP components.
BCOMHYD	This routine performs inter-block communication for cell-centered arrays UC, PR, RHO, SIE, TE, SS, RA, and IFLG.
BCOMQ	This routine performs inter-block communication for cell-centered array Q.
BCOMSET	This routine generates offsets and other quantities required for inter-block communication.
BCOMUF	This routine performs inter-block communication for face-centered array UF.
BCOMW	This routine performs inter-block communication for cell-centered array W.
BCOMXV	This routine performs the inter-block communication for the vertex array XV.
BNDFN	This routine computes the unit normal components for symmetry boundaries.

BNDGRAD This routine zeros the gradient in the ghost cells.

BNDHYD This routine loads the boundary elements in arrays UC, PR, RHO, SIE, TE, SS, RA, CM, and TEMP.

BNDPRES This routine loads the face pressures on applied pressure boundaries.

BNDREFL This routine loads the boundary elements in array W assuming reflective boundaries.

BNDRIEM This routine loads the boundary elements in arrays UF and PF for the case of a reflective, applied pressure, or inflow boundary.

BNDUM This routine sets to zero the component of the mesh velocity normal to a reflecting, inflow, or outflow boundary. It also sets both velocity components to zero on a boundary that is collapsed to a point (e.g., the origin in a spherical mesh).

BNDUMRZ This routine sets to zero the component of the mesh velocity normal to a reflecting boundary. On a fluxing boundary it reverses the sign of the normal component of um. It also computes a common rezone velocity for a boundary collapsed to a point to form the center of convergence for a polar mesh.

BNDXV This routine generates ghost vertex locations for block IBLK.

CELCNTR This routine computes the coordinates of the cell centers by averaging the coordinates of the four cell vertices.

CONTOUR This routine generates contour plots of variable W.

CRNR3UM This routine computes the Lagrangian vertex velocities for the vertices that lie at the junction of three blocks.

CRNR3XV This routine computes the coordinates for the vertices that lie at the junction of three blocks as the average of the coordinates provided from each of the three blocks.

CRNRHYD This routine loads the corner ghost cells of block iblk for cell-centered arrays UC, PR, RHO, SIE, TE, SS, and RA, if a given corner ghost cell is degenerate.

CRNRUF This routine loads the corner ghost face velocity for block IBLK.

CRNRXV This routine loads the corner ghost vertex coordinates for block IBLK.

DIMCHECK This routine determines whether the dimension limits are exceeded and, if so, aborts the job. It also aborts the job if certain required input quantities have not been specified.

DRAWLINE This routine draws a sequence of line segments or vectors. Coordinates are stored consecutively (NSTRIDE = 1) or in pairs (NSTRIDE = 2) in the one-dimensional arrays X and Y.

DRAWMESH This routine draws the entire computational mesh when IBRDR = 0 and only the Lagrangian surfaces otherwise.

DRVG This routine draws a line from (X1,Y1) to (X2,Y2).

DUMPRD This routine reads dump NRESTART from file RS2D.

DUMPWR This routine writes a dump containing the contents of most of the labeled common blocks into file DP2D.

EOSDRV This routine computes the cell-centered pressures and sound speeds from the equation of state.

FACEPR This routine computes the cell-face quantities RHL, RHR, PRL, and PRR required by the Riemann solver routines RIEMANN and BNDRIEM.

FACEVEL This routine computes the cell-face normal velocities UNL and UNR and cell-face tangential velocities UTL and UTR required by the Riemann solver routines RIEMANN and BNDRIEM.

FLAGSET This routine generates the flag array IFLG.

GCLR	This routine sets the color in a range from blue to red depending on the value of X, which assumes values between 0. and 1.
GEOM	This routine computes various geometrical quantities.
GLOSSARY	This routine contains descriptions for most of the variables used in the CAVEAT code.
GRADDRV	This routine is a driver for computing gradients. Flag IRIEMADV specifies whether gradients are for the Riemann calculation or for the advection calculation (in which case the fields are multiplied by density).
GRADIENT	This routine computes the gradient GV of V in each cell. This provides for a linear variation of V within a cell. Monotonic limiting constrains the gradient such that no overshoots or undershoots occur relative to neighboring cells. More aggressive van Leer limiting (default) is also available, but its local sawtooth oscillations can occasionally lead to instability.
GRAPHICS	This routine is the driver for the graphical output.
GRIDBOX	Draw an x-y coordinate axes and put grid tick marks on all 4 sides of the plot.
GRPHINIT	This routine initializes the graphics routines.
GVECT	This routine draws a vector from point (X,Y) to point (XX,YY). The head of the vector is a triangle.
HOMEOS	Given the specific volume and one variable of the set specific internal energy, pressure and temperature, calculate the other two variables of the set, using the HOM analytic equation of state.
HYDINIT	This routine initializes arrays UC, RHO, SIE, TE, CM, and RA.
HYDROCYC	This routine advances the hydrodynamic solution one time step. For the Lagrangian phase we use Godunov's method which requires solving a Riemann problem for each of the cell faces in the mesh.
HYDROOUT	This routine provides graphical, printed, and disk output at times specified by the input arrays TWFILM, TWPRNT, and TWDUMP.
ICONV	This function takes the 64 bit number X (floating point or integer) and masks the exponent and fraction to form a new word of length = NBITS. NEXPB is the number of exponent bits in the new word.
IEOSSET	This routine loads the array IEOS for the gather/scatter operations in routine EOSDRV.
LABEL1D	This routine generates labels for the 1-D plots.
LABEL2D	This routine generates labels for the 2-D plots.
LAGHYDR	This routine computes the pressure acceleration and div(P*V) work and updates the velocities and total energies accordingly.
LAGVEL	This routine calculates Lagrangian vertex velocities by a least squares method using the four normal face velocities surrounding the vertex.
LANLLOGO	This routine writes "Los Alamos", in hollow letters, on the line from (XORG,YORG) to (XORG+WIDTH,YORG) in window coordinates.
MESHGEN	This routine generates the computational mesh.
NEIGHBRS	This routine computes offsets needed to reference neighboring cells, faces, and vertices.
NEWCYC	This routine updates the state variables and geometrical arrays and finds the new time step in preparation for another hydro cycle.
NEWPROB	This routine performs the setup for a new problem.
PACKM	This routine takes 20-bit right-justified zero-filled words and packs them 3 per word.

PLOT1D This routine generates 1-D profiles of density, specific internal energy, pressure, and velocity magnitude.

PLOTSIZE This routine sizes the 2-D plots. Plots are scaled to be slightly undersize to insure proper display of velocity vectors at mesh boundaries.

PNTRBLK This routine computes array pointers for block IBLK and stores them in common block /PNTR2/.

PNTRSET This subprogram computes the pointers for the mesh-wide arrays and adjusts the field length to accommodate these arrays.

PRECON This routine computes the field at the vertices and finds the resulting field limits over all the mesh blocks.

PREMESH This routine finds the mesh limits over all the blocks.

PREVEL This routine computes the velocity scaling factor.

PRNTOUT This routine prints the time, cell-centered coordinates, density, specific internal energy, pressure, and velocity components.

RDINPUT This routine reads the input data as namelist 'INPUT' from file IN2D.

REMAP This routine performs the remap phase of the hydrodynamics cycle.

REMESH This routine calculates new vertex coordinates by adding the product of the time step and the vertex velocity to the old vertex coordinates.

RESTART This routine reads a restart dump from file RS2D and creates the arrays needed for resumption of the calculation.

RESTORE This routine restores the default graphics environment.

REZBND This routine loads array UMR on fluxing boundaries with values corresponding to the component of array UM normal to the boundary. Elsewhere UMR is set to zero.

REZCOEF This routine computes the arrays of coefficients required in the iterative Brackbill rezoning algorithm.

REZINIT This routine finds the indices of the interface vertices to be treated in a special fashion, computes the initial vertex locations to be used in the rezoning procedure, and calls the routine that generates the rezoning weight function. Note that refined zoning in certain regions of the mesh may be obtained by proper choice of the rezone weight function WT (see subroutine WEIGHT).

REZITER This routine performs a single successive-over-relaxation iteration of the Brackbill rezoning algorithm in block IBLK.

REZLIM This routine limits the mesh velocity such that the non-Lagrangian displacement is no larger than 0.8 times the distance to the nearest vertex for block IBLK.

REZONE This routine modifies the mesh velocities based on a rezone algorithm developed by J. U. Brackbill. Fine zoning in blocks of interest is obtained by proper choice of a rezone weight function WT (see subroutine WEIGHT).

REZTANG This routine rezones block boundaries to provide an almost uniform spacing of vertices along straight segments and closer spacing in sections of large curvature. This latter feature is controlled by the parameter CRVWT.

RIEMANN This routine solves the Riemann problem on the cell faces.

RIEMDRV This routine is a driver for the Riemann solver routines.

ROUND Rounds or truncates the elements of array y to NSD significant digits.

SESAME This routine computes pressures and sound speeds from densities and internal energies using the SESAME tables.

SESEOS	This routine computes pressures and sound speeds from densities and internal energies using the SESAME tables.
SESSET	This routine calls the EOSPAC routines to initialize the SESAME equation of state tables.
SESSIE	This routine computes specific internal energies from pressures and densities using the SESAME tables.
SIEINIT	This routine computes the specific internal energy SIE and the sound speed SS, given the pressure PR, density RHO, and material index IMAT.
SPHRGEN	This routine generates a 180 degree circular mesh consisting of four blocks.
TIMING	This routine outputs the timing information accumulated in the arrays "SEC" and "NOPS".
TIMSTP	This routine calculates the hydro time step in block IBLK.
UPDATE	This routine updates the density and specific internal energy in block IBLK for the next hydrodynamics cycle.
VELOCITY	This routine scales and computes velocity vectors.
VERTEX	This routine computes a vertex array W based on the cell-centered array V by volume averaging over the four neighboring cells.
VMINMAX	This routine finds the minimum and maximum values of the array V.
VOLUME	This routine computes the cell volumes.
VOLVTX	This routine computes the vertex-centered volumes.
WEIGHT	This routine computes the rezone weight function.
WRGAS	This routine directs the writing of a restart dump and handles: 1) the writing of the master directory, 2) the familying of gas files (if necessary), 3) the truncation of the gas file.
WRGAS1	This routine writes the GAS dump file.

A.4 VARIABLES

This appendix contains all the variables, in alphabetical order, that are used in CAVEAT, along with brief definitions. Note that this information is also contained in the listing of CAVEAT in the subroutine GLOSSARY. The meaning of the arguments for the variables, if required, are given in the Notes on Nomenclature section and in §A.1 above. If a variable is an input parameter, the symbol '*' follows the variable; if the input variable has no default parameter, the symbol used to identify the input parameter is '@'. Following the definition in parentheses, the common block that contains the variable, the default value if the variable is an input parameter, and the section in this report that discusses the variable are given.

ALECOEF*	Parameter specifying the extent of remapping. A value of zero implies calculation is in the Eulerian limit, while a value of one implies the Lagrangian limit. Intermediate values result in both rezoning of the mesh and fluxing across cell faces. (CNTRL, 0.9; §4.3.2h, §4.4.6, §5.3)

ANTIDIF*	Factor for controlling the amount of 'anti-diffusion' applied in the second order Godunov treatment. A value of zero (default) applies no anti-diffusion, while a value of one applies the maximum allowable amount. A value of one typically provides sharper resolution of shocks with no discernable negative consequences. In some situations, however, it is marginally unstable and leads to obvious spurious pressure oscillations. (CNTRL, 0.0; §4.3.1b)
BNDCOEF*	Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on boundaries in routine REZBND. (REZN1, 0.5; §4.3.2h)
CLOCK	Eight-character Hollerith wall clock time. (GRAF2)
CM	Cell masses. (pointer variable; §4.1.3a)
CONVD*	Conversion factor for density units from SESAME tables. (EOS02, 1.0 implies units of g/cm^3; §4.4.4)
CONVE*	Conversion factor for energy units from SESAME tables. (EOS02, 0.01 implies units of $Mbar\text{-}cm^3/g$; §4.4.4)
CONVP*	Conversion factor for pressure units from SESAME tables. (EOS02, 0.01 implies units of Mbar; §4.4.4)
CRVWT*	Parameter for weighting the effect of curvature in the tangential rezone of block boundaries. Increasing the value of CRVWT produces finer zoning in blocks of high curvature. (REZN1, 1; §4.3.2h)
CUSHION*	Time increment in seconds before job time limit is reached that soft abort of job is initiated. (TIME1, 0; §4.4.6)
DAY	Eight-character Hollerith date. (GRAF2)
DT(IBLK)	Time step computed for block IBLK. (TIME2)
DT0@	Initial time step for the hydro cycle. (TIME2, no default; §4.4.6, §5.2)
DTDUMP	Time increment between calls to dumpwr. (TIME3)
DTFAC*	Fraction of the Courant time step limit allowed for hydro time step. (TIME2, 0.5; §4.4.6, §5)
DTFILM	Time increment between calls to graphics. (TIME3)
DTHYDRO	Current time step for the hydro cycle. (TIME2)
DTMAX*	User specified maximum for hydro time step. (TIME2, 1000*DT0; §4.4.6, §5.2)
DTMIN*	User specified minimum for hydro time step. (TIME2, 0.0001*DT0; §4.4.6, §4.5f, §5.2)
DTMOVI	Time increment between calls to movie. (TIME3)
DTPRNT	Time increment between calls to prntout. (TIME3)
DX(KPRT,M,IBLK)@	Initial mesh spacing in each part, mesh direction, and block. (MESH2, no default; §4.4.1)
FA	Areas of the cell faces. (pointer variable; §4.1.3a)
FDONOR*	Extent of donor-cell treatment in the advection calculation. A value of one results in full donor-cell treatment. (CNTRL, 1.0; §4.3.3b)
FLUX	Fluxing volumes across cell faces. (pointer variable; §4.1.3a, §4.3.3a)
FN	Unit normal vectors for the cell faces. (pointer variable; §4.1.3a)
FNBC(M,IB,IBLK)	Unit normal direction components for symmetry boundaries of block IBLK. (BNDR1; §4.4.5)
GASSCL(30)*	Multiplicative scale factors for GAS variables. (GRAF1, 1.0; §4.5e)

GE	Cell specific total energy gradients. (pointer variable; §4.1.3a)
GP	Cell pressure gradients. (pointer variable; §4.1.3a)
GR	Cell density gradients. (pointer variable; §4.1.3a)
GU	Cell velocity gradients. (pointer variable; §4.1.3a)
HANDED	Parameter indicating the handedness of the mesh. A value of +1. implies a right-handed mesh while a value of -1. implies a left-handed mesh. This parameter is computed automatically by the code. (MESH3, 1)
I1BC(IB,IBLK)	Starting index of the do-loop over the mesh for each of the four boundaries of block IBLK. (BNDR1; §4.4.5)
I2BC(IB,IBLK)	Ending index of the do-loop over the mesh for each of the four boundaries of block IBLK. (BNDR1; §4.4.5)
I2DAXIS*	Flag indicating whether or not coordinate axes with tic marks are to be drawn on 2-D plots. I2DAXIS= 0 implies no; I2DAXIS= 1 implies yes. (GRAF1, 1; §4.5d)
I2DPLOT*	Flag indicating whether or not 2-D plots of pressure, density, internal energy, and velocity are desired. I2DPLOT = 0 implies no; I2DPLOT = 1 implies yes. (GRAF1, 0; §4.5d)
I3BC(IB,IBLK)	Stride of the do-loop over the mesh for each of the boundaries of block IBLK. (BNDR1; §4.4.5)
IA(IB)	Mesh direction along each side IB of a block. For instance, IA(1) equals 1 in the current CAVEAT framework. (BCOM1; §4.1.4)
IBC(IB,IBLK)*	Indices specifying the boundary types for the four boundaries of block IBLK as follows: IBC = 1 (non-curved) reflective boundary IBC = 2 applied pressure IBC = 3 outflow IBC = 4 inflow IBC = 5 boundary between blocks--use ghost cells for inter-block communication IBC = 8 curved, undeformable free-slip boundary IBC = 9 specified velocity (piston) boundary The order for the four boundaries is bottom, top, left, and right. (BNDR1, 1; §4.4.5)
IBCOM	Flag set equal to 1 if block communication by ghost cells is part of the current run. (BCOM1; §4.1.4)
IBCREZN(IB,IBLK)*	Parameter specifying application of a special tangential rezone treatment on boundary IB of block IBLK. When set to one this parameter provides uniform spacing of vertices along straight boundaries and concentrated spacing in blocks of high curvature. This treatment is appropriate, for example, when jetting is expected. (REZN1, 0; §4.3.2h)
IBF(M,IB,IBLK)	Additional offset for certain faces in acceptor block. (BCOM2; §4.1.4)
ICELL(KPRT,M,IBLK)	Index of first cell in part KPRT for mesh direction M in block IBLK. (MESH3)
ICN(M,IBLK)	Offset from the base index for the neighboring cell on the opposite side of the base cell face referenced by the two mesh directions. (MESH5; §4.1.3c)
ICV(4,IBLK)	Offsets from the base index for the four vertices associated with a cell. The negatives are the offsets from the base index for the four cells surrounding a vertex. (MESH5; §4.1.3c)
IDELBC(IB,IBLK)	Offsets for cells on the inner sides of the four mesh boundaries for block IBLK. (BNDR1; §4.4.5)

IDEN(5)	Eight-character Hollerith names of fields for which 2-D plots are to be generated. (GRAF2)
IEOS	Array of gather/scatter indices required in EOSDRV. (pointer variable; §4.1.3a)
IEOSPNT(30,IBLK)	Index of last element for a given material in array ieos for block IBLK. (EOS01)
IFLG	Flags for ghost cells, non-fluxing mesh boundaries, applied pressure boundaries, and material interfaces. (pointer variable; §4.1.3a, §4.1.5)
IFLUX*	Flag to specify method for computing fluxing volumes in routine ADVFLUX. A value of one specifies a method based of vertex velocities. Any other value specifies a method based on the face velocities. (CNTRL, 1.0; §4.3.3)
IFRST(IBLK)	Starting index for most do-loops, equal to N1+5. (MESH4; §4.1.3c)
IFV(M,IBLK)	Offset from the base index for the second vertex associated with a cell face in the two mesh directions. (MESH5; §4.1.3c)
IGAS*	Flag indicating whether or not GAS dumps are desired. (GRAF1, 0; §4.5e)
IGEOM*	Parameter specifying the geometry type (MESH3, 1; §4.4.1): IGEOM = 1 generates a 'x-y' rectangular mesh IGEOM = 2 generates an 'r-z' cylindrical mesh IGEOM = 3 generates a polar 'x-y' cylindrical mesh IGEOM = 4 generates an 'r-z' spherical mesh IGEOM = 5 generates a polar 'x-y' cylindrical mesh composed of four mesh blocks IGEOM = 6 generates an 'r-z' spherical mesh composed of four mesh blocks
ILINE(M,10)*	Array specifying lines along which graphical output is desired. Elements of first index specify respectively the mesh direction of the line and the indices identifying the line in the other mesh direction. (GRAF1, -1; §4.5d)
INF(M,IB,IBLK)	Additional offset for certain faces in donor block. (BCOM2; §4.1.4)
INITRZN*	Number of rezone iterations to be applied to the mesh during problem initialization. If greater than zero, this parameter allows mesh to relax toward the solution of the Winslow generator equations prior to starting the hydrodynamic calculations. (REZN1, 0; §4.3.2h, §4.4.1)
IORDER*	Flag to indicate first or second order spatial treatment in the advection and Riemann routines. IORDER=1 implies first order, while IORDER=2 implies second order. (CNTRL, 1; §4.3.1b, §4.3.3b, §5.3)
IPSESTB	Location of beginning of SESAME tables in memory. (EOS02; §4.1.3a)
IRADIAL*	Parameter that controls the orientation of the cylindrical axis. For IRADIAL = 1, the cylindrical axis coincides with coordinate direction two and the radial direction corresponds with coordinate direction one and the I1=1 mesh line is the cylindrical axis. When IRADIAL = 2, the radial direction corresponds to coordinate direction two and the I2=1 mesh line is the cylindrical axis. When IRADIAL = 0, there is no cylindrical axis in the plane of the mesh. Selecting IGEOM = 1, 3, or 5 automatically sets IRADIAL = 0. (MESH3, 1; §4.4.1)
ISEQNO(30)	Material sequence number for SESAME materials. (EOS02)
ISESOPT*	Parameter specifying interpolation method for SESAME tables. Choices are: biline for 4-point bilinear; 6hBIQUAD for 6-point biquadratic; 6hBIRATF for 12-point bi-rational-function. (EOS02, 6hBIRATF; §4.4.4)
ITREZN*	Number of iterations for Winslow rezone procedure. (REZN1, 3; §4.3.2h)
LASTC(IBLK)	Ending index for most do-loops over cells, equal to (N1+3)*(N2+1). (MESH4; §4.1.3c)

LASTV(IBLK)	Ending index for most do-loops over mesh vertices and faces, equal to (N1+3)*(N2+2). (MESH4; §4.1.3c)
LIMGRAD*	Flag to indicate type of limiting used in second-order treatment. Set LIMGRAD=1 for monotonic limiting, LIMGRAD=2 for van Leer limiting, and LIMGRAD=0 for for van Leer limiting on all variables except velocity. (CNTRL, 2; §4.3.1b, §5.3)
LMCL(IBLK)	Index of cell in block iblk responsible for time step limitation in that block. (TIME2)
LOC(IB,IBLK)	Mesh offset in donor block for cell-centered quantities. (BCOM2; §4.1.4)
LOE(IB,IBLK)	Mesh offset in donor block for boundary quantities. (BCOM2; §4.1.4)
LOF(IB,IBLK)	Mesh offset in donor block for face-centered quantities. (BCOM2; §4.1.4)
LOGO*	Flag specifying whether LANL logo is to be included on on plots. LOGO = 0 implies no; LOGO = 1 implies yes. (GRAF2, 0; §4.5d)
LOV(IB,IBLK)	Mesh offset in donor block for vertex-centered quantities. (BCOM2; §4.1.4)
LS1(IB,IBLK)	Mesh stride in donor block (block NBC(1,IB,IBLK) along side NBC(2,IB,IBLK). (BCOM2; §4.1.4)
MATBC(KPRT,IB,IBLK)*	Material numbers outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1; §4.4.5)
MATNUM(IP,IBLK)*	Material number by part and block. This number ranges between 1 and 30 and identifies which material of those specified by array proprty resides in a given part and block. (EOS01, 1; §4.4.2)
MOB(IBLK)	Beginning index offset for scalar arrays in block IBLK. (MESH1; §4.1.3b)
MOC(IB,IBLK)	Mesh offset in acceptor block for cell-centered quantities. (BCOM2; §4.1.4)
MOE(IB,IBLK)	Mesh offset in acceptor block for boundary quantities. (BCOM2; §4.1.4)
MOF(IB,IBLK)	Mesh offset in acceptor block for face-centered quantities. (BCOM2; §4.1.4)
MOV(IB,IBLK)	Mesh offset in acceptor block for vertex-centered quantities. (BCOM2; §4.1.4)
MS1(IB,IBLK)	Mesh stride in acceptor block (block IBLK) along side IB. (BCOM2; §4.1.4)
MSIZV(IBLK)	Net length of most do-loops over mesh vertices and faces, equal to (N1+3)*(N2+1). (MESH4; §4.1.3c)
MSKFIX	Mask for freezing the position of a mesh vertex. (MASKS; §4.1.5)
MSKGST	Mask for ghost cell. (MASKS; §4.1.5)
MSKIVX	Mask for vertex on a material interface. (MASKS; §4.1.5)
MSKLF1	Mask for Lagrangian interface along mesh direction 1. (MASKS; §4.1.5)
MSKLF2	Mask for Lagrangian interface along mesh direction 2. (MASKS; §4.1.5)
MSKLVX	Mask for Lagrangian vertex. (MASKS; §4.1.5)
MSKMAT	Mask for material index. (MASKS; §4.1.5)
MSKREV	Mask for Sesame EOS reversibility indication. (MASKS; §4.1.5)
MSKTR1	Mask for tangential rezone along mesh direction 1. (MASKS; §4.1.5)
MSKTR2	Mask for tangential rezone along mesh direction 2. (MASKS; §4.1.5)
MSZ(IBLK)	Basic scalar array dimension for block IBLK, equal to (N1(IBLK) + 3)*(N2(IBLK) + 3). (MESH1; §4.1.3)
N1(IBLK)*	Number of cells in mesh direction one in block IBLK. (MESH1, 1; §4.4.1)
N2(IBLK)*	Number of cells in mesh direction two in block IBLK. (MESH1, 1; §4.4.1)

NADVSKP* Number of Lagrangian steps for every advection step in ALE hydrodynamic calculation. This parameter must be four or greater for any advection steps to be skipped. (CNTRL, 1; §4.3.1b, §5.3b)

NAMES(30) Subroutine names used in accounting routine TIMING. (ACCNT)

NB Parameter for maximum number of blocks. (parameter variable, 4; §4.4.1, §4.1.4, §5.3)

NBC(2,IB,IBLK)* Input array specifying the block communication setup. NBC(1,IB,IBLK) is set equal to the block with which side IB communicates. NBC(2,IB,IBLK) is set equal to the side of block NBC(1,IB,IBLK) with which side IB communicates. (BCOM1, 0; §4.1.4)

NBLKS* Number of logically rectangular blocks comprising the problem domain. Complex geometries can be generated by joining blocks of logically rectangular mesh which communicate across their common boundaries. (MESH1, 1; §4.4.1)

NCELL(KPRT,M,IBLK)@ Number of cells associated with each part, mesh direction, and block. Dimensions allow up to NS different parts in each mesh direction. (MESH2, no default; §4.4.1)

NCON Number of contour intervals for contour plots. (GRAF2, 10; §4.5d)

NCYC Hydro cycle number. (TIME1)

NCYC0 Initial cycle number--used to maintain cycle count upon problem restart. (TIME1)

NCYCMAX* Problem hydro cycle limit. (TIME1, 10000; §4.4.6)

NDUMP Number of next call to DUMPWR. (TIME3)

NEDG(M,IBLK) Number of cells along each mesh direction to be included in communication routines. (BCOM1; §4.1.4)

NEDGF(M,IBLK) Number of faces along each mesh direction to be included in communication routines. (BCOM1; §4.1.4)

NFILM Number of next call to graphics. (TIME3)

NMOVI Number of next call to movie. (TIME3)

NOPS(30) Cumulative operation count for individual subroutines. (ACCNT)

NP Parameter for maximum number of part in any block. (parameter variable, 40; §4.4.1, §4.5f, §5.3)

NPRNT Number of next call to prntout. (TIME3)

NPRTS(M,IBLK)* Number of parts or subdivisions in each mesh direction within block IBLK. This feature allows for multiple initial states, materials and/or initial mesh spacings to be specified within a single block. (MESH2, 1; §4.4.1)

NRESTART* Dump number on file RS2D for problem restart. (TIME1, -1; §4.5c)

NS Parameter for maximum number of parts divisions along a mesh direction. (parameter variable, 8; §4.4.1, §4.5f, §5.3)

NSMOOTH* Number of smoothing iterations in the calculation of the rezone weight function. (REZN2, 0; §4.3.2h)

NWK(I)* Size of additional work arrays as needed for development of new material models or numerical methods. See §4.1.3d for a detailed description and use. (pointer variable, 0; §4.1.3d)

PBC(KPRT,IB,IBLK)* Pressures outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)

PF Pressures on cell faces. (pointer variable; §4.1.3a)

PNAME	Eight-character Hollerith program name; e.g., CAVEAT . (GRAF2)
PR	Cell pressures. (pointer variable; §4.1.3a)
PR0(IP,IBLK)@	Initial pressure associated with each part and block. (INIT1, no default; §4.4.1)
PROPRTY(100,30)*	Material property array.location (EOS01, see §4.4.3 for defaults) 1: EOS type (Values 1-8 imply gamma-law, linear, quadratic, blank, HOM, blank, blank, and SESAME, respectively.) 2: Reference density for EOS 3: Pressure floor 4: Riemann strong shock parameter 5-100: EOS model parameters
RA	Cell Riemann strong shock parameters. (pointer variable; §4.1.3a)
RABC(KPRT,IB,IBLK)	Riemann strong shock parameters outside the four mesh boundaries of block IBLK by part. (BNDR2; §4.4.5)
RAV	Cell average radii (used in routine LAGHYDR). (pointer variable; §4.1.3a)
RHO	Cell densities. (pointer variable; §4.1.3a)
RHO0(IP,IBLK)@	Initial density associated with each part and block. (INIT1, no default; §4.4.1)
RHOBC(KPRT,IB,IBLK)*	Material densities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, 1.0; §4.4.5)
SCLARRW*	Scale factor for arrows in 2-D velocity plots. (GRAF2, 1; §4.5d)
SEC(30)	Cumulative CPU time for individual subroutines. (ACCNT)
SIE	Cell specific internal energies. (pointer variable; §4.1.3a)
SIE0(IP,IBLK)	Initial specific internal energy associated with each part and block. (INIT1; §4.4.1)
SIEBC(KPRT,IB,IBLK)	Specific internal energies outside each of the four mesh boundaries of block IBLK by part. (BNDR2; §4.4.5)
SS	Cell sound speeds. (pointer variable; §4.1.3a)
SS0(IP,IBLK)	Initial sound speed associated with each part and block. (INIT1; §4.4.1)
SSBC(KPRT,IB,IBLK)	Sound speeds outside each of the four mesh boundaries of block iblk by part. (BNDR2; §4.4.5)
T	Problem time. (TIME1)
TANREZN*	Parameter specifying fraction of first-order solution to be used in stabilizing rezone treatment on interfaces. (REZN1, 0.02; §4.3.2h)
TDUMP	Time at which next call to dumpwr is to occur. (TIME3)
TE	Cell specific total energies. (pointer variable; §4.1.3a)
TEL	Lagrangian-phase cell specific total energies. (pointer variable; §4.1.3a)
TFILM	Time at which next call to graphics is to occur. (TIME3)
TITL(5)*	Hollerith string containing problem title. (GRAF1, blank)
TMOVI	Time at which next call to movie is to occur. (TIME3)
TPRNT	Time at which next call to prntout is to occur. (TIME3)
TPULSE*	Width of the applied pressure pulse for cylindrical phased implosion problems. (PRES1, 0.0)
TWDUMP(10)*	Same as twprnt, except restart dump on file DP2D. (TIME1, 1.E50; §4.5c)
TWFILM(10)*	Same as twprnt, except for graphics output. (TIME1, 1.E50; §4.5d)

TWFIN*	Problem time limit. (TIME1, no default; §4.4.6)
TWMOVI(10)*	Same as twprnt, except for movie output. (TIME1, 1.E50)
TWPRNT(10)*	Times at which printer output is written to TAPE6. If TWPRNT(3) is less than TWPRNT(2), TWPRNT(3) is interpreted as the time increment between output times. (TIME1, 1.E50; §4.5b)
U	Cell material velocities. (pointer variable; §4.1.3a)
UBC(M,KPRT,IB,IBLK)*	Material velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0); §4.4.5)
UC0(M,IP,IBLK)*	Initial Cartesian velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)
UCL	Lagrangian-phase cell velocities. (pointer variable; §4.1.3a)
UF	Normal components of the material velocity on cell faces. (pointer variable; §4.1.3a)
UM	Mesh velocities at the vertices. (pointer variable; §4.1.3a)
UR0(IP,IBLK)*	Initial radial velocity associated with each part and block. (INIT1, (0.,0.); §4.4.1)
URBC(KPRT,IB,IBLK)*	Radial velocities outside each of the four mesh boundaries of block IBLK by part. (BNDR2, (0.,0,); §4.4.5)
VANLEER*	Factor for reducing the magnitude of the van Leer limited gradients to maintain stability. (CNTRL, 1.0)
VOL	Cell volumes. (pointer variable; §4.1.3a)
VPHASE*	Phase velocity in the y-direction for cylindrical phased implosion problems. (PRES1, 0.0)
WFMT(30)*	Weighting of individual materials. (REZN2, 0.0; §4.3.2h)
WFGR*	Weighting of spatial gradients of various scalar fields: zero implies none; positive values specify the pressure; negative the density field. (REZN2, 0.0; §4.3.2h)
WFLM*	Limiting value for the gradient, above which the gradient contribution saturates. Small values allow regions with smaller gradients to find expression in the weighting. (REZN2, 1.e100; §4.3.2h)
WFVO*	Weighting of equal volume. (REZN2, 1.0; §4.3.2h)
WTAMPL*	Ratio of maximum to minimum values for the rezone weight function. The default value of one gives a constant function corresponding to uniform volume weighting. Values greater than one should be used when a polar geometry is used (recommended range is 5.0-10.0) or if any of the other weight function options are exercised. (REZN2, 1.0; §4.3.2h)
X0(M,IBLK)*	Origin offset relative to the physical origin for block IBLK. (MESH2, (0.,0.); §4.4.1)
XC	Cartesian coordinate positions of the cell centers. (pointer variable; §4.1.3a)
XV	Cartesian coordinate positions of the mesh vertices. (pointer variable; §4.1.3a)

APPENDIX B

USING CAVEAT AT LANL

This section outlines the procedure necessary to create an executable version of CAVEAT on the Los Alamos Central Computing Facility (CCF). The procedure entails getting the proper files, compiling the source code if necessary, and loading or combining the files to create an executable code. This section assumes the user has access to the open partition of the CCF.

CAVEAT is maintained with the HISTORIAN utility (for those outside LANL and using CAVEAT on a Cray system, the Cray UPDATE utility is similar in function). HISTORIAN is a utility that provides simplification of repetitive tasks and record keeping when maintaining a large code. Manuals on this utility are available for the Computer Information Center at Los Alamos National Laboratory.

To change the code one produces an HISTORIAN "update" file that consists of sets of modifications to the source code (examples of these update files are given in §6). In the discussion below, the update file refers to the current collection of modifications that make corrections or extensions to the source code. When these updates accumulate, they are combined with the source code to create a new version of the code. Note that the file that contains the source code is a permanent file and is never altered; once a new source code is created, all modifications are done using updates. An experienced user or developer of CAVEAT adds to the update file to make required modifications to the source code.

All the files necessary to create an executable version of the code are stored in a single *library* file on the Common File System (CFS). A library file is a single file that contains many files and is generated and maintained using the CFS library utility, LIB. For the version documented in this manual, the library file for CAVEAT, CORE2D01, can be obtained using the command: CFS GET /CAVEAT2D/CORE2D01. (Access to this CFS node, CAVEAT, can be obtained by contacting any of the authors of this report.) This library file contains the source code for CAVEAT (CORE.S), the complied binary (CORE.B) of the source code that *does not include any updates*, the current update file (CORE.U) for the source code, the controllers for compilation and loading (ending in .P), and a collection of input files (ending in .I) and the corresponding updates if required (ending in .U) that are needed to run the test problem. The input files for the example problems in §6 are included in the library file.

Extracting files from the library file

Once the library file CORE2D01 is in the users local file space, the utility for accessing the files LIB is used to extract the necessary files by entering:

```
LIB CORE2D01
EXTRACT ALL.
END
```

All the files in the library file should now be in the local file space. (See documentation on the utility LIB in order to extract only the files of interest.)

Updating and compiling the source files

To compile the source file, CORE.S, with the provided updates or user supplied updates, BASE.U, execute the controller , CF77.P, using XEQ:

```
XEQ CF77.P
```

This controller calls HISTORIAN to create a source file, calls HISTRORIAN again to insert the updates from CORE.U, compiles the new source code using the Cray FORTRAN 77 compiler (CFT77) and, creates an executable using LDR. The main files that are produced are

CORE.B:	the new binary file,
NCORE.S	the new histrorian source file,
CMPL	the source code that is compiled, and
XH2D	the executable code.

The files NCORE.S and CMPL can be used to verify the changes made to the original source.

Changing the dimensioning parameters (NB, NS, NP)

In some applications the initial problem setup is sufficiently complex that the parameters NB, NS, NP must be increased in size (if these are too small for the problem, an error message is given; see §4.5f). This is accomplished by modifying the parameter statement in the source file, CORE.S either by directly editing the file or by creating an update using historian to modify this line. Because historian distributes this parameter statement throughout the code, the parameter statement need only be modified at one location.

Obtaining a hard copy of the source

A hard copy of the source can be obtained by having a copy of CORE.S in the local file space and then entering:

```
HSP CORE.S  -V  -SEQ  -SEQ1
```

The addition of commands to include line sequence numbers (from SEQ and SEQ1) are useful for locating the lines numbers required by HISTORIAN updates.

APPENDIX C

CRAY SYSTEM ROUTINES AND ALTERNATIVES

Throughout the CAVEAT code, non-standard FORTRAN statements, special Cray system calls, and graphics subroutine calls are used. This section describes these functions and alternatives when available. In many instances, the non-Cray users can create a set of local functions to do the same job.

C.1 VECTOR MERGES (CVMG_)

Table C-1 defines the CVMG_ functions.

Table C-1. Cray CVMG_ functions. The functions have the general form: $x = CVMG\_(x_1,x_2,x_3)$.

CVMGP:	x_1 returned if $x_3 \geq 0$, x_2 otherwise.
CVMGM:	x_1 returned if $x_3 < 0$, x_2 otherwise.
CVMGZ:	x_1 returned if $x_3 = 0$, x_2 otherwise.
CVMGN:	x_1 returned if $x_3 \neq 0$, x_2 otherwise.
CVMGT:	x_1 returned if x_3 is true, x_2 otherwise.

C.2 ARRAY SORTING (WHENF__)

CAVEAT makes use of two Cray system routines for sorting purposes. These are WHENFGT, called in routine REZINIT, and WHENFLE, called in RIEMANN. The syntax and description are given in Table C-2. Referring to the variables defined in Table C-2, on entry, *n* is the number of points in *array*, *inc* is the increment to be taken in the searching process (stride). On return, the array *index* contains the locations in the array *array* for which the test is satisfied; there are *nval* points in array *index*.

Table C-2. Cray WHENF__ functions.

WHENFGT(*n,array,inc,target,index,nval)*	Returns all locations in real *array* that are greater than the real *target*.

WHENFLE(*n,array,inc,target,index,nval)*	Returns all locations in real *array* that are less than or equal to the real *target*.

C.3 SPECIAL ROUTINES (EBM, EXITA, MEMADJ)

CAVEAT uses a variety of special routines on the Cray. The first is EBM which enables bi-directional use of memory on the Cray. This call results in faster use of memory. Non-Cray users can eliminate this call.

To stop the program, EXITA is used so that a "drop file" is left in the user's local file space. This "drop file" can be used to examine the state of the program at termination using a debugging program. Non-Cray users can replace all calls to EXITA with a Fortran STOP.

Finally, MEMADJ is called in PNTRSET in order to allocate the needed memory space required during the problem execution. Non-Cray users will find it necessary to replace PNTRSET with an equivalent routine that will depend upon the actual hardware used. Alternatively the mesh-wide arrays can be dimensioned to a specific mesh size in PNTRSET if the target computer does not support adjustable memory size at execution.

C.4 GRAPHICS ROUTINES

Because CAVEAT contains graphics output, a large number of of low level graphic calls are made. The graphics routines use the Los Alamos Common Graphics System and are not generally available at other facilities. Because the graphics routines used are simple graphics calls (for example, line drawing routines) the graphics capability within CAVEAT can be modified to use other systems with minimal difficulty.

APPENDIX D

SESAME TABULAR EQUATION OF STATE LIBRARY

The SESAME Equation of State Library is a standardized, computer-based library of tables of thermodynamic properties for a wide range of materials that is maintained and distributed by the Los Alamos National Laboratory. A good introduction and summary of SESAME is provided in the Handbook, edited by K. Holian [Holian, 1984]. Each material in the library is identified by an identification number, listed in the Handbook.

SESAME internally assumes units of Mg/m^3, GPa, and MJ/kg for density, pressure, and energy, respectively. These units may be changed for use in CAVEAT by specifying the corresponding conversion factors **CONVD**, **CONVP**, and **CONVE,** in the input file IN2D. The default values are 1.0, 0.01, 0.01, which convert the SESAME output values to units of g/cm^3, MBar, and $MBar\text{-}cm^3/g$ for density, pressure, and energy, respectively.

CAVEAT accesses the SESAME tables through a subroutine library called EOSPAC. This is a package of FORTRAN software, developed and documented by C. Cranfill [Cranfill, 1983], which is vectorized for efficient use on Cray computers.