The Formation of Short Period Binary Star Systems From Stable, Self-Gravitating, Gaseous Bars.

John E. Cazes Jr
Louisiana State University and Agricultural & Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_disstheses

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_disstheses/6982

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Historical Dissertations and Theses by an authorized administrator of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600
THE FORMATION OF SHORT PERIOD BINARY STAR SYSTEMS 
FROM STABLE, SELF-GRAVITATING, GASEOUS BARS

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
The Department of Physics and Astronomy

by
John E. Cazes, Jr.
B.S., Louisiana Tech University, 1990
M.S., Louisiana Tech University, 1992
August, 1999
ACKNOWLEDGEMENTS

First and foremost, I thank my wife, Christie, for standing by me throughout this journey. Without her support, I would have fallen by the wayside long ago. I also thank my daughter, Elizabeth, for lifting my spirits when I needed it the most and reminding me that there is more to life than work. Special thanks also go to my parents, who have supported me throughout my academic career, and whose constant inquiries about my expected graduation date are about to end!

Some of the best years of my life have been spent being a part of the "Hydro"-group here at LSU. In large part, this is due to the support and enthusiasm of our advisor, Joel Tohline. The encouragement and freedom he has given our group has allowed me to enjoy being a graduate student and to look forward to coming to work every day. Without Joel's formidable editorial skills, this dissertation would still be a work in progress.

The rest of our group also deserves much thanks. Along with Joel, they are the ones who have made this journey so much fun. Thanks go to Kim New, my former officemate, and Howie Cohl, my current officemate, for many entertaining discussions, diversions, and laughs. Thanks also go to Patrick Motl, Paul Fisher, and Eric Barnes for much of the same. I also owe a large debt of gratitude to my predecessor, John Woodward. His initial incarnation of the hydrocode provided the basic tool for my research and his initial aid in programming parallel computers has stood me in good stead over the years. The other senior member of our group, Juhan Frank, also deserves mine and
the group's gratitude, by keeping us focused on astrophysics when we tend to wander into purely computational realms.

This work has been supported, in part, by funding from the U.S. National Science Foundation through grant AST-9528424, from the Louisiana Board of Regents, LEQSF, under agreement NASA/LSU-(91-96)-01 and NASA/LaSPACE under grant NGT5-40335. This work was also supported, in part, by grants of high-performance-computing time at the NAVOCEANO DoD Major Shared Resource Center in Stennis, MS, through the PET program and at the San Diego Supercomputer Center. Thanks also go to the now defunct MasPar corporation for providing a parallel machine that was a joy to work on.
# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS** ............................................................... ii

**LIST OF TABLES** ................................................................. vi

**LIST OF FIGURES** ............................................................... vii

**ABSTRACT** .............................................................................. ix

## 1. INTRODUCTION
1.1 Star Formation Methods and Observations ................................................ 1
1.2 Review of Analytical Equilibrium Models ................................................... 6
1.3 Fission Hypothesis ...................................................................................... 11
1.4 Recent Related Work .................................................................................. 16
1.5 This Work .................................................................................................. 20

## 2. NUMERICAL METHODS
2.1 Self-Consistent-Field Technique .................................................................. 23
2.2 Finite Difference Hydrodynamics Code ....................................................... 25
2.3 Heterogeneous Computing Environment ..................................................... 28
2.4 Measuring Virial Balance ............................................................................ 30

## 3. INITIAL MODELS ......................................................................... 34

## 4. ADIABATIC, DYNAMICAL EVOLUTION ......................................... 40
4.1 Initial Development of Nonaxisymmetric Structure ....................................... 42
4.2 Nonlinear Phase .......................................................................................... 43
4.3 Steady-State Behavior ................................................................................ 49
4.3.1 Steady-State Evolution .......................................................................... 50
4.3.2 Similarities to Riemann Ellipsoids .......................................................... 60
4.3.3 Departure from Analytical Compressible Ellipsoids ............................... 62
4.4 Relationship to Physical Systems .................................................................. 63

## 5. COOLING EVOLUTION AND THE FORMATION OF A BINARY .......... 67
5.1 Cooling Paradigm ......................................................................................... 67
5.2 Cooling Simulation ....................................................................................... 70
5.3 Comparisons with Observations .................................................................. 81

## 6. CONCLUSION .................................................................................. 85

**REFERENCES** .............................................................................. 91

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX A: HYDROCODE ............................ 94
APPENDIX B: INPUT FILES ............................ 180
VITA ...................................................... 181
**LIST OF TABLES**

3.1 Initial Model Parameters .................................................. 37
4.1 Model A Parameters ......................................................... 51
4.2 Model B Parameters ......................................................... 52
4.3 Protostellar Systems ........................................................ 65
4.4 Compact Object Systems ................................................. 66
5.1 Model A Parameters ......................................................... 75
5.2 Protobinary Systems ....................................................... 83
# LIST OF FIGURES

1.1  The axis ratio plane for ellipsoidal equilibrium states. .................. 8

3.1  Specific angular momentum $j$ as a function of cylindrical enclosed mass $M_c$ for Models A and B at time $t = 0$. .................. 36

3.2  Same as Fig. 1.1 with the added triangles and squares representing Model A and Model B, respectively. .................. 39

4.1  Plot of global energy ratios as a function of time for Model A. ...... 44

4.2  Same as Fig. 4.1 for Model B. .................................................. 45

4.3  Time evolution of density contours along with vectors representing the momenta in the equatorial plane of Models A and B. .... 48

4.4  Equatorial density contours for Model A at the beginning and end of the steady-state phase. ........................................ 54

4.5  Same as Fig. 4.4 for Model B. .................................................. 55

4.6  Equatorial momenta for Model A at the beginning and end of the steady-state phase. ........................................ 57

4.7  Same as Fig. 4.6 for Model B. .................................................. 58

4.8  $\Phi_{eff}$ surface in the equatorial plane of Model B at the beginning of the steady-state phase. ........................................ 59

5.1  Plot of $T/|W|$, $T_{rot}/|W|$, and $\Pi/|W|$ as a function of time for the cooling simulation. ........................................ 71

5.2  Plot of radial position of the density maximum as a function of time for the cooling simulation. ........................................ 74

5.3  Same as Fig. 4.4 for Model A in the ellipsoidal and binary states of the binary instability oscillation. ........................................ 77

5.4  Same as Fig. 4.6 for Model A in the ellipsoidal and binary states of the binary instability oscillation. ........................................ 78

5.5  $\Phi_{eff}$ surface in the equatorial plane of Model A in the ellipsoidal and binary states of the binary instability oscillation. .... 80
6.1 Plot of $T_{\text{rot}}/|W|$ and $\Pi/|W|$ as a function of time for the complete evolution of Model A ................................................................. 86

6.2 Image of three-dimensional isodensity surfaces of the binary state of the binary instability oscillation ........................................... 88
ABSTRACT

Although we have a general understanding of how stars form, and there are accepted theories that explain the formation of long-period binaries, we do not yet understand how short period binaries form. Here we present simulations that clearly demonstrate how such systems may form naturally from dense interstellar gas clouds.

First, we present two models of compressible, self-gravitating fluid configurations with bar-like structures and supersonic internal motions. Both models have been constructed via dynamical simulations that have started from initially axisymmetric, rapidly rotating polytropes that were known to be dynamically unstable toward the development of a bar-like or two-armed spiral structure. The two initial models differed mainly in their initial angular momentum distributions. In each case, the nonlinear development of the dynamical instability results in the formation of a bar-like configuration that is spinning with a well-defined pattern speed. By all accounts, these models appear to be compressible analogs of Riemann ellipsoids. Our final "steady-state" configurations appear to be dynamically stable and include a mild standing shock front.

We have allowed one of these dynamically stable, triaxial configurations to cool slowly and have continually followed its dynamical evolution. A "binary instability," results after reducing the mean pressure of the configuration to \( \sim 50\% \) of its original value. The instability appears as an oscillation between two configurations: One that resembles a common envelope binary
system with circulation around the two local density maxima, and the other
that appears to be an ellipsoidal configuration with density maxima near
the center. Unfortunately, as the model cools, it continues to contract and
becomes less well resolved in our numerical grid. Hence, we have not been
able to follow this instability to its ultimate fate. However, the strength and
nature of the instability lead us to conclude that fission will be the outcome.
This work provides the strongest evidence, to date, that short period binary
stars form in a very natural way through a fission instability, as proposed by
Lebovitz (1987), that fission is the only possible outcome.
1. INTRODUCTION

1.1 Star Formation Methods and Observations

From observational surveys of the local stellar population of our Galaxy, we know that "binary formation is the primary branch of the star-formation process" (Mathieu 1994). More than fifty percent of all stars are formed in binary systems with orbital periods ranging from days to hundreds of thousands of years. As shown, for example in Figure 1 of Mathieu (1994), this distribution of binary systems peaks at a period of a few hundred years corresponding to a separation that is the approximate size of our solar system, — that is, about 50 astronomical units (AU), where one AU ($1.5 \times 10^{13}$ cm) is the distance from the earth to the sun. Why and how do stars preferentially form in pairs? This is the fundamental question that motivates the research described here.

In a general sense, we understand that stars form from the gravitational collapse of large, slowly rotating gas clouds (Larson 1973 and Shu, Adams, \& Lizano 1987). Initially, the collapse is dynamical and occurs on a free-fall timescale which, for clouds containing on the order of one solar mass ($1 \, M_\odot$) of hydrogen gas, is approximately a few times $10^4$ years. Although it is possible for clouds to fragment as they collapse dynamically, binaries produced during this phase of a protostellar cloud's evolution will have relatively long orbital periods — on the order of hundreds to hundreds of thousands of years. For example, recent hydrodynamical simulations performed by Burkert \& Bodenheimer (1993, 1996) and Truelove et al. (1997,1998) that follow...
the collapse of a 1 M\(_\odot\) cloud show that a binary system forms with a separation on the order of 500 AU with an orbital period of 10,000 years (Truelove et al. 1998).

This relatively violent, dynamical phase of collapse occurs when the density of an interstellar gas cloud climbs above \(\sim 10^{-19}\text{g cm}^{-3}\) and proceeds isothermally with a temperature \(\sim 10^6\text{ K}\) until the density reaches \(10^{-13}\text{g cm}^{-3}\) (Shu et al. 1987). At higher densities, the gas is unable to cool efficiently and the collapse is halted by the effects of thermal pressure. This limitation prevents protostellar gas clouds from collapsing further on a dynamical timescale. It also limits the "direct fragmentation" process to creating binaries or other multiple systems with relatively large separations and relatively long periods, as mentioned above.

Also, it is important to appreciate that whether a cloud remains a single object or fragments directly into two pieces (forming a binary system) or into a larger number of pieces (forming a multiple star system) during this dynamical phase of collapse is a strong function of the type of perturbation that was present in the cloud at the onset of collapse. For example, the clouds modeled by Truelove et al. (1998) broke into two pieces because they began the simulations by superposing \(\cos(2\phi)\) perturbations of amplitudes 10% and 50% upon an otherwise axisymmetric gas cloud prior to collapse. This seed guaranteed the formation of a binary system. Meanwhile, Miyama, Hayashi, & Narita (1984) modeled this phase of collapse beginning with random density perturbations of less than 5%. Their simulations produced
systems with multiple numbers of fragments. Hence, the direct fragmentation process is very sensitive to initial conditions.

If there is no direct fragmentation during the dynamical collapse phase, the self-gravitating gas cloud settles into an equilibrium configuration that resembles an oblate spheroid rotating about its short axis with a mean density \( \bar{\rho} \geq 10^{-13}\text{g cm}^{-3} \) and a size \( R \lesssim 10^2 \text{ AU} \). The cloud is supported against further dynamical collapse by a combination of thermal pressure and rotation, but cloud contraction generally continues on a longer (thermal) timescale as each cloud continues to slowly lose thermal energy via radiation processes. Still conserving angular momentum throughout this slow contraction phase, the cloud rotates faster and becomes more and more flattened. At a sufficiently rapid rate of rotation, it becomes energetically favorable for a self-gravitating gas cloud to deform into a nonaxisymmetric configuration. Analytical studies of incompressible fluids suggest that initially the energetically preferred structure will be ellipsoidal in shape. (See §1.1 below.) It has also been hypothesized that as this kind of nonaxisymmetric structure continues to cool, it will suffer an additional instability that causes it to spontaneously fission into two centrally condensed clouds. (See §1.2 below.) If such a “fission instability” indeed arises, it could explain why the vast majority of stars form in pairs.

Binary systems are not the only option besides single star systems. Stars also exist in larger multiple systems. As was first pointed out by Bodenheimer (1978), these systems are best described as hierarchical binary systems where the length scales between the stars are different enough that a quadruple sys-
tem, for example, can be modeled as a multiple binary. Hierarchical systems are a natural result of the fragmentation process described above, but may also result from a two step process. Since the direct fragmentation process produces widely separated fragments before the collapse phase ends, each fragment may be unstable to fission after it comes into hydrodynamical equilibrium. This two-step process would easily explain observations of multiple hierarchical systems.

Capture is another method suggested in the past for binary and multiple system formation. However, dynamical studies have shown that capture would be such a rare event that this method now seems improbable (Heggie 1975 and Bodenheimer et al. 1993). Also, observations of pre-main-sequence star systems show that the frequency of binaries exceeds or is at least as great as the frequency of binaries observed among main sequence1 star systems (Mathieu 1994). This implies that binaries are formed in the protostellar phase before nuclear fusion begins and the gas cloud becomes a star.

Observing systems in this protostellar phase is a difficult task since the central core has not ignited yet and the central object is still embedded in the dusty molecular cloud from which it collapsed. Once there is ignition, the stellar wind blows away most of the obscuring dust and debris and the star may be observed directly.

Complicating matters even further is the small size of the central protostellar core. This inner region may be as small as 1 AU across and therefore,

1A star reaches the main sequence when nuclear fusion begins in its core. A pre-main-sequence star is a star that is still contracting towards the main sequence but has not reached densities sufficient for nuclear fusion to occur.
in all but the nearest interstellar gas clouds, subtends an angle on the sky that is too small to be resolved observationally. Current observations using the Wide Field Planetary Camera on the Hubble Space Telescope, are able to resolve objects approximately the size of the solar system in the Orion Nebula, a nearby region of star formation. Observations of this region have produced striking pictures of "proplyds" (O'Dell et al. 1993), where "proplyd" is an abbreviation for "protoplanetary disk." Proplyds are best described as disks of circumstellar material surrounding a collapsing protostellar core. These proplyds range in size from about 50-200 AU or about 1-4 times the size of our solar system. We are able to resolve these objects with the Hubble Space Telescope because the Orion nebula is relatively nearby at only 500 parsecs (1 parsec = 206,265 AU).

Another promising method of observation is the use of millimeter- and submillimeter-wavelength interferometers. Interferometers can provide high spatial resolution observations, usually much better than any optical observations. Also, the surrounding dusty envelope from which the protostellar system is formed is mostly transparent to millimeter- and submillimeter-wavelengths detected by the interferometer arrays. In recent years, there has been an explosion of observations with this technique. (See Lay et al. [1997], Testi & Sargent [1998], and Hogerheijde et al. [1999].) These observations are able to limit the size of the many protostellar cores to ~50-100 AU, which is approximately the extent of the protostellar systems we are modelling.

Over the next decade, as these observational techniques improve and new telescopes come on line, direct observations of star formation will finally
be possible. Current dynamical simulation results may then be compared in detail to observations of the gas density and velocity maps of the gas flow. The simulations discussed here and elsewhere will provide observers important clues as to the interpretation of the data. This interaction will greatly improve our understanding of the general processes by which single stars, binary systems, and larger multiple systems form. For these reasons, we believe the results discussed here will be of great value in the next decade as compact protostellar systems are probed more deeply than ever before.

Even before we are able to make direct, detailed comparisons between model simulations and observations, it is critically important to use modelling techniques to examine the general validity of physical mechanisms that have been proposed to explain how multiple star systems form. Most importantly, until now the fission hypothesis has remained untested, because it is not amenable to an analytical solution and, to be tested numerically, requires fully three-dimensional simulations with relatively high spatial resolution, which heretofore have not been possible. Also, the analytical studies have, in most cases, been limited to incompressible fluids whereas in reality, protostellar gas clouds are very compressible. We do both here, for the first time. That is, we construct compressible analogs of the analytically known incompressible ellipsoidal figures of equilibrium (Chandrasekhar 1969) and we present evidence that fission can occur spontaneously.

1.2 Review of Analytical Equilibrium Models

Before we attempt to describe how these compressible, nonaxisymmetric models have been constructed or how their properties compare in detail to
the well-known incompressible ellipsoids that were first studied by Riemann, it will be useful to review what is known about the structural and stability properties of the, now classic, analytical models.

By far the most complete review of the structure and stability of rotating, incompressible, self-gravitating "ellipsoidal figures of equilibrium," has been by Chandrasekhar (1969). As Chandrasekhar points out, this elegant analytically tractable problem has interested mathematicians of the likes of Jacobi, Dedekind, and Riemann, and dates back over 250 years to the work of Maclaurin (1742). Here we present a much briefer overview of these classical analytical works, referring the interested reader to Chandrasekhar's (1969) review for more details and for an exhaustive list of references on this topic.

The two-dimensional parameter space identified in Figure 1.1 can be used to identify individual, or entire sequences of incompressible equilibrium models, that have been studied extensively through analytical techniques if we let \( a > b > c \) represent the three principal axes of an ellipsoid. For example, the point marked \( M \) in the top, right-hand corner of Figure 1.1 identifies a sphere, while the right-hand edge of the figure (line segment \( MT \)) defines the sequence of uniformly rotating, oblate "Maclaurin" spheroids. As one moves down the line segment marked \( MT \), the Maclaurin spheroids become more and more flattened and exhibit a higher ratio of rotational kinetic to gravitational potential energy, \( T_{rot}/|W| \) (see Eq. [2.21] in §2.3 for a mathematical definition of \( T_{rot} \)).

A tensor virial analysis at the point marked \( S \) in Figure 1.1 \( (T_{rot}/|W| = 0.14) \), Maclaurin spheroids first become unstable to an ellipsoidal deforma-
Figure 1.1: The axis ratio plane for ellipsoidal equilibrium states. The ratio $b/a$ is plotted along the $x$ axis and $c/a$ along the $y$ axis, where $a$ is defined as the length of the major axis, $b$ as the length of the axis perpendicular to $a$ and the axis of rotation, and $c$ as the length of the axis of rotation. The Maclaurin spheroid equilibrium sequence lies upon the line $MT$, with the point $M$ representing a sphere and the point $T$ representing an infinitesimally thin disk. Points $S$ and $D$ mark the points of secular and dynamical instability respectively for Maclaurin spheroids. Maclaurin spheroids on the line segment $SD$ are secularly unstable. Maclaurin spheroids on the line segment $DT$ are dynamically unstable. The curve $SO$ traces the equilibrium sequence for Jacobi and Dedekind ellipsoids. The curve $DO$ traces the equilibrium sequence for Riemann ellipsoids with $f = 2$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
tion. As Christodoulou et al. (1995a) have demonstrated, the point marked S also is significant when ellipsoidal systems are examined in terms of their free-energy. That is, for a given total mass and total angular momentum, at points below S along the Maclaurin sequence the axisymmetric configuration is no longer the lowest energy state. Instead, as was first discovered by Jacobi, there is a triaxial equilibrium configuration of the same mass and angular momentum that is of lower free-energy. The curve SO that bifurcates from the Maclaurin sequence at point S in Figure 1.1 identifies the sequence of lower-energy, equilibrium ellipsoids that was discovered by Jacobi. Like the objects along the Maclaurin sequence from which it bifurcates, Jacobi ellipsoids are rigid rotators. Hence, from an inertial frame of reference a Jacobi ellipsoid appears to be spinning about its shortest axis as a solid body with a well-defined angular frequency $\Omega_0$. Alternatively, as viewed from a frame that is rotating with an angular frequency $\Omega$, Jacobi ellipsoids exhibit no internal motion.

Interestingly enough, as was first realized by Dedekind, the curve marked SO in Figure 1.1 also identifies a second sequence of equilibrium ellipsoidal models — one that is formally “adjoint” to the Jacobi sequence. In stark contrast to Jacobi ellipsoids, the ellipsoidal surface of each Dedekind configuration is perfectly stationary in inertial space as the model’s rotational energy is stored completely in internal motions. Hence, as viewed from a frame that is rotating with the ellipsoidal figure (in this case, $\Omega_0 = 0$), each fluid element in a Dedekind ellipsoid moves along an elliptical path with
a velocity \( \vec{u} \) in inertial space that is a linear function of its instantaneous coordinate position and the configuration as a whole has a nonzero vorticity.

As was ultimately understood by Riemann, the Jacobi and Dedekind objects are simply examples of an entire range of possible equilibrium ellipsoidal configurations that are available to rotating, self-gravitating, incompressible fluids. Riemann was able to identify a whole host of equilibrium sequences besides the Jacobi/Dedekind sequence that branch off of the Maclaurin spheroid sequence at points between \( \text{M} \) and \( \text{D} \) in Figure 1.1. (The curve marked \( \text{DO} \) in Figure 1.1 identifies one such sequence.) In general, Riemann configurations can be thought of as superpositions of Jacobi and Dedekind configurations: as viewed from an inertial reference frame the surface of a Riemann ellipsoid rotates with a well-defined pattern frequency \( \Omega_0 \), and from a frame that is rotating with the surface at this angular frequency, the configuration displays nonzero vorticity.

As we extend our discussion of nonaxisymmetric equilibrium structures to compressible gas configurations with nontrivial internal motions, it will be useful to note that Riemann configurations often are identified in terms of the dimensionless ratio

\[
f \equiv \zeta_z / \Omega_0, \tag{1.1}\]

where, as viewed from a cylindrical coordinate frame that is rotating with the figure, the \( z \)-component of the fluid vorticity,

\[
\zeta_z \equiv \hat{k} \cdot (\nabla \times \vec{u}), \tag{1.2}
\]
and the fluid velocity in the rotating frame,

\[ \bar{u} \equiv \vec{v} - \hat{k}\Omega_0 R. \] (1.3)

(Here, and throughout the remainder of this dissertation, we will assume that the z-axis is aligned with the axis about which the ellipsoidal figure spins and that this axis, in turn, is aligned with the shortest c axis of the ellipsoidal figure.) In terms of this dimensionless vorticity parameter, models along the Jacobi sequence have \( f = 0 \) and the curve labeled DO in Figure 1.1 identifies Riemann configurations with \( f = 2.0 \).

1.3 Fission Hypothesis

As discussed briefly in §1.1, when a protostellar gas cloud collapses from molecular cloud densities \( \rho \sim 10^{-18} - 10^{-20} \text{g cm}^{-3} \) toward a configuration that is dense enough and hot enough to promote nuclear fusion, it is initially able to cool itself very efficiently (Shu et al. 1987). As a result, the cloud's initial collapse is expected to be fairly violent, with significant changes in the cloud's properties occurring on a dynamical timescale, \( \tau_{\text{dyn}} \sim [G \rho]^{-1/2} \). At densities \( \rho \gtrsim 10^{-13} \text{g cm}^{-3} \), however, a typical cloud will become optically thick to its primary cooling radiation. At this point, the gravitational collapse phase ends and the cloud enters a phase of contraction which proceeds on a Kelvin-Helmholtz (thermal) timescale which, for a spherical, 1 M\(_\odot\) protostellar cloud, is approximately 10 million years — many orders of magnitude greater than the dynamical timescale of the collapse.

More realistically, however, we must consider that most protostellar gas clouds will contain a significant amount of angular momentum initially and
that they will become more and more rotationally flattened as they contract toward the main sequence. In a qualitative sense at least, the quasistatic contraction of a rotating, protostellar gas cloud can be presented in the context of a diagram such as Figure 1.1 cf., Durisen & Tohline 1985; Lebovitz 1987). A gas cloud that conserves its mass and angular momentum will evolve slowly down the axisymmetric Maclaurin sequence, from a point near M in Figure 1.1 toward the point marked S. However, as was pointed out in §1.2, at points below S in Figure 1.1 there exist ellipsoidal configurations that have lower total energy than the Maclaurin spheroids, so there is an opportunity for the contracting gas cloud to naturally evolve from an axisymmetric shape into a fully triaxial configuration.

As it turns out, though, at point S in Figure 1.1 the Jacobi ellipsoid exhibits a different radial distribution of angular momentum than does its axisymmetric counterpart on the Maclaurin sequence. Hence, some form of dissipation is required in order to drive evolution from the Maclaurin sequence to the Jacobi sequence at point S in Figure 1.1, and models along the Maclaurin sequence are understood to be only secularly unstable toward the development of an ellipsoidal deformation at (or just beyond) this point. In situations where ordinary fluid viscosity acts as an effective dissipation mechanism and facilitates the redistribution of angular momentum on a timescale that is short compared to the Kelvin-Helmholtz timescale, the cloud will be susceptible to the development of an ellipsoidal deformation at point S along the Maclaurin spheroid sequence. Upon further contraction, the cloud should evolve through a sequence of more and more distorted ellipsoids (the axis ra-
tio $b/a$ should steadily drop from its value of 1 at point $S$) and, schematically at least, evolution should proceed to the left along curve $SO$ in Figure 1.1.

Analytical stability analyses also have indicated that if a rotating cloud evolves far enough to the left along the Jacobi sequence, it can become susceptible to the development of higher-order surface distortions. As Chandrasekhar (1969) has illustrated (see especially his Fig. 15), these higher-order modes can lead, for example, to pear-shaped (third-order) or dumbbell-shaped (fourth-order) surface deformations. Although it has not been possible through analytical techniques to construct equilibrium models with finite-amplitude distortions of these types, recognition that sufficiently elongated ellipsoids are unstable toward the development of such distortions led early investigators to suggest that further slow contraction along equilibrium sequences might lead to fission and thereby explain in a natural way how binary stars form.

This general idea has to some extent been strengthened by the relatively recent work of Eriguchi & Hachisu (1984). Using numerical techniques, Eriguchi & Hachisu have been able to construct equilibrium models of uniformly rotating, incompressible fluids with pear-shaped and dumbbell-shaped configurations, and they have shown furthermore that the dumbbell sequence smoothly connects to an equal-mass binary sequence. (Note, however, that it is still not clear whether the models along these numerically generated sequences are of lower free-energy than models along the Jacobi sequence from which they bifurcate [Christodoulou et al. 1995].) The idea that a rapidly rotating gas cloud might contract quasistatically along a sequence of ellip-
soids and then, in a very natural way, onto a dumbbell-binary sequence has come to be known as the fission hypothesis of binary star formation (Durisen & Tohline 1985).

Unfortunately, as Lebovitz (1987) has emphasized, this classical fission hypothesis has one particularly serious flaw. A cloud that evolves from point S toward point O along the Jacobi sequence will encounter a bifurcation to the pear-mode sequence before it reaches the dumbbell sequence. Since linear stability analyses have shown that Jacobi ellipsoids are dynamically unstable toward the development of this third-order surface distortion, quasistatic evolution will almost certainly terminate before a viscous cloud reaches the dumbbell-binary sequence. Hence, it seems unlikely that such an evolution will end in the formation of a binary system.

However, Lebovitz (1987) has offered an alternate scenario which we will refer to as the modified fission hypothesis. If protostellar gas clouds are relatively inviscid — that is, if the Kelvin-Helmholtz timescale that governs their rate of contraction is short compared to the time it would take for naturally occurring viscous processes to effect a significant redistribution of angular momentum within the cloud — they will not be susceptible to the secular instability that arises at point S along the Maclaurin sequence and, therefore, will not make a transition to the Jacobi sequence. Instead, contraction should proceed along the axisymmetric sequence to a point marked D in Figure 1.1 ($T_{rot}/|W| = \beta_D \equiv 0.2738$) where linear stability analyses have shown that Maclaurin spheroids first become dynamically unstable to an ellipsoidal deformation. (See also Christodoulou et al. 1995a for a dis-
discussion of the significance of point D in terms of the system’s free-energy.)
As Lebovitz has argued, it is at this point that an inviscid cloud will begin to deform significantly into an ellipsoidal configuration. Hence, the cloud’s subsequent evolution should proceed along curve DO as illustrated in Figure 1.1 — that is, along the \( f = 2.0 \) Riemann sequence — rather than along the Jacobi sequence. As Lebovitz has argued, this is a much more promising route to fission because contracting clouds that evolve along this Riemann sequence will become susceptible to a fourth-order (dumbbell-shaped) surface deformation before they encounter a third-order (pear-shaped) deformation.

In order to ascertain whether or not Lebovitz’s modified fission hypothesis plays a significant role in defining the process by which multiple star systems form, it will be necessary to follow the nonlinear development of nonaxisymmetric structures in dynamically evolving, self-gravitating fluid systems. At the same time it will be important to consider how the relative stability of such systems is altered when the effects of gas compressibility are included because realistic protostellar gas clouds are not represented well by an incompressible equation of state. This is a very tall order considering the fact that, to date, relatively little progress has been made toward developing techniques that will even permit the construction of equilibrium models of compressible ellipsoidal figures. Progress along these lines has been hampered largely by the fact that equilibrium figures of this type must include nontrivial internal flows. As a result, the techniques that have been developed to construct incompressible ellipsoidal figures of equilibrium or axisymmetric models with a compressible equation of state become dysfunctional. As subsequent chap-
ters of this dissertation will make clear, we are now in a position to test the viability of Lebovitz’s modified fission hypothesis because, utilizing numerical hydrodynamic techniques, we are able to construct dynamically stable triaxial models with nontrivial internal flows that appear to be compressible analogs of the Riemann ellipsoids. (By “triaxial,” we mean that the object has three principal axes of unequal length, but it is not necessarily a perfect ellipsoid.)

1.4 Recent Related Work

While our interest in constructing compressible analogs of the Riemann ellipsoids is motivated primarily by our desire to understand why stars preferentially form in pairs, interest in such models extends to groups who are investigating the structure of rapidly rotating compact objects (like white dwarfs and neutron stars) and to groups who are attempting to build self-consistent models of triaxial galaxies. The most extensive work along these lines in recent years has been that of Lai, Rasio, & Shapiro (1993; hereafter, LRS). Utilizing an energy variational principle, LRS constructed a variety of “approximate” equilibrium models of rapidly rotating spheroids, ellipsoids, and close binaries having polytropic equations of state (see Eq.[2.5] below). LRS imposed the following principal constraints on their approximate equilibrium models: (1) Isodensity surfaces form a set of concentric ellipsoids; (2) \( \vec{u} \cdot \nabla \rho = 0 \), that is, the velocity vector at any point in the fluid must be tangent to the isodensity surface passing through that point; and (3) each model exhibits either uniform angular velocity or uniform vorticity with \( \hat{\mathbf{k}} \times \vec{\sigma} = 0 \) everywhere. Unfortunately, with these constraints in place, LRS were only
able to construct analogs of Riemann ellipsoids that were either very nearly incompressible or very nearly irrotational.

In an attempt to construct compressible analogs of Dedekind ellipsoids (i.e., triaxial configurations that are stationary as viewed in the inertial frame), Chambat (1994) imposed a similar set of constraints to those imposed by LRS, but looked for equilibria by means of the hydrodynamic equations. Chambat concluded that no such equilibrium configurations exist. In an investigation very similar to Chambat’s, but extended to include ellipsoidal configurations that exhibit a nonzero pattern frequency $\Omega_0$ as viewed from an inertial frame, Filippi, Ruffini & Sepulveda (1996) have concluded that, when using constraints of the type imposed by LRS, it is not possible to construct any “heterogeneous generalized Riemann ellipsoid,” even if the velocity is allowed to be a quadratic function of the coordinate.

These studies seem to suggest that the variational principle that LRS used to identify “approximate” equilibria provides a necessary but not sufficient condition for defining structures that can be recognized as compressible analogs of the Riemann ellipsoids. Unfortunately the Chambat (1994) and Filippi et al. (1996) studies leave the reader wondering whether it is possible at all for compressible analogs of the Riemann ellipsoids to exist in nature.

Three lines of evidence have led us to suspect that such configurations do exist and that the results reported by Chambat (1994) and Filippi et al. (1996) stem mostly from the fact that the constraints imposed on the equilibrium models by these authors and by LRS have been too restrictive. First, while investigating the nonlinear development of the triaxial instability that
arises in initially axisymmetric, rapidly rotating \( T/|W| > \beta_d \) polytropes some years ago, Durisen et al. (1986) and Williams & Tohline (1988) found that an initial phase of spiral arm development and rapid redistribution of angular momentum led to the development of a centrally condensed object of lower \( T/|W| \) that was surrounded by a disk of ejected debris and that, more often than not, the central object was triaxial in shape. Little was said about this central triaxial figure at the time but, according to the reported simulations, the object exhibited a coherent pattern frequency \( \Omega_0 \), and it was long-lived enough to establish a well-defined resonant corotation radius in its surrounding disk. Second, Uryū & Eriguchi (1996) have demonstrated that Dedekind-like configurations can be constructed from a compressible fluid by allowing the azimuthal component of the internal flow velocity to vary with height (i.e., by relaxing LRS constraint 3, as defined above) and by not forcing the velocity vectors to everywhere lie tangent to isodensity surfaces (i.e., by relaxing LRS constraint 2, as defined above.) Third, Andalib (1998) recently has developed a self-consistent-field technique that permits the construction of two-dimensional, self-gravitating, polytropic configurations with nonaxisymmetric structures and nontrivial internal flows. (By “two-dimensional” we mean that, in the vertical direction, Andalib’s configurations are either infinite in extent or infinitesimally thin.) With this new technique, Andalib has been able to construct ellipsoidal, boxy, dumbbell, and “binary” structures that exhibit a wide variety of coherent pattern frequencies \( \Omega_0 \) and nontrivial internal flows. Although it is not yet clear how Andalib’s technique can be extended to permit the construction of config-
urations that have a finite vertical extent, his results strongly suggest that compressible analogs of the Riemann ellipsoids do exist. We stress, however, that it was necessary for Andalib (1998) to relax all three of the LRS constraints (as defined above) in order to be able to construct such configurations.

Because Andalib's (1998) work has been important in motivating our present study, we should explain a bit more fully how he specified and constrained the internal flow of his equilibrium nonaxisymmetric figures. Following Papaloizou & Savonije (1991), Andalib defines vortensity $\tilde{\xi}$ as the reciprocal of the load (Lynden-Bell & Katz 1982), that is,

$$\tilde{\xi} \equiv \frac{\zeta + 2\Omega_0}{\Sigma}, \quad (1.4)$$

where $\zeta \equiv \nabla \times \vec{u}$ is the fluid vorticity, $\Omega_0$ is the angular velocity of the rotating frame, and $\Sigma$ is the surface density. By demanding that each fluid configuration exhibit uniform vortensity (rather than uniform vorticity), or that throughout the configuration the vortensity should be at worst a linear function of a pseudo stream function, Andalib was able to cast the equation of motion (2.2) into a steady-state form which, at least in a two-dimensional system, is amenable to solution via Hachisu's (1986) iterative self-consistent-field technique. The wide range of nonaxisymmetric configurations Andalib has been able to create suggests the constraint of uniform vortensity to be meaningful. Although we have not yet been able to create triaxial equilibrium models with this constraint, we have been able to create an axisymmetric equilibrium model with uniform vortensity, by substituting the vertically

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
integrated column density for $\Sigma$. This equilibrium model is described in detail in §3.1.

While relevant, these compressible models suggest only the existence of triaxial compressible objects, not necessarily the validity of the fission hypothesis. There is other evidence that has led us to believe that the path described by the fission hypothesis will lead to binary systems. New & Tohline (1997) recently have been able to create equilibrium sequences of equal-mass binary systems in synchronous rotation with polytropic and white dwarf equations of state. (See Eq. [2.5] in Chapter 2 for the definition of a polytropic equation of state.) They also tested the dynamical stability of these sequences and discovered that, for polytropic sequences having indices $n \geq 1$, the systems remained dynamically stable from widely separated binary configurations to dumbbell configurations. Hence, if the cooling process for an $n = 3/2$ triaxial, polytropic model leads to a binary or dumbbell configuration, neglecting internal motions, the configuration should be dynamically stable.

1.5 This Work

In this dissertation, we present detailed numerical models of two rapidly rotating, triaxial self-gravitating configurations that appear to be compressible analogs of Riemann ellipsoids. Via nonlinear hydrodynamic simulations we demonstrate that both models are dynamically stable and that their structures are, to a high degree of approximation, steady-state; this despite the fact that both models exhibit fluid velocities that exceed the local sound
speed where the fluid streams along the length of the "bar," as viewed from a frame that is rotating with the overall pattern speed of the bar.

With these steady-state triaxial models of compressible, self-gravitating gas clouds in hand, we also investigate Lebovitz's modified fission hypothesis as outlined in Tohline, Cazes, & Cohl (1998). By allowing the gas to cool uniformly in our hydrodynamic simulation, we witness the onset of a global instability which exhibits key characteristics of a common-envelope binary system. Although present computational capabilities prevent us from being able to follow this simulation all the way to the formation of a steady-state binary system, the strength and nature of the instability lead us to believe that spontaneous fission of the cloud is the most probable outcome. Our work offers the strongest evidence to date that short period binary systems do form via Lebovitz's (1987) modified fission hypothesis.

Throughout this dissertation we will emphasize the relevance that our computational models have to studies of protostellar clouds and the formation of binary star systems because it is in this context that our studies have been primarily motivated. However, our models also strengthen the idea that, during the late stages of stellar evolution, relatively long-lived non-axisymmetric structures may be formed that give rise to measurable levels of (continuous, rather than burst) gravitational radiation and they should prove useful in future attempts to understand the structure and evolution of long-lived gaseous bars in galaxies.
2. NUMERICAL METHODS

Our simulations involve the solution of the following set of equations, which govern the structure and adiabatic evolution of a self-gravitating, non-relativistic inviscid fluid as viewed from a frame of reference that is rotating uniformly with angular velocity $\Omega_0$: The continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0; \quad (2.1)$$

the equation of motion,

$$\frac{\partial \vec{u}}{\partial t} + (\vec{\zeta} + 2k\Omega_0) \times \vec{u} = -\nabla(H + \Phi + \frac{1}{2}u^2 - \frac{1}{2}\Omega_0^2R^2); \quad (2.2)$$

an adiabatic representation of the 1st law of thermodynamics,

$$\frac{\partial (\epsilon \rho)^{1/\gamma}}{\partial t} + \nabla \cdot [(\epsilon \rho)^{1/\gamma} \vec{u}] = 0; \quad (2.3)$$

and the Poisson equation,

$$\nabla^2 \Phi = 4\pi G\rho. \quad (2.4)$$

These four PDEs must be supplemented by an equation of state that specifies an appropriate relationship between the mass density $\rho$, the enthalpy $H$, and the specific internal energy $\epsilon$ of the gas. Among the variables in these equations that have not yet been defined, $G$ is the gravitational constant, $\Phi$ is the gravitational potential, and $\gamma$ is the effective ratio of specific heats.
defining the adiabat along which Lagrangian fluid elements evolve upon compression or expansion. The models presented in this dissertation have each been constructed initially using an \( n = 3/2 \) polytropic equation of state, that is the enthalpy is everywhere related to the density via the relation,

\[
H = (n + 1)K\rho^{1/n} = \frac{5}{2}K\rho^{5/3},
\]

and the models have been evolved adiabatically assuming an effective ratio of specific heats \( \gamma = 5/3 \) and an ideal gas relationship between the various state variables. Hence the models remain homentropic throughout each illustrated evolution with the selected value of the polytropic constant \( K \) specifying every fluid element's specific entropy. Also, the choice of a polytropic equation of state allows us to define the pressure \( P \) purely as a function of the density via the expression,

\[
P = K\rho^{1+1/n} = K\rho^{5/3},
\]

where in the last step we have set \( n = 3/2 \).

2.1 Self-Consistent-Field Technique

We have used Hachisu's (1986) self-consistent-field (hereafter, HSCF) technique to construct rapidly rotating, axisymmetric equilibrium configurations as initial models for each of our dynamical simulations. The HSCF technique is designed to construct steady-state models with arbitrarily flattened (or even toroidal-shaped) gas distributions as long as the gas is constrained to execute circular motion about the symmetry axis of the equilibrium configuration, i.e., as long as in cylindrical coordinates \( \vec{v} = (0, v_\phi, 0) \). The equations
that govern the equilibrium properties of such structures are readily obtained from equations (2.1) - (2.4) by setting the partial time derivatives to zero and by realizing that, for any scalar variable \( Q \), \( \nabla \cdot (Q \vec{u}) = 0 \) when the fluid executes purely circular motion. Equations (2.1) and (2.3) are satisfied trivially, and from equation (2.2) we derive the condition:

\[
-\nabla (H + \Phi + \frac{1}{2}u^2 - \frac{1}{2}\Omega_0^2R^2) = (\zeta + 2k\Omega_0) \times \vec{u} \quad (2.7)
\]

\[
= -\hat{e}_R u_\phi (\zeta_x + 2\Omega_0). \quad (2.8)
\]

If we furthermore set \( \Omega_0 = 0 \) (i.e., view the model in an inertial frame of reference) and select a specific angular momentum distribution such that \( v_\phi \) is only a function of the cylindrical coordinate \( R \), we may write,

\[
\zeta_x = \frac{v_\phi}{R} + \frac{\partial v_\phi}{\partial R} = \frac{1}{R} \frac{\partial}{\partial R} (R v_\phi), \quad (2.9)
\]

and the condition for steady-state equilibrium becomes,

\[
\nabla (H + \Phi) = \hat{e}_R \frac{v_\phi^2}{R}. \quad (2.10)
\]

But when \( v_\phi \) is only a function of \( R \), we also can define a centrifugal potential \( \Psi(R) \) such that,

\[
\nabla (h_0^2 \Psi) = -\hat{e}_R \frac{v_\phi^2}{R}, \quad (2.11)
\]

where \( h_0^2 \) is an appropriate scaling constant. As a result, equation (2.10) takes the form,
\[ \nabla (H + \Phi + h_0^2 \Psi) = 0, \]  
\hspace{2cm} (2.12)

and we may conclude that, throughout the entire volume of the equilibrium configuration,

\[ H + \Phi + h_0^2 \Psi = C, \]  
\hspace{2cm} (2.13)

where \( C \) is a constant. As Hachisu (1986) has described in detail, it is this algebraic expression that the HSCF technique aims to satisfy simultaneously with a selected barotropic equation of state (2.5) and the Poisson equation (2.4) as it iterates toward a self-consistent, equilibrium configuration.

For this investigation, each initial axisymmetric model has been constructed on a cylindrical coordinate grid with \( 64 \times 64 \) uniformly spaced zones in the radial and vertical directions, and assuming reflection symmetry through the equatorial plane.

2.2 Finite-Difference Hydrodynamics Code

Our dynamical simulations have been performed using a three-dimensional, finite-difference technique to integrate the nonlinear fluid equations (2.1)-(2.3) forward in time in a fully self-consistent fashion. Implemented on a uniformly zoned, cylindrical coordinate grid and patterned after the ZEUS-2D code (Stone & Norman 1992), our "hydrocode" algorithm is designed to be second-order accurate in both time and space. The algorithm is designed to solve the general partial differential equation

\[ \frac{\partial Q}{\partial t} + \nabla \cdot (Q \vec{u}) = S_Q \]  
\hspace{2cm} (2.14)
where $Q$ is the physical quantity being advected and $S_Q$ is the source term. The scalar Equations (2.1) and (2.3) are already expressed in this form, where the $Q$ parameter is, respectively, the density $\rho$ and the specific entropy tracer $(\epsilon\rho)^{1/\gamma}$, and the source term for both equations is zero. The vector equation of motion (2.2), can be rearranged to fit the form above and gives the following three components,

$$\frac{\partial S}{\partial t} + \nabla \cdot (S \bar{u}) = -\rho \frac{\partial}{\partial R} (H + \Phi - \frac{1}{2} \Omega_0^2 R^2) + \frac{A^2}{\rho R^3} + \frac{2\Omega_0 A}{R}, \quad (2.15)$$

$$\frac{\partial T}{\partial t} + \nabla \cdot (T \bar{u}) = -\rho \frac{\partial}{\partial z} (H + \Phi), \quad (2.16)$$

$$\frac{\partial A}{\partial t} + \nabla \cdot (A \bar{u}) = -\rho \frac{\partial}{\partial \phi} (H + \Phi) - 2\Omega_0 RS; \quad (2.17)$$

where $S$ is the radial momentum density, $T$ is the vertical momentum density, and $A$ is the angular momentum density. The physical quantities $Q$ to be advected are now the momentum densities, $S$, $T$, and $A$; the source terms $S_Q$ are the forces per unit volume that appear on the right-side of the above equations. The hydrocode and its general implementation has been described in considerable detail by Woodward(1992), Woodward, Tohline, & Hachisu (1994), and New(1996).

In order to obtain a self-consistent gravitational field for the instantaneous mass density distribution at each integration time step, the Poisson equation (2.4) is solved implicitly using a two-dimensional ADI (alternating direction implicit) method (Cohl et al. 1997) in conjunction with a one-dimensional, discrete Fourier method. Initially, the value of the gravitational potential on the outer boundary of the cylindrical coordinate grid was de-
term ined via an expansion in spherical harmonics through \( l = 10, m = \pm 10 \).

Unfortunately, to execute expediently on a cylindrical grid, the boundary solver technique required that the radial and vertical extent of our numerical grid be equal. Because the axis ratio \( c/a \) of each of our physical models is at most 0.3 (see Chapter 3), the extra vertical space required by the boundary solver was mostly wasted. Recently, Cohl & Tohline (1999) have developed a new method to solve for the gravitational potential on a cylindrical grid boundary via an expansion in terms of half-integer degree Legendre functions of the second kind. This method allows the vertical size of the grid to differ from the radial size, thus allowing us to utilize a numerical grid that matches the vertical extent of our model. (For more details see Cohl 1999.)

Each of our dynamical simulations has been performed initially on a cylindrical coordinate grid with an effective resolution of \( 128 \times 256 \times 256 \) zones in the radial, vertical, and azimuthal directions, respectively. Later simulations utilizing the new boundary-solver technique for the Poisson equation were performed with an effective grid resolution of \( 128 \times 64 \times 256 \). The hydrocode has been run utilizing one of two defined symmetries: equatorial symmetry, or equatorial plus \( \pi \)-symmetry. When reflection symmetry through the equatorial plane is imposed, the number of required vertical zones is reduced by half. When \( \pi \)-symmetry is adopted, every physical variable \( f \) is assumed to exhibit a periodic symmetry in the azimuthal coordinate direction of the form \( f(R, Z, \phi) = f(R, Z, \phi + \pi) \), in which case the number of required azimuthal zones is also reduced from 256 to 128.
The hydrocode was originally written in MPF, a parallel programming language for LSU's MasPar MP1, and has been ported to the High Performance Fortran (HPF) programming language to execute on a variety of different massively parallel computing platforms, such as the IBM SP2, the SGI Origin 2000 and the Cray T3E. The Poisson solver also has been written in HPF to ensure the same degree of portability (see Cohl, Sun, & Tohline 1997). All of the simulations described in this dissertation have been executed on 64 nodes of the Cray T3E-900 at the NAVOCEANO MSRC in Stennis, MS or the Cray T3E-600 at the San Diego Supercomputer Center. Generating these results required approximately 136,000 CPU hours or, because every simulation was performed while using 64 nodes concurrently, approximately 2,125 hours of wall clock time.

### 2.3 Heterogeneous Computing Environment

The numerical data that was generated during each of the simulations reported here would have filled a very large, four-dimensional (4D) data array for each of the principal fluid variables. The arrays are 4D because each variable is defined on a three-dimensional numerical grid as a function of time. Figuring out how to sort through this large amount of data to garner useful generic physical information about the fluid flow is a major challenge by itself.

We have found the most useful diagnostic tool to be an animation sequence which shows the time-evolution of isodensity surfaces in each evolving cloud. In order to generate such an animation sequence, historically we (as well as other researchers) usually have adopted the following plan:
• Introduce into the computational fluid dynamics (CFD) code a logical loop including a print statement which instructs the computer to periodically write to disk a 3D data array containing a description of the gas density at every location on the numerical grid.

• At the end of an evolutionary sequence, read the 3D data arrays into another computer (usually a graphical workstation) one at a time, and use a volume-rendering algorithm to generate a 2D image of one isodensity surface for each data array.

• View an animation of the evolution by instructing the workstation to rapidly "flip through" the sequence of 2D images.

Although effective, this plan which relegates the data analysis (visualization) to a post-processing task is generally inefficient and puts large demands on data storage. Also, local workstations usually lack the sophisticated software and specialized hardware used by Visualization Centers to allow the interactive exploration of large 3D data sets and create high quality images of multiple isosurfaces.

Our solution to the stated problem has been to develop a heterogeneous computing environment (HCE) through which our two primary computational tasks (the CFD simulation and visualization) are performed simultaneously on two separate computing platforms, each of which has been configured to handle the assigned task in an optimum fashion. Communication between the tasks (the link) is accomplished over existing local area networks.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
At predetermined intervals of time during the CFD simulation, the HPF program would recognize that a volume-rendered image of the flow should be constructed for inclusion in an animation sequence. At each of these instants in time, data was transferred from the hydrocode executing on the Cray T3E to an SGI Onyx by means of a FTP script or a rcp command. (rcp is a remote file copy utility found on UNIX machines.) Process control for the visualization task was then passed from the T3E to the Onyx via another standard UNIX command remsh. (remsh initiates a shell script on a remote UNIX workstation.) After spawning the visualization task, the T3E would continue following the evolution of the fluid flow, running the CFD simulation task in parallel with the visualization task on the SGI. The volume-rendering task was controlled by the shell script started by the remsh command. As the volume-rendering algorithm finished generating each image, it would immediately delete the 3D density data array, thereby conserving substantial disk space, and FTP the final image file to our local workstations for archiving. Using this technique, we have been able to produce one or more series of images that for each dynamical evolution could easily be converted into one or more animations. The animations produced by this technique have been very useful in interpreting the results described here in Chapters 4, 5 and 6. (See Cazes et al. 1999 for a more detailed description of the HCE.)

2.4 Measuring Virial Balance

In simulations of the type being presented here, the scalar virial equation provides one good measure of the degree to which a given configuration has or
has not reached an equilibrium state. Specifically, if we define a dimensionless ratio,

\[ \delta \equiv \frac{1}{|W|} [T + \Pi - \frac{1}{2}] \]  

in terms of the global kinetic \((T)\), internal \((\Pi)\), and gravitational potential \((W)\) energies as defined, for example, by Chandrasekhar (1969), in a steady-state equilibrium configuration we should formally expect that \(\delta = 0\). In practice, with presently available numerical techniques it is impossible to construct models in which \(\delta = 0\) precisely, but one should at least expect that an equilibrium configuration will exhibit a value of \(|\delta| \ll 1\). The magnitude of \(\delta\) therefore can be used as a measure of how close to a steady-state configuration any given model is, and when a system is out of equilibrium, \(\delta\) provides a global measure of the time rate of change of the system's moment of inertia, \(I\). Specifically, according to the scalar virial theorem (Chandrasekhar 1969) \(\delta\) is related to \(I\) via the expression,

\[ \delta = \frac{1}{2|W|} \frac{d^2I}{dt^2}. \]  

We shall also find it useful to subdivide the global kinetic energy into its "rotational" and "meridional" kinetic energy components. Specifically, when viewed from an inertial frame of reference, we will use the expressions,

\[ T = T_{rot} + T_{rz}, \]  

where,

\[ T_{rot} \equiv \frac{1}{2} \int p v^2_{\phi} dV, \]  

\[ T_{rz} \equiv \frac{1}{2} \int p v^2_{\phi} dV, \]
\[ T_{rz} \equiv \frac{1}{2} \int \rho (v_R^2 + v_z^2) dV. \]  
\[(2.22)\]

As viewed from a frame that is rotating with a uniform frequency \( \Omega_0 \) that is set to match a natural pattern frequency of the system, we will find it useful furthermore to subdivide the rotational kinetic energy into three component pieces, namely,

\[ T_{rot} = T_{frm} + T_{int} + \int \rho R u_\phi \Omega_0 dV, \]  
\[(2.23)\]

where,

\[ T_{frm} \equiv \frac{1}{2} \int \rho R^2 \Omega_0^2 dV, \]  
\[(2.24)\]

is a measure of the rotational energy that is tied up in the pattern rotation and,

\[ T_{int} \equiv \frac{1}{2} \int \rho u_\phi^2 dV, \]  
\[(2.25)\]

is a measure of the rotational energy due to any residual “internal” fluid motions that are present when the configuration is viewed from a rotating frame of reference.

If a model is in perfect equilibrium or a steady-state configuration, all time derivatives of global parameters should vanish; thus \( \delta \) should be zero. Hence, \( \delta \) is a good check of the accuracy of the initial models and provides a rough measure of the dynamical equilibrium of the models during their evolution. Also, by scaling the various energies to the gravitational energy,
we are able to easily compare the evolution and end states of the two different models discussed in this dissertation.
3. INITIAL MODELS

Since our primary objective is to test the viability of Lebovitz's modified fission hypothesis, we begin each of our simulations by constructing an axisymmetric equilibrium object that is a compressible analog of a Maclaurin spheroid having $T/|W| \gtrsim \beta_D$. As discussed in Chapter 1, such axisymmetric objects are expected to be naturally unstable toward the dynamical development of a triaxial nonaxisymmetric structure. In Chapters 4 and 5, we present the long term evolution of two such models that differ mainly in their initial radial distributions of specific angular momentum $j(r)$. As described in some detail by Pickett, Durisen & Davis (1996; hereafter PDD), the initial distribution of specific angular momentum determines to a large degree the shape of the nonaxisymmetric distortion (i.e. the precise eigenfunction) into which the spheroidal structure will initially deform. Following the lead of PDD, we have specifically constructed initial models where the angular momentum distributions are a function of $M_c(R)$, the mass interior to a cylinder of radius $R$. Our first model — Model A — is similar to one that has been studied by PDD, as well as by a number of other groups (Durisen et al. 1986), Williams & Tohline (1987), Smith et al. (1996), and PDD). It is an $n = 3/2$ polytrope with the same angular momentum profile as a uniformly rotating, incompressible sphere, also known as an $n' = 0$ model (Bodenheimer & Ostriker 1973). It has been well documented that this model is unstable toward the development of a pure $m = 2$, nonaxisymmetric distortion that has a loosely wound, trailing spiral character.
Andalib’s (1998) work on two-dimensional nonaxisymmetric equilibrium models has inspired us to create a second initially axisymmetric model — Model B — with uniform vortensity, as defined by Equation (1.4). Since our models have a finite vertical extent, we have used a vertically integrated column density for $\Sigma$ and have assumed that $\nu_\phi$ is not a function of $z$ in our specification of the vortensity $\vec{\xi}$ in the inertial frame. In such an axisymmetric initial model, the only nonzero component of $\vec{\xi}$ is the $z$-component, and, according to Equations (2.9) and (1.4),

$$\xi_z = \frac{1}{\Sigma R} \frac{d}{dR} (R \nu_\phi).$$  \hspace{1cm} (3.1)

By demanding that the model initially have a uniform vortensity, $\xi_0$, we readily derive the following expression for the initial distribution of specific angular momentum in Model B:

$$j = R \nu_\phi = \xi_0 \int \Sigma R dR = \frac{\xi_0}{2\pi} M_c(R).$$  \hspace{1cm} (3.2)

Hence, when using the HSCF technique to construct Model B, we enforced the relationship, $j(R) \propto M_c(R)$.

Figure 3.1 illustrates the differences between the angular momentum distributions of Models A and B. Clearly, the uniform vortensity model contains more angular momentum concentrated toward the center. Table 3.1 lists values of various physical parameters for both initial models. By selecting the polar-to-equatorial axis ratios $c/a$ listed in Table 3.1 as input to the HSCF technique, we were able to create models with $T/|W| = 0.300$ and 0.282 for Models A and B, respectively. As desired both models have $T_{rot}/|W| > \beta_D$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 3.1: Specific angular momentum $j$ as a function of cylindrical enclosed mass $M_c$ for Models A and B at time $t = 0$. Each curve is normalized such that the area under the curve is unity.
Table 3.1: Initial Model Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Sphere</th>
<th>Model A</th>
<th>Model B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$J$</td>
<td>0.00</td>
<td>1.22</td>
<td>1.07</td>
</tr>
<tr>
<td>$M$</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>0.109</td>
<td>4.94(-3)</td>
<td>4.94(-3)</td>
</tr>
<tr>
<td>$\bar{\rho}$</td>
<td>1.82(-2)</td>
<td>8.73(-4)</td>
<td>1.80(-3)</td>
</tr>
<tr>
<td>$\rho_{\text{max}}/\bar{\rho}$</td>
<td>5.99</td>
<td>5.66</td>
<td>2.75</td>
</tr>
<tr>
<td>$R_{eq}$</td>
<td>2.36</td>
<td>10.95</td>
<td>7.95</td>
</tr>
<tr>
<td>$c/a$</td>
<td>1.000</td>
<td>0.208</td>
<td>0.264</td>
</tr>
<tr>
<td>$\tau_{\text{dyn}}$</td>
<td>4.18</td>
<td>19.09</td>
<td>13.30</td>
</tr>
<tr>
<td>$\delta$</td>
<td>---</td>
<td>1.66(-5)</td>
<td>6.93(-4)</td>
</tr>
<tr>
<td>$K$</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

In addition to a variety of global parameters that have already been defined, we have included in Table 3.1 each model’s total angular momentum $J$, maximum density $\rho_{\text{max}}$, equatorial radius $R_{eq}$ ($= a$), and the dynamical time associated with the configuration’s mean density $\bar{\rho}$, where

$$\tau_{\text{dyn}} \equiv \frac{1}{\sqrt{\pi G \bar{\rho}}}. \quad (3.3)$$

For physical parameters that are not automatically dimensionless, we have adopted units such that, $M = G = K = 1$. For comparison, the parameters of a spherical polytrope with the same total mass, polytropic index, and polytropic constant also have been listed.

Because Models A and B fill a larger volume than their spherically symmetric counterpart (i.e., their mean densities are a factor of 21 and 10, respectively, below the mean density of the sphere), their characteristic dynamical times are a factor of $3 - 5$ longer than the dynamical time of a polytropic
sphere of the same mass and polytropic constant. This should be kept in mind as we discuss the time-evolution of each model because all time units have been scaled to $\tau_{dyn}$. It should also be noted that when each model was constructed via the HSCF technique, its equatorial radius was positioned at radial zone 53 in the cylindrical computational grid. This ensured that both models were being constructed with comparable grid resolutions but, because the equatorial radius $R_{eq}$ of the two models was not the same, the computational grids had slightly different linear scales.

In an attempt to emphasize the connection between the simulations being presented here and the classical work on (incompressible) ellipsoidal figures of equilibrium as presented in Chapter 1, we have reproduced Figure 1.1 here as Figure 3.2 and have located the positions of our initially axisymmetric Models A and B along the line $MT$ in Figure 3.2 based on their initial values of $c/a$. (The triangle identifies Model A and the square identifies Model B.) When viewing Models A and B in the context of Figure 3.2, however, it is important to keep in mind that their connection with the classical ellipsoidal figures of equilibrium is by no means a direct one because they are not uniform in density, are not uniformly rotating, and their surfaces deviate significantly from perfect spheroids.
Figure 3.2: Same as Fig. 1.1 with the added triangles and squares representing Model A and Model B, respectively. The symbols along line MT represent the initial configurations of Models A and B as described in Table 3.1. The other symbols represent Models A and B at the times described in the last two columns of Tables 4.1 and 4.2.
4. ADIABATIC, DYNAMICAL EVOLUTION

To date, it has not been possible to directly construct compressible analogs of Riemann ellipsoids, but it is possible to construct compressible analogs of Maclaurin spheroids (the initial configurations of Models A and B). According to Lebovitz’s (1987) modified fission hypothesis, a Maclaurin spheroid formed just past the point of dynamical stability, point D in Figure 3.2, will be unstable to an ellipsoidal deformation leading to a Riemann ellipsoid along the equilibrium sequence DO in Figure 3.2. Hence, by continuing the analogy between the compressible figures presented here and the incompressible configurations discussed in Chapter 1, Models A and B, should suffer a dynamical instability leading to the formation of a compressible analog of a Riemann ellipsoid along the equilibrium sequence DO. While there is no direct way to construct a compressible analog of a Riemann ellipsoid with nontrivial fluid flows, according to our analogy, Models A and B should dynamically evolve to such a steady-state triaxial configuration.

With this in mind, Models A and B were put into the hydrocode and allowed to evolve dynamically. The initial models constructed via the HSCF method, were designed such that they could be put directly onto the numerical grid of the hydrocode without interpolation or rescaling, resulting in an equilibrium balance in the hydrocode equal to that in the HSCF code. So, even though both models were constructed with $T/|W|$ such that they are guaranteed to be dynamically unstable, the formation of dynamical instabilities in the hydrocode may be slow to appear. It is for this reason that we
introduce small nonaxisymmetric perturbations into both models in order to speed up the formation of the dynamical instabilities. (See below in §4.1 for details.)

The specific nature of this initial dynamical instability is not of primary interest to us because it has been examined in greater detail elsewhere. Past studies performed by Durisen et al. (1986), Williams & Tohline (1987), Smith et al. (1996), and PDD, have documented the formation of dynamical instabilities in similar compressible analogs to Maclaurin spheroids. These simulations have always resulted in the formation of an extended disk surrounding a compact central object (as illustrated below), but earlier studies have not examined in detail the internal structure of the central object that is created via this dynamical instability.

In both of our models, we are able to separate the evolution into three distinct phases. In §§4.1 and 4.2, we briefly discuss the initial growth of the $m = 2$ mode and the subsequent phase involving redistribution of angular momentum via nonlinear-amplitude gravitational torques. (See PDD for a more detailed discussion of the early development of this instability for a wider variety of models.) In §4.3 we discuss in greater detail the steady-state structure and behavior of the central triaxial object. Finally, §4.4 draws an association between these models and a number of physically relevant systems. Movies of three dimensional isodensity surfaces of these evolutions may be seen at: http://www.phys.lsu.edu/astro/barmode.
4.1 Initial Development of Nonaxisymmetric Structure

Numerical simulations reported in PDD have clearly shown that the $m = 2$ mode is the dominant mode of dynamical instability in initially axisymmetric, $n = 3/2$ polytropes with angular momentum distributions from $n' = 0$ to $n' = 1.0$ models. As is shown in Figure 3.1, which is similar to Figure 1 in PDD, the angular momentum distribution of Model B is distributed more towards the axis than the equator. This is indicative of a trend noticed by PDD in which the angular momentum distribution decreases at the outer edge and increases towards the center with decreasing $n'$. This implies that Model B is analogous to a model having an $n' < 0$ angular momentum distribution, which led us to suspect that the $m = 2$ mode would be the only mode to develop in Model B.

This is also as stated in Lebovitz’s modified fission hypothesis (1987). As discussed in more detail in Chapter 1, an incompressible, inviscid fluid model with $T/|W| > \beta_D$ should evolve along the equilibrium sequence DO in Figure 3.2. The incompressible model first deforms into an ellipsoidal shape, then further along the sequence may become susceptible to a fourth-order surface deformation. Hence, if our compressible models are good analogs of the incompressible systems, their early evolution should not involve the nonlinear growth of any odd modes.

Because of the early dominance of the $m = 2$ mode and the absence of strong vertical flows, we chose to use $\pi$-symmetry and equatorial symmetry in the initial evolution of our models to reduce computing costs. Models A and B were also given a low-amplitude, $\cos(2\phi)$ density perturbation in an
effort to speed up the development of the instability and save computational
time. The amplitude of the imposed perturbation was 1% and 5% for Models
A and B, respectively. In contrast to the fragmentation models discussed
in Chapter 1, the $m = 2$ perturbation was chosen for expediency. Past
numerical simulations (Durisen et al. 1986, Williams & Tohline 1987, and
PDD) and linear analyses (Toman et al. 1998) of Model A ($n' = 0$) and
similar models have all shown that the $m = 2$ mode is the most unstable mode
to develop during the early evolution regardless of the initial perturbation.

In Figures 4.1 and 4.2, we track the time-dependent behavior of various
global energy parameters during the dynamical adiabatic evolutions of Mod­
els A and B, respectively. (Note: Unless stated otherwise, these global values
come from integrations over the total computational grid.) This phase of the
evolution begins at time $t = 0$ with the initial axisymmetric model. We mark
the end of this “initial development” phase at a time ($\sim 12\tau_{dy}$) when the
growing $m = 2$ distortion begins to reach a nonlinear amplitude. This also
corresponds with the time in Figures 4.1 and 4.2 when the contribution of
the meridional kinetic energy begins to noticeably increase after about 12
$\tau_{dy}$ for each model.

4.2 Nonlinear Phase

The next phase is one of visible nonlinear change in both models, but the
precise nature of the change differs between the two models. After approx­
imately 12 dynamical times into the evolution of each model, the $T_{rs}/|W|
parameter begins to increase noticeably in Figures 4.1 and 4.2, reflecting the
deformation of the gas flow into nonaxisymmetric orbits. This coincides with
Figure 4.1: Plot of global energy ratios as a function of time for Model A. $T_{\text{rot}}$ is the total rotational kinetic energy, $T_{rz}$ is the total meridional kinetic energy, $T_{\text{frm}}$ is the rotational kinetic energy due solely to the motion of the rotating frame, $T_{\text{int}}$ is the rotational kinetic energy due solely to the internal flow within the rotating frame, $\Pi$ is the total thermal energy, and $W$ is the total gravitational energy. Each of these global quantities is calculated from an integral over the entire numerical grid. The beginning of the steady-state evolution is marked at the bottom of the figure by a heavy vertical line. (The discontinuity in Fig. 4.1 results from a miscalculation of the total meridional kinetic energy in the early evolution. This miscalculation only affected the diagnostic parameters $T_{rz}/|W|$ and $T/|W|$.)
Figure 4.2: Same as Fig. 4.1 for Model B.
the visible appearance of the first set of $m = 2$ spiral arms (see frames a and d of Fig. 4.3). At approximately 22 dynamical times, a second set of spiral arms appear out of sync with the first set. This destroys the coherency of the spiral mode and marks the beginning of a series of convulsions that lead to the formation of the steady-state bar structure.

The spiral arm instability transports mass and angular momentum to the outer edge of Model A resulting in some shedding of high specific angular momentum material. The spiral arm mode is the dynamical instability by which the internal angular momentum distribution of Model A is reconfigured to form the lower energy triaxial object. The mass that is shed via this process forms an extensive circumstellar disk that we do not attempt to follow in detail here. The formation and evolution of this disk has been previously studied in considerable detail by Durisen et al. (1986) and Williams & Tohline (1987). The gas in the circumstellar disk remains in Keplerian orbits outside of the corotation radius. (See Eq. 4.1 for the definition of the corotation radius and Table 4.1 for the amount of gas and angular momentum lost to the circumstellar disk.) In our simulations, any mass ejected by the model is allowed to flow freely off the numerical grid. The dynamical evolutions performed by PDD also exhibit this two-armed spiral character.

The time evolution of Model A may be followed in Figures 4.3a, 4.3b, and 4.3c. The contours shown in Figure 4.3 are curves of uniform density in the equatorial plane with the vector momenta superimposed. Figure 4.3a shows the beginning alignment of the $m = 2$ mode, and Figures 4.3b and 4.3c show the initial formation of the spiral arms. As these convulsions lessen
in intensity, we are left with the classic bar-shaped object with a rotational period of 8.65 dynamical times. Hence, the final triaxial figure of Model A has formed and stabilized in less than 4 pattern rotations.

In contrast to Model A's evolution, this phase in Model B is remarkably mild. The spiral character of the initial disturbance in Model B is hardly noticeable when compared to that of Model A. The triaxial object elongates and becomes more bar shaped. This much milder evolution is pictured in Figures 4.3d, 4.3e, and 4.3f. As in Figure 4.3a, Figure 4.3d shows the initial alignment of the bar mode, but Figures 4.3e and 4.3f show only a slight hint of spiral arms. These events also coincide with the local minima in the time-evolutionary behavior of the $T_{rot}/|W|$ parameter, as illustrated in Figures 4.1 and 4.2.

The final triaxial object in Model B forms much more quickly than in Model A. Here, the bar-shaped configuration forms in ~20 dynamical times, but the pattern period is ~13 dynamical times; therefore, Model B formed a bar in less than two pattern rotations, or twice as quickly as Model A. In part this is due to the greater initial perturbation given to Model B; but also, we suspect, due to a more appropriate choice of the initial angular momentum distribution.

This phase of the evolution also follows a pattern spotted by PDD. They noted that the strength and twisting of the initial spiral instability increased with $n'$ for an $n = 3/2$ polytrope. Since Model B is analogous to an $n' < 0$ model, the lack of a strong initial spiral instability is to be expected from their results. Although the initial spiral instability in Model B was very weak, at
Figure 4.3: Time evolution of density contours along with vectors representing the momenta in the equatorial plane of Models A and B. Model A is plotted in Figs. 4.3a, 4.3b, and 4.3c. Model B is plotted in Figs. 4.3d, 4.3e, and 4.3f. The contours are at $\rho/\rho_{\text{max}} = 0.01, 0.05, 0.1, 0.5,$ and $0.95$. The times in units of dynamical times are printed in the upper left-hand corner of each figure. The grids for Figs. 4.3a, 4.3b, and 4.3c are scaled to the initial equatorial radius for Model A. The grids for Figs. 4.3d, 4.3e, and 4.3f are scaled to the initial equatorial radius for Model B.
sufficiently low densities one can still observe the formation of a circumstellar disk via mass shedding. But for Model B, the percentage of mass and angular momentum shed during the spiral instability phase is less than half that shed by Model A. Again, this leads us to believe that uniform vortensity may be a constraint needed to build a three-dimensional, nonaxisymmetric equilibrium polytrope.

4.3 Steady-State Behavior

After an evolution of 46 and 32 dynamical times, respectively, Models A and B both appear to be in steady-state configurations, assuming the $\pi$-symmetry constraint is still valid. The absence of nonlinear growth of the odd modes during the initial formation of the triaxial object does not preclude their development at later stages; consequently, we allowed the models to evolve with a full $2\pi$-symmetry after reaching a steady-state configuration in $\pi$-symmetry. (This switch to a full $2\pi$-symmetry is marked by a heavy vertical line at the bottom of Figures 4.1 and 4.2.) This was accomplished by doubling the size of the grid in the azimuthal direction and replicating the models from $\pi$ to $2\pi$, thus retaining the same spatial resolution as the earlier $\pi$-symmetric evolution. Each model was also put into a reference frame rotating with the frequency of the $m = 2$ pattern. Models A and B were then evolved through two full pattern rotations on the larger $128 \times 128 \times 256$ grid, imposing only equatorial symmetry.

We label the final models as steady-state configurations, whereby we define steady-state as "unchanging on a dynamical time scale." Due to numerical dissipation in the hydrocode and a small amount of mass shedding, the
model structures appear to still be changing slowly, but on a secular time scale.

4.3.1 Steady-State Evolution

During this steady-state phase, the models continue to be dominated by the \( m = 2 \) and \( m = 4 \) modes with the amplitude of the \( m = 4 \) mode being an order of magnitude less than that of the \( m = 2 \) mode. Although the \( \pi \)-symmetry constraint has been removed, the models remain stable against the development of odd-mode nonaxisymmetric distortions. Hence, neither model suffers the "pear"-mode instability that has been discussed in connection with evolution along the Jacobi sequence. (See §1.3)

Tables 4.1 and 4.2 list the intrinsic properties of Models A and B in their initial axisymmetric configurations as well as at the beginning and end of their steady-state, adiabatic evolution. Unlike the global parameters plotted in Figures 4.1 and 4.2, the values listed in these tables involve only the material within the bar, where the bar is defined in the hydrocode as all numerical grid cells with a density \( \rho > 0.05\rho_{\text{max}} \). The final object in the case of Model A retains approximately 73\% of its initial angular momentum and 90\% of its initial mass. By contrast, Model B retains 90\% of its angular momentum and 95\% of its initial mass because it did not undergo a strong spiral arm phase with significant mass shedding. Notice that during this phase of the evolution in both models, the density maxima remain \( \sim 30\% \) greater than their initial values and are located further from the axis.

According to Figures 4.1 and 4.2, the various global energy parameters change very little during this phase of the evolution and converge to similar
Table 4.1: Model A Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Time$</td>
<td>0.00</td>
<td>46.26</td>
<td>64.59</td>
</tr>
<tr>
<td>$T/</td>
<td>W</td>
<td>$</td>
<td>0.300</td>
</tr>
<tr>
<td>$T_{rot}/</td>
<td>W</td>
<td>$</td>
<td>0.300</td>
</tr>
<tr>
<td>$T_{frm}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{int}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{rs}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$II/</td>
<td>W</td>
<td>$</td>
<td>0.200</td>
</tr>
<tr>
<td>$\delta$</td>
<td>1.66(-5)</td>
<td>3.97(-3)</td>
<td>6.73(-3)</td>
</tr>
<tr>
<td>$J$</td>
<td>1.22</td>
<td>0.881</td>
<td>0.843</td>
</tr>
<tr>
<td>$J_{frm}$</td>
<td>0.562</td>
<td>0.514</td>
<td></td>
</tr>
<tr>
<td>$J_{int}$</td>
<td>0.319</td>
<td>0.329</td>
<td></td>
</tr>
<tr>
<td>$M$</td>
<td>1.000</td>
<td>0.916</td>
<td>0.909</td>
</tr>
<tr>
<td>$\rho_{max}$</td>
<td>4.94(-3)</td>
<td>6.21(-3)</td>
<td>6.27(-3)</td>
</tr>
<tr>
<td>$R_{eq}$</td>
<td>10.95</td>
<td>9.19</td>
<td>8.37</td>
</tr>
<tr>
<td>$R_{p_{max}}/R_{eq}$</td>
<td>0.000</td>
<td>0.123</td>
<td>0.142</td>
</tr>
<tr>
<td>c/a</td>
<td>0.208</td>
<td>0.218</td>
<td>0.235</td>
</tr>
<tr>
<td>b/a</td>
<td>1.000</td>
<td>0.540</td>
<td>0.605</td>
</tr>
<tr>
<td>$f$</td>
<td>1.26</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>$\Omega_0$</td>
<td></td>
<td>0.726</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.2: Model B Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.00</td>
<td>32.05</td>
<td>59.09</td>
</tr>
<tr>
<td>$T/</td>
<td>W</td>
<td>$</td>
<td>0.282</td>
</tr>
<tr>
<td>$T_{rot}/</td>
<td>W</td>
<td>$</td>
<td>0.283</td>
</tr>
<tr>
<td>$T_{frm}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{int}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{rs}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$\Pi/</td>
<td>W</td>
<td>$</td>
<td>0.217</td>
</tr>
<tr>
<td>$\delta$</td>
<td>6.93(-4)</td>
<td>1.81(-3)</td>
<td>3.14(-3)</td>
</tr>
<tr>
<td>$J$</td>
<td>1.071</td>
<td>1.018</td>
<td>0.941</td>
</tr>
<tr>
<td>$J_{frm}$</td>
<td>0.722</td>
<td>0.582</td>
<td></td>
</tr>
<tr>
<td>$J_{int}$</td>
<td>0.296</td>
<td>0.359</td>
<td></td>
</tr>
<tr>
<td>$M$</td>
<td>1.000</td>
<td>0.975</td>
<td>0.958</td>
</tr>
<tr>
<td>$\rho_{max}$</td>
<td>4.94(-3)</td>
<td>6.37(-3)</td>
<td>6.69(-3)</td>
</tr>
<tr>
<td>$R_{eq}$</td>
<td>7.95</td>
<td>9.67</td>
<td>8.47</td>
</tr>
<tr>
<td>$R_{gmax}/R_{eq}$</td>
<td>0.140</td>
<td>0.387</td>
<td>0.311</td>
</tr>
<tr>
<td>$c/a$</td>
<td>0.264</td>
<td>0.213</td>
<td>0.239</td>
</tr>
<tr>
<td>$b/a$</td>
<td>1.000</td>
<td>0.433</td>
<td>0.522</td>
</tr>
<tr>
<td>$f$</td>
<td></td>
<td>1.34</td>
<td>1.41</td>
</tr>
<tr>
<td>$\Omega_0$</td>
<td></td>
<td></td>
<td>0.488</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
values. Specifically, the values of $T/|W|$ and $\Pi/|W|$ both appear to converge to 0.25 in both models. We are not sure if this is a general characteristic that will ultimately be attributable to a wide variety of triaxial steady-state configurations or if it is just a coincidence in these two cases. As can be seen by the low $T_{rs}/|W|$ ratio, the kinetic energy of both models is dominated by the rotational flow. The parameters $T_{frm}/|W|$ and $T_{int}/|W|$ are measures of the rotational kinetic energy due to the rotating frame and the internal flow of the pattern, respectively.

Figures 4.4 and 4.5 show isodensity contours in the equatorial plane of Models A and B, respectively, at the same two points in time that we have specified in Tables 4.1 and 4.2 as the beginning and end of this phase of evolution. The axes for each plot are scaled to the equatorial radius of the initial model. The outermost thin line contour at $\rho = 0.05\rho_{\text{max}}$ identifies the lowest density level that we have used to define the bar. The dashed circle identifies the corotation radius, $R_{\text{cor}}$, where

$$R_{\text{cor}} \equiv \left[\frac{GM_{\text{bar}}}{\Omega_0^2}\right]^{1/3},$$

and $M_{\text{bar}}$ represents only the mass of the final barred object, as opposed to the mass spread over the entire computational grid. The corotation radius defines the circular orbit of a particle rotating with the period of the pattern about a central point mass with a mass equal to the mass of the bar. The gas outside the corotation radius should be following Keplerian trajectories.

The heavy curves in Figures 4.4 and 4.5 which resemble an outline of the main body of a violin trace out the surface where the magnitude of the fluid
Figure 4.4: Equatorial density contours for Model A at the beginning and end of the steady-state phase. The times listed are in units of $\tau_{\text{dyn}}$. Density contours are for $\rho/\rho_{\text{max}} = 0.95, 0.75, 0.5, 0.25, 0.1, \text{ and } 0.05$. The dashed circle marks the corotation radius $R_{\text{cor}}$. The heavy curve marks the equatorial contour of the violin mach surface. All flow within this curve is subsonic in the rotating frame.
Figure 4.5: Same as Fig. 4.4 for Model B.
velocity in the rotating frame equals the local sound speed. Henceforth we will refer to this as the violin mach surface. Material interior to this surface is moving subsonically in the rotating frame. This internal flow is key to maintaining the barred shape of the final object. As the fluid streams down one side of the bar, the flow becomes supersonic then the fluid experiences a mild shock and the flow becomes subsonic as it climbs back out of the potential well and attempts to turn the corner to flow around the tip of the bar. This standing shock appears to be a steady-state feature of both models.

The isodensity contours range from dumbbell to binary in shape, but each model cannot be thought of as a common-envelope binary because, in each case, there is flow through the density maxima rather than around them. As in Figure 4.3, Figures 4.6 and 4.7 show the momentum vectors in the equatorial plane of Models A and B, respectively, but here the superimposed contours are those of the effective potential, $\Phi_{eff}$, which we define as,

$$\Phi_{eff} \equiv \Phi - \frac{1}{2} \Omega_0^2 R^2.$$  

(4.2)

Although our primary interests are in star formation, this effective potential compares remarkably well to a rotating barred potential of the type used to model the centers of barred-spiral galaxies (cf., Fig. 3.13 of Binney & Tremaine 1987). For comparison, we also identify the five Lagrange points L1-L5. The L1 and L2 points (identified by crosses) denote the inflection points of the effective potential $\Phi_{eff}$; L3 (the diamond) denotes the central extremum, which in our case is actually a slight maximum, and L4 and L5 (asterisks) denote the maxima of the effective potential $\Phi_{eff}$. Dynamical
Figure 4.6: Equatorial momenta for Model A at the beginning and end of the steady-state phase. The heavy circle marks the corotation radius $R_{\text{cor}}$. Solid line contours are of $\Phi_{\text{eff}}$. The two dashed line contours are of the density at $\rho/\rho_{\text{max}} = 0.01$ and 0.05. Cross symbols mark the L1 and L2 points, asterisks mark the L4 and L5 points, and a diamond symbol marks the L3 point. Notice that the corotation radius falls between the L4-L5 radius and the L1-L2 radius.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 4.7: Same as Fig. 4.6 for Model B.
studies of galactic bars have shown that the corotation radius should lie between the L4-L5 and L1-L2 radii which agrees with our models (Binney & Tremaine 1987). In Figure 4.8, where we have plotted a surface representing $\Phi_{\text{eff}}$ in the equatorial plane of Model B at the beginning of its steady-state evolution, the nature of the Lagrange points may be more clearly seen.

As stated earlier in §2.4 (see especially Eq.[2.18]), one measure of equilibrium is the sum of $T/|W|$ and $\Pi/|W|$. If the model is in a perfect equilibrium, these two values should sum to 0.5. We see from Figures 4.1 and 4.2, that in both models, the parameters $T/|W|$ and $\Pi/|W|$ remain respectively, 0.26 and 0.24 throughout the steady-state phase of the evolution. Another non-global measure of steady-state equilibrium can be obtained by calculating the inner
product of the gradient of the Bernoulli function, $B$, and the velocity, where the Bernoulli function is defined as:

$$B = \frac{1}{2} (u^2 - \Omega_0^2 R^2) + H + \Phi$$  \hspace{1cm} (4.3)$$

From the equation of motion (2.2), we know that

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \times (\vec{\zeta} + 2\Omega_0) = -\nabla B$$  \hspace{1cm} (4.4)$$

Therefore, taking the inner product of $\vec{u}$ with the equation of motion (2.2) gives,

$$\frac{1}{2} \frac{\partial u^2}{\partial t} = -\vec{u} \cdot \nabla B,$$  \hspace{1cm} (4.5)$$

which should be zero in a steady-state configuration. In both Models A and B, this inner product was compared to the magnitudes of $\nabla B$ and $\vec{u}$ and found to be everywhere at least an order of magnitude less than their product.

**4.3.2 Similarities to Riemann Ellipsoids**

To better understand the nature of these models and provide closer comparisons to the classical understanding of Riemann ellipsoids, we must discuss the parameters $T_{frm}/|W|$ and $T_{int}/|W|$ defined by Equations (2.24) and (2.25), respectively, in §2.4. By utilizing these two parameters, we are able to compare the influence of the rotational energy due to the frame to the rotational energy due to internal flow within the bar. In Tables 4.1 and 4.2 (see also Figures 4.1 and 4.2), the ratio $T_{frm}/|W|$ is approximately twice the ratio $T_{int}/|W|$ during the entire steady-state evolution. In Tables 4.1
and 4.2, the total angular momentum of each object is listed along with the contributions from the rotating frame and the fluid motion within the bar. Notice that in each model, the angular momentum due to the rotating frame is more than twice that due to the fluid motion within the bar. Therefore, these objects have a "Jacobi"-like flow and are analogous to the Riemann ellipsoidal sequences in which \( f \) approaches \( f = 2 \) from the Jacobi sequence, \( f = 0 \).

While these objects exhibit neither uniform vorticity nor uniform vortensity, we can obtain a global measure of the \( f \) parameter of each model. We do this by volume averaging the \( z \)-component of the vorticity over the volume of the bar to get a representative value of \( \zeta_z \) to use in Equation (1.1). The resulting values of \( f \), listed in Tables 4.1 and 4.2, are between 1 and 2, which is also the correct range for a "Jacobi"-like ellipsoid near the curve DO.

In Figure 3.2, Models A and B are placed in the figure according to their respective axis ratios at both the beginning and the end of the steady-state evolution, as defined by the data in Tables 4.1 and 4.2. (The triangles represent Model A, and the squares represent Model B.) At the beginning of the steady-state phase of their evolution, both models appear to fall upon the curve DO then, as they evolve, they move slowly to the right along this curve back towards point D. We believe this slow progression towards axisymmetry is a result of numerical dissipation within our code. The hydrocode is an inviscid code which handles shock front situations in the fluid by dropping from a second order advection method to a first order method. The lasting appearance of a mild shock front associated with the violin mach surface in
both models guarantees that there will be a section of the numerical grid in which the fluid is being advected only to first order accuracy.

### 4.3.3 Departure from Analytical Compressible Ellipsoids

Even though, to first order, our two triaxial models compare favorably to the classical Riemann ellipsoids, both models contain features which clearly distinguish them from the Riemann ellipsoids and the previous analytical work done regarding compressible ellipsoids. First, the isodensity surfaces of Model A and Model B are neither ellipsoidal or concentric. As one can clearly see in Figures 4.4 and 4.5, our final steady-state figures are better described as bar-like in shape with isodensity surfaces that range from barred to dumbbell in shape. Each configuration also contains slight off-axis density maxima instead of a maximum density located at the center of the object. As discussed in Chapter 1 (§1.4), in earlier studies (Chambat 1994; Fillipi et al. 1996) that have claimed to prove “non-existence” of compressible ellipsoids, equilibrium models were restricted to ellipsoidal shapes with concentric ellipsoidal isodensity surfaces and a central maximum density. The equilibrium configurations of Uryū & Eriguchi (1996) are more closely related to our models, although their models have forced symmetries about the $x$-$z$ and $y$-$z$ planes while our models do not. The absence of this type of symmetry in our models is due mainly to the mild shock which causes a kink in the isodensity contours approximately 135° away from the density maximum.

This brings us to another major difference between our models and their analytical counterparts. Both of our steady-state models contain supersonic flows in the rotating frame and support a mild, standing shock front. No such
flow arises in the incompressible ellipsoids modelled by Riemann, and the possibility of such flows has not been considered in any previous study. We believe that the lasting presence of this shock front may be key to maintaining the steady-state nature of compressible, triaxial objects.

Finally, the fluid flow in both Model A and Model B appears to be horizontally planar and similar in character at different heights, but not to the degree enforced in the studies of Chambat (1994) and Fillipi et al. (1996). They required the vorticity to be aligned with the rotation axis, but in our models, the vorticity $\zeta$ is slightly misaligned with the axis of rotation. This is consistent with the results of Uryū and Eriguchi (1996) who were only able to build compressible triaxial models if they allowed the polar flow to vary slightly in the vertical direction. This vertical variance of the polar flow results in a misalignment of $\zeta$ with $\Omega_0$.

4.4 Relationship to Physical Systems

The steady-state bar pattern that has been formed by the numerical simulations discussed in the preceding sections of this chapter is relevant to a wide variety of astrophysical systems. Although the length and time scales of these various systems span a range greater than twelve orders of magnitude, they all share key characteristics of the simulations described above. Each is considered to be a rapidly rotating, self-gravitating gaseous configuration with an equation of state that may be described to first order by a polytrope. The systems we have in mind vary in size from a thousand parsecs (for a galactic bar) to a hundred AU (for a protostellar cloud), to 10 kilometers (for a neutron star). In order to see if our numerical models
exhibit properties that are consistent with observations and theoretical ideas associated with these various systems, we need to scale our model parameters to the appropriate length scales. Up until this point, the simulations have been discussed using dimensionless units scaled such that $G = K = 1$ and, initially, $M = 1$. By using instead a value of $G$ that has the proper units and fixing two physical parameters of the numerical model — such as the mass and period — we are able to scale all of our model parameters in a consistent way to any physical system. (For details of how this is done, see appendix A in Williams & Tohline [1987].)

For the protostellar case, the defining characteristics of a cloud are its mass and its rotational period. Table 4.3 lists four systems that we have “constructed” from our Model A assuming a total mass of $1 \text{M}_\odot$ and pattern periods of 1 year, 10 years, 100 years, 250 years, and 1000 years. In each case, the mean temperature $T$ was calculated using the ideal gas law

$$P = \frac{R}{\mu} \rho T$$

(4.6)

where $R$ is the gas constant and $\mu$ is the mean molecular weight. For protostellar systems, the gas is assumed to be neutral hydrogen with $\mu = 1$. The first entry in Table 4.3 is for a system with a pattern period of one year. The bar extends to approximately one AU which is the radius of corotation for a $1 \text{M}_\odot$ system with a period of one year. The fourth entry, with a period of 250 years, scales our numerical model to the approximate size of the solar system. For comparison, Pluto's orbit is 39.4 AU with a period of 248.6 years. Even though the period and size are correct for our solar system, the
Table 4.3: Protostellar Systems

<table>
<thead>
<tr>
<th>Mass(M(_\odot))</th>
<th>Period(yrs)</th>
<th>R(_{eq})(AU)</th>
<th>(\bar{\rho})(g cm(^{-3}))</th>
<th>T(° K)</th>
<th>J(g cm(^2)s(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>1</td>
<td>0.985</td>
<td>1.51(-7)</td>
<td>4860</td>
<td>2.53(52)</td>
</tr>
<tr>
<td>1.00</td>
<td>10</td>
<td>4.57</td>
<td>1.51(-9)</td>
<td>1050</td>
<td>5.46(52)</td>
</tr>
<tr>
<td>1.00</td>
<td>100</td>
<td>21.2</td>
<td>1.51(-11)</td>
<td>224</td>
<td>1.18(53)</td>
</tr>
<tr>
<td>1.00</td>
<td>250</td>
<td>39.1</td>
<td>2.42(-12)</td>
<td>122</td>
<td>1.60(53)</td>
</tr>
<tr>
<td>1.00</td>
<td>1000</td>
<td>98.5</td>
<td>1.51(-13)</td>
<td>49</td>
<td>2.53(53)</td>
</tr>
</tbody>
</table>

angular momentum is much too large. For our solar system, we can calculate the approximate total angular momentum using the mass of the planets, their orbital radii, and orbital periods, while ignoring any extra contributions due to the individual rotation of each planet. Using this method, the angular momentum of the solar system is \(\approx 3 \times 10^{60}\)g cm\(^2\)s\(^{-1}\), or about 500 times less than that of a solar system sized bar. Therefore, unless our solar system shed 99.8% of its initial angular momentum during the final contraction phase, this model is unfeasible for it. However, the angular momentum of two \(\frac{1}{2}M\(_\odot\)\) stars in a circular orbit about each other with a separation of 40 AU results in an angular momentum of approximately \(3 \times 10^{53}\)g cm\(^2\)s\(^{-1}\), which is only off by a factor of two.

The steady-state bar pattern of our final models suggests a comparison of a different sort. Many spiral galaxies contain a prominent central region that is bar-shaped. Indeed, it is suspected that our own Galaxy has a weakly barred nucleus. Barred galaxies usually exhibit dust lanes that appear to run along the side of the bar. In some cases, it is known that this is due to the streaming of the gas along the length of the bar (Mihalas & Binney 1981). For an excellent image of this phenomenon, see NGC1300 in the Hubble
Table 4.4: Compact Object Systems

<table>
<thead>
<tr>
<th>Mass(M☉)</th>
<th>Period(s)</th>
<th>R_eq(km)</th>
<th>ρ(g cm⁻³)</th>
<th>ρ_max(g cm⁻³)</th>
<th>T(° K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>38.7</td>
<td>16900</td>
<td>1.00(5)</td>
<td>1.71(6)</td>
<td>2.12(7)</td>
</tr>
<tr>
<td>1.50</td>
<td>1.22(-3)</td>
<td>19.4</td>
<td>1.00(14)</td>
<td>1.71(15)</td>
<td>2.78(10)</td>
</tr>
</tbody>
</table>

Atlas (Sandage 1961). Observations of the center of our Galaxy show that there is an inner disk of molecular hydrogen with a radial extent of about 300 pc and a total estimated mass of 10⁶ M☉ (Mihalas & Binney 1981). If our steady-state Model A bar is scaled to these values, the resulting object has a period of approximately 5 × 10⁷ years with an average density of 10⁻²³ g cm⁻³ and a temperature of 7730°K. This roughly matches what is observed at the center of our Galaxy; specifically, the molecular hydrogen has an orbital period \( \sim 9 \times 10⁷ \) years.

For completeness, we have also scaled our steady-state, triaxial model to average densities representative of white dwarfs (10⁶ g cm⁻³) and neutron stars (10¹⁴ g cm⁻³). Table 4.4 shows the results of such a scaling.¹

Although the period of the neutron star is about one millisecond, a more realistic neutron star equation of state would be much stiffer, i.e. the triaxial deformation would not be nearly as strong as in the \( n = 1.5 \) gas. Hence, for the purposes of estimating the strain due to gravitational radiation, these results should be taken very lightly. For an in-depth study of this problem from an observational standpoint, see New et al. (1995).

¹The polytropic index \( n = 1.5 \) is not realistic for the compact systems mentioned here. A more appropriate choice of polytropic index would be \( n = 3 \) for white dwarfs and possibly \( n = 1 \) or \( n = 0.5 \) for neutron stars. With these caveats in mind, Table 4.4 should be appreciated as a crude estimate of the global parameters of these compact objects if they were rapidly rotating.
5. COOLING EVOLUTION AND THE FORMATION OF A BINARY

5.1 Cooling Paradigm

As was discussed briefly in Chapter 1, once a protostellar gas cloud collapses to densities $\sim 10^{-13}\text{g cm}^{-3}$, the thermal energy generated through contraction cannot escape freely, causing the cloud to heat up and permitting it to come into hydrodynamical equilibrium with thermal pressure forces and centrifugal forces offsetting gravitational forces. From this point on, the gas cloud evolves nearly adiabatically with a loss of thermal energy occurring only at the surface. Further contraction is limited by how quickly the protostellar cloud can radiate away energy.

The time taken for a star to contract from the initial point of hydrodynamical equilibrium (the point at which the free-fall collapse phase ends) to the main sequence (the point at which hydrogen fusion begins) is referred to as the thermal, or Kelvin-Helmholtz age ($\tau_{KH}$). Detailed spherically symmetric stellar evolution calculations by Stahler (1983) have shown that the Kelvin-Helmholtz age of a $1\ M_\odot$ star is approximately 20 million years. Stahler checked his calculations by comparing the numerical results with observations of stellar systems in nearby star-forming regions. Thus, an order of magnitude estimate for the contraction time for a $1\ M_\odot$ protostellar cloud from the end of the collapse phase to the onset of hydrogen burning is 10 million years. Being spherically symmetric, Stahler’s calculations were per-
formed with no rotation, so the early collapse was halted solely by thermal pressure.

To our knowledge, there have been no numerical studies performed that include the effects of rotation throughout this slow thermal contraction phase. There have been some studies performed on the effects of rotation near the end of the contraction phase (cf. Durisen, Yang, Cassen, & Stahler 1989), but in these studies the ratio of rotational to gravitational energy has been relatively small ($T_{rot}/|W| \ll 0.06$). Therefore, we are not yet sure what the exact thermal timescale for the evolution of our models should be, but we can safely assume that the thermal contraction phase lasts much longer than a dynamical time.

Because the thermal timescale is orders of magnitude greater than the dynamical timescale, a protostellar cloud should remain in hydrostatic balance as it undergoes thermal contraction toward the main sequence. Due to the large difference in timescales, however, our present numerical simulation techniques do not enable us to follow the evolution of the protostellar cloud on a true thermal timescale. But we are able to follow the evolution over many dynamical times. Thus, by cooling our steady-state model on a modified thermal timescale $\tau_{th} \ll \tau_{KH}$ but at the same time making sure that $\tau_{th}$ is many dynamical times, we ensure that the model remains in a state that is very close to hydrostatic balance as it contracts.

To test the modified fission hypothesis of Lebovitz (1987), we have implemented a simplistic method of allowing our protostellar cloud to cool. At the end of the steady-state evolution, Model A has been cooled uniformly by
allowing the polytropic constant $K$ in Equation (2.6) to decrease linearly as a function of time, as follows:

$$K(t) = K_0(1 - \frac{1}{\tau_{th}}(t - t_0)), \quad (5.1)$$

where $t_0$ is the time at which the cooling started and $K_0$ is the polytropic constant at time $t_0$. The modified thermal timescale $\tau_{th}$ has been chosen such that the corresponding evolution would be many dynamical times, but not so long that the simulation would take an excessive amount of computer simulation time to finish. Specifically, we have chosen $\tau_{th} = 35\tau_{dyn}$, such that after approximately 18 dynamical times, the thermal pressure and the polytropic constant $K$ will have dropped to half their original value.

Although this cooling method is rather simplistic, it should give us a rough picture of the type of instability that will occur during a phase of slow, thermal contraction. To be done correctly, however, techniques that model the radiation interaction with the gas, radiative transport techniques, would have to be included in the hydrocode along with a function to simulate thermal emission as a function of density, pressure, and temperature. And, as discussed above, one would have to perform a modification of the technique to shorten the thermal timescale of the radiative transport processes, such that the dynamical evolution of the protostellar cloud could be followed with a dynamical code such as the hydrocode. A fully three-dimensional simulation of this sort is presently beyond the capabilities of any group. Hence, we feel justified in our first attempt to create a binary system via the modified fission hypothesis utilizing this more simplified cooling technique.
5.2 Cooling Simulation

To simulate the slow contraction phase of a protostellar cloud, Model A has been cooled uniformly via the technique described above. The numerical simulations have been performed exactly as before with the exception that $K$ is now a function of time. The cooling simulation begins at the end of the steady-state evolution discussed in Chapter 4. As before, this simulation was performed on a 128 × 32 × 256 cylindrical grid with equatorial symmetry.

Figure 5.1 plots the global parameters $T/|\omega|$, $T_{rot}/|\omega|$, and $\Pi/|\omega|$ as a function of time, for this Model A cooling evolution. The point in time $(t/\tau_{dyn} \approx 64)$ at which the cooling technique was implemented is marked at the bottom of the figure by the leftmost heavy vertical line. The transition appears to be fairly smooth. Oscillations in the plotted energy variables are not noticeable until Model A has been cooled by approximately 50% $(t/\tau_{dyn} \approx 87)$.

As Model A cooled, it contracted somewhat as expected. As a result, fewer and fewer zones in our fixed computational grid were actually being used to resolve the central object. In an attempt to solve this spatial resolution problem, at time $t/\tau_{dyn} \approx 78$ we interpolated Model A to a finer numerical grid, giving the hydrocode more grid cells with which to resolve the central object. We continued to use a numerical grid with 128 × 32 × 256 zones as before, but the spacing between the grid points was reduced by a factor of 2.5, i.e. the radial and vertical extent of the numerical grid was reduced by a factor of 2.5 allowing the central object in Model A to fill a larger percentage of the grid. The regridding of Model A was implemented
Figure 5.1: Plot of $T/|W|$, $T_{\text{rot}}/|W|$, and $\Pi/|W|$ as a function of time for the cooling simulation. Each of the vertical lines at the bottom of the figure mark significant points in the cooling evolution of Model A. The tallest vertical line farthest to the left marks the initial starting point of cooling. The second shorter vertical line marks the point at which Model A was regridded and cooling was stopped. The shortest vertical line marks the point at which cooling was restarted on the higher resolution grid. The vertical line farthest to the right marks the point at which cooling was again stopped.
after allowing the model to cool by approximately 40%. This point is indi-
cated by the second vertical line at the bottom of Figure 5.1. In an effort
to allow any perturbations that may have been introduced into Model A by
the interpolation process to subside, the model was subsequently evolved for
5 dynamical times without cooling. The point in time at which the cooling
process was restarted \( t/\tau_{\text{dyn}} \approx 83 \) is marked by the shortest vertical line
along the bottom axis in Figure 5.1.

Conservation of angular momentum provides another complication to the
cooling simulation. Ideally, we would have liked the simulation to be per-
formed in a rotating frame that at all times corresponds to the central triaxial
object's pattern rotation frequency, but as Model A contracted, it spun up.
Rather than try to introduce an angular acceleration into the hydrocode,
we periodically chose to jump into a new rotating frame that was rotating
with the object's new pattern speed. Ultimately, the cooling simulation was
performed in rotating frames having three different frequencies: \( \Omega_0 = 0.726, 
0.97, \text{ and } 1.33 \). Since past simulations have shown that switching from one
rotating frame to another has little if any effect on the dynamical evolution,
Model A was not allowed a relaxation period between each rotating frame
change.

As mentioned above, the curves plotted in Figure 5.1 begin to display
an oscillatory behavior — suggesting the onset of an instability — after
Model A has cooled by approximately 50\% \( t/\tau_{\text{dyn}} \approx 87 \), but these plots are
misleading. By the time the oscillations appear in Figure 5.1, an evolutionary
transition has already been apparent in the shape of the isodensity surfaces
and flow. (See Figs. 5.3 and 5.4.) The first clear indication of a growing instability appears in a plot of the radial location of the density maximum as a function of time, as shown here in Figure 5.2. The vertical lines at the bottom of this figure mark the same points of interest as in Figure 5.3. At the beginning of the cooling simulation, the radial density profile was relatively flat along the major axis (see Figure 4.8 for an illustration of how flat the bottom of the corresponding potential well was at this point in the evolution), so a change in density of a few percent was enough to shift the radial location of the density maximum by a few radial grid zones. Initially, this effect was not noticeable in the isodensity surfaces or in the fluid flow, but as the cooling continued, the oscillation increased in frequency and amplitude. Eventually, the radial density profile was no longer flat but developed a more “peaked” structure (see corresponding plots of $\Phi_{eff}$ in Fig. 5.5).

Shortly after cooling was resumed on the finer numerical grid ($t/\tau_{dyn} \approx 83$) Model A began oscillating noticeably between an ellipsoidal-like shape and a pronounced binary shape as illustrated by the two frames of Figure 5.3. But this, in and of itself was not proof that a binary system was forming. Remember that, as illustrated in Figure 4.4, during the phase of steady-state evolution, the isodensity surfaces of Model A also displayed an off-axis density maximum and hence, to some extent, resembled a binary. But in the steady-state model, the fluid was observed to be flowing through instead of around each density maximum (see Fig. 4.6). By contrast, Figure 5.4 illustrates the flow at two separate instances in time ($t/\tau_{dyn} = 87.4$ and 88.7) that correspond, respectively, to extremes in the oscillation that developed during
Figure 5.2: Plot of radial position of the density maximum as a function of time for the cooling simulation. The vertical marks at the bottom of the figure represent the same points of interest as those of Fig. 5.1. The step function quality of the plot is due to the coarseness of our numerical grid. The density maximum is not interpolated between grid points and is assumed to be at a specific grid point. Notice how the plot smooths out after the regridding process was performed at \( t/\tau_{dyn} \approx 80 \).
Table 5.1: Model A Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Initial</th>
<th>Steady-State Evolution</th>
<th>Cooling Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start</td>
<td>End</td>
</tr>
<tr>
<td>$T_{\text{time}}$</td>
<td>0.00</td>
<td>46.26</td>
<td>64.59</td>
</tr>
<tr>
<td>$T/</td>
<td>W</td>
<td>$</td>
<td>0.300</td>
</tr>
<tr>
<td>$T_{\text{rot}}/</td>
<td>W</td>
<td>$</td>
<td>0.300</td>
</tr>
<tr>
<td>$T_{\text{frm}}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{\text{int}}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$T_{\text{ra}}/</td>
<td>W</td>
<td>$</td>
<td>0.000</td>
</tr>
<tr>
<td>$\Pi/</td>
<td>W</td>
<td>$</td>
<td>0.200</td>
</tr>
<tr>
<td>$\delta$</td>
<td>1.66(-5)</td>
<td>3.97(-3)</td>
<td>6.73(-3)</td>
</tr>
<tr>
<td>$J$</td>
<td>1.22</td>
<td>0.881</td>
<td>0.843</td>
</tr>
<tr>
<td>$J_{\text{frm}}$</td>
<td>—</td>
<td>0.562</td>
<td>0.582</td>
</tr>
<tr>
<td>$J_{\text{int}}$</td>
<td>—</td>
<td>0.319</td>
<td>0.109</td>
</tr>
<tr>
<td>$M$</td>
<td>1.000</td>
<td>0.916</td>
<td>0.909</td>
</tr>
<tr>
<td>$\rho_{\text{max}}$</td>
<td>4.94(-3)</td>
<td>6.21(-3)</td>
<td>6.27(-3)</td>
</tr>
<tr>
<td>$R_{\text{eq}}$</td>
<td>10.95</td>
<td>9.19</td>
<td>8.37</td>
</tr>
<tr>
<td>$R_{\text{max}}/R_{\text{eq}}$</td>
<td>0.000</td>
<td>0.123</td>
<td>0.142</td>
</tr>
<tr>
<td>$c/a$</td>
<td>0.208</td>
<td>0.218</td>
<td>0.235</td>
</tr>
<tr>
<td>$b/a$</td>
<td>1.000</td>
<td>0.540</td>
<td>0.605</td>
</tr>
<tr>
<td>$f$</td>
<td>—</td>
<td>1.26</td>
<td>1.21</td>
</tr>
<tr>
<td>$\varpi_0$</td>
<td>—</td>
<td>0.726</td>
<td>1.391</td>
</tr>
</tbody>
</table>

the cooling evolution. When the configuration is shaped like a common-envelope binary, one can clearly see that a portion of the flow is around not through each density maximum. Furthermore, as the plot of $\Phi_{\text{eff}}$ in Figure 5.5 shows, a pronounced off-axis minimum in the global potential is now associated with both off-axis density maxima when the configuration takes on the appearance of a binary. This has convinced us that the system is oscillating between an ellipsoidal and a true binary configuration.

The right-hand column of Table 5.1 lists intrinsic properties of the instantaneous binary configuration that has been illustrated in the bottom frames.
of Figures 5.3, 5.4, and 5.5. (The first four columns of Table 5.1 are copied from Table 4.1 for purposes of comparison.) As in Tables 4.1 and 4.2, the values listed here represent only the material within the primary (binary) configuration, which we have defined as all numerical grid points with densities \( \rho \geq 0.05\rho_{\text{max}} \). Due to a small amount of mass shedding that occurs during each oscillation, the total mass of the object has been reduced by approximately 9% from its value at the beginning of the cooling evolution. Similarly, the loss of some high specific angular momentum material has reduced the total angular momentum of the object by 18%. While the total angular momentum has decreased, the radial extent of the pattern has also decreased, (from \( R_{\text{eq}} \approx 8.4 \) to 6.5), so, overall, the pattern frequency of the object has increased. Notice also that the virial error \( S \) has increased an order of magnitude, reflecting the fact that this configuration is not in a true steady-state.

The oscillation of Model A between binary and ellipsoidal "states" is indicative of oscillations between two relative minima of the system's global free energy. Model A has enough available kinetic energy to move freely between both relative minimum states even though there is likely to be a free energy barrier formally separating the two. Since each energy minimum is itself a function of time due to the cooling of Model A, at one particular instant in time we stopped the cooling process and allowed the model to settle into its "preferred" energy minimum for the present value of \( K \) in order to determine the true ground state of Model A. The instance at which cooling was halted is denoted in Figures 5.1 and 5.2 by the heavy line that is furthest
Figure 5.3: Same as Fig. 4.4 for Model A in the ellipsoidal and binary states of the binary instability oscillation.
Figure 5.4: Same as Fig. 4.6 for Model A in the ellipsoidal and binary states of the binary instability oscillation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
to the right at the bottom of the figure. The model continues to "ring" from the instability long after the cooling has been halted. However, the oscillation between the binary and ellipsoidal state does gradually decline in amplitude over 10 dynamical times, leaving a bar-like structure similar to the one that was present during the earlier steady-state phase of the evolution.

The analytical results discussed in Chapter 1 suggest that, eventually, as Model A cools, the energy minimum will shift towards the binary configuration. In an attempt to follow the cooling evolution completely to fission, we have further cooled Model A beyond the point in time shown in Figure 5.1. This extended cooling simulation proceeded as before, with the binary instability growing in amplitude and the oscillation frequency remaining approximately twice the orbital frequency, while the overall size of the system continued to contract. We were able to reduce the polytropic constant $K$ by a factor of 72% overall before reaching a point in the simulation where it became clear that another grid interpolation would have to be performed in order to again achieve a satisfactory spatial resolution. We chose to end the simulation at this point rather than continue through a second episode of regridding.

The increasing severity of the oscillations observed in this extended cooling simulation strongly suggests that the end state of Model A due to cooling will ultimately be a binary system, as hypothesized by Lebovitz (1987). In Table 5.1, if we compare the distance between the density maxima to the length of the major axis at the beginning of the cooling evolution and during our binary snapshot, the ratio has more than doubled. Hence, as the
Figure 5.5: $\Phi_{\text{eff}}$ surface in the equatorial plane of Model A in the ellipsoidal and binary states of the binary instability oscillation.
binary is forming, the local density maxima are migrating toward the edges of the object. In the time snapshot of the binary recorded in Table 5.1, the distance between the density maxima is nearly equal to the corotation radius (see Figs. 5.3 and 5.4). That is, if the total mass of the system were divided equally into two point masses and located at the positions of the density maxima with an orbital frequency of the pattern, they would be in an approximately circular orbit. Therefore, it indeed appears that the binary instability is pushing Model A towards an equal-mass binary system with an approximately circular orbit.

5.3 Comparisons with Observations

Although we have followed the phase of thermal contraction on an accelerated timescale, we are still nowhere near the end of the phase of Kelvin-Helmholtz contraction to the main sequence. If we assume that, at the beginning of the cooling simulation, Model A has a total mass of 1 M⊙ and a rotation period of one year (see the top row of Table 4.3), the average density will have to increase by 7 orders of magnitude (to $\rho \sim 1$ g cm$^{-3}$) in order to reach densities sufficient for nuclear fusion. If the total mass is conserved, this is analogous to a reduction in radius by a factor of approximately 200, whereas we have only been able to follow the cooling evolution of Model A such that the model contracts to approximately half its original extent. So, it is not possible for us to follow Model A completely to stellar densities. But it seems clear that fission of the object into two new (less massive) protostellar objects that are in orbit about one another will occur long before the system
has reached a size and density sufficient to promote the burning of nuclear fuel.

Now, if we assume that the properties of binary protostellar systems that will ultimately arise from Model A are similar to the properties portrayed by a "snapshot" of the binary during our cooling simulation — i.e., the properties listed in the last column of Table 5.1 — we can draw a comparison between our Model A binary and observed binary systems. Columns 2-4 of Table 5.3 list key parameters for two observed systems as drawn from Table 2 in the review by Mathieu (1994). The entries used are for the two equal mass binary systems, *HD155555* and *W134*. The angular momenta of the observed systems (column 5) were calculated assuming each binary system was composed of two point masses each containing half the total mass of the system in a circular orbit about one another. For comparison, the last two columns of Table 5.3 contain values of $a$ and $J$ that we have obtained by scaling the binary snapshot of Model A, as profiled in column 4 of Table 5.1, to match the total mass and orbital period of the two observed binary systems. The major axis $a$ of our binary model is taken to be the distance between the density maxima.

Even though we have made simplistic assumptions about our numerical model for these comparisons, the fact that the results agree to within a factor of two is very reassuring. The major axis of the orbit for the system *HD155555* is only off by about 10%! The binary systems we are able to form should provide much insight for astronomers observing these protobinary systems. As was discussed at the end of Chapter 4, within the next
Table 5.2: Protobinary Systems

<table>
<thead>
<tr>
<th>Object</th>
<th>Mass($M_\odot$)</th>
<th>Period(days)</th>
<th>a(AU)</th>
<th>$J$(g cm$^2$s$^{-1}$)</th>
<th>a(AU)</th>
<th>$J$(g cm$^2$s$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD155555</td>
<td>0.964</td>
<td>3.20</td>
<td>1.68</td>
<td>0.013</td>
<td>6.35</td>
<td>0.099</td>
</tr>
<tr>
<td>W184</td>
<td>3.20</td>
<td>6.35</td>
<td>0.099</td>
<td>8.08(52)</td>
<td>0.053</td>
<td>3.35(52)</td>
</tr>
</tbody>
</table>
decade, the tools needed to observe more of these types of systems in detail will become available to astronomers and, serendipitously, so will the computational means to create models to explain and predict what will be observed.
6. CONCLUSION

Lebovitz (1987) has suggested that as rotating, inviscid protostellar gas clouds evolve toward the main sequence through a relatively slow, quasi-adiabatic contraction, they may experience a global fission instability and thereby explain in a natural way why a very large fraction of stars reside in short period binary systems. Specifically, as we have detailed in Chapter 1, Lebovitz has suggested that when an initially axisymmetric cloud in thermal equilibrium (positioned schematically between points M and D in Fig. 3.2) slowly contracts to a configuration at or just below the point marked D in Figure 3.2, it should spontaneously deform into a triaxial configuration that resembles a Riemann type ellipsoid. Upon further slow contraction, this triaxial configuration should encounter an instability that leads to fission, i.e., the formation of two gaseous clumps in orbit about one another. In this dissertation we have performed dynamical simulations that, for the first time, critically test Lebovitz's "modified fission hypothesis."

As indicated by the small solid squares and triangles in Figure 3.2, and as described in detail in Chapter 3, we initially constructed two models of rapidly rotating axisymmetric gas clouds that were rotationally flattened enough to be susceptible to the $m = 2$, dynamical bar-mode instability. As discussed in detail in Chapter 4, via a three-dimensional hydrodynamics simulation, we followed in a fully self-consistent fashion, the nonlinear development of this instability to the point where both models settled into steady-state triaxial configurations that, by all accounts, were compressible
Figure 6.1: Plot of $T_{rot}/|W|$ and $\Pi/|W|$ as a function of time for the complete evolution of Model A. The vertical line farthest to the left at the bottom of the figure marks the beginning of the evolution in the rotating frame, the middle vertical line marks the beginning of the cooling evolution, and the vertical line farthest to the right marks the end of the cooling evolution.

analogs of the Riemann ellipsoids lying along the DO sequence shown in Figure 3.2. We then slowly cooled Model A and discovered that it did naturally encounter an instability that appears to lead to the formation of a binary system, as predicted by Lebovitz.

By way of summary, the complete evolution of Model A — from an axisymmetric, triaxial configuration to a steady-state configuration, then to a binary configuration — may be followed in Figure 6.1 where, for brevity, only the $T_{rot}/|W|$ and $\Pi/|W|$ parameters have been plotted. The large oscillations
in the $T_{rot}/|W|$ parameter near the beginning of the evolution are a signature that the initial axisymmetric model is suffering the dynamical instability that led to the formation of a steady-state, triaxial model. The vertical line farthest to the left at the bottom of the figure marks the beginning of the steady-state, triaxial evolution in the rotating frame; the middle vertical line marks the point at which cooling was begun; and the short vertical line farthest to the right marks the point at which cooling was stopped. Near the end of the evolution, small amplitude oscillations in the $T_{rot}/|W|$ parameter signify the onset of the binary instability. Figure 6.2 shows a picture of the three-dimensional isodensity surfaces of Model A in the binary state. (This object has also been illustrated in Figs. 5.3, 5.4, and 5.5.) These detailed simulations provide the strongest evidence, to date, that short period binary stars are formed via a natural fission instability.

A secondary success of these simulations has been the demonstration that steady-state, triaxial configurations can be constructed from compressible gases. Specifically, as we have described in detail in Chapter 4, both of our Models A and B evolve to configurations that are unchanging on a dynamical timescale and exhibit complex internal flows. Features common to both models are standing shock fronts, off-axis density maxima, bar-shaped isodensity surfaces (as illustrated in Figs. 4.3 and 4.4), and highly nonaxisymmetric flow fields (as illustrated in Figs. 4.5 and 4.6). Both models may be envisioned as compressible analogs of Riemann ellipsoids.

Although recent studies (Chambat 1994, Fillippi & Sepulveda 1996) have suggested that compressible analogs of Riemann ellipsoids do not exist, we
Figure 6.2: Image of three-dimensional isodensity surfaces of the binary state of the binary instability oscillation. The density levels are at $\rho/\rho_{max} = 0.004$ (blue), 0.04 (red), 0.4 (yellow), and 0.8 (green).
believe these conclusions have resulted mainly from the unnatural constraints that have been placed on these earlier compressible models. The key differences between our final triaxial models and those described in the previous studies are:

- Previous configurations have been restricted to ellipsoidal configurations or configurations with symmetries about the principal axes, whereas our final models exhibit neither geometric restriction.

- The flow in the earlier studies has been limited to incompressible flows, (i.e., \( \nabla \cdot \vec{u} = 0 \)), whereas our models exhibit a compressible flow that even contains a mild standing shock front.

- The fluid vorticity in earlier studies was forced to be aligned with the axis of rotation whereas, in our models, the vorticity is slightly misaligned with the axis of rotation.

Scaling our numerical models to physical dimensions has allowed us to check their validity. When Model A's steady-state triaxial configuration is scaled such that its total mass and pattern period match those of systems that are at most twice the size of the solar system, the rescaled average density and temperature of the gas cloud match well with what is expected from star formation theory (Shu et al. 1987). (See Table 4.3 in Chapter 4.) Although our primary interest in these steady-state models has been in connection with star formation processes, we have shown that these models also relate to other astrophysical systems in which self-gravity and rotation
are important, such as, the gaseous bars in the center of many spiral galaxies and rapidly rotating compact stellar objects.

Additional advances in numerical techniques will have to be realized before models of the fission instability can be used to match detailed observations of protostellar gas clouds over the coming decade. For example, adaptive mesh refinement techniques would allow the simulation to automatically adjust the spatial resolution of the numerical grid as the model contracts; radiation diffusion algorithms would allow the model to cool via a more realistic process; and there are different numerical techniques that, although very computationally expensive, would allow the model to evolve on a thermal timescale. However, the present simulations represent a milestone in the sense that they demonstrate the full feasibility of the fission hypothesis for the first time.
REFERENCES

Andalib, S. W. 1998, Ph.D. Thesis, Louisiana State University, Baton Rouge


Cohl, H. S. 1999, Ph.D. Thesis, Louisiana State University, Baton Rouge


Maclaurin, C. 1742, A Treatise on Fluxions (Edinburgh: T. W. & T. Ruddimans)
Uryu, K., & Eriguchi, Y. 1996, MNRS, 282, 653

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX A: HYDROCODE

C----------------------------------------------------------------
C  arslop(isyma)
C-----------------------------------------------
C----------------------------------------------------------------
subroutine arslop(isyma)
  include 'grid.h'
  
cjc  This subroutine calculates slopes between grid cells

  real, dimension (jmax2, kmax2, lmax) :: phi, rho
  !hpf$ align with hydtpl :: phi, rho
  common /pois/ phi, rho

  real, dimension (jmax2, kmax2, lmax) :: p, eps
  !hpf$ align with hydtpl :: p, eps
  common /states/ p, eps

  real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
  !hpf$ align with hydtpl :: s, t, a, u, w, jn
  common /som/ s, t, a, u, w, jn

  real, dimension (jmax2, kmax2, lmax) :: aj, ak, al, rj, rl, ej, el
  !hpf$ align with hydtpl :: aj, ak, al, rj, rl, ej, el
  common /slopes/ aj, ak, al, rj, rl, ej, el

  integer isyma
  real, dimension (jmax2, kmax2, lmax) :: qp1, qm1, qdiff
  !hpf$ align with hydtpl :: qp1, qm1, qdiff

  real limit

  cjc  here we are initializing the variables used by this subroutine

  qp1 = 0.0
  qm1 = 0.0
  qdiff = 0.0
  limit = 1.0e-20

  c  here we are calculating the monotonic condition... if the central zone
  c  is a local extremum, set slope to zero.  also, if slope is zero, force
  c  it to zero (so as to avoid fp overflows...)

  c  what is calculated is the Van Leer second order slope.  A standard
  c  second order slope calculated from two Taylor expansions is unstable.
  c  What Van Leer found is that the harmonic mean of a forward difference
  c  and a backwards difference slope, coupled with conditions of
  c  monotonicity, results in a second order accurate, STABLE slope which
  c  can be used in fluxing material from one grid zone to the next
  c  (see woodward dissertation for sources).

  c  since the eoshift command will not allow a variable for a dimension,
  c  we have to handle each direction seperately.

  c***  slopes are for a, rho, and eps interpolated values.
slope(xl,xc,xr) = 2.0*(xr-xc)*(xc-xl)/(xr-xl)

c  determine all the slopes needed for the advection of a, rho, and eps.
cjc  call vslope(a, aj, ak, al)
    qpl = eoshift(a, dim=1, shift=1)
    qml = eoshift(a, dim=1, shift=-1)
    qdiff = qpl - qml
    aj = 2.0 * (qpl - a) * (a - qml)
    where (aj .lt. 0.0)
        aj = 0.0
        end where
    where (abs(qdiff) .le. limit)
        aj = 0.0
    elsewhere
        aj = aj / qdiff
    endwhere
    qpl = eoshift(a, dim=2, shift=1)
    qml = eoshift(a, dim=2, shift=-1)
    qdiff = qpl - qml
    ak = 2.0 * (qpl - a) * (a - qml)
    where (ak .lt. 0.0)
        ak = 0.0
        end where
    where (abs(qdiff) .le. limit)
        ak = 0.0
    elsewhere
        ak = ak / qdiff
    endwhere
    qpl = cshift(a, dim=3, shift=1)
    qml = cshift(a, dim=3, shift=-1)
    qdiff = qpl - qml
    al = 2.0 * (qpl - a) * (a - qml)
    where (al .lt. 0.0)
        al = 0.0
        end where
    where (abs(qdiff) .le. limit)
        al = 0.0
    elsewhere
        al = al / qdiff
    endwhere
cjc  call vslope(a, aj, ak, al)
cjc  call vslope(rho, rj, rk, rl)
    qpl = eoshift(rho, dim=1, shift=1)
    qml = eoshift(rho, dim=1, shift=-1)
    qdiff = qpl - qml
    limit = 1.0e-20
    rj = 2.0 * (qpl - rho) * (rho - qml)
    where (rj .lt. 0.0)
        rj = 0.0
end where
where (abs(qdiff) .le. limit)
   rj = 0.0
elsewhere
   rj = rj / qdiff
endwhere

qpi = eoshift(rho, dim=2, shift=1)
qmi = eoshift(rho, dim=2, shift=-1)
qdiff = qpi - qmi

rk = 2.0 * (qpi - rho) * (rho - qmi)
where (rk .lt. 0.0)
   rk = 0.0
end where
where (abs(qdiff) .le. limit)
   rk = 0.0
elsewhere
   rk = rk / qdiff
endwhere

qpi = cshift(rho, dirx=3, shift=l)
qml = cshift(rho, dirx=3, shift=-l)
qdiff = qpi - qml

rl = 2.0 * (qpi - rho) * (rho - qml)
where (rl .lt. 0.0)
   rl = 0.0
end where
where (abs(qdiff) .le. limit)
   rl = 0.0
elsewhere
   rl = rl / qdiff
endwhere
cjc   call vslope(rho, rj, rk, rl)
cjc   call vslope(eps, ej, ek, el)
qpi = eoshift(eps, dim=1, shift=1)
qmi = eoshift(eps, dim=1, shift=-1)
qdiff = qpi - qmi

ej = 2.0 * (qpi - eps) * (eps - qmi)
where (ej .lt. 0.0)
   ej = 0.0
end where
where (abs(qdiff) .le. limit)
   ej = 0.0
elsewhere
   ej = ej / qdiff
endwhere

qpi = eoshift(eps, dim=2, shift=1)
qmi = eoshift(eps, dim=2, shift=-1)
qdiff = qpi - qmi

ek = 2.0 * (qpi - eps) * (eps - qmi)
where (ek .lt. 0.0)
   ek = 0.0
end where
where (abs(qdiff) .le. limit)
  ek = 0.0
elsewhere
  ek = ek / qdiff
endwhere

qpl = cshift(eps, dim=3, shift=l)
qml = cshift(eps, dim=3, shift=-l)
qdiff = qpl - qml
el = 2.0 * (qpl - eps) * (eps - qml)
where (el .lt. 0.0)
  el = 0.0
end where
where (abs(qdiff) .le. limit)
  el = 0.0
elsewhere
  el = el / qdiff
endwhere

cjc call vslope(eps, ej, ek, el)
c
c take care of boundary condition slopes here!!!
c please note that only the equatorial symmetry combined with or without
c pi-symmetry case, and dirichlet (not
c wall) outer boundary conditions

ak(:,2,:) = 0.0
ak(:,1,:) = 0.0
ak(:,kmax,:) = 0.0
ak(:,kmax1,:) = 0.0
ak(:,kmax2,:) = 0.0

rk(:,2,:) = 0.0
rk(:,1,:) = 0.0
rk(:,kmax,:) = 0.0
rk(:,kmax1,:) = 0.0
rk(:,kmax2,:) = 0.0

ek(:,2,:) = 0.0
ek(:,1,:) = 0.0
ek(:,kmax,:) = 0.0
ek(:,kmax1,:) = 0.0
ek(:,kmax2,:) = 0.0

aj(jmax,:,:) = 0.0
aj(jmax1,:,:) = 0.0
aj(jmax2,:,:) = 0.0

rj(jmax,:,:) = 0.0
rj(jmax1,:,:) = 0.0
rj(jmax2,:,:) = 0.0

ej(jmax,:,:) = 0.0
ej(jmax1,:,:) = 0.0
ej(jmax2,:,:) = 0.0

if (isyma.ne.1 .and. isyma.ne.2) then
  aj(2,:,:) = 0.0
  aj(1,:,:) = 0.0
rj(2,:,:)= 0.0
rj(1,:,:)= 0.0
ej(2,:,:)= 0.0
ej(1,:,:)= 0.0
else
   aj(1,:,:)= 0.0
   rj(1,:,:)= 0.0
ej(1,:,:)= 0.0
endif
return
end

C----------------------------------------------------------------
C     stslp(isyma)
C----------------------------------------------------------------
subroutine stslp(isyma)
include 'grid.h'
cjc This subroutine calculates slopes between grid cells
cjc for s, and t.
   integer isyma

    real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
   common /pois/ phi, rho

    real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
   common /eom/ s, t, a, u, w, jn

    real, dimension (jmax2, kmax2, lmax) :: sj, sk, sl, tj, tk, tl
!hpf$ align with hydtpl :: sj, sk, sl, tj, tk, tl
   common /slop2/ sj, sk, sl, tj, tk, tl

    real, dimension (jmax2, kmax2, lmax) :: qpl, qml, qdiff
!hpf$ align with hydtpl :: qpl, qml, qdiff

    real limit

cjc here we are initializing the variables used by this subroutine
    qpl = 0.0
    qml = 0.0
    qdiff = 0.0
    limit = 1.0e-20

c here we are calculating the monotonic condition... if the central zone
    c is a local extremum, set slope to zero. also, if slope is zero, force
    c it to zero (so as to avoid fp overflows...)

c what is calculated is the Van Leer second order slope. A standard
    c second order slope calculated from two Taylor expansions is unstable.
c What Van Leer found is that the harmonic mean of a forward difference
    c and a backwards difference slope, coupled with conditions of
    c monotonicity, results in a second order accurate, STABLE slope which
    c can be used in fluxing material from one grid zone to the next
    c (see woodward dissertation for sources).

c since the eoshift command will not allow a variable for a dimension,
c we have to handle each direction separately.
c calculate slopes for s and t

cjc call vslope(s, sj, sk, sl)
    qpl = eoshift(s, dim=1, shift=1)
    qml = eoshift(s, dim=1, shift=-1)
    qdiff = qpl - qml

    sj = 2.0 * (qpl - s) * (s - qml)
    where (sj .lt. 0.0)
        sj = 0.0
    end where
    where (abs(qdiff) .le. limit)
        sj = 0.0
    elsewhere
        sj = sj / qdiff
    endwhere

    qpl = eoshift(s, dim=2, shift=1)
    qml = eoshift(s, dim=2, shift=-1)
    qdiff = qpl - qml

    sk = 2.0 * (qpl - s) * (s - qml)
    where (sk .lt. 0.0)
        sk = 0.0
    end where
    where (abs(qdiff) .le. limit)
        sk = 0.0
    elsewhere
        sk = sk / qdiff
    endwhere

    qpl = cshift(s, dim=3, shift=1)
    qml = cshift(s, dim=3, shift=-1)
    qdiff = qpl - qml

    sl = 2.0 * (qpl - s) * (s - qml)
    where (sl .lt. 0.0)
        sl = 0.0
    end where
    where (abs(qdiff) .le. limit)
        sl = 0.0
    elsewhere
        sl = sl / qdiff
    endwhere

cjc call vslope(t, tj, tk, tl)
    qpl = eoshift(t, dim=1, shift=1)
    qml = eoshift(t, dim=1, shift=-1)
    qdiff = qpl - qml

    tj = 2.0 * (qpl - t) * (t - qml)
    where (tj .lt. 0.0)
        tj = 0.0
    end where
    where (abs(qdiff) .le. limit)
        tj = 0.0
    elsewhere
        tj = tj / qdiff
end where

qpl = eoshift(t, dim=2, shift=1)
qml = eoshift(t, dim=2, shift=-1)
qdiff = qpl - qml

tk = 2.0 * (qpl - t) * (t - qml)
where (tk .lt. 0.0)
tk = 0.0
end where
where (abs(qdiff) .le. limit)
tk = 0.0
elsewhere
  tk = tk / qdiff
endwhere

qpl = cshift(t, dim=3, shift=1)
qml = cshift(t, dim=3, shift=-1)
qdiff = qpl - qml
tl = 2.0 * (qpl - t) * (t - qml)
where (tl .lt. 0.0)
tl = 0.0
end where
where (abs(qdiff) .le. limit)
tl = 0.0
elsewhere
  tl = tl / qdiff
endwhere

cjc call vslope(t, tj, tk, tl)
c
c take care of boundary condition slopes here!!!!
c
c please note that these take care of only equatorial symmetry
c combined with or without pi-symmetry case and

c dirichlet (not wall) outer boundary conditions.
c
  sk(:,2,:) = 0.0
  sk(:,1,:) = -sk(:,3,:)
  sk(:,kmax1,:) = 0.0
  sk(:,kmax2,:) = 0.0

tk(:,2,:) = t(:,3,:)
tk(:,1,:) = tk(:,3,:)
tk(:,kmax1,:) = 0.0
  tk(:,kmax2,:) = 0.0

  sj(jmax,:, :) = 0.0
  sj(jmax1,:, :) = 0.0
  sj(jmax2,:, :) = 0.0

  tj(jmax1,:, :) = 0.0
  tj(jmax2,:, :) = 0.0

if (isyma.ne.1 .and. isyma.ne.2) then
  sj(2,:, :) = s(3,:, :)
sj(1,:, :) = sj(3,:, :)
tj(2,:, :) = 0.0
tj(1,:,:) = -tj(3,:,:)  
else  
sj(1,:,:) = 0.0  
tj(1,:,:) = 0.0  
endif  
return  
end

C-------------------------------------------------------
C  delta(itstrt, itstop)
C-------------------------------------------------------

subroutine delta(itstrt, itstop)

c This routine calculates the time increment to use in the fluid evolution.
c We use the Courant-Frederichs-Lesewy condition -> Information should not
c be allowed to traverse more than a single grid cell in a time step.
c Therefore, the maximum allowable time-step is related to the grid spacing
c and the sum of sound speed and a velocity.

include "grid.h"

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtp1 :: phi, rho
common /pois/ phi,rho

real, dimension (jmax2, kmax2, lmax) :: r,z,rhf,zhf
!hpf$ align with hydtp1 :: r,z,rhf,zhf
common /grid/ r,z,rhf,zhf

real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpf$ align with hydtp1 :: rplus,zplus,rhfminus,zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtp1 :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtp1 :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn

real :: rcloud,constp,delt,bdytem,den,time,cormas
common /blok7/ rcloud,constp,delt,bdytem,den,time,cormas

real :: delt, deltax
common /jgrid2/ delt, deltax

integer :: indx, isoadi, itstep
real :: allow
common /timest/ indx, isoadi, allow, itstep

real :: xn, xni, konst, xnyd
common /ptrope/ xn, xni,konst,xnyd

real :: rco, tco, zco, vrco, vtco, vzco
common /codefs/ rco, tco, zco, vrco, vtco, vzco

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real :: dmax0
common /fdelta/ dmax0

real :: rof3n, zof3n, ainewr, ainewz, dtheta
common /grid2/ rof3n, zof3n, ainewr, ainewz, dtheta

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

integer :: jl, kl, l1, itstrt, itstop
integer locmin(3)
real :: velco, chgmax, ssmx
real :: amin, dmax, denlim
real :: factor, gamma, adiso
real sp(4)

!hpf$ align with hydtpl :: vmr, vmz, vml, vmlz
real, dimension (jmax2,kmax2,lmax) :: vmr, vmz, vml, vmlz
!hpf$ align with hydtpl :: sndspd, jvel, kvel, lvel
real, dimension (jmax2,kmax2,lmax) :: sndspd, jvel, kvel, lvel
!hpf$ align with hydtpl :: deltj, deltk, delt1
real, dimension (jmax2,kmax2,lmax) :: deltj, deltk, delt1

!cjc here i am initializing the variables used
!gamma = 0.0
!adiso = 0.0
ji = 0
kl = 0
li = 0
velco = 0.0
chgmax = 0.0
sndspd = 0.0
jvel = 0.0
kvel = 0.0
lvel = 0.0
deltj = 0.0
deltk = 0.0
delt1 = 0.0
amin = 0.0
dmax = 0.0
denlim = 0.0
sp = 0.0
locmin = 0

!c factor = fraction of courant time to use
!allow = max fractional change in maximum density (not used)
factor = 0.4
allow = 0.15

gamma = xnhyd

adiso = 1.0
if (isoadi .eq. 2 .or. isoadi .eq. 3) adiso = gamma
if (itstep .eq. 1) dmax0 = den

!c
c calculate 4 speeds: sound speed, 3 fluid speeds (rad, z, ang).
c First we set dummy values of the sound speed and the times,
c Then calculate the velocities in the three directions,
c Finally, calculate the delta t allowed in each direction.
c
sndspd = 0.0
deltj = 10000.0
deltk = 10000.0
deltl = 10000.0

where (rho .ne. 0.0)
    sndspd = sqrt(adiso*p / rho )
endwhere

ssmx = maxval(sndspd(2:jmax,2:kmax,:))

lvel = abs(jn/rhf)

vmr = (eoshift(u, dim=1, shift=1))
vmz = (eoshift(u, dim=2, shift=1))
vml = (cshift(u, dim=3, shift=1))
vmlz = eoshift(vml, dim=2, shift=1)

where (rho.gt.denlim)
    jvel = absCC ((u+vmz+vml+vmlz)*(rhf**2-r**2) +
    (vmr+eoshift(vmr, dim=1, shift=1)
    +eoshift(vml, dim=1, shift=1)
    +eoshift(vmlz, dim=1, shift=1))*(rplus**2-rhf**2) )
    /((4.0*(rplus**2-r**2)))
elsewhere
    jvel = 0.5*ssmx
endwhere

vmr = (eoshift(w, dim=1, shift=1))
vmz = (eoshift(w, dim=2, shift=1))
vml = (cshift(w, dim=3, shift=1))
vmlz = eoshift(vml, dim=2, shift=1)

where (rho.gt.denlim)
    kvel = absCC ((w+vmz+vml+vmlz)*(rhf**2-r**2) +
    (vmr+eoshift(vmr, dim=1, shift=1)
    +eoshift(vml, dim=1, shift=1)
    +eoshift(vmlz, dim=1, shift=1))*(rplus**2-rhf**2) )
    /((4.0*(rplus**2-r**2)))
elsewhere
    kvel = 0.5*ssmx
endwhere

where (sndspd + jvel .ne. 0.0)
    deltj = deltar/(sndspd + jvel)
endwhere

where (sndspd + kvel .ne. 0.0)
    deltik = deltar/(sndspd + kvel)
endwhere

where (sndspd + lvel .ne. 0.0)
    deltik = rhf*dtheta/(sndspd + lvel)
endwhere

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
c deltj on axis non-meaningful, really... set to a dummy

deltj(2,:,i) = deltj(2,:,i)

c Now that we have maximum allowed delta t for every grid cell, we just need
c to determine the minimum of these delta t's for our time step.
c Get minimum radial delta t. Compare with vertical delta t, then
with delta t from angular direction. Result is our delta t

amin = minval(deltj(2:jmax,2:kmax,:))
locmin = minloc(deltj(2:jmax,2:kmax,:))

if (amin > minval(deltk(2:jmax,2:kmax,:))) then
    amin = minval(deltk(2:jmax,2:kmax,:))
    locmin = minloc(deltk(2:jmax,2:kmax,:))
endif

if (amin > minval(deltl(2:jmax,2:kmax,:))) then
    amin = minval(deltl(2:jmax,2:kmax,:))
    locmin = minloc(deltl(2:jmax,2:kmax,:))
endif

c find zone this occurred in (add one because of 2:xxx above...c
has effect of offsetting location by 1).

ji = locmin(1)+1
ki = locmin(2)+1
li = locmin(3)

c no longer to chopping described below.
c... now chop this courant-determined time by density change criterion.
c get the maximum density so we can print out change. Set dmax0 to
c current dmax for next time through. Also get the velocities from
c the zone which limited the time step.

dmax = maxval(rho)
chgmax=abs(1.0-dmax0/dmax)
dmax0 = dmax
sp(1) = sndspd(ji,ki,li)
sp(2) = jvel(ji,ki,li)
sp(3) = kvel(ji,ki,li)
sp(4) = lvel(ji,ki,li)
delt = factor * amin

c does c.o. limit the time step?
c cartesian style, vtco = same units as vrco

velco = sqrt(vrco*vrco + vtco*vtco + vzco*vzco)
if (velco*delt > deltar) then
    delt = factor*(deltar)/velco
write(6,*)' time step limited by c.o., v = ',velco
endif

c write out the time step, limiting zone, speeds, and change.

write(6,100) delt,ji,ki,li,sp,allow,chgmax
100 format(10x,lp,e!3.5,3i5,5x,lp,4el3.5,5x,lp,2el0.2)
return
C This routine sets up the variables for the fluxing of rho, a, and eps. 
C It uses the generic fluxing routine in 'flux.f'.
C
C ***** uses van leer second order monotonic scheme *********
C
include "grid.h"
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi,rho
real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
real, dimension (jmax2, kmax2, lmax) :: si, ti, ai, rhoi, epsi
!hpf$ align with hydtpl :: si, ti, ai, rhoi, epsi
common /eom1/ si, ti, ai, rhoi, epsi
real, dimension (jmax2, kmax2, lmax) :: aj, ak, al, rj, rk, rl, ej, ek, el
!hpf$ align with hydtpl :: aj, ak, al, rj, rk, rl, ej, ek, el
common /slopes/ aj, ak, al, rj, rk, rl, ej, ek, el
real, dimension (jmax2, kmax2, lmax) :: sj, sk, sl, tj, tk, tl
real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydtpl :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf
real :: deltar, deltaz
common /jgrid2/ deltar, deltaz
real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta
real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpf$ align with hydtpl :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ p, eps
real, dimension (jmax2, kmax2, lmax) :: voln, wh, v, ug
!hpf$ align with hydtpl :: voln, wh, v, ug
real, dimension (jmax2, kmax2, lmax) :: transu, transw, transv
!hpf$ align with hydtpl :: transu, transw, transv
real, dimension (jmax2, kmax2, lmax) :: retflx, tvel1, tvel2
!hpf$ align with hydtpl :: retflx, tvel1, tvel2
real, dimension (jmax2, kmax2, lmax) :: rettot
!hpf$ align with hydtpl :: rettot

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /bloc7/ rcloud, constp, delt, bdytem, den, time, cormas

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: symfact
common /symmetry/ symfact

real :: mlost, jlost
common /outer/ mlost, jlost

real :: omgfrm
common /rotfrm/ omgfrm

integer itime, j, k

real :: outervoln

c **** calculate van linear monotonic interpolated values
** slopes determined before reaching this point!!!
c
vlil(xl, xs) = xl + 0.5*xs
c
vlir(xr, xs) = xr - 0.5*xs
c
c voln = 1/volume of grid cells

voln = 1.0/(0.5*dtheta*(rplus**2 - r**2) * deltaz)
c calculate face center velocities:
c 
ug is radial velocity, vertex centered (j+1, k, k+1, l, l+1 involved)
c so ug(j,k,l) refers to the velocity on the outward face.
c 
wh is z velocity, w is vertex centered (k+1, j, j+1, l, l+1 involved)
c so wh(j,k,l) refers to the velocity on the upward face.
c 
v is vel in ang. dir., jn is cell centered (1, l+1 involved)
c v(j,k,l) refers to the velocity on the increasing theta face.

tvel1 = eoshift(w, dim=2, shift=1)
tvel2 = eoshift(tvel1, dim=1, shift=1)
wh = 0.25 * ( tvel1 + cshift(tvel1, dim=3, shift=1)
 & + tvel2 + cshift(tvel2, dim=3, shift=1))
v = 0.5* ( cshift(jn, dim=3, shift=1) + jn ) / rhf

tvel1 = eoshift(u, dim=1, shift=1)
tvel2 = eoshift(tvel1, dim=2, shift=1)
ug=0.25* ( tvel1 + cshift(tvel1, dim=3, shift=1)
 & +tvel2 + cshift(tvel2, dim=3, shift=1))

The following bc work for dirichlet bc on side and top

to keep stuff from coming back onto grid and to allow
flow off of the grid
ug(jmax,:,:)=0.5*(abs(ug(jmax-1,:,:))+ug(jmax-1,:,:))
ug(jmax1,:,:)=0.0
ug(jmax2,:,:)=0.0
ug(:,kmax2,:) = 0.0
wh(:,kmax,:) = 0.5*(abs(wh(:,kmax-1,:))+wh(:,kmax-1,:))
wh(:,kmax1,:) = 0.0
wh(:,kmax2,:) = 0.0
wh(jmax2,:) = 0.0

c transx is the velocity times area of a face. calculate this for all
c faces.

transu = ug * rplus * dtheta * deltaz
transw = wh * 0.5 * dtheta * (rplus**2 - r*r)
transv = v * deltaz * deltar

c when we update density, etc, on the first call we use "current" values
c as we are working toward velocity(t+dt/2). on the second time, we
c use saved parameters, as we are using the updated velocity to update
c the original parameters.

c fluxing of rho
call flux(rho, rj, transu, delt, voln, 1, retflx)
rettot = retflx
call flux(rho, rk, transw, delt, voln, 2, retflx)
rettot = rettot + retflx
call flux(rho, rl, transv, delt, voln, 3, retflx)
rettot = rettot + retflx
if (itime .eq. 1) then
  rho = rho + rettot
else
  rho = rho + rettot
endif

c fluxing of a
call flux(a, aj, transu, delt, voln, 1, retflx)
rettot = retflx
call flux(a, ak, transw, delt, voln, 2, retflx)
rettot = rettot + retflx
call flux(a, al, transv, delt, voln, 3, retflx)
rettot = rettot + retflx
if (itime .eq. 1) then
  a = a + rettot
else
  a = a + rettot
endif

c fluxing of eps
call flux(eps, ej, transu, delt, voln, 1, retflx)
rettot = retflx
call flux(eps, ek, transw, delt, voln, 2, retflx)
rettot = rettot + retflx
call flux(eps, el, transv, delt, voln, 3, retflx)
rettot = rettot + retflx
if (itime .eq. 1) then
  eps = eps + rettot
else
  eps = eps + rettot
endif
mlost = mlost + symfact*sum((rho(jmaxl,2:kmax,:)-denex) / voln(jmaxl,2:kmax,:))
mlost = mlost + symfact*sum((rho(2:jmax,kmaxl,:)-denex) / voln(2:jmax,kmaxl,:))
jlost = jlost + symfact*sum((rho(jmaxl,2:kmax,:)-denex) / voln(jmaxl,2:kmax,:))
jlost = jlost + symfact*sum((rho(2:jmax,kmaxl,:)-denex) / voln(2:jmax,kmaxl,:))
2 *(jn(jmaxl,2:kmax,:)+ rhf(jmaxl,2:kmax,:)**2*omgfrm))
jlost = jlost + symfact*sum((rho(2:jmax,kmaxl,:)-denex) / voln(2:jmax,kmaxl,:))
jlost = jlost + symfact*sum((rho(2:jmax,kmaxl,:)-denex) / voln(2:jmax,kmaxl,:))
2 *(jn(2:jmax,kmaxl,:)+ rhf(2:jmax,kmaxl,:)**2*omgfrm))

rho(jmaxl,:,:)=denex
rho(:,kmaxl,:)=denex
a(jmaxl,:,:)=0.0
a(:,kmaxl,:)=0.0

return
end

---------------------------------------------------------------------
C efxtk2(isyma, sidbdy)
---------------------------------------------------------------------
subroutine efxtk2(isyma, sidbdy)
    include "grid.h"

    real, dimension (jmax2, kmax2, lmax) :: phi, rho
    !hpf$ align with hydtpl : :  phi, rho
    common /pois/ phi, rho
    real, dimension (jmax2, kmax2, lmax) :: p, eps
    !hpf$ align with hydtpl : :  p, eps
    common /states/ p, eps
    real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
    !hpf$ align with hydtpl : :  r, z, rhf, zhf
    common /grid/ r, z, rhf, zhf
    real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
    !hpf$ align with hydtpl : :  s, t, a, u, w, jn
    common /eom/ s, t, a, u, w, jn
    !hpfs align with hydtpl : :  s, t, a, u, w, jn
    common /blok6b/ pi, grav
    common /normal/ cirp
    real :: rof3n, zof3n, alnewr, alnewz, dtheta
    common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta
    real :: denex, epsmin, epsmax
    common /freez/ denex, epsmin, epsmax
    real :: xn, xnl, konst, xnyd
    common /ptrope/ xn, xnl, konst, xnyd
real :: omgfrm
common /rotfrm/ omgfrm

!hpf$ align with hydtpl :: dphi, omgkep

integer :: isyma, k
integer :: mxrhoat(3)
integer :: jdmx, kdmx, ldmx
character*6 sidbdy
real power
real :: gamma, d, fudge, rotmax, xnorm

gamma = xnhyd
power = 1.O/gamma

... set limits on rho and eps.

where ( rho .le. denex)  
  rho = denex  
  eps = (epsmin*denex)**power
endwhere

where ( eps .le. (epsmin*denex)**power)  
  eps = (epsmin * denex)**power
endwhere

... make eps = internal energy per unit volume.
eps = eps ** xnhyd

... make eps = internal energy per gram again
eps = eps/rho

... force j=2 zone to be axisymmetric
xnorm=1.0/float(lmax)
!hpf$ independent
forall(k=1:kmax2)
  rho(2,k,:) = sum(rho(2,k,:), dim = 1)*xnorm
  a(2,k,:) = sum( a(2,k,:), dim = 1)*xnorm
end forall

c... redefine omega and jn... and ...
c... to prevent time step from being controlled by high angular
velocities in low density regions, set jn and omega to zero
if rho.1e.d and if omega.gt.fudge*(central omega).
d=1.0e3*denex
fudge = 0.40
mxrhoat = maxloc(rho(2:jmax,2:kmax,:))
jdmx=mxrhoat(1)+1
kdmx=mxrhoat(2)+1
ldmx=mxrhoat(3)
rotmax = fudge*abs(jn(jdmx,kdmx,ldmx)/rhf(jdmx,kdmx,ldmx)**2+1
1 + omgfrm)
c redefine jn

dphi = eoshift(phi,dim = 1,shift = 1) - eoshift(phi,dim = 1,shift = -1)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[
d\phi = \frac{0.5}{\text{ref}3\text{n}}d\phi
\]
where
\[
\phi = \frac{d\phi}{0.0} \quad \text{if } (d\phi > 0.0)
\]
elsewhere,
\[
\phi = -\frac{d\phi}{\text{ref}3\text{n}}
\]
endwhere
\[
j = a / \rho
\]
where
\[
(\rho \geq 0.0) \quad \text{and} \quad (\text{abs}(jn/\text{ref}3\text{n}^2 + \phi) \geq \text{rotmax})
\]
cjc If already in the rotating frame, \(\phi\) should = 0
\[
cjc \quad jn = (\phi - \phi) \times \text{ref}3\text{n}^2
\]
end where
\[\]
C... force j=2 zone to be axisymmetric
!hpfs independent
\[
\text{forall}(k=1:kmax2)
\]
a(2,k,:) = sum(a(2,k,:), dim = 1)*xnorm
\[
jn(2,k,:) = \text{sum}(jn(2,k,:), \text{dim} = 1)*xnorm
\]
eps(2,k,:) = sum(eps(2,k,:), dim = 1)*xnorm
end forall
\[\]
c... set boundary conditions at z axis and on side of grid
\[\]
c... z axis...
\[\]
ceumann condition holds unless isyma = 1 or 2.
\[
\text{if } (\text{isyma} \neq 1 \quad \text{and} \quad \text{isyma} \neq 2) \text{ then}
\]
\[
rho(1,:) = \rho(2,:)
\]
\[
eps(1,:) = \rho(2,:)
\]
a(1,:) = a(2,:)
jn(1,:) = jn(2,:)
\[\]
eelse
\[
rho(1,:) = \text{cshift} \rho(2,:), \text{dim}=2, \text{shift}=lmax/2
\]
\[
eps(1,:) = \text{cshift} \eps(2,:), \text{dim}=2, \text{shift}=lmax/2
\]
a(1,:) = \text{cshift} \ a(2,:), \text{dim}=2, \text{shift}=lmax/2
\[\]
jn(1,:) = \text{cshift} \ jn(2,:), \text{dim}=2, \text{shift}=lmax/2
\[\]
eendif
\[\]
c... side of grid --- neumann conditions at j=jmax1
\[\]
\[
\text{if } (\text{sidbdy eq 'wall' or sidbdy eq 'free') then}
\]
\[
rho(jmax1,:) = \rho(jmax,:),
\]
\[
eps(jmax1,:) = \eps(jmax,:),
\]
a(jmax1,:) = a(jmax,)
jn(jmax1,:) = jn(jmax,)
\[\]
eendif
\[\]
c... if sidbdy ne. free or wall, then a dirichlet boundary condition
\[\]
is assumed and no modification of jmax1 is made.
\[\]
return
\[\]
end
\[\]
C-------------------------------------------------------
C eocflx(isym, botbdy, topbdy, sidbdy, itime)
C-------------------------------------------------------
C subroutine eocflx(isym, botbdy, topbdy, sidbdy, itime)
C
C As in the hyd counterpart, this routine simply calls arslop to calculate
C the slopes, then calls sflixtk to perform the fluxing. These calls should
C probably be rolled into here.
include 'grid.h'

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi, rho

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn

real, dimension (jmax2, kmax2, lmax) :: si, ti, a1, rho1, eps1
!hpf$ align with hydtpl :: si, ti, a1, rho1, eps1
common /eom1/ si, ti, a1, rho1, eps1

real :: xn, xni, konst, xnhyd
common /ptrope/ xn, xni, konst, xnhyd

integer isym, itime
character*6 botbdy, topbdy, sidbdy
real :: power, gamma
integer :: isyma

isyma = iabs(isym)
gamma = xnhyd
power = 1.0/gamma

c... make eps = lagrangian entropy of sorts.
c also, if calling fluxing second time, must also do eps1.

eps = (eps * rho) ** power
if (itime .ne.l) epsl = (epsl * rhol)**power

c calculate the slopes needed for the van leer interpolation
c arslop calculates slopes for a, rho, and eps.

call arslop(isyma)
call efxtk(itime)

c now, dividing up by k slices, we set some limits on eps, rho...
c take care of some axisymmetry at j=2, + redefine jn, omega... etc.
call efttk2(isyma, sidbdy)
c
... set boundary conditions on rho, eps, a, jn, and omega.
c ( top & side of grid done in task )
c... bottom of grid.
if(isyma.eq.1.or.isyma.eq.8)go to 296

c neumann condition if symmetry thru equatorial plane assumed.
c
294 continue
rho(:,1,:) = rho(:,2,:)
epsl(:,1,:) = epsl(:,2,:)
a(:,1,:) = a(:,2,:)
jn(:,1,:) = jn(:,2,:)
go to 910

296 continue

c if botbdy = 'wall' or 'free', then reflection symmetry here

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
amounts to the same thing as neumann condition.
if (botbdy .eq. 'wall' .or. botbdy .eq. 'free') go to 294
if botbdy .ne. free or wall, then a dirichlet boundary condi-
tion is assumed and no modification of k=1 is made.
910 continue

c . . .  top of grid.
if (topbdy .ne. 'wall' .and. topbdy .ne. 'free') go to 920
  neumann condition at k=kmax1.
  rho(:,kmax1,:) = rho(:,kmax,:)
  eps(:,kmax1,:) = eps(:,kmax,:)
  a(:,kmax1,:) = a(:,kmax,:)
  jn(:,kmax1,:) = jn(:,kmax,:)
920 continue

if (topbdy .ne. free or wall, then a dirichlet boundary condi-
tion is assumed and no modification of kmax1 is made.
c . . .  finished boundary conditions on rho, eps, a, jn, and omega.
c
return
end

Eocsrc(isym, botbdy, topbdy, sidbdy)
C-----------------------------------------------------------------
subroutine eocsrc(isym, botbdy, topbdy, sidbdy)
C
C perform the sourcing of the angular momentum density, a
C
include "grid.h"

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydpl :: phi, rho
  common /pois/ phi, rho
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydpl :: p, eps
  common /states/ p, eps
real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hynpl :: s, t, a, u, w, jn
  common /eom/ s, t, a, u, w, jn
real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydpl :: r, z, rhf, zhf
  common /grid/ r, z, rhf, zhf
real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpf$ align with hydpl :: rplus, zplus, rhfminus, zhfminus
  common /jgrid/ rplus, zplus, rhfminus, zhfminus
real :: deltar, deltaz
common /jgrid2/ deltar, deltaz
real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta
real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas
real :: denex, epsmin, epsmax
common /freeze/ denex,epsmin,epsmax

real :: omgfrm
common /rotfrm/ omgfrm

c
character*6 botbdy, topbdy, sidbdy
integer :: isyma, isym, k, j
real :: dtinv, d, fudge, rotmax
real, dimension (jmax2, kmax2, lmax) :: shjkl
real, dimension (jmax2, kmax2, lmax) :: denom
real, dimension (jmax2, kmax2, lmax) :: xmr, xmz, xml, xmlz, xmzr
real, dimension (jmax2, kmax2, lmax) :: xmlr, xmlzr
!hpf$ align with hydtpl :: shjkl, denom, xmr, xmz, xml, xmlz, xmzr
!hpf$ align with hydtpl :: xmlr, xmlzr
real, dimension (jmax2, kmax2, lmax) :: dphi, omgkep
!hpf$ align with hydtpl :: dphi, omgkep

ingterger mxrhoat(3)
ingterger jdmx,kdmx,ldmx

cjc here i initialize the variables
isyma = 0
dtinv = 0.0
d = 0.0
fudge = 0.0
rotmax = 0.0
c
isyma = iabs(isym)
c
dtinv=1.0/dtheta
c
do average s at grid center
xml = cshift(s, dim=3, shift=+1)
xmr = eoshift(s, dim=1, shift=+1)
xmz = eoshift(s, dim=2, shift=+1)
xml = eoshift(xmr, dim=1, shift=+1)
xmz = eoshift(xmz, dim=2, shift=+1)
xmr = eoshift(xmr, dim=1, shift=+1)
xmlz = eoshift(xmz, dim=1, shift=+1)
xmlzr = eoshift(xmzr, dim=1, shift=+1)
shjkl = ( & ( s + xml ) * (rhf**2 - r**2) * (zhf - z) + & ( xmr + xmlr ) * (rplus**2 - rhf**2) * (zhf - z) + & ( xmz + xmlz ) * (rhf**2 - r**2) * (zplus - zhf) + & ( xmr + xmlzr) * (rplus**2 - rhf**2) * (zplus - zhf))
denom = 2.0 * (rplus*rplus - r*r)*deltaz
where (denom .ne. 0.0)
shjkl = shjkl/denom
elsewhere
shjkl = 0.0
endwhere

cccc angular momentum density
\[ a = a - \text{delt} \cdot \text{dtinv} \cdot 0.5 \cdot (\rho \cdot \text{cshift}(\phi, \text{dim} = 3, \text{shift} = 1) - \\
& \text{cshift}(\phi, \text{dim} = 3, \text{shift} = -1)) + \\
& \text{cshift}(p, \text{dim} = 3, \text{shift} = 1) - \\
& \text{cshift}(p, \text{dim} = 3, \text{shift} = -1)) \\
& - \text{delt} \cdot 2 \cdot \text{omgfrm} \cdot \text{rhf} \cdot \text{shjkl} \]

... force j=2 zone to be axisymmetric

!hpf$ independent

forall (k~l: kmax2)
  \[ a(2,k,:) = \text{sum}(a(2,k,:), \text{dim} = 1)/ \text{float}(lmax) \]
end forall

... redefine \( \omega \) and \( jn \) ... also ...

to prevent time step from being controlled by high angular velocities in low density regions, set \( jn \) and \( \omega \) to zero if \( \rho \le \text{d} \) and if \( \omega > \text{fudge} \cdot \text{(central \( \omega \))} \).

\[ d = 1.0e3 \cdot \text{denex} \]
\[ \text{fudge} = 0.40 \]
\[ \text{mxrhoat} = \text{maxloc}((\rho(2:jmax,2:kmax,:)) \]
\[ jdmx = \text{mxrhoat}(1)+1 \]
\[ kdmx = \text{mxrhoat}(2)+1 \]
\[ ldmx = \text{mxrhoat}(3) \]
\[ \text{rotmax} = \text{fudge} \cdot \text{abs}(jn(jdmx,kdmx,ldmx)/\text{rhf}(jdmx,kdmx,ldmx)**2 + \text{omgfrm}) \]

r redefine \( jn \)

\[ \text{dphi} = \text{esoshift}(\phi, \text{dim} = 1, \text{shift} = 1) - \\
1 \cdot \text{esoshift}(\phi, \text{dim} = 1, \text{shift} = -1) \]
\[ \text{dphi} = 0.5/\text{rof3n} \cdot \text{dphi} \]

where (dphi .gt. 0.0)
  \[ \text{omgkep} = \text{sqrt}(\text{abs}(\text{dphi}/\text{rhf})) \]
elsewhere
  \[ \text{omgkep} = -\text{sqrt}(\text{abs}(\text{dphi}/\text{rhf})) \]
endwhere

where (\rho .ne. 0.0) \[ jn = a / \rho \]
where (\rho .le. 0.0) .and. (\rho .gt. 0.0) .and. 
& (\text{abs}(jn/\text{rhf}\cdot2 + \text{omgfrm}) .gt. \text{rotmax})

cjc If already in the rotating frame, omgfrm should = 0

cjc \[ jn = (\text{omgkep} - \text{omgfrm}) \cdot \text{rhf}\cdot2 \]
end where

\[ a = jn \cdot \rho \]

... force j=2 zone to be axisymmetric

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
!hpf$ independent
for all (k=1:kmax2)
   jn(2,k,:) = sum(jn(2,k,:), dim = 1)/ float(lmax)
   a(2,k,:) = sum(a(2,k,:), dim = 1)/ float(lmax)
end forall

c*** do bc's on z axis and on side of grid
  c neumann condition holds unless isyma = 1 or 2
  if (isyma .ne. 1 .and. isyma .ne. 2) then
     a(1,:,:)= a(2,:,:)
     jn(1,:,:)= jn(2,:,:)
  else
     a(1,:,:)= cshift( a(2,:,:),dim=2,shift=lmax/2)
     jn(1,:,:)= cshift(jn(2,:,:),dim=2,shift=lmax/2)
  endif
  c... side of grid (neumann conditions at j=jmaxi)
  if (sidbdy .eq. 'wall' .or. sidbdy .eq. 'free') then
     a(jmaxi,:,:)= a(jmax,:,:)
     jn(jmaxi,:,:)= jn(jmax,:,:)
  endif
  c... if sidbdy .ne. free or wall, dirichlet boundary condition
  c... is assumed and no modification of jmaxi is made.
  c
  c (remaining boundary conditions)
  c... bottom of grid.
  if(isyma.eq.1.or.isyma.eq.8)go to 296
  c neumann condition if symmetry thru equatorial plane assumed.
  294 continue
  a(:,1,:) = a(:,2,:)
  jn(:,1,:) = jn(:,2,:)
  go to 910
  296 continue
  c... if botbdy = 'wall' or 'free', then reflection symmetry here
  c amounts to the same thing as neumann condition.
  if(botbdy.eq.'wall'.or.botbdy.eq.'free')go to 294
  c if botbdy.ne. free or wall, then a dirichlet boundary
  c condition is assumed and no modification of k=1 is made.
  910 continue
  c
  c... top of grid.
  if(topbdy.ne.'wall' .and. topbdy.ne.'free')go to 920
  c neumann condition at k=kmaxi.
  a(:,kmaxi,:)= a(:,kmaxi,:)
  jn(:,kmaxi,:)= jn(:,kmaxi,:)
  920 continue
  c... if topbdy.ne. free or wall, then a dirichlet boundary
  c condition is assumed and no modification of kmaxi is made.
  c
  c... finished boundary conditions on jn and a.
  c
  return
end

function epmach(dum)
  epmach = 100.0/2.**64.0
  return
end
subroutine eq_dump(isym,first,frnum)
include 'grid.h'
cjc Dumps an equatorial slice of rho,s,and jn for analysis.
real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpfs$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpfs$ align with hydtpl :: phi, rho
common /pois/ phi, rho
real, dimension (jmax2, kmax2, lmax) :: mass
!hpfs$ align with hydtpl :: mass
common /cellmass/ mass
real, dimension (jmax2, kmax2, lmax) :: r,z,rhf,zhf
!hpfs$ align with hydtpl :: r,z,rhf,zhf
common /grid/ r,z,rhf,zhf
real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpfs$ align with hydtpl :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus
real :: xn, xnl, konst, xnhyd
common /ptrope/ xn, xnl, konst, xnhyd
real :: deltaz, deltaz
common /jgrid2/ deltaz, deltaz
real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta
real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ tmass, enew, elost, edif, phichk, klocat
real :: symfact
common /symmetry/ symfact
real :: tcirp, egrav, totjn, rzkin, ekin, dmax, virial
real :: cd, etot, echeck, alpha, edd, ell, bbeta, betarot
common /eq.dump3/ tcirp, egrav, totjn, rzkin, ekin, dmax, virial,
1 cd, etot, echeck, alpha, edd, ell, bbeta, betarot
integer :: numfr
common /eqdump/ numfr
real, dimension (jmax2, kmax2, lmax) :: x, y
!hpfs$ align with hydtpl :: x, y
common /eqdump2/ x, y
real, dimension (jmax2, kmax2, lmax) :: ixx, iyy, izz
!hpfs$ align with hydtpl :: ixx, iyy, izz
real, dimension (jmax2, kmax2, lmax) :: umr, umz, uml, umlz
!hpfs$ align with hydtpl :: umr, umz, uml, umlz

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real, dimension (jmax2, kmax2, lmax) :: jvel
!
hpf$ align with hydtp :: jvel
integer :: isym, first, keq, 1, frnum
real :: tixx, tiyy, tizz
real :: xcm, ycm, zcm, rcm
keq = 2
xcm = 0.0
ycm = 0.0
zcm = 0.0
if(isym .eq. 1) keq = kmax/2
if(first .gt. 0) then
open(unit=9, file='/work/ul5149/junk/bessel/eq/param',
& status='unknown', form='formatted')
write(9,100) jmax2, lmax, deltar, dtheta, isym, konst, xnhyd
open(unit=10, file='/work/ul5149/junk/bessel/eq/ieee.eq_unf',
& status='unknown', form='unformatted')
open(unit=11, file='/work/ul5149/junk/bessel/eq/global_values',
& status='unknown', form='formatted')
open(unit=21, file='/work/ul5149/junk/bessel/eq/cen_mass',
& status='unknown', form='formatted')
write(11,103)
numfr = 0
!
hpf$ independent
forall(1:1:max)
x(:,:,1) = rhf(:,:,1)*sin(dtheta*1)
y(:,:,1) = rhf(:,:,1)*cos(dtheta*1)
endforall
else if(first .eq. 0) then
  mass(2:jmax, 2:kmax,: ) 2 deltar*deltaz*dtheta*rhf(2:jmax,2:kmax,:)
  
calculation of the center of mass
  tmass = sum(mass(2:jmax, 2:kmax,:))
  if(isym .eq. 3) then
    xcm = sum(x(2:jmax,2:kmax,:)*mass(2:jmax,2:kmax,:))/tmass
    ycm = sum(y(2:jmax,2:kmax,:)*mass(2:jmax,2:kmax,:))/tmass
    rcm = sqrt(xcm**2 + ycm**2)
  endif
  write(21,107) xcm, ycm, zcm, rcm

calculation of the moment of inertia
  ixx = mass*(x**2 + zhf**2)
iy = mass*(y**2 + zhf**2)
iy = mass*zhf**2
tixx = symfact*sum(ixx)
tiyy = symfact*sum(iyy)
tizz = symfact*sum(izz)
tmass = symfact*tmass
write(11,104) f mum, tcirp, tmass, virial, totjn, alpha, ekin
write(11,105) bbeta, betarot, tixx, tiyy, tizz
umr = (eoshift(u, dim=1, shift=1))
umz = (eoshift(u, dim=2, shift=1))
uml = (cshift(u, dim=3, shift=1))
umlz = eoshift(uml, dim=2, shift=1)

jvel = ( (umr+umz+uml+umlz)*(rhf**2-r**2) +
(umr+eoshift(umz, dim=1, shift=1))
(umr+eoshift(uml, dim=1, shift=1))
(umr+eoshift(umlz, dim=1, shift=1))*(rplus**2-rhf**2) )
& /((4.0*(rplus**2-rhf**2))
write(10) jvel(:,keq,:), jn(:,keq,:), rho(:,keq,:)
numfr = numfr + 1
else
write(9,*) numfr
write(9,101)
write(9,102) numfr
write(21,109)
close(9)
close(10)
close(11)
close(21)
endif
100 format(ix,i3,4x,i3,3x,2(lpei5.7),2x,i3,2x,2(lpei5.7)>1
101 format('jmax2 lmax dltar dtheta isym konst xnhyd (gamma) ')
102 format('number of frames = ',i5)
103 format('frnum cirps/beta mass/betarot virial/ixx
totjn/iyy alpha/izz totkin')
104 format(is,2x,6(1pei5.7))
105 format(7x,5(1pei5.7))
106 format(5x,4(1pei5.7))
109 format(' xcm ycm zcm rcm')
return
end

C-----------------------------------------------
C fftout(isyma, data)
C-----------------------------------------------
subroutine fftout(isyma, data)
include 'grid.h'
cjc Calls and dumps FFT data for analysis.
integer, parameter :: lmax2 = lmax/2

integer :: ncount
common /mpfftc/ ncount

real :: cirp
common /normal/ cirp

real :: pi, grav
common /blok6b/ pi, grav

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

integer :: fftstart, callpot
common /vids/ fftstart, callpot

integer :: indx, isoadi, itstep
real :: allow
common /timst/ indx, isoadi, allow, itstep

real, array(jmax2, kmax2, lmax) :: data
!hpf$ distribute data(block, block, *)

integer :: jintrvl, jouter, klvl
common /fftoutput/ jintrvl, jouter, klvl

real, array(jmax2, lmax2, 20) :: amp, phaso
!hpf$ align amp(i, j, *) with hydtpl(i, *, 2*j)
!hpf$ align phase(i, j, *) with hydtpl(i, *, 2*j)
common /outfft/ amp, phase

real, array(jmax2) :: norml
!hpf$ align norml(:) with amp(:, :, *)

real, array(20) :: tmbuf
!hpf$ align tmbuf(:) with amp(*, *, :)
common /out2fft/ tmbuf

integer :: i, j, m, ml, n, je
integer skip, isyma
integer tmploc(3)
real tmpamp, tmpphase, angcnv

c change data from real to complex form & call the fft
angcnv = 180.0/pi

ncount = ncount + 1

tmbuf(ncount) = time/cirp
amp(:, 1, ncount) = sqrt(data(:, klvl, 1)**2 + data(:, klvl, 2)**2)

norml = 2.0/amp(:, 1, ncount)
amp(:, 1, ncount) = 1.0

do j=2, lmax2
ml = 2*j
m = ml - 1
amp(:, j, ncount) = sqrt(data(:, klvl, m)**2 + data(:, klvl, ml)**2) * norml

where (data(:, klvl, m).ne.0.0)
phase(:,j,ncount)=atan(data(:,klvl,m))/data(:,klvl,m)*angcnv
elsewhere
phase(:,j,ncount)=90.0
endwhere

where (data(:,klvl,m).eq.0.0 and data(:,klvl,m).eq.0.0)
phase(:,j,ncount)=720.
endwhere
where (data(:,klvl,m).lt.0.0)
phase(:,j,ncount)=(phase(:,j,ncount) + 180.0)
endwhere
where (phase(:,j,ncount).lt.0.0)
phase(:,j,ncount)=(phase(:,j,ncount) + 360.)
endwhere
end do

C COEF OUTPUT FOR EXCEL OR MATHEMATICA ANALYSIS. If we are without
C pi-symmetry, the coefs calculated above are the m=1, 2, ...
C components. If we are running with pi-symmetry, the modes
C are compacted to just the evens (m=2, 4, ...). This is governed
C by the "skip" parameter.

9156 format(6(I13.7,1X))
if (ncount.eq.20) then
i=1,ncount
write(26,9156) tmbuf(i)
write(26,9156)
1 (amp(je,m,i), je=fftstart,fftstart+jouter,jintrvl), m=2,5 ),
2 (phase(je,m,i), je=fftstart,fftstart+jouter,jintrvl), m=2,5 )
end do
write(27,*), tstep = ' , itstep, ' , cirp = ' , time/cirp
write(27,*), phasevsradius = ' ,
do 200 j=2,jmax-1
isyma = 3 then
write(27,*), ' ,j , ,phase(j,1+1,ncount)/2 , ,
else
write(27,*), ' ,j , ,phase(j,1+2,ncount) ,
endif
200 continue
if (isyma == 3) then
write(27,*), ' ,jmax> , ,phase(jmax,1+1,ncount)/2 ,
else
write(27,*), ' ,jmax> , ,phase(jmax,1+2,ncount)
endif
write(27,*), ampvsradius = ' ,
do 300 j=2,jmax-1
write(27,*), ' ,j> , ,log(amp(j,1+1,ncount)+1E-15)
300 continue
write(27,*), ' ,jmax> , ,log(amp(jmax,1+1,ncount)+1E-15)
ncount = 0
endif
return
end

C----------------------------------------------------------------------------------------------------------------------------
C flux(quant, slope, varea, deltx, voln, dir,retval)
C----------------------------------------------------------------------------------------------------------------------------
subroutine flux(quant, slope, varea, deltx, voln, dir,
& retval)

include 'grid.h'

cjc VERY important kernel. Calculates the amount of each physical
cjc variable to be fluxed in each timestep. This subroutine is called
cjc 30 times every timestep.

real, dimension (jmax2, kmax2, lmax) :: quant, slope, varea, voln
!hpf$ distribute quant(block,block,*)
!hpf$ align slope(i,j,k) with quant(i,j,k)
!hpf$ align varea(i,j,k) with quant(i,j,k)
!hpf$ align voln(i,j,k) with quant(i,j,k)

real, dimension (jmax2, kmax2, lmax) :: retval, temp
!hpf$ align retval(i,j,k) with quant(i,j,k)
!hpf$ align temp(i,j,k) with quant(i,j,k)

real deltx
integer dir

cjc here i initialize the variables
temp = 0.0

c the full fluxing is a sum over the six faces of a grid cell.
c ds/dt = del dot (s v) is the flux equation. This can be expressed
c as a sum, s(t+dt) = s(t) + (dt/volume)sum( s v area ).
c The problem arises in the flux term, (dt/volume)sum( s v area ) -->
c what value of s to use. The area is just the area of the face through
c which the variable is being fluxed.
c
1) In a donor cell technique, the quantity s used is just the
c value given in one cell. For example, fluxing density, the
c cell centered value of rho would be used unmodified, coupled
c with the velocity at the face.
c
2) In the Van Leer technique, the fluxing is done to second order
c by interpolating the value to be fluxed out to the face. Hence,
c we are using a value of rho, for example, calculated on the
c face through which material is flowing. The slope used to
c interpolate out to the face is calculated in arslop.f.

c for this subroutine, we pass the quantity to be fluxed, the slopes,
c the velocity times area, time step, 1/volumes of cells, and which
c direction (1=radial, 2=z, 3=theta) to be fluxed.

c define a macro to do interpolation to face using van lear slopes.
c slopes must be calculated before reaching this point.
c
vlil(xl,xs) = xl + 0.5*xs
vli(xr,ys) = xr - 0.5*ys

c do front half of fluxing
nc if the velocity is positive, material is removed from this cell.
c if the velocity is negative, material is added from forward cell.

if (dir .eq. 1) then
  where (varea .gt. 0.0)
    temp = (quant + 0.5*slope)varea
  elsewhere
temp =
& (eoshift(quant,dim=1,shift=1) - 0.5 *
& eoshift(slope,dim=1,shift=1)) * varea
end where
retval = - (deltax*voln)*(temp -
& eoshift(temp,dim=1,shift=-1))
endif

if (dir .eq. 2) then
where (varea .gt. 0.0)
temp = (quant + 0.5*slope)*varea
elsewhere
temp =
& (eoshift(quant,dim=2,shift=1) - 0.5 *
& eoshift(slope,dim=2,shift=1)) * varea
end where
retval = - (deltax*voln)*(temp -
& eoshift(temp,dim=2,shift=-1))
endif

if (dir .eq. 3) then
where (varea .gt. 0.0)
temp = (quant + 0.5*slope)*varea
elsewhere
temp =
& (cshift(quant,dim=3,shift=1) - 0.5 *
& cshift(slope,dim=3,shift=1)) * varea
end where
retval = - (deltax*voln)*(temp -
& cshift(temp,dim=3,shift=-1))
endif

return
end

C--------------------------------------------
C hflxtk(isyma, botbdy, topbdy, sidbdy, itime)
C--------------------------------------------
subroutine hflxtk(isyma, botbdy, topbdy, sidbdy, itime)
include 'grid.h'
cjc Fluxes s and t by calling flux subroutine.

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtemp :: s, t, a, u, w, jn
common /eom/ s, t, a ,u ,w, jn
real, dimension (jmax2, kmax2, lmax) :: sl, tl, al, rhol, epsl
!hpf$ align with hydtemp :: sl, tl, al, rhol, epsl
common /eom1/ sl, tl, al, rhol, epsl
real, dimension (jmax2, kmax2, lmax) :: sj, sk, sl, tj, tk, tl
!hpf$ align with hydtemp :: sj, sk, sl, tj, tk, tl
common /slop2/ sj, sk, sl, tj, tk, tl
real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydtemp :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf
real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
This routine fluxes $s$ and $t$ by setting up variables for a call to the
generic fluxing program 'flux.f'.

cjc here i initialize the variables

ug = 0.0
wh = 0.0
v = 0.0
voln = 0.0
cvoln = 0.0
retnx = 0.0
transu = 0.0
transw = 0.0
tettot = 0.0

c as in other routines, special treatment of the axis has been eliminated.
c we must treat $t$ on the axis in a special manner: cylindrical volume,
c many zones fluxing into it.

c protect $j=1$

where $(rhf**2 .ne. rhfminus**2)$

voln = 1.0/(0.5*dtheta * deltaz * (rhf**2 - rhfminus**2))
elsewhere
voln = 0.0
derwhere

ug = 0.5 * ( u + eoshift(u,dim=1,shift=1))
wh = 0.5 * ( w + eoshift(w,dim=2,shift=1))

The following be work for dirichlet bc on side and top
c to keep stuff from coming back onto grid and to allow
c flow off of the grid

ug(jmax,:) = u(jmax,:,:)
ug(jmax1,:,:)=0.0
ug(jmax2,:,:)=0.0
ug(:,kmax2,:)=0.0
wh(:,kmax,:)=w(:,kmax,:)
wh(:,kmax1,:)=0.0
wh(:,kmax2,:)=0.0
wh(jmax2,:,:)=0.0

! Crude, John... shifting too much in calculations exhausts memory...

retflx = eoshift(jn, dim=2, shift=-1)
v = 0.25 * (jn + retflx)/rhf
v = v + eoshift(v, dim=1, shift=-1)
v(2,:,:)= 0.0

transu = ug * dtheta * rhf * deltaz
transv = vh * 0.5 * dtheta * (rhf**2 - rhfrainus**2)
transv = v * deltaz * deltar

c do fluxing of s

cjnop print*, 'Calling flux for s'
call flux(s, sj, transu, delt, voln, 1, retflx)
rettot = retflx
call flux(s, sk, transw, delt, voln, 2, retflx)
rettot = rettot + retflx
call flux(s, sl, trans, delt, voln, 3, retflx)
rettot = rettot + retflx

if (itime .eq. 1) then
  s = s + rettot
else
  s = sl + rettot
endif

c... treat z-axis
s(2,:,:)= 0.0

c fluxing of t

cjnop print*, 'Calling flux for t'
call flux(t, tj, transu, delt, voln, 1, retflx)
rettot = retflx
call flux(t, tk, transw, delt, voln, 2, retflx)
rettot = rettot + retflx
call flux(t, tl, trans, delt, voln, 3, retflx)
rettot = rettot + retflx

if (itime .eq. 1) then
  t = t + rettot
else
  t = tl + rettot
endif

c
ccc now do top, side, & bottom b.c. (do z-axis when out of task)
c
ccc k=2
if(isyma.eq.1.or.isyma.eq.8)go to 96

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
neumann condition if symmetry thru equatorial plane assumed.
94 continue
s(:,1,:) = s(:,3,:)
t(:,1,:) = -t(:,3,:)
t(:,2,:) = 0.0
go to 910
96 continue
c if botbdy = 'wall', then set neumann conditions.
if(botbdy.eq.'wall')go to 94
c if botbdy = 'free', then set velocities at k=1 to those
derived for k=2; i.e., a free-flowing boundary condition.
if(botbdy.ne.'free')go to 910
s(:,1,:) = s(:,2,:)
t(:,1,:) = t(:,2,:)
910 continue
c if botbdy.ne. free or wall, then a dirichlet boundary conditi
c is assumed and no modification of k=1 is made.
c
c top of grid.
if(topbdy.ne.'wall')go to 915
c neumann conditions at k=kmax1
s(:,kmax2,:) = s(:,kmax,:)
t(:,kmax2,:) = -t(:,kmax,:)
t(:,kmax1,:) = 0.0
goto 921
915 if(topbdy.ne.'free')go to 920
c free boundary condition being used at k=kmax1.
s(:,kmax2,:) = s(:,kmax1,:)
t(:,kmax2,:) = t(:,kmax1,:)
goto 921
920 continue
c if topbdy.ne. free or wall, then a dirichlet boundary
c condition is assumed
c the following assignment assures flow off of grid
c but not back on
t(:,kmax1,:) = 0.5*(abs(t(:,kmax,:))+t(:,kmax,:))
t(:,kmax2,:) = 0.0
s(:,kmax2,:) = 0.0
921 continue
c side of grid.
if(sidbdy.ne.'wall')go to 925
c neumann conditions at j=jmax1.
s(jmax2,:,:) = -s(jmax,:,:)
t(jmax2,:,:) = t(jmax,:,:)
s(jmax1,:,:) = 0.0
return
925 if(sidbdy.ne.'free')go to 930
c free boundary condition being used at j=jmax1.
s(jmax2,:,:) = s(jmax1,:,:)
t(jmax2,:,:) = t(jmax1,:,:)
return
930 continue
c if sidbdy.ne. free or wall, then a dirichlet boundary
c condition is assumed
c the following assignment assures flow off of grid
c but not back on
s(jmax1,:,:) = 0.5*(abs(s(jmax,:,:))+s(jmax,:,:))
s(jmax2,:,:) = 0.0
t(jmax2,:,:) = 0.0
c... finished boundary conditions on s and t.
c  return
cend
C-----------------------------------------------
C hsrctk(isyma, sidbdy)
C-----------------------------------------------
subroutine hsrctk(isyma, sidbdy)
includ \ "grid.h"
cjc Sources s and t.
  real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl : phi, rho
common /pois/ phi, rho
  real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl : p, eps
common /staten/ p, eps
  real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl : s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
  real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydtpl : r, z, rhf, zhf
common /grid/ r, z, rhf, zhf
  real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
!hpf$ align with hydtpl : rplus, zplus, rhfminus
common /jgrid/ rplus, zplus, rhfminus
  real :: deltar, deltaz
common /jgrid2/ deltar, deltaz
  real :: omgfrm
common /rotfrm/ omgfrm
  real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

C perform the sourcing of s and t.

character*6 sidbdy
integer isyma
integer :: i, k, j
  real, dimension (jmax2, kmax2, lmax) :: rhjkl, drp, dzp
!hpf$ align with hydtpl : rhjkl, drp, dzp
  real, dimension (kmax2) :: st
!hpf$ align st(:) with hydtpl(*,:,*)
  real, dimension (jmax2, kmax2, lmax) :: drphi, dzphi, ahjkl, rotahjkl
!hpf$ align with hydtpl : drphi, dzphi, ahjkl, rotahjkl
  real, dimension (jmax2, kmax2, lmax) :: denom
!hpf$ align with hydtpl : denom

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real, dimension (jmax2, kmax2, lmax) :: xmr, xmxz, xml, xmlz, xmlr
!
!hpfs$ align with hydtp1 :: xmr, xmxz, xml, xmlz, xmlr
!
real, dimension (jmax2, kmax2, lmax) :: xmlr, xmlzr
!
!hpfs$ align with hydtp1 :: xmlr, xmlzr

CJC here I initialize the variables
l = 0

C... if constp is .gt. 0.0, apply constant pressure boundary condition.
if (constp .gt. 0.0) then
  where (p > constp)
    p = constp
  endwhere
endif

xml = cshift(rho, dim=3, shift=-1)
xmr = eoshift(rho, dim=1, shift=-1)
xmz = eoshift(rho, dim=2, shift=-1)
xmlr = eoshift(xml, dim=1, shift=-1)
xmlz = eoshift(xml, dim=2, shift=-1)
xmzr = eoshift(xmz, dim=1, shift=-1)
xmlzr = eoshift(xmlz, dim=1, shift=-1)

rhjkl = (n
& ( rho + xml ) * (rhf**2 - r**2) * (zhf - z) +
& ( xmr + xmlr ) * (r**2 - rhfminus**2) * (zhf - z) +
& ( xmz + xmlz ) * (rhf**2 - r**2) * (z - zhfminus) +
& ( xmlr + xmlzr ) * (r**2 - rhfminus**2) * (z - zhfminus))

where (denom .ne. 0.0)
  rhjkl = rhjkl/denom
elsewhere
  rhjkl = 0.0
endwhere

denom = 2.0 * (rhf*rhf - rhfminus*rhfminus)*deltaz

rotahjkl = omgfrm*
& ( rho +xml )*rhf**2*(rhf**2 - r**2)*(zhf - z) +
& ( xmr + xmlr )*rhfminus**2*(r**2 - rhfminus**2)*(zhf - z) +
& ( xmz + xmlz )*rhf**2*(r**2 - rhf**2)*(z - zhfminus) +
& ( xmlr + xmlzr)*rhfminus**2*(r**2 - rhfminus**2)*(z - zhfminus))

where (denom .ne. 0.0)
  rotahjkl = rotahjkl/denom
elsewhere
  rotahjkl = 0.0
endwhere

xml = cshift(a, dim=3, shift=-1)
xmr = eoshift(a, dim=1, shift=-1)
xmz = eoshift(a, dim=2, shift=-1)
xmlr = eoshift(xml, dim=1, shift=-1)
xmlz = eoshift(xml, dim=2, shift=-1)
xmzr = eoshift(xmz, dim=1, shift=-1)
xmlzr = eoshift(xmlz, dim=1, shift=-1)

ahjkl = (n

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
& ( a + xml ) * (rhf**2 - r**2) * (zhf - z) +
& ( xmr + xmlr ) * (r**2 - rhfminus**2) * (z - zhfminus) +
& ( xmz + xmlz ) * (rhf**2 - r**2) * (z - zhfminus) +
& ( xmrz + xmlzr) * (r**2 - rhfminus**2) * (z - zhfminus))

where (denom .ne. 0.0)
    ahjkl = ahjkl / denom
elsewhere
    ahjkl = 0.0
endwhere

c
    xml = cshift(p, dim=3, shift=-1)
    xmr = eoshift(p, dim=1, shift=-1)
    xmz = eoshift(p, dim=2, shift=-1)
    xmlr = eoshift(xml, dim=1, shift=-1)
    xmlz = eoshift(p, dim=2, shift=-1)
    xmlr = eoshift(xmr, dim=1, shift=-1)
    xmlzr = eoshift(xmlz, dim=1, shift=-1)

    drp = 0.25 * ( p + xml + xmlz + xmz
                  - xmr - xmlr - xmlzr - xmrz)

    dzp = 0.25 * (p + xml + xmlr + xmr
                  - xml - xmlz - xmlzr - xmrz)

    xml = cshift(phi, dim=3, shift=-1)
    xmr = eoshift(phi, dim=1, shift=-1)
    xmz = eoshift(phi, dim=2, shift=-1)
    xmlr = eoshift(xml, dim=1, shift=-1)
    xmlz = eoshift(xmr, dim=2, shift=-1)
    xmlr = eoshift(xmz, dim=1, shift=-1)
    xmlzr = eoshift(xmzr, dim=1, shift=-1)

    drphi = 0.25 * ( phi + xml + xmlz + xmz
                    - xmr - xmlr - xmlzr - xmrz)

    dzphi = 0.25 * (phi + xml + xmlr + xmr
                    - xml - xmlz - xmlzr - xmrz)

c this protects the j=1 zone...

where (r .ne. 0.0)
    s = s - delt * ((drp + rhjkl*drphi)/deltar
                   - ahjkl**2/(rhjkl*(r**3))
                   - (2.*omgfrm*ahjkl/r)
                   - rotahjkl**2/(rhjkl*(r**3)))
    t = t - delt * (dzp + rhjkl*dzphi)/deltaz
endwhere

s(2,:,:)= 0.0

c... finished general solution.
c
c... establish boundary conditions on s and t.
c
c... z-axis

c neumann condition holds except for isyma = 1 or 2.
if(isyma.ne.1.and.isyma.ne.2) then
    s(1,:,:) = -s(3,:,:)
    t(1,:,:) = t(3,:,:)
!hpf$ independent
for all \( k=3:k_{\text{max}} \) \( st(k) = \text{sum}(t(3,k,:), \text{dim}=1) \)

!hpf$ independent
for all \( k=3:k_{\text{max}} \) \( t(2,k,1) = st(k)/\text{float}(l_{\text{max}}) \)

!hpf$ independent
for all \( l=1:l_{\text{max}} \) \( t(2,:,l) = t(2,:,1) \)
else
\( s(1,:,1) = \text{csift}(s(3,:,1), \text{dim}=2, \text{shift}=l_{\text{max}}/2) \)
\( t(1,:,1) = \text{csift}(t(3,:,1), \text{dim}=2, \text{shift}=l_{\text{max}}/2) \)

!hpf$ independent
for all \( k=3:k_{\text{max}} \) \( st(k) = \text{sum}(t(3,k,:), \text{dim}=1) \)

!hpf$ independent
for all \( k=3:k_{\text{max}} \) \( t(2,k,1) = st(k)/\text{float}(l_{\text{max}}) \)

!hpf$ independent
for all \( l=1:l_{\text{max}} \) \( t(2,:,l) = t(2,:,l) \)
endif

c... side of grid.
if(sidebdy.ne.'wall')go to 925
c
neumann conditions at \( j=j_{\text{max}} \).
\( s(j_{\text{max}}2,:,1) = -s(j_{\text{max}},,:,1) \)
\( t(j_{\text{max}}2,:,1) = t(j_{\text{max}},,:,1) \)
\( s(j_{\text{max}},,:,1) = 0.0 \)
return
925 if(sidebdy.ne.'free')go to 930
c
free boundary condition being used at \( j=j_{\text{max}} \).
\( s(j_{\text{max}}2,:,1) = s(j_{\text{max}}1,:,1) \)
\( t(j_{\text{max}}2,:,1) = t(j_{\text{max}}1,:,1) \)
return
930 continue
c
if sidebdy.ne. free or wall, then a dirichlet boundary
c
condition is assumed
c
the following assignment assures flow off of the grid
c
but not back on to it
\( s(j_{\text{max}}1,:,1) = 0.5*(\text{abs}(s(j_{\text{max}},,:,1)) + s(j_{\text{max}},,:,1)) \)
\( s(j_{\text{max}}2,:,1) = 0.0 \)
\( t(j_{\text{max}}2,:,1) = 0.0 \)
c
... finished these boundary conditions on \( s \) and \( t \).
c
return
end

C-----------------------------------------------------
C hydflx(isym, botbdy, topbdy, sidbdy, itime)
C-----------------------------------------------------
subroutine hydflx(isym, botbdy, topbdy, sidbdy, itime)
include 'grid.h'

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpil :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

 This routine does nothing more than set up the slopes with a
 subroutine call, and then does the fluxing through a subroutine call.
 The slope call and the fluxing call should probably be folded into
 one routine, but for now they are left as they are.
integer isym, itime
character*6 botbdy, topbdy, sidbdy
integer :: isyma,k,l

real, dimension (kmax2) :: st
!hpf$ align st(:) with hydtpi(*,*)

!cjc here i initialize the variables
isym = 0
isyma = abs(isym)

!c calculate slopes for s and t

!call stslcp(isyma)
!... finished boundary conditions on s and t.
!c
!call hflxk(isyma, botbdy, topbdy, sidbdy, itime)
!... finished general solution.

!c... finish the boundary conditions on s and t.
!c ... z-axis
!c remember that t(2,k,l) is in savtnw(k,l)
!c neumann condition holds except for isyma * 1 or 2.
!if (isyma.ne.1 .and. isyma.ne.2) then
!cerr s(1,:) = -s(3,:)-mistake!
!cerr t(1,:) = t(3,:)
!cerr t(2,:) = savtnw(:,:)
!hpf$ independent
!forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)
!hpf$ independent
!forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)
!hpf$ independent
!forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)
!hpf$ independent
!forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)
!else
!s(1,:) = -cshift(s(3,:),dim=2, shift=lmax/2)
!t(1,:) = cshift(t(3,:),dim=2, shift=lmax/2)
!t(2,:) = savtnw(:,:)
!hpf$ independent
!forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)
!hpf$ independent
!forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)
!hpf$ independent
!forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)
!endif
!
c
return
end
call a routine to do the sourcing, then do bc's.

integer isym
character*6 botbdy, topbdy, sidbdy
integer :: isyma

cjc here i initialize the variables

isyma = 0

isyma = iabs(isym)

call hrctk(isyma, sidbdy)

c... finish the boundary conditions on s and t.

k=2

if(isyma.eq.1.or.isyma.eq.8)go to 96

neumann condition if symmetry thru equatorial plane assumed.

94 continue

s(:,1,:) = s(:,3,:)

= -t(:,3,:)

= 0.0

go to 910

96 continue

if(botbdy = 'wall', then set neumann conditions.

if(botbdy.eq. 'wall')go to 94

if botbdy = 'free', then set velocities at k=1 to those
derived for k=2; i.e., a free-flowing boundary condition.

if(botbdy.ne. 'free')go to 910

s(:,1,:) = s(:,2,:)

t(:,1,:) = t(:,2,:)

910 continue

if botbdy.ne. free or wall, then a dirichlet boundary condition is assumed and no modification of k=1 is made.

c... top of grid.

if(topbdy.ne. 'wall')go to 915

neumann conditions at k=kmax1

s(:,kmax2,:) = s(:,kmax,:),

t(:,kmax2,:) = -t(:,kmax,:),

t(:,kmax1,:) = 0.0

return

915 if(topbdy.ne. 'free')go to 920

free boundary condition being used at k=kmax1.

s(:,kmax2,:) = s(:,kmax1,:),

t(:,kmax2,:) = t(:,kmax1,:),

return

920 continue

if topbdy.ne. free or wall, then a dirichlet boundary condition is assumed

the following assignment assures flow off of grid but not back on:

t(:,kmax1,:) = 0.5*(abs(t(:,kmax,:))+t(:,kmax,:))

t(:,kmax2,:) = 0.0

s(:,kmax2,:) = 0.0
c

... finished boundary conditions on s and t.
 c

return
end

C-------------------------------------------------------------------
C init
C-------------------------------------------------------------------
subroutine init
include 'grid.h'
cjc Initializes all global variables.

real :: pi, grav
common /blok6b/ pi,grav

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

integer :: ncount
common /mpfftc/ ncount

real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hp$ align with hydtpl :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf

real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta

integer :: igrid
common /old2/ igrid

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, v, jn
!hp$ align with hydtpl :: s, t, a, u, v, jn
common /som/ s, t, a, u, v, jn

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hp$ align with hydtpl :: phi, rho
common /pois/ phi, rho

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hp$ align with hydtpl :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: mass
!hp$ align with hydtpl :: mass
common /cellmass/ mass

integer :: indx, isoadi, istep
real :: allow
common /timst/ indx, isoadi, allow, istep

real :: xn, xni, konst, xnhyd
common /ptrope/ xn, xni, konst, xnhyd

real :: rco, tco, zco, vrco, vtco, vzco
common /codefs/ rco, tco, zco, vrco, vtco, vzco

real :: cirp
common /normal/ cirp
real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: omgfrm
common /rotfrm/ omgfrm

real, dimension (jmax2, kmax2, lmax) :: si, ti, ai, rho1, eps1
!hpf$ align with hydtmp :: si, ti, ai, rho1, eps1
common /eom1/ si, ti, ai, rho1, eps1

real, dimension (jmax2, kmax2, lmax) :: aj, ak, al, rj, rk, rl, ej, ek, el
!hpf$ align with hydtmp :: aj, ak, al, rj, rk, rl, ej, ek, el
common /slopes/ aj, ak, al, rj, rk, rl, ej, ek, el

real, dimension (jmax2, kmax2, lmax) :: sj, sk, sl, tj, tk, tl
!hpf$ align with hydtmp :: sj, sk, sl, tj, tk, tl
common /slopes2/ sj, sk, sl, tj, tk, tl

real :: cvheat
common /tempcv/ cvheat

real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ tmass, enew, elost, edif, phichk, klocat

real :: vmxin, vmxout
common /vmaxx/ vmxin, vmxout

real :: ptmass
common /cenpob/ ptmass

real :: egold, ekold, pdvold
common /inrite/ egold, ekold, pdvold

integer :: icom
common /comflg/ icom

real :: dmax0
common /fdelta/ dmax0

real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpf$ align with hydtmp :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus

real :: deltarr, deltax
common /jgrid2/ deltarr, deltax

real :: incline, J20
common /pstr/ incline, J20

integer :: fftstart, callpot
common /vids/ fftstart, callpot

integer :: zeroout
common /zerot/ zeroout

real :: symfact
common /symmetry/ symfact

pi = 0.0
grav = 0.0

rcloud = 0.0
constp = 0.0
delt = 0.0
bdytem = 0.0
den = 0.0
time = 0.0
cormas = 0.0

ncount = 0

r = 0.0
z = 0.0
rhf = 0.0
zhf = 0.0

rof3n = 0.0
zof3n = 0.0
alnewr = 0.0
alnewz = 0.0
dtheta = 0.0

igrid = 0

s = 0.0
t = 0.0
a = 0.0
u = 0.0
w = 0.0
jn = 0.0

phi = 0.0
rho = 0.0

p = 0.0
eps = 0.0

mass = 0.0

indx = 0
isoadi = 0
itstep = 0

xn = 0.0
xni = 0.0
konst = 0.0
xnyd = 0.0

rco = 0.0
tco = 0.0
zco = 0.0
vrco = 0.0
vtco = 0.0
vzco = 0.0

cirp = 0.0
denex = 0.0
epsmin = 0.0
epsmax = 0.0
omgfrm = 0.0
si = 0.0
t1 = 0.0
ai = 0.0
rho1 = 0.0
eps1 = 0.0
aj = 0.0
ak = 0.0
al = 0.0
rj = 0.0
rk = 0.0
rl = 0.0
ej = 0.0
ek = 0.0
e1 = 0.0
sj = 0.0
sk = 0.0
sl = 0.0
tj = 0.0
tk = 0.0
tl = 0.0
cvheat = 0.0
tmass = 0.0
enev = 0.0
elost = 0.0
edif = 0.0
phichk = 0.0
klocat = 0
vmxin = 0.0
vmxout = 0.0
ptmass = 0.0
egold = 0.0
ekold = 0.0
pdvoid = 0.0
icom = 0
dmax0 = 0.0
rplus = 0.0
zplus = 0.0
rhfminus = 0.0
zhfminus = 0.0
deltar = 0.0
deltaz = 0.0
incline = 0.0
J20 = 0.0
fftstart = 0
callpot = 0
zeroout = 0
symfact = 0.0
return
end

C-----------------------------------
C izum(ical1,sigma,ampO,isyma)
C-----------------------------------
subroutine izum(ical1,sigma,ampO,isyma)

c This routine needs a good cleaning. It was only given a cursory
convention from f77 to f90. Some points:
c The only perturbation given is an m=2. The code exists for
of a number of comments. It should be restructured back to the
c original 4=random, 5=m pert.
c include 'grid.h'

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi, rho

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn

real :: rof3n, zof3n, ainewr, ainewz, dtheta
common /grid2/ rof3n, zof3n, ainewr, ainewz, dtheta

real :: xn, xni, konst, xnyhd
common /ptrope/ xn, xni, konst, xnyhd

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

integer, parameter :: M0 PERT = 3, NULL PERT = 4
integer, parameter :: RAW PERT = 5, M PERT = 6

c
real :: rpi max, zmax, ratio

real, dimension (jmax2, kmax2) :: denny, anggy
!hpf$ align denny(i,j) with hydtpl(i,j,*)
!hpf$ align anggy(i,j) with hydtpl(i,j,*)
common /fiss/ denny, anggy

real, dimension (jmax, kmax) :: pert
!hpf$ align pert(i,j) with hydtpl(i,j,*)

integer xxx(2)
integer :: ical1, ntape
real :: sigma, dencen, ddolr, ddelz, pindex, rrr2, omcen
real :: tovers, ampO, sml, theta, emt, pertl, con2
common /jetadd/ toverw
real :: dchek, gamma
integer :: jjmax, kkmax, j, k, kzmax, jreq, kzpol
common /fiss2/ jreq, kzpol
integer :: l, isyma
integer mxloc(3), mnloc(3)

cjc Hpl wrapper for generating a 2 dimensional array
cjc of random numbers.
cjc

cmpf mpl myp_rnd

j=1
l=1
k=1
dencen=maxval(denny)

c zero out u, w, jn, and rho arrays.
s = 0.0
t = 0.0
jn = 0.0
rho = 0.0

select case(icall)
    case(IO_PERT)
        print*, 'IO_PERT'
    endif
    if (isyma .ne. 1) then
        do 1=1,Imax
            rho(1:jmax,1:kmax,1)=denny(l:jmax,1:kmax)
            jn(1:jmax,1:kmax,1)=anggy(1:jmax,1:kmax)
        end do
    endif
    if (isyma == 1) then
        do 1=1,Imax
            rho(1:jmax,kmax/2+1:kmax,1)=denny(l:jmax,2:kmax/2+1)
            rho(1:jmax,kmax/2+1:kmax,1)=denny(l:jmax,2:kmax/2+1)
            jn(1:jmax,kmax/2+1:kmax,1)=anggy(1:jmax,2:kmax/2+1)
            jn(1:jmax,kmax/2+1:kmax,1)=anggy(1:jmax,2:kmax/2+1)
        end do
    endif
    ! I think this one can be deleted
    case(NULL_PERT)
    if icall=5, perturb model with random density perturbation having
    maximum amplitude of ampO.
write(6,1001)amp0
if (isym(ne. 1) then
do l=1,lmax
   jn(1:jmax,1:kmax,l) = anggy(1:jmax,1:kmax)
end do
else
do l=1,lmax
   jn(1:jmax,kmax2/2+l:kmax,l)=anggy(1:jmax,2:kmax/2+l)
end do
endif

if (isym(ne. 1) then
do l=1,lmax
   cjcran cedi myp_rnd(pert)
pert = amp0*(2.0*pert-1.0)
rho(1:jmax,1:kmax,l) = (1.0+pert)*denny(1:jmax,1:kmax)
end do
else
do l=1,lmax
   cjcran cedi myp_rnd(pert)
pert = amp0*(2.0*pert-1.0)
rho(1:jmax,kmax2/2+l:kmax,l) =
   (1.0+pert(:,1:kmax/2))*denny(1:jmax,2:kmax/2+l)
cjcran cedi myp_rnd(pert)
pert = amp0*(2.0*pert-1.0)
rho(1:jmax,kmax2/2:2:-l,l) = (1.0+pert(:,1:kmax/2))
   *denny(1:jmax,2:kmax/2+l)
end do
endif

c case(M_PERT)
print*, 'M_PERT'
c if icall=6, perturb model with straight m=int(sigma) perturbation
  of amplitude amp0. (actually, ramp amplitude up to amp0)
write(6,1002)amp0,sigma
theta=-dtheta
eml=sigma
if(isym.ne.1) then
do l=1,lmax
   theta=theta+dtheta
   emt=eml*theta
   pertl=amp0*cos(emt)
   jn(2:jmax,2:kmax,l)=anggy(2:jmax,2:kmax)
   rho(2:3,2:kmax,l) = denny(2:3,2:kmax)
   rho(4:6,2:kmax,l) = (1.0+0.25*float(j)*pertl)
   *denny(4:6,2:kmax)
   rho(7:jmax,2:kmax,l)=(1.0 + pertl) * denny(7:jmax,2:kmax)
end do
write(6,* 'isym = ',isym.
else
do l=1,lmax
   theta=theta+dtheta
   emt=eml*theta
end do

pertl=ampO*cos(emt)

jn(2:jmax, kmax/2:1:1,l)=anggy(2:jmax, 2:kmax/2+1)
rho(2:3, kmax/2:1:-1,1) = denny(2:3, 2:kmax/2+1)
rho(4:6, kmax/2:1:-1,1) = (1.0+0.5*pertl)*denny(4:6, 2:kmax/2+1)
rho(7:jmax, kmax/2:1:-1,1)=(1.0 + pertl) *
  denny(7:jmax, 2:kmax/2+1)
1

jn(2:jmax, kmax/2+1:kmax, 1)=anggy(2:jmax, 2:kmax/2+1)
rho(2:3, kmax/2+1:kmax, 1) = denny(2:3, 2:kmax/2+1)
rho(4:6, kmax/2+1:kmax, 1) = (1.0+0.5*pertl)*
  denny(4:6, 2:kmax/2+1)
rho(7:jmax, kmax/2+1:kmax, 1)=(1.0 + pertl) *
  denny(7:jmax, 2:kmax/2+1)
1

end do
write(6,*) 'isym = ',isyma
endif
end select

on 12/3/85, changed limits of loop from max2 to maxi because
outermost zones really shouldn't need any density,
dchek = 1.0e-7 * dencen
print*, 'denex = ',denex,' dchek','
dchek = ,dchek

where (rho .lt. dchek) rho = denex

ccccbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c.... on 12/3/85, changed limits of loop from max2 to maxi because
.... outermost zones really shouldn't need any density.
dchek = 1.0e-7 * dencen
print*, 'denex = ',denex,' dchek','
dchek = ,dchek

where (rho .lt. dchek) rho = denex

cccccbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c.... on 12/18/86 at LSU, added initialization of eps here in order
to allow gamma.ne.(1+i/xnl) adiabatic evolutions.
c

c.... initialize eps:
gamma = xnhyd
where (rho .ne. 0.0)
  eps = konst*rho**xnl / (rho * (gamma - 1.0))
elsewhere
  eps = 0.0
endwhere

cforce j=2 zone to be axisymmetric
forall (k=l: kmox)
rho(2,k,:) = sum(rho(2,k,:),dim=1)/float(lmax)
jn(2,k,:) = sum( jn(2,k,:),dim=1)/float(lmax)
eps(2,k,:) = sum(eps(2,k,:),dim=1)/float(lmax)
endforall

cmake sure values of variables on borders of grid are
correctly initialized
if (isyma.ne.1 .and. isyma.ne.2) then
  rho(1,:,)= rho(2,:,)
  eps(1,:,)= eps(2,:,)
  jn(1,:,)= jn(2,:)
else
  rho(1,:,)= cshift(rho(2,:,),dim=2,shift=lmax/2)
  eps(1,:,)= cshift(eps(2,:,),dim=2,shift=lmax/2)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
jn(:,1,:) = cshift(jn(:,2,:),dim=2,shift=1 max/2)
endif

rho(:,1,:) = rho(:,2,)
eps(:,1,:) = eps(:,2,)
nj(:,1,:) = nj(:,2,)

134 format(5x,9p,5e20.7)
1001 format(5x, 'got to random, amp0 =', 1p, e15.4)
1002 format(' perturbing initial model straight amp0 =', 1p, e10.3, ' mode =', 0p, f6.3)

c
return
end

program main
include 'grid.h'

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi, rho

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl:: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn

real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydtpl:: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

integer :: zeroout
common /zerot/ zeroout

integer :: fftstart, callpot
common /vids/ fftstart, callpot

real :: cirp
common /normal/ cirp

integer :: igrid
common /old2/ igrid

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: incline, J20
common /pstr/ incline, J20

integer :: ncount
common /mpfttc/ ncount

integer :: indx, isoadi, itstep
real :: allow
common /time/ indx, isoadi, allow, itstep
character*6 botbdy, topbdy, sidebdy
character(len=26) whichone
integer mtime, mpTimerElapsed, frnum, intrvl
integer minusl, zero, one, two, three, four, five, eight, f15, fifty
data minus1, zero, one, two, three, four, five, eight, f15, fifty
1 / -1 , 0 , 1 , 2 , 3 , 4 , 5 , 8 , 15, 50/
integer :: isdiag, itstrt, itstop, idia, istor, itype
& , isym, maxtrm, isyma, ihead, ihchk,
& iflag, j, eq_wr, icool
real :: sptm, sptm2, timef
real :: cirpo, cirpn, cirpd
real, dimension (jmax2, kmax2, lmax) :: rhotmp
!hp$ align with hydtpl :: rhotmp

INTERFACE
EXTRINSIC(f77_LOCAL) SUBROUTINE send (frnum)
INTEGER, INTENT(IN) :: frnum
END SUBROUTINE send
END INTERFACE

open(unit=26, file='fftout/fort.26', status='unknown')
open(unit=27, file='fftout/fort.27', status='unknown')

call init
whichone='/work/ul5149/junk/disk1000'

isdiag=5
indx=0
call setup(itstrt, itstop, idia, isym, isi, itype, isym, maxtrm, 1
& botbdy, topbdy, sidbdy, frnum, intrvl, icool)

cjc I added this to write out equatorial slices every time a frame
cjc is written out for the movie. The call below just sets up the
cjc output files.
eq_wr=1
call eq_dump(isym, eq_wr, frnum)
eq_wr=0
write(6,*) 'frnum = ', frnum, ' cirp interval = ', intrvl
write(6,*) 'incline = ', incline, ' J20 strength is = ', J20
ncount=0
call rad(isoadi)

isyma = iabs(isym)
sptm = timef()

c begin the time-step loop.
do itstep=itstrt, itstop

ccc c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c

ccc c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c

c c set some diagnostic switches.  c

ccc

index=itstep-itsr
tHEAD=0
if (mod(index, idia).eq.0) ihead=-1
ihchk = itstep-idia
if (ihchk.gt.0 .and. mod(ihchk, fifty).eq.0) ihead=1

ccc c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c

call delta(itstrt, itstop)
c)iw want to source for only a half t-step

delt = 0.5 * delt

c apply sources
    call hydsrc(isym, botbdy, topbdy, sidbdy)
    call eocsrc(isym, botbdy, topbdy, sidbdy)

c recalculate u and w velocities (jn already done in eocsrc).
    call vel(isym, botbdy, topbdy, sidbdy)

c save the sources after 1/2 tstep (to finish them after full v update)
    call save

c advect all quantities for 1/2 tstep (to get v at 1/2t)
c call with 'one' to use current values of rho, v's
    call hydflx(isym, botbdy, topbdy, sidbdy, one)
    call eocflx(isym, botbdy, topbdy, sidbdy, one)

c recalculate u and w velocities (jn already done in eoc).
    call vel(isym, botbdy, topbdy, sidbdy)

    delt = 2.0 * delt

c advect the full tstep
    c call 'two' to use saved values of rho, v's -> values after 1/2 source
    call hydflx(isym, botbdy, topbdy, sidbdy, two)
    call eocflx(isym, botbdy, topbdy, sidbdy, two)

c calculate the velocities again
    call vel(isym, botbdy, topbdy, sidbdy)
    call state(icool)
    call rad(isoadi)

    iflag = 0
    if (igrid.eq.0) iflag=1
    if (callpot.gt.0) then
        call pot(iflag, isyma)
    endif

c apply sources for the last 1/2 tstep
    delt = 0.5*delt
    call hydsrc(isym, botbdy, topbdy, sidbdy)
    call eocsrc(isym, botbdy, topbdy, sidbdy)
    call vel(isym, botbdy, topbdy, sidbdy)

c put delt back to what it should be....

    delt = 2.0*delt

c move the central object with just disk potential
    c if (isyma .eq. 2 .or. isyma .eq. 1) call moveco
c artificially kill all velocities/momentum in innermost zeroout zones!!!
c inserted death of density --- jww 8/10/90 (because of junk in center)

s(l:zeroout,:,: ) = 0.0
u(l:zeroout,:,: ) = 0.0
t(l:zeroout,:,: ) = 0.0
w(l:zeroout,:,: ) = 0.0
a(l:zeroout,:,: ) = 0.0
jn(l:zeroout,:,: ) = 0.0
rho(l:zeroout,:,: )=denex

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c c
c printed diagnostics c
c c
rtes(zero,...) gives 2-line summary and equatorial coefs c
(it also writes coefs to disk if mod(itstep,isdiag) = 0) c

c
if(indx.eq.0)go to 87
if(mod(itstep,isdiag).ne.0)go to 88
87 call rite_diag(ihead,isym,frnum)
c c
ccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
88 continue

 time = time + delt

if((intrvl*time/cirp).GT.frnum) then
 write(6,*) 'frnum = ',frnum, ' time/cirp = ',time/cirp
call num_three(frnum,whichone)
rhotmp= rho
rhotmp(jmax-l:jmax2,:,:)=0.0
ccjc I added this subroutine to dump the equatorial values of s,t,jn,
cjc and rho.
call eq_dump(isym.eq_wr,frnum)
open(unit=50,form='unformatted',status='unknown',
1 file=whichone)
write(unit=50) rhotmp
close(unit=50)
c we are ready to call fountain here
print *,"Wrote out density file for imaging"
call send(frnum)
frnum = frnum + 1
endif
ccjc 1/11/96 if(time/cirp gt 9.0) exit
end do
cc end time-step loop.
c
sptm = timefO
write(6,*) " Time elapsed : ",sptm/1000.0, " seconds."
itstep=itstep
close(26)
close(27)
cjc Call eq_dump to close files and do cleanup for the equatorial files

eq_wr = -1

call eq_dump(isym,eq_wr,frnum)

call rite_cont(isym,frnum,icool)

stop
end

C-----------------------------------
C num_three(frnum,whichone)
C-----------------------------------------------
subroutine num_three(frnum,whichone)
cjc Given a number, calculates a filename of the form
cjc disk##.#

cjc

integer i,j,k,l,lpi,frnum,lnth
character(*) whichone

lnth = len(whichone)
l = frnum/1000
k = (frnum - l*1000)/100
j = (frnum - l*1000 - k*100)/10
i = frnum - l*1000 - k*100 - j*10
lpi = l + 1

whichone(lnth-3:lnth-3) = char(lpi +48)
whichone(lnth-2:lnth-2) = char(k+48)
whichone(lnth-1:lnth-1) = char(j+48)
whichone(lnth:lnth) = char(i+48)

write(6,*) 'whichone is ', whichone

return
end

C-----------------------------------------------
C rad(i)
C-----------------------------------------------
subroutine rad(i)
include 'grid.h'
cjc Calculates the total internal energy.

cjc
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtp1 :: phi, rho
common /pois/ phi,rho

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtp1 :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: r,z,rhf,zhf
!hpf$ align with hydtp1 :: r,z,rhf,zhf
common /grid/ r,z,rhf,zhf

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus

!hpf$ align with hydtpl :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus

real :: deltar, deltaz
common /jgrid2/ deltar, deltaz

real :: rof3n, zof3n, aineqr, aineqz, dtheta
common /grid2/ rof3n, zof3n, aineqr, aineqz, dtheta

real :: xn, xni, konst, xnyd
common /ptrope/ xn, xni, konst, xnyd

real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ tmass, enew, elost, edif, phichk, klocat

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

real :: cvheat
common /tempcv/ cvheat

real :: symfact
common /symmetry/ symfact

integer i, k

c... eventually this routine will do the radiative transfer. now,
c... however, it only treats isothermality or adiabatic conditions.
c... if i=1, isothermal cloud.
c... if i=2, adiabatic collapse.
c... if i=3, polytropic case.

real, dimension (jmax2, kmax2, lmax) :: voln
!hpf$ align voln(i,j,k) with hydtpl(i,j,k)

real :: epssum, epsnew, gamma

cjc here i initialize the variables
voln = 0.0
epssum = 0.0
epsnew = 0.0
gamma = 0.0

voln = 0.5*dtheta*(rplus**2 - r**2) * deltaz

if(i.eq.1) then
  c... isothermal case.
c... edif: energy lost via rad. this timestep alone.
c... elost: total energy rad. away since initial model.
c... enew: total internal energy. should remain const. in time.
  epssum = sum(eps*voln*rho - cvheat*bdytem*voln*rho)
  epsnew = sum( cvheat*bdytem*voln*rho)
  eps = cvheat * bdytem
\[
\begin{align*}
\epsilon(1,:,:) &= \epsilon(2,:,:) \\
\epsilon(:,1,:) &= \epsilon(:,2,:) \\
\text{edif} &= \text{symfact} \times \epsilon \text{sum} \\
\epsilon_{\text{lost}} &= \epsilon_{\text{lost}} + \text{edif} \\
\epsilon_{\text{new}} &= \text{symfact} \times \epsilon_{\text{new}} \\
\text{endif}
\end{align*}
\]

if (i.eq.3) then
\begin{itemize}
  \item \text{polytropic case}
    \begin{align*}
    \gamma &= \text{xnhyd} \\
    \text{where (rho .ne. 0.0)} \\
    \epsilon &= p/(\text{rho} \times (\gamma - 1))
    \end{align*}
  \end{itemize}
endwhere

\begin{itemize}
  \item \text{adiabatic case (or continuation of polytropic case),}
  \item \text{edif is increase in internal energy this time step.}
  \item \text{total internal energy is in enew.}
\end{itemize}

\begin{align*}
\epsilon_{\text{sum}} &= \text{symfact} \times \text{sum}(\epsilon \times \text{volit} \times \text{rho}) \\
\epsilon_{\text{sum}} &= \text{symfact} \times \text{sum}(\epsilon(2:jmax,2:kmax,:) \times \text{voln}(2:jmax,2:kmax,:) \times \text{rho}(2:jmax,2:kmax,:)) \\
\text{edif} &= \epsilon_{\text{sum}} - \epsilon_{\text{new}} \\
\epsilon_{\text{new}} &= \epsilon_{\text{sum}} \\
\text{endif}
\end{align*}

return

end

subroutine realft(data, nx, ny, nz, isign)

include 'proc.h'

\text{real,} \text{dimension(nx,ny,nz)}:: \text{data}
!hpf$ \text{distribute data(block,block,*)}

\text{real,} \text{dimension(nx,ny)}:: \text{hii, hir, h2i, h2r}
!hpf$ \text{distribute(block,block)} :: \text{hii, hir, h2i, h2r}

\begin{align*}
\text{real:: theta, wi, vpi, vpr, wr, wtemp} \\
\text{theta}=3.141592653589793/(\text{nz}/2) \\
c1=0.5 \\
\text{if (isign.eq.1) then} \\
c2=0.5
\end{align*}
call fouri(data,nx,ny,nz/2,+1)
else
  c2=0.5
  theta=-theta
endif
wpr=-2.0*sin(0.5*theta)**2
wpi=sin(theta)
wr=1.0+wpr
wi=wpi
n2p3=nz+3
do i=2,nz/4
  ii=2*i-1
  i2=ii+1
  i3=n2p3-i2
  i4=i3+i
  wrs=wr
  wis=wi
  hir=c1*(data(:,:,ii)+data(:,:,i3))
hii=c1*(data(:,:,i2)-data(:,:,i4))
h2r=-c2*(data(:,:,i2)+data(:,:,i4))
h2i=c2*(data(:,:,ii)-data(:,:,i3))
data(:,:,i1)=hir+wr*wri+wsi*h2i
data(:,:,i2)=hii+wri+w2r+wis*h2i
data(:,:,i3)=hir+w2r+wsi+h2i
data(:,:,i4)=hii+w2r+wr*wi+wsi*h2i
wtemp=wr
wr=wr*wpr-wi*wpi+wi
wi=wi*wpr+wtemp*wpi+wi
endo
dif (isign.eq.1) then
  hir(:,i1)=data(:,i1)
data(:,i1)=hir(:,i1)+data(:,i2)
data(:,i2)=hir(:,i2)-data(:,i2)
else
  hir(:,i1)=data(:,i1)
data(:,i1)=c1*(hir(:,i1)+data(:,i2))
data(:,i2)=c1*(hir(:,i2)-data(:,i2))
call fouri(data,nx,ny,nz/2,-1)
endif
return
end

C---------------------------------------------------------------
C  fouri(data,nx,ny,nnz,isign)
C---------------------------------------------------------------
subroutine fouri(data,nx,ny,nnz,isign)
C---------------------------------------------------------------
include 'proc.h'

  real,dimension(nx,ny,2*nnz)::data
  !hpfs distribute data(block,block,*)

  real,dimension(nx,ny)::tempi,tempr
  !hpfs distribute(block,block) :: tempi,tempr

  real::theta,wi,wpi,wpr,wr,wtemp
  integer::nx,ny,nnz,isign,i,istep,j,m,mmax,nz

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
nz=2*nz
j=1
do i=1,nz,2
  if (j .gt .i) then
    temp=data(:,:,j)
    temp1=data(:,:,j+1)
    data(:,:,j)=data(:,:,i)
    data(:,:,j+1)=data(:,:,i+1)
    data(:,:,i)=temp
    data(:,:,i+1)=temp1
  endif
  m=nz/2
  continue
  if ((m .ge. 2).and.(j .gt .m)) then
    j=j-m
    m=m/2
    goto 1
  endif
  j=j+m
  enddo
  mmax=2
  continue
  if (nz .gt. mmax) then
    istep=2*mmax
    theta=6.28318530717959/((isign*nmax)
    wpr=-2.*sin(0.5*theta)**2
    wp=sin(theta)
    wr=1.
    wi=0.
    do m=1,mmax,2
      do i=m,nz,istep
        j=i+mmax
        temp=wr+data(:,:,j)-wi*data(:,:,j+1)
        temp1=wr+data(:,:,j+1)+wi*data(:,:,j)
        data(:,:,j)=data(:,:,i)-temp
        data(:,:,j+1)=data(:,:,i+1)-temp1
        data(:,:,i)=data(:,:,i)+temp
        data(:,:,i+1)=data(:,:,i+1)+temp1
      enddo
      wt=wr
      wr=wr*wpr+wp+wi
      wi=wi*wpr+wt*wp+wr
    enddo
    mmax=istep
    goto 2
  endif
  return
end

C---------------------------------------------------------------
C rite_cont(isyma,frnum,icool)
C---------------------------------------------------------------
subroutine rite_cont(isyma,frnum,icool)
  include 'grid.h'
cjc Writes out the continuation file.
  real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
  common /pois/ phi, rho

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ p, eps

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a ,u ,w , jn

real, dimension(jmax2,kmax2,1max)::rhop,phi,phic
!hpf$ distribute(block,block,*) :: rhop,phi,phic
common /potarray1/ rhop,phi,phic

real :: deltar, deltaz
common /jgrid2/ deltar, deltaz

real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta

integer :: indx, isoadi, itstep
real :: allow
common /timst/ indx, isoadi, allow, itstep

real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas

real :: cirp
common /normal/ cirp

real :: omgfrm
common /rotfrm/ omgfrm

real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ tmass, enew, elost, edif, phichk, klocat

real :: ptmass
common /cenpob/ ptmass

real :: xn, xnl, konst, xnhyd
common /ptrope/ xn, xnl, konst, xnhyd

real :: rco, tco, zco, vrc, vtc, vzco
common /codefs/ rco, tco, zco, vrc, vtc, vzco

integer :: icom
common /comfg/ icom

real :: vmxin, vmxout
common /velmax/ vmxin, vmxout

integer :: fftstart, callpot
common /vids/ fftstart, callpot

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: mlost, jlost
common /outer/ mlost, jlost

real :: tchirp, egrav, totjn, rzkin, skin, dmax, virial
real :: cd, etot, echeck, alpha, edd, ell, bbeta, betarot
common /eq_dump3/ tcirp, egrav, totjn, rskin, skin, dmax, virial,
  cd, etot, echeck, alpha, edd, ell, bbeta, betarot

150

4/25/96 These are the interval and stopping point that the fft
4/25/96 data is taken at.

cjc

integer :: jintrvl, jouter
common / fftoutput / jintrvl, jouter

real :: coolst, coolrt
common / cool / coolst, coolrt

integer :: frnum, ntapes, isyma, icool
real :: percool

110 format (1hl,///, ' sssssssssssssssssssssssssssssssssssssssssssss',///,
  1' s',4lx,'s',//, ' s model from time step number',i6,' has s',//,
  2' s' been stored on disk. it is stored on tape unit',i3,'.',8x,'s',
  3', ' s',4lx,'s',//,
  4' sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss...
cjc
if (isyma .eq. 1 .or. isyma.eq.2) then
  write(ntapes)s, t,jn,rho,eps,phi,rof3n,zof3n,alnewr,alnewz,
  delt, time, elost, frnum, cirm, ptmass, konst, vmxin, fftstart,
  jouter, jintrvl, denex, mlost, jlost, rco, tco, zco, vrco, vtco,
  vso, icom
else
  write(ntapes)s, t,jn,rho,eps,phi,rof3n,zof3n,alnewr,alnewz,
  time, elost, frnum, cirm, ptmass, konst, vmxin, fftstart,
  jouter, jintrvl, denex, mlost, jlost
endif
end

cjc
write(diag,ihead,isyma,frnum)

-------------------------------------------------------------

C subroutine rite.diag(ihead,isyma,frnum)
-------------------------------------------------------------

include 'grid.h'
cjc
C Writes out diagnostics.
c
abs(ihead) = 1 prints heading once per call to subroutine.
c ihead = negative skips to a new page; ihead.ge.0 does not.
c ihead = negative skips to a new page; ihead.ge.0 does not.
c
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydpi :: phi, rho
common /pois/ phi, rho
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydpi :: p, eps
common /states/ p, eps
real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydpi :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydpi :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf
real :: deltar, deltaz
common /jgrid2/ deltar, deltaz
real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta
integer :: indx, isoadi, itstep
real :: allow
common /timst/ indx, isoadi, allow, itstep
real :: rcloud, constp, del, bdytem, den, time, cormas
common /blok7/ rcloud, constp, del, bdytem, den, time, cormas

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real :: cirp
common /normal/ cirp

real :: pi, grav
common /blok6b/ pi,grav

real :: omgfrm
common /rotfrm/ omgfrm

real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ tmass, enew, elost, edif, phichk, klocat

real :: egold, ekold, pdvold
common /inrite/ egold, ekold, pdvold

real :: ptmass
common /cenpob/ ptmass

real :: xn, xni, konst, xnhyd
common /ptrope/ xn, xni, konst, xnhyd

real :: rco, tco, zco, vrco, vtco, vzco
common /codefs/ rco, tco, zco, vrco, vtco, vzco

integer :: icom
common /comflg/ icom

real :: vmxin, vmxout
common /velmax/ vmxin, vmxout

integer :: fftstart, callpot
common /vids/ fftstart, callpot

real :: symfact
common /symmetry/ symfact

cjc 4/25/96  These are the interval and stopping point that the fft
cjc 4/25/96  data is taken at.
cjc

integer :: jintrvl, jouter
common /fftoutput/ jintrvl, jouter

real, dimension(jmax2, kmax2, lmax)::rhop, phip, phic
!hpf$ distribute(block, block, *) :: rhop, phip, phic
common /potarray1/ rhop, phip, phic

real :: tcirp, egrav, totjn, rzkin, ekin, dmax, virial
real :: cd, etot, echeck, alpha, edd, ell, bbeta, betarot
common /eq_dump3/ tcirp, egrav, totjn, rzkin, ekin, dmax, virial,
1   cd, etot, echeck, alpha, edd, ell,
2   bbeta, betarot

real :: eeg, totj, er, errot, erfrm, rzk, factor
real :: eerot, eerotfrm, rhomax
real :: pdv, egg

integer :: ihead, isyma, frnum
integer :: jd, kd, ld
integer locmax(3)

real, dimension (jmax2, kmax2, lmax) :: frmfac

!hpf$ align frmfac(i,j,k) with hydtpl(i,j,k)

100 format(ihl)
103 format(ihl)
104 format(' tstep', 3x, 'time', 8x, 'delt', 6x, 'etot/jt', 4x, 'egrav/beta'
  1, 2x, 'ekin/rzkin', 3x, 'alpha/cd', 3x, 'edif/dmax', 3x, 'elost/jkl',
  2 3x, 'tmass/virial', 6x, 'betarot', 4x, 'k',//)
105 format(i6, lp, 10el2.4, i4)
106 format(5x, lp, el2.4, * cirps', 6x, 1p, 5e12.4, i4, 2i3, 2x, 1p, 2e12.4,//)
109 format(5x, 'val at rhomax =', 1p, 4e12.4,/) ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

if(ihead.lt.0) write(6,100)
if(ihead.gt.0) write(6,103)
if(iabs(ihead).eq.1) write(6,104)
tcirp=time/cirp
cc c... find egrav, totj, erot.
c c
cc note : dvolnl = dtheta*dr*rhf, centered on rho,jn,eps
cc note : dvoln2 = dtheta*dr2, centered on s,t

eeg = 0.5 * dtheta*deltaz*deltar*sum( rhf(2:jmax,2:kmax,:)
  & * (phi(2:jmax,2:kmax,:) + phic(2:jmax,2:kmax,:))
  & *rho(2:jmax,2:kmax,:))
where (jn .ne. 0.0)
  frmfac = rho*omgfrm*rhf**2
elsewhere
  frmfac = 0.0
delsewhere
  totj = dtheta*deltaz*deltar*sum(rhf(2:jmax,2:kmax,:)
  1 *a(2: jmax, 2:kmax,:) + rhf(2: jmax,2:kmax,:)*frmfac(2:jmax,2:kmax,:))

cjc 3/25/98
cjc This is the rotational energy from the gas in the rotating
cc frame.
cjc cjc

errot = 0.5*dtheta*deltaz*deltar*sum(rhf(2:jmax,2:kmax,:)*
  & a(2:jmax,2:kmax,:)*
  & jn(2:jmax,2:kmax,:)/ rhf(2:jmax,2:kmax,:)**2)

  frmfac = jn/rhf**2
c c... find rzhk

er = errt + erfrm

cjc 3/25/98

cjc This is the rotational energy from the gas in the rotating
cc frame.
cjc cjc

errot = 0.5*deltax*deltaz*deltar*sum(rhf(2:jmax,2:kmax,:)*
  & a(2:jmax,2:kmax,:)*
  & jn(2:jmax,2:kmax,:)/ rhf(2:jmax,2:kmax,:)**2)

  frmfac = jn/rhf**2
cc c... find rzhk

er = errt + erfrm
\[
\text{rzk} = \text{rzk} + 0.5 \cdot \pi \cdot \text{deltax} \cdot \text{deltar} \cdot 2^* \\
\text{sum}(w(2,:,1) \cdot t(2,:,1), \text{dim}=1)
\]

\[
\text{if}(\text{isym} \neq 1) \text{ then}
\]
\[
\text{rzk} = \text{rzk} + 0.25 \cdot \text{deltar} \cdot \text{deltaz} \cdot \text{dtheta} \cdot \text{sum}(r(3:jmax,2,:),u(3:jmax,2,:)) \\
\text{endif}
\]

\[
\text{multiply integral properties by factor, depending on symmetry being used.}
\text{factor} = \text{symfact}
\]
\[
\text{eggrav} = \text{factor} \cdot \text{eeg}
\text{totjn} = \text{factor} \cdot \text{totj}
\text{eerot} = \text{factor} \cdot \text{er}
\text{eerotfrm} = \text{factor} \cdot \text{eerot}
\text{rzkin} = \text{factor} \cdot \text{rzk}
\text{ekin} = \text{eerot} + \text{rzkin}
\]

\[
\text{rhomax} = \text{maxval}(\text{rho}(2:jmax2,2:kmax2,:))
\text{locmax} = \text{maxloc}(\text{rho}(2:jmax2,2:kmax2,:))
\]
\[
\text{dmax} = \text{rhomax}
\text{jd} = \text{locmax}(1)+1
\text{kd} = \text{locmax}(2)+1
\text{ld} = \text{locmax}(3)
\text{cd} = \text{rho}(2,2,1)
\text{pdv} = 0.0
\text{etot} = \text{eggrav} + \text{enew} + \text{elost} - \text{pdv}
\text{echck} = 0.0
\text{echck} = (\text{ekin} - \text{ekold} + \text{edd})/(\text{egold} - \text{eggrav} + \text{pdv} \cdot \text{pdvold}) - 1.0
\text{egold} = \text{eggrav}
\text{pdvold} = \text{pdv}
\text{ekold} = \text{ekin}
\]
\[
\text{inserts by joel at lsu 5/16/83 to write alpha, beta, etc.}
\text{c}
\]
\[
\text{cjc total grav. energy}
\text{egg} = \text{abe}(\text{eggrav})
\text{cjc total energy/grav. energy}
\text{etot} = \text{etot}/\text{egg}
\text{cjc kinetic energy/grav. energy}
\text{ekin} = \text{ekin}/\text{egg}
\text{cjc internal energy/grav. energy}
\text{alpha} = \text{enew}/\text{egg}
\text{edd} = \text{edd}/\text{egg}
\text{ell} = \text{elost}/\text{egg}
\text{cjc rotational energy/grav. energy}
\text{bbeta} = \text{eerot}/\text{egg}
\text{betarot} = \text{eerotfrm}/\text{egg}
\text{c}
\text{cjc stop insert.}
\text{c}
\text{virial} = \text{alpha} + \text{ekin} - 0.5
\]
\[
\text{write}(6,105) \text{itstep, time, delt, etot, egrav, ekin, alpha, edd, ell, tmass, betarot, klocat}
\text{write}(6,106) \text{tcirp, totjn, bbeta, rzkin, cd, dmax, jd, kd, ld, virial}
\]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
SUBROUTINE SAVE
C as part of the Lax-Wendroff time splitting, we must save variables
C from
C the first half of the time-step sourcing. Here, we save the relevant
C variables.
C
INCLUDE 'GRID.H'
REAL, DIMENSION (JMAX2, KMAX2, LMAX) :: PHI, RHO
!HPF$ ALIGN WITH HYDTPL :: PHI, RHO
COMMON /POIS/ PHI, RHO
REAL, DIMENSION (JMAX2, KMAX2, LMAX) :: P, EPS
!HPF$ ALIGN WITH HYDTPL :: P, EPS
COMMON /STATES/ P, EPS
REAL, DIMENSION (JMAX2, KMAX2, LMAX) :: S, T, A, U, W, JN
!HPF$ ALIGN WITH HYDTPL :: S, T, A, U, W, JN
COMMON /EOM/ S, T, A, U, W, JN
REAL, DIMENSION (JMAX2, KMAX2, LMAX) :: SI, TI, AI, RHOI, EPSI
!HPF$ ALIGN WITH HYDTPL :: SI, TI, AI, RHOI, EPSI
COMMON /EOM1/ SI, TI, AI, RHOI, EPSI
C
SI = S
TI = T
AI = A
RHOI = RHO
EPSI = EPS
RETURN
END
C---------------------------------------------------------------
C SCFIN(SIGMA)
C---------------------------------------------------------------
SUBROUTINE SCFIN(SIGMA)
INCLUDE 'GRID.H'
CJC Reads in the initial 2d model from the SCF code.
REAL, DIMENSION (JMAX2, KMAX2, LMAX) :: P, EPS
!HPF$ ALIGN WITH HYDTPL :: P, EPS
COMMON /STATES/ P, EPS
REAL :: ROF3N, ZOF3N, ALNEWR, ALNEWZ, DTHETA
COMMON /GRID2/ ROF3N, ZOF3N, ALNEWR, ALNEWZ, DTHETA
REAL :: RCL, CONSTP, DELT, BDBYTEM, DEN, TIME, CORMAS
COMMON /BLOK7/ RCL, CONSTP, DELT, BDBYTEM, DEN, TIME, CORMAS
real :: xn, xni, konst, xnhyd
common /ptrope/ xn, xni, konst, xnhyd

real :: cirp
common /normal/ cirp

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: ptmass
common /cenpob/ ptmass

real :: vmxin, vmxout
common /velmax/ vmxin, vmxout

c

real :: rpimax, zzmax, ratio

real, dimension (jmax2, kmax2) :: denny, anggy
!hpf$ align denny(i,j) with hydpl(i,j,*)
!hpf$ align anggy(i,j) with hydpl(i,j,*)
common /fiss/ denny, anggy

integer xxx(2)
integer :: icall, ntape
real :: sigma, dencen, ddelr, ddelz, pindex, rrr2, omcen
real :: toverw, con2, gamma
common /jetadd/ toverw

integer :: jjmax, kkmax, j, k, kzmax, jrmax, kreq, kpols
common /fiss2/ jreq, kpols
integer :: l

... inserts made 11/13/79 to read durisen's polytropes.
100 format (lp, 5dl5.8)
134 format (5x, 0p, 5e20.7)
140 format (lp, 6dl3.5>

c

denny = 0.0
anggy = 0.0
ntape = int(sigma)

c.....read in izumi's model.
c
read(ntape, 176) jjmax, kkmax
176 format (2i5)
read(ntape, *) jjmax, kkmax

cjw add this to stop stupid errors
if (jjmax .gt. jmax2 .or. kkmax .gt. kmax2) then
write(6, *) model grid structure too large...
stop
endif

cjw finished addition

CJC Changed this read statement to read dr and dz in
CJC directly from the SCF or an analytic solver.
CJC
read(ntape, 140) rpimax, zzmax
read(ntape, 140) ddelr, ddelz
read(ntape, 140) ratio, cirp
read(ntape,140) konst,vmxin
read(ntape,140) (denny(j,k),j=1,jjmax),k=1,kkmax)
read(ntape,140) ((angvy(j,k),j=1,jjmax),k=1,kkmax)
write(6,*) 'cirp = ',cirp

cc dencen = maxval(denny)
xxx = maxloc(denny(2:jmax,2:kmax))
kkmax = xxx(2)+1
jmax = xxx(1)+1
den = dencen
ptmass = ratio
print*, 'dencen = ',dencen
print*, 'maxdenloc = ',xxx(1),' ',xxx(2)

denex = 1.0e-7 *dencen
gamma = xnhyd
epsmin = konst/(gamma - 1.0)*denex**(1.0/xn)
epsmax = konst/(gamma - 1.0)*(1e-4*den)**(1.0/xn)
alnewz=1.0
rof3n=ddelr
zof3n=ddelz
ainewr=1.0

cc following are dummy values for izumis model.
pindex = xn
con2 = ptmass
rrr2 = ddelr*(jjmax-1)
omen = ddelz*(kkmax-3)
toverw = 1.0
jreq = jmax
kzpol = kmax
write(6,136)pindex,con2,rrr2,omen,dencen,toverw,ddelr,
1 ddelz,ainewz,jreq,kzpol
136 format(///,5xi 'pindex = ', Ip,ell .3,/,5x, 'ptmass = ',lp,ell .3,/,5x, 'rpimax = ' ,ip,ell .3,/,5x, 'zzmax = ',ip,ell .3,/,5x, 'dencen = ',ip,ell .3,/,5x, 'tovewr = ',ip,ell .3,/,5x, 'ddelr = ',ip,ell .3,/,5x, 'jmax = ',
4 i5,/,5x,'kzmax = ',i5,//)
return
end

C--------------------------------------------------------------
C scfin3d(isyma)
C--------------------------------------------------------------
subroutine scfin3d(isyma)
include 'grid.h'
cjc Reads in 3D initial files from the 3D SCF code.
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpfs align with hydpl :: phi, rho
common /pois/ phi,rho
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpfs align with hydpl :: p, eps
common /states/ p, eps
real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpfs align with hydpl :: s, t, a, u, w, jn
common /som/ s, t, a, u, w, jn

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
real :: scfre, scfrhomax, scfomega
common /scfout/ scfre, scfrhomax, scfomega

real :: rof3n, zof3n, alnewr, alnewz, dtheta
common /grid2/ rof3n, zof3n, alnewr, alnewz, dtheta

real :: rcloud, constp, deltax, den, time, cormas
common /blok7/ rcloud, constp, deltax, den, time, cormas

real :: xn, xni, konst, xnhyd
common /ptrope/ xn, xni, konst, xnhyd

real :: ptmass
common /cenpob/ ptmass

real :: cirp
common /normal/ cirp

real :: pi, grav
common /blok6b/ pi, grav

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

real :: rpi, ratio
integer xxx(3)
real :: sigma, denex, ddelr, ddelz, ddelth, pindex, rrr2, omcen
real :: epsmin2, epsmax2
real :: toverw, con2, gamma, konst1, konst2, omgsq
common /jetadd/ toverw
integer :: jjmax, kkmax, llmax, j, k, l, kmmax, jrmmax, lthmax
integer :: lxmax, lymax, isym

/* inserts made 11/13/79 to read durisen's polytropes. */
100 format(1p,5d15.8)
134 format(5x,0p,5e20.7)
140 format(1p,6d13.5)

open(unit=8, file='input', form='formatted', status='old')

read(8,*) jjmax, kkmax
read(8,*) ddelr, ddelz, ddelth

/* add this to stop stupid errors */
if ((jjmax .gt. jmax2) .or. (kkmax .gt. kmax2) .or. 
   (llmax .gt. lmax)) then
   write(6,*) 'model grid structure too large...'
   stop
endif

/* finished addition */
read(8,*) konst1, konst2
read(8,*) omgsq
close(8)
write(6,*) 'jjmax = ', jjmax,' kkmax = ',kkmax,' llmax = ',llmax
write(6,*) 'ddelr = ',ddelr,' ddelz = ',ddelz,' ddelth = ',ddelth
write(6,*) 'konst1 = ',konst1,' konst2 = ',konst2
write(6,*) 'omgsq = ',omgsq
open(unit=9,file='/work/ul5149/junk/density2',form='unformatted'
1 ,status='old')
if(isyma .ne. 1) then
  read(9) rho
else
  if (kkmax .gt. kmax2/2) then
    write(6,*)' model grid structure too large for no symmetry...'
    stop
  endif
  read(9) rho(:,kmax2/2+l:kmax2,: )
  rho(:,kmax2/2:1:-1,:) = rho(:,kmax2/2+l:kmax2,:)
endif
close(9)
cjc Setting up jn
cjc
scfomega » sqrt(omgsq)
circp = 2.0*pi/scfomega
forall (j*=2:jmaxl) jn(j,:,:,::) = ((1.*j-l.S)*ddelr)**2*scfomega
dencen = maxval(rho)
xxx » maxloc(rho(2: jmaxl,2:kmaxl,:))
jrmax = xxx(1)+1
kzmax = xxx(2)+1
Ithmax = xxx(3)
den = dencen
ptmass = 1.0
print*, 'dencen = ',dencen
print*, 'maxdenloc = ',xxx(1), ',xxx(2), ',xxx(3)
denex = 1.0e-7 *dencen
where (rho .le. denex) rho = denex
gamma = xnhyd
cjc These binaries have different K values
cjc konst1 for -Pi/2 < theta <= Pi/2
cjc konst2 for Pi/2 < theta <= 3/2*Pi
cjc
lmx1o4 = 1max4
lmx3o4 = 3*lmx1o4
eps(:,:,:lmx1o4) = konst1*rho(:,:,:lmx1o4)**((1.0/xn)/(gamma - 1.0))
eps(:,:,:lmx3o4+1:1max4) = konst1*rho(:,:,:lmx3o4+1:1max4)**((1.0/xn)
1 /(gamma - 1.0)
eps(:,:,:lmx1o4+1:lmx3o4) = konst2*rho(:,:,:lmx1o4+1:lmx3o4)**((1.0/xn)
1 /(gamma - 1.0)
cjc This part handles epsmin, since the density maximum can have one of
cjc two konst.
cjc
konst = konst1
if((1thmax .gt. lmx1o4) .and. (1thmax .le. lmx3o4)) konst = konst2
epsmin = konst/(gamma - 1.0)*denex**(1.0/xn)
epsmax = konst/(gamma - 1.0)**((1e-4*den)**(1.0/xn)
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
force j=2 zone to be axisymmetric

forall (k=l:kmax2)
   rho(2,k,:) = sum(rho(2,k,:),dim=1)/float(lmax)
   jn(2,k,:) = sum( jn(2,k,:),dim=1)/float(lmax)
   eps(2,k,:) = sum(eps(2,k,:),dim=1)/float(lmax)
end forall

c make sure values of variables on borders of grid are correctly initialized

if (isyma.ne.1 .and. isyma.ne.2) then
   rho(1,:,:> = rho(2,:,:)
   eps(1,:,:) = eps(2,:,:)
   jn(1,:,:) = jn(2,:,:)
else
   rho(1,:,:) = cshift(rho(2,:,:),dim=2,shift=lmax/2)
   eps(1,:,:) = cshift(eps(2,:,:),dim=2,shift=lmax/2)
   jn(1,:,:) = cshift( jn(2,:,:),dim=2,shift=lmax/2)
endif

rho(:,l,:) = rho(:,2,:)
eps(:,l,:) = eps(:,2,:)
jn(:,l,:) = jn(:,2,:)
rho(:,kmax2,:) = rho(:,kmax1,:)
eps(:,kmax2,:) = eps(:,kmax1,:)
jn(:,kmax2,:) = jn(:,kmax1,:)

alnewz=1.0
rof3n=ddelr
zof3n=ddelz
dtheta = ddelth
alnewr=1.0
c .............. following are dummy values for izumi's model.
pindex = xn
con2 = ptmass
rrr2 = ddelr*(jjmax-1)
omcen = ddelz*(kkmax-3)
toverw = 1.0
write(6,136)pindex,con2,rrr2,omcen,dencen,towerw,ddelr,
ddelz,alnewz,almax,kzmax
136 format(///,5x,'pindex = ',lp,ell.3,/,5x,'ptmass = ',lp,ell.3,/,5x,
      ' rpimax = ',lp,ell.3,/,5x,' zmax = ',lp,ell.3,/,5x,' denmax = ',
      2 lp,ell.3,/,5x,' toverw = ',lp,ell.3,/,5x,'delr = ',lp,ell.3,5x,
      3 ' delz = ',lp,ell.3,/,5x,'alnewz = ',lp,ell.3,/,5x,'almax = ',
      4 ' l5,/,5x,'kzmax = ',i5,///)
return
end
c up initial variables that are used globally, reads the input deck,
c and calls izumi to set up the model.
c
c... itype tells whether initial or read in model.
c
= -2, read in sandford-whitaker-klein model from ntap.
c = 1  means read model number nmodl from unit ntap.
c = 2  polytrope model, initially axisymmetric.
c = 3  polytrope model, perturbed w/ maclaurin bar mode.
c = 4  polytrope model, perturbed w/ random density perturbation
=c = 5  polytrope, perturbed w/ straight perturbation.
c = 6  3D polytrope, no perturbation

ntape = tape unit from which to read model.
c
nmodl = number of model in that file to be used.
c
c include 'grid.h'

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ : phi, rho

real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ : p, eps

real, dimension (jmax2, kmax2, lmax) :: mass
!hpf$ align with hydtpl :: mass
common /cellmass/ : mass

real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, js
!hpf$ align with hydtpl :: s, t, a, u, w, js
common /som/ : s, t, a, u, w, js

real, dimension (jmax2, kmax2, lmax) :: r,z,rhf,zhf
!hpf$ align with hydtpl :: r,z,rhf,zhf
common /grid/ : r,z,rhf,zhf

real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rminus
real, dimension (jmax2, kmax2, lmax) :: zplus
real, dimension (jmax2, kmax2, lmax) :: zminus
!hpf$ align with hydtpl :: rplus, zplus, rminus, zminus
common /jgrid/ : rplus, zplus, rminus, zminus

real :: rcloud,constp,delt,bdytem,den,time,cormas
common /bloc7/ : rcloud, constp, delt, bdytem, den, time, cormas

real :: x0, xzn, konst, xnhyd
common /ptrope/ : x0, xzn, konst, xnhyd

real :: tmass, enew, elost, edif, phichk
integer :: klocat
common /inside/ : tmass, enew, elost, edif, phichk, klocat

real :: deltar, deltaz
common /jgrid2/ : deltar, deltaz

real :: omgfrm
common /rotfrm/ : omgfrm

real :: rof3n, zof3n, alnewr, alnewz, dtheta

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
common /grid2/ rof3n, zof3n, ainewr, ainezw, dtheta
real :: ptmass
common /cenpob/ ptmass

real :: scfre, scfrhoma, scfomega
common /scfout/ scfre, scfrhoma, scfomega

real :: cirp
common /normal/ cirp

real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax

integer :: fftstart, callpot
common /vids/ fftstart, callpot

real :: rco, tco, zco, vrco, vtco, vzco
common /codefs/ rco, tco, zco, vrco, vtco, vzco

real :: vmxin, vmxout
common /velmax/ vmxin, vmxout

integer :: zeroout
common /zerot/ zeroout

integer :: indx, isoadi, itstep
real :: allow
common /timst/ indx, isoadi, allow, itstep

integer :: igrd2
common /old2/ igrd2

real :: pi, grav
common /blok6b/ pi, grav

real :: incline, J20
common /pstr/ incline, J20

real :: cvheat
common /tempcv/ cvheat

integer :: icom
common /comflg/ icom

c

I add these arrays so setup could see phip and phic.
real, dimension(jmax2,kmax2,lmax) :: rhop, phip, phic
!hpfs$ distribute(block,block,*): rhop, phip, phic
common /potarray1/ rhop, phip, phic

cjc 3/8/96 These are the interval and stopping point that the fft
cjc 3/8/96 data is taken at.
cjc

integer :: jintrvl, jouter, klvl
common /ffoutput/ jintrvl, jouter, klvl

real :: symfact
common /symmetry/ symfact
real :: mlost, jlost
common /outer/ mlost, jlost

real :: coolst, coolrt
common /cool/ coolst, coolrt

real, dimension (jmax2, kmax2, lmax) :: rhomr, rhomz, rhoml
!hpf$ align with hydtpl :: rhomr, rhomz, rhoml

real, dimension (jmax2, kmax2, lmax) :: rholmr, rhomlr, rhomzr, rhomzlr
!hpf$ align with hydtpl :: rhomlr, rhomzr, rhomzlr

character*6 botbdy, topbdy, sidbdy

real, dimension (jmax2, kmax2, lmax) :: rhjkl, denom
!hpf$ align with hydtpl :: rhjkl, denom

integer :: itstrt, itstop, idiag, itype, ntape, nmodl, istor
integer :: isym, maxtrm, isyma, j, k, l, icall, icool, calcnt
integer :: tmploc(3)
integer :: isoadix, one, zero, frnum, intrvl
integer :: iomg, inewomg, rotfrm
real :: egrav
real :: newomgfrm
real :: dummy, tapen
real :: theta, delr, delz
real :: curlyr, gxxxx, xmu/83.14, 1.666667e0, 2.0/
data curlyr, gxxxx, xmu/83.14, 1.666667e0, 2.0/
100 format(lp,5elS.8)
102 format(12i7)
103 format(' this will be an isothermal collapse starting at timestep
1 number',i4,' and',/', going through timestep number',i6,
2 full diagnostics every',i3,' steps.','///)
104 format(' this will be an adiabatic collapse starting at timestep
1 number',i4,' and', '/', going through timestep number',i4,
2 full diagnostics every',i3,' steps.','///)
105 format(' reading in model number',i3,' from tape unit',i3,
3 , ///)
106 format(///,10x,'rotating polytropes... n =',Opf5.2,/',27x,'1 + 1/n =',
1 Op, e10.3,/',32x,'k =',Ip, e10.3,/
107 format(7x,a6,x,a6,x,a6,
110 format(///, 'input deck values:',/,8x, 'rcloud, constp, ptmass, den, cormas',/',8x,
2 'sigm, del, amp0, wxin, mwxout', '/',8x,
3 'itstrt, itstop, itdy, isoadi, itypes, ntape, nmodl, istor, igrid, isym',
4 '13x,'maxtrm', '/',8x,
5 'botbdy, topbdy, sidbdy', ',//)
112 format(6x, 'no symmetries assumed.')
113 format(6x, 'full 2-pi, but symmetry thru equatorial plane. ')
114 format(6x, 'pi-symmetry and symmetry thru equatorial plane. ')
115 format(6x, '2d run; no other symmetries. ')
116 format(6x, '2d run with symmetry thru equatorial plane. ')
117 format(6x, 'boundary condition at bottom of grid is ",a6," ,
1 //,6x, 'boundary condition at top of grid is ",a6,",',
2 /,6x,'boundary condition at side of grid is "",a6,"",',//)
135 format(//,10x,'-----this run assumes pi-symmetry-----',//)
137 format(//,10x,'rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr',//)
138 format(//,10x,'adjusting jm to put into rotating frame.',//)

itstep = 1
open(unit=7)
read(7,*) xn,xnl,konst,xnhyd
read(7,*) rcloud,consp,ptmass,den,cormas
read(7,*) omgfrm
read(7,107) botbdy,topbdy,sidbdy
read(7,*) incline,J20
read(7,*) fftstart,callpot
read(7,*) zeroout
read(7,*) frnum,intrvl
read(7,*) callcnt,rotfrm
read(7,*) romsg,inewomg,newomgfrm

isyma=abs(isym)
if(botbdy.eq.'WALL') botbdy = 'wall'
if(botbdy.eq.'DIRICH') botbdy = 'dirich'
if(topbdy.eq.'WALL') topbdy = 'wall'
if(topbdy.eq.'DIRICH') topbdy = 'dirich'
if(sidbdy.eq.'WALL') sidbdy = 'wall'
if(sidbdy.eq.'DIRICH') sidbdy = 'dirich'

for polytropes, isoadi must = 3.
if(isoadi.eq.2.or.isoadi.eq.3.or.isoadi.eq.9) write(6,104) itstrt,istop,idiag
if(isoadi.eq.1) write(6,103) itstrt,istop,idiag
if(vmxin.le.0.0) vmxin=0.5
if(vmxout.le.0.0) vmxout=0.25

for polytropes, isoadi must = 3.
if(isoadi.eq.1) write(6,112)
if(isoadi.eq.2) write(6,113)
if(isoadi.eq.3) write(6,114)
if(isoadi.eq.8) write(6,115)
if(isoadi.eq.9) write(6,116)
write(6,117) botbdy,topbdy,sidbdy
if(isoadi.eq.3) write(6,106) xn,xnl,konst

/* initialize following parameters for all runs.
pi = 3.1415926535897e0
grav = 1.0
*/
\[
d\theta = 2.0 \pi / l_{\text{max}} \\
\text{if} (|i_{\text{abs}}(\text{isym}).eq.3) \text{d\theta} = 0.5 \cdot \text{d\theta} \\
\text{if} (|i_{\text{abs}}(\text{isym}).eq.3) \text{write}(6,135) \\
\gamma = x_{\text{hyd}} \\
c_{\text{vheat}} = \frac{\text{curlr}}{(x_{\mu} \cdot (\gamma - 1.0))} \\
\theta = -0.5 \cdot \text{d\theta} \\
\text{c if you are reading in old model from disk (itype=1), then} \\
\text{c some of the following parameters may be reset at that time.} \\
\text{time} = 0.0 \\
\text{c (note: epsmin will be used inside eoc to limit eps.)} \\
\text{c (note: epsmax used inside state to limit eps.)} \\
\text{tmass} = 0.0 \\
enew = 0.0 \\
elost = 0.0 \\
edif = 0.0 \\
phichk = 0.0 \\
klocat = 0 \\
cjww \\
\text{c the following are for central object wandering} \\
\text{icon} = 0 \\
c \text{ finished with central object (may be reset reading in old model)} \\
c \text{ finished setting up general parameters.} \\
c \text{ set up r's and z's.} \\
c \text{ first determine what ainear, ainevz, rof3n, and zof3n are by} \\
c \text{ either reading in a new model structure from input file...} \\
icall = 1 \\
\text{write}(6,110) \\
\text{write}(6,100) x_{n}, x_{n1}, k_{\text{onst}}, x_{\text{nyd}} \\
\text{write}(6,100) r_{\text{cloud}}, c_{\text{onstp}}, p_{\text{tmax}}, d_{\text{en}}, c_{\text{ormas}} \\
\text{write}(6,100) \sigma_{\text{ma}}, c_{\text{irp}}, a_{\text{mp0}}, v_{\text{mixin}}, v_{\text{mout}} \\
\text{write}(6,102) i_{\text{tsr}}, i_{\text{tstop}}, i_{\text{diag}}, i_{\text{soadi}}, i_{\text{itype}}, n_{\text{tape}}, n_{\text{modl}}, \\
i1_{\text{istor}}, i_{\text{grid}}, i_{\text{sym}}, m_{\text{xtrm}} \\
\text{write}(6,107) b_{\text{otbdy}}, t_{\text{opbdy}}, a_{\text{idbdy}} \\
\text{if}(i_{\text{soadi}}.eq.3 .and. i_{\text{itype}}.gt.1) \text{ then} \\
\text{if}(i_{\text{itype}}.eq.6) \text{ then} \\
tapen = \text{float}(\text{ntape}) \\
\text{open(unit=ntape, file='d\text{nsity}', form='formatted', status='old')} \\
& \text{call scfin(tapen,itype)} \\
\text{else} \\
\text{call scfin3d(isyma)} \\
\text{endif} \\
\text{endif} \\
c \text{ or reading in an old model from disk...} \\
\text{if}(i_{\text{itype}}.gt.1) \text{go to 440} \\
\text{write}(6,105) n_{\text{modl}}, n_{\text{tape}} \\
c \text{......length} = 8 \cdot (5 \cdot j_{\text{max}}^2 \cdot k_{\text{max}}^2 \cdot l_{\text{max}} + 7) \\
c = 5744696 \text{ for } (64,32,64) \\
c = 1436216 \text{ for } (64,32,16) \\
c = 739896 \text{ for } (32,32,16) \\
cjww \text{open(unit=ntape)} \\
\text{write}(6,*)' isyma = ', isyma \\
\text{open(unit=ntape, file='/work/ul5149/junk/bessel/fort.32.8')}

I added frnum, cirp, and ptmass for continuation files.
I changed u and w to t and s, so I read in momenta instead of velocities. I also read in phi for a starting point for the poisson solver.

\[ \text{if(isyma .eq. 1 .or. isyma.eq.2) then} \]
\[ \text{read(ntape)s , t , jn, rho, eps, phip, rof3n, zof3n, alnewr, alnewz,} \]
\[ \text{delta, time, elost, frnum, cirp, ptmass,konst, vxin, ftstart,} \]
\[ \text{write(6,*)' read in model: initial values for central object'} \]
\[ \text{write(6,*)'rco = ',rco,*tco * ',tco} \]
\[ \text{write(6,*)'vrco= ',vrco,*vtco= ',vtco} \]
\[ \text{write(6,*)'denex = ',denex} \]
\[ \text{else} \]
\[ \text{read(ntape)s , t , jn, rho, eps, phip, rof3n, zof3n, alnewr, alnewz,} \]
\[ \text{delta, time, elost, frnum, cirp, ptmass,konst, vxin, ftstart,} \]
\[ \text{write(6,*)' read in model: initial values for central object'} \]
\[ \text{write(6,*)'rco = ',rco,*tco * ',tco} \]
\[ \text{write(6,*)'vrco= ',vrco,*vtco= ',vtco} \]
\[ \text{write(6,*)'denex = ',denex} \]
\[ \text{endif} \]
\[ \text{close(ntape) } \]
\[ \text{open(unit=21, file='cool.input', form='formatted')} \]
\[ \text{read(21,*)' icool, coolst, coolrt} \]
\[ \text{close(21) } \]
\[ \text{den=maxval(rho)} \]
\[ \text{denex=1.e-7*den} \]
\[ \text{epsmin = konst/(gamma - 1.0) * denex**(1.0/xn)} \]
\[ \text{note: epsmin will be used inside eoc to limit eps.} \]
\[ \text{epsmax = konst/(gamma - 1.0) * (1.e-4*den)**(1.0/xn)} \]
\[ \text{note: epsmax used inside state to limit eps.} \]
\[ \text{440 continue} \]
\[ \text{then calculate r,z,rhf, and zhf} \]
\[ \text{deltar=rof3n} \]
\[ \text{deltaz=zof3n} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) r(j,k,l)=(float(j)-2.0)*deltar} \]
\[ \text{rcloud = r(jmax1,1,1)} \]
\[ \text{if (isym == 1) then} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) z(j,k,l)=(float(k)-1.5)*deltaz} \]
\[ \text{else} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) z(j,k,l)=(float(k)-1.5)*deltaz} \]
\[ \text{endif} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) rhf(j,k,l) = (float(j)-1.5)*deltar} \]
\[ \text{if (isym == 1) then} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) zhf(j,k,l) = (float(k)-0.5-kmax2/2)*deltaz} \]
\[ \text{else} \]
\[ \text{forall(j=1:jmax2,k=1:kmax2,l=1:lmax) zhf(j,k,l) = (float(k)-0.5)*deltaz} \]
\[ \text{endif} \]
\[ \text{set these up as globals to ease shifting in execution 5/3/92} \]
\[ \text{deltar = delr} \]
\[ \text{deltaz = delz} \]
\[ \text{rplus = eoshift(r, dim=1, shift= 1)} \]
rplus(jmax2,:, :) = rplus(jmax1,:, :) + deltar
zplus = eoshift(z, dim=2, shift=1)
zplus(:, kmax2, :) = zplus(:, kmax1, :) + deltax
rhfminus = eoshift(rhf, dim=1, shift=-1)
rhfminus(1, :) = rhfminus(2, :) - deltar
zhfminus = eoshift(zhf, dim=2, shift=-1)
zhfminus(:, 1, :) = zhfminus(:, 2, :) - deltax

cjw end of new globals

cjcrum scfomega = 2.*pi/cirp, scfomega is not used if moving frame is off.
cjcmaybe if(itype .ne. 6) scfomega = 2.*pi/cirp
  scfomega = 2.*pi/cirp
  if(rotfrm .gt. 0) then
    if(iomg.eq.0) then
      omgfrm=scfomega
    else
      omgfrm=newomgfrm
    endif
  else
    omgfrm=0.e0
    newomgfrm=0.e0
  endif
if(itype.eq.1) go to 450

450 if((itype.eq.1).and.(inewomg .eq. 1)) go to 550

if(iomg.eq.0) then
  if(omgfrm.eq.0.0e0) go to 457
  write(6,138)
  jn = jn - omgfrm*rhf**2
else
  if(newomgfrm.eq.0.0e0) go to 457
  write(6,138)
  jn = jn - newomgfrm*rhf**2
endif
where (rho.le.denex) jn 2 0.0
457 continue

450 if((itype.eq.1).and.(inewomg .eq. 1)) go to 550

if(iomg.eq.0) then
  if(omgfrm.eq.0.0e0) go to 457
  write(6,138)
  jn = jn - omgfrm*rhf**2
else
  if(newomgfrm.eq.0.0e0) go to 457
  write(6,138)
  jn = jn - newomgfrm*rhf**2
endif
where (rho.le.denex) jn 2 0.0
457 continue

550 continue

if(omgfrm.ne.0.0) write(6,137) omgfrm

... from rho, find p and eps for the polytrope.
on 12/18/86 at lsu, changed this so that
a. eps(j,k,l) is calculated inside izumi
b. p(j,k,l) is determined, as at every other
time step, by calling state(3)
do 560 l=1,lmax
do 560 k=1,kmax1
do 560 j=1,jmax1

on 4/26/86 at lsu, commented out array cv(j,k,l) to allow code to fit into 4-megabytes.

On 7/18/95 Goto added a few lines of text.

Note: cv(j,k,l) only used here and in 'rad'.

3/25/98 I changed this to calculate the mass array.
(cmass of each fluid element)
symfact = 1.0
if (isym .eq. 2) symfact = 2.0
if (isym .eq. 3) symfact = 4.0
mass(2: jmax,2:kmax, :) = rho(2: jmax,2:kmax,:) * rhf(2: jmax,2:kmax, :)
mass = mass * dtheta * deltar * deltaz
 tmass = sum(mass)
 tmass = symfact * tmass
print*, 'tmass = ', tmass
if (callcnt .eq. 1) then
 print*, 'ptmass = ', ptmass
 print*, 'cmass = ', 1.0/ptmass
else
 print*, 'no central mass'
endif

call state(icool)

3/14/96 I changed the continuation file to write out the momenta
3/14/96 for r and z instead of the velocities. So, now I need
3/14/96 to get the velocities instead of vice versa as it was
3/14/96 previously.
3/14/96 This calculates the velocities from the momenta.
call vel(isym,botbdy,topbdy,sidbdy)

3/14/96 This calculates the angular momenta. (Not done in vel)
c a = j * rho
 tmploc = maxloc(a)
print*, 'max(a) = ', a(tmploc(1), tmploc(2), tmploc(3))
print*, 'at j = ', tmploc(1), 'at k = ', tmploc(2), 'at l = ', tmploc(3)
tmploc = maxloc(jn)
print*, 'max(jn) = ', jn(tmploc(1),tmploc(2),tmploc(3))
print*, 'at j = ',tmploc(1),'at k = ',tmploc(2),'at l = ',tmploc(3)
print*, 'max(jn) = ',maxval(jn)
a(1,:) = a(2,:,1)
a(:,1,:) = a(:,2,1)

set position of central object

if (icom .eq. 0) call comset
icom = 0
icall = 1
if (callcnt .eq. 1) icall = 0
zero = 0
one = 1

egraw = symfact*0.5 * dtheta*deltar*deltar*sum( rhf(2:jmax,2:kmax,:)
\ & * phip(2:jmax,2:kmax,:)
\ & *rho(2:jmax,2:kmax,:))
print*, 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
print*, 'egraw = ',egraw
if(callpot.gt'.O) then
call potsetup(isyma)
call potdcall, isyma)
else
call centril(zero,isyma)
phi = phic
endif
egraw = symfact*0.5 * dtheta*deltar*deltar*sum( rhf(2:jmax,2:kmax,:)
\ & * phip(2:jmax,2:kmax,:)
\ & *rho(2:jmax,2:kmax,:))
print*, 'egraw = ',egraw
print*, 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

klvl = 2
if(isym .eq. 1) klvl = kmax2/2
if(itype.ne.1) then
tmploc = maxloc(rho(2:jmax1,2:kmax1,:))
print*, 'tmploc(1) = ',tmploc(1),', tmploc(2) = ',tmploc(2),
\ , klvl = ',tmploc(3)
tmploc(1) = tmploc(1) + 1
tmploc(2) = tmploc(2) + 1
endif

changed this so it would write out the mode data at
the max density and at dmmax+5 and dmmax+10.

ffstart =tmploc(1) - 5
fftstart = tmploc(1) - 5
klvl = tmploc(2)
do j = jmax,fftstart,-1
if(rho(j,tmploc(2),tmploc(3)) .gt. denex) then
jouter = j
exit
endif
end do
jintrl = (jouter - fftstart)/3
print*, 'jouter = ',jouter,' jintrl = ',jintrl
klvl = klvl
jouter = 2*jintrl
endif
print*, 'ffstart = ', ffstart
print*, 'jouter = ', jouter, ' jintrvl = ', jintrvl
print*, ''
print*, 'omgfrm = ', omgfrm

.. the following statement is put just to initialize 'chmax'
for subroutine delta.
if (rho(jmax2, kmax1, l).eq.0.0) rho(jmax2, kmax1, l) = 1.0
return
end

-----------------------------------------------------------------
subroutine state(icool)
-----------------------------------------------------------------
real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi, rho
real, dimension (jmax2, hmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps
common /states/ p, eps
real :: xn, xnl, konst, xnhyd
common /ptrope/ xn, xnl, konst, xnhyd
real :: rcloud, constp, delt, bdytem, den, time, cormas
common /blok7/ rcloud, constp, delt, bdytem, den, time, cormas
real :: cirp
common /normal/ cirp
real :: coolest, coolrt
common /cool/ coolest, coolrt
integer :: icool
real :: konstnew

.. polytropic relation.
when calling from main routine, eps(j,k,l) = epsilon
if (icool .gt. 0) then
  konstnew = konst*(1.0 - coolrt*(time/cirp - coolest))
p = konstnew * rho**xnl
else
  p = konst * rho**xnl
endif

.. changed to handle polytropes with different K's
p = rho*(xnhyd - 1.0)*eps
c----------------------------------------------------------------
c Calculate the velocities from the specific momenta. c
C centered differently than the density.

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpfs$ align with hydtpl :: phi, rho
common /pois/ phi, rho

real, dimension (jmax2, kmax2, lmax) :: s, t, u, w, jn
!hpfs$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn

real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpfs$ align with hydtpl :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf

real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpfs$ align with hydtpl :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus

real :: deltard, deltaz
common /jgrid2/ deltard, deltaz

character*6 botbdy, topbdy, sidbdy
integer isym, isyma

real, dimension (kmax2) :: srhok, srhokm
!hpfs$ align with hydtpl(*,*,*) :: srhok, srhokm

real, dimension (jmax2, kmax2, lmax) :: rhjkl, denom
!hpfs$ align with hydtpl :: rhjkl, denom

real, dimension (jmax2, kmax2, lmax) :: rhomr, rhomz, rhoml
!hpfs$ align with hydtpl :: rhomr, rhomz, rhoml

integer :: k, l

c here i initialize the variables
isym = 0
rhjkl = 0.0
denom = 0.0
rhomr = 0.0
rhomz = 0.0
rhoml = 0.0
rhomlr = 0.0
rhomlz = 0.0

return
end
rhomzr = 0.0
rhomlr = 0.0
k = 0
1 = 0

c have a processor fetch the surrounding values of rho for use in
c future calculations (if we put all these eoshifts in the calculation,
c so much temporary stack space is used we run out of memory).

isyma = iabs(isym)
rhomr = eoshift(rho, dim=1, shift=-1)
rhomz = eoshift(rho, dim=2, shift=-1)
rhoml = cshift(rho, dim=3, shift=-1)
rhomlr = eoshift(rhoml, dim=1, shift=-1)
rhomlz = eoshift(rhoml, dim=2, shift=-1)
rhomzr = eoshift(rhomz, dim=1, shift=-1)
rhomlrr = eoshift(rhomlr, dim=1, shift=-1)

c... now calculate u and w velocities from s and t momenta for
c all j,k,l. rhjkl is density in a cell centered
about a vertex of our usual coordinate system (hence, an
c volume averaged eight-point type summation).

rhjkl = ( (rhomzr + rhomlr) * (r**2-rhfminus**2)
& + (rhomz + rhomlz) * (rhf**2-r**2) ) * (z-zhfrainus)
& + ( (rhomr + rhomlr) * (r**2-rhfminus**2)
& + ( rho + rhoml) * (rhf**2-r**2) ) * (zhf - z))

c to protect that first zone, check where denom is zero.
c not used to protect now that denom isn't zero at j=2
c this protection is now performed in other places
denom = (2.0*(rhf**2-rhfminus**2) * deltaz)
where (denom .ne. 0.0)
rhjkl = rhjkl / denom
elsewhere
rhjkl = 0.0
endwhere

c Since s and t are vertex centered specific momenta, the velocities are
c just divided by the density at that point.

where (rhjkl .ne. 0.0)
  u = s/rhjkl
  w = t/rhjkl
elsewhere
  u = 0
  w = 0
endwhere

c This special treatment of the z-velocity on the z-axis was removed
c for the moment, as currently the code is used for disk models,
c eliminating all material from the central regions. This probably
c should be put back in if centrally condensed objects are to be evolved.

c... special treatment of t, on z-axis.
!hpfs$ independent
forall (k=3:kmaxl)
srhok(k) = sum(rho(2,k,:))
srhokm(k)= sum(rho(2,k-1,:))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[
w(2,k,1) = t(2,k,1) \times \text{float}(lmax)\\
& \times (rthok(k) + (zhf(2,k,1) - z(2,k,1)) + \\
& \times rthokm(k) + (z(2,k,1) - zhf(2,k,1))) \\
\]
endforall

c... to make sure u on z-axis is zero
\[
u(2,:,1) = 0.0
\]
c... establish boundary conditions on u and w (just to be sure... if
\c the boundary conditions were properly established on s and t,
\c these should be okay).
c
\c z-axis -> neumann condition holds except for isyma = 1 or 2.
c
if (isyma ne 1 and isyma ne 2) then
cerr u(1,:,1) = u(3,:,1) NOT u(3,:,1)
u(1,:,1) = -u(3,:,1)
w(1,:,1) = w(3,:,1)
!hpf$ independent
forall (1 = 1 : lmax)
w(2,:,1) = w(2,:,1)
endforall
else
\[u(1,:,1) = -cshift(u(3,:,1), \text{dim}=2, \text{shift}=lmax/2)\]
w(1,:,1) = cshift(w(3,:,1), \text{dim}=2, \text{shift}=lmax/2)
!hpf$ independent
forall (1 = 1 : lmax)
w(2,:,1) = w(2,:,1)
endforall
endif

c... k=2
if (isyma eq 1 or isyma eq 8) go to 96

c neumann condition if symmetry thru equatorial plane assumed.
c
94 continue
\[u(:,1,:) = u(:,3,:)\]
w(:,1,:) = -w(:,3,:)
w(:,2,:) = 0.0
goto 910
96 continue
\c if botbdy = 'wall', then set neumann conditions.
if (botbdy eq 'wall') go to 94
\c if botbdy = 'free', then set velocities at k=1 to those
c derived for k=2; i.e., a free-flowing boundary condition.
if (botbdy ne 'free') go to 910
\[u(:,1,:) = u(:,2,:)
\]
w(:,1,:) = w(:,2,:)
910 continue
\c if botbdy ne free or wall, then a dirichlet boundary condi
c is assumed and no modification of k=1 is made.
c
\c top of grid.
if (topbdy ne 'wall') go to 915
\c neumann conditions at k=kmaxi
\[u(:,kmax2,:) = u(:,kmax,)
w(:,kmax2,:) = w(:,kmax,)
w(:,kmax1,:) = 0.0
\]
goto 921
915 if (topbdy ne 'free') go to 920
\c free boundary condition being used at k=kmaxi.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
u(:,kmax2,:) = u(:,kmax1,:)
w(:,kmax2,:) = w(:,kmax1,:)
goto 921

920 continue
c    if topbdy.ne. free or wall, then a dirichlet boundary
c    condition is assumed
c    the following assignment assures flow off of grid
    but not back on
w(:,kmax1,:) = 0.5*(abs(w(:,kmax,:))+w(:,kmax,:))
w(:,kmax2,:) = 0.0
u(:,kmax2,:) = 0.0

921 continue
c    side of grid.
    if(sidbdy.ne.'wall')go to 925
c    neumann conditions at j=jmax1.
u(jmax2,:,:) = -u(jmax,:,:)
w(jmax2,:,:) = w(jmax,:,:)
u(jmax1,:,:) = 0.0
goto 931

922 continue
925 if(sidbdy.ne.'free')go to 930
c    free boundary condition being used at j=jmax1.
u(jmax2,:,:) = u(jmax1,:,:)
w(jmax2,:,:) = w(jmax1,:,:)
goto 931

930 continue
c    if sidbdy.ne. free or wall, then a dirichlet boundary
c    condition is assumed
c    the following assignment assures flow off of grid
    but not back on
u(jmax1,:,:) = 0.5*(abs(u(jmax,:,:))+u(jmax2,:,:))
u(jmax2,:,:) = 0.0
w(jmax2,:,:) = 0.0

931 continue
c    finished boundary conditions on u and w.
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

C-------------------------------------------------------------------
C vlimit(isyma)
C-------------------------------------------------------------------

real, dimension (jmax2, kmax2, lmax) :: phi, rho
!hpf$ align with hydtpl :: phi, rho
common /pois/ phi,rho
!
real, dimension (jmax2, kmax2, lmax) :: p, eps
!hpf$ align with hydtpl :: p, eps

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
common /states/ p, eps
   real, dimension (jmax2, kmax2, lmax) :: s, t, a, u, w, jn
!hpf$ align with hydtpl :: s, t, a, u, w, jn
common /eom/ s, t, a, u, w, jn
   real, dimension (jmax2, kmax2, lmax) :: r, z, rhf, zhf
!hpf$ align with hydtpl :: r, z, rhf, zhf
common /grid/ r, z, rhf, zhf
   real, dimension (jmax2, kmax2, lmax) :: rplus, zplus
real, dimension (jmax2, kmax2, lmax) :: rhfminus
real, dimension (jmax2, kmax2, lmax) :: zhfminus
!hpf$ align with hydtpl :: rplus, zplus, rhfminus, zhfminus
common /jgrid/ rplus, zplus, rhfminus, zhfminus
   real :: xn, xnl, konst, xnhyd
common /ptrope/ xn, xnl, konst, xnhyd
   real :: denex, epsmin, epsmax
common /freez/ denex, epsmin, epsmax
   real :: deltar, deltaz
common /jgrid2/ deltar, deltaz
   real :: vxin, vmaxout
common /velmax/ vxin, vmaxout
   integer :: indx, isoadi, itstep
real :: allow
common /timst/ indx, isoadi, allow, itstep
   real, dimension (jmax2, kmax2, lmax) :: rhjkl, denom
!hpf$ align with hydtpl :: rhjkl, denom
   real, dimension (jmax2, kmax2, lmax) :: rhmr, rhmz, rhml
!hpf$ align with hydtpl :: rhmr, rhmz, rhml
   real, dimension (jmax2, kmax2, lmax) :: rhmlr, rhmlz, rhmzl
!hpf$ align with hydtpl :: rhmlr, rhmlz, rhmzl
   integer :: l, k, isyma
real :: factor, d, gamma, emax, ascund, vlire
   real, dimension (kmax2) :: st
!hpf$ align st(:, :) with hydtpl(*, :, *)

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

set d = factor*denex, where denex (= 1.e-7*den, typically) is the initial value of background density. Then limit u, v, s, and t differently inside and outside of the gas cloud, depending on if rho-average is greater than or less than, respectively, the value of d. vmxin and vmaxout are read in from unit 5 inside setup. but, if isoadi=3, vmaxout is overruled by the value of vlim = 0.5*(sound speed at maximum density).
check the following...

cjc 1. array 'usign' determines the sign given to vlim; c

cjc it is +1 if j.le.jdecid, c

cjc it is -1 if j.gt.jdecid. c

cjc Changes 1/23/98 jdecid is no longer used c

c 2. jdecid = 41; this value may need to be changed. c

c 3. factor = 500.0 c

c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

data factor/500.0/

c have a processor pick up surrounding values of rho for later c calculations (placing eoshifts in calculations exhausts stack memory).

rhmr = eoshift(rho, dim=1, shift=-1)
rhomz = eoshift(rho, dim=2, shift=-1)
rhml = eoshift(rho, dim=3, shift=-1)
rhomlr = eoshift(rhoml, dim=1, shift=-1)
rhomlz = eoshift(rhoml, dim=2, shift=-1)
rhomzr = eoshift(rhomz, dim=1, shift=-1)
rhomlzr = eoshift(rhomlz, dim=1, shift=-1)

c set governing parameters
d = factor*denex

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c change added 12/18/86 at Isu to allow (gamma .no. 1+1/xn)
c

c want to find maximum sound speed in fluid.....
c since eps = p/(rho * (gamma - 1)), c

c and a**2 = gamma * p/rho, c

c then maximum sound speed can be gotten from maximum eps. c

gamma = xnhyd
emax = maxval(eps)
asound = gamma * (gamma - 1.0) * emax
vlim = 0.5*sqrt(asound)

c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
if(isodi.ne.3)vlm=vmxout

c
rhjkl = (
& (rhomzr + rhomlr) * (r**2-rhminus**2)
& + ( rhomz + rhoml ) * (rhf**2-r**2) ) * (z-zhfminus)
& + ( rho + rhoml ) * (rhf**2-r**2) ) * (z-hf - z))

c to protect j=1 again...
denom = 2.0*(rhf**2-rhminus**2)* deltaz

where (denom .ne. 0.0)
rhjkl = rhjkl / denom
elsewhere
rhjkl = 0.0
endwhere

c as in vel, special treatment of the vertical velocity on the axis
has been removed. Should be placed back in if centrally condensed
c objects are to be evolved.
c... special treatment of t, on z-axis.
c------ jwu 12/89 ----- 
c first of all, find average rho on the axis.
rhjkl(2,:,1)=(sum(rho(2,:,i),dim=2)*(zhf(2,:,1)-z(2,:,1))+
 & sum(rhonz(2,:,i), dim=2)*
 & (z(2,:,1)-eoshift(zhf(2,:,1),dim=1,shift=-1))
 & (zhf(2,:,1)-zhfminus(2,:,1))
 & )/(zhf(2,:,1)-zhfminus(2,:,1))
 & /float0.max)
!hpfs independent
forall (1=1:lmax)
rhjkl(2,:,1) = rhjkl(2,:,1)
endforall

c... the following two where statements are now unnecessary because
their tasks are performed below-(now that rhjkl isn't zero at j=2)
c where (rhjkl(2,:,1) .ge. d .and. abs(w(2,:,:)) .gt. vmxin)
c w(2,:,1) = vmxin * w(2,:,1) / abs(w(2,:,:))
c t(2,:,1) = w(2,:,1) * rhjkl(2,:,1)
c endwhere

c where (rhjkl(2,:,1) .lt. d .and. abs(w(2,:,:)) .gt. vlim)
c w(2,:,1) = vlim

t(2,:,1) = w(2,:,1) * rhjkl(2,:,1)
c endwhere

c... limit the allowed maximum velocities
c Perform the limiting of the velocities.
cjc 1/23/98 Hot denom holds the sign of u

cjc 1/23/98 and then w.
cjc where (u < 0.0 )
denom = -1.0 
elsewhere
denom = 1.0 
endwhere

where ((rhjkl >= d) .and. (abs(u) > vmxin))
u = vmxin*denom 
s = u * rhjkl 
end where

where ((rhjkl < d) .and. (abs(u) > vlim))
u = vlim*denom 
s = u * rhjkl 
end where

where (u < 0.0 )
denom = -1.0 
elsewhere
  denom = 1.0 
endwhere

where ((rhjkl >= d) .and. (abs(w) > vmxin))
w = vmxin * denom 
t = w * rhjkl 
end where

where ((rhjkl < d) .and. (abs(w) > vlim))

\[ w = v \lim \cdot \text{denom} \]
\[ t = w \cdot \text{rhjkl} \]
end where

c... set \( u \) on \( z \)-axis.
\[ u(2,:,:) = 0.0 \]
\[
\text{if (isyma.ne.1.and.isyma.ne.2) then}
\text{cerr u(1,:,:)=u(3,:,:) NOT u(3,:,:)}
\text{u(1,:,:)=\(-u(3,:,:)
\text{w(1,:,:)=w(3,:,:)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)}
\text{else}
\text{s(1,:,:)=\(-s(3,:,:), dim=2, shift=\text{lmax}/2\)
\text{t(1,:,:)=\(t(3,:,:), dim=2, shift=\text{lmax}/2\)
\text{!hpf$ independent}
\text{forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) t(2,k,1) = st(k)/float(lmax)}
\text{!hpf$ independent}
\text{forall (k=3:kmax1) st(k)=sum(t(3,k,:),dim=1)}
\text{!hpf$ independent}
\]
\[ c \]
forall (k=3:kmaxl) t(2,k,1) = st(k)/float(lmax)
!hpf$ independent
forall (l=1:lmax) t(2,:,l) = t(2,:,1)
endif

c... set s,t on equatorial plane, equatorial sym.
s(:,1,:) = s(:,3,:)
t(:,2,:) = 0.0
t(:,1,:) = -t(:,3,:)
c the following assignment assures flow off of grid
but not back on for dirichlet side and top conditions

return
end
APPENDIX B: INPUT FILES

C-------------------------------------------------------------
C           cool.input
C-------------------------------------------------------------
# note !!! Check polytropic index on first line!!!
# n=3/2 gamma = 4/3 /30/97
# n=3/2 gamma = 5/3 9/4/97
1 29.447 0.0625
VITA

John Edward Cazes, Jr. was born in Pineville, Louisiana, on July 10, 1967. He graduated from Louisiana Tech University with a bachelor of science degree in applied physics in 1990, and a master's degree two years later. In 1992, he began his graduate career in the Department of Physics and Astronomy at Louisiana State University. He expects to receive the Degree of Doctor of Philosophy in physics at the end of the summer of 1999.
Candidate:  John E. Cazes, Jr.

Major Field:  Physics

Title of Dissertation:  The Formation of Short Period Binary Star Systems from Stable Self-Gravitating Gaseous Bars

Approved:

[Signatures]

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

[Signatures]

Date of Examination:

June 30, 1999