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 , City, State, 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

LA-13685-T
Thesis

Issued: January 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).

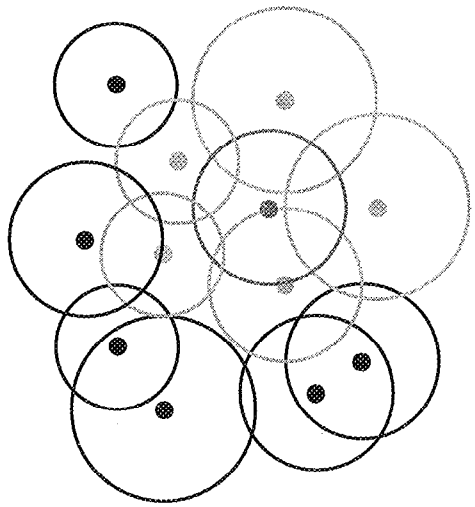

Fig. I. 1. An example of particles (dots) and their circles of influence, which may be of different radii and change with time. Each particle has a local set of neighbors influencing its motion. The particles may have different physical properties such as position, velocity, density, pressure, internal 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 $$\frac{d}{dt}\mathbf{v} = -\left(\frac{1}{\rho}\right)\nabla P, \tag{2.1}$$

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

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

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 $m_1, \ldots, m_N$, 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).

$$W_{ij} = \begin{cases} C(1 - \frac{3}{2}u_{ij}^2 + \frac{3}{4}u_{ij}^3), & \text{for}(0 \le u_{ij} < 1) \\ C\frac{1}{4}(2 - u_{ij})^3, & \text{for}(1 \le u_{ij} < 2) \\ 0, & \text{for}(u_{ij} \ge 2) \end{cases} \quad \text{, where} \tag{2 .28}$$

$$u_{ij} \equiv \frac{r_{ij}}{h}, \quad \text{and} \quad r_{ij} \equiv \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad ; \tag{2 .29}$$

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,

$$\text{for 1D: } C \equiv \frac{2}{3h}, \quad \text{for 2D: } C \equiv \frac{10}{7\pi h^2}, \quad \text{and for 3D: } C \equiv \frac{1}{\pi h^3}. \tag{2 .30}$$

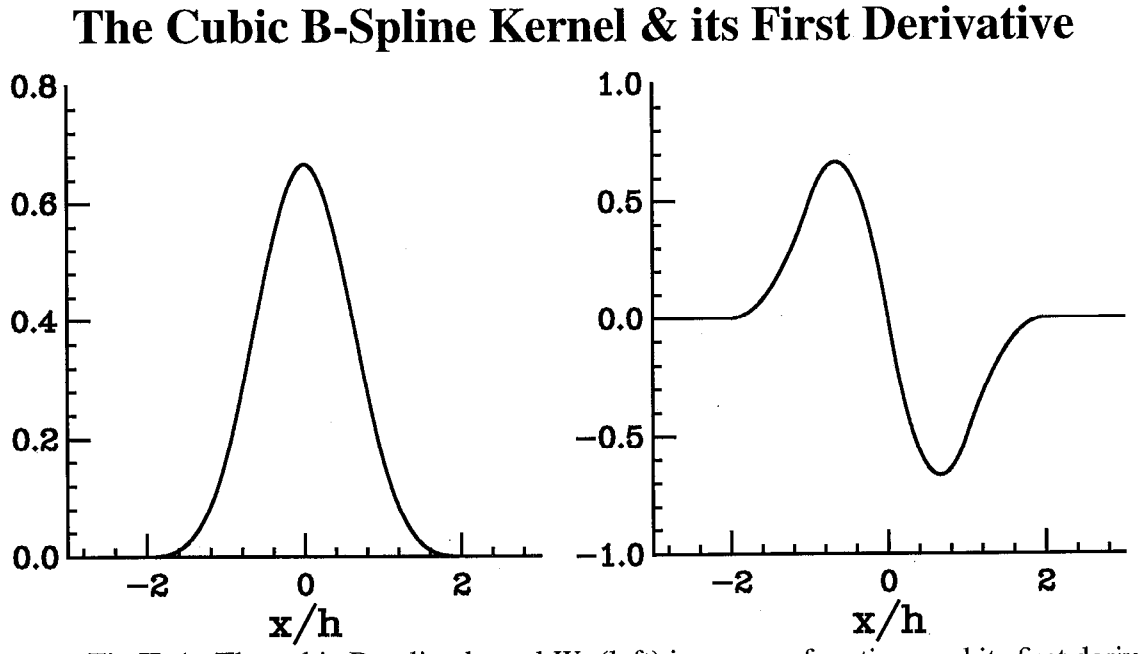

Fig II. 1 The cubic B-spline kernel W_{ij} (left) is an even function, and its first derivative (right) is odd. The kernel has zero slope at the origin and for x > 2h. Since x is always positive, only the right half of these functions is actually used in the calculations.

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).

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

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:

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

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:

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

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 **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 **f(Y)** be an arbitrary vector function of the vector **Y,** which may be nonlinear. For SPH, **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:

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

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

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

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

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

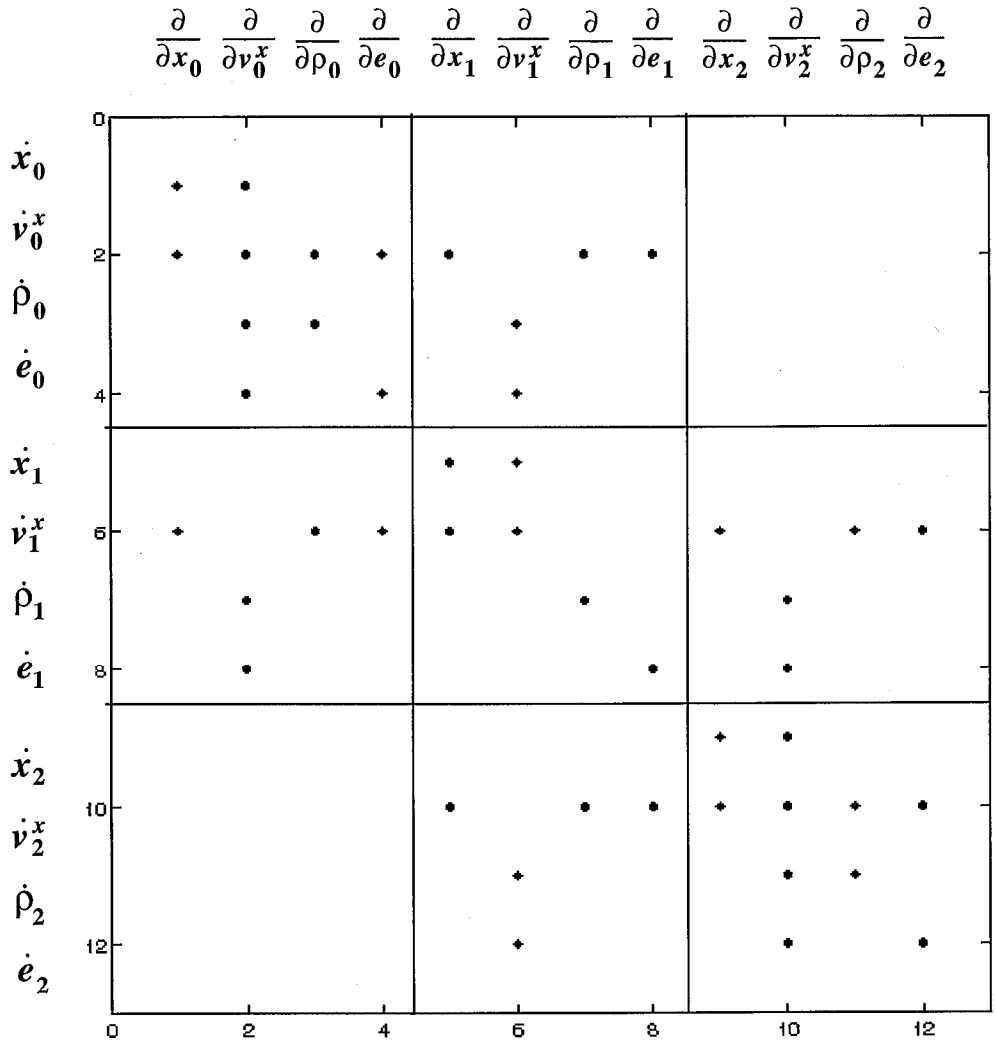

Fig. III. 1. This is a simple 12 x 12 Jacobian spot matrix for a 1D 3-particle problem, showing a dot wherever there is a non-zero element. Down the left side are indicated the time-derivative equations (note the dot over each variable) of which the partial derivatives are taken. The partial derivatives 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, $rhs(\mathbf{Y})$ is first run using the unperturbed values of $\mathbf{Y}$, and all the resulting time derivatives for each particle are stored in a vector function $\mathbf{f}_o$. Then one element in the state vector $\mathbf{Y}$ is perturbed by $\varepsilon\mathbf{Y}_k$, and $rhs(\mathbf{Y}+\varepsilon\mathbf{Y}_k)$ is run again. Differencing the values of the vector $\mathbf{f}_o$ 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 $\mathbf{Y}$ is set back to its original value, the next element of $\mathbf{Y}$ 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 **x** and **b** are vectors and **A** is a non-singular matrix to be inverted, the general problem is of the form $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, where **A** and **b** are given and **x** is the unknown. Instead of inverting **A**, the iterative methods solve $\mathbf{Ax} - \mathbf{b} = \mathbf{0}$ approximately, by guessing at a solution, $\mathbf{x}_0$, and then iterating on **x** until a vector of residual errors **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 **M** and **c** are modifications to **A** and **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 **A** into parts:

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

where, using a modified notation of Saad [71], **D** is the diagonal of **A**, and **E** and **F** are the two triangular parts of **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 $\mathbf{x}_0$, 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 $\mathbf{A}$ 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:

$$\phi(\mathbf{x}) = (1/2)\,\mathbf{x}^{\mathbf{T}}\,\mathbf{A}\,\mathbf{x} - \mathbf{x}^{\mathbf{T}}\,\mathbf{b}. \tag{3 .80}$$

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),

$$\nabla\phi(\mathbf{x}) = \mathbf{A}\,\mathbf{x} - \mathbf{b} = \mathbf{0}. \tag{3 .81}$$

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 **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.