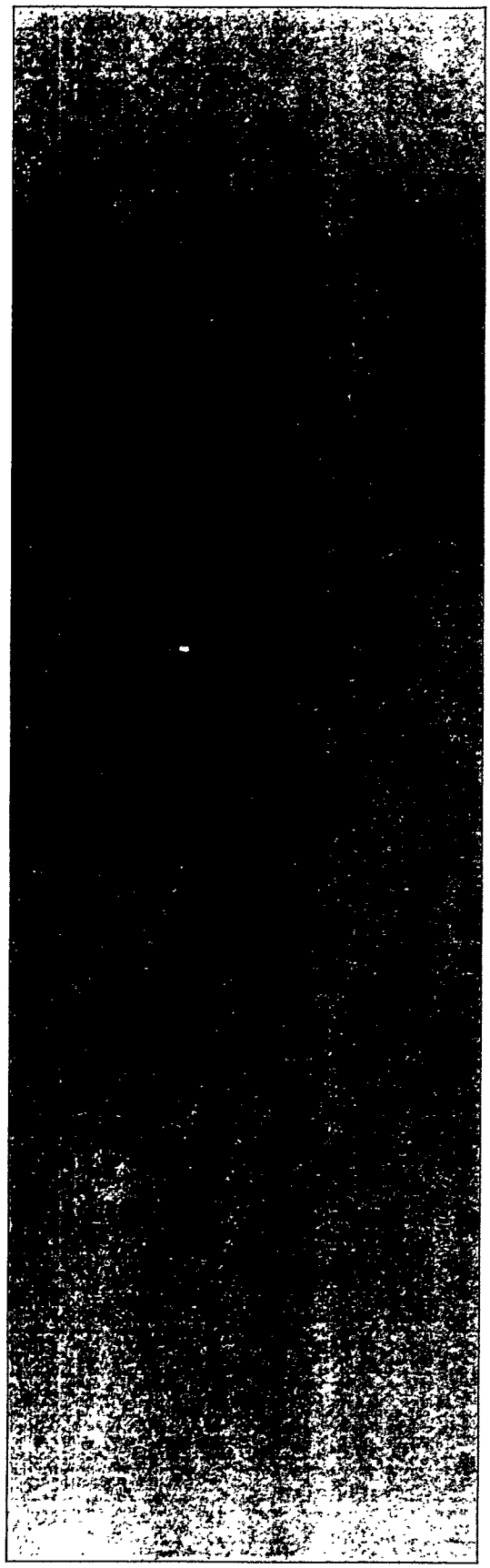
LA-13685-T
Thesis



An Implicit Smooth Particle Hydrodynamic Code

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

This thesis was accepted by the Department of Engineering, University of New Mexico, Albuquerque, New Mexico, in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The text and illustrations are the independent work of the author and only the front matter has been edited by the CIC-1 Writing and Editing Staff to conform with Department of Energy and Los Alamos National Laboratory publication policies.

An Affirmative Action/Equal Opportunity Employer

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LA-13685-T
Thesis

Issued: April 2000

An Implicit Smooth Particle Hydrodynamic Code

Charles E. Knapp

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545

DEDICATION

This is dedicated to my parents Kenneth and Barbara Knapp.

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy
Engineering

The University of New Mexico
Albuquerque, New Mexico
May 2000

Table of Contents

List of Figures

List of Tables

An Implicit Smooth Particle Hydrodynamic Code

by

Charles E. Knapp

A. S., Physics, Mesa Jr. College, Grand Junction, Colorado, 1965
B. A., Physics, University of California, Riverside, 1967
M. S., Astro-Geophysics, University of Colorado, Boulder, 1976
Ph. D., Engineering, University of New Mexico, Albuquerque, 2000

ABSTRACT

An implicit version of the Smooth Particle Hydrodynamic (SPH) code SPHINX has been written and is working. In conjunction with the SPHINX code the new implicit code models fluids and solids under a wide range of conditions. SPH codes are Lagrangian, meshless and use particles to model the fluids and solids. The implicit code makes use of the Krylov iterative techniques for solving large linear-systems and a Newton-Raphson method for non-linear corrections. It uses numerical derivatives to construct the Jacobian matrix. It uses sparse techniques to save on memory storage and to reduce the amount of computation. It is believed that this is the first implicit SPH code to use Newton-Krylov techniques, and is also the first implicit SPH code to model solids.

A description of SPH and the techniques used in the implicit code are presented. Then the results of a number of tests cases are discussed, which include a shock tube problem, a Rayleigh-Taylor problem, a breaking dam problem, and a single jet of gas problem. The results are shown to be in very good agreement with analytic solutions, experimental results, and the explicit SPHINX code. In the case of the single jet of gas case it has been

demonstrated that the implicit code can do a problem in much shorter time than the explicit code. The problem was, however, very unphysical, but it does demonstrate the potential of the implicit code. It is a first step toward a useful implicit SPH code.

Chapter I

An Overview & the Goals

A. Introduction

The goal of the research discussed in this dissertation is to develop a code, which is an implicit version of the Smooth Particle Hydrodynamic (SPH) approach to modeling fluid motion, and then to use it to study a select set of examples. The new code has been developed as an addition to an existing explicit SPH code called SPHINX. The SPHINX code was developed at Los Alamos National Laboratory [18], [22], [78], [79], [92], [93], [94], and has the capability to model fluids and solids, using SPH techniques. The desire is to move the SPHINX code into a new regime where it can use larger time-steps and model low-speed flow and near-steady-state problems. Ultimately, it is envisioned that SPHINX will be able to switch automatically between explicit and implicit time-stepping as conditions change within a given problem, although this is not part of this dissertation.

The number of possible new applications that the implicit code could bring to the SPHINX code would be numerous. Problems that change slowly with time or are near-steady-state, such as plastic flow, would be possible. For example, Oran and Boris [64] discuss the use of implicit methods in their Chapter 3 in which they discuss the modeling of a laminar flame propagating through a tube of combustible gas, and estimate that the computation could take up to 3000 years of computer time using conventional explicit methods. To remain stable, explicit methods are restricted to very small time-steps because of the need to resolve shock waves and velocities on the order of the sound speed. Implicit methods only need to model velocities on the order of the speed of the flame,

which is typically three orders of magnitude slower than the sound speed, and these methods could remain numerically stable but would give up some accuracy. That reduces the computer time to about 3 years. They further claim that another reduction of a factor of 500 can be gained by using adaptive-gridding to avoid gridding up voids, which brings the computational time down to about two days, which is more reasonable. SPH codes do not use grids, so that advantage would automatically be built in.

Astronomers want to calculate stellar and galactic models over very long periods of time, and to run the models many times with different parameters in an effort to fit their calculations to the observations. Very large time-steps are essential to be able to do this in one's lifetime, so implicit methods are commonly used in astronomy. Stellingwerf [76] [77] enumerated a number of ways for analyzing astrophysical models once the Jacobian matrix has been constructed. Solving this matrix equation is the crux of the implicit method. He points out that once the Jacobian is set up, then the "options include (1) forward time integration, (2) relaxation to steady-state, (3) stability of steady-state and time evolution, (4) numerical stability check, and (5) driven oscillations." Of course, these methods can be used in many other fields of numerical modeling besides astronomy.

B. Fluid Methods

Modeling of fluids usually follows one of two basic techniques. One method uses the fluid equations referenced to the laboratory frame, by defining a fixed mesh or grid and modeling the fluid flowing through the mesh. This technique uses the fluid equations in the Eulerian form. The other method fixes the mesh to the fluid and calculates the distor-

tion of the mesh as the fluid moves. In this method the mass within each cell of the mesh remains constant. This technique uses the fluid equations in the Lagrangian form.

The SPH technique, which was originally developed for astrophysical work and is fundamentally a Lagrangian approach, does not define a mesh, but instead models fluids and solids using particles, with each particle having its own set of physical properties assigned to it, such as position, velocity, density, internal energy, and more (see Fig. I. 1).

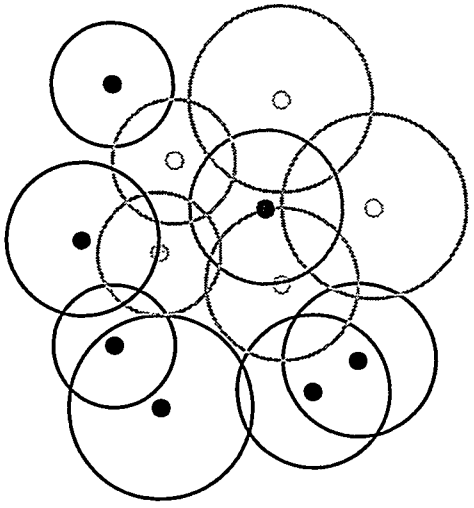


Fig. I. 1. An example of particles (dots) and their circles of influence, which may be of different radii and change with time. Each particle has a local set of neighbors influencing its motion. The particles may have different physical properties such as position, velocity, density, pressure, internal energy, and more.

This method starts with the Lagrange form of the fluid equations, and then by using two approximations, reduces the partial differential equations (PDEs) to ordinary differential equations (ODEs). These approximations are referred to as the kernel approximation and the particle approximation. Each particle has constant mass, which is analogous to the usual Lagrangian approach in which the mass within each cell of the mesh is held constant. Each particle has a sphere or circle of influence and set of neighbors as

determined by overlapping circles of influence. For instance, in Fig. I. 1. the circle of the medium gray particle has as its neighbors the light gray particles, but the black ones are not neighbors because their circles do not overlap with the medium gray particle. The circle represents a smoothing function, which is a Gaussian-like function that is highest at the particle, or the center of the circle, and falls off radially to zero at the edge of the circle. The particles are moved according to the fluid equations, and the smoothing functions interpolate the fluid properties between the particles.

Since the SPH method has no grid of cells, one of its main advantages is that there is no mesh tangling, which is a problem for most Lagrangian codes. Also, because there is no mesh, empty space does not have to be included in the grid, as is often required in the typical Eulerian code, even with the use of such methods as adaptive-gridding.

The SPH method also has the usual advantage of a Lagrangian code over an Eulerian code, in that contact discontinuities between fluids can be tracked. As two or more materials mix, they can be tracked because each particle has its own material properties. SPH can go beyond that, because particles can become thoroughly mixed, which is very difficult for a gridded Lagrangian code to calculate. Another advantage of SPH is that it is not much more difficult to write a three-dimensional (3D) code than to write a one- or two-dimensional code. Once the 1D code is written, the 2D and 3D parts can be added very easily to the same code.

There are some disadvantages with SPH. It is generally not as accurate as the gridded codes. There is an instability that is unique to SPH in the modeling of solids, where the particles can unphysically clump together when under tension. This problem is

referred to as the tension instability. Non-conservation of angular momentum is another problem that has been encountered in SPH. This problem has been addressed successfully by Dilts [22], [23] using a moving-least-square (MLS) method, but it is in general a more time-consuming computation than SPH. Another problem encountered in SPH is that boundaries are not modeled well. The particles at the edge of objects have no neighbors outside the object, so their densities are less than those for particles internal to the object, and one would like the density to be the same all the way out to the edge. MLS can handle this problem quite well also, but again it is a more time-consuming method.

One other problem encountered in SPH is that the spherical kernels can prove to be insufficient for unevenly distributed particles. For example, if the particles are stretched or squeezed in one direction more than another, then the particles can move apart so far - for example in the horizontal - that the spheres or circles of influence no longer overlap, but in the vertical they may be squeezed so tightly that they have many neighbors in the vertical but none in the horizontal. The calculation falls apart when particles that should be influencing each other are not. Attempts to solve this problem have been tried with varying degrees of success. One approach is to use elliptical kernels that stretch out as particles move apart. Another approach is to introduce more particles in the gaps as the original particles move apart. This approach is referred to as particle-splitting because the mass must remain constant, and therefore it has to be split up appropriately among the particles.

The explicit version of SPHINX uses primarily a Runge-Kutta method to do the time-stepping for solving the set of ODEs. It also has packages to do Leap-Frog and

Predictor-Corrector time-stepping, both of which are explicit methods.

C. Implicit Methods

The main subject of this dissertation is another time-stepping package, which will be the first implicit method to be added to the SPHINX code. One other implicit SPH code has been written, but for astrophysical use by Timmes [86], and it will be discussed later in this chapter. The SPHINX code is used primarily for modeling interacting solids and fluids, as opposed to astrophysical use, so the new implicit SPH code is believed to be the first one to model solids. The new code is also believed to be the first implicit SPH code to use Newton-Krylov methods for solving the linear system, which will be discussed later in Chapter III.

Implicit codes are used mainly because they are usually unconditionally stable with any time-step size. They do lose accuracy with increased time-step size, but the solutions do not become unstable; that is, they do not go off to infinity, or go to zero and stay there, or oscillate wildly (see Oran and Boris [64], page 94) as explicit codes do if the time-step size exceeds a limit known as the Courant condition. The Courant condition basically says that the spatial-step size divided by the time-step (which can be thought of as a velocity) should be greater than the greatest velocity expected in the fluid being modeled. Typically the largest velocities of interest are sound waves, but when modeling low-velocity flow, these velocities are of little interest and are usually ignored. The Courant condition requires an explicit code to take such tiny time-steps to remain stable that the code can take much too long to solve the problem.

The implicit code is not restricted by the Courant condition to remain stable, but,

to help maintain accuracy of the desired features of a problem, the time-step should still be as close as is practical to that prescribed by the Courant condition associated with the physics of interest. What is "practical" is decided by a trade-off between the amount of computer time to run the problem with the desired accuracy on the implicit code, as compared to the run time and accuracy of the explicit code. That is, if one is willing to give up some accuracy in exchange for shorter total run time, then the implicit code may be the one to use. One would like to run the implicit code with a large enough time-step so that the total computer time would beat the total run-time of the explicit code and still maintain an acceptable accuracy, which is often the case if the problem is near a steady-state solution, or the problem is not changing much over large periods of time. The choice of time-step for the implicit code has not been well defined yet, but it would ideally be based on the desired accuracy. A first attempt is discussed in Chapter III, Section I.

The main disadvantage of an implicit code is that it requires the solution to a huge number of simultaneous equations or a linear system, and hence requires the formation of a very large matrix. Inversion of the matrix has been the conventional method for finding a solution, and so implicit methods are typically computationally intensive. The large matrix can grow to take up most of the memory of any computer because the user will want more resolution and details included. More modern methods of solving linear systems use iterative methods rather than actually inverting the large matrices. The iterative methods do help speed up implicit codes. but they are still computationally intensive per time-step as compared to explicit codes. Iterative methods have become the subject of a major effort in research of numerical methods and the topic of a large body of journal arti-

cles and textbooks. Some conventional and iterative methods will be discussed in Chapter III.

To the knowledge of the author, only one other implicit SPH code has been written, and that is by Dr. Francis X. Timmes [86]. His code was developed for astrophysical use and includes self-gravity between particles. It uses the momentum equation and the energy equation, but not the continuity equation. He calculates densities by a summation method. The neighbor search routine in his code is different from that used in SPHINX in that, for a given particle, its neighbor particles are determined by whether or not the other particles fall within the radius of its sphere of influence, as opposed to overlapping spheres of influence. By implication then, given two particles with different smoothing lengths, the one with the larger radius may influence the other but not vice versa. Because equal and opposite action is not maintained between particles, energy is not necessarily conserved. However, Dr. Timmes claims that this problem can be minimized.

An example of the way neighbors are counted in Timmes' code can be seen in Fig I. 1. The two particles in the lower left have different size circles. The one with the smaller circle falls within the larger circle, so it is a neighbor of the one with the larger circle. But since the particle (the dot) with the larger circle does not fall within the smaller circle, it is not a neighbor of that particle. One way to maintain equal and opposite reaction between particles, in Timmes' method, would be to keep the circles all of equal radius. The radii, or smoothing lengths, could still change, but they would have to change equally for all particles, which is probably a reasonable approach for many problems. Timmes does, however, use variable smoothing lengths or radii.

The new implicit code, which is the focus of this research, has a number of differences. First, the continuity equation is included as an option to the user. Second, particles are counted as neighbors if their spheres of influence overlap. This feature has the effect that any two particles have an equal and opposite reaction on each other, which allows for conservation of energy. Other differences include the use of iterative techniques, also known as Krylov methods, for solving the linear system. Also, a version of the implicit code has been written that makes use of matrix-free methods within the iterative techniques. This version has only been partially successful, but the method will be discussed in more detail in Chapter III.

The development of the new implicit code for SPHINX has involved five major stages. The first stage was to develop a code based on the analytic derivation of the implicit form of the SPH fluid equations. This set of equations involves a Jacobian matrix of derivatives. The first version of the implicit code, following Timmes' approach, used the Lower and Upper (LU) decomposition method to factor the matrix, and used a fourth-order Rosenbrock solver. The second stage replaced the analytic Jacobian matrix with a numerical Jacobian. The third stage replaced the LU decomposition with a selection of Krylov solvers. The fourth stage attempted a modification to the Krylov solvers to make them matrix-free. The modification replaces the step in the iterative solver where the matrix-vector multiply appears with an approximation that involves only vector operations. This stage was not completely successful. The fifth stage involved adding a Newton-Raphson iteration to improve the nonlinear convergence and going to sparse storage and sparse calculations. The implicit method is covered in more detail in Chapter III.

Presented in Chapter II are the basic concepts, assumptions, and mathematics used in SPH. The implicit approach is presented in Chapter III. The last two chapters discuss a set of examples on which both the implicit and explicit codes have been tested. Chapter IV includes a three-particle problem, a rarefaction problem, a shock-tube problem, a Rayleigh-Taylor instability problem, a breaking dam problem, and a single expanding jet of gas. Chapter V discusses a set of problems involving neutral plasma jets in 2D and 3D, that have application to nuclear fusion.

Chapter II

Basics of Smooth Particle Hydrodynamics

A. Introduction To Smooth Particle Hydrodynamics

Smooth Particle Hydrodynamics (SPH) is a relatively new numerical approach to simulating hydrodynamic problems on the computer. SPH was introduced by Lucy (1977) [51], Gingold & Monaghan (1977) [30], and Monaghan (1982) [58], and has been improved on by a growing community of users since then (see Benz [10], Hernquist & Katz [34], Swegle et al. [82], Libersky & Randles [49]). It was used initially by the astrophysical community to model galaxies and star formation [30], [34], [57], [73], [86]. With the inclusion of material-strength models it has also been found to be useful for modeling solids [48]. It has been used to model projectiles, solid or fluid, impacting targets of various kinds to study cratering, damage, and breakup [39], [79], [93]. SPH has also been found to be useful for modeling fracturing of solids such as rock with granular boundaries [11], [52], [83]. Several good reviews exist by Benz [10], Monaghan [59], and Wingate [92]. The current discussion, however, will be restricted mainly to fluids.

As briefly discussed in Chapter I, fluid dynamic problems are usually solved numerically, and the fluid equations are typically cast into one of two common frames of reference. One is the lab frame and the other is the fluid frame of reference. The resulting sets of equations are known, respectively, as the Eulerian and Lagrangian forms. One form can be converted into the other with an appropriate coordinate transformation. Many different ways of solving these equations numerically have been developed. Both formulations generally use a grid or mesh which divides space into cells. The codes for

either method can be written in one, two, or three dimensions, each dimension adding increasing complexity because the phenomena occurring at each boundary of each cell have to be taken into account.

The Eulerian method keeps track of the fluid as it flows in and out of the different boundaries of each cell. The cells are fixed in space and do not move. It is a rigid grid. One disadvantage of this approach is that, if there is more than one fluid, it is difficult to keep track of the two fluids as they mix. There are ways to handle this problem but they can make the code very complicated. One example is known as Front or Interface Tracking [33], [64], which will not be covered in this dissertation.

The Lagrangian method overcomes the mixing problem by not allowing the fluid to leave the cell in which it starts, but rather the cells move and deform to account for the fluid motion. The mass in each cell remains constant, but the density can change as the cell size changes, depending on the pressures and temperatures in each cell. The interface between two fluids is easy to keep track of as long as the cells do not become too distorted. The cells typically start out in a regular grid or pattern but can soon become highly deformed and even tangled, which is a serious problem with this method. The codes are usually programmed to stop running or redo the mesh at this point because the results often become unphysical under these conditions.

The SPH method is a Lagrangian approach and is derived from the Lagrangian equations, but each cell can be thought of as having been reduced to a point, which is referred to as a particle, and the mass of each particle is constant. As a result, the SPH approach is mesh free, because it has no grid of cells. A lucid discussion of the SPH theory can be found in the Ph. D. dissertation by Fulk [27].

Each particle has the various fluid properties associated with it, and the particles are moved in time according to the fluid equations. Each particle has a position (x, y, z), a velocity $\mathbf{v} = (v_x, v_y, v_z)$, a mass m, internal energy e, and a smoothing length h assigned to it; from these, pressure P, temperature T, density ρ, etc. are computed for each particle. Each particle has a set of SPH equations that are derived from the usual Lagrangian fluid equations:

The Momentum Equation $$\frac{d}{dt}\mathbf{v} = -\left(\frac{1}{\rho}\right)\nabla P, \qquad (2.1)$$

The Continuity Equation $$\frac{d\rho}{dt} = -(\rho)\nabla\bullet\mathbf{v}, \qquad (2.2)$$

The Energy Equation $$\frac{de}{dt} = -\left(\frac{P}{\rho}\right)\nabla\bullet\mathbf{v}. \qquad (2.3)$$

Each particle also has a sphere of influence defined by a kernel function that determines how strongly each particle interacts with its neighbors as a function of distance between them. The kernel function is a bell-shaped function and is commonly made up of B-spline functions with compact support on the particle's sphere of influence. The Gaussian function has also been used as a kernel.

Some of the advantages of the SPH approach are the following:

1. There is no mesh tangling.
2. It is almost as easy to write a 2D or 3D code as it is a 1D code, which is not true for some approaches.
3. Different types of fluids are easy to track as they mix because each particle has its own material identity.
4. Empty space does not have to be zoned up as is often required in Eulerian mesh codes.
5. Fracturing and breaking up of solid objects can be modeled.

B. The SPH Approximations and the SPH Equations

The approximations used to reduce the Lagrangian fluid equations from PDEs to ODEs are the kernel approximation and the particle approximation. The kernel approximation, also called the kernel estimate, is based on using a bell-shaped interpolating function W, and is used in the same manner as the Dirac delta function. Either can be used to approximate an arbitrary function. The particle approximation divides the fluid into particles, which in general are much larger than atoms or molecules.

Any function $A(\boldsymbol{r})$ can be written as a superposition of delta functions $\delta(\boldsymbol{r}-\boldsymbol{r}')$:

$$A(\boldsymbol{r}) = \int A(\boldsymbol{r}')\delta(\boldsymbol{r}-\boldsymbol{r}')d\boldsymbol{r}' \ , \tag{2.4}$$

and following Monaghan [59], the interpolating function, or kernel, is used similarly where $W(|\boldsymbol{r}-\boldsymbol{r}'|, h) \rightarrow \delta(\boldsymbol{r}-\boldsymbol{r}')$ as $h \rightarrow 0$, where h determines the width of the function:

$$\langle A(\boldsymbol{r})\rangle = \int A(\boldsymbol{r}')W(\boldsymbol{r}-\boldsymbol{r}', h)d\boldsymbol{r}' , \tag{2.5}$$

where the angle brackets indicate an approximation. By multiplying the fluid equations by $W(|\boldsymbol{r}|, h)$ and integrating, the kernel approximation is formed.

To evaluate the integral, the particle approximation is used. Assume the fluid is divided into particles with masses $m_1, ..., m_N$, and volume elements (m_j / ρ_j), then the contribution to Eq. (2.5) by the jth particle can be represented as:

$$A(\boldsymbol{r}'_j)W(\boldsymbol{r}-\boldsymbol{r}'_j, h)\frac{m_j}{\rho(\boldsymbol{r}'_j)}, \tag{2.6}$$

and summing over all such terms will approximate the integral in Eq. (2.5). The kernel W has units of inverse volume, so that, when multiplied by the mass over density, the units cancel. Hence the units of term (2.6) are those of $A(\boldsymbol{r})$. Thus, using the particle approx-

imation, Eq. (2 .5) becomes a summation over all particles:

$$\langle A(\boldsymbol{r})\rangle \approx \sum_{j=1}^{N} \frac{m_j}{\rho(\boldsymbol{r}'_j)}\ (A(\boldsymbol{r}'_j))\ W(\boldsymbol{r}-\boldsymbol{r}'_j, h). \tag{2 .7}$$

The fluid equations are the momentum equation, the continuity equation, and the internal energy equation. The momentum equation, Eq. (2 .1), can be rewritten as,

$$\frac{d}{dt}\mathbf{v} = -\left(\frac{1}{\rho}\right)\nabla P = -\nabla\left(\frac{P}{\rho}\right) - \frac{P}{\rho^2}\nabla\rho, \tag{2 .8}$$

and then approximating the operands of the gradients for the kernel approximation:

$$\frac{d}{dt}\mathbf{v} \approx -\nabla\left\langle\frac{P}{\rho}\right\rangle - \frac{P}{\rho^2}\nabla\langle\rho\rangle. \tag{2 .9}$$

Replacing the averages with Eq. (2 .7), and using a more brief notation $W(|\boldsymbol{r}_i-\boldsymbol{r}_j|, h) = W_{ij}$, where the subscripts indicate that W_{ij} is a function of both particles i and j:

$$\frac{d\mathbf{v}_i}{dt} = -\nabla_i \sum_{j=1}^{N} \frac{m_j}{\rho_j}\left(\frac{P}{\rho}\right)_j W_{ij} - \left(\frac{P}{\rho^2}\right)_i \nabla_i \sum_{j=1}^{N} \frac{m_j}{\rho_j}(\rho)_j W_{ij}\ . \tag{2 .10}$$

The gradients operate only on the quantities for particle i, and W_{ij} is the only thing within the summations that is a function of i, so the gradients operate only on the kernels, hence:

$$\boxed{\frac{d\mathbf{v}_i}{dt} \approx -\sum_j m_j\left[\left(\frac{P}{\rho^2}\right)_j + \left(\frac{P}{\rho^2}\right)_i\right]\nabla_i W_{ij}} \tag{2 .11}$$

The continuity equation Eq. (2 .2) can be rewritten as, and approximated by:

$$\frac{d\rho}{dt} = -(\rho)\nabla\bullet\mathbf{v} = -\nabla\bullet(\rho\mathbf{v}) + \mathbf{v}\bullet\nabla\rho \approx -\nabla\bullet\langle\rho\mathbf{v}\rangle + \mathbf{v}\bullet\nabla\langle\rho\rangle\ . \tag{2 .12}$$

Then using the SPH approximation Eq. (2 .7):

$$\boxed{\frac{d\rho_i}{dt} \approx -\nabla_i\bullet\sum_{j=1}^{N}\frac{m_j}{\rho_j}(\rho\mathbf{v})_j W_{ij} + \mathbf{v}_i\bullet\nabla_i\sum_{j=1}^{N}\frac{m_j}{\rho_j}(\rho)_j W_{ij}} \tag{2 .13}$$

which simplifies to:

$$\boxed{\frac{d\rho_i}{dt} \approx \sum_{j=1}^{N} m_j (\mathbf{v}_i - \mathbf{v}_j) \bullet \nabla_i W_{ij}} \tag{2.14}$$

The energy equation Eq. (2.3) follows the continuity equation fairly closely:

$$\frac{de}{dt} = -\left(\frac{P}{\rho}\right)\nabla\bullet\mathbf{v} = -\left(\frac{P}{\rho}\right)\frac{1}{\rho}[\nabla\bullet(\rho\mathbf{v}) - \mathbf{v}\bullet\nabla\rho] \approx -\left(\frac{P}{\rho^2}\right)[\nabla\bullet\langle\rho\mathbf{v}\rangle - \mathbf{v}\bullet\nabla\langle\rho\rangle] \tag{2.15}$$

Making the kernel and particle approximations:

$$\frac{de_i}{dt} \approx -\left(\frac{P}{\rho^2}\right)_i \left[\nabla_i \bullet \sum_{j=1}^{N} \frac{m_j}{\rho_j}(\rho\mathbf{v})_j W_{ij} - \mathbf{v}_i \bullet \nabla_i \sum_{j=1}^{N} \frac{m_j}{\rho_j}(\rho)_j W_{ij}\right], \tag{2.16}$$

and simplifying;

$$\boxed{\frac{de_i}{dt} \approx \left(\frac{P}{\rho^2}\right)_i \sum_{j=1}^{N} m_j (\mathbf{v}_i - \mathbf{v}_j) \bullet \nabla_i W_{ij}} \tag{2.17}$$

There have been a number of forms of the three SPH equations derived using different approximations (for a list, see [28] or [95]). Another form of the momentum equation seems a bit contrived, but it has proven to be the most robust form and is the one used in SPHINX. It starts with Eq. (2.1) and adds a term with the gradient of a constant, which is, of course, zero, and it is always valid to add zero to an equation.

$$\frac{d\mathbf{v}}{dt} = -\left(\frac{1}{\rho}\right)\nabla P - \left(\frac{P}{\rho}\right)\nabla(1) \approx -\left(\frac{1}{\rho}\right)\nabla\langle P\rangle - \left(\frac{P}{\rho}\right)\nabla\langle 1\rangle . \tag{2.18}$$

Then the gradients are replaced by the SPH approximations.

$$\frac{d\mathbf{v}_i}{dt} \approx -\left(\frac{1}{\rho}\right)_i \nabla_i \sum_{j=1}^{N} \frac{m_j}{\rho_j}(P)_j W_{ij} - \left(\frac{P}{\rho}\right)_i \nabla_i \sum_{j=1}^{N} \frac{m_j}{\rho_j}(1) W_{ij}, \tag{2.19}$$

which simplifies to:

$$\boxed{\frac{d\mathbf{v}_i}{dt} \approx -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j}\right)\nabla_i W_{ij}} \tag{2.20}$$

An alternative way to derive the energy equation is to allow the pressure to have spatial gradients. Then the following treatment yields a somewhat different result:

$$\frac{de}{dt} = -\left(\frac{P}{\rho}\right)\nabla\bullet\mathbf{v} = -\nabla\bullet\left(\frac{P}{\rho}\mathbf{v}\right) + \mathbf{v}\bullet\nabla\left(\frac{P}{\rho}\right) \approx -\nabla\bullet\left\langle\frac{P}{\rho}\mathbf{v}\right\rangle + \mathbf{v}\bullet\nabla\left\langle\frac{P}{\rho}\right\rangle . \qquad (2.21)$$

$$\frac{de_i}{dt} \approx -\left[\nabla_i\bullet\sum_{j=1}^{N}\frac{m_j}{\rho_j}\left(\frac{P}{\rho}\mathbf{v}\right)_j W_{ij} - \mathbf{v}_i\bullet\nabla_i\sum_{j=1}^{N}\frac{m_j}{\rho_j}\left(\frac{P}{\rho}\right)_j W_{ij}\right], \qquad (2.22)$$

and simplifying;

$$\frac{de_i}{dt} \approx \sum_{j=1}^{N} m_j\left(\frac{P}{\rho^2}\right)_j (\mathbf{v}_i - \mathbf{v}_j)\bullet\nabla_i W_{ij} . \qquad (2.23)$$

According to Monaghan [59], either Eq. (2.17) or Eq. (2.23) is satisfactory for ideal gases, but for metallic equations of state, the latter has slightly better energy conservation.

The continuity equation is sometimes omitted and replaced with a summation using Eq. (2.7), in which A($\boldsymbol{r}$) is replaced with $\rho(\boldsymbol{r})$:

$$\boxed{\langle\rho\rangle_i \approx \sum_{j=1}^{N} m_j W_{ij}} \qquad (2.24)$$

This summation is an approximation of the density of the *i*th particle and is in agreement with the kernel and particle approximations.

C. The Kernel Function

The basic equations for the SPHINX code consist of three conservation laws, Eqs. (2.20), (2.14), and (2.17), sometimes (2.24), and a kernel function that determines how strongly each pair of particles interacts. Each particle *i* interacts only with its neighbor particles *j* that fall within a certain radius or sphere of influence.

The sphere of influence is an interpolating kernel function, usually a bell-shaped function, and determines the pressures acting on neighboring particles. It can be any of a number of functions such as a Gaussian or B-spline. Other kernels are discussed in the papers by Fulk [28] and Monaghan [56] and [58]. The kernel function is denoted as $W(\boldsymbol{r}, h)$, where $\boldsymbol{r}$ is the distance between particle i and its neighbor particle j, and has a smoothing length h, which determines the width of the function. That is, $\boldsymbol{r} = \boldsymbol{r}_i - \boldsymbol{r}_j$, where $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ are the position vectors of the particles i and j. The interpolating kernel function is required to have the following properties:

1. it is a symmetric, or even, function about $\boldsymbol{r} = 0$, and reduces to a Dirac delta function in the limit as h approaches zero,

$$\lim_{h \to 0} W(|\boldsymbol{r}|, h) = \delta(|\boldsymbol{r}|), \tag{2.25}$$

2. it is normalized to one,

$$\int W(|\boldsymbol{r}|, h) d\boldsymbol{r} = 1, \tag{2.26}$$

3. and while the following is not a requirement, the function usually has compact support to limit the number of neighbors, and for the SPHINX code it is assumed to be zero outside of $|\boldsymbol{r}| = 2h$,

$$W((|\boldsymbol{r}| > 2h), h) = 0. \tag{2.27}$$

So the radius of the sphere of influence for each particle is twice its smoothing length h, and the smoothing length does not have to be the same for all particles. In SPHINX, h is often allowed to vary with density. In this case, the average smoothing length between two particles i and j is used in the kernel: $\bar{h} = (h_i + h_j) / 2$. Note: the circles of Fig. I. 1 are of radius h, so when they just touch, the particles are $2h$ apart, or $2\bar{h}$ when h is variable.

The interpolating kernel function for the SPHINX code is a cubic B-spline curve of the form shown in the following Eq. (2 .28) (see also Fig II. 1) and is a function of the distance $r_{ij} \equiv |r|$ between the particles i and j with positions (x_i, y_i, z_i) and (x_j, y_j, z_j).

$$W_{ij} = \begin{cases} C(1 - \frac{3}{2}u_{ij}^2 + \frac{3}{4}u_{ij}^3), & \text{for}(0 \le u_{ij} < 1) \\ C\frac{1}{4}(2 - u_{ij})^3, & \text{for}(1 \le u_{ij} < 2) \\ 0, & \text{for}(u_{ij} \ge 2) \end{cases} \text{, where} \qquad (2.28)$$

$$u_{ij} \equiv \frac{r_{ij}}{h}, \text{ and } \quad r_{ij} \equiv \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \ ; \qquad (2.29)$$

and the positive root of Eq. (2 .29) is assumed for r_{ij} . C is a constant for normalizing the area under W_{ij} to one, and is different for each of the three dimensions; that is,

$$\text{for 1D: } C \equiv \frac{2}{3h}, \quad \text{for 2D: } C \equiv \frac{10}{7\pi h^2}, \quad \text{and for 3D: } C \equiv \frac{1}{\pi h^3}. \qquad (2.30)$$

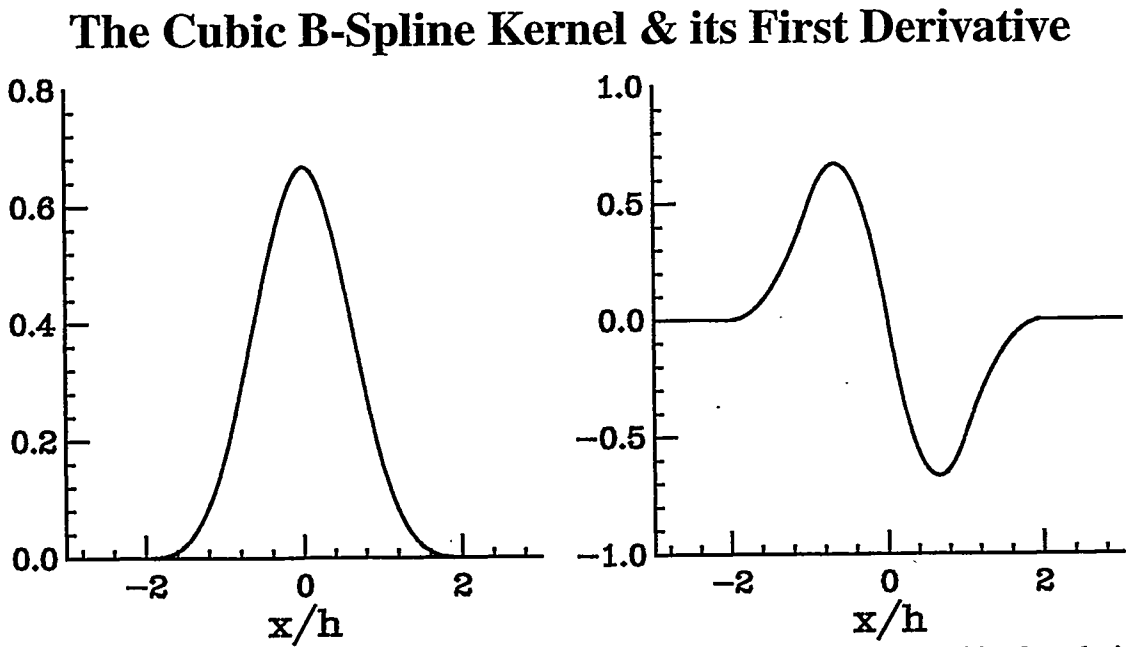


Fig II. 1 The cubic B-spline kernel W_{ij} (left) is an even function, and its first derivative (right) is odd. The kernel has zero slope at the origin and for x > 2h. Since x is always positive, only the right half of these functions is actually used in the calculations.

From the above equations one can see that W_{ij} is a function of r_{ij} and that i and j can be swapped in Eq. (2 .29) without affecting the value of r_{ij}. The same is true for W_{ij}, because W_{ij} is symmetric, which can also be seen in Fig. II. 1. The first derivative of W_{ij}, however, is an odd function, so there is a sign change for the derivatives when i and j are swapped, as is discussed in Section D, Chapter II. The index i runs from 0 to N–1, where N is the total number of particles and the index j runs over the number of neighbors for particle i. (The code described here is written in the programing language C, so the indices conveniently start at zero.)

For a 3D implicit SPH code there are eight ODEs per particle to describe their motions, derived from three conservation laws. (A 2D code requires six ODEs per particle, and a 1D code requires 4 ODEs per particle.) These are the rate equations of the particle's x_i, y_i, z_i positions, the x y z velocities (v_i^x, v_i^y, v_i^z), the density ρ_i, and the internal energy e_i. The eight dependent variables are x_i, y_i, z_i, v_i^x, v_i^y, v_i^z, ρ_i, and e_i, and the independent variable is time t.

The rate equations for position in the 3D implicit SPH code consist of three velocity equations that describe the motion of the particles:

$$\frac{dx_i}{dt} = v_i^x , \qquad \frac{dy_i}{dt} = v_i^y , \qquad \frac{dz_i}{dt} = v_i^z . \tag{2 .31}$$

The rate equation of the velocity, also known as the momentum equation, can be expressed in a number of ways. In the terminology of the SPHINX code they are of the form known as "Hydro-form 2" (Wingate and Stellingwerf, 1995 [95]), and are the same as Eq. (2 .20) but with artificial viscosity terms added. The momentum equation is a vector equation, so for the ith particle there is a rate equation for each velocity component:

$$\frac{dv_i^x}{dt} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} + \Pi_{ij} \right) \frac{\partial W_{ij}}{\partial x_i}, \tag{2.32}$$

$$\frac{dv_i^y}{dt} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} + \Pi_{ij} \right) \frac{\partial W_{ij}}{\partial y_i}, \tag{2.33}$$

$$\frac{dv_i^z}{dt} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} + \Pi_{ij} \right) \frac{\partial W_{ij}}{\partial z_i}. \tag{2.34}$$

The summations are over all particles j that are neighbors of particle i. Particle i is always included in its own neighbor list. The pressures of the ith and jth particles are given by P_i and P_j. Their densities are ρ_i and ρ_j, and the masses are m_i and m_j. The Π_{ij} is the artificial viscosity (for a definition see Monaghan [59] and [62]), which is assumed to be zero for the discussion of the analytic Jacobian in Chapter III. The derivatives of W_{ij} are discussed later (Section D of Chapter II, & Section B.2 of Chapter III).

The rate equation for density ρ_i of particle i, Eq. (2.14) is derived from the mass continuity equation, and its expanded SPH form is:

$$\frac{d\rho_i}{dt} = \sum_j m_j \frac{\rho_i}{\rho_j} \left\{ (v_i^x - v_j^x) \frac{\partial W_{ij}}{\partial x_i} + (v_i^y - v_j^y) \frac{\partial W_{ij}}{\partial y_i} + (v_i^z - v_j^z) \frac{\partial W_{ij}}{\partial z_i} \right\} \tag{2.35}$$

The rate equation for the internal energy e_i of particle i in the SPH form used in the SPHINX code (referred to as "Energy form 3") is:

$$\frac{de_i}{dt} = \sum_j m_j \left(\frac{P_i}{\rho_i \rho_j} + \frac{\Pi_{ij}}{2} \right) \left\{ (v_i^x - v_j^x) \frac{\partial W_{ij}}{\partial x_i} + (v_i^y - v_j^y) \frac{\partial W_{ij}}{\partial y_i} + (v_i^z - v_j^z) \frac{\partial W_{ij}}{\partial z_i} \right\} \tag{2.36}$$

One more equation is needed for closure, and that is usually the Equation of State (EOS) for calculating the pressure as a function of the densities and internal energies. The EOS can be formulated from any of a number of different models. For the derivation

of the analytic Jacobian of Chapter III, the perfect gas model is assumed and is given by:

$$P_i \equiv (\gamma - 1) e_i \rho_i \tag{2 .37}$$

where γ is the ratio of specific heats.

D. The First-Order Derivatives of the Kernel

For the explicit code, only the first-order spatial derivatives of the kernel are needed. For the implicit code both the first-order and second-order are needed, but the second-order spatial derivatives will be discussed in Chapter III. The kernel W_{ij} is given by Eq. (2 .28), and the spatial derivatives of W_{ij} are needed for Eqs. (2 .32) through (2 .36). The derivatives of W_{ij} are partial derivatives because it is a function of the three position variables (x_i, y_i, z_i). As noted from Eqs. (2 .28) and (2 .29), W_{ij} is a not a function of velocity, density, or energy. The first-order derivatives can be evaluated by first finding the derivatives of r_{ij} from Eq. (2 .29).

$$\frac{\partial r_{ij}}{\partial x_i} = \frac{1}{r_{ij}}(x_i - x_j) \equiv \frac{\Delta x_{ij}}{r_{ij}}, \text{ similarly: } \frac{\partial r_{ij}}{\partial y_i} \equiv \frac{\Delta y_{ij}}{r_{ij}}, \text{ and } \frac{\partial r_{ij}}{\partial z_i} \equiv \frac{\Delta z_{ij}}{r_{ij}}, \tag{2 .38}$$

where $\Delta x_{ij} \equiv (x_i - x_j)$ and similarly for $\Delta y_{ij} \equiv (y_i - y_j)$ and $\Delta z_{ij} \equiv (z_i - z_j)$. Then the derivatives of u_{ij}, from Eq. (2 .29), are:

$$\frac{\partial u_{ij}}{\partial x_i} = \frac{1}{h}\frac{\partial r_{ij}}{\partial x_i} \equiv \frac{1}{h}\frac{\Delta x_{ij}}{r_{ij}}, \qquad \frac{\partial u_{ij}}{\partial y_i} \equiv \frac{1}{h}\frac{\Delta y_{ij}}{r_{ij}}, \text{ and } \frac{\partial u_{ij}}{\partial z_i} \equiv \frac{1}{h}\frac{\Delta z_{ij}}{r_{ij}}. \tag{2 .39}$$

Hence, the derivatives of the first two parts of W_{ij}, Eq. (2 .28) (labeled ${W^a}_{ij}$ and ${W^b}_{ij}$) with respect to x_i, y_i, and z_i are:

$$\frac{\partial W^a{}_{ij}}{\partial x_i} = -C(3u_{ij} - \frac{9}{4}u_{ij}^2)\frac{\partial u_{ij}}{\partial x_i} = -\frac{C}{h^2}(3 - \frac{9}{4}u_{ij})\Delta x_{ij}, \quad \text{for } (0 \le u_{ij} < 1), \quad (2.40)$$

$$\frac{\partial W^b{}_{ij}}{\partial x_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2 - u_{ij})^2 \Delta x_{ij}, \quad \text{for } (1 \le u_{ij} < 2), \quad (2.41)$$

$$\frac{\partial W^a{}_{ij}}{\partial y_i} = -\frac{C}{h^2}(3 - \frac{9}{4}u_{ij})\Delta y_{ij}, \quad \text{for } (0 \le u_{ij} < 1), \quad (2.42)$$

$$\frac{\partial W^b{}_{ij}}{\partial y_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2 - u_{ij})^2 \Delta y_{ij}, \quad \text{for } (1 \le u_{ij} < 2), \quad (2.43)$$

$$\frac{\partial W^a{}_{ij}}{\partial z_i} = -\frac{C}{h^2}(3 - \frac{9}{4}u_{ij})\Delta z_{ij}, \quad \text{for } (0 \le u_{ij} < 1), \quad (2.44)$$

$$\frac{\partial W^b{}_{ij}}{\partial z_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2 - u_{ij})^2 \Delta z_{ij}. \quad \text{for } (1 \le u_{ij} < 2). \quad (2.45)$$

Swapping i and j in Eqs. (2.40) through (2.45) shows that the derivatives with respect to the jth particle are anti-symmetric; the negative sign comes from the Δ term in each equation: that is, $\frac{\partial W_{ij}}{\partial x_i} = -\frac{\partial W_{ij}}{\partial x_j}$ and similarly for the y and z first-order derivatives. So only the three first-order derivatives for the ith particle need to be calculated, and those for the jth particle are found by negating those for the ith particle. The relationship between W_{ij} and the first-order derivatives are given here:

$$W_{ij} = W_{ji}, \quad (2.46)$$

$$\partial W_{ij} / \partial x_i = -\partial W_{ij} / \partial x_j, \quad (2.47)$$

$$\partial W_{ij} / \partial y_i = -\partial W_{ij} / \partial y_j, \quad (2.48)$$

$$\partial W_{ij} / \partial z_i = -\partial W_{ij} / \partial z_j. \quad (2.49)$$

E. The Neighbor Search Routine

Typically the most time-consuming aspect of an SPH code is the neighbor searching, and a great deal of effort has gone into finding the most efficient method. There are a number of different neighbor search routines used in the field of SPH, Fulk [27] Section 2.3.10. For the SPHINX code, two particles are defined as neighbors if they have overlapping spheres of influence, that is, the distance between them is less than the sum of their smoothing lengths.

The simplest known search scheme is the N-squared routine. Let the total number of particles be N, then each particle *i* is compared with particles *i* through N. In other words, the 0th particle will be compared to all the other particles, but the next particle will be compared to all the other particles except the 0th, and the next particle will be compared to all but the 0th and 1st particles, and so on. Each particle is included in its own neighbor list because, for instance, its own mass has to be included in the calculation of its density along with its other neighbors. The number of operations for this is proportional to N x N, hence its name.

The neighbor search most often used, and the default in the SPHINX code, is an octree search for 3D, a quadtree search for 2D, and a bitree search for 1D, (see Hernquist and Katz [34]). It takes on the order of N log (N) operations to find the neighbors of each particle, which is more efficient than the N-squared method. Depending on the dimension of the problem, each axis is divided in two. For 3D, using the octree method, space is divided into eight equal cubes; then each of those can be divided into eight and so on. If there are no particles or just one particle in a subcube, then it is not divided any further.

This procedure can be viewed as forming a tree of finer and finer branching until each particle is isolated in its own cube. Thus each particle becomes a leaf on the tree. Then by traversing the tree the neighbors can be found by looking at the hierarchy of the branches. For a given particle its neighbors will be found along one branch and not the others, eliminating a search through most of the particles.

A third method, known as a linked-list, or cells method, is generated by placing a temporary grid, with a cell spacing of about $2h$, on the space or volume of the problem. If h is constant for all particles, then the neighbors of particle i will be in either its cell or the immediately adjacent cells. Depending on the dimension of the problem, the number of cells is 3, 9, or 27 for 1D, 2D, or 3D respectively. A single pass through all the particles can assign each particle to a cell, and all the particles within a cell are linked together. Then the neighbors for particle i are determined by searching only through the linked particles of its associated cells. If the average number of neighbors is N_n, then the number of operations would be on the order of $N_n N$, and if $N >> N_n$ then its efficiency can approach that of order N. If, however, h is variable, then the choice of the cell size becomes more difficult, and this method can become less efficient than the octree method. The octree is usually the method of choice when h is variable.

Chapter III

The New Implicit SPH Code

A. Introduction to the Implicit Code

If the fluid problem being modeled does not have rapidly changing properties and is not being dominated by shock waves, but the time-steps determined by the Courant condition are still very short because of the sound speed of the fluid, then by using an implicit code, larger time-steps can generally be taken to get through the problem in a reasonable amount of time. An explicit code has to use the time-steps dictated by the Courant condition or else the solutions may become unstable. Most implicit schemes, however, can be shown to be unconditionally stable for any time-step size. They do lose accuracy, with increased time-step size, but remain stable. The main disadvantage of an implicit code is that it is computationally intensive because a huge linear system or matrix needs to be solved. There is a trade-off region, above which the time-step size versus total computing time makes it more advantageous to use an implicit code, and below which it is more advantageous to use an explicit code.

Using the implicit SPH method, the number of equations for a problem of N particles in dimension D is (D+1)2N, hence a large number of simultaneous equations must be solved. For a 3D problem of N particles, there are 8N equations and 8N dependent variables. To solve the 8N fluid equations implicitly leads to the computation of a Jacobian matrix of partial derivatives, which are the derivatives of the 8N equations with respect to the 8N dependent variables. The result is an 8N x 8N matrix in 3D, or a 6N x 6N in 2D, or a 4N x 4N matrix in 1D. This matrix is very large when N is several hundred to over a

million particles, no matter what the dimension is. These are the numbers of particles the explicit SPHINX code is capable of handling, depending on the type of computer used. Thus the implicit method is computationally intensive and is best for problems for which large time-steps are possible.

Various techniques have been developed for implicit calculations to shorten the computational time and save on computer memory. For a sparse matrix, such as the one generated by the implicit SPH code, sparse techniques have been developed in which only the non-zero elements of the matrix are stored, and matrix-vector products can still be performed within these sparse constructs. Along with the storage issues, sparse computations can be done if it is known a priori where the non-zero elements are going to be, thus saving on computer time. Another way to shorten the computational time is to use iterative techniques to obtain an approximate solution to inverting the matrix. Another method for saving memory is known as the matrix-free method in which all matrix-vector products are replaced by terms from a Taylor expansion, obviating the need to form the matrix at all; the matrix-vector product is replaced by a sum of vector operations.

For a set of linear ODEs with constant coefficients, the matrix equation to be solved is of the general form, following Press et al. [66]

$$\mathbf{Y}' = -\mathbf{A} \bullet \mathbf{Y}, \tag{3.1}$$

where **A** is a matrix, **Y** is the state vector, and the prime indicates a derivative with respect to the independent variable, time. Using n to represent the current time-step number, an example of explicit time differencing is:

$$\mathbf{Y}'_n \equiv (\mathbf{Y}_{n+1} - \mathbf{Y}_n) / \Delta t, \tag{3.2}$$

or solving for $\mathbf{Y}_{n+1}$ at the new time-step Δt

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + \Delta t\, \mathbf{Y}'_n = (\mathbf{I} - \Delta t\, \mathbf{A}) \bullet \mathbf{Y}_n, \tag{3 .3}$$

where Eq. (3 .1) has been used to replace $\mathbf{Y}'_n$. The quantity in the parentheses is a matrix, where $\mathbf{I}$ is a unit matrix, and this matrix does not need to be inverted to solve the system of equations.

On the other hand, an example of <u>implicit time differencing</u> is:

$$\mathbf{Y}'_{n+1} \cong (\mathbf{Y}_{n+1} - \mathbf{Y}_n) / \Delta t, \tag{3 .4}$$

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + \Delta t\, \mathbf{Y}'_{n+1} = (\mathbf{I} + \Delta t\, \mathbf{A})^{-1} \bullet \mathbf{Y}_n, \tag{3 .5}$$

where $\mathbf{Y}'_{n+1}$ was replaced, as before, by Eq. (3 .1) and Eq. (3 .5) is solved for $\mathbf{Y}_{n+1}$. The quantity in the parentheses is a matrix that is to be inverted. Modern iterative techniques, however, avoid actually inverting the matrix. Instead they solve the linear system:

$$(\mathbf{I} + \Delta t\, \mathbf{A}) \bullet \mathbf{Y}_{n+1} = \mathbf{Y}_n \tag{3 .6}$$

by guessing at a solution for $\mathbf{Y}_{n+1}$ and iterating on it until Eq. (3 .6) is satisfied to within some tolerance.

For a set of nonlinear ODEs, things have to be handled differently. Let $\mathbf{f(Y)}$ be an arbitrary vector function of the vector $\mathbf{Y}$, which may be nonlinear. For SPH, $\mathbf{f(Y)}$ would represent the right-hand sides of the velocity, momentum, continuity, and energy equations. Then a general nonlinear set of equations can be represented as:

$$\mathbf{Y}' = \mathbf{f(Y)}, \tag{3 .7}$$

where the prime indicates a time derivative. After implicit differencing, Eq. (3 .8), one can linearize $\mathbf{f}(\mathbf{Y}_{n+1})$ by keeping the first two terms of the Taylor expansion, Eq. (3 .9), and

then collecting the $\mathbf{Y}_{n+1}$ terms, Eq. (3 .10):

$$\mathbf{Y}_{n+1} \cong \mathbf{Y}_n + \Delta t\, \mathbf{f}(\mathbf{Y}_{n+1}), \tag{3 .8}$$

$$\cong \mathbf{Y}_n + \Delta t\, [\mathbf{f}(\mathbf{Y}_n) + \partial \mathbf{f}/\partial \mathbf{Y}|_{\mathbf{Y}_n} \bullet (\mathbf{Y}_{n+1} - \mathbf{Y}_n)], \tag{3 .9}$$

$$= \mathbf{Y}_n + \Delta t\, [\mathbf{I} - \Delta t\, \partial \mathbf{f}/\partial \mathbf{Y}]^{-1} \bullet \mathbf{f}(\mathbf{Y}_n), \tag{3 .10}$$

$$\mathbf{Y}_{n+1} \equiv \mathbf{Y}_n + \mathbf{J}^{-1} \bullet \mathbf{Y}'_n, \equiv \mathbf{Y}_n + \mathbf{dY} \tag{3 .11}$$

where $\mathbf{J} \equiv [\mathbf{I}/\Delta t - \partial \mathbf{f}/\partial \mathbf{Y}]$ is a Jacobian matrix containing partial derivatives of $\mathbf{f}(\mathbf{Y})$ with respect to the dependent variables. This is the Jacobian of the residuals, which will be discussed later in sections F and K of this chapter. The Jacobian has diagonal elements consisting of a unit matrix divided by the time-step, which makes the diagonal, in general, non-zero. As the time-step is decreased, the matrix becomes more diagonally dominant, which makes the matrix easier to invert.

The last term of Eq. (3 .11) is an inverse-matrix vector multiply and can be regarded as $\mathbf{dY}$, which is added to $\mathbf{Y}_n$ to update to $\mathbf{Y}_{n+1}$. So $\mathbf{dY} \equiv \mathbf{J}^{-1} \bullet \mathbf{Y}'_n$ is the usual form for the inverse matrix problem. Various methods have been developed for inverting matrices. To get started on the implicit SPH code, the first attempt was just to try the LU decomposition method, and the Jacobian was derived analytically.

B. The Analytic Jacobian

B.1. Derivatives for the Implicit SPH Code

The implicit version of the SPH code requires the computation of the Jacobian matrix. The 3D Jacobian is a matrix of the derivatives of all 8N equations, Eqs. (2 .31)

through (2 .36), with respect to all 8N dependent variables. The derivation of the equations to fill the 8N x 8N Jacobian matrix is the subject of this section.

In the following equations, the time derivatives of Eqs. (2 .31) through (2 .36) (which are total derivatives) are denoted by a dot above the variable on the left-hand side of the equation. The subscripts i and j are used to denote a pair of particles. The index i represents a particular particle and in that sense is considered fixed. The index j will range over all the neighbors of particle i and is represented as a summation in the equations. Another index, k, needs to be introduced at this point to indicate with respect to which of the 8N dependent variables the derivative is being taken. The new index runs from 0 to N-1. Like index i, the index k is considered fixed in the sense that each Jacobian element is the derivative with respect to only the kth dependent variable. For an example matrix of a 1D 3-particle problem see Fig. III. 1, at the end of this section.

The following derivatives have been taken with respect to the kth dependent variable, and it is found that, except for $k = i$ or $k = j$, the derivatives are zero; that is because the kth variable does not appear in the equations except when $k = i$ or $k = j$. The derivatives are also zero when a pair of particles are not neighbors. Taking the derivatives of Eq. (2 .31) is simple since only the components of v_i appear in the equations. Hence all derivatives are zero, except for those with respect to the appropriate component of the velocity, and those derivatives always equal 1. Thus,

$$\frac{\partial \dot{x}_i}{\partial v^x_{k=i}} = 1, \qquad \frac{\partial \dot{y}_i}{\partial v^y_{k=i}} = 1, \qquad \frac{\partial \dot{z}_i}{\partial v^z_{k=i}} = 1, \tag{3 .12}$$

and all others are zero.

For the derivatives of Eqs. (2 .31) through (2 .36) there are summations, over the

neighbors of particle i, and it has been found that for $k = i$ the summation survives in the derivative. These terms show up in 8 x 8 blocks along the diagonal of the Jacobian matrix (see Fig. III. 1). But for $k = j$ only one term survives from each summation. These terms show up in 8 x 8 blocks off the diagonal and represent the terms due to interaction with the neighboring particles. Since most of the particles will have only a small number of neighbors relative to the total number of particles N, most of the off-diagonal 8 x 8 blocks will be filled with zeros. Therefore, the Jacobian will be a sparse matrix. Timmes estimates [86] that a typical Jacobian will have only 1.12% non-zero elements.

a. The derivatives of $\dot{v}_i^x$, Eq. (2 .32), with respect to the kth dependent-variables follow:

$$\frac{\partial \dot{v}_i^x}{\partial x_{k=i}} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} \right) \frac{\partial^2 W_{ij}}{\partial x_i \partial x_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial x_{k=j}} = -m_k \left(\frac{P_i + P_k}{\rho_i \rho_k} \right) \frac{\partial^2 W_{ik}}{\partial x_k \partial x_i}, \quad (3.13)$$

$$\frac{\partial \dot{v}_i^x}{\partial y_{k=i}} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} \right) \frac{\partial^2 W_{ij}}{\partial y_i \partial x_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial y_{k=j}} = -m_k \left(\frac{P_i + P_k}{\rho_i \rho_k} \right) \frac{\partial^2 W_{ik}}{\partial y_k \partial x_i}, \quad (3.14)$$

$$\frac{\partial \dot{v}_i^x}{\partial z_{k=i}} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} \right) \frac{\partial^2 W_{ij}}{\partial z_i \partial x_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial z_{k=j}} = -m_k \left(\frac{P_i + P_k}{\rho_i \rho_k} \right) \frac{\partial^2 W_{ik}}{\partial z_k \partial x_i}, \quad (3.15)$$

$$\frac{\partial \dot{v}_i^x}{\partial v_k^x} = 0, \qquad \frac{\partial \dot{v}_i^x}{\partial v_k^y} = 0, \qquad \frac{\partial \dot{v}_i^x}{\partial v_k^z} = 0. \quad (3.16)$$

Of Eqs. (3 .13) to (3 .15), the three equations with the summations contribute to the blocks along the diagonal, and the other three contribute to the off-diagonal blocks. Note too the latter three equations have k subscripts, since k is considered a fixed number and j is not.

Because the pressure is a function of the density and energy, the next two sets of derivatives use the perfect gas law, Eq. (2 .37). When this EOS is used, ρ_i cancels in one

term within the parenthesis of Eq. (2 .32) and ρ_j cancels in the other term, so the derivative with respect to $\rho_{k=i}$ is different than when taken with respect to $\rho_{k=j}$. Again, note that for the case of $k=j$, only the one term, $k=j$, survives from the summation:

$$\frac{\partial \dot{v}_i^x}{\partial \rho_{k=i}} = (\gamma-1)\sum_j m_j \frac{e_j}{\rho_i^2}\frac{\partial W_{ij}}{\partial x_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial \rho_{k=j}} = (\gamma-1)\, m_k \frac{e_i}{\rho_k^2}\frac{\partial W_{ik}}{\partial x_i}, \tag{3 .17}$$

$$\frac{\partial \dot{v}_i^x}{\partial e_{k=i}} = -(\gamma-1)\sum_j \frac{m_j}{\rho_j}\frac{\partial W_{ij}}{\partial x_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial e_{k=j}} = -(\gamma-1)\frac{m_k}{\rho_i}\frac{\partial W_{ik}}{\partial x_i}. \tag{3 .18}$$

b. The 8 derivatives of $\dot{v}_i^y$, Eq. (2 .33), are similar:

$$\frac{\partial \dot{v}_i^y}{\partial x_k} = -\sum_j m_j\left(\frac{P_i+P_j}{\rho_i\rho_j}\right)\frac{\partial^2 W_{ij}}{\partial x_k \partial y_i}, \tag{3 .19}$$

$$\frac{\partial \dot{v}_i^y}{\partial y_k} = -\sum_j m_j\left(\frac{P_i+P_j}{\rho_i\rho_j}\right)\frac{\partial^2 W_{ij}}{\partial y_k \partial y_i}, \tag{3 .20}$$

$$\frac{\partial \dot{v}_i^y}{\partial z_k} = -\sum_j m_j\left(\frac{P_i+P_j}{\rho_i\rho_j}\right)\frac{\partial^2 W_{ij}}{\partial z_k \partial y_i}, \tag{3 .21}$$

$$\frac{\partial \dot{v}_i^y}{\partial v_k^x} = 0, \qquad \frac{\partial \dot{v}_i^y}{\partial v_k^y} = 0, \qquad \frac{\partial \dot{v}_i^y}{\partial v_k^z} = 0, \tag{3 .22}$$

$$\frac{\partial \dot{v}_i^y}{\partial \rho_{k=i}} = (\gamma-1)\sum_j m_j \frac{e_j}{\rho_i^2}\frac{\partial W_{ij}}{\partial y_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial \rho_{k=j}} = (\gamma-1)\, m_k \frac{e_i}{\rho_k^2}\frac{\partial W_{ik}}{\partial y_i}, \tag{3 .23}$$

$$\frac{\partial \dot{v}_i^y}{\partial e_{k=i}} = -(\gamma-1)\sum_j \frac{m_j}{\rho_j}\frac{\partial W_{ij}}{\partial y_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial e_{k=j}} = -(\gamma-1)\frac{m_k}{\rho_i}\frac{\partial W_{ik}}{\partial y_i}. \tag{3 .24}$$

Evaluation of Eqs. (3 .19) to (3 .21) at $k=i$ and $k=j$ is simple and is analogous to that shown in Eqs. (3 .13) to (3 .15), but for Eqs. (3 .23) and (3 .24) the results are different again.

c. The 8 derivatives of $\dot{v}_i^z$, Eq. (2 .34), are also similar:

$$\frac{\partial \dot{v}_i^z}{\partial x_k} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j}\right) \frac{\partial^2 W_{ij}}{\partial x_k \partial z_i}, \tag{3 .25}$$

$$\frac{\partial \dot{v}_i^z}{\partial y_k} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j}\right) \frac{\partial^2 W_{ij}}{\partial y_k \partial z_i}, \tag{3 .26}$$

$$\frac{\partial \dot{v}_i^z}{\partial z_k} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j}\right) \frac{\partial^2 W_{ij}}{\partial z_k \partial z_i}, \tag{3 .27}$$

$$\frac{\partial \dot{v}_i^z}{\partial v_k^x} = 0, \qquad \frac{\partial \dot{v}_i^z}{\partial v_k^y} = 0, \qquad \frac{\partial \dot{v}_i^z}{\partial v_k^z} = 0, \tag{3 .28}$$

$$\frac{\partial \dot{v}_i^z}{\partial \rho_{k=i}} = (\gamma - 1) \sum_j m_j \frac{e_j}{\rho_i^2} \frac{\partial W_{ij}}{\partial z_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial \rho_{k=j}} = (\gamma - 1)\, m_k \frac{e_i}{\rho_k^2} \frac{\partial W_{ik}}{\partial z_i}, \tag{3 .29}$$

$$\frac{\partial \dot{v}_i^z}{\partial e_{k=i}} = -(\gamma - 1) \sum_j \frac{m_j}{\rho_j} \frac{\partial W_{ij}}{\partial z_i}, \qquad \frac{\partial \dot{v}_i^x}{\partial e_{k=j}} = -(\gamma - 1) \frac{m_k}{\rho_i} \frac{\partial W_{ik}}{\partial z_i}. \tag{3 .30}$$

d. The 8 derivatives of $\dot{\rho}_i$, Eq. (2 .35), are the following.

$$\frac{\partial \dot{\rho}_i}{\partial x_k} = \sum_j m_j \frac{\rho_i}{\rho_j} \left\{ (v_i^x - v_j^x) \frac{\partial^2 W_{ij}}{\partial x_k \partial x_i} + (v_i^y - v_j^y) \frac{\partial^2 W_{ij}}{\partial x_k \partial y_i} + (v_i^z - v_j^z) \frac{\partial^2 W_{ij}}{\partial x_k \partial z_i} \right\}, \tag{3 .31}$$

$$\frac{\partial \dot{\rho}_i}{\partial y_k} = \sum_j m_j \frac{\rho_i}{\rho_j} \left\{ (v_i^x - v_j^x) \frac{\partial^2 W_{ij}}{\partial y_k \partial x_i} + (v_i^y - v_j^y) \frac{\partial^2 W_{ij}}{\partial y_k \partial y_i} + (v_i^z - v_j^z) \frac{\partial^2 W_{ij}}{\partial y_k \partial z_i} \right\}, \tag{3 .32}$$

$$\frac{\partial \dot{\rho}_i}{\partial z_k} = \sum_j m_j \frac{\rho_i}{\rho_j} \left\{ (v_i^x - v_j^x) \frac{\partial^2 W_{ij}}{\partial z_k \partial x_i} + (v_i^y - v_j^y) \frac{\partial^2 W_{ij}}{\partial z_k \partial y_i} + (v_i^z - v_j^z) \frac{\partial^2 W_{ij}}{\partial z_k \partial z_i} \right\}. \tag{3 .33}$$

The next three derivatives differ in sign depending on whether $k = i$ or $k = j$, because the v_i^x term is positive and the v_j^x term is negative.

$$\frac{\partial \dot{\rho}_i}{\partial v^x_{k=i}} = \sum_j m_j \frac{\rho_i}{\rho_j}\left\{\frac{\partial W_{ij}}{\partial x_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^x_{k=j}} = -m_k \frac{\rho_i}{\rho_k}\left\{\frac{\partial W_{ik}}{\partial x_i}\right\}, \tag{3 .34}$$

$$\frac{\partial \dot{\rho}_i}{\partial v^y_{k=i}} = \sum_j m_j \frac{\rho_i}{\rho_j}\left\{\frac{\partial W_{ij}}{\partial y_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^y_{k=j}} = -m_k \frac{\rho_i}{\rho_k}\left\{\frac{\partial W_{ik}}{\partial y_i}\right\}, \tag{3 .35}$$

$$\frac{\partial \dot{\rho}_i}{\partial v^z_{k=i}} = \sum_j m_j \frac{\rho_i}{\rho_j}\left\{\frac{\partial W_{ij}}{\partial z_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^z_{k=j}} = -m_k \frac{\rho_i}{\rho_k}\left\{\frac{\partial W_{ik}}{\partial z_i}\right\}. \tag{3 .36}$$

The derivative with respect to density also differs depending on whether $k = i$ or $k = j$, because ρ_i is in the numerator and ρ_j is in the denominator of Eq. (2 .35).

$$\frac{\partial \dot{\rho}_i}{\partial \rho_{k=i}} = \sum_j \frac{m_j}{\rho_j}\left\{(v_i^x - v_j^x)\frac{\partial W_{ij}}{\partial x_i} + (v_i^y - v_j^y)\frac{\partial W_{ij}}{\partial y_i} + (v_i^z - v_j^z)\frac{\partial W_{ij}}{\partial z_i}\right\}, \tag{3 .37}$$

$$\frac{\partial \dot{\rho}_i}{\partial \rho_{k=j}} = -\rho_i \frac{m_k}{\rho_k^2}\left\{(v_i^x - v_k^x)\frac{\partial W_{ik}}{\partial x_i} + (v_i^y - v_k^y)\frac{\partial W_{ik}}{\partial y_i} + (v_i^z - v_k^z)\frac{\partial W_{ik}}{\partial z_i}\right\}. \tag{3 .38}$$

The derivative with respect to energy is zero for both $k = i$ and $k = j$, because e_k does not appear in Eq. (2 .35).

$$\frac{\partial \dot{\rho}_i}{\partial e_k} = 0. \tag{3 .39}$$

e. The remaining set of Jacobian derivatives consists of the 8 derivatives of $\dot{e}_i$, Eq. (2 .36). For brevity, let $\Delta v^x_{ij} \equiv (v_i^x - v_j^x)$, and similarly for the y and z components.

$$\frac{\partial \dot{e}_i}{\partial x_k} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\Delta v^x_{ij}\frac{\partial^2 W_{ij}}{\partial x_k \partial x_i} + \Delta v^y_{ij}\frac{\partial^2 W_{ij}}{\partial x_k \partial y_i} + \Delta v^z_{ij}\frac{\partial^2 W_{ij}}{\partial x_k \partial z_i}\right\} = \frac{P_i}{\rho_i^2}\frac{\partial \dot{\rho}_i}{\partial x_k}, \tag{3 .40}$$

$$\frac{\partial \dot{e}_i}{\partial y_k} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\Delta v^x_{ij}\frac{\partial^2 W_{ij}}{\partial y_k \partial x_i} + \Delta v^y_{ij}\frac{\partial^2 W_{ij}}{\partial y_k \partial y_i} + \Delta v^z_{ij}\frac{\partial^2 W_{ij}}{\partial y_k \partial z_i}\right\}, \tag{3 .41}$$

$$\frac{\partial \dot{e}_i}{\partial z_k} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\Delta v^x_{ij}\frac{\partial^2 W_{ij}}{\partial z_k \partial x_i} + \Delta v^y_{ij}\frac{\partial^2 W_{ij}}{\partial z_k \partial y_i} + \Delta v^z_{ij}\frac{\partial^2 W_{ij}}{\partial z_k \partial z_i}\right\}. \tag{3 .42}$$

The next three derivatives differ in sign depending on whether $k = i$ or $k = j$, because the v_i^x term is positive and the v_j^x term is negative.

$$\frac{\partial \dot{e}_i}{\partial v^x_{k=i}} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\frac{\partial W_{ij}}{\partial x_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^x_{k=j}} = -m_k \frac{P_i}{\rho_i \rho_k}\left\{\frac{\partial W_{ik}}{\partial x_i}\right\}, \tag{3 .43}$$

$$\frac{\partial \dot{e}_i}{\partial v^y_{k=i}} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\frac{\partial W_{ij}}{\partial y_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^y_{k=j}} = -m_k \frac{P_i}{\rho_i \rho_k}\left\{\frac{\partial W_{ik}}{\partial y_i}\right\}, \tag{3 .44}$$

$$\frac{\partial \dot{e}_i}{\partial v^z_{k=i}} = \sum_j m_j \frac{P_i}{\rho_i \rho_j}\left\{\frac{\partial W_{ij}}{\partial z_i}\right\}, \qquad \frac{\partial \dot{\rho}_i}{\partial v^z_{k=j}} = -m_k \frac{P_i}{\rho_i \rho_k}\left\{\frac{\partial W_{ik}}{\partial z_i}\right\}. \tag{3 .45}$$

Because the pressure is a function of the density and the energy, the next two derivatives use the perfect gas law, Eq. (2 .37). When this is done, ρ_i cancels and so the derivative with respect to $\rho_{k=i}$ is zero but with respect to $\rho_{k=j}$ is non-zero. It is the opposite for the derivative with respect to $\dot{e}_k$.

$$\frac{\partial \dot{e}_i}{\partial \rho_{k=i}} = 0, \tag{3 .46}$$

$$\frac{\partial \dot{e}_i}{\partial \rho_{k=j}} = -m_k \frac{(\gamma-1)e_i}{\rho_k}\left\{\Delta v_{ij}^{\,x}\frac{\partial W_{ij}}{\partial x_i} + \Delta v_{ij}^{\,y}\frac{\partial W_{ij}}{\partial y_i} + \Delta v_{ij}^{\,z}\frac{\partial W_{ij}}{\partial z_i}\right\}, \tag{3 .47}$$

$$\frac{\partial \dot{e}_i}{\partial e_{k=i}} = (\gamma-1)\sum_j \frac{m_j}{\rho_j}\left\{\Delta v_{ij}^{\,x}\frac{\partial W_{ij}}{\partial x_i} + \Delta v_{ij}^{\,y}\frac{\partial W_{ij}}{\partial y_i} + \Delta v_{ij}^{\,z}\frac{\partial W_{ij}}{\partial z_i}\right\}, \tag{3 .48}$$

$$\frac{\partial \dot{e}_i}{\partial e_{k=j}} = 0. \tag{3 .49}$$

B.2. The Second-Order Derivatives of the B-Spline W_{ij}

The implicit version of the SPH code requires 1st and 2nd-order derivatives of W_{ij}, Eq. (2 .28), with respect to the three spatial dependent variables. The first-order deriva-

tives were described in Chapter II, Section D.

A study of the second-order derivatives shows that there are six basic forms needed, and the others can be found from them with just a sign change. The six basic forms are:

$$\frac{\partial^2 W_{ij}}{\partial x_i \partial x_i}, \quad \frac{\partial^2 W_{ij}}{\partial x_i \partial y_i}, \quad \frac{\partial^2 W_{ij}}{\partial x_i \partial z_i}, \quad \frac{\partial^2 W_{ij}}{\partial y_i \partial y_i}, \quad \frac{\partial^2 W_{ij}}{\partial y_i \partial z_i}, \quad \frac{\partial^2 W_{ij}}{\partial z_i \partial z_i}. \tag{3.50}$$

The relationship between these six and the others is shown below (the first-order derivatives are also included from Chapter II Section D for completeness).

$$W_{ij} = W_{ji}, \tag{3.51}$$

$$\partial W_{ij} / \partial x_i = -\partial W_{ij} / \partial x_j, \tag{3.52}$$

$$\partial W_{ij} / \partial y_i = -\partial W_{ij} / \partial y_j, \tag{3.53}$$

$$\partial W_{ij} / \partial z_i = -\partial W_{ij} / \partial z_j, \tag{3.54}$$

$$\partial^2 W_{ij} / \partial x_i \partial x_i = \partial^2 W_{ij} / \partial x_j \partial x_j = -\partial^2 W_{ij} / \partial x_i \partial x_j, \tag{3.55}$$

$$\partial^2 W_{ij} / \partial y_i \partial y_i = \partial^2 W_{ij} / \partial y_j \partial y_j = -\partial^2 W_{ij} / \partial y_i \partial y_j, \tag{3.56}$$

$$\partial^2 W_{ij} / \partial z_i \partial z_i = \partial^2 W_{ij} / \partial z_j \partial z_j = -\partial^2 W_{ij} / \partial z_i \partial z_j, \tag{3.57}$$

$$\partial^2 W_{ij} / \partial x_i \partial y_i = \partial^2 W_{ij} / \partial x_j \partial y_j = -\partial^2 W_{ij} / \partial x_i \partial y_j = -\partial^2 W_{ij} / \partial x_j \partial y_i, \tag{3.58}$$

$$\partial^2 W_{ij} / \partial x_i \partial z_i = \partial^2 W_{ij} / \partial x_j \partial z_j = -\partial^2 W_{ij} / \partial x_i \partial z_j = -\partial^2 W_{ij} / \partial x_j \partial z_i, \tag{3.59}$$

$$\partial^2 W_{ij} / \partial y_i \partial z_i = \partial^2 W_{ij} / \partial y_j \partial z_j = -\partial^2 W_{ij} / \partial y_i \partial z_j = -\partial^2 W_{ij} / \partial y_j \partial z_i. \tag{3.60}$$

Because the cubic B-splines of W_{ij} form a continuous function out to the second derivative, the order in which the partials are taken does not make a difference.

The six second-order derivatives for both $W^a{}_{ij}$,and $W^b{}_{ij}$ follow. The derivative of $W^a{}_{ij}$, is a special case when $i = j$, because the derivative of $\Delta x = x_i - x_j$ with respect to x_i is zero, not one. A useful notation for the derivative of Δx is $d\Delta x/dx_i = (1 - \delta_{ij})$.

$$\frac{\partial^2 W^a{}_{ij}}{\partial x_i \partial x_i} = \frac{9}{4}\frac{C}{h^3}\left\{\frac{\Delta x^2}{r_{ij}} + \left(r_{ij} - \frac{4}{3}h\right)(1-\delta_{ij})\right\}, \quad \text{for } (0 \le u_{ij} < 1), \ (3.61)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial x_i \partial x_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2-u_{ij})\left\{-2\frac{\Delta x^2}{hr_{ij}} + (2-u_{ij})\left(1 - \frac{\Delta x^2}{r_{ij}^2}\right)\right\}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.62)$$

$$\frac{\partial^2 W^a{}_{ij}}{\partial y_i \partial y_i} = \frac{9}{4}\frac{C}{h^3}\left\{\frac{\Delta y^2}{r_{ij}} + \left(r_{ij} - \frac{4}{3}h\right)(1-\delta_{ij})\right\}, \quad \text{for } (0 \le u_{ij} < 1), \ (3.63)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial y_i \partial y_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2-u_{ij})\left\{-2\frac{\Delta y^2}{hr_{ij}} + (2-u_{ij})\left(1 - \frac{\Delta y^2}{r_{ij}^2}\right)\right\}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.64)$$

$$\frac{\partial^2 W^a{}_{ij}}{\partial z_i \partial z_i} = \frac{9}{4}\frac{C}{h^3}\left\{\frac{\Delta z^2}{r_{ij}} + \left(r_{ij} - \frac{4}{3}h\right)(1-\delta_{ij})\right\}, \quad \text{for } (0 \le u_{ij} < 1), \ (3.65)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial z_i \partial z_i} = -\frac{3}{4}\frac{C}{hr_{ij}}(2-u_{ij})\left\{-2\frac{\Delta z^2}{hr_{ij}} + (2-u_{ij})\left(1 - \frac{\Delta z^2}{r_{ij}^2}\right)\right\}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.66)$$

$$\frac{\partial^2 W^a{}_{ij}}{\partial x_i \partial y_i} = \frac{9}{4}\frac{C\Delta x \Delta y}{h^3 r_{ij}}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.67)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial x_i \partial y_i} = \frac{3}{4}\frac{C\Delta x \Delta y}{h^2 r_{ij}^2}(2-u_{ij})\left\{1 + (2-u_{ij})\frac{h}{r_{ij}}\right\}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.68)$$

$$\frac{\partial^2 W^a{}_{ij}}{\partial x_i \partial z_i} = \frac{9}{4}\frac{C\Delta x \Delta z}{h^3 r_{ij}}, \quad \text{for } (0 \le u_{ij} < 1), \ (3.69)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial x_i \partial z_i} = \frac{3}{4}\frac{C\Delta x \Delta z}{h^2 r_{ij}^2}(2-u_{ij})\left\{1 + (2-u_{ij})\frac{h}{r_{ij}}\right\}, \quad \text{for } (1 \le u_{ij} < 2), \ (3.70)$$

$$\frac{\partial^2 W^a{}_{ij}}{\partial y_i \partial z_i} = \frac{9}{4}\frac{C\Delta y \Delta z}{h^{3(2)} r_{ij}}, \quad \text{for } (0 \le u_{ij} < 1), \ (3.71)$$

$$\frac{\partial^2 W^b{}_{ij}}{\partial y_i \partial z_i} = \frac{3}{4}\frac{C\Delta y \Delta z}{h^2 r_{ij}^2}(2-u_{ij})\left\{1 + (2-u_{ij})\frac{h}{r_{ij}}\right\}. \quad \text{for } (1 \le u_{ij} < 2). \ (3.72)$$

Initial Jacobian Matrix for the 1D 3-Particle Problem

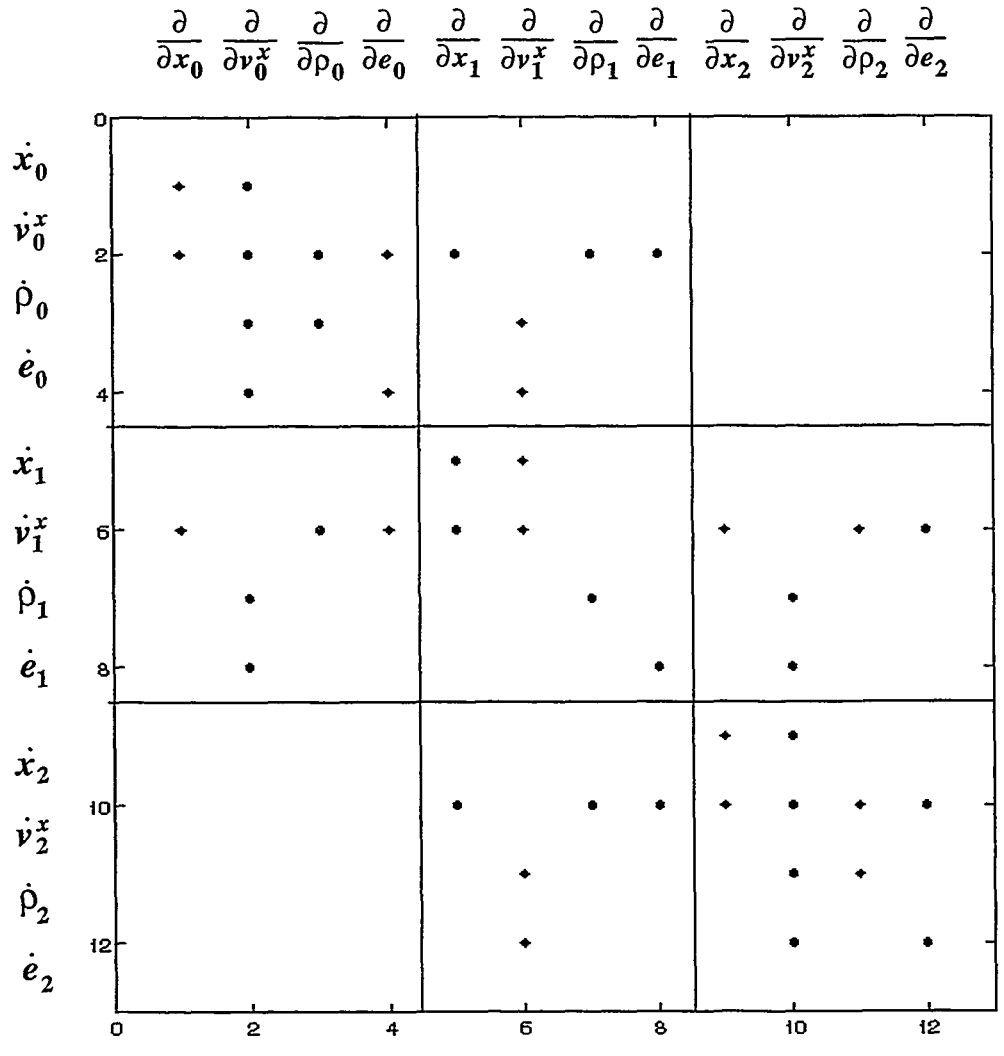


Fig. III. 1. This is a simple 12 x 12 Jacobian spot matrix for a 1D 3-particle problem, showing a dot wherever there is a non-zero element. Down the left side are indicated the time-derivative equations (note the dot over each variable) of which the partial derivatives are taken. The partial derivatives for each dependent variable are indicated across the top of the matrix. The two horizontal and two vertical bars are placed in the matrix to show how each particle contributes a 4 x 4 block of elements along the diagonal of the matrix and two 4 x 4 off-diagonal blocks for each neighbor with which it interacts. Particles 0 and 2 are not interacting, so their off-diagonal blocks are filled with zeros. The off-diagonal blocks are symmetric about the diagonal, but the matrix itself is non-symmetric.

C. Lower and Upper (LU) Decomposition

The implicit SPH method leads to (D+1)2 equations per particle, and thus for N particles, there are (D+1)2N simultaneous equations to be solved. In 3D there are 8N equations and 8N dependent variables. Methods that actually manipulate the matrix are known as direct methods. The Lower and Upper (LU) decomposition method with back-substitution is a direct method. It decomposes a matrix **A** into an upper and a lower triangular matrix. The problem then becomes $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$, and by setting $\mathbf{y} = \mathbf{Ux}$, the problem is split into two parts. First solve $\mathbf{Ly} = \mathbf{b}$ to find the vector **y** by forward substitution. Then solve $\mathbf{Ux} = \mathbf{y}$ for the vector **x** using back-substitution.

The LU decomposition method has the advantage over Gaussian elimination in that the vector **b** is not altered in the process. For Gaussian elimination each row manipulation alters the vector **b** by scaling its elements and adding them or swapping them around. Once **A** has been decomposed into the matrices **L** and **U**, however, a sequence of different vectors **b** could be run for a variety of conditions. Both methods perform about the same number of operations, and so they take about the same amount of time to run, but **LU** can be used over again with different **b**s. Both of these methods, however, require fewer operations than the Gauss-Jordan elimination technique (see Press et al. [66], Sections 2.1 to 2.3).

The LU decomposition coding used in the implicit code was modified from that developed by Press et al. [66] which was used in conjunction with a fourth-order Rosenbrock method, found in Section 16.6 of the same reference. Rosenbrock methods are a generalized implicit Runge-Kutta technique and are also known as Kaps-Rentrop meth-

ods. Rosenbrock developed the theory [70] and Kaps and Rentrop [40] were the first to implement the technique as a practical code. The Rosenbrock method as implemented reuses the LU factorization by making four different estimates of the solution, with each subsequent estimate modified by the previous ones, and then the four estimates are averaged together appropriately to obtain fourth-order accuracy. Following an example in Press et al. [66], the LU decomposition method, with back-substitution coupled with the Rosenbrock method, has been implemented in the implicit code and is working, but its run time and memory usage are not very competitive with the explicit code.

D. The Numerical Jacobian

Each term in the Jacobian can be approximated numerically by using the definition of a derivative. To use a numerical Jacobian instead of the analytic technique, described in Section B of this chapter, a number of significant advantages are realized. By approximating the derivatives using existing software packages in the SPHINX code, all the existing physics packages become available to the new implicit code automatically, as well as any new ones to be added in the future. This advantage also automatically includes any new kernel routines or neighbor-search routines. There is a new moving least-squares (MLS) package being added by Dilts [22], [23] for calculating the interpolants more exactly than the standard SPH approach. This package is also automatically available to the new implicit time-stepping code. In addition, the coding for the numerical Jacobian is much simpler to implement and hence easier to debug than the analytic Jacobian because it is making use of existing code that has been independently and previously tested.

The SPH equations (2 .31) to (2 .36) are of the general form given by $d\mathbf{Y}/dt = \mathbf{f}(\mathbf{Y})$, where $\mathbf{f}(\mathbf{Y})$ represents the right-hand sides and is a vector function of the state vector $\mathbf{Y}$. The state vector contains all the dependent variables (position, velocity, density, and internal energy) for each of the particles. The right-hand side $\mathbf{f}(\mathbf{Y})$ is evaluated, at the "current" time, to obtain the rate of change for each of the dependent variables (velocity, acceleration, and the time derivatives of density $d\rho/dt$ and energy de/dt). The Jacobian matrix involves the derivative of $\mathbf{f}(\mathbf{Y})$ with respect to each of the elements Y_k of the state vector $\mathbf{Y}$. The numerical approximation for the Jacobian derivatives is given by:

$$\frac{\partial}{\partial Y_k}\mathbf{f}(\mathbf{Y}) \cong \frac{\mathbf{f}(\mathbf{Y}) - \mathbf{f}(\mathbf{Y} + \varepsilon \mathbf{Y}_k)}{\varepsilon Y_k}, \tag{3 .73}$$

where ε is a small perturbation weighted by the kth element of $\mathbf{Y}$. The bold notation $\mathbf{Y}_k$ represents a vector of zeros except for the one element Y_k in the kth position. In the definition of a derivative, ε in the limit should go to zero; on the computer, however, it is a small number, chosen mainly by consideration of the precision being used on the computer. For instance, in double precision, which carries digits out to fifteen places, $\varepsilon = 10^{-7}$ works well. Weighted by Y_k, ε perturbs the sixth digit of each value, one at a time, in the state vector $\mathbf{Y}$, irrespective of the magnitude of the value. That is, some of the values in the vector $\mathbf{Y}$, such as the energy, are going to be very large, and others, such as position, can be near zero. So weighting ε by Y_k perturbs each value by the same percentage. For single precision computations with eight digits of accuracy, $\varepsilon = 10^{-4}$ would probably be a good choice, since that would perturb the fourth to the last digit of each value in the vector $\mathbf{Y}$.

The existing explicit SPHINX code has the function *rhs*($\mathbf{Y}$), which calculates the

right-hand sides of the SPH equations. To form the numerical derivative, *rhs*(**Y**) is first run using the unperturbed values of **Y**, and all the resulting time derivatives for each particle are stored in a vector function $\mathbf{f}_0$. Then one element in the state vector **Y** is perturbed by $\varepsilon\mathbf{Y}_k$, and *rhs*($\mathbf{Y}+\varepsilon\mathbf{Y}_k$) is run again. Differencing the values of the vector $\mathbf{f}_0$ with those of the perturbed $\mathbf{f}(\mathbf{Y}+\varepsilon\mathbf{Y}_k)$, and dividing by εY_k, gives one column of the Jacobian matrix. Then the perturbed element of **Y** is set back to its original value, the next element of **Y** is perturbed, and the differencing is done all over again. Each repetition of this process calculates another column of the Jacobian.

In this fashion the numerical Jacobian matrix of the implicit code is built up during each time-step, and this approach now replaces the analytically derived Jacobian equations of Section B of this chapter. The next step is to find the solution to the inverse problem.

E. Iterative Solvers

Since the LU decomposition method is very time consuming, it has been replaced by iterative solvers. Iterative solvers are algorithms that solve a linear system $\mathbf{Ax} = \mathbf{b}$ by starting with a guess to the solution and then iterating on it until a desired accuracy has been reached without actually inverting the matrix. The iterative methods can significantly shorten the computational time over directly inverting the matrix if they converge quickly. Convergence can be accelerated by a judicious choice of a preconditioner matrix. Iterative methods also have another advantage over direct methods in that a direct method cannot be stopped part way through and have any useful results. Direct methods have to be run to completion each time, where iterative solvers can usually be stopped

after a few iterations and the result is an approximate solution, which can be useful, depending on the accuracy desired.

If $\mathbf{x}$ and $\mathbf{b}$ are vectors and $\mathbf{A}$ is a non-singular matrix to be inverted, the general problem is of the form $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, where $\mathbf{A}$ and $\mathbf{b}$ are given and $\mathbf{x}$ is the unknown. Instead of inverting $\mathbf{A}$, the iterative methods solve $\mathbf{Ax} - \mathbf{b} = \mathbf{0}$ approximately, by guessing at a solution, $\mathbf{x}_0$, and then iterating on $\mathbf{x}$ until a vector of residual errors $\mathbf{R}$ is near zero, where

$$\mathbf{R} \equiv \mathbf{Ax} - \mathbf{b} \cong \mathbf{0}. \tag{3.74}$$

In other words, the intercepts, or zero crossings, of each of the equations in the linear system is being sought.

E.1. Stationary Methods

The earliest iterative solvers, for solving $\mathbf{Ax} = \mathbf{b}$, are referred to as stationary methods [7], [41]. These are iterative methods that can be written in the form $\mathbf{x}_{k+1} = \mathbf{M}\mathbf{x}_k + \mathbf{c}$, where $\mathbf{M}$ and $\mathbf{c}$ are modifications to $\mathbf{A}$ and $\mathbf{b}$, and do not depend on the previous iteration count k. The most popular methods are the Jacobi, the Gauss-Seidel, and the Successive Overrelaxation methods. These methods are based on splitting the matrix $\mathbf{A}$ into parts:

$$\mathbf{A} \equiv \mathbf{D} + \mathbf{E} + \mathbf{F}, \tag{3.75}$$

where, using a modified notation of Saad [71], $\mathbf{D}$ is the diagonal of $\mathbf{A}$, and $\mathbf{E}$ and $\mathbf{F}$ are the two triangular parts of $\mathbf{A}$ below and above the diagonal. Equation (3.75) is not an LU decomposition but a simple splitting of the matrix. These methods are usually not as efficient as the Krylov methods but can serve as preconditioners in the Krylov methods.

The Jacobi method makes use of the fact that it is trivial to invert the diagonal and

uses an iterative equation of the form:

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1}\{\mathbf{b} - (\mathbf{E}+\mathbf{F})\mathbf{x}_k\}, \tag{3 .76}$$

So, starting with an initial guess of $\mathbf{x}_o$, then $\mathbf{x}_1$ can be obtained using (3 .76). Then, using $\mathbf{x}_1$ as the next guess, $\mathbf{x}_2$ is obtained, and so on until the desired accuracy is reached. The matrix $\mathbf{D}^{-1}(\mathbf{E}+\mathbf{F})$ is known as the iteration matrix and remains unchanged with each iteration, hence the term *stationary.*

The Gauss-Seidel method is based on the fact that the triangular matrix (**D**+**E**) is straightforward to invert; it is simply a back-substitution process. This method uses an iteration equation of the form:

$$\mathbf{x}_{k+1} = (\mathbf{D}+\mathbf{E})^{-1}\{\mathbf{b} - \mathbf{F}\mathbf{x}_k\}. \tag{3 .77}$$

The Successive Overrelaxation (SOR) method splits the matrix **A** differently. If Eq, (3 .75) is multiplied by an extrapolation factor ω, and then the diagonal **D** is added and subtracted, to give the following splitting of **A**:

$$\omega\mathbf{A} = (\mathbf{D}+\omega\mathbf{E}) + [\omega\mathbf{F} - (1-\omega)\mathbf{D}], \tag{3 .78}$$

then the iteration equation is given by:

$$\mathbf{x}_{k+1} = (\mathbf{D}+\omega\mathbf{E})^{-1}\{\omega\mathbf{b} - [\omega\mathbf{F} - (1-\omega)\mathbf{D}]\mathbf{x}_k\}. \tag{3 .79}$$

The value of ω is $0 < \omega < 2$. If ω is outside this region, this method goes unstable, and if $\omega = 1$, the method just reduces to the Gauss-Seidel method. For the region $0 < \omega < 1$, it should be called underrelaxation, but traditionally the whole span of zero to two is referred to as overrelaxation. The choice of ω can have a significant effect on the rate of conver-

gence and hence shorten the number of iterations, but the optimal value is not easy to find and varies with the problem. One method is to vary ω slightly and see if it improves the rate of convergence. This variation can be done by dithering ω while the code is running or by making several runs with different values of ω. Once an optimal ω has been determined, then it is best to leave it constant to make the code run economically.

Each of the above methods has an iteration matrix that remains unchanged with each iteration and is, therefore, called a stationary method. For an excellent discussion of the above methods see Strang [81], or Golub & Van Loan [32].

E.2. Non-Stationary Methods

In the 1950s the Conjugate Gradient (CG) method and related methods referred to as non-stationary methods were developed (See Barrett et al. [7], Golub & Van Loan [32], Kelley [41], Saad [71], and Strang [81]). "Non-stationary" means that with each iteration the information for doing the computation changes. These methods have no iteration matrix but rather are based on the orthogonalization of the residual vectors and a minimization of the residual at each iteration. The early methods, such as the CG method, could only be guaranteed to converge if **A** was a symmetric positive-definite matrix.

Lanczos had proposed a biorthogonal method to handle non-symmetric matrices in his 1950s papers [45] and [47], but the idea lay unused for over twenty years. In 1986 a method known as the Generalized Minimal Residual (GMRES) method was introduced that could handle non-symmetric matrices. Since then, a number of methods for handling non-symmetric matrices have been developed. For several texts books on the subject see Barrett et al. [7], Cullum & Willoughby [19], Golub & Van Loan [32], Kelley [41], Saad

[71], and Zlatev [97]. Some of the more successful iterative methods for non-symmetric matrices are:

(GMRES) – Generalized Minimal Residual,

(BiCGSTAB) – BiConjugate Gradient Stabilized,

(CGS) – Conjugate Gradient Squared,

(QMR) – Quasi-Minimal Residual.

These methods can be real 'race horses' compared to the direct method of LU decomposition, but they can also be unpredictable. Usually one or more will converge in much less time than that required by the LU decomposition method. One technique that has been used to try to assure convergence by Barrett [8] is to run several of the methods in parallel, and when one converges, computation on that time-step is stopped, and the code moves on to the next time-step.

The non-stationary iterative methods for solving $\mathbf{R} \equiv \mathbf{Ax} - \mathbf{b} \cong \mathbf{0}$, are based on generating a sequence of orthogonal residual vectors $\mathbf{R}_i$ that are also the gradients of quadratic functions, which, when minimized, lead to a solution $\mathbf{x}$ of the linear system. Since the residual vectors are orthogonal, it follows that they are linearly independent. These methods are also known as Krylov methods because the residuals are projections onto vectors of a Krylov subspace, which is defined as a span or a set of vectors: $\mathbf{K^k} = \{\mathbf{R}_0, \mathbf{AR}_0, \mathbf{A^2R}_0, \ldots, \mathbf{A^{k-1}R}_0\}$. They follow one of four orthogonalization procedures put forth by Gram-Schmidt [81], Householder [38], Lanczos [45], or Arnoldi [6]. Projection is analogous to finding the projection of a vector onto a plane, except that it is an N-Vector projected onto an N-space, or Krylov space.

E.3. Symmetric Positive-Definite Matrices

The Conjugate Gradient (CG) method typifies the fundamentals of the non-stationary iterative methods, and the others are generally variations of this one. The CG method requires that the matrix **A** be symmetric and positive-definite for a minimum of Eq. (3 .80) below to exist. Orthogonalization is done using the Lanczos method for symmetric matrices.

Following Kelley [41], Chapter 2, the Lanczos method reduces a real symmetric matrix **A** to a tridiagonal matrix **T**, and the columns form an orthonormal basis for the projection of **b** onto the Krylov subspace. The residual vectors are each made orthogonal to the previous residuals and to the Krylov subspace. It is then very straightforward to factor the tridiagonal matrix into a triangular and diagonal matrix $\mathbf{T} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}}$. For a tridiagonal matrix, the triangular matrix L consists of only the main diagonal and the first subdiagonal below it. The problem, then, is reduced to solving $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathbf{T}}\mathbf{x} = \mathbf{b}$, which is done in three steps. First, find **y** from $\mathbf{L}\mathbf{y} = \mathbf{b}$, which is simple since **L** has only two diagonals. The second step is to solve for **z** from $\mathbf{D}\mathbf{z} = \mathbf{y}$, which is even easier since **D** is just a diagonal matrix. Third, solve for **x** from $\mathbf{L}^{\mathbf{T}}\mathbf{x} = \mathbf{z}$.

The minimization is accomplished by taking the gradient of the polynomial:

$$\phi(\mathbf{x}) = (1/2)\,\mathbf{x}^{\mathbf{T}}\,\mathbf{A}\,\mathbf{x} - \mathbf{x}^{\mathbf{T}}\,\mathbf{b}. \qquad (3.80)$$

If the vectors **x** and **b** and the matrix **A** are all multiplied out, the result is a polynomial. Setting the gradient of the polynomial to zero yields the linear system being solved and the extremum of Eq. (3 .80),

$$\nabla\phi(\mathbf{x}) = \mathbf{A}\,\mathbf{x} - \mathbf{b} = \mathbf{0}. \qquad (3.81)$$

Thus, minimizing $\phi(\mathbf{x})$ is the same as finding the solution to the linear system. The minimum of Eq. (3 .80) can be found by the Least Squares procedure.

E.4. Non-Symmetric Matrices

If **A** is non-symmetric, one way to handle that is to multiply **A** by its transpose $\mathbf{A}^T$, because the product $\mathbf{AA}^T$ or $\mathbf{A}^T\mathbf{A}$ is symmetric and positive-definite, assuming **A** is non-singular. Using the first product leads to a method called the Conjugate Gradient on the Normal Equations (CGNE), where **x** is redefined as $\mathbf{x} = \mathbf{A}^T\mathbf{y}$, and then two problems are solved. First $(\mathbf{AA}^T)\mathbf{y} = \mathbf{b}$ is solved for **y** using the CG method, and then $\mathbf{x} = \mathbf{A}^T\mathbf{y}$ is computed. The second product, $\mathbf{A}^T\mathbf{A}$, leads to the method called the Conjugate Gradient on the Normal equations Residual (CGNR) where both sides of the linear system are multiplied from the left by the transpose of **A**, that is, $(\mathbf{A}^T\mathbf{A})\mathbf{x} = \mathbf{A}^T\mathbf{b}$, and the equation is solved using the CG method. Both of these methods, however, converge rather slowly. Also, the transpose has to be generated.

More efficient techniques have now been developed to solve $\mathbf{R} \equiv \mathbf{Ax} - \mathbf{b} \cong \mathbf{0}$, when **A** is non-symmetric, and they have taken two major branches, one based on Arnoldi orthogonalization and the other on non-symmetric Lanczos biorthogonalization. The Arnoldi process is used in the GMRES iterative technique and was introduced by Saad & Schultz [72]. The Lanczos biorthogonalization process has led to the iterative techniques BiCG, BiCGSTAB, CGS, and QMR. The Arnoldi process is the easier of the two to analyze, so GMRES has been more extensively studied than the others. A good comparison of the various methods is found in the book by Barrett et al. [7]. The general conclusion they reached is that GMRES is the more robust in that it will converge eventually, but it

uses up a lot of memory. The others may converge much faster and use less memory, but it is possible they might not converge.

E.5. Arnoldi Orthogonalization for Non-symmetric Matrices

The Arnoldi process [6] uses the Gram-Schmidt orthogonalization method coupled with ideas of Hestenes and Stiefel [35] and allows the solution for non-symmetric matrices. Instead of reducing **A** to a tridiagonal matrix **T**, it is reduced to Hessenberg form, in which the elements of the matrix are all zero below the first subdiagonal. (A tridiagonal matrix is also in Hessenberg form, but it is the result of starting from a symmetric matrix.)

The Generalized Minimal Residual (GMRES) is a method for handling non-symmetric matrices, and is based on the Arnoldi procedure. For GMRES the Gram-Schmidt orthogonalization is commonly used, although the Householder method is also used. The Gram-Schmidt method, however, is better for parallelization, [7] p. 21.

The main problem with this technique is that the entire sequence of orthogonal vectors for each iteration needs to be saved, which can require a large amount of memory. Because the solution is not formed for each iteration, the residual can be minimized without it. Restarting the procedure, by forming the approximate solution and starting over after some number of iterations m, can alleviate this problem. It can be difficult, however, to decide what value of m to use. The GMRES method may be somewhat slower and use more memory than the following methods, but it is commonly used because it is considered to converge more reliably.

E.6. Lanczos Biorthogonalization for Non-symmetric Matrices

Lanczos proposed a method for handling non-symmetric matrices that uses two orthogonal bases and two Krylov subspaces, one for a sequence on $\mathbf{A}$ and the other on $\mathbf{A}^T$. The two sequences are made mutually orthogonal, instead of orthogonalizing each sequence. The resulting method is called Bi-orthogonal Conjugate Gradient (BiCG) (also known as BCG in some texts), but this method proved to have unreliable convergence. More stable convergence can be obtained, however, by using a different update on the $\mathbf{A}^T$ sequence, and this method is called Bi-Conjugate Gradient STABilized (BiCGSTAB).

The Conjugate Gradient Squared (CGS) method is a modification such that the sequence for the transpose $\mathbf{A}^T$ does need to be found, and therefore it can converge about twice as fast as BiCG. It was put forth by Sonneveld in 1989 [75]. Some claim in the literature that this method is more likely to have convergence problems than BiCG.

The Quasi-Minimal Residual (QMR) algorithm was introduced by Freund and Nachtigal [26] in 1991, and uses a "look ahead" technique to stabilize the BiCG method. It also converges more smoothly.

For the implicit version of the SPHINX code, the GMRES, the CGS, and the BiCGSTAB methods have been written and tested and are working. These Krylov solvers have been compared to the versions in the commercial code MATLAB, which is an excellent code for matrix manipulation. The CGS method has converged a little faster than the GMRES and BiCGSTAB methods for the SPH matrices tried to date. For the type of matrices generated from the implicit code, the CGS method has proven to be very reliable and hence has become the one most used for this dissertation.

F. The Newton-Raphson Iteration

The Krylov solvers solve the linear problem, but in general the equations are non-linear, and there are several approaches to solving the nonlinear problem. The Newton-Raphson iteration method for solving nonlinear problems is used in the implicit version of the code. In 1D, for example, this method works by iterating on the residual until its intercept at zero is reached within a given tolerance by extending the tangent of the curve at a point and then taking the intercept of the tangent line as a new estimate to the intercept of the curve. Extending it to multiple dimensions is similar. The iterations are done within one time-step to improve the accuracy of the nonlinear problem. This process converges quadratically, so it can usually be stopped in just a few iterations. The book by C. T. Kelly [41] discusses iterative methods for both linear and nonlinear systems of equations.

The residual is defined as: $\mathbf{R(Y)} \equiv d\mathbf{Y}/dt - \mathbf{f(Y)} \cong 0$, which is the approximation of Eq. (3 .7). For the current problem it represents the SPH equations (2 .31) to (2 .36) where $\mathbf{Y}$ is the state vector of dependent variables and $\mathbf{f(Y)}$ is a vector function representing all of the right-hand sides of the SPH equations for each particle.

The Newton-Raphson iteration is given by:

$$\mathbf{Y}_{j+1} \cong \mathbf{Y}_j - [\mathbf{R}'(\mathbf{Y}_j)]^{-1}\, \mathbf{R}(\mathbf{Y}_j), \tag{3 .82}$$

where, for the implicit code $\mathbf{R}'(\mathbf{Y}_j) = \partial \mathbf{R}(\mathbf{Y}_j) / \partial \mathbf{Y} \equiv \mathbf{J}$, which is the Jacobian matrix of the residuals $\mathbf{R(Y)}$. The subscript j indicates the sequence of estimates for the vector of dependent variables $\mathbf{Y}$. The iteration equation (3 .82) is derived from the Taylor expansion of $\mathbf{R(Y)} = \mathbf{0}$,

$$\mathbf{R(Y)} = \mathbf{R}(\mathbf{Y}_0) + (\mathbf{Y}_1 - \mathbf{Y}_0)\, \mathbf{R}'(\mathbf{Y}_0) + \ldots = \mathbf{0}. \tag{3 .83}$$

Keeping the two terms shown, Eq. (3 .83) can be solved for $\mathbf{Y}_1$ and obtain:

$$\mathbf{Y}_1 \cong \mathbf{Y}_o - [\mathbf{R}'(\mathbf{Y}_o)]^{-1}\ \mathbf{R}(\mathbf{Y}_o). \tag{3 .84}$$

Given a first guess $\mathbf{Y}_o$, then a new estimate can be calculated $\mathbf{Y}_1$, which can then be put back in and another estimate $\mathbf{Y}_2$ can be found, and in most cases each iteration will be a better solution to $\mathbf{R(Y) = 0}$. Iterating in this fashion is best represented by Eq. (3 .82). This technique is used within the code to find the step $\mathbf{dY}$, so that $\mathbf{Y}$ of Eq. (3 .82) is substituted if for $\mathbf{dY}$ of Eq. (3 .11).

Occasionally an estimate is on the wrong side of, or too close to, a minimum or a maximum, and the tangent line takes the next estimate way out of range or off to infinity. Other conditions can cause the residuals to oscillate between two points and not converge. Techniques exist that correct for that within the Newton iteration. One method, known as a line-search [41] [66], tests the residuals from one Newton iteration to the next to make sure they are monotonically decreasing. If the residuals increase, all the elements of the vector $\mathbf{dY}$ are cut by some fraction, usually 1/2. The residuals are then recalculated and compared to the residuals of the previous Newton iteration to see if it is smaller. If not, $\mathbf{dY}$ is cut in half again, and the line-search iterates like this until the residual is smaller. The idea is simple. It is using the fact that $\mathbf{dY}$ is the direction to go, sometimes called the Newton-direction, but it is just going too far. In cutting $\mathbf{dY}$ in half again and again, it is moving $\mathbf{dY}$ back toward the previous estimate, but maintaining the Newton-direction. It is doing nothing to the time-step size, or the final solution. It is just iterating within a Newton iteration to improve the next estimate. One to five iterations of the line-search is usually sufficient.

G. Sparse Storage and Computations

One of the main problems with implicit computations is the storage of the very large matrix within the computer memory. This problem can be ameliorated since each particle has only a few neighbors, and hence a major portion of the matrix is composed of zeros. So to conserve on memory, only the non-zero elements of the matrix need to be stored. This method can allow for more particles, and hence higher resolution, to be used in a given problem. The most straightforward method of storing a matrix is to store each non-zero element along with its m and n indices in correlated vectors. This method of storage is not the most efficient, but it is the easiest for the user to read, and the order in which the elements are stored can be completely random as long as each element is correlated with its indices. This method is used in the present sparse implicit code with the diagonal elements being stored at the beginning of the matrix, so that the time-step can be reduced if the Krylov solver fails to converge, or so that the diagonal can be used in a Jacobi preconditioner. Multiplication of a matrix in this form by a vector of the appropriate length is very simple because the code just steps down through the vectors, the sums of the products can be added in any order, and the indices will direct the addition of each product to the appropriate element in the resultant vector.

One can also take advantage of the sparseness of the matrix in another way by eliminating computations where zeros are known to occur, hence saving time. In the SPHINX code each particle has a list or neighbors, so only the off-diagonal blocks due to neighbors need to be calculated. All other off-diagonal blocks are known to be filled with zeros. The Jacobian function of the code has been written such that it calculates only the non-zero

blocks due to neighbors and none of the zeros of the blocks where there are no neighbors.

More time can be saved by using a right-hand-side (RHS) function that calculates the function for just one particle instead of all of the particles as the existing SPHINX code does. This function allows the implicit code to calculate the RHS for just the neighbors of each particle instead of all the particles for each particle. In forming the Jacobian, the RHS function has to be called for each column of the matrix, and so calculating the RHS of just the neighbors saves time because the rest of the summation terms are zero.

There was one unexpected problem in switching from the full matrix to the sparse storage, and that is, there is a difference in the Jacobian between including the continuity equation and replacing it with a summation method. It turns out that the summation method generates extra non-zero elements in the Jacobian which arise from the neighbors of the neighbors, but only in the columns for the derivatives with respect to velocity. Hence, there is currently a sparse version of the code for use with the continuity equation, and a full matrix version for use with the summation method. Modifying the sparse version of the code to handle these extra matrix elements, so that one code can handle both density methods, has been left as a future task.

H. Preconditioners

The convergence rate of the Krylov solvers has been found to be greatest if the eigenvalues of the Jacobian matrix are clustered near one. (The collection of eigenvalues of a matrix is referred to as the spectrum of the matrix.) Most matrices, however, do not have such nice spectral properties, but many can be transformed into a matrix with better

spectral properties. This transformation is referred to as preconditioning the matrix.

The transformation is accomplished by finding a matrix **M**, which is easy to invert and is approximately the coefficient matrix **A**, of the equation $\mathbf{Ax} = \mathbf{b}$, or finding a matrix that approximates $\mathbf{A}^{-1}$ directly. In either case, the basic idea is that from $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$ the product matrix $\mathbf{M}^{-1}\mathbf{A}$ would be approximately the unit matrix, which has the desired eigenvalues or spectral properties.

Preconditioning the matrix can be very effective in helping the Krylov solver to converge, but it is a trade-off. The preconditioner uses computer time and memory, so it must be simple and cheap to operate. The savings comes when it can reduce the number of Krylov iterations significantly, that is, improve the convergence rate.

The simplest and easiest preconditioners to implement are those that use the Jacobian matrix itself in various disassembled forms. These include use of the iterative methods, described in Section E.1 of this chapter, such as the Jacobi (J), the Gauss-Seidel (GS), and the Symmetric Successive Over-relaxation (SSOR) iterative methods. They are typically used multipass (see Goulb & Van Loan [32]). The single pass Jacobi uses just the diagonal of the matrix. The multipass Jacobi has been implemented in the three Krylov solvers CGS, BiCGSTAB, and GMRES and has been tried as a 1-, 2-, 3-, and 4-pass preconditioner. From the results of several test cases, the 2-pass Jacobi preconditioner was found to be the most effective in minimizing the number of Krylov iterations. Because of the success of the 2-pass Jacobi, the SSOR has not been implemented but has been recommended as being better than the Jacobi preconditioner. It would require different storage of the Jacobian matrix and reprogramming the matrix-vector multiply.

I. A Time-step Method for the Implicit Code

A time-step method has been added that allows for time-steps larger than explicit time-steps. The largest time-step allowed is based on the idea that no two particles should be allowed to hit during one time-step. The minimum time, dT, for any two particles to collide is found based on current values, and this time can then be cut in half or a quarter or whatever fraction is found to be appropriate for a given problem. If r is the distance between particles i and j, and v_r is the relative velocity between them, then $dT = r/v_r$. So a search through all particles and their neighbors for a minimum dT uses both r and v_r for each pair, and a fraction of dT is used as the maximum allowed time-step. It has been implemented such that it does not matter if the relative velocity of the pair is toward or away from each other.

The relative velocity is found by using the dot product of the difference vector $\mathbf{dv} = \mathbf{v}_i - \mathbf{v}_j$ of the two particles *i* and *j* with a unit vector $\mathbf{r}/|\mathbf{r}|$. That is, $v_r = \mathbf{dv} \cdot (\mathbf{r}/|\mathbf{r}|)$; or $v_r = \text{abs}((dx*dv_x + dy*dv_y + dz*dv_z)/r)$; where $r = \sqrt{(dx*dx + dy*dy + dz*dz)}$; and v_r and $r = |\mathbf{r}|$ are both scalar, and both use the positive root so that dT is positive.

There are other limits placed on the time-step size. The explicit code ramps the initial time-steps up to its own computed time-step, and any portion of this ramp can be used by the implicit code. In addition to that, the implicit code has its own ramp which is a steadily increasing multiplier, dtmult, of the explicit time-step, and it will continue to increase until it reaches either a maximum, dtmultmax, or dT times a fraction, whichever is smaller. Currently the fraction of dT, the rate of increase of dtmult, and dtmultmax are arbitrarily chosen by the user. If either the Krylov solver or the Newton-Raphson iterations fail to converge, then dtmult is reduced by some fraction, which is also user deter-

mined. Then ramping-up of dtmult starts again, so this ramping might be used at any point during the calculation not just for the initial time steps. There may be a rational way of deciding what the user determined values should be, but to date it has been trial and error, and it may need to be adjusted for various problems.

Since the implicit code uses the explicit time-step as a basis, it is using the same considerations in choosing the time-step as the explicit code. These depend on the physics of the particular problem being run, and they might include such things as material strength, viscosity, or the Courant time-step.

Another approach to determining the time-step may be to base the time-step on dT only, except for an initial ramp-up. The increasing dtmult might be replaced by varying the fraction that multiplies dT. The fraction might be ramped up to some value similarly to dtmult. The implicit time-step would then be independent of the explicit time-step.

J. A Matrix-Free Method

Within the iterative solvers there are typically one or two occurrences of a matrix-vector multiply that yields another vector, and this is the only place in the iterative techniques where the matrix is actually needed. This product can be replaced with a Taylor expansion, which is just a summation of vector operations. The product of the Jacobian matrix **J** and the vector **v** can be replaced by the difference of the residual **R(Y)** and the perturbed residual, which is perturbed by ε weighted by the vector **v**:

$$\mathbf{q} = \mathbf{J} \bullet \mathbf{v} \cong \frac{\mathbf{R}(\mathbf{Y} + \varepsilon \mathbf{v}) - \mathbf{R}(\mathbf{Y})}{\varepsilon}, \tag{3.85}$$

where the residual is defined as: $\mathbf{R}(\mathbf{Y}) \equiv d\mathbf{Y}/dt - \mathbf{f}(\mathbf{Y}) = 0.$ (3.86)

Following an example by Knoll et al. [43] to illustrate the concept behind this matrix-free method, consider the product of a two-by-two Jacobian matrix $\mathbf{J} = \partial \mathbf{R}(\mathbf{Y}_j) / \partial \mathbf{Y}$ and a 2-vector $\mathbf{v}$, which yields a new 2-vector:

$$\mathbf{J} \bullet \mathbf{v} = \begin{bmatrix} \frac{\partial R_1}{\partial Y_1} & \frac{\partial R_1}{\partial Y_2} \\ \frac{\partial R_2}{\partial Y_1} & \frac{\partial R_2}{\partial Y_2} \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} v_1\left(\frac{\partial R_1}{\partial Y_1}\right) + v_2\left(\frac{\partial R_1}{\partial Y_2}\right) \\ v_1\left(\frac{\partial R_2}{\partial Y_1}\right) + v_2\left(\frac{\partial R_2}{\partial Y_2}\right) \end{bmatrix}. \tag{3.87}$$

Multiplying and dividing both elements of the new vector by a scalar perturbation ε and then adding and subtracting the respective error for each element to the numerators, leaves it unchanged in value.

$$\mathbf{J} \bullet \mathbf{v} = \begin{bmatrix} \frac{\varepsilon v_1\left(\frac{\partial R_1}{\partial Y_1}\right) + \varepsilon v_2\left(\frac{\partial R_1}{\partial Y_2}\right)}{\varepsilon} \\ \frac{\varepsilon v_1\left(\frac{\partial R_2}{\partial Y_1}\right) + \varepsilon v_2\left(\frac{\partial R_2}{\partial Y_2}\right)}{\varepsilon} \end{bmatrix} = \begin{bmatrix} \frac{\left\{R_1 + \varepsilon v_1\left(\frac{\partial R_1}{\partial Y_1}\right) + \varepsilon v_2\left(\frac{\partial R_1}{\partial Y_2}\right)\right\} - R_1}{\varepsilon} \\ \frac{\left\{R_2 + \varepsilon v_1\left(\frac{\partial R_2}{\partial Y_1}\right) + \varepsilon v_2\left(\frac{\partial R_2}{\partial Y_2}\right)\right\} - R_2}{\varepsilon} \end{bmatrix}. \tag{3.88}$$

Then, recognizing that the first three terms of each numerator are the zeroth and first-order terms of a Taylor expansion of a function of two variables, Eq. (3.88) can be rewritten approximately as:

$$\mathbf{J} \bullet \mathbf{v} \approx \begin{bmatrix} \frac{R_1(Y_1 + \varepsilon v_1, Y_2 + \varepsilon v_2) - R_1(Y_1, Y_2)}{\varepsilon} \\ \frac{R_2(Y_1 + \varepsilon v_1, Y_2 + \varepsilon v_2) - R_2(Y_1, Y_2)}{\varepsilon} \end{bmatrix}, \tag{3.89}$$

or back to vector notation:

$$\mathbf{J} \bullet \mathbf{v} \approx \frac{\mathbf{R}(\mathbf{Y} + \varepsilon \mathbf{v}) - \mathbf{R}(\mathbf{Y})}{\varepsilon}, \tag{3.90}$$

and similarly for function of three or more variables. A prescription for the scalar perturbation is given by:

$$\varepsilon = \frac{1}{N\|\mathbf{v}\|_2}\sum_{m=1}^{N}(b|\mathbf{Y}_m|), \tag{3.91}$$

where N is the dimension of the Jacobian, and b is a constant determined by machine round-off. For single precision they recommend $b = 10^{-5}$.

This method has been attempted in the three Krylov solvers of the implicit code, but has worked for only a few of the test cases, when the single-pass Jacobi preconditioner is used. It was initially hoped that the single-pass Jacobi, which is just the diagonal of the Jacobian, could be used routinely, but it does not work as well as the 2-pass Jacobi, which requires the complete Jacobian matrix, and that defeats the purpose of going to matrix-free methods. For matrix-free to work well, a more memory-efficient preconditioner will need to be developed. The preconditioner is itself a matrix, and so for the matrix-free method to be advantageous, the preconditioner should use significantly less memory than the Jacobian matrix. One idea is to use just the diagonal blocks along the diagonal of the Jacobian for the preconditioner. An algorithm for inverting each block, taking advantage of the known zeros, has been developed, but has not been programmed or tested. Since using the diagonal blocks as the preconditioner is an unknown approach, it has been put off as a future task. Elimination of the huge Jacobian matrix would allow larger problems to be run because more memory would be available. In running some of the problems that the explicit code runs easily, the implicit code, to date, can only run if the number of particles is reduced; otherwise it will not fit in the memory of the same computer.

K. The Theta Parameter

The results of Section A of this chapter can be modified by introducing a parameter theta, (θ), that adjusts the degree of implicitness or explicitness by rewriting Eq. (3 .8) as an average of the current and next time-steps, and then doing the Taylor expansion of Eq. (3 .9). This is a standard technique found in many texts such as Oran & Boris [64], Section 4-2, for instance. Assuming θ is a fraction between zero and one, then the following equation can switch continuously between fully implicit and fully explicit. That is, when $\theta = 0$, it is fully explicit; when $\theta = 1$, it is fully implicit; and in between it is semi-implicit. Making the fractional parts of $\mathbf{Y}_n$ and $\mathbf{Y}_{n+1}$ add to 1, Eq. (3 .8) becomes:

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + \Delta t\,[(1-\theta)\mathbf{f}(\mathbf{Y}_n) + \theta\,\mathbf{f}(\mathbf{Y}_{n+1})]\,. \tag{3 .92}$$

Then by expanding $\mathbf{f}(\mathbf{Y}_{n+1})$ in a Taylor series, keeping only the first two terms, and collecting the $\mathbf{Y}_{n+1}$ terms, as in Eqs. (3 .8) through (3 .11), only a slight change occurs; a θ is introduced in the Jacobian.

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + \Delta t\,(1-\theta)\,\mathbf{f}(\mathbf{Y}_n) + \Delta t\,\theta\,[\mathbf{f}(\mathbf{Y}_n) + \partial\mathbf{f}/\partial\mathbf{x}|_{\mathbf{x}_n} \bullet (\mathbf{Y}_{n+1} - \mathbf{Y}_n)]\,, \tag{3 .93}$$

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + [\mathbf{I}/\Delta t - \theta\,\partial\mathbf{f}/\partial\mathbf{Y}]^{-1} \bullet \mathbf{Y}'_n\,. \tag{3 .94}$$

Comparing to Eq. (3 .10) the Jacobian is $\mathbf{J} = [\mathbf{I}/\Delta t - \theta\,\partial\mathbf{f}/\partial\mathbf{Y}]$. Using a value for θ between 0.55 and 0.75 has been found to improve the agreement between the explicit and implicit codes.

With the introduction of θ, the other equations are also found to be affected. Expanding the residual or error equation $\mathbf{R}(\mathbf{Y}) \equiv d\mathbf{Y}/dt - \mathbf{f}(\mathbf{Y}) = 0$, Eq. (3 .86), in terms of the θ parameter one obtains:

$$\mathbf{R}(\mathbf{Y}) \cong (\mathbf{Y}_{n+1} - \mathbf{Y}_n)/\Delta t - [(1-\theta)\mathbf{f}(\mathbf{Y}_n) + \theta\,\mathbf{f}(\mathbf{Y}_{n+1})] \cong 0, \tag{3 .95}$$

from which the Jacobian of Eq. (3 .94) can be derived in another way, by taking the derivative of $\mathbf{R(Y)}$ with respect to $\mathbf{Y}_{n+1}$:

$$\mathbf{J} \equiv \mathbf{R}'(\mathbf{Y}) = \partial \mathbf{R}(\mathbf{Y}) / \partial \mathbf{Y}_{n+1} = [\mathbf{I}/\Delta t - \theta\, \partial \mathbf{f}/\partial \mathbf{Y}]. \qquad (3.96)$$

This also demonstrates that $\mathbf{J}$ as used in this dissertation is the Jacobian of the residual function $\mathbf{R(Y)}$.

The θ version of the Newton-Raphson iteration of Eq. (3 .82), along with Eqs. (3 .95) and (3 .96), then becomes:

$$\mathbf{Y}_{n+1}^{j+1} = \mathbf{Y}_{n+1}^{j} - \frac{(\mathbf{Y}_{n+1}^{j} - \mathbf{Y}_{n}^{j})/\Delta t - [(1-\theta)\mathbf{f}(\mathbf{Y}_{n}^{j}) + \theta \mathbf{f}(\mathbf{Y}_{n+1}^{j})]}{[\mathbf{I}/\Delta t - \theta\, \partial \mathbf{f}(\mathbf{Y}_{n+1}^{j})/\partial \mathbf{Y}_{n+1}^{j}]}. \qquad (3.97)$$

The matrix-free method of Eq. (3 .85) requires the residual be perturbed by $\varepsilon \mathbf{v}$:

$$\mathbf{R}(\mathbf{Y}+\varepsilon \mathbf{v}) \cong (\mathbf{Y}_{n+1} + \varepsilon \mathbf{v} - \mathbf{Y}_{n})/\Delta t - [(1-\theta)\mathbf{f}(\mathbf{Y}_{n}) + \theta\, \mathbf{f}(\mathbf{Y}_{n+1}+\varepsilon \mathbf{v})] \cong 0, \qquad (3.98)$$

so by subtracting Eqs. (3 .95) and (3 .98) and dividing by ε, a θ version is obtained:

$$\mathbf{q} = \mathbf{J} \bullet \mathbf{v} \cong \frac{\mathbf{R}(\mathbf{Y}+\varepsilon \mathbf{v}) - \mathbf{R}(\mathbf{Y})}{\varepsilon} \cong \frac{\mathbf{v}}{\Delta t} - \theta \left\{ \frac{\mathbf{f}(\mathbf{Y}_{n+1} + \varepsilon \mathbf{v}) - \mathbf{f}(\mathbf{Y}_{n+1})}{\varepsilon} \right\}. \qquad (3.99)$$

These relations have all been fully implemented into the full-matrix, the sparse-matrix, and the matrix-free versions of the implicit code.

Chapter IV

Test Cases

A. Introduction

The next step in developing the new implicit code is to test it for accuracy and speed. To check out the accuracy, the results of the new code can be compared to analytic solutions or to experimental results, or if neither of those exist, then they can be compared to the results of other codes. The implicit code has been compared to all three types of results for several different fluid problems, and compares very well. The code has been tested using 1D, 2D, and 3D cases. To test the speed of the new code, it has been compared to the SPHINX code, and a regime has been found for which the implicit code can perform the computation faster than the explicit code, as is discussed in Section G of this chapter.

The fluid problems to be discussed in this chapter include a three-particle problem, a rarefaction problem, a shock-tube problem, a Rayleigh-Taylor instability, a breaking dam problem, and a single gas-jet problem. In brief, for the three-particle and the single gas-jet problems, both the implicit and explicit codes agree very well with each other. For the rarefaction and shock-tube problems there are analytic solutions, and both codes agree very well with each other and with the analytic solutions. For the Rayleigh-Taylor instability the two codes agree well with each other but only agree with the analytic solution for the first e-folding time. For the breaking dam problem there is experimental data to compare to, and both codes agree well with the data.

B. A Three-Particle Problem

The three-particle test problem is useful because the Jacobian matrix is small enough to be calculated by hand using the equations developed in Chapter III. The matrix is 12 x 12 for the 1D problem, 18 x 18 for the 2D problem, and 24 x 24 for the 3D problem. Both the analytic and numerical Jacobian algorithms generated matrix elements that were in excellent agreement with the hand calculations for 1D and 3D. 2D was not calculated by hand, but most of the elements would be the similar. These matrices are also easily handled by the commercial software packages Mathematica and Matlab. There are also Krylov solvers available for Matlab, so that one can compare step-by-step the results of the Krylov solvers of the new implicit code to those for Matlab. This process was done for the three solvers incorporated in the implicit code, CGS, BiCGSTAB, and GMRES.

This test problem consists of three particles in a row and was run in one and three dimensions. In 1D (shown in Fig. IV. 1) the particles were set up 1 cm apart with zero initial velocity, an initial temperature of 300 ^{o}K, and a density of 2.7 g/cc. The perfect-gas model was used for the Equation of State (EOS), with a ratio of specific heats of $\gamma = 1.4$ and an average molecular weight of $\mu = 1.0$. Initially the smoothing length h was set equal to the particle separation, and was allowed to vary. That means that each end particle had only the middle particle as a neighbor, and the middle particle had both end particles as neighbors. The density method used was the continuity equation.

Because of the pressure between the particles, the two outer particles accelerate out in opposite directions symmetrically with each time-step, and the middle one remains stationary. The density associated with each particle decreases as they move apart until their smoothing lengths no longer overlap, at which point the densities remain constant as

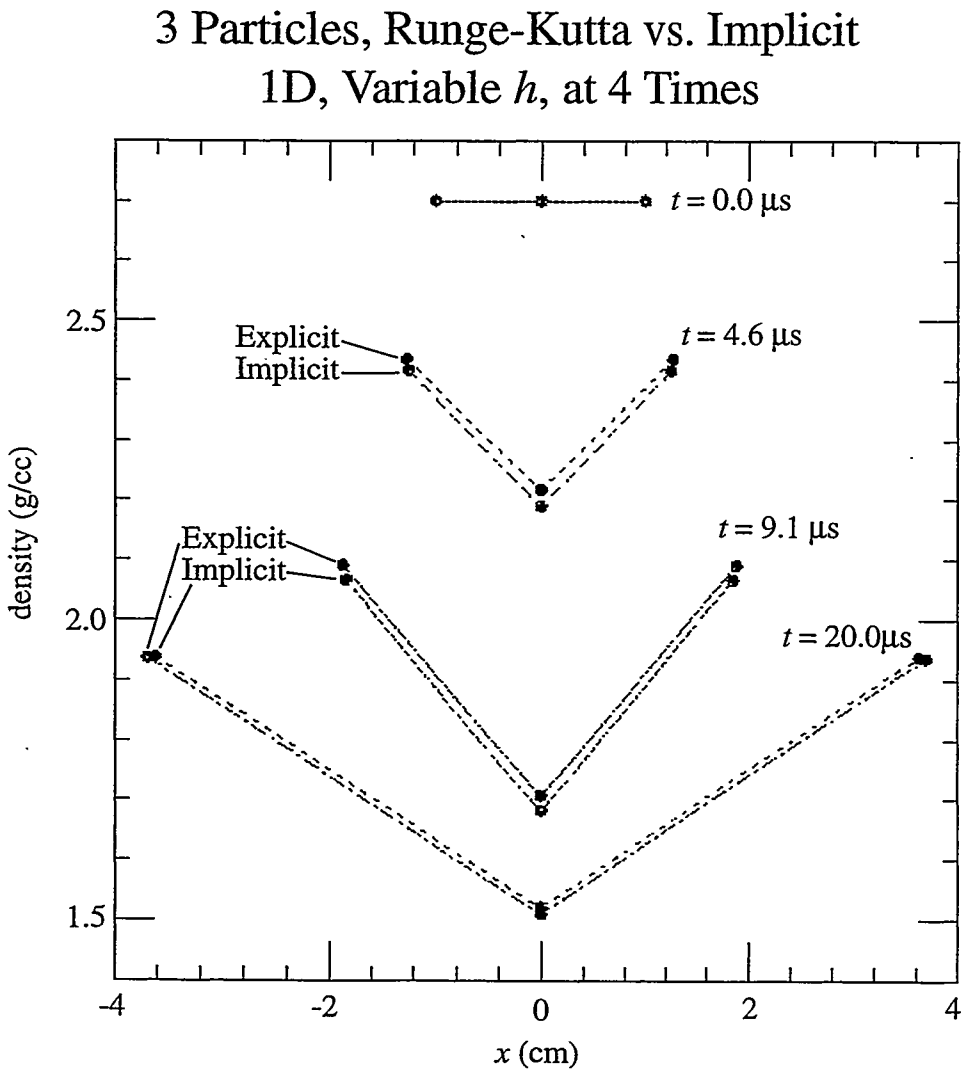


Fig. IV. 1. A plot of the 3-particle problem in one dimension with variable h shows a comparison of the explicit and implicit cases at four different times. The explicit code was run using the Runge-Kutta time-stepping method. The lines drawn between particles connect particles of the same time and case.

the particles continue to separate at constant speed. The density for the middle particle decreases most because there are terms in the summations contributed by two neighbors, whereas each end particle has contributions from only one neighbor. Both the explicit and implicit codes agree fairly well for $\theta = 1.0$, and 0.5, and very well for $\theta = 0.75$ for change in density and distance traveled per time-step. The theta parameter is discussed in

Chapter III Section K and varies the amount of implicitness in the code.

Figure IV. 1 shows a comparison of the explicit code using the Runge-Kutta time-step routine versus the implicit code using the same time-steps for both, and shows how the density changes with time. These results were run using the full matrix. Theta was set to 0.75. The particles are shown connected by lines that are at the same time and same case. They start out at the top of the plot at $t = 0.0$ μsec with all the particles at the same density for both cases. Then at $t = 4.6$, 9.1, and 20.0 μsec they form a V because the density of the middle particle has dropped faster than that of the end particles.

One problem encountered in doing the three-particle problem is when to recalculate the variable smoothing-length h within the code. There is a function in the code called *rhs* which calculates all the right-hand-sides of all the equations for each of the particles. The problem is when to recalculate h relative to the *rhs* function. For the sparse matrix version of the implicit code, the *rhs* function needs to be performed before h is changed. For the matrix-free method it has to be the other way around, because if it is not, then spurious derivatives are generated in the Jacobian matrix. For the full-matrix version, similar results are obtained with either arrangement; however, to obtain the best agreement with h changed before *rhs* is recalculated for the explicit code, θ needs to be changed from $\theta = 0.75$ to $\theta = 0.55$.

The three-particle problem was also run in 3D, and the row of particles was oriented along each of the three axes x, y, and z, in three separate cases. This was done to verify that the code obtained identical results in each of the three directions, which it did. This was the sole purpose of doing the 3D three-particle problem.

C. A Rarefaction Problem

The rarefaction test problem models a 1D gas, bounded on the left, expanding into a vacuum to the right, causing a rarefaction wave to move backward into the gas. The gas is hydrogen, modeled as a perfect-gas, with a ratio of specific heats of $\gamma = 1.4$ and an average molecular weight of $\mu = 1.0$, and it is initially contained in a cylinder 1 cm in length. The radius of the cylinder is unnecessary here because this is a 1D computation. Initially the hydrogen is at rest at a temperature of 300 ^{o}K.

This problem has an analytic solution in the form of density versus x, (see Vol. I, Chapter 1, Section 10 of Zel'dovich & Raizer [96]):

$$\rho = \rho_o\left[1 - \frac{(\gamma-1)v}{2c}\right]^{(2\gamma)/(\gamma-1)}, \quad \text{where} \quad v = \frac{2}{(\gamma-1)}\left[c + \frac{(x-x_o)}{t}\right]. \qquad (4.1)$$

The density is denoted by ρ, where ρ_o is the initial density, the speed of sound by c; γ is the ratio of specific heats, and x and t are space and time variables respectively.

The gas is modeled by 97 particles set up in a row along the x-axis, with an initial density of 1.0 g/cc along a distance of 1cm. As the particles move out into the vacuum on the right hand side they drop in density to approximately 0.0 g/cc. The smoothing length h is allowed to vary, and is initially set to 1.5 times the distance between particles. Figure IV. 2 show the implicit result (solid curve) of the rarefaction problem as compared with the analytic solution (dashed curve), using explicit time-steps.

This test problem runs correctly for both the explicit and implicit codes when the summation method, Eq. (2 .24), is used instead of the continuity equation. When using the continuity equation, however, both codes give a result that "hooks up" in density as a function of x, which is physically incorrect. This problem is not well understood, but is

inherent in the explicit code and is not a function of the time-stepping method used.

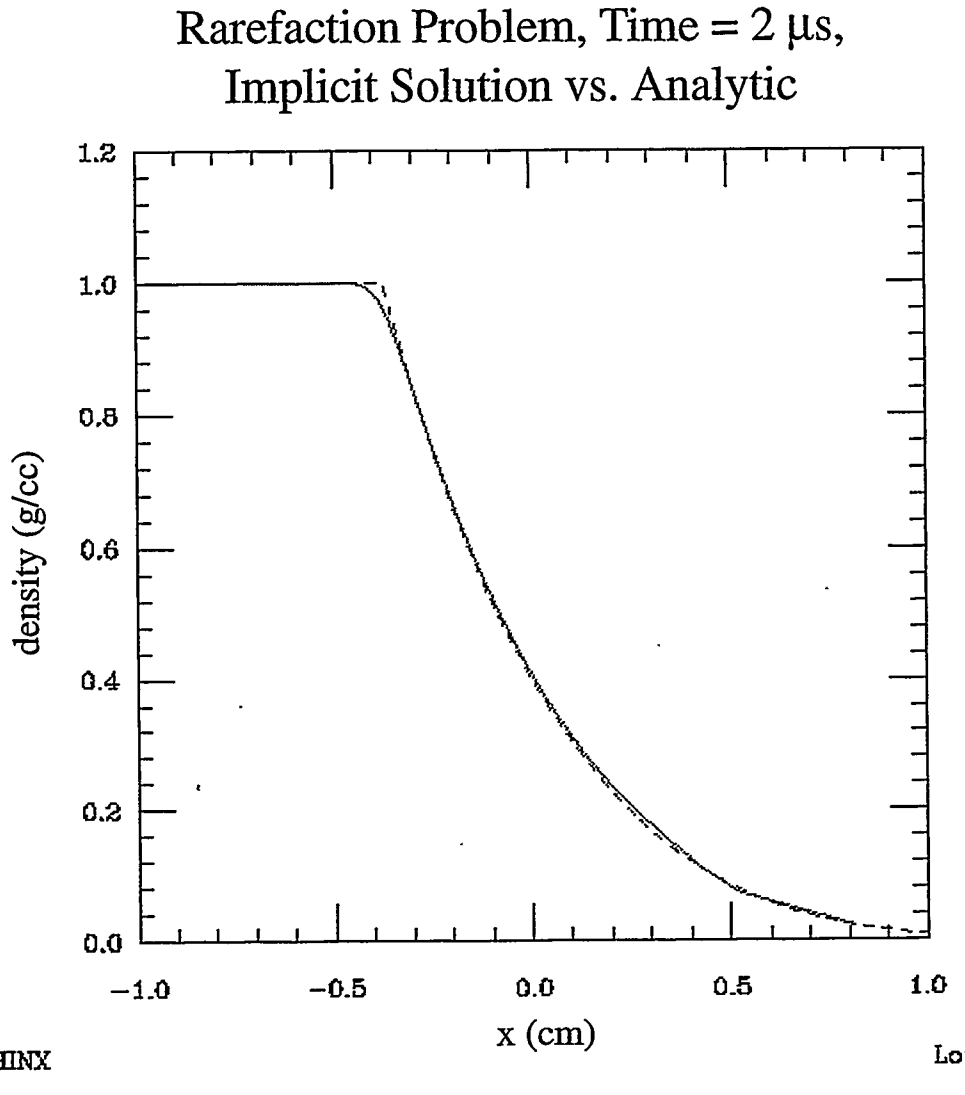


Fig. IV. 2. This plot shows both the Implicit (solid) and analytic (dashed) solutions. Shown here are the lines connecting the particles. The particles are initially placed along the x-axis between $x = -1.0$ and 0.0 cm, all at the same density of 1.0 g/cc. By 2 μs they have expanded out to 1.0 cm. The explicit solution is essentially the same as the implicit solution, so the solid curve could represent either.

The smoothing lengths are calculated differently when the continuity equation is used, and the first particles to move out into the vacuum soon become detached from the others, and from then on their densities cannot change. The "hook" is generated as they

drift on out with a constant density, whereas the ones behind them continue to drop in density because they still have overlapping h's and are interacting with each other. Running this problem with the continuity equation can still be used, however, to compare the two codes for consistency, and they do agree well in that they both produce similar density profiles for either density method.

Another aspect of this disagreement between the use of the summation method or the continuity equation to calculate the density became apparent when the sparse storage was implemented in the implicit code. The full-matrix version of the code agrees with the explicit code for both density methods, but this is not true for the version with sparse storage. It was found that the summation method, unexpectedly, generates extra elements in the velocity columns of the Jacobian (the derivatives with respect to velocity) due to the neighbors of neighbors. Some of the values of the neighbor particles are changed when the summation density method is used. This change, in turn, causes their neighbors to contribute elements of the Jacobian. When the density method is set to the continuity equation, for either the implicit or explicit codes, the values of the neighbors are unchanged, so that only the neighbors contribute the Jacobian elements. The extra elements of the Jacobian, generated by the summation method, may be what is needed to make the continuity equation handle the rarefaction problem correctly, instead of "hooking up" as discussed above.

As a result, the version of the implicit code that uses sparse storage is now restricted to problems that use the continuity equation. Although there is probably a way to make it handle either density method, it has not been pursued any further at this time. Currently, if the summation method is needed, then the full-matrix version will be used, so

that the extra terms it generates are not lost. The full-matrix version gives the same result as the explicit code for either density method but is very wasteful of memory and time.

D. A Shock-Tube Problem

The shock-tube problem is 1D with 2 different gases, of two different densities and pressures within a tube, initially separated by an interface at $x = 0.4$ cm. The densities are 1.0 g/cc on the left, and 4.0 g/cc on the right, and the density method used is the summation method. The gas on the left is at the higher pressure. Both gases are treated as a perfect-gas, and the temperature is initially 300 ^{o}K. The tube is closed off at either end, at $x = -0.3$ cm and $x = 1.1$ cm. At $t = 0$, a boundary separating the gases disappears, a shock wave moves from the interface into the less dense gas, and a rarefaction wave moves in the opposite direction into the denser gas. The smoothing length h was allowed to vary, and it was initially set to 1.5 times the distance between the particles. The example is run to $t = 2$ μsec, and neither wave reaches the end of the tube. If allowed to run long enough, the waves will reflect off the ends of the tube and come back and interact with each other.

The results can be compared to analytic results (See Zel'dovich & Raizer Vol. I, 1967 [96]) and are displayed in Fig. IV. 3 as density versus x. At 2 μsec, the implicit solution is shown as the solid curve, and the analytic solution as the dashed curve. The shock wave has moved to the right into the less dense gas, which is the lower vertical step to the right. The rarefaction wave has moved into the more dense gas to the left, is the left-most step, and has a slope to it. The interface is at the middle vertical step.

Agreement with the analytic solution is fairly good except at most of the corners,

where diffusion has rounded them off. Setting $\theta = 0.75$, as in the rarefaction problem, yields the best agreement between the implicit and explicit codes; in fact, they are almost indistinguishable. However, the implicit code was run with the explicit time-steps. Also, no nonlinear corrections were made because the Newton-Raphson method was turned off.

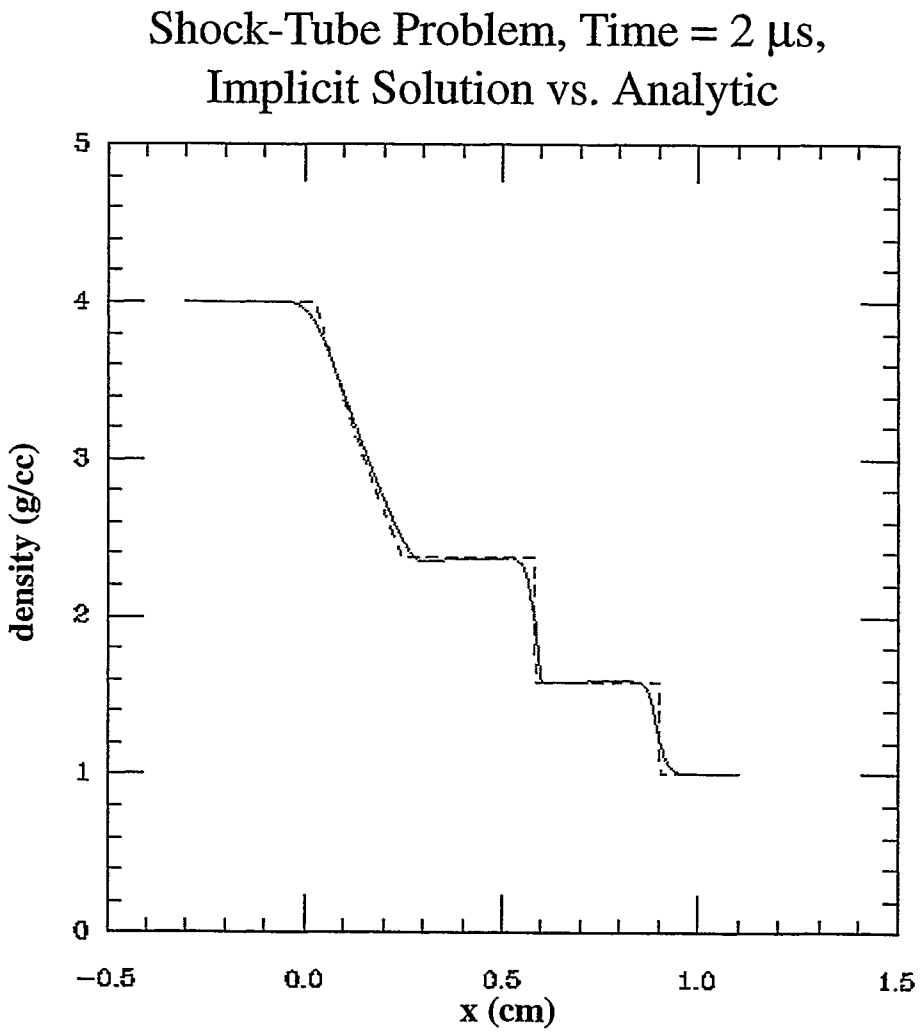


Fig. IV. 3. The implicit (solid) and the analytic (dashed) solutions are shown with lines connecting the particles. The shock wave is the right-most step, the middle step is the interface, and the left-most step is the rarefaction wave. The implicit solution was run with explicit code time-steps and a multiplier of one. Newton-Raphson corrections were not used for this problem. The value of Theta was 0.75.

E. A Rayleigh-Taylor Instability

This example is a 2D problem with a heavy fluid, of density ρ_H, sitting initially on top of a less dense fluid, of density ρ_L, both under the influence of a gravitational acceleration g. Both fluids are assumed to be incompressible. If the interface between the two fluids is flat, in principle it could remain in this state of unstable equilibrium indefinitely. However, any perturbation on the interface will grow exponentially, and in two or three e-folding times go nonlinear, forming the well known "bubble and spike" situation. Then as the spike drops through the less dense gas, depending on the relative densities of the two fluids, the spike can turn into an upside down "mushroom" shape due to sheering, which is also known as the Kelvin-Helmholtz instability. Eventually the system will reach a stable equilibrium with the heavy fluid on the bottom. The early portion of the problem, before it goes nonlinear, is known as the linear Rayleigh-Taylor problem, [13], [14], [31], [37], and this is the only portion to be considered in this section.

During the time when the linear Rayleigh-Taylor instability dominates, the perturbation grows exponentially with time t as $e^{\gamma t}$, where the growth rate γ is given by $\gamma = [(2\pi / \lambda)\, gA]^{1/2}$, λ is the wave length of the perturbation, and A is the Atwood number: $A \equiv (\rho_H - \rho_L)/(\rho_H + \rho_L)$. The exponential solution holds if all the appropriate fluid parameters are initially perturbed together, such as interface position, velocity, and energy. The initial perturbations should also fall off appropriately in moving away from the interface.

A simpler problem to set up on the computer is to perturb either the position or the velocity of the interface, but not both. If only one interface quantity is perturbed, an analytic solution can be found by reworking the boundary conditions. The general solution

for the time dependence of the linear Rayleigh-Taylor problem is the sum of exponentials (see Choi [14]),

$$v(t) = B\,e^{\gamma t} + C\,e^{-\gamma t} = B_1 \cosh(\gamma t) + C_1 \sinh(\gamma t), \qquad (4.2)$$

where B, C, B_1, and C_1 are arbitrary constants determined by the boundary conditions (Note: $B_1 = B+C$ and $C_1 = B-C$). The energy equation has not been included.

If, at $t = 0$, the interface has a spatial perturbation, $y = y_0$, but is stationary, i. e., $dy/dt = v_0 = 0$, then the velocity and spatial perturbations will have the following form:

$$v(t) = \gamma y_0 \sinh[\gamma t], \qquad \text{and} \qquad y(t) = y_0 \cosh[\gamma t], \qquad (4.3)$$

where the spatial solution is found by integrating the velocity solution with respect to time, and the boundaries are assumed to be at infinity. If, however, the interface is initially flat, $y_0 = 0$, but its velocity is perturbed, then the solution is of the following form:

$$v(t) = v_0 \cosh[\gamma t], \qquad \text{and} \qquad y(t) = (v_0 / \gamma) \sinh[\gamma t]. \qquad (4.4)$$

Eqn. (4 .3) is the problem considered in this section.

The question arose: what effect do the finite boundaries have on the solutions, since the Eqs. (4 .3) and (4 .4) are for an infinite medium? Extending the derivations in Hoffman [37] and Choi [14], by putting boundaries at $y = a$, the perturbation velocities are found to have the following spatial dependence (see Appendix for derivation):

$$v(y) = v_0 (e^{-ky} - e^{-2ka}\, e^{ky}) \quad \text{for} \quad 0 < y < a, \text{ and} \qquad (4.5)$$

$$v(y) = v_0 (e^{ky} - e^{-2ka}\, e^{-ky}) \quad \text{for} \quad -a < y < 0, \qquad (4.6)$$

where $k \equiv 2\pi / \lambda$, and the growth rate becomes

$$\gamma^2 = k\, gA[(1 - e^{-2ka})/(1 + e^{-2ka})]. \qquad (4.7)$$

Note: for $a = \infty$ the velocities reduce to $v_0\, e^{-k\,y}$ for $y > 0$ and $v_0\, e^{k\,y}$ for $y < 0$, which is the usual spatial dependence for the linear Rayleigh-Taylor problem with infinite boundaries. Also, note that the velocities go to zero at the boundaries where $y = a$, and the velocities are matched, or equal, on either side of the interface where $y = 0$.

It can be seen from Eqs. (4.5) to (4.7) that the velocities and the growth rate are affected insignificantly by finite boundaries if they are more than λ away from the interface. If $a = \lambda$ and since $k \equiv 2\pi / \lambda$ then $e^{-2ka} = e^{-4\pi} = 3.5e{-}6$, which is small compared to one.

Since the particles should not be allowed to collide (i.e., come within some fraction of the their smoothing lengths) or pass through each other within a time-step, the time-stepping routine imposes an upper limit on how large the time-step can be in problems such as the Rayleigh-Taylor problem where the fluid is being confined or compressed. The fraction multiplying dT is chosen arbitrarily, and for this example has been increased to 1/2, whereas other problems have been run at 1/4 to 1/7. There may be an analytic way to determine what this fraction should be ideally for each example, but this issue has not been pursued.

Figure IV. 4 shows an example where the densities were $\rho_H = 0.05$ g/cc and $\rho_L = 0.01$ g/cc, so that the Atwood number was $A = 2/3$. Initially each fluid was a 2x2 cm square of particles, with a perturbed interface between them at $y = 0$. The perturbation had a wave length of $\lambda = 2$ cm, and an amplitude of 0.2 cm from crest to trough. The

gravitational acceleration was g = −10,000 cm/sec^2 in the y-direction. The growth rate was, therefore, $\gamma = 144.7$ sec^{-1}, and the e-folding time was $t = (1 / \gamma) = 6.9 \times 10^{-3}$ sec. The EOS was a perfect-gas model for both fluids, with $\gamma = 1.4$ (this is a different γ from the growth rate, see Eqn. (2 .37)). A pressure gradient was applied to balance the gravitational force, and matched at the interface. This gradient was actually accomplished by putting in an energy gradient, since energy is one of the dependent variables, and pressure is not. Density variations were calculated using the continuity density method.

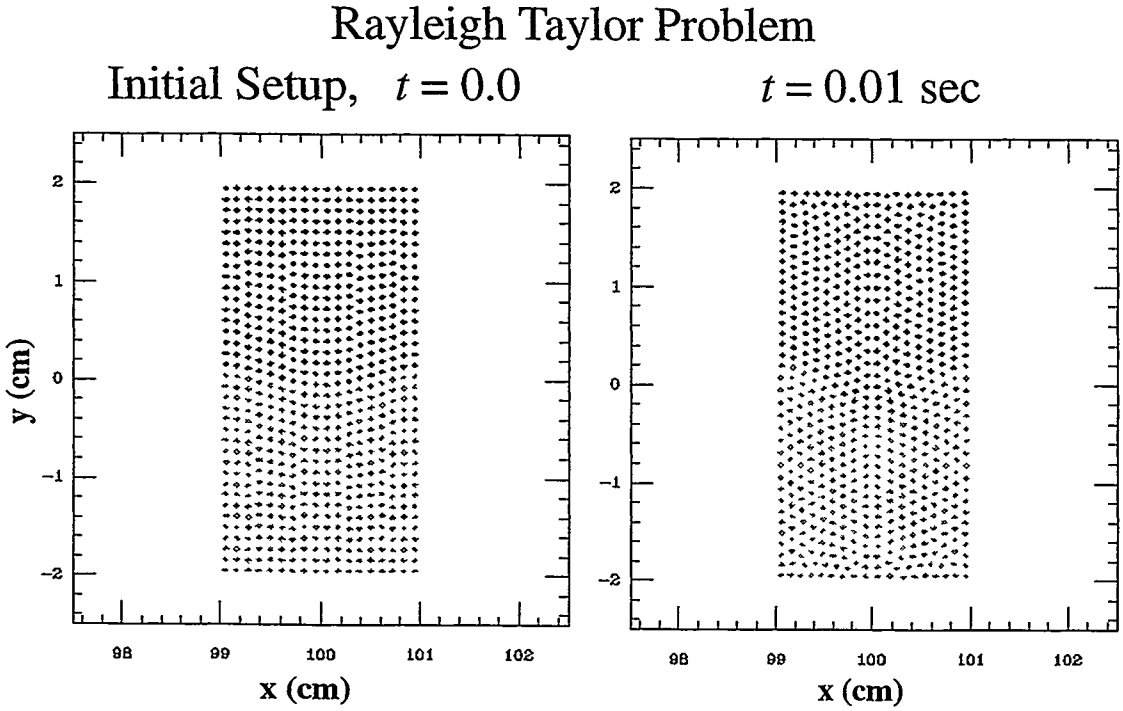


Fig. IV. 4. These plots are an example of a Rayleigh-Taylor problem with the heavy gas on top of a light one. Only half of the problem was calculated, from x = 100 to 101 cm, and then was reflected about the x = 100 cm axis. Shown on the left is the initial setup with a sine wave perturbation of 0.2 cm. The computation at 0.01 sec shows the heavy fluid moving down in the middle.

Because of the symmetry of the problem about a vertical axis, only the right half of the problem was calculated, and then the result was reflected about the left edge of the

result to show one full wave length. There were 326 particles used in the half problem, or 163 in each fluid. The smoothing length was held constant, and was set initially to 1.0, or the distance between particles. Because of the perturbation, each column of particles initially had a slightly different smoothing length, because the spacing was slightly different from column to column and the density had to be constant throughout each fluid, but once set, all h's were held constant during the run.

The plots in Fig. IV. 5 compare the analytic solutions to the implicit and explicit codes. The curve labeled **a** in both parts is the exponential analytic solution. The slope of this curve at $t = 0$ is non-zero, whereas the curves **b** and **c** have an initial slope of zero. To fit the curve **a**, the initial conditions would have to include a perturbation in velocity, position, and energy, but not density because the fluid is assumed incompressible.

The purpose of curve **c** in Fig. IV. 5 is to show that the code results are following a cosh solution at early times, as opposed to an exponential, even though it is not the correct γ. The γ that fits the implicit code in the early part of curve **c**, part (A), is a factor of 1.75 larger than that for curve **b**, and a factor of 1.65 for the explicit code, curve **c**, part (B).

The departure of the code results from the cosh function at later times is due to all the particles "locking up". That is, they drop into a hexagonal close-packed arrangement, and then the interface and the particles simply oscillate up and down instead of dropping on through as expected. Both the implicit and explicit codes behave the same in this respect. This problem appears to be one of resolution, because explicit cases have been run with increasing number of particles, and a run with 5000 particles in the half-problem finally dropped through, but still bounced up slightly once. The implicit code has only recently

been able to handle 5000 particles, and it will take more work to make this problem run at this resolution. Hopefully the matrix-free method will help this problem, but that will require finding a preconditioner that uses less memory than the sparse matrix does.

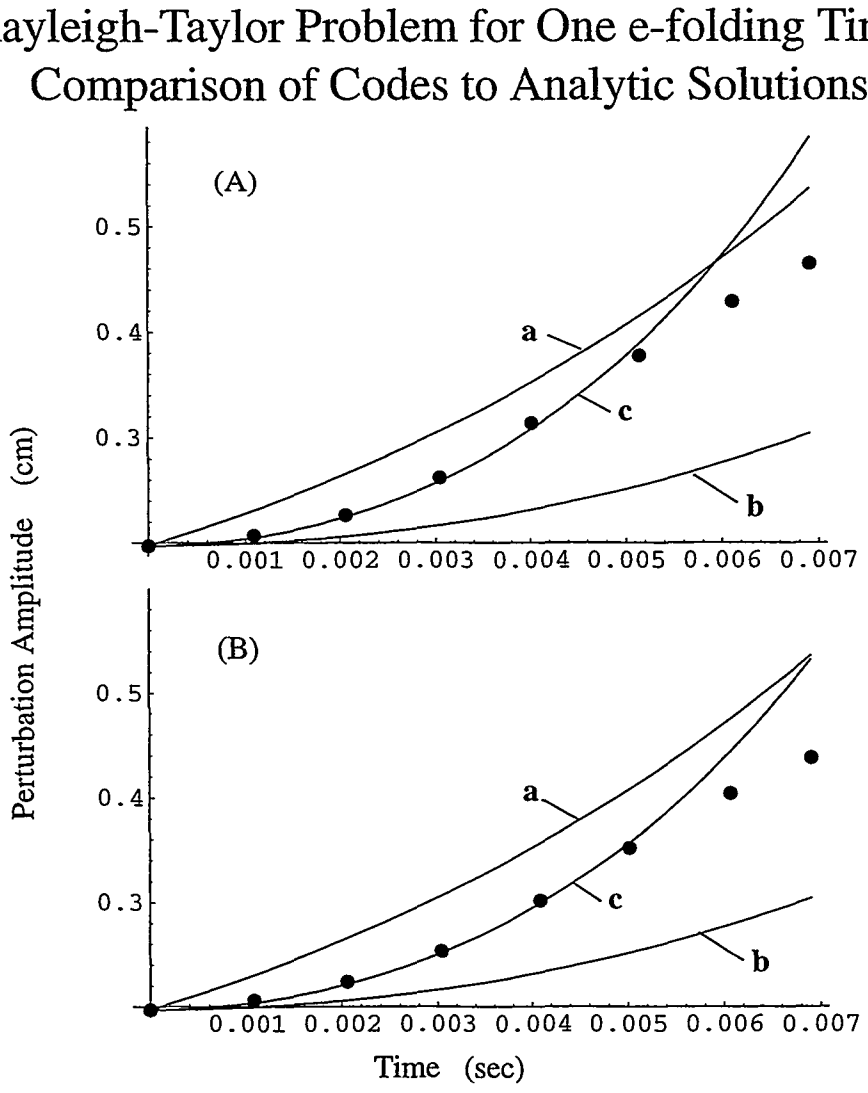


Fig. IV. 5. These plots show the change in the amplitude of the interface as a function of time for a case of 326 particles. The dots indicate the implicit run in part (A), and the explicit run in part (B). In both parts, the curves labeled **a** and **b** are, respectively, the exponential and cosh solutions for $\gamma = 144.7$, and curve **c** is a cosh solution fitted to the code results at early times. In part (A), γ needed to be a factor of 1.75 larger for curve c, and 1.65 larger in part (B).

The "locking up" of the particles does not appear to be a function of the size of the

perturbation. For this setup, the particles lock up at about 10^{-2} sec, independent of the perturbation. Smaller perturbations were tried because it is noted in Hoffman [37] that the growth rate of a sinusoidal perturbation will decrease as the perturbation approaches approximately 10% of the wave length λ, so perturbations of 0.1 and 0.02 cm were tried. It was hoped that, by starting with smaller perturbations, the interface would still drop to about the same depth. However, the hexagonal pattern appeared in about one e-folding time in all cases, and oscillations would start around two e-folding times. These other cases are not illustrated here because the interface tended to become somewhat irregular and not remain as sinusoidal as the perturbation of 0.2 cm did, making it difficult to analyze the interface.

The reason given in Hoffman [37] for the slowing growth rate is that higher harmonics are beginning to grow, and the problem is moving into the nonlinear regime. It may be that the hexagonal pattern of particles does not allow the higher harmonics to grow correctly. Possibly the higher resolution, i. e., increased number of particles, prevents the particles from becoming organized so quickly throughout the problem, so that the explicit code, with 5000 particles, was finally able to drop through.

Figure IV. 6 shows that the explicit code with increased resolution approaches the cosh analytic solution. Curves **a**, **b**, and **c** are the same curves as seen in Fig. IV. 5, part (b), where **a** and **b** are the analytic exponential and cosh solutions for $\gamma = 144.7$ sec^{-1}, and **c** is a cosh fit to the results with 326 particles, which was off by a factor of 1.65. Curve **d** is a cosh fit to the results with 6000 particles, and the γ for **d** is only off by a factor of 1.2

The Rayleigh-Taylor theory assumes the fluids are incompressible. The theory for

SPH assumes compressibility, so to approach incompressibility, the sound speed must be kept much higher than any of the velocities dominant in the problem. For this example, only low-velocity flows are present, and the sound speed is relatively high. Therefore, the implicit code should be able to take very large time-steps in this type of problem compared to the explicit code. The explicit code will be required to take such tiny time-steps to remain stable, because of the Courant condition, that it will take a very long time to run the Rayleigh-Taylor problem out to a few e-folding times. So the implicit code would be expected to have overall lower total run-times. This has not been the case, however.

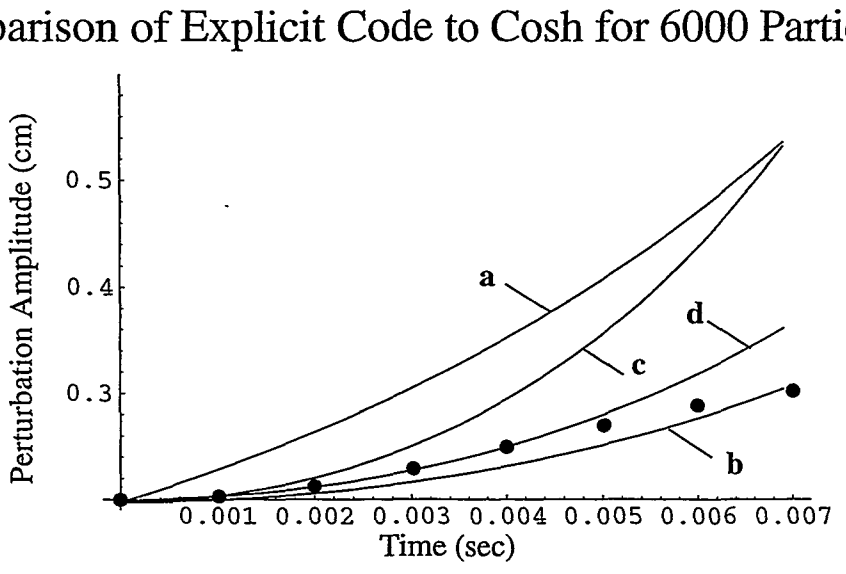


Fig. IV. 6. This graph shows improved agreement, curve **d**, between the explicit code and the analytic cosh solution for 6000 particles. Curves **a** & **b** are the exponential and cosh analytic solutions, and curve **c** is the cosh fit for 326 particles, see Fig. IV. 5,(B).

When larger time-steps are attempted, larger than five to ten times the explicit time-step, a number of problems appear. In some cases hot particles appear and cause explosions among the particles, or particles escape through the reflecting boundaries, or the interface becomes irregular. In some cases the Newton-Raphson nonlinear corrections seem to make things less stable, so that for this problem it appears to be better to turn

it off for the Rayleigh-Taylor problem. The regime being analyzed is linear, so nonlinear corrections should not be needed.

F. Breaking Dam problem

This example, referred to by some as the breaking dam, is also under the influence of gravitational acceleration, but it starts from a non-equilibrium situation. A column of water, initially at rest, is modeled as it collapses and flows out horizontally along a rigid channel. For this calculation the implicit code will be compared to the explicit SPHINX code, the experimental data of Martin and Moyce [53], and the calculational results of Monaghan [61].

In comparing the implicit and explicit codes, the problem is considered to be 2D, and consists of a block of water in the left half of a box, behind a barrier located in the middle, and at time t = 0, the barrier disappears (see the first frame of Fig. IV. 7). The setup includes reflecting boundaries on four sides (a solid boundary, in the SPHINX code, is modeled by placing mirror-image particles across the boundary from the real particles that are close to the boundary, so the real particles "see" themselves, and hence repulse themselves at the boundary). The boundaries are located at the top and bottom at $y = 200$ and -100 cm, and the left and right side at $x = 0$ and 1000 cm. Within the box, the water initially occupies the left side from $x = 0$ to 500 cm and $y = 100$ to -100 cm. The water is initially at rest and has a pressure gradient to balance the forces of gravitation, which is in the negative y-direction. On the right side of the water the pressure is zero, so the water will begin to flow to the right and start dropping due to gravity, forming a wave or surge front. It then flows across the bottom of the box, hitting the right side and splashing up.

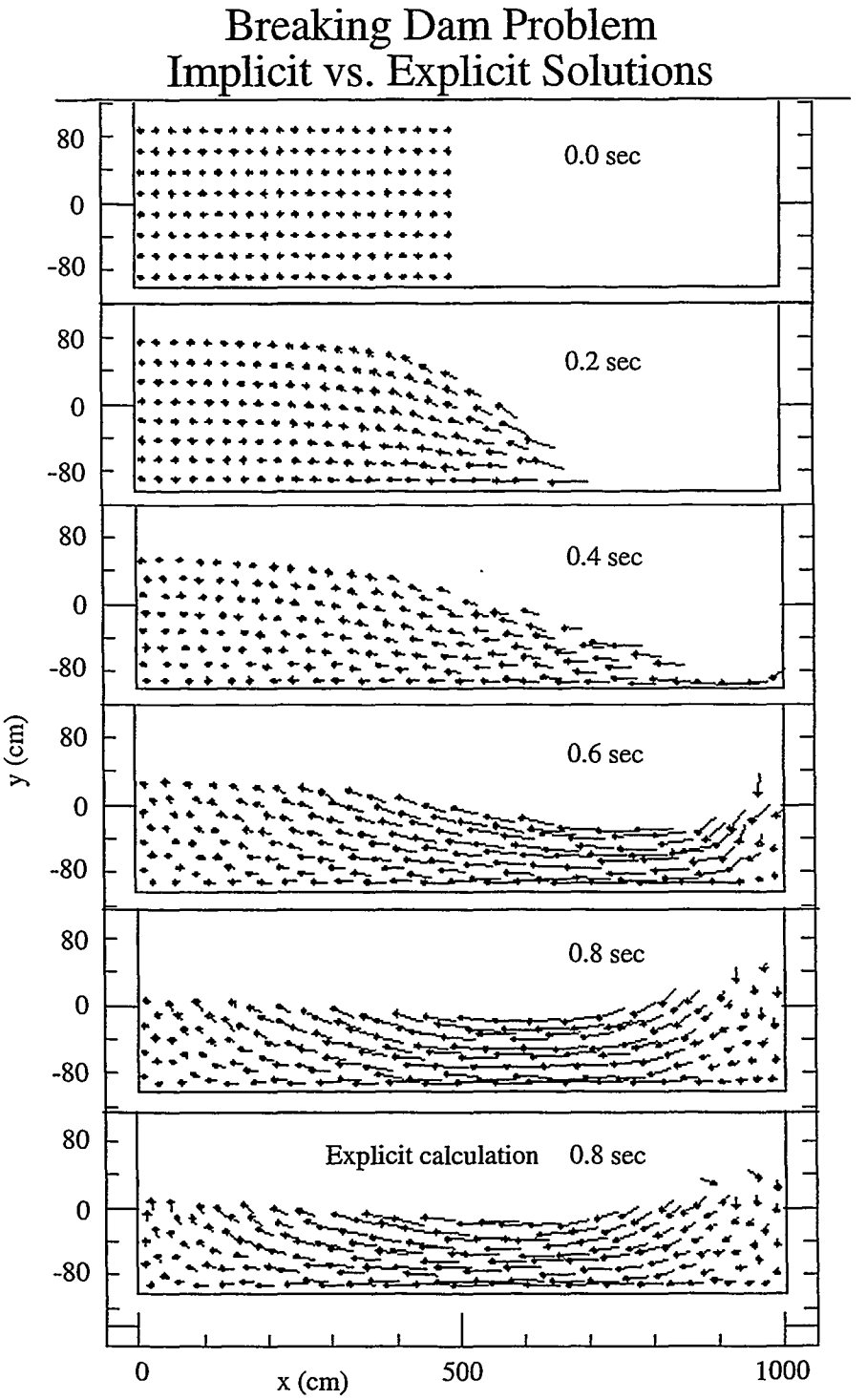


Fig. IV. 7. The top frame shows the initial configuration for the implicit and explicit runs. The middle four frames show a sequence of an implicit calculation. The last frame is the explicit end result for comparison with the last implicit frame at 0.8 seconds. The vectors indicate the direction and relative magnitude of the particle velocities.

The density was set to 1.0, and the EOS model used was a water model developed and installed in SPHINX by Maxwell Labs (see Rice [68], and Pritchett [67]). This EOS model does not have any input constraints such as γ, the ratio of specific heats, or μ, the average molecular weight. The continuity equation was the method used to calculate the density. The gravitational acceleration was set to $g = -10{,}000$ cm/sec^2. The particle smoothing lengths were not allowed to change during the run and were initially set to 1.0, which means they were set equal to the initial distance between the particles.

The example shown in Fig IV. 7 was modeled with 168 particles, and was run out to a total time of 0.8 sec. The time-step multiplier for the implicit code was about 5 times over the explicit time-step. However, it took 12.6 hours to run, whereas the explicit code ran the same problem in only 0.553 hours, or a factor of 23 times faster. The implicit code took 1589 time-steps and the explicit code took 7855 time-steps.

A comparison of the last two frames, which are the last frames from the implicit and explicit runs, show very similar qualitative results. The lead particle, however, obtained a somewhat higher peak velocity just prior to hitting the right wall. It reached a maximum speed of 1950 cm/sec for the implicit run and 1680 cm/sec for the explicit code.

This example can be compared to the experimental data of Martin and Moyce [53]. They performed a series of laboratory experiments with different shaped, water-filled, transparent containers with a film barrier on one side that they would burn away electrically. When the barrier was destroyed, the column of water would begin to collapse and flow along a horizontal transparent channel. They then recorded photographically the time evolution of the surge front and the descending level of water in the container.

Since most fluids obey scaling laws, the results of Martin and Moyce are reported in a scaled and unitless format. They define:

n^2 = the ratio of the column height to the base, which for a square is 1,
a = a characteristic dimension for the base,
z = the surge front position,
η = the column height.

Their scaling formulas, which are unitless, are:

$Z = z/a$, the scaled surge front position,
$H = \eta/an^2$, the scaled column height, and
$T = nt(g/a)^{1/2}$, the scaled time.

To compare with their results, a somewhat different setup was run. For this the block of water was a 200-by-200 cm square of particles. The bottom boundary was extended out to x = 2000 cm, so the particles would not reach the right hand boundary in the total time of 0.8 sec. Hence, for the case presented here: n = 1, a = 200, and $T = t(50)^{1/2}$, where $g = -10{,}000$ cm/sec^2.

Monaghan [61] did a comparison of his SPH code with the results of Martin and Moyce, and got "satisfactory" agreement with the experimental results, after noting the experimental timing error and that there was probably drag between the fluid and the bottom plate. Figure IV. 8 shows a comparison of the implicit code, the explicit code, the experimental results of Martin and Moyce, and the results of Monaghan. It is a plot of surge front position versus time. The triangles mark the experimental result, the squares and stars indicate the explicit and implicit code results respectively, and the diamonds mark Monaghan's code results. Monaghan's results are above the experimental results

implying that the experimental result may have viscosity between the bottom plate and the fluid. The implicit and the explicit SPHINX results are below the experimental results, so they may have more viscosity, and SPHINX does have a default artificial viscosity. which was not adjusted.

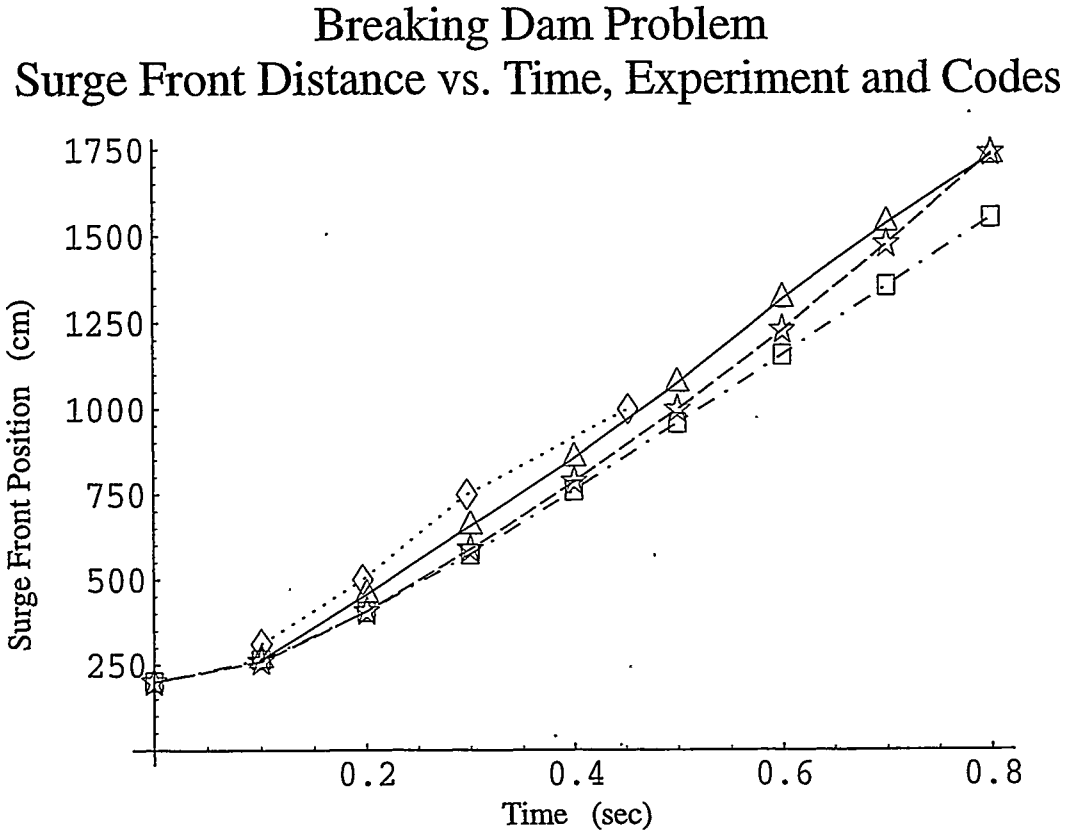


Fig. IV. 8. This plot shows a comparison between the experimental and three different code results. The triangles mark experimental results of Martin & Moyce, the diamonds are code results of Monaghan, and the stars and boxes mark the results of the implicit and explicit codes respectively, which are nearly coincident at early times, and slowly diverge at later times. (See text for details.)

Fig IV. 9 shows curves for the column height versus time of the implicit, explicit, experimental, and Monaghan's results, using the same symbols as in Fig IV. 8. The earliest point of the explicit result is higher than the other curves because the top row of particles actually moved upward in the first few time-steps and then started to fall. The

implicit code did not do this, but at later times does not agree as well with the experiment as the explicit code.

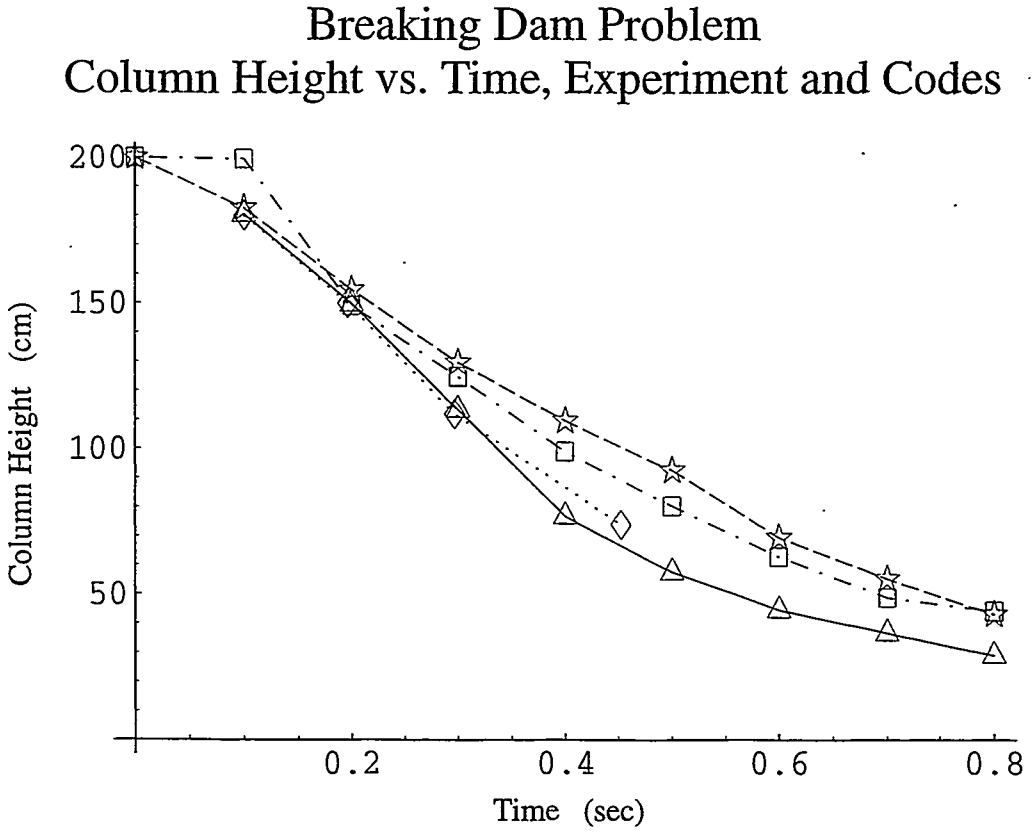


Fig. IV. 9. This plot shows a comparison between the experimental and three different code results. The triangles mark experimental results of Martin & Moyce, the diamonds are code results of Monaghan, and the stars and boxes mark the results of the implicit and explicit codes respectively. The diamonds are nearly coincident with the experimental data, triangles, at early times. (See text for details.)

There was no great effort to get the results shown here. A few different EOSs were tried in the explicit code, but only the default viscosity was used. The agreement between the implicit and explicit codes is good, and it is assumed that as more physics is turned on or adjusted in the explicit code, that better agreement between the explicit code and experiment will be achieved, and that the implicit code will follow since it uses the

same physics packages. Most of the effort was put into trying to get the implicit code to run faster and handle larger numbers of particles, so as to be competitive with the explicit code in some regime.

Attempts to increase the time-step multiplier have been tried. Implicit cases that were run to 0.2 seconds with multipliers of 5 and 10 took 127.4 and 82.42 minutes of wall clock time respectively. The explicit code, running the same problem, took 7.672 minutes, or factors of 17 and 11 times faster, respectively, than the implicit code. So it appears that, to match the speed of the explicit code, a time-step multiplier of at least 20 will have to be attained. To run the problem beyond 0.2 seconds with a multiplier of 10 or greater has led to more spectacular but unphysical particle motions.

G. A Single Jet of Gas

The single jet of gas problem is 2D and consists of a 1 x 1 square centimeter packet of gas with an initial bulk velocity of 1.0e+7 cm/sec in the y-direction and a temperature of 11600 ^{o}K, and using a perfect-gas EOS. The problem is run out to 1 μsec. The gas is free to expand from the internal pressure as it translates along the y-axis. This is a comparison of the explicit and implicit codes only.

The purpose of this exercise was to find a region where the implicit code could excel over the explicit code in total run-time. This was finally achieved in this test case, by going to some extreme and unphysical input values. Also, this problem can be run in just a few minutes, so the many parameters of the implicit code could be optimized fairly quickly. The increase in speed was achieved by relaxing the tolerances for both the Kry-

lov solver and the Newton-Raphson iterations, and increasing the size of the perturbation in the numerical derivatives. These changes not only improved the speed of the implicit code, but also improved the accuracy and robustness of the code. However, with these improvements no effort has been made to go back to rework the previous test cases.

One region where the implicit code can excel is when the gas is incompressible, or nearly so. This implies high sound-speeds, hence short Courant time-steps. The sound speed c is given by $c = (\gamma p/\rho)^{1/2}$, and can be raised by increasing γ (the ratio of specific heats, $\gamma = c_p / c_v$) in the perfect-gas EOS model. It can also be achieved by decreasing the density or increasing the temperature (temperature, or internal energy, appears in the pressure). It is, however, desirable to keep the temperature and particle velocities low while increasing the sound-speed so that the particles do not fly apart too fast; therefore, only γ was varied. So basically one wants to find a region where the explicit code is bogged down by very tiny Courant time-steps but does not affect that of the implicit code.

The implicit set of cases was run using the CGS Krylov solver (tolerance: Ktol = 1.0 e–3), the Newton-Raphson iterations (tolerance: Ntol = 1.0 e–2), with the line-search on, and $\theta = 1.0$. Also the implicit time-steps were calculated so that no two particles were allowed to collide within one time-step. The gas had an initial density of 0.006 gm/cc and was modeled with 400 particles in a square array (see Fig. IV. 10, frame (a.)). The cases also had the continuity equation turned on for the density method, and the smoothing length was allowed to vary.

The values of γ were chosen to be γ = 10, 20, 30, 100, 1e3, 1e4, 1e5, and 1e6. At the value of γ = 1e5 the implicit code performed the calculation faster than the explicit

code, with the explicit code taking 7.0 minutes of cpu time, and the implicit code taking 4.4 cpu minutes. For $\gamma = 1e6$ the explicit code took 88 minutes, and the implicit code took 5.3 minutes (both wall-clock times).

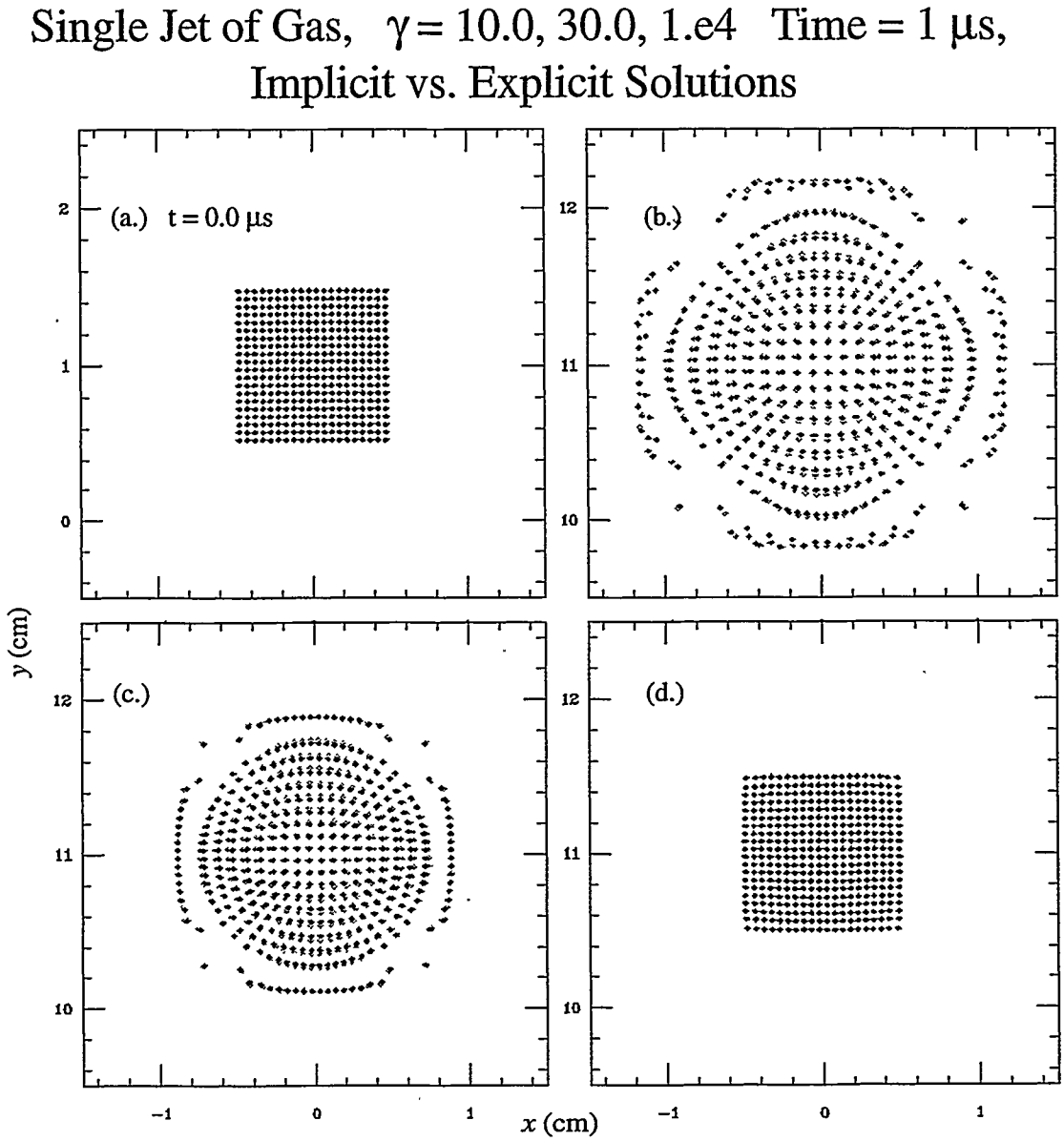


Fig. IV. 10. This figure is a comparison of the implicit (black dots) and explicit (gray dots) codes for the results of the test case of the single jet of gas, for different values of gamma γ. Frame (a.) is the initial setup at $t = 0.0$ μs. The remaining frames are at the time $t = 1.0$ μs, after having translated to the $y = 11$ cm position and expanded. They were run with different values of gamma, (b.) $\gamma = 10.0$, (c.) $\gamma = 30.0$, (d.) $\gamma = 1.e4$.

The results agree fairly well between the implicit and explicit codes for all values of γ, and are illustrated in Fig. IV. 10 for $\gamma = 10.0$, 30.0, and 1e4. The accuracy is even better yet if theta is set to 0.74. As γ increases, there is less and less expansion of the jet during the 1 μsec, and for $\gamma = 1e4$ and greater there is very little difference between the final plots (see frame (d) of Fig, IV.10).

For $\gamma = 1e4$ and greater, the plot for the final problem-time of 1 μsec has hardly changed from the initial setup. The final plot is still a square, slightly expanded. The sides are only slightly bowed out, and the particle at the center of the right edge of the jet has only moved from x = 0.475 cm to 0.497389 cm for the $\gamma = 1e4$ case.

The implicit code took from 32 to 35 time-steps for the different cases, but the explicit code took from 101 to 20,248, that is:

gamma	10	1e3	1e4	1e5	1e6
Sound speed	3.11e6	3.11e7	9.82e7	3.11e8	9.82e8
Number of Time-steps	101	232	996	3818	20,248
R-K cpu run-time	0.45	0.49	1.5	7.0	88.2(wall-clock)

So the sound-speed needs to be in the range of about 1e8 or greater to significantly bog down the explicit code.

These are unphysical values for gamma, but since the sound-speed is c = sqrt(g p/d), there may be physical regimes where the ratio of pressure and density can give similar results. But this is at least a first step.

Precision (as opposed to accuracy) of the code can be demonstrated by the symmetry of this problem. That is, a plot of Fig. IV. 10 can be reflected about its axes of symme-

try at $x = 0$ cm, and the $y = 11$cm line, and superimposed on itself. The particles, ideally, should remain unchanged in shape. But if they do not lie on top of each other, making some particles appear elongated, then there is lack of symmetry and hence a lack of precision. With the tolerance for the Krylov solver set to 1.0 e–3, the final result of this case

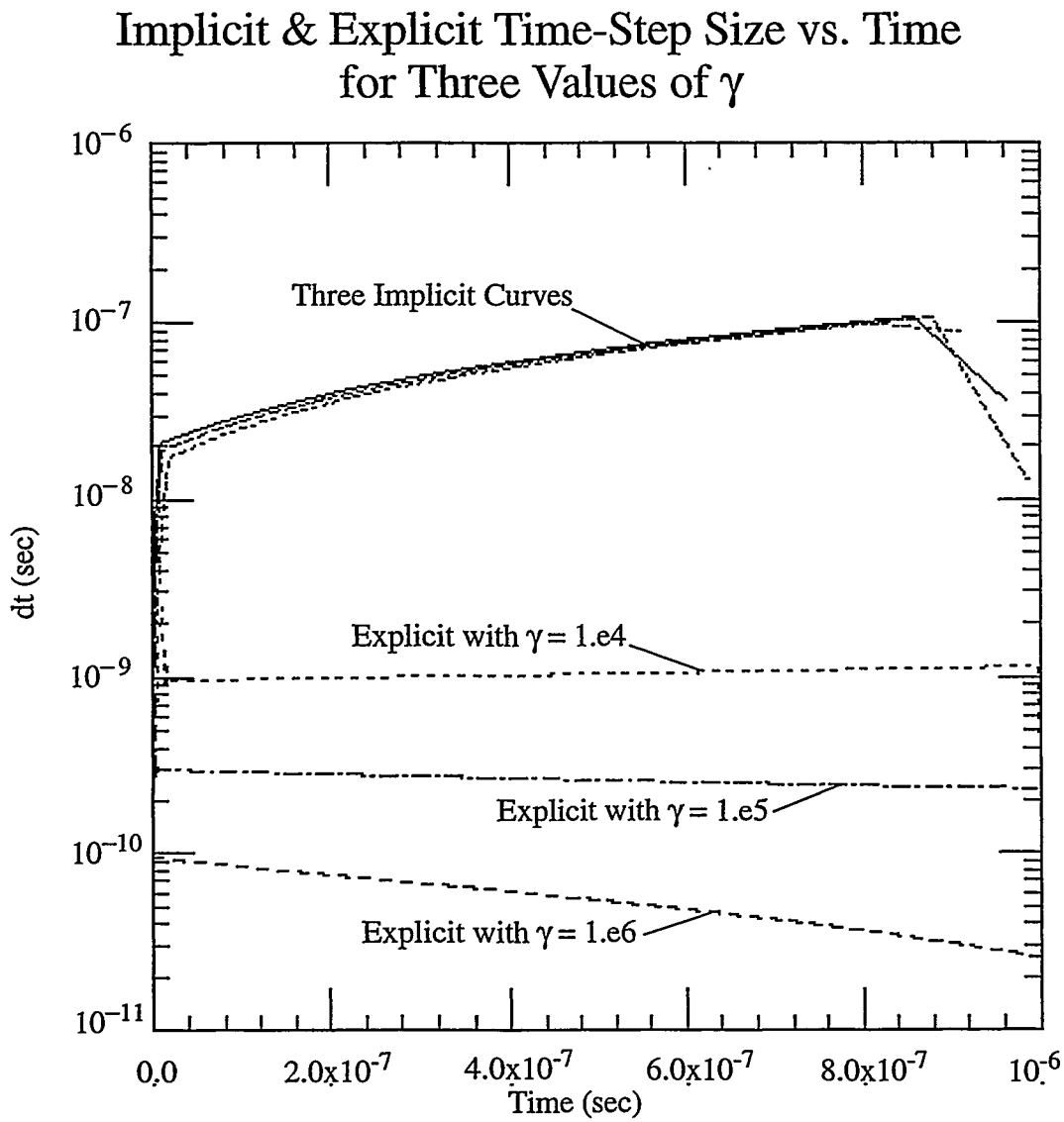


Fig. IV. 11. These are log plots of the time-step size versus time for both the implicit and explicit code for γ = 1.0e4, 1.0e5, and 1.0e6. The three implicit curves are essentially the same so have not been distinguished. The mid-time Courant numbers for the implicit code for these three case are approximately 63, 290, 1200 respectively.

shows excellent symmetry between the final plot and either its reflection about $x = 0.0$ cm or $y = 11.0$ cm. While the reflections are not displayed in the figures, the symmetries are fairly evident in Fig. IV. 10.

Figure IV. 11 shows the time-step size versus time for the three cases of = 1.0e4, 1.0e5, and 1.0e6, for both the implicit and explicit codes. By dividing the maximum implicit time-step by the respective explicit time-step, an approximate Courant number can be obtained, and for the three cases they are 90, 400, 3000 respectively. At the mid-time step sizes the Courant numbers are approximately 60, 290, 1200 respectively. The real saving in time would come if one wanted to run the $y = 1.0e6$ out to several seconds instead of a microsecond. It would take the explicit code at least a 1000 times longer to run this problem. At the end of the 1 μsec calculation, the time-step size was still increasing for the implicit code and was still decreased for the explicit code.

H. Comments and lessons learned

There are many choices to be made when running the SPHINX code and most of them affect the solutions when running the implicit code. In addition to those choices there are many to be made that are unique to the implicit code, such choices as which Krylov solver to use and which preconditioner to use to solve the linear problem. The three Krylov solvers that were tried all seem to do equally well on the type of SPH matrices generated by the current implicit code, with CGS running slightly faster than the other two. The preconditioner used for all of the test cases is the multipass Jacobi, and two passes seemed to be optimal. There is also the choice of whether or not to use the New-

ton-Raphson iterations and the line-search to solve the nonlinear problem.

There are also many limits and tolerances to be adjusted that affect the speed and accuracy. The following is a list of items and current values that can be varied to change the performance of the implicit code (some values are based on double precision):

Use either CGS, BiCGSTAB, or GMRES.		
theta	–> Degree of implicitness. (optimum: ~.74)	0.5 to 1.0
frac	–> Perturbation for numerical Jacobians.	1.e–8
Ktol	–> Tolerance for Krylov iteration convergence.	1.e–3
Ntol	–> Tolerance for Newton iteration convergence.	1.e–2
Kmaxiter	–> Maximum Krylov iterations allowed.	100
Nmaxiter	–> Max Newton-Raphson (NR) iterations allowed.	15
dtmultmax	–> Maximum value of dt multiplier.	15 to 1e5
slope	–> Factor of increase of dtmult up to the max.	1.1
RKsteps	–> Number of explicit steps for start-up.	1
dT*f	–> dT: Min time for any 2 particles to hit * factor.	f=1/4
dt*f : Factor of decrease of dt if Krylov or NR fail to converge.		f=3/4

The method of optimizing the list of parameters has been to go down the list and vary only one at a time until the run-time is minimum, and/or the agreement with the explicit code is best, and/or the symmetry of the final plot is best. This method may not be the most systematic one to follow, since some of the parameters are linked to each other. So the next step was to repeat the procedure for at least parts of the list. This process has been applied mainly to the three 1D problems and the single-jet problem. The optimal set of values for the single-jet problem seems to change somewhat with the number of particles. The more particles, the harder it is to keep the problem stable, so the tolerances needed to be tightened up. It may be that the set of values for this list may change with each problem too. So keeping the code robust with different problems is another cri-

terion for choosing values for the list of variables. The most useful set of values may not be optimum for a particular problem, but it will allow the code to run reliably for most problems.

Obtaining the minimum run-time is a matter of keeping the Krylov and Newton iterations to a minimum and avoiding the use of the line-search routine. Experience with the code has shown that the parameter with the most influence on this is the slope of the initial ramping up of the time-step size. If it increases too fast, the Krylov solver may not converge, which will trip the routine that reduces the time-step by some factor. Even if the Krylov solver is converging very well, the Newton iterations may not converge, which will also trip the routine that reduces the time-step. Allowing too many iterations is wasteful of time if either the Krylov or Newton iterations are not converging, but stopping too soon, if it is close to convergence, will also waste time because it will go to a smaller time-step and start the time-step process all over again. Varying the tolerance for the Krylov solver does not have much effect on the accuracy of the problem. It has more effect on the speed. It is better to get an approximate solution from the Krylov or linear solver, and let the Newton iterations or nonlinear solver maintain the accuracy. Typically, when the calculations are going smoothly, there should be 1-to-3 Krylov iterations within each of 1- to-3 Newton iterations. The other parameters that affect the speed of the code are RKsteps, Ktol, dtmultmax, dT*f, dt*f, Kmaxiter, and Nmaxiter.

Accuracy and precision are both influenced mainly by the time-step size, which is controlled by the slope or the ramp-up rate. The slope has been found to affect the accuracy significantly, but in the opposite direction of the speed. The slope should be more

gentle to increase accuracy. Other variables that affect the accuracy are RKsteps, and theta. Theta is the only one that does not affect the speed significantly. The time-step size is controlled by the same things that control the speed of the code, so the accuracy and the speed are intimately interlinked.

Chapter V

Application to a Fusion Problem

A. Introduction

The series of calculations discussed in this chapter was done in support of a newly proposed Magnetized Target Fusion (MTF) concept (Thio, C. F. et al., 1998 [84], [85]) and the results were presented at the 1998 Innovative Confinement Concepts Workshop, at Princeton, NJ, April 6-9, and at ICOPS '98, Raleigh, NC. The results presented at ICOPS '98 were done with the explicit code. In the present discussion some comparison of the implicit and explicit code is presented.

It was hoped that the implicit code could be shown to model parts of the computations of a practical problem faster than the explicit code. This has not happened yet, but this application has demonstrated that the two codes are in good agreement in 2D and 3D. A regime where the implicit code has functioned faster than the explicit code was discussed in Chapter IV, Section G, but it came too late for this ability to be expanded into a practical problem. The regime was found after this dissertation was essentially written.

The new MTF concept consists of sixty or more neutral plasma jets in a spherical arrangement, imploding one or two small magnetized toroidal plasma targets. The plasma jets and targets are to consist of deuterium and tritium (D -T), and since they are in a neutral plasma state, they can be approximately modeled using hydrodynamic methods. The jets, as envisioned, will merge together and form a spherically imploding piston or liner. Once the target ignites, the liner will supply fuel for the imploded target to burn. The jets are accelerated from plasma guns by a heavier ionized gas, possibly argon. The

heavier argon would also act as a tamper during implosion, and an absorber for x-rays to help protect the chamber and plasma guns from x-ray damage. Another layer of plasma could be introduced behind the argon, such as lithium or boron to absorb neutrons.

Each target can be thought of as the plasma equivalent of a smoke ring confined by a stable toroidal magnetic field. It is well understood how to produce the plasma rings from specially designed plasma guns, and the toroidal configuration of the magnetic field in the targets is known to be stable [89]. The magnetic pressures will be weak compared to the hydrodynamic forces; hence, for the present calculations the target is replaced with a sphere or cylinder of neutral gases, and the magnetic field is ignored.

It was hoped that the implicit code would be able to perform the calculations up to the time of impact of the jets with the target including the merging of the jets. This, however, has not been demonstrated yet. The implicit code has been run on all three cases discussed in this chapter.

Magnetized Target Fusion is a cross between Magnetic Fusion Energy (MFE), which uses a magnetically confined D-T plasma (for example the Tokamak), and Inertially Confined Fusion (ICF), in which a D-T target is imploded by laser or particle beams. MTF starts with a magnetized target and implodes it with any one of a number of means. The magnetized target is the important difference, because the magnetic fields of the target confine the ions and electrons, reducing the loss of energy from the target. It has been shown that such a target does not need as large a driver to initiate fusion as ICF requires [42], [50], [74]. The targets will be much larger and will require a less energetic driver.

The explicit SPHINX code was used for all of these calculations, and a few of the

simpler calculations were repeated with the implicit code. The calculations are preliminary and address only the hydrodynamic aspects of the problem. The study was mainly concerned with the merger of the jets and the formation of the imploding piston. Fusion was not considered in detail, except that the temperatures and confinement durations for ignition were sought within a few attempts, but no optimization was performed.

The following assumptions were made for these calculations. The D -T plasma jets are considered neutral, so that no magnetic or electric fields are present. Losses due to radiation and thermal convection were not included. It is assumed that radiation losses will be insignificant up to the time of maximum compression. The argon driver plasma might not be neutral, so it might have electric and magnetic fields in it, and hence was not modeled in the present set of calculations.

The first set of calculations considered a simple case of two jets merging at an angle and then accelerating a solid aluminum projectile. The next set of problems consisted of a cylindrical ring of jets aimed at the origin and a target of D -T. The third set of calculations modeled the 3D spherical arrangement of jets, and consisted of 60 jets in a soccer ball configuration, similar to the target chamber of the Omega laser fusion system at the University of Rochester, N.Y.. A soccer ball is covered with an arrangement of hexagons and pentagons, and the jets were placed at each of the sixty vertices.

B. Neutral Plasma Jets Merging onto a Projectile

The goals of this first set of calculations were (1) to see how the neutral plasma jets merge together, and (2) to find to what maximum velocity a small projectile can be acceler-

ated using plasma jets of different masses and velocities. The momentum was varied but not the energy, to see if this affected the coupling of the energy into the target. The acceleration of projectiles to high velocities is of interest to the weapons community. The gas for the jets in this case is not D -T but a more massive gas. The problem was done with two jets, in 2D, using the explicit and implicit codes. The initial setup is shown in Fig. V. 1.

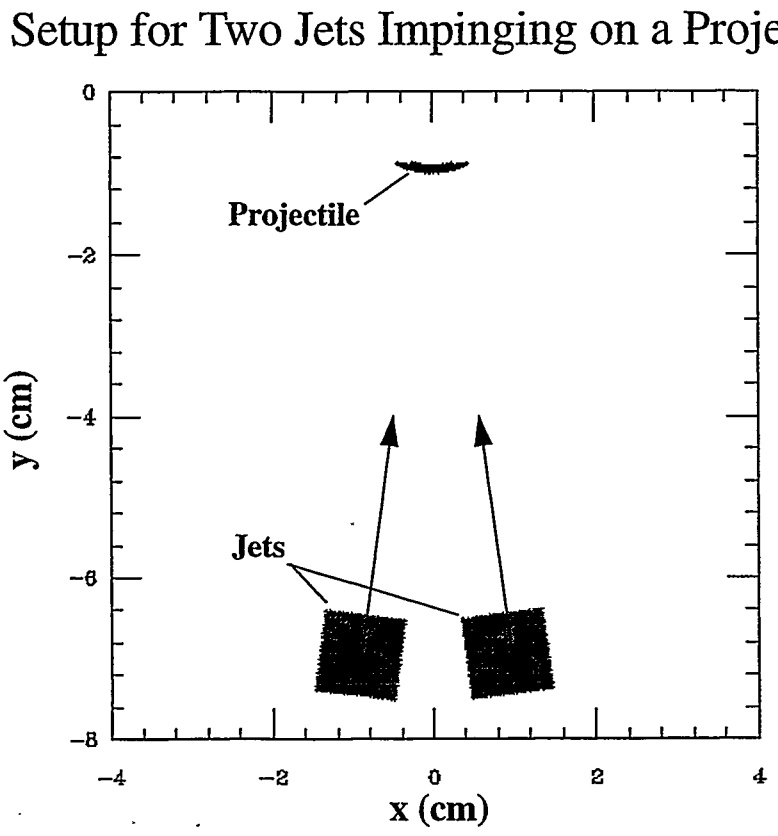


Fig. V. 1. This is the initial setup for two plasma jets impinging on an aluminum projectile. The jets are aimed at the origin, and the target is placed ahead of the origin.

The initial setup consisted of two jets, 15 degrees apart, and aimed at the origin, which was behind a stationary aluminum projectile. As the jets move toward the projectile, they expand due to internal pressure. On the way to the projectile, the jets collide with each other, forming a more dense core between them, and the squeezing of the core

produces what will be referred to here as a "super-jet", where the particles at the leading edge of the core are accelerated forward toward the origin and those at the rear of the core are accelerated backward. In some cases the super-jets were found to cut the projectile into pieces. This problem can be lessened by starting the jets farther away from the projectile thus allowing the super-jet to spread out before striking the target.

The velocities v and masses m of the jets were varied so as to increase the momentum but maintain constant energy between case studies. If the mass of the jets is increased by a factor of four and the velocity decreased by a factor of two, the momentum (mv) will be doubled while the energy $1/2(mv^2)$ is kept constant between cases. The question then is: can the energy of the jets be transferred more efficiently to the target by increasing the momentum of the jets while maintaining constant energy?

Three cases were run: (1) with the density of the jets set to 0.006 gm/cc, and a bulk velocity of 100 km/sec, (2) with a density of 0.024 gm/cc and a velocity of 50 km/sec, and (3) with a density of 0.096 gm/cc and a velocity of 25 km/sec. The jets were modeled as a perfect gas with $\gamma = 1.4$, the ratio of specific heats, and $\mu = 1$, the average molecular weight. The initial temperature of the jets was set to 11,600 degrees Kelvin (1 ev). Each jet was represented by a 1x1 cm square rectangle of particles. There were 900 particles in each jet and 66 in the projectile. The projectile was aluminum using a linear_u_s_u_p material-strength model, and the input for aluminum included: a solid density of 2.7 gm/cc, a sound speed of $c_o = 5.376e5$ cm/sec, a slope of $s = 1.55$, $\gamma_0 = 2.1$, and $\gamma_1 = 0.0$.

Preliminary computer runs showed that the distance from the leading edge of the jets to the projectile should be about 7 to 5 cm, so that there was not too much overall

expansion in transit, and the super-jets had a chance to spread out. It was necessary to shorten the distance for the runs with slower velocities, so that they would all have about the same transit time; otherwise the expansion would become too great.

An inward velocity can be added to each particle in a jet, as a function of its distance from the axis of the jet, to counter the expansion due to the internal pressure, which would approximately model the inward focusing that a slight taper inside the plasma gun might impart to the jet. It was found by trial and error that the best range of focusing velocities was for the inward component of the velocity at the edge of the jet to be an order of magnitude less than the bulk velocity. This feature was used only in this two-jet problem, where the emphasis was on getting the most particles to strike the projectile. It was not used in the implosion calculations because there the jets will confine each other.

The first case, with the density of 0.006 gm/cc and the velocities of 100 km/sec, was set up with the Al projectile positioned −1.0 cm from the origin and the leading edge of the jet started at a position of −6.5 cm from the origin, so that the jets were located 5.5 cm from the projectile. A focusing velocity, of 10 km/sec, was added to the jets.

The jets first started to merge when they were about 3.4 cm from the projectile at time $t = 0.202$ μsec, and the pressure, temperature, and density between the jets began to increase. The two jets formed a butterfly shaped pattern (see Fig. V.2) with a denser core between the jets and less dense wings out to the sides. Then at time $t = 0.460$ μsec the jets first impacted the projectile. At the time of impact the wing pattern was about the same width as the projectile, so that most of the jet particles contributed to the impact. The final velocity of the projectile was about 5.9 km/sec as obtained with the explicit code.

Comparison of the Implicit and Explicit Codes at $t = 4.5$ μsec, for Two Jets Merging

(a)

(b)

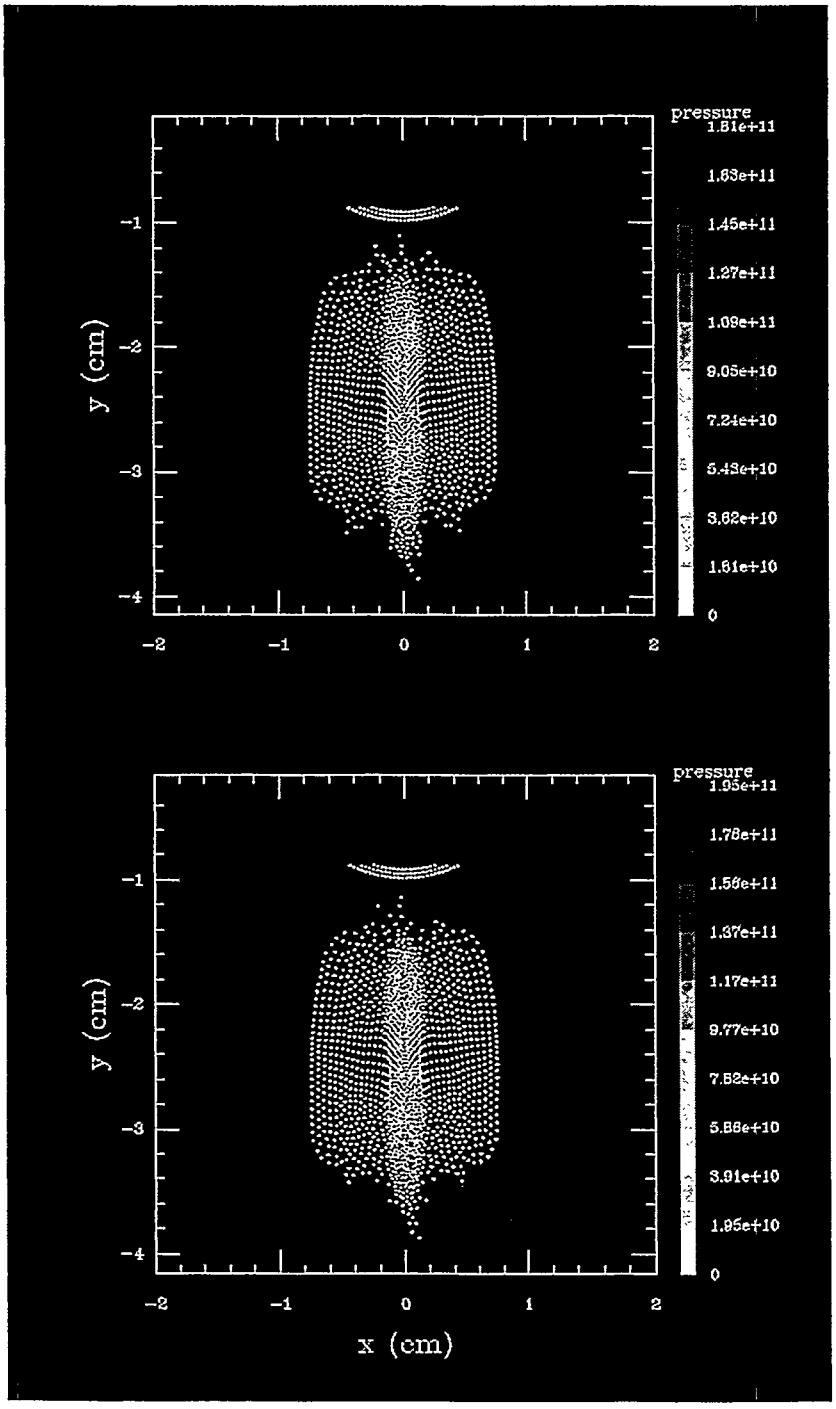


Fig. V. 2 This is a comparison of the implicit (a) and explicit (b) codes for two merged jets at 4.5436 and 4.5227 μsec respectively. Both codes had 1866 particles. An overlay of the two plots shows that the particle positions, in general, are in very good agreement.

This case was also run using the implicit code. An overlay of the two plots of Fig. V. 2 would show that the particle positions are in very good agreement except for a few particles at the leading and trailing edges. Unfortunately the implicit code stopped running at the point of impact, so there is no comparison for the projectile motions.

The jets were initially displaced relative to each other in the y-direction by half a smoothing length, which is why the results are not exactly symmetric. This small displacement was made to see if there would be a mixing of particles. If they were left completely symmetrical, one jet would be the mirror of the other, and there would be no mixing. Even with the offset there was very little mixing except within the super-jets.

The second case, with the density of 0.024 gm/cc, was set up with the projectile positioned as before, but the jets were set only 5.0 cm from the projectile, to accommodate the slower velocities. Since the jet velocities were reduced by a factor of two, to 50 km/sec, the focusing velocities had to be reduced by the same factor of two so that they were still an order of magnitude less than the jet velocities.

The jets first interacted at about $t = 0.15$ μsec when they were about 5.2 cm from the projectile. At about $t = 0.7$ μsec the jets impacted the projectile. At the time of impact the denser core between the jets had expanded to engulf the wings, but the width of the jets was about the same as the projectile, so that again most of the particles contributed to the impact. The projectile was separated into several chunks and several individual particles. The final velocities ranged from 3.7 to 7.9 km/sec for the individual particles whereas the main chunks had velocities of 5.3, 5.9, and 7.1 km/sec.

The third case, with the density of 0.096 gm/cc and a velocity of 25 km/sec, was

set up with the projectile also at −1.0 cm from the origin, and a 5.0 cm separation between the jets and the projectile. For this case the jets were moving too slowly for the focusing to be of any use, because the reduced focusing velocity in x was quickly overwhelmed by the pressure within expanding jets. The wings expanded too fast to be of much use, so the distance was chosen to adjust the core width to fit the projectile.

The jets started to interact at about 0.25 μsec at a distance of about 5.2 cm from the projectile. For this case the core had expanded to a little more than the width of the projectile and the wings had expanded quite a bit more so that they missed the projectile. In this case, approximately half of the particles hit the projectile, so the momentum was inefficiently transferred to the projectile. The projectile was cut in two, and a few individual particles were scattered around. The velocity of the bulk of the particles was roughly 4.5 km/sec. Variations on this case could be tried in which the focusing velocity could be increased or the distance between the jets and the projectile could be decreased.

Drawing a conclusion from this, it appears that there are other aspects to consider when trying to go to higher momentums. The higher momentums seem to cut the projectile up. Also the slower velocities allow the jets to expand more in transit to the projectile making it more difficult to get all of each jet to hit the projectile, and an added focusing velocity has less effect on keeping the jet together because it is more quickly overwhelmed by the pressure. It appears that the high velocity of the lower momentum case helped more by getting the jets to the projectile before they had a chance to expand. So there is a trade off between the higher velocities and the higher momentum and distances.

C. The 2D Ring of Jets

This set of calculations was performed to learn how uniform an imploding plasma piston ring can be made and how long it can confine the target particles. It was also learned what densities and temperatures could be achieved, from a hydrodynamic stand-point, for the implosion of a given target. The initial setup is shown in Fig. V. 3.

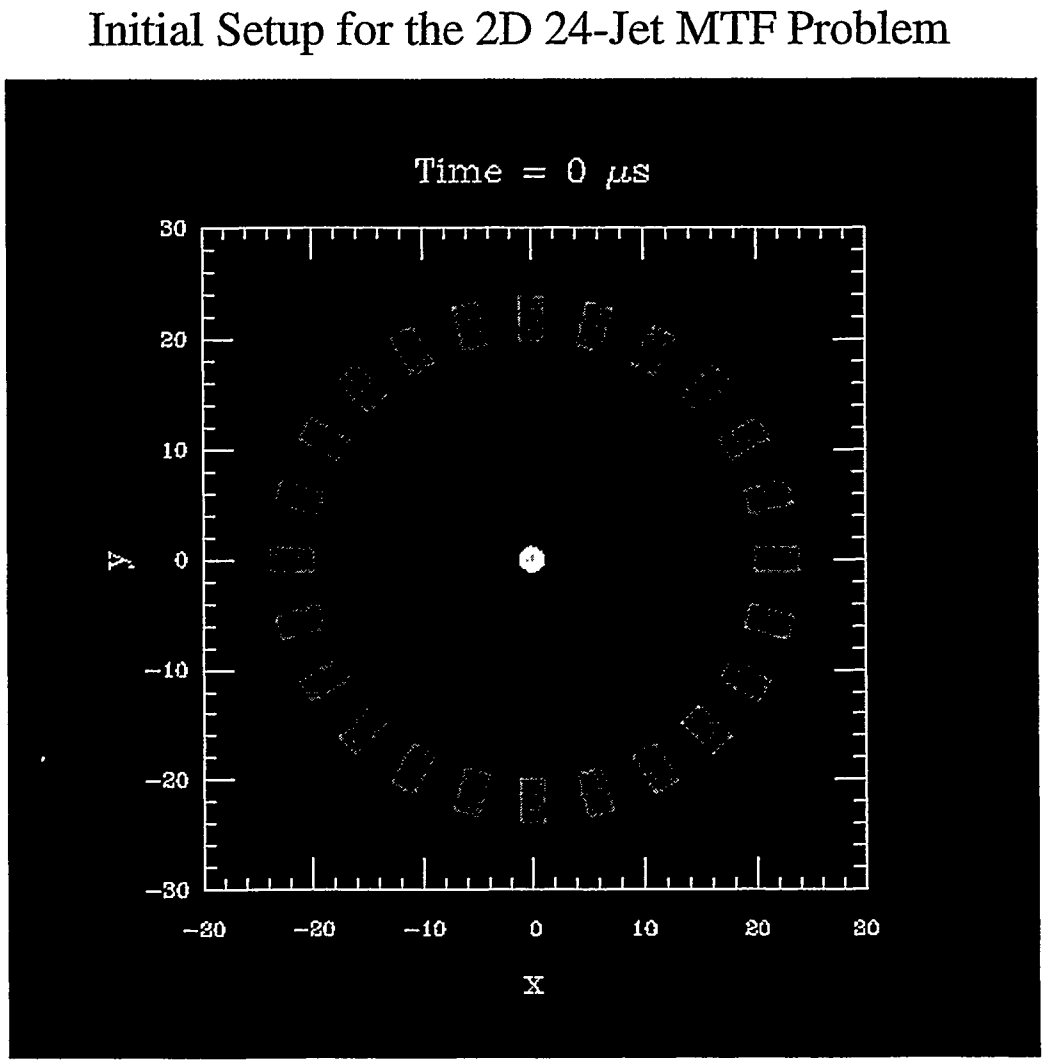


Fig. V. 3. This is the initial setup for the 24-jet case in 2D. The jets are about 20 cm from the target. Only one quadrant was calculated and then reflected around into the other three quadrants.

These calculations were done in 2D, and considered 24 jets in a ring positioned at every 15 degrees, and 20 cm from the origin. Each jet was 1 cm in radius and 3.8 cm in length The jets are assumed to exit the plasma guns at an initial velocity of $5x10^7$ cm/sec with a density of 10^{-5} gm/cc, and all aimed at the origin. They were each given an initial temperature of 11,600 ^oK (or 1 ev). The target was modeled as a cylinder of radius 1cm, and given an initial temperature of 23,200 ^oK. The case discussed here used the perfect gas EOS. The total number of particles used was 12,322, with 2,016 per jet, and 226 in the quarter target for runs using the explicit code.

Because of the symmetry of the problem, only one quadrant of the problem was run with reflecting boundaries along the x- and y-axes. Taking advantage of the symmetry allows one to save on memory and computational time or to increase the resolution.

When the jets start merging with each other, they form the super-jets as discussed in the previous section on two-jets. An earlier attempt had only 12 jets positioned at every 30 degrees and closer to the target, but that did not allow the super-jets to spread out into a uniform front before hitting the target, and the super-jets tended to cut the target up. So the jets were moved back to 20 cm from the target and the number of jets increased to 24. This arrangement allowed the super-jets to merge farther away from the target and form a more uniform front by the time they reached the target.

Figure V. 4 shows the system at maximum compression. The initial contact of the super-jets with the target caused some fluting on the target surface. But the growth of the fluting ceased when the main part of the jets arrived, and then all the target particles were swept up and compressed. The fluting caused the filigree pattern around the compressed

target in Fig. V. 4. The jets have merged together to form an imploding cylindrical piston (this is also referred to as a liner in some literature) that is the uniform circular region around the target. The bulk of the jet particles are still streaming in and are stagnating in the piston, which is confining the target. Each particle of Fig. V. 4 has a velocity vector attached to it, hence the radial lines on the jet particles that are still imploding.

Maximum Compression for a 2D 24-Jet Case, $t = 0.406$ μs

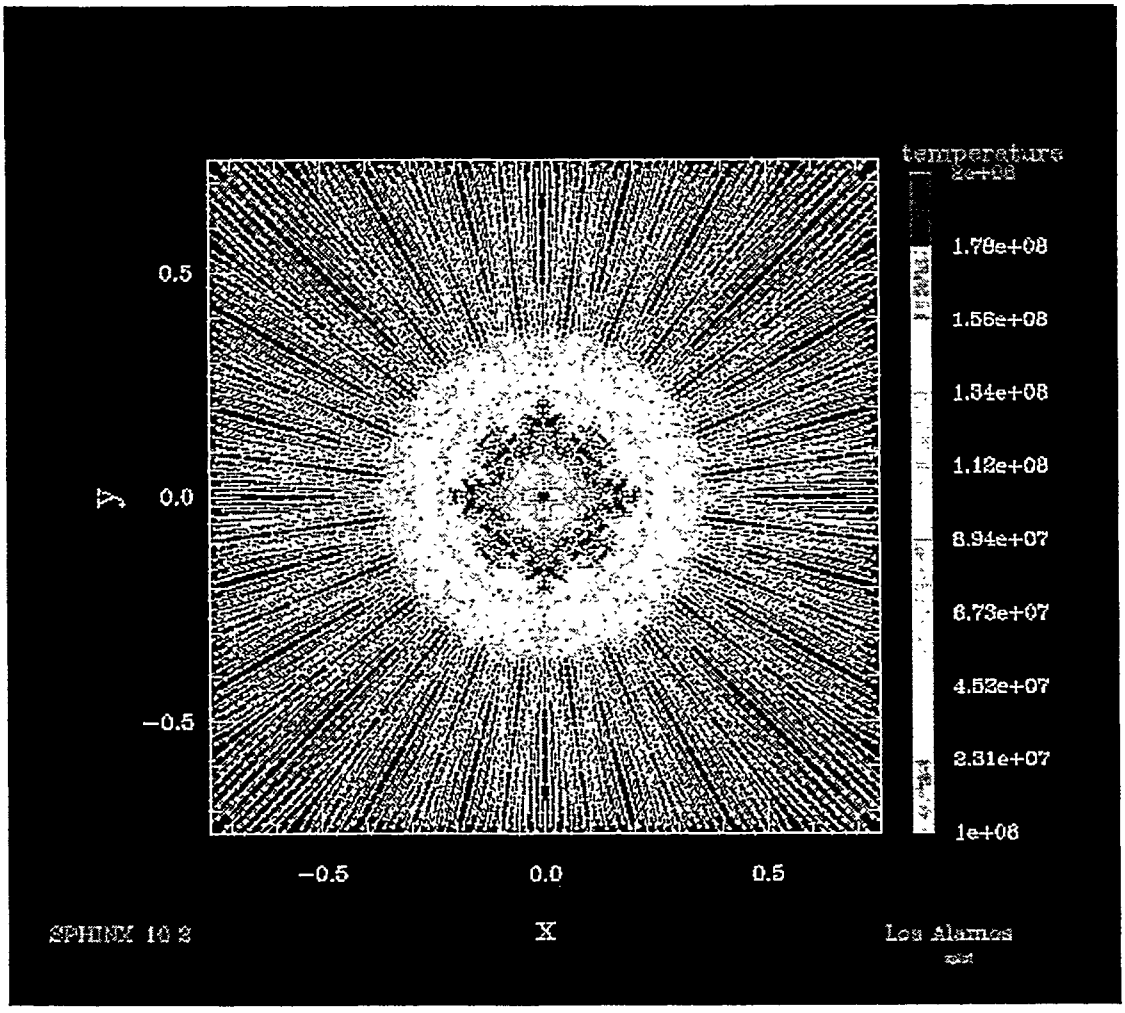


Fig. V. 4. This figure shows the 24-jet case at maximum compression. The target is the irregularly shaped pattern in the middle, of radius ~0.3 cm. The piston is forming around the target uniformly and is the circular region around it. The jets are still streaming in and have a velocity vector attached to each particle, hence the radial rays.

The uniformity of density and shape of the piston met expectations. Most of the particles in the piston, from time $t = 0.4$ μsec to $t = 0.45$ μsec, fall within a density range of 0.0002 gm/cc and 0.002 gm/cc. The pressure for most of the piston particles is in a range of about 2e11 to 2e12 dynes/cm The temperature was within a range of 6e6 to 5e7 °K. The trailing edge of the jets impacted the piston at about $t = 0.45$ μsec.

The target diameter, except for a few target particles, reached a minimum at $t =$ 0.406 μsec with a diameter of about 0.3 cm, stayed there until about $t = 0.45$ μsec, and was under 0.4cm until about 0.47 μsec. At this point it started to expand more rapidly but was still within a diameter of 0.6cm at $t = 0.5$ μsec. The duration of tight confinement of the target was about 70 nsec. The target material was compressed by a factor of about 37 times. Most of the target particles fell within a temperature band of 6e7 to 2e8 °K, and the pressure was between about 5e11 to 1.1e12 dynes/cm^2 during maximum compression.

The following results were obtained from a calculation around the maximum compression of the target.

Piston particles:		
Max temperature	T = 1.18e8 °K	at $t = 0.408009$ μsec.
Max pressure	P = 1.07e13	at $t = 0.412022$ μsec.
Max density	D = 0.00589 gm/cc	at $t = 0.412022$ μsec.
Target particles:		
Max temperature	T = 3.15e8 °K	at $t = 0.408009$ μsec.
Max pressure	P = 2.02e12	at $t = 0.408009$ μsec.
Max density	D = 0.000502 gm/cc	at $t = 0.406022$ μsec.

A few implicit runs were made repeating the explicit case discussed above to see how the implicit code would compare. It was hoped that the implicit code could compute

the early portion of the problem where the jets are running in and merging. However, the explicit code does this portion so quickly that it is difficult to beat it. The implicit code was run with 1,101 particles, 180 per jet, and 21 in the quarter target. It was run to a point close to where the particles first contacted the target. The implicit code's best wall clock time was about 28 minutes, whereas the explicit code took about a minute. An overlay of the results shows that agreement between the two codes is very good. If the implicit code is run through maximum compression of the target, it runs best at about the same time-step size as the explicit code. Both codes compress the target to about the same size and they both produce about the same size piston.

D. The 3D Sphere of 60 jets

For the 3D case, 60 D -T neutral-plasma jets, placed in a soccer ball arrangement of hexagons and pentagons, were all aimed at the origin, where a spherical D -T target was positioned (see Fig. V.5). The jets then formed a spherically imploding piston. The sides of the pentagons and hexagons are all equal length, and the distances from all vertices to the origin are equal. Therefore the angle between any two adjacent jets, on one edge of either a pentagon or hexagon, are all equal, because they all form equal triangles with the origin. However, the angle between the two jets across a pentagon or hexagon is different. Therefore, once the jets have merged, looking along a normal to the center of a pentagon or hexagon toward the origin, one will see a decrease in density of the spherical piston, as opposed to looking radially through one of the vertices. This, however, does not appear to affect the confinement of the target significantly for the duration of maximum implosion.

The Initial Setup for the 3D 60-Jet MTF Concept

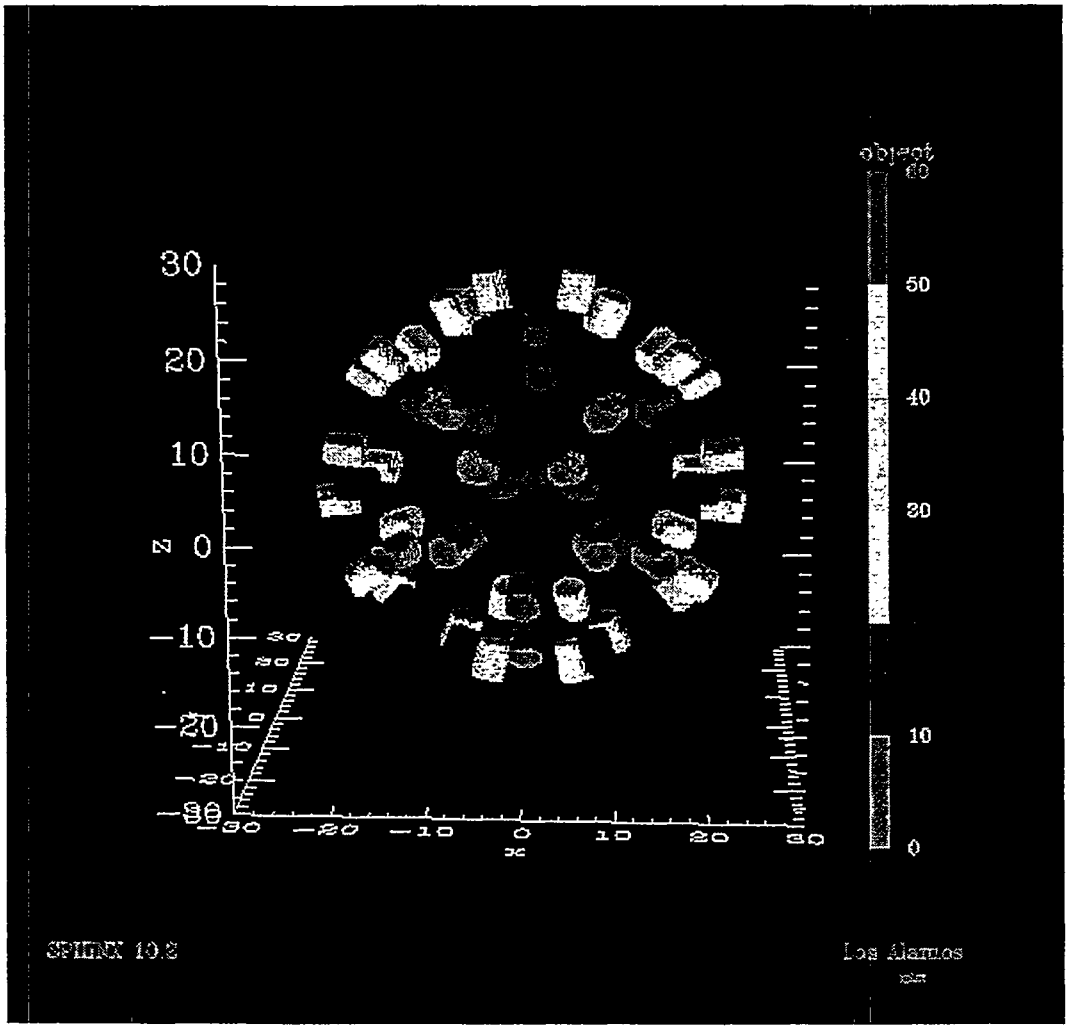


Fig. V. 5. The 60 cylindrical jets are arranged in a soccer ball pattern about a spherical target in the center. The jets are shaded according to their object number to help them show up against each other.

Initially the jets were made the same size as that used in the 24-jet case, 1cm in radius, but they started interacting only after they got fairly close to the target. So the jets were made wider, 1.5 cm in radius, so that they would start to interact farther away from the target. The super-jetting would, therefore, have time to spread out before striking the target. In three dimensions, the super-jets became sheets, interacting at different angles.

The sheets formed perpendicular to the sides of the hexagons and pentagons. The sheets interacted when they reached the center of a pentagon or a hexagon.

The total number of particles was 43,368, with 720 in each jet, and 168 in the target, using the explicit SPHINX code. All jets were modeled, i.e., no reflecting boundaries were used. The vertex positions were taken from a table generated for the Omega laser fusion target chamber. There is a plane of symmetry, but it does not coincide with either the x-y, x-z, or y-z plane. Since time was limited, it was simpler just to model all 60 jets rather than rotate all the vertex positions to an appropriate position to make the plane of symmetry coincide with one of the principal planes.

The initial conditions for the 60 cylindrical jets and a spherical target are specified in the following:

3D Case, 60-Jet Soccer Ball Pattern Imploding a Target Initial Conditions

(43,368 particles total: explicit code)
(5,768 particles total: implicit code)

D-T Jets

60 Jets in a soccer ball pattern, 20 cm from the Target.

Temperature:	11600 °K (1 ev).
Density:	2.0e–5 gm/cc.
Velocity:	5.0e7 cm/sec.
Dimension:	Cylinder, 1.7cm radius x 3.8cm length.
EOS:	Perfect Gas.

D-T Target

Temperature:	23200 °K (2 ev).
Density:	1.5e–5 gm/cc.
Dimension:	Sphere, 2.5 cm radius.
EOS:	Perfect Gas.

The results of the explicit code at $t = 0.4$ μsec are illustrated in Fig. V. 6. The par-

ticles shown in the figure are near maximum compression, however, only jet particles in a slab 0.25 cm on either side of the y-z plane are shown. The target particles have been removed from the plot to help illustrate the formation of the piston. The jets are coming from various angles, so the slab takes a slice through only a few of the jets and at different angles.

Maximum Compression for the 3D 60-Jet Case, $t = 0.4$ μs

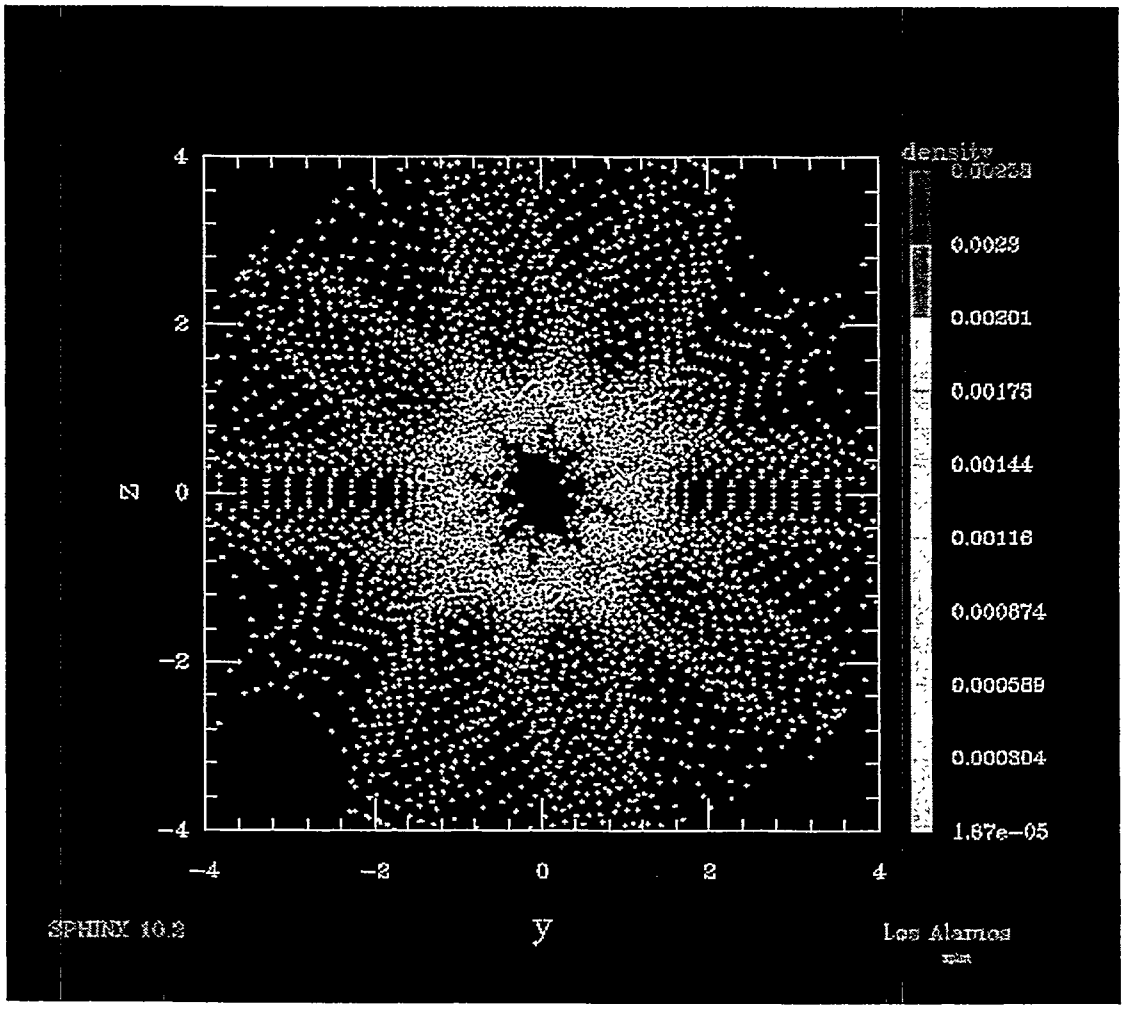


Fig. V. 6. This figure shows only jet particles from a 0.5 cm thick slab. The target particles have been removed to show the formation of the piston more clearly.

At maximum compression the target was heated to a temperature of 4.2×10^8 °K, and

compressed to a peak density of 0.0002 gm/cc and an average density of 0.00013 gm/cc for a duration of about 50 nsec. The piston reached an average density of 0.00065 gm/cc for a duration of about 70 nsec, and a peak density of 0.0023 gm/cc. The results are summarized in Table V. 1.

The following table shows a comparison of the 2D 24-jet case, the 3D 60-jet case, and the desired results.

Table 1: Comparison, at Maximum Compression, of SPHINX Calculations vs. Desired, t=0.4 μs.

	Desired	2D, 24 jets	3D, 60 jets
Target			
Temperature	10 kev	10 kev	10 kev
Density	0.001 gm/cc	0.0003 gm/cc	0.0005 gm/cc
Confinement	100 nsec	50 nsec	50 nsec
Diameter	1.0 cm	0.3 cm	1.2 cm
Piston			
Temperature	–	2.4 kev	3.0 kev
Density	–	0.002 gm/cc	0.0006 gm/cc
Confinement	100 nsec	70 nsec	70 nsec

This 3D problem was tried using the implicit code with the number of particles significantly reduced. There were a total of 5,768 particles, with 96 per jet, and 8 in the target. This case did run for a few time-steps, and all 60 jets were propagated toward the target properly, but before the jets started to interact, the allocation for memory was exceeded. The allocation could have been increased, but it was apparent that the implicit code was not going to do the problem any faster than the explicit code. However, this run

did demonstrate that the 3D aspects of the implicit code are working correctly, and it was able to handle nearly 6,000 particles.

E. Conclusions of the MTF study

The results of this study have shown that a fairly uniform spherical piston can be formed from an array of neutral plasma jets that start merging from some distance out from the target. When the jets first encounter each other, "super-jets" squirt out from between them. By varying the distance from the target, the super-jets are allowed to spread out and form a more uniform front before impacting the target. If the super-jets form too close to the target, they are found to cut up the target and not compress it as uniformly as desired.

Since this work has shown that a spherical piston might be formed from neutral plasma jets, and that conditions near D-T fusion can be reached, it would be very interesting to pursue this with an optimization study. One would want to look at varying the initial conditions such as the distance between the target and the jets; the densities, temperatures, and dimensions of the jets and the target; and the velocities and number of jets. Additional physics could be included in the explicit code, such as radiation losses and magneto-hydrodynamic capabilities.

Chapter VI

Conclusions

An implicit Smooth Particle Hydrodynamic code has been written, and has been demonstrated to give good agreement with analytic solutions, the explicit code, and experimental data for the various test cases tried. It has recently been demonstrated that for one case it can perform the calculations faster than the explicit code with Courant numbers of over 3,000, albeit a very unphysical problem (see Chapter IV, section G). This one case, however, does prove that it can be done, and it is a first step in expanding the code's capabilities to practical problems. In another case the code handled nearly 6,000 particles

This study is probing the boundaries of a new aspect of the field of SPH, and hence it may or may not be found to be a useful region for SPH to operate. The existing SPHINX code is typically used for high-velocity impact studies. An implicit code is best used for low-velocity, incompressible fluid problems. The new implicit code will need some changes to improve on the explicit code, but further work may yet make it useful for SPH.

The capabilities of the new implicit code, which is 1D, 2D, or 3D, include a choice of three Krylov solvers along with a multi-pass Jacobi preconditioner to solve the linear system. It also uses Newton-Raphson iteration methods with a line-search to solve the nonlinear problem.

The code uses sparse storage and sparse computations to minimize the amount of memory and time used. It also makes use of a new right-hand-side function in the SPHINX code, written just for these sparse matrices. Also, the Jacobian matrix is calcu-

lated numerically, as opposed to analytically, which allows the new code to take advantage of almost all of the existing physics that is already in the SPHINX code.

Five distinctly different versions of the code exist, and each newer version has shown a marked improvement in performance over the older ones. The first one uses LU Decomposition, a fourth-order Rosenbrock solver, and analytic derivatives for the Jacobian matrix, and stores the full matrix. It was very slow and cumbersome to use. The next version uses numerical derivatives, Krylov solvers, and Newton-Raphson iterations, but still calculates and stores the full Jacobian matrix. As a result it uses large amounts of memory, and performs many unnecessary calculations of zero. However, because of this factor it does catch some unexpected matrix elements that the summation density method generates. The sparse versions of the code will need to be modified to handle the summation method and are currently restricted to using the continuity equation.

The two sparse versions of the code are very similar, and both store only the non-zero elements of the Jacobian and do not calculate the portions of the matrix that are known to be zero, that is, the portions where there is no neighbor relation between particles. The difference is in the right-hand-side function, the function that calculates the time derivatives for each equation. One uses a function that calculates the rhs for all the particles, and the other uses a function that calculates the rhs for only one particle at a time. The latter one saves on calculations because for each particle the rhs function only has to be calculated for its neighbors, not all particles. This version is the most efficient of the five.

The fifth version, which was attempted but temporarily set aside, uses a matrix-

free technique. It worked for only the simplest problems. It was probably attempted prematurely, because it needs a very good preconditioner. The diagonal-block preconditioner will, hopefully, allow the matrix-free method to work reliably. The equations for this preconditioner have been worked out in what appears to be an efficient arrangement, but it would have to be programed and tested, and is being left for future work.

The following are several other ideas to improve the implicit code.

1. Set numbers in the Jacobian matrix that are near zero to zero. There would then be fewer elements and perhaps faster convergence, and less memory required.
2. Add damping for greater stability. That is, do not let dY change by more than some fixed percent of Y.
3. Parallelize the code to run on a multiprocessor computer.
4. Optimize settings of the limits and tolerances for a more general set problems. There is some indication that the values for the limits and tolerances need to be changed for different problems, and they are interrelated.
5. Currently the effort has been to try to avoid cutting the time step size at all, which means cutting it in half, but perhaps a smaller decrease in the time-step size would permit a smaller limits on the Newton and Krylov iterations.
6. Explore the possibility that other SPH hydro-forms of the fluid equations might perform better in the implicit code than those being used.

The implicit code may prove useful when used in conjunction with the MLS features being added to SPHINX because MLS is more computationally intensive than standard SPH, as was mentioned in Chapter I Section B. It may help speed up MLS in parts of problems where things are changing very slowly.

There have been a number of unexplained features encountered in the SPHINX code, which, if understood, may help both the implicit and explicit codes. One is the unexplained escaping of particles through the reflecting boundaries. This escape can hap-

pen even when the time-step multiplier is just one. And it has been observed to happen for the explicit code too, but it appears to be a problem only at low velocities. For this reason, the single-jet test case was studied to get away from all reflecting boundaries. This may be an indication of a fundamental problem in the SPH code.

Another problem noted is the occurrence of unexpected elements in the Jacobian when the summation method is used to calculate the density. It is not understood why the explicit code cannot do the rarefaction and shock-tube problems when using the continuity equation. These extra elements in the Jacobian for the summation method may lead to an understanding of why use of the continuity equation does not give the right answer.

Another problem is: when the smoothing length h is allowed to vary, should h be computed before the right-hand-side is computed, or after? If before, it seems to keep h from varying, and if after, h is not consistent with the time derivatives of the rhs() function. The matrix-free method seems to work only if h is consistent with the time derivatives of the right-hand-side computations, but then h does not vary. Where to do the computations for new h's has not been resolved in either the implicit or the explicit code.

The goals of this research have been reached. An implicit code has been written, and it has been shown that it can run a test case with Courant numbers on the order of 3000. It has been shown in several test cases that the accuracy and precision of the code is very good. With more improvements it promises to become a useful working code that can be included as another time-step package for the production code SPHINX.

Appendix

A Rayleigh-Taylor problem with finite boundaries

A derivation of the Rayleigh-Taylor problem with finite boundaries at $z = a$ can be found by starting with the general solution to the equation derived in Hoffman [37], for which he started with the fluid equations and applied perturbation theory and Fourier transformed them and obtained his Eq. (A-27):

$$\frac{\partial}{\partial z}\left(\rho_o \frac{\partial v}{\partial z}\right) = k^2 \rho_o v\left(1 - \frac{g}{\gamma^2 \rho_o}\frac{\partial \rho_o}{\partial z}\right). \qquad (A.1)$$

If ρ_o is considered to be constant, Eq. (A .1) reduces to:

$$\frac{\partial^2 v}{\partial z^2} = k^2 v, \qquad (A.2)$$

for which the general solution is a sum of exponentials.

$$v(z) = A e^{kz} + B e^{-kz}. \qquad (A.3)$$

The velocity is to vanish at both boundaries and be continuous at the interface. Therefore at $z = +a$, $v(a) = 0$. That is

$$v(a) = A_1 e^{ka} + B_1 e^{-ka} = 0 \quad \text{or} \quad A_1 = -B_1 e^{-2ka}, \qquad (A.4)$$

$$\text{thus} \quad v(z>0) = B_1 (e^{-kz} - e^{-2ka} e^{kz}). \qquad (A.5)$$

Similarly for $z = -a$, $v(-a) = 0$, and

$$v(-a) = A_2 e^{-ka} + B_2 e^{ka} = 0 \quad \text{or} \quad B_2 = -A_2 e^{-2ka}, \qquad (A.6)$$

$$\text{thus} \quad v(z<0) = A_2 (e^{kz} - e^{-2ka} e^{-kz}). \qquad (A.7)$$

At the interface the velocities are to be continuous, thus for $z = 0$

$$B_1(1-e^{-2ka}) = A_2(1-e^{-2ka}), \tag{A .8}$$

or $B_1 = A_2 \equiv v_o$, which is the initial velocity of the interface. Therefore, for the Rayleigh-Taylor problem with finite boundaries at $z = a$ Eq. (A .3) becomes:

$$v(z) = \begin{cases} v_o(e^{-kz} - e^{-2ka}e^{kz}), z>0 \\ v_o(e^{kz} - e^{-2ka}e^{-kz}), z<0 \end{cases} . \tag{A .9}$$

The growth rate of the Rayleigh-Taylor problem with finite boundaries can be derived by integrating Eq. (A .1) over a small region of z from $-\varepsilon$ to ε that includes the interface, and then letting ε go to zero. It is assumed that the densities above and below the interface are each constant within their respective regions, so that $\rho(z>0) \equiv \rho_{above}$ and $\rho(z<0) \equiv \rho_{below}$. Integrating the left hand side of Eq. (A .1) and using the derivatives of the results of Eq. (A .9) with respect to z, one obtains:

$$I_1 \equiv \int_{-\varepsilon}^{\varepsilon} \frac{\partial}{\partial z}\left(\rho_o \frac{\partial v}{\partial z}\right) dz = \rho_o \frac{\partial v}{\partial z}\Big|_{-\varepsilon}^{\varepsilon} = -kv_o(\rho_{above} + \rho_{below})(e^{-k\varepsilon} + e^{-2ka}e^{k\varepsilon}) , \tag{A .10}$$

which, in the limit as ε goes to zero, becomes

$$I_1 = -kv_o(\rho_{above} + \rho_{below})(1 + e^{-2ka}) \tag{A .11}$$

The integral of the first term of the right hand side of Eq. (A .1) can be split into integrals above and below the interface:

$$I_2 \equiv \int_{-\varepsilon}^{\varepsilon} k^2 \rho_o v dz = \int_{0}^{\varepsilon} k^2 \rho_{above} v dz + \int_{-\varepsilon}^{0} k^2 \rho_{below} v dz \tag{A .12}$$

$$I_2 = \int_{0}^{\varepsilon} k^2 \rho_{above} v_o(e^{-kz} - e^{-2ka}e^{kz}) dz + \int_{-\varepsilon}^{0} k^2 \rho_{below} v_o(e^{kz} - e^{-2ka}e^{-kz}) dz \tag{A .13}$$

$$\mathbf{I}_2 = k^2 \mathrm{v}_0 \left[\rho_{above} \left(\frac{1}{-k} e^{-kz} - e^{-2ka} \frac{1}{k} e^{kz} \right) \Big|_0^{\varepsilon} + \rho_{below} \left(\frac{1}{k} e^{kz} - e^{-2ka} \frac{1}{-k} e^{-kz} \right) \Big|_{-\varepsilon}^{0} \right], \quad \text{(A .14)}$$

$$\mathbf{I}_2 = k^2 \mathrm{v}_0 \left[\frac{1}{-k} \rho_{above} \{ (e^{-k\varepsilon} - 1) + e^{-2ka} (e^{k\varepsilon} - 1) \} \right. \quad , \quad \text{(A .15)}$$
$$\left. + \frac{1}{k} \rho_{below} \{ (1 - e^{-k\varepsilon}) + e^{-2ka} (1 - e^{k\varepsilon}) \} \right]$$

which, in the limit as ε goes to zero, becomes

$$\mathbf{I}_2 = 0. \quad \text{(A .16)}$$

Integrating the last term of the right hand side can be done using integration by parts, and the remaining integral being split up into parts above and below the interface.

$$\mathbf{I}_3 \equiv -\int_{-\varepsilon}^{\varepsilon} k^2 \mathrm{v} \frac{g}{\gamma^2} \frac{\partial \rho_0}{\partial z} dz = -k^2 \frac{g}{\gamma^2} \left[\rho_0 \mathrm{v} \Big|_{-\varepsilon}^{\varepsilon} + \int_{-\varepsilon}^{\varepsilon} \rho_0 \frac{\partial \mathrm{v}}{\partial z} dz \right], \quad \text{(A .17)}$$

$$\mathbf{I}_3 = -k^2 \frac{g}{\gamma^2} \left[\rho_0 \mathrm{v} \Big|_{-\varepsilon}^{\varepsilon} + \int_0^{\varepsilon} \rho_{above} \partial \mathrm{v} + \int_{-\varepsilon}^{0} \rho_{below} \partial \mathrm{v} \right], \quad \text{(A .18)}$$

$$\mathbf{I}_3 = -k^2 \frac{g}{\gamma^2} [\rho_{above} \mathrm{v}_0 (e^{-k\varepsilon} - e^{-2ka} e^{k\varepsilon}) - \rho_{below} \mathrm{v}_0 (e^{k\varepsilon} - e^{-2ka} e^{-k\varepsilon}) \quad , \quad \text{(A .19)}$$
$$+ \rho_{above} \mathrm{v}_0 \{ (e^{-k\varepsilon} - 1) - e^{-2ka} (e^{k\varepsilon} - 1) \}$$
$$+ \rho_{below} \mathrm{v}_0 \{ (1 - e^{-k\varepsilon}) - e^{-2ka} (1 - e^{k\varepsilon}) \}]$$

which, in the limit as ε goes to zero, becomes

$$\mathbf{I}_3 = -k^2 \frac{g}{\gamma^2} \mathrm{v}_0 [(\rho_{above} - \rho_{below}) (1 - e^{-2ka})]. \quad \text{(A .20)}$$

From $\mathbf{I}_1 = \mathbf{I}_2 + \mathbf{I}_3$ and Eqs. (A .11), (A .16), and (A .20) the following is obtained:

$$\gamma^2 = k g \frac{(\rho_{above} - \rho_{below})}{(\rho_{above} + \rho_{below})} \frac{(1 - e^{-2ka})}{(1 + e^{-2ka})} . \quad \text{(A .21)}$$

The new part of the result is the ratio of the exponential function. A plot of the function $F = [(1 - e^{-2ka})/(1 + e^{-2ka})]^{1/2}$ shows that for $a = \lambda$ the new function F is equal to approximately one (F = 0.999997), hence does not affect the growth rate, and therefore behaves the same as for the Rayleigh-Taylor problem with infinite boundaries. Not until a is below about $\lambda/2$ does it start to affect the growth rate, and then F = 0.998, which is still a small change. However, it rapidly drops to zero below $\lambda/2$.

References

[6] Arnoldi, W. E. (1951). The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem, *Quarterly of Applied Mathematics*, **9**, pp. 17-29.

[7] Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., & van der Vorst, H. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods,* SIAM, Philadelphia.

[8] Barrett, R. (July, 1994). Algorithmic Bombardment for the Iterative Solution of Linear Systems: A Poly-Iterative Approach, Masters thesis, University of Tennessee, Knoxville, Tennessee.

[9] Belytschko, T., Lu, Y. Y., & Gu, L. (1994). Element-Free Galerkin Methods, *Int. J. for Numerical Methods in Engineering*, **37**, pp. 229-256.

[10] Benz, W. (1989). Smooth Particle Hydrodynamics: A Review, *Harvard-Smithsonian Center for Astrophysics*, No. 2884.

[11] Benz, W. & Asphaug, E. (1995). Simulations of Brittle Solids Using Smooth Particle Hydrodynamics, *Computer Physics Communications*, **87**, pp. 253-265.

[12] Cash, J. R. & Psihoyios, Y. (1996). The MOL Solution of Time Dependent Partial Differential Equations, *Computers Math. Applic.*, **31**, No. 11, pp. 69-78.

[13] Chandrasekhar, S. (1981). *Hydrodynamic and Hydromagnetic Stability,* Dover Publications, New York.

[14] Choi, C. K., Hoffman, N. M., Clover, M. R., Powers, W. J. (1997), Simulations of Linear and Nonlinear Rayleigh-Taylor Instability Under High Atwood Numbers, Proceeding of 1997 Laser Interactions and Plasma Phenomena (LIRPP).

[15] Cloutman, L. D. (1991). An Evaluation of Smoothed Particle Hydrodynamics, an article in: *Advances in the Free-Lagrange Method*, Proceedings of the Next Free-Lagrange Conference, Jackson Lake, Moran, Wy., USA, H. E. Trease, M. J. Fritts, W. P. Crowley (Eds.), Lecture Notes in Physics **395**, Springer-Verlag, New York, pp. 229-238.

[16] Coleman, T. F., Garbow, B. S. & More, J. J. (Sept., 1984). Software for Estimating Sparse Jacobian Matrices, *ACM Transactions on Mathematical Software*, **10**, No. 3, pp. 329-345.

[17] Coleman, T. F. & Van Loan, C. (1988). *Handbook for Matrix Computations*, SIAM, Philadelphia.

[18] Crotzer, L. A., Dilts, G. A., Knapp, C. E., Morris, K. D., Swift, R. P., & Wingate, C. A. (April, 1998). SPHINX Manual Version 11.0, Los Alamos National. Lab. Manual: LA-13436-M.

[19] Cullum, J. K. & Willoughby, R. A. (1985). *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol. I Theory, (Vol. II Programs)*, Birkhäuser, Boston.

[20] Curtis, A. R., Powell, M. J. D., & Reid, J. K. (1974). On the Estimation of Sparse Jacobian Matrices, *J. of the Inst. of Maths. and it's Applics.*, **13**, pp. 117-119.

[21] Degnan, J. H., Baker, W. L., Cowan, M, Jr., Graham, J. D., Holmes, J. L., Lopez, E. A., Price, D. W., Ralph, D., & Roderick, N. F. (May, 1999). Operation of Cylindrical Array of Plasma Guns, *Fusion Technology*, **35**, pp.354-360.

[22] Dilts, G. A. (1997). Moving-Least-Squares-Particle Hydrodynamics I Consistency and Stability, Los Alamos National. Lab. Report: LA-UR-97-4168.

[23] Dilts, G. A. (1998). Conservative Moving-Least-Squares Methods for Lagrangian Hydrodynamics, Los Alamos National. Lab. Report: LA-UR-98-304.

[24] Duff, I. S. & Reid, J. K. (June, 1978). An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix, *ACM Transactions on Mathematical Software*, **4**, No. 2, pp. 137-147.

[25] Dukowicz, J. K. & Meltz, B. J. A. (1991). Vorticity Errors in Multidimensional Lagrangian Codes, an article in: *Advances in the Free-Lagrange Method*, Proceedings of the Next Free-Lagrange Conference, Jackson Lake, Moran, WY., H. E. Trease, M. J. Fritts, W. P. Crowley (Eds.), Lecture Notes in Physics 395, Springer-Verlag, New York, pp. 289-292.

[26] Freund, R. W. & Nachtigal, N. M. (1991). QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems, *Numerische Mathematik*, **60**, pp. 315-339.

[27] Fulk, D. A. (1994). A Numerical Analysis of Smoothed Particle Hydrodynamics, Ph.D dissertation, Dept. of Mathematics and Statistics, Air Force Institute of Technology, Wright Patterson AFB, Ohio.

[28] Fulk, D. A. & Quinn, D. W. (1996). An Analysis of 1-D Smoothed Particle Hydrodynamics Kernels, *Journal of Computational Physics*, **126**, No. 1, pp. 165-180.

[29] Gear, G. W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall Inc., Englewood Cliffs, New Jersey.

[30] Gingold, R. A. & Monaghan, J. J. (1977). Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars, *Mon. Not. Roy. Astron. Soc.* **181**, pp. 375-389.

[31] Goldston, R. J. & Rutherford, P. H. (1995), *Introduction to Plasma Physics*, Institute of Physics Publishing, Bristol and Philadelphia.

[32] Golub, G. H. & Van Loan, C. F. (1996). *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, Baltimore, (Note: the 1st and 2nd Eds. do not cover iterative methods for non-symmetric linear systems).

[33] Graham, M. J. (1996). A Numerical Study of the Richtmyer-Meshkov Instability in Cylindrical Geometry, a Ph.D. dissertation for the Dept. of Applied Mathematics and Statistics from the State University of New York.

[34] Hernquist, L. & Katz, N. (June, 1989). TREESPH: A Unification of SPH with the Hierarchical Tree Method, *The Astrophysical Journal Supplement Series*, **70**, pp. 419-446.

[35] Hestenes, M. R. & Stiefel, E. L. (1952). Methods of Conjugate Gradients for Solving Linear Systems, *J. of Research of the National Bureau of Standards*, Section B, **49**, pp. 409-436.

[36] Hockney, R. W. & Eastwood, J. W. (1988). Computer Simulation Using Particles, Adam Hilger, Bristol.

[37] Hoffman, N. M. (1995). "Hydrodynamic Instabilities in Inertial Confinement Fusion," *Laser Plasma Interactions 5: Inertial Containment Fusion*, Hooper, M. B., ed., Institute of Physics Publishing, Bristol, pp. 105-137.

[38] Householder, A. S. (1964). *Theory of Matrices in Numerical Analysis*, Dover Publications, New York.

[39] Johnson, G. R. (1996). Artificial Viscosity Effects for SPH Impact Computations, *Int. J. Impact Engineering.*, **18**, No. 5, pp. 477-488.

[40] Kaps, P. & Rentrop, P. (1979). Generalized Runge-Kutta Methods of Order Four with Stepsize Control for Stiff Ordinary Differential Equations, *Numer. Math.*, **33**, pp. 55-68.

[41] Kelley, C. T. (1995). *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia.

[42] Kirkpatrick, R. C,. & Lindemuth, I. R, (1997) Magnetized Target Fusion, An Overview of the Concept, *Current Trends in International Research*, ed. E. Panarella, Plenum Press, New York.

[43] Knoll, D. A., Rider, W. J., & Olson, G. L. (1998). An Efficient Nonlinear Solution Method for Nonequilibrium Radiation Diffusion, Los Alamos National. Lab. Report: LA-UR-98-2154. Also submitted for publication to: *J. of Quantitative Spectroscopy and Radiative Transfer.*

[44] Kopal, Z. (1961). *Numerical Analysis*, 2nd Ed., John Wiley & Sons Inc., New York.

[45] Lanczos, C. (1950). An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators, *J. of Research of the National Bureau of Standards,* **45**, pp. 255-482.

[46] Landau, L. D. & Lifshitz, E. M. (1975). *Fluid Mechanics*, Pergamon Press, Oxford.

[47] Lanczos, C. (1952). Solution of Systems of Linear Equations by Minimized Iterations, *J. of Research of the National Bureau of Standards,* **49**, pp. 33-53.

[48] Libersky, L. D. & Petschek, A. G. (1991). Smooth Particle Hydrodynamics with Strength of Materials, an article in: *Advances in the Free-Lagrange Method*, Proceedings of the Next Free-Lagrange Conference, Jackson Lake, Moran, Wy., USA, H. E. Trease, M. J. Fritts, W. P. Crowley (Eds.), Lecture Notes in Physics 395, Springer-Verlag, New York, pp. 248-257.

[49] Libersky, L. D. & Randles, P. W. (1998). Boundary Conditions in a Meshless Staggered Particle Code, Los Alamos National. Lab. Report: LA-UR-98-590.

[50] Lindemuth, I. R., & Kirkpatrick, R. C. (1991). The Promise of Magnetized Fuel: High Gain in Inertial Confinement Fusion, *Fusion Technology*, **20**, pp. 829-833. Also a Los Alamos National. Lab. Report: LA-UR-91-2498.

[51] Lucy, L. (1977). A Numerical Approach to Testing the Fission Hypothesis, *Astron, J.*, **82**, pp. 1013-1024.

[52] Mandell, D. A., Wingate, C. A., Dilts, G. A., Schwalbe, L. A. (Oct. 1996). Computational Brittle Fracture Using Smooth Particle Hydrodynamics (U), Los Alamos National. Lab. Report: LA-CP-96-209.

[53] Martin, J. C. & Moyce, W. J. (1952) Part IV. An Experimental Study of the Collapse of Liquid Columns on a Rigid Horizontal Pane, *Phil. Trans. of the Royal Soc. of London*, **244**, pp. 312-324.

[54] Mendelson, A. (1970). *Plasticity: Theory and Application*, The Macmillan Co., New York, New York. Reprinted in 1991 by University Microfilms International (UMI) Out-of-Print Books on Demand, Ann Arbor, Michigan.

[55] McCormick, S. T. (1983). Optimal Approximation of Sparse Hessians and its Equivalence to a Graph Coloring Problem, *Mathematical Programming*, **26**, pp. 153-171.

[56] Monaghan, J. J. (1982). Why Particle Methods Work, *SIAM J. on Scientific and Statistical Computing*, **3**, No. 4, pp. 422-433.

[57] Monaghan, J. J. (1985). A Refined Particle Method for Astrophysical Problems, *Astronomy and Astrophysics*, **149**, pp. 135-143.

[58] Monaghan, J. J. (1985). Particle Methods for Hydrodynamics, *Computer Physics Reports*, **3**, pp. 71-124, (North-Holland, Amsterdam).

[59] Monaghan, J. J. (1988). An Introduction to SPH, *Computer Physics Communications,* **48**, pp. 89-96, (North-Holland, Amsterdam).

[60] Monaghan, J. J. (1992). Smoothed Particle Hydrodynamics, *Annual Review of Astronomy and Astrophysics*, **30**, pp. 543-574.

[61] Monaghan, J. J. (1994). Simulating Free Surface Flows with SPH, *Journal of Computational Physics*, **110**, pp. 399-406.

[62] Monaghan, J. J. & Gingold, R. A (1983). Shock Simulation by the Particle Method SPH, *Journal of Computational Physics*, **52**, pp. 347-389.

[63] Mousseau, V. A. (May, 1996). Fully Implicit Kinetic Modelling of Collisional Plasmas, a Ph. D. dissertation from the University of Idaho, also report number INEL-96/0149 Idaho National Engineering Laboratory.

[64] Oran, E. S. & Boris, J. P.(1987). *Numerical Simulation of Reactive Flow*, Elsevier Science Publishing Co. Inc., New York.

[65] Press, W. H. & Teukolsky, S. A. (May/June 1989). Integrating Stiff Ordinary Differential Equations, *Computers in Physics*, pp. 88-91.

[66] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1995). *Numerical Recipes in C the Art of Scientific Computing, 2nd Ed.*, and *Numerical Recipes Example Book [C], 2nd Ed*, Cambridge University Press, Cambridge.

[67] Pritchett, J. W. & Rice, M. H. (1975). User's Guide to the AQUA Subprogram System - A Comprehensive Constitutive Package for Water, S-Cubed Report: SSS-IR-75-2544.

[68] Rice, M. H., Gurtman, G. A., & Skoller, B. (1997). SPHINX Code Enhancements Appropriate to Tactical Missile Defense Engagements, DSWA-TR-97-35.

[69] Rider, W. J. (1993). Stability of Semi- and Nearly-Implicit Schemes for Thermal-Hydraulics, presented at the National Heat Transfer Conference 1993.

[70] Rosenbrock, H. H. (1963). Some General Implicit Processes for the Numerical Solution of Differential Equations, *Computer Journal*, **5**, pp. 329-330.

[71] Saad, Y. (1996). *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston.

[72] Saad, Y. & Schultz, M. H. (1986). GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. on Scientific and Statistical Computing*, **7**, pp. 856-869.

[73] Serna, A., Alimi, J. M., & Chieze, J. P. (1996). Adaptive Smooth Particle Hydrodynamics and Particle-Particle Coupled Codes: Energy and Entropy Conservation, *Astrophysical Journal*, **461**, No. 2, pt. 1, pp. 884-896.

[74] Siemon, R. E., Lindemuth, I. R., & Schoenberg, K. F. (Dec. 1997). Why Magnetized Target Fusion Offers A Low-Cost Development Path for Fusion Energy, *Comments on Plasma Physics and Controlled Fusion.*

[75] Sonneveld, P. (1989). CGS, a Fast Lanczos-Type Solver for Nonsymmetric Linear Systems, *SIAM J. on Scientific and Statistical Computing*, **10**, No. 1, pp. 36-52.

[76] Stellingwerf, R. F., Buff, J. (April, 1978). Stability of Astrophysical Gas Flow. I. Isothermal Accretion, *The Astrophysical Journal*, **221**, pp. 661-671.

[77] Stellingwerf, R. F. (August, 1983). HYDRA: An Implicit Partial Differential Equation, Relaxation, Stability Analysis Package, *The Astrophysical Journal*, **271**, pp. 876-878.

[78] Stellingwerf, R. F. (1991). Smooth Particle Hydrodynamics, an article in: *Advances in the Free-Lagrange Method*, Proceedings of the Next Free-Lagrange Conference, Jackson Lake, Moran, Wy., USA, H. E. Trease, M. J. Fritts, W. P. Crowley (Eds.), Lecture Notes in Physics 395, Springer-Verlag, New York, pp. 239-247.

[79] Stellingwerf, R. F. & Wingate, C. A. (1992). Impact Modeling With Smooth Particle Hydrodynamics, (submitted to the 1992 Hypervelocity Impact Symposium, November17-20, 1992), Los Alamos National. Lab. Report: LA-UR-92-1981.

[80] Stoer, J. & Bulirsch, R., English Translation: (1980). *Introduction to Numerical Analysis*, Springer-Verlag Inc., New York.

[81] Strang, G. (1986). *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, Mass.

[82] Swegle, J. W., Hicks, D. L., & Attaway, S. W. (1993). Smoothed Particle Hydrodynamics Stability Analysis, *J. of Computational Physics*, **116**, pp. 123-134.

[83] Swift, R. P., Hagelberg, C. R., Carney, T. C., Greening, D., Hiltl, M. (Feb. 14-17, 2000). Modeling Stress-Induced Damage from Impact Recovery Experiments, Proceedings of the ETCE/OMAE 2000 Joint Conference: Energy for the New Millennium, New Orleans, La.

[84] Thio, Y. C. F., Knapp, C. E., & Kirkpatrick, R. C. (1998). The Feasibility of Merged Compact Toroids Compressed By Multiple Plasma Jets as a Possible Embodiment of MTF, a poster presented at the 1998 Innovative Confinement Concepts Workshop, at Princeton, NJ, April 6-9 1998.

[85] Thio, Y. C. F., Kirkpatrick, R. C., Knapp, C. E., Panarella, E., Wysocki, F. J., & Parks, P. (1998). An Embodiment of Magnetized Target Fusion in a Spherical Geometry with Stand-off Drivers, a Los Alamos Natl. Lab. Report: LA-UR-98-269. Also presented at the 1998 Innovative Confinement Concepts Workshop, at Princeton, NJ, April 6-9 1998. Also presented at the ICOPS '98, Raleigh, NC, June 1-4, 1998. Also to be published in *Current Trends in International Fusion Research - Proceedings of the Second Symposium*, E. Panarella (ed.), National Research Council of Canada Press, 1999.

[86] Timmes, F. X. (1993). Reactive Flows in Compact Objects, Ph.D dissertation, Dept. of Astronomy and Astrophysics, University of California, Santa Cruz.

[87] Warren, M. S. & Salmon, J. K. (1995). A Portable Parallel Particle Program, *Computer Physics Communications*, **87**, pp. 266-290.

[88] Weaver, T. A., Zimmerman, G. B., & Woosley, S. E. (Nov. 1, 1978). Presupernova Evolution of Massive Stars, *The Astrophysical Journal*, **225**, pp. 1021-1029.

[89] Wells, D. R., Ziajka, P. E., & Tunstall, J. L. (1986). Hydrodynamic Confinement of Thermonuclear Plasmas Trisops VIII (Plasma Liner Confinement), *Fusion Technology*, **9**, pp. 83-96.

[90] Wendroff, B. (1966). *Theoretical Numerical Analysis*, Academic Press Inc., New York.

[91] Wendroff, B. (1969). *First Principles of Numerical Analysis*, Addison Wesley Publishing Co., Reading, Massachusetts.

[92] Wingate, C. A., Dilts, G. A., Mandell, D. A, Crotzer, L. A., Knapp, C. E., & Libersky, L. D. (1998). Progress in Smooth Particle Hydrodynamics, (submitted to the Fourth World Congress on Computational Mechanics, Buenos Aires, Argentina, June 29 - July 2, 1998) Los Alamos National. Lab. Report: LA-UR-98-582.

[93] Wingate, C. A., Stellingwerf, R. F., Davidson, R. F., & Burkett, M. W. (1992). Models of High Velocity Impact Phenomena, (submitted to the 1992 Hypervelocity Impact Symposium, November17-20, 1992), Los Alamos National. Lab. Report: LA-UR-92-1982.

[94] Wingate, C. A. & Stellingwerf, R. F. (1993). Smooth Particle Hydrodynamics - The SPHINX and SPHC Codes (submitted to the 1993 ASME Winter Meeting, November 28 - December 4, 1993), Los Alamos National. Lab. Report: LA-UR-93-1938.

[95] Wingate, C. A. & Stellingwerf, R. F. (1995). Los Alamos SPHINX Manual Version 7.6, Los Alamos National. Lab. Report: LA-UR-93-2476

[96] Zel'dovich, Ya. B. & Raizer, Yu. P. (1967). Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena, Volumes I & II, Academic Press, New York.

[97] Zlatev, Z. (1991). *Computational Methods for General Sparse Matrices*, Kluwer Academic Publishers, Boston.

ACKNOWLEDGEMENTS

I wish to acknowledge and thank the people who made this dissertation possible. My advisor at the University of New Mexico (UNM) Chemical and Nuclear Engineering (ChNE), Dr. Norman F. Roderick, and my advisor at the Los Alamos National Labs (LANL), Dr. Charles A. Wingate have both contributed immensely to my learning and understanding of the broad range of subjects needed to conduct this research. In addition to my two advisors, I also want to thank the others on my committee: Dr. Deborah L. Sulsky (Mathematics and Statistics, UNM); Dr. Anil K. Prinja (ChNE, UNM); and Dr. Gary W. Cooper (ChNE, UNM). I also want to acknowledge the significant technical consultation given by the following people (alphabetically): Dr. Daniel C. Barnes, Dr. Richard F. Barrett, Dr. Michael R. Clover, Dr. Stephen V. Coggeshall, Dr. Charles W. Cranfill, Dr. Gary A. Dilts, Dr. Doran R. Greening, Dr. Nelson M. Hoffman, Dr. Ronald C. Kirkpatrick, Dr. Dana A. Knoll, Dr. William D. Nystrom, Dr. William J. Rider, Dr. Peter T. Sheehey, Dr. D. Palmer Smitherman, Dr. Robert F. Stellingwerf, Dr. Y. C. Francis Thio, and Dr. Robert B. Webster. I also thank the group leaders at LANL who have supported me financially in doing this work: Dr. Douglas C. Wilson, Dr. Robert C. Little, and Dr. Len G. Margolin. Thank you all.