LA-12984 c. 2

LA -12984 c.2

Introduction to Finite-Difference Methods for Numerical Fluid Dynamics

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.

Edited by Patricia W. Mendius, Group CIC-1

LA-12984

UC-700
Issued: September 1995

Introduction to Finite-Difference Methods for Numerical Fluid Dynamics

Evan Scannapieco
Francis H. Harlow

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545

TABLE OF CONTENTS

INTRODUCTION TO FINITE-DIFFERENCE METHODS FOR NUMERICAL FLUID DYNAMICS

by

Evan Scannapieco and Francis H. Harlow

ABSTRACT

This work is intended to be a beginner's exercise book for the study of basic finite-difference techniques in computational fluid dynamics. It is written for a student level ranging from high-school senior to university senior. Equations are derived from basic principles using algebra. Some discussion of partial-differential equations is included, but knowledge of calculus is not essential. The student is expected, however, to have some familiarity with the FORTRAN computer language, as the syntax of the computer codes themselves is not discussed. Topics examined in this work include: one-dimensional heat flow, one-dimensional compressible fluid flow, two-dimensional compressible fluid flow, and two-dimensional incompressible fluid flow with additions of the equations of heat flow and the $K - \epsilon$ model for turbulence transport. Emphasis is placed on numerical instabilities and methods by which they can be avoided, techniques that can be used to evaluate the accuracy of finite-difference approximations, and the writing of the finite-difference codes themselves. Concepts introduced in this work include: flux and conservation, implicit and explicit methods, Lagrangian and Eulerian methods, shocks and rarefactions, donor-cell and cell-centered advective fluxes, compressible and incompressible fluids, the Boussinesq approximation for heat flow, cartesian tensor notation, the Boussinesq approximation for the Reynolds stress tensor, and the modeling of transport equations. A glossary is provided which defines these and other terms.

I. INTRODUCTION

One of the most important techniques used in the computer modeling of physical systems, finite differencing represents an essential part of modern theoretical physics. Able to generate solutions to systems that are far too intricate to be solved analytically, this technique has given physicists the ability to model, examine, and better understand complex physical situations. From the study of microscopic systems to the modeling of the world's climate, finite-difference programs have opened up an entire field of research that has only been possible within the the past 40 years.

In the following paper we will examine a series of finite-difference programs, gaining a clearer understanding of their underlying physical principles and the techniques by which these are implemented. It is our intention to represent these physical systems so that they will be easily understood both by those who are dealing with them for the first time and those familiar with their partial-differential representations. Partial differential equations will be used but only as a result of a discussion of the basic principles from which they are derived. The mathematics will follow, as it should, from a clear set of physically meaningful observations.

It will be essential, however, for the reader to have a clear understanding of the FORTRAN computer language in which all programs will be written. While there are many finite-difference simulations that are written in other languages, FORTRAN has proved to be an efficient, easily understood, and widely accepted language for scientific computing. For these reasons we will limit our discussion to simulations written in this language, and for reasons of scope we will not discuss the meanings of each of its commands. We assume that the reader is already familiar with computers and desires to apply this knowledge to the study of physical systems. It is our intention to apply physical principles to the creation of computer simulations, not to discuss the syntax of the simulations themselves.

There are two ways to approach this work. It can be interpreted as a guide to writing one's own finite difference simulations, a type of handbook for creating one's own codes, or it can be read as a textbook, omitting actual programming by the reader. We hope that the reader will adopt the former approach. While it will take more time for the reader to create his own simulations, the extra time will prove to be time well spent.

The reasons for writing one's own code are twofold. First, the reader who is able to structure his own code will be sure to have a full understanding of the concepts involved. There is only so much that can be explained about the process; true understanding will result only from experience. Secondly, by writing his own programs, the reader gains the advantage of being able to examine the results obtained from various initial parameters. Only a limited number of results will be presented in this work, thereby leaving the reader with a vast set of cases which can be independently investigated.

As we move through this series of programs, we will examine a broad spectrum of physical systems. We will begin with the simulation of heat transfer in one dimension, examining various forms of numerical instabilities and explicit and implicit solution techniques. Our discussion will then move to compressible fluid flow in one dimension, examining both Eulerian and Lagrangian methods of simulation of a number of different systems. We will show a method for determining the accuracy of our finite-difference representations and use it to examine numerical instabilities. We will discuss the simulation of incompressible fluid flow in two dimensions, calculate incompressible fluid flow in conjunction with heat, discuss two-dimensional compressible fluid flow, and finally implement the equations of turbulence transport in an incompressible code.

Likewise, our discussion will cover an equally broad set of topics in a range of technical fields. We will discuss various physical equations and the systems that they represent, the mathematical properties of these equations and how these relate to solving them with a computer, and the structuring of the programs themselves. Although we will be dealing with these varying subjects, a single underlying thrust must remain clear in our minds. We must remember that what we are doing is taking observable physical phenomena and translating them into terms which can be dealt with by the computer. The form will be greatly changed, but the basic physical principles will always be faithfully represented.

In as much as we can simulate reality, we can use the computer to make predictions about what will occur in a certain set of circumstances. Finite-difference techniques can create an artificial laboratory for examining situations which would be impossible to observe otherwise, but we must always remain critical of our results. Finite-differencing can be an extremely powerful tool, but only when it is firmly set in a basis of physical meaning. In order for a finite-difference code to be successful, we must start from the beginning, dealing with simple cases and examining our logic each step of the way. Building further insights on what we have done in the past, we will start with the simplest case possible: heat transfer in a single dimension. The rest will follow logically.

II. ONE-DIMENSIONAL HEAT FLOW

A. Flux and Conservation

The first system that will be examined in this series of studies is that of heat conduction in a single dimension. In this chapter, we will write a program that numerically solves a single equation of heat transfer over a one-dimensional array. This program can most easily be pictured as the simulation of a metal rod that is initially at an even temperature and is insulated along its sides. As the program progresses, the simulated rod is heated from one end, and the resulting change in temperature along the rod is recorded as output. A diagram of this system appears in Fig. II-1.

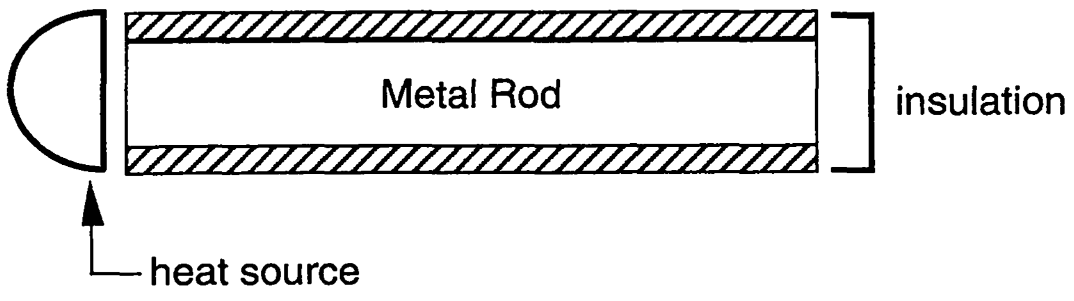

Fig. II-1

Each section of this rod is represented by an element in an array that corresponds to its position. These elements, called ZONES, record the temperature at finite distances along the rod and at finite time intervals, hence the name *finite difference.* This type of representation can be thought of as similar to a motion picture, where each frame exists for a small but finite time step. Motion is not fluid as in reality but is instead approximated by a series of small changes from one "frame" to the next.

Our simulation of heat flow in this manner will introduce two basic concepts that are essential to the understanding of the underlying principles on which many finite-difference codes are based: FLUX and CONSERVATION. Flux is the amount of something passing through a unit area in a unit time. In our current example, the flux that is of interest is HEAT FLUX, the transport of heat from one zone to another. But flux is by no means limited to only heat. It can represent the movement of mass, momentum, energy, or any other value that describes the amount of something that is present in a zone. No matter what is being fluxed, the concept remains essentially the same. Flux represents motion from one place to another, the rate at which something moves through a given area.

Conservation means that the total amount of something never changes regardless of its motion from one region to another. If this same concept is viewed in terms of the amount of something that exists in a finite region, conservation means that in any region of space the change in something equals the amount that goes in minus the amount that comes out plus the change of that amount within the region. Once again, this principle holds true for many different quantities. Mass, momentum, and energy, while different physically, are identical in the fact that they are conserved.

These two concepts of flux and conservation are critical to the way that our finite difference codes are structured. Their implementation and a simple equation obtained from experimental observation are all that is necessary to represent the transfer of heat numerically.

B. Numerical Representation

In order to represent this system in a manner that can be solved computationally, we must first examine the structure represented by each zone in our simulated metal rod. Looking at an individual zone (also called a CELL), we find a physical system as in Fig. II-2.

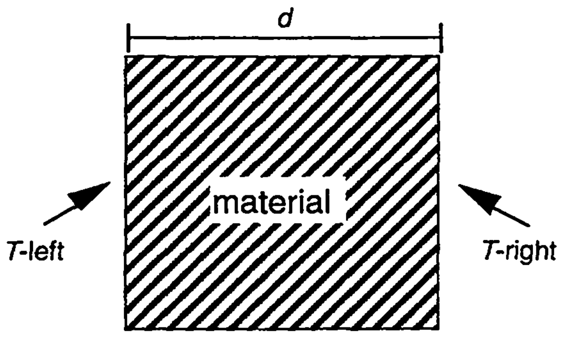

Fig. II-2

Here T-left and T-right represent the temperature along either edge of the zone, and d represents the thickness. The heat flux of this system is defined by an equation known as Fick's Law. A result of direct experimental observations, Fick's Law is as follows:

$$\text{flux of heat} = \frac{k\,(T_{\text{left}} - T_{\text{right}})}{d}\ . \qquad \text{(II-1)}$$

In this equation, k is called THE COEFFICIENT OF HEAT CONDUCTIVITY and is proportional to the rate at which a given material conducts heat across a temperature gradient. k is an intrinsic property of the material being represented and must be chosen based on the conductive properties of that material. For example, silver, a very conductive metal, is represented by a high value of k, around 4 J/s·cm· °C. Wood on the other hand, is a poor heat conductor and is consequently associated with a low k value, 1.3×10^{-3} J/s·cm· °C. The conductivity of iron is somewhere between these two materials, yielding an intermediate value for k, 0.67 J/s·cm· °C.

Taking this equation as it applies to a single cell, we can now make a generalization as to how it can be implemented over an array. Given a rod of length D, this length can be divided into an array of size $\bar{\jmath}$. Each zone will now have a length dx, which is defined as $D/\bar{\jmath}$. Such a system is shown in Fig II-3.

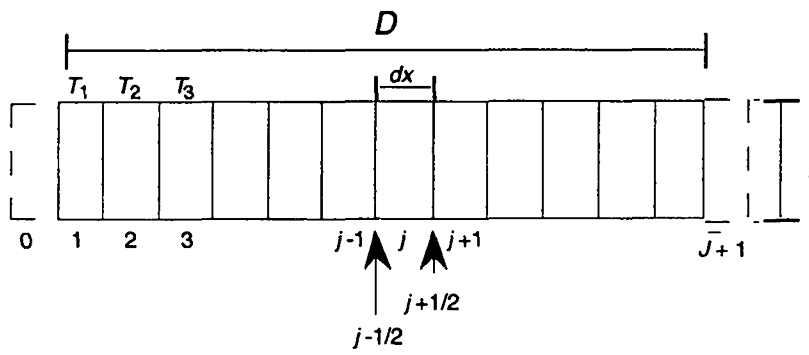

Fig. II-3

Each zone in this array can now be indexed with a counter j, with zones $j-1$ and $j+1$ being the zones at the left and right respectively. Note that the flux between zones will occur at the walls and will therefore occur at points such as $j+1/2$ and $j-1/2$ in the diagram. Note also that the diagram contains both a zone 0 and a zone $\bar{j}+1$, existing beyond the normal bounds of the rod. These zones are used to implement BOUNDARY CONDITIONS, equations that represent the external conditions that affect the values of the real zones. The heating source at the left of the pipe is represented by one such boundary condition. Temperatures for each cell are defined at the center of the cell, existing at positions 1, 2, 3, etc. Density of cells and cross-sectional area between cells are defined as ρ and A respectively.

These definitions can be used to write an expression for the heat energy contained in any given cell:

$$\text{Volume} = A\, dx \tag{II-2}$$

$$\text{Heat energy of cell } j = \text{Mass } b\, T_j\ , \tag{II-3}$$

where $b =$ the specific heat of the material. As

$$\text{Mass} = \text{Volume density} = A\, dx\, \rho\ , \tag{II-4}$$

the heat energy can also be written as

$$\text{Heat energy of cell } j = A\, dx\, \rho\, b\, T_j \, . \tag{II-5}$$

We now apply our conservation of energy principle to derive an equation for the change in energy. The letter n will be used as a TIME CYCLE COUNTER, an integer that represents the number of time cycles that have been calculated. These cycles, also called time steps, can be thought of as individual frames in our analogy of the motion picture. The time at a cycle n is represented by t^n, which is computed as follows:

$$t^n = n\, dt \, . \tag{II-6}$$

In this equation dt is equal to the time increment per cycle, i.e., the change in time between each "frame." The superscript n in t^n notates that the value being expressed occurs at time cycle n. It does *not* indicate t raised to the power n. We will continue to use superscipts in this manner, combining them with the subscripts used earlier to represent position. T_j^n will therefore be defined as the temperature in cell j at time step n, and similarly, T_j^{n+1} will represent the temperature in cell j at the time step $n + 1$.

By using this notation and assuming that our heat source is always placed at the left, energy conservation can be expressed as

$$\begin{aligned} &[\text{Heat Energy}]_j^{n+1} - [\text{Heat Energy}]_j^n = \\ &[\text{Amount in}]_{j-1/2}^n - [\text{Amount out}]_{j+1/2}^n \, . \end{aligned} \tag{II-7}$$

Referring now to our principle of flux:

$$[\text{Amount in}]_{j-1/2}^n = [\text{Flux}]_{j-1/2}^n\, A\, dt \tag{II-8}$$

and

$$[\text{Amount out}]_{j+1/2}^n = [\text{Flux}]_{j+1/2}^n\, A\, dt \tag{II-9}$$

Using Fick's law to determine flux at $j + 1/2$ and $j - 1/2$, and using the equation for heat energy of a cell (II-5), we can express Eq. (II-7) as follows:

$$A\,\rho\,dx\,b\,T_j^{n+1} - A\,\rho\,dx\,b\,T_j^n = k\frac{(T_{j-1}^n - T_j^n)}{dx}\,A\,dt - k\frac{(T_j^n - T_{j+1}^n)}{dx}\,A\,dt\,. \qquad \text{(II-10)}$$

This equation can be algebraically manipulated to obtain

$$T_j^{n+1} - T_j^n = \frac{kdt}{b\rho\,dx^2}(T_{j-1}^n - T_j^n - T_j^n + T_{j+1}^n)\,. \qquad \text{(II-11)}$$

$\frac{k}{b\rho}$ is often called the THERMOMETRIC CONDUCTIVITY of a material and is represented by the Greek letter σ. Thus, our conservation equation in final form appears as follows:

$$T_j^{n+1} - T_j^n = (\sigma\frac{dt}{dx^2})(T_{j-1}^n - 2T_j^n + T_{j+1}^n) \qquad \text{(II-12)}$$

This equation expresses heat flow in a manner that can be computationally solved. Based upon our knowledge of the previous time step, this equation allows us to calculate the new temperatures for every zone along the rod. By carrying out this equation repeatedly, the overall flow of heat can be observed.

Based on our discussion so far, it is now possible to begin writing the finite-difference code itself; but before this process is begun, let us first examine the nature of our equation. Although this equation has been generated from basic principles, it is obtained more often through the manipulation of partial differential equations. While not necessary to the writing of simple finite-difference codes, these partial-differential equations (P.D.E.'s) give scientists greater insight into simple systems and allow for analysis of much more complicated physical phenomena. Because these equations are continually being applied to finite-difference codes, it is important that they be examined and related to the problem at hand.

C. Partial-Differential Equations

For those familiar with partial-differential equations and their use, the following discussion of heat flow in analytical terms will serve to provide a different viewpoint into the construction of our finite-difference codes. However, this section is not essential to the writing of this code and should, therefore, not deter the reader who is unfamiliar with these expressions. Such a reader should try to work through these concepts without intimidation; they are merely provided as an alternate method to examining this problem.

Going back to Eq. (II-11) and distributing the dx^2 term among the temperatures, we obtain the following equation:

$$\frac{T_j^{n+1} - T_j^n}{dt} = \sigma \left[\frac{\frac{T_{j+1}^n - T_j^n}{dx} - \frac{T_j^n - T_{j-1}^n}{dx}}{dx} \right] . \tag{II-13}$$

By changing our nomenclature to more clearly represent T as a function of position and time, we can rewrite T_j^n as $T(x_j, t^n)$, T_{j+1}^n as $T(x_j + dx, t^n)$, and T_j^{n+1} as $T(x_j, t^n + dt)$. Our equation now takes the form

$$\frac{T(x_j, t^n + dt) - T(x_j, t^n)}{dt} = \sigma \left[\frac{\frac{T(x_j+dx,t^n)-T(x_y,t^n)}{dx} - \frac{T(x_y,t^n)-T(x_j-dx,t^n)}{dx}}{dx} \right] . \tag{II-14}$$

Using the definition of the derivative of $f(x)$, namely

$$\frac{df}{dx} \equiv \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx} ,$$

we take the limit as dt and $dx \to 0$ and obtain the following terms:

$$\lim_{dt \to 0} \frac{T(x_j, t^n + dt) - T(x_j, t^n)}{dt} = \frac{\partial T}{\partial t} \tag{II-15}$$

$$\lim_{dx \to 0} \frac{T(x_j + dx, t^n) - T(x_j, t^n)}{dx} = \left(\frac{\partial T}{\partial x} \right)_{j+1/2} \tag{II-16}$$

$$\lim_{dx \to 0} \frac{T(x_j + t^n) - T(x_j - dx, t^n)}{dx} = \left(\frac{\partial T}{\partial x} \right)_{j-1/2} . \tag{II-17}$$

Thus Eq. (II-14) can be rewritten:

$$\frac{\partial T}{\partial t} = \sigma \left[\frac{\left(\frac{\partial T}{\partial x}\right)_{j+1/2} - \left(\frac{\partial T}{\partial x}\right)_{j+1/2}}{dx} \right] . \tag{II-18}$$

Once again taking the limit as $dx \rightarrow 0$:

$$\frac{\partial T}{\partial t} = \sigma \frac{\partial^2 T}{\partial x^2} \,. \tag{II-19}$$

This is the heat-flow equation for a single direction. Starting with this equation, one would have been able to work backwards, choosing "finite differences" for each derivative and eventually generating Eq. (II-12). Derived from the same principles as our finite-difference equations, partial-differential equations provide a different outlook from which to approach computation.

D. Computational Implementation of Equations

Having derived an expression for heat flow in finite-difference form, the question still remains of how it will be computationally implemented. To complete this final stage in the writing of our code, three major issues must be examined: boundary conditions, redefinition of variables, and the structure of the program itself.

Our first major issue is the construction of boundary conditions. As was previously discussed, boundary conditions represent the external conditions that act to change a system. This representation is accomplished by the placing of zones beyond the normal boundaries of the array. The values of these FICTITIOUS ZONES or GHOST ZONES are chosen in such a way that they accurately express the external environment of the system in question. In this chapter, the conditions to be simulated will be a heat source at the left and an uninsulated area at the right of the rod.

While the temperature in each true zone within the rod will be determined by successive calculations of Eq. (II-12), the values at the fictitious zones will be calculated to represent fixed temperatures at either end of the array. In order to determine these values, consider the situation at either end of the array as represented in Fig. II-4.

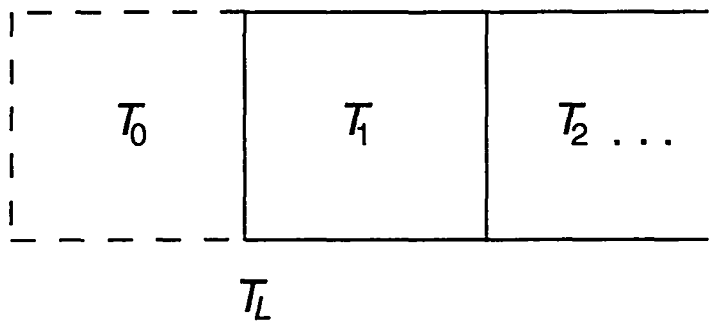

Fig. II-4

In this diagram, T_L represents the temperature along the left end of the rod, the value that should remain constant throughout the simulation. Although not present in the actual array, this constant temperature can be thought of as $T_{1/2}$, the average of the temperatures at zone T_1 and ghost zone T_0. Therefore T_L can be expressed as follows:

$$T_L = \frac{T_0 + T_1}{2} . \tag{II-20}$$

Solving this equation for T_0,

$$T_0 = 2T_L - T_1 . \tag{II-21}$$

Similarly, if T_R is defined as the temperature along the right end of the rod, $T_{\bar{j}+1}$ can be expressed as follows:

$$T_{\bar{j}+1} = 2T_R - T_{\bar{j}} . \tag{II-22}$$

By implementing these two equations, boundary conditions can be expressed for both the left and right of the system. Expressions for the other surfaces of the rod will not be needed, as they are assumed to be completely insulated, thus reducing the problem to one dimension.

The second major issue in solving of Eq. (II-12) computationally is the redefinition of variables. Thus far in this chapter, our equations have been represented in a manner that is not accepted by the FORTRAN programming language. Therefore certain modifications must be made in the way that various variables are represented; they must be redefined in terms of computationally accepted symbols:

$$\bar{j} \equiv \text{jbar}$$

$$\sigma \equiv \text{sig}$$

T_j^n, the temperature at zone j at time cycle n, will now be defined as T(j), an element in an array T defined from T(0) to T(jbar +1). Likewise, T_j^{n+1} will be defined as Tnew (j), an element in an array defined from Tnew (1) to Tnew (jbar).

The following new variables will also be defined:

T0 ≡ the intial temperature of the rod

stime ≡ the time at which the program ceases to run

ptime ≡ the time between successive displayings of the values of the zones in the array

pt ≡ a counter for ptime

st ≡ a counter for stime

Our finite-difference code will be divided into five sections, each with a clearly defined task. The first of these sections is the initialization procedure that dimensions the arrays and assigns initial values to all variables. This initialization is done in a subroutine that is called only once at the beginning of the program.

The second section of our code sets up a loop that repeats each time cycle. This section determines if the time counters pt and st have reached ptime and stime respectively and then increments the counters. If pt has reached ptime, it is reset to zero, and the current array of zones is sent to the output subroutine. If st reaches stime, the program terminates.

The third major section is the definition of boundary conditions, which occurs after the test for ptime and stime and before the actual computation of the next time cycle. In our particular program, this section should carry out Eq. (II-21) and Eq. (II-22) on the array T, updating values for the ghost zones at each time cycle.

The fourth section is the portion of the program that implements Eq. (II-12), which also occurs within the time-counting loop. This implementation is made up of two loops, the first of which assigns *Tnew* according to this equation, and the second of which transfers the values of *Tnew* back into *T*. The code for this section is as follows:

```
      do 100 j =1, jbar
      Tnew(j) = T(j) + sig*dt/dx**2 * (T(j+1) + T(j-1)-2*T(j))
100   continue
      do 200 j=1, jbar
      T(j) = Tnew(j)
200   continue
```

This two-loop structure is essential to the successful computation of T at the new time step. If one were to forego the computation of Tnew and directly compute T, the temperature terms at the right hand of the equation would not exist at the same time cycle. While T(j+1) and T(j) would still be at time n, T(j-1), having already been computed in the previous iteration of this do loop, would exist at time $n + 1$. By creating a second array and moving these values into T after they are all computed, we are able to avoid this problem.

The final section in our code is the output subroutine, which occurs when pt=ptime. This procedure could contain various graphics routines, write results to an output file, or simply display the various array values on the screen. A diagram of these sections and their interactions appears in Fig. II-5.

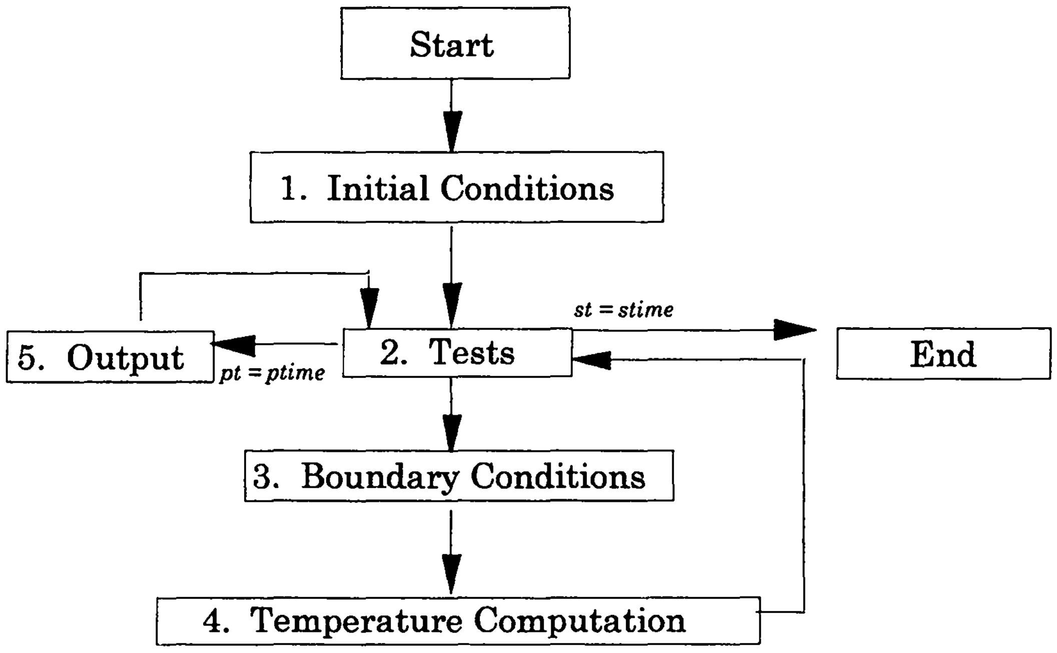

Fig. II-5

E. Programming and Results

We have now reached a point where the reader should be able to write his own finite-difference code for heat transfer. In this work a limited series of examples are examined in order to demonstrate the output of our code.

Figures II-6 through II-10 below show the results of a simulation of an insulated rod that is originally at 0°C, with a fixed temperature at the left (T_L) of 400°C, fixed temperature at the right (T_R) of 0°C, a thermometric conductivity (σ) of 1.0 m^2 per sec, a zone length (dx) of one meter, a $\bar{\jmath}$ of 50, and a time step (dt) of 0.1 seconds. Results are shown at 10, 50, 100, 250, and 1000 seconds.

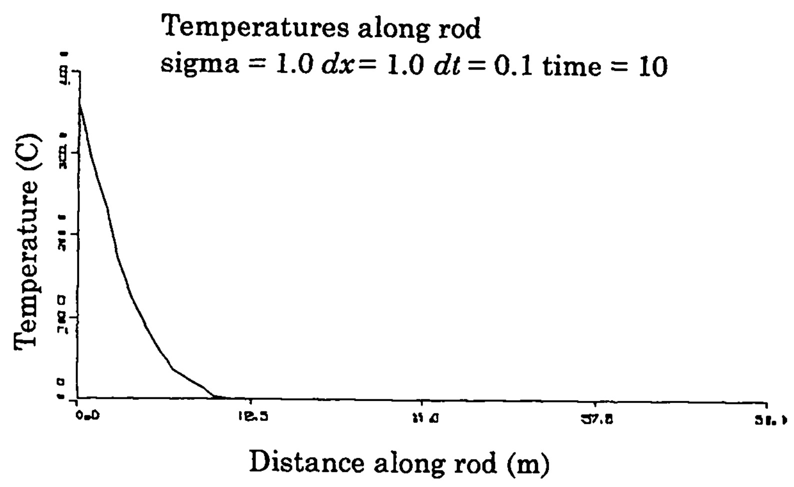

Figure II-6

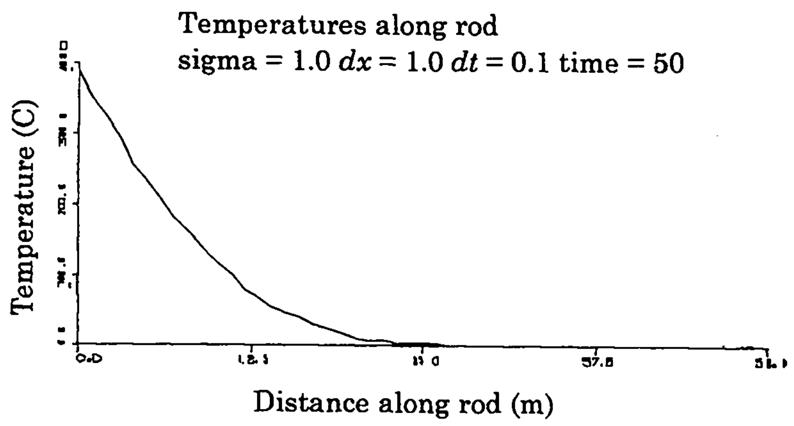

Figure II-7

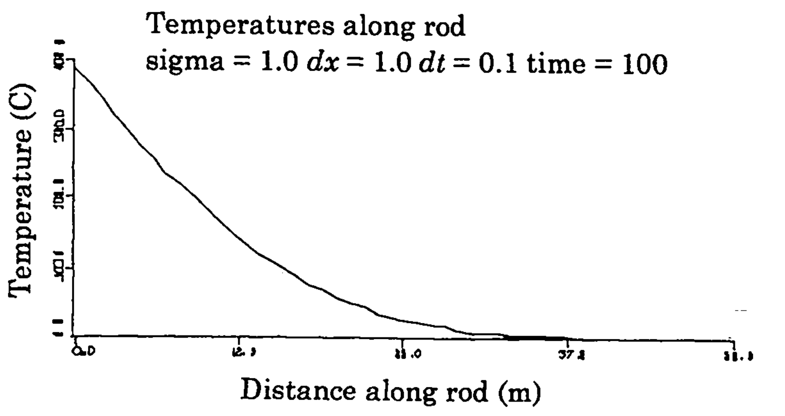

Figure II-8

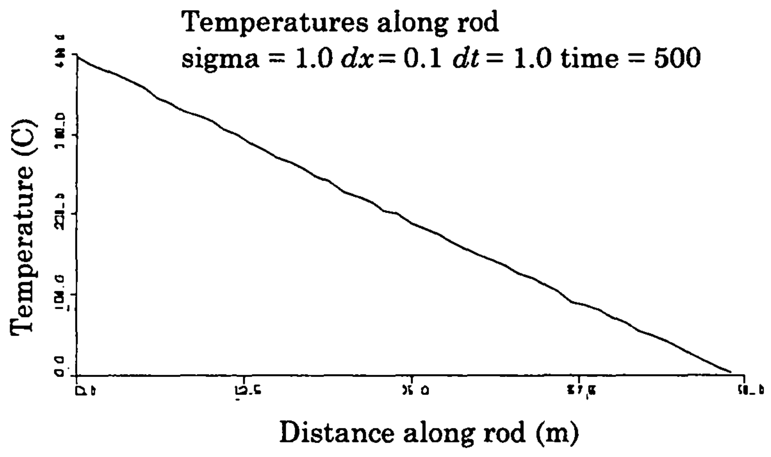

Figure II-9

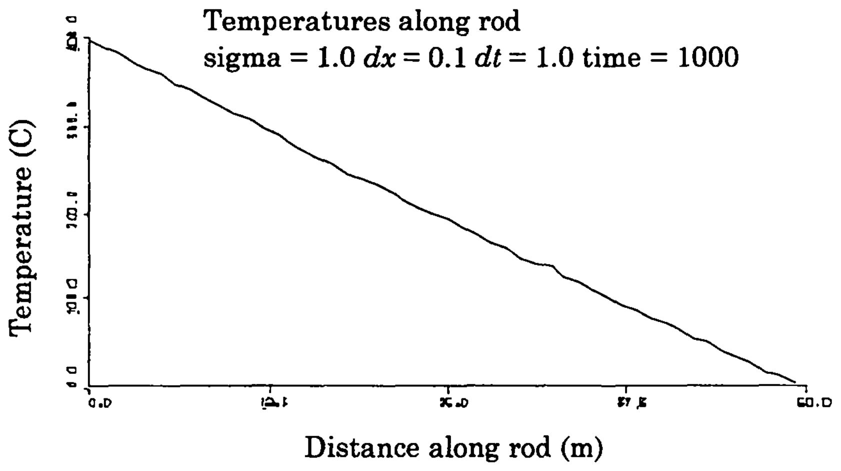

Figure II-10

Notice how the zone temperatures approach a straight line as the simulation progresses. Such a line is the final steady-state solution of this system, regardless of thermometric conductivity. Solutions at earlier time steps can be approximated using Eq. (III-55), which is derived at the end of Chapter III.

In the next three graphs, the effects of changes in time step are shown in relation to a simulation which is otherwise identical to the one above. Figure II-11 shows a temperature curve for a system at a time of 100 seconds with a time step of 0.495, just under 1/2.

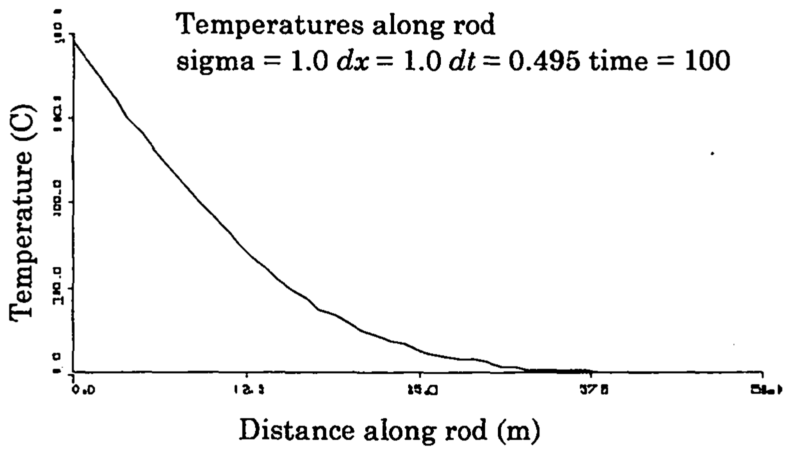

Figure II-11

This figure is almost identical to Fig. II-8, indicating that there is little difference between the results obtained with a time step of 0.1 and the results obtained with a time step of 0.495. Results are quite different, however, when a time step of 0.5 is used, as in Fig. II-12.

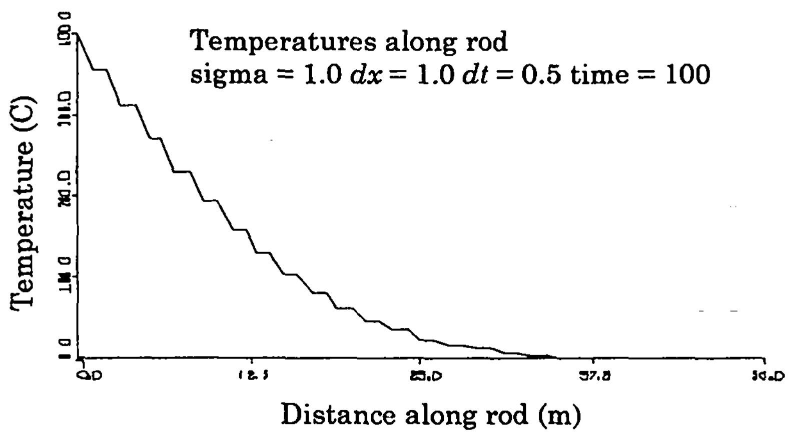

Figure II-12

The stair-step type temperatures that can be seen in this graph are a result of a numerical instability. This instability becomes even more violent when the time step is further increased to 0.505, as in Fig. II-13.

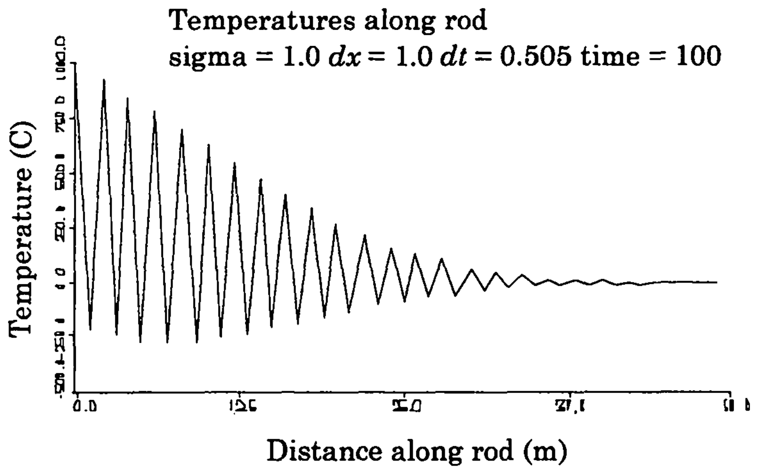

Figure II-13

Notice that in this figure the highest temperatures are much greater than 400°C, whereas the lowest temperatures are below −250°C. Obviously, this does not accurately represent the transfer of heat down the rod.

The numerical instability seen in Figs. II-12 and II-13 arises whenever the quantity $\frac{\sigma dt}{dx^2}$ is greater than 1/2. The presence of this instability means that the more accurately one wishes to resolve a set of circumstances, the shorter the time step that must be chosen. This problem highly limits the sorts of cases that can be simulated, yet there is a method by which it can be overcome. The following chapter examines this numerical instability and discusses the use of an implicit method of solution—a method that increases the speed, accuracy, and applicability of our finite-difference codes.

III. NUMERICAL INSTABILITY AND IMPLICIT CALCULATIONS

A. A Graphical Explanation of the Diffusional Stability Condition

In the cases presented at the end of the last chapter, we discovered that our finite-difference code is numerically unstable when the value of $\frac{\sigma dt}{dx^2}$ exceeds 1/2. This constraint is known as the diffusional stability condition, and it is one of two important conditions that we will examine in our series of finite-difference codes.

Consider the simplest case possible for our simulation: that of a rod at a constant temperature, T_0, with this same temperature at either end. Now consider the case in which this system of constant temperatures is perturbed by slightly increasing the temperatures of the odd-numbered zones by an amount ϵ and slightly decreasing the temperatures of the even-numbered zones by the same amount. The result is a system such as depicted in Fig. III-1.

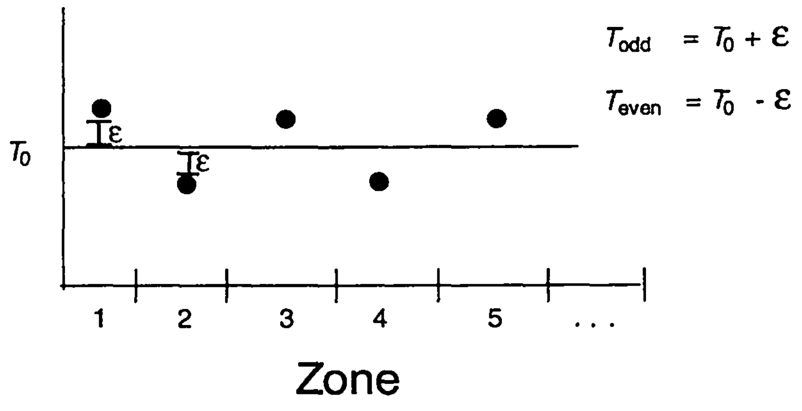

Fig. III-1

Let us now chose an odd numbered zone, j, and examine the calculation of its temperature at each time step. We begin with

$$T_j^{n+1} - T_j^n = \frac{\sigma dt}{dx^2}\left[T_{j+1}^n + T_{j-1}^n - 2T_j^n\right] \qquad \text{(II-12)}$$

and substitute our new definition of T to obtain

$$T_j^{n+1} = T_0 + \epsilon + \frac{\sigma dt}{dx^2}\left[T_0 - \epsilon + T_0 - \epsilon - 2(T_0 + \epsilon)\right] \; , \tag{III-1}$$

which can be reduced to

$$T_j^{n+1} = T_0 + \epsilon - \frac{4\epsilon\sigma dt}{dx^2} \tag{III-2}$$

or

$$T_j^{n+1} = T_0 + \epsilon\left[1 - \frac{4\sigma dt}{dx^2}\right] \; . \tag{III-3}$$

If a constant ξ is defined such that

$$\xi \equiv \frac{4\sigma dt}{dx^2} \; ,$$

our equation becomes

$$T_j^{n+1} = T_0 + \epsilon[1 - \xi] \; . \tag{III-4}$$

Note that ξ in this equation is made up of all positive components; therefore, $\xi > 0$ and $(1 - \xi) < 1$ in all circumstances.

These constraints leave us with four cases to examine, the first of which occurs when $0 < \xi < 1$. In this case, $1 - \xi$ is a fraction between 0 and 1. T_j^{n+1} is therefore computed as $T_0 + \epsilon$ (fraction), yielding a value closer to T_0 than the previous time step. Subsequent iterations of this equation generate a series of temperatures such as shown in Fig. III-2.

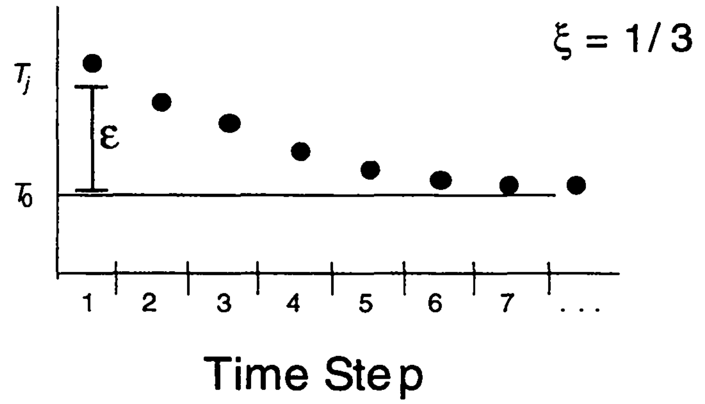

Fig. III-2

In this case, the temperature converges towards T_0, moving towards the array of constant temperatures that defines a correct solution.

In our second case $\xi = 1$, leaving us with the equation $T_j^{n+1} = T_0$. The graph of this case converges immediately, as illustrated in Fig. III-3.

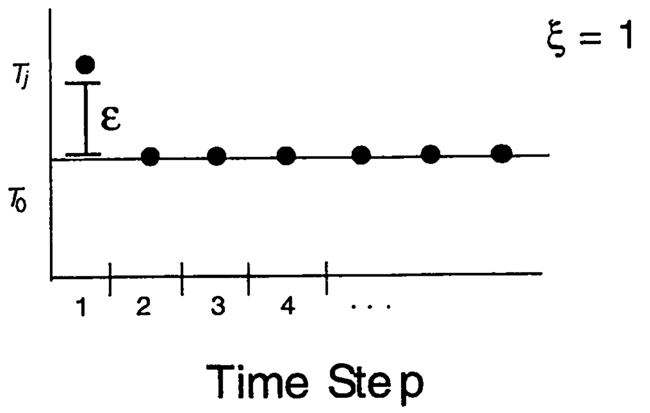

Fig. III-3

Our third curve is similar to the first and occurs when $1 < \xi < 2$. When this is true, $1 - \xi$ is again a fraction, but this time it is a number between 0 and −1. The result is a set of values which alternate above and below T_0 but converge toward that value as shown in Fig. III-4.

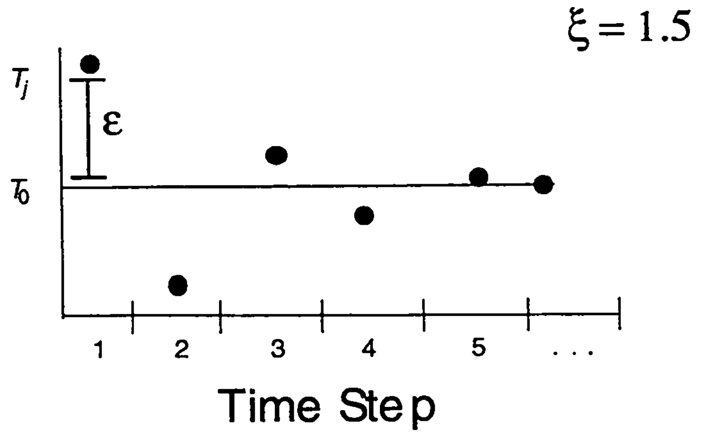

Fig. III-4

A corollary to this case occurs when $\xi = 2$. For this value the graph oscillates but does not converge, as in Fig. III-5. This case, while not convergent, is still considered to represent the bounds of numerical instability.

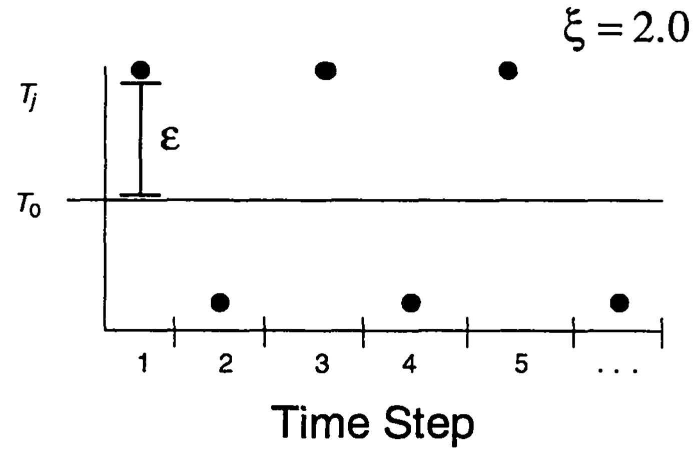

Fig. III-5

Our fourth and final case occurs as soon as this bound is crossed, when $\xi > 2$. In this set of circumstances $1 - \xi < -1$, yielding values of T^{n+1} that not only oscillate but diverge from the correct solution. The graph of temperatures appears as in Fig. III-6, with values diverging until the program is terminated.

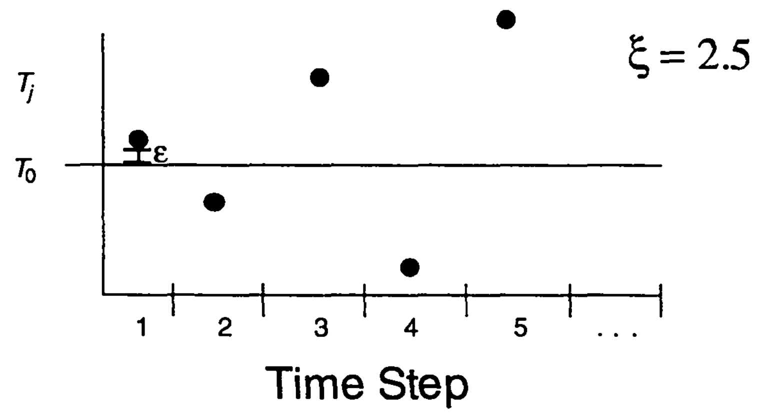

Fig. III-6

In order to avoid this condition, as well as the stable but nonconverging state pictured in Fig. III-5, we must choose ξ such that $\xi < 2$. Referring to our definition of ξ as $\frac{4\sigma dt}{dx^2}$, we obtain

$$\frac{4\sigma dt}{dx^2} < 2 \,, \qquad \text{(III-5)}$$

which is the diffusional stability condition

$$\frac{\sigma dt}{dx^2} < \frac{1}{2} \,. \qquad \text{(III-6)}$$

We have therefore demonstrated graphically that this condition must be met for a solution to converge.

B. A Mathematical Derivation of the Diffusional Stability Condition

Once again, we will turn to a mathematical explanation to reinforce an argument that has been made graphically. This section, like section II-C, is not essential to the writing of our codes; it is simply another method of arriving at the diffusional stability condition and better explaining the manner in which it can be overcome. Again, one should follow as closely as possible, gaining familiarity with the application of various mathematical methods towards this problem.

We will begin with the heat-flow equation as expressed in Eq. (II-12), this time substituting our definition for ξ:

$$T_j^{n+1} = T_j^n + \frac{\xi}{4}\left(T_{j+1}^n + T_{j-1}^n - 2T_j^n\right) \,. \qquad \text{(III-7)}$$

Let us examine the behavior of T_j^n with the following trial solution:

$$T_j^n = Ae^{i\,k\,j\,dx}e^{i\,w\,n\,dt} \,. \qquad \text{(III-8)}$$

Here T_j^n represents T at a time step n and zone j, whereas $e^{i\,k\,j\,dx}$ and $e^{i\,w\,n\,dt}$ represent e raised to $i\,k\,j\,dx$ and $i\,w\,n\,dt$ respectively, where i is the imaginary number. If r is defined as

$$r \equiv e^{i\,w\,dt} \,,$$

Eq. III-8 becomes

$$T_j^n = Ae^{i\,k\,j\,dx} r^n \,. \tag{III-9}$$

From this equation, we see that r must be between 1 and -1 for T_j^n to converge. If $r > 1$, the solution will diverge monotonically, moving farther and farther towards either positive or negative infinity. If $r < -1$, the solution will diverge in an oscillatory manner, alternating between positive and negative values but always moving away from convergence.

Keeping these restrictions on r in mind, let us now use our test definition of T_j^n to substitute for temperature terms in Eq. (III-7):

$$Ae^{i\,k\,j\,dx} r^{n+1} = Ae^{i\,k\,j\,dx} r^n + \frac{\xi}{4}\left[Ae^{ik(j+1)} r^n + Ae^{i\,k\,(j-1)} r^n - 2Ae^{i\,k\,j} r^n\right] \,. \tag{III-10}$$

Dividing both sides by $Ae^{i\,k\,j\,dx} r^n$ gives

$$r = 1 + \frac{\xi}{4}\left[e^{i\,k\,dx} + e^{-i\,k\,dx} - 2\right] \,. \tag{III-11}$$

Using the identity $e^{i\theta} = \cos\theta + i\sin\theta$ we can rewrite this equation as

$$r = 1 + \frac{\xi}{4}\left[2\cos k\,dx - 2\right] \tag{III-12}$$

or

$$r = 1 - \frac{\xi}{2}\left[1 - \cos k\,dx\right] \,. \tag{III-13}$$

Consider the extreme cases for the $\cos kdx$ term, namely $\cos k\,dx = +1$ and -1. If $\cos k\,dx = +1$, then Eq. (III-13) reduces to $r = 1$, a valid statement according to the restrictions that we have placed on r. This case poses no problems.

Taking the other extreme, $\cos kdx = -1$, we are left with

$$r = 1 - \xi \,. \tag{III-14}$$

Because ξ is always positive, r can never exceed 1 in this case. It can, however, be less than -1, a problem which places the following condition on ξ:

$$-1 < 1 - \xi \tag{III-15}$$

or

$$\xi < 2\,. \tag{III-16}$$

Using our definition of ξ, we find that Eq. (III-16) is simply another statement of the diffusional stability condition:

$$\frac{\sigma dt}{dx^2} < \frac{1}{2}\,. \tag{III-17}$$

Our analytical method arrives at precisely the same result as the pictorial analysis; the diffusional stability condition must be met in order to ensure numerical stability.

C. Implicit Calculations

We have now derived the diffusional stability condition mathematically as well as graphically and have demonstrated that it is essential to the numerical solving of Eq. (II-12). It is possible, however, to solve the heat-flow equation numerically without meeting this condition, by expressing the ξ terms of the heat-flow equation at time $n+1$ rather than at time n. In this method, heat flow is not represented by Eq. (II-12), but instead by the following:

$$T_j^{n+1} = T_j^n + \frac{\sigma dt}{dx^2}\left[T_{j+1}^{n+1} + T_{j-1}^{n+1} - 2T_j^{n+1}\right]\,. \tag{III-18}$$

Let us now examine this equation mathematically as we did in Section B. Inserting a similar trial solution and dividing by $Ae^{ikj\,dx}r^n$, we obtain

$$r = 1 + \frac{\xi}{4}\left[e^{ikdx}r + e^{ikdx}r - 2r\right]\,. \tag{III-19}$$

or

$$r = 1 + \frac{\xi}{4}\left[2r\cos kdx - 2r\right]\,, \tag{III-20}$$

which can be algebraically manipulated to obtain

$$r = \frac{1}{1 + \frac{\xi}{2}(1 - \cos kdx)} \,. \tag{III-21}$$

Examining the upper and lower limits for $\cos kdx$, we find that

$$r = 1 \tag{III-22}$$

or

$$r = \frac{1}{1 + \xi} \,. \tag{III-23}$$

Because ξ is always positive, r will be between 0 and 1 in all cases, indicating that our solution will be numerically stable regardless of the value of $\frac{\sigma dt}{dx^2}$. Equation (III-18) will therefore remain stable at any resolution and time step; all that remains is to solve it numerically.

Equation (III-18) represents an IMPLICIT METHOD of calculation. In this method, values at the new time cycle are not directly calculated from old values as they were in the EXPLICIT METHOD used in the previous chapter. They are instead calculated using an iterative process that begins with a trial solution and modifies that solution with each iteration until it has converged to within a specified value. In our program this iteration will be done using Newton's Method.

Newton's Method is an iterative process that uses successive approximations to solve an equation in the form $f(x) = 0$. In Newton's method a trial value for x_1 is first chosen, then the following equation is applied iteratively:

$$x_{n+1} = x_n - \frac{f(x_n)}{\left(\frac{df}{dx}\right)_{x_n}} \,. \tag{III-24}$$

This equation generates successive approximations for x, each more accurate than the one before it. When x has converged to within a specified range, x_n is then taken as the final solution.

We can better understand how this method arrives at a solution by examining an example equation, $f(x) = x^2 - 2$. Choosing $x_1 = 2$ as a trial value, we will let the solution converge to within three decimal places.

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n} \qquad \text{(III-25)}$$

$$x_1 = 2.000$$

$$x_2 = 1.500$$

$$x_3 = 1.466$$

$$x_4 = 1.414$$

$$x_5 = 1.414$$

A plot of these values along the graph of the equation appears in Fig. III-7.

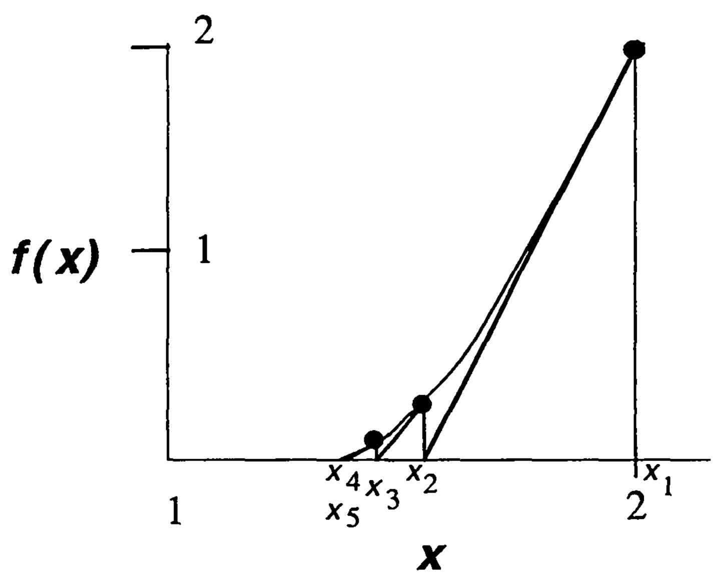

Fig. III-7

This figure illustrates how one solution is used to obtain the next. The tangent is used to approximate the graph of $f(x_n)$ at the value x_n, and x_{n+1} is given the value of

x for which the tangent crosses the x-axis. This value is then substituted for x and the process is repeated until it has converged.

Let us now apply this process to Eq. (III-18). Expressing this equation as a function of T_j^{n+1}, we obtain

$$f\left(T_j^{n+1}\right) = T_j^{n+1} - T_j^n - \frac{\sigma dt}{dx^2}\left[T_{j+1}^{n+1} + T_{j-1}^{n+1} - 2T_j^{n+1}\right] . \qquad \text{(III-26)}$$

Taking now the derivative with respect to T_j^{n+1}, we find that

$$f'\left(T_j^{n+1}\right) = 1 + \frac{2\sigma dt}{dx^2} . \qquad \text{(III-27)}$$

Using both of these definitions in Eq. (III-24), we are left with the final formula:

$$T_j^{n+1}(\text{new guess}) = T_j^{n+1}(\text{old guess}) - \left(\frac{f(T_{j(oldguess)}^{n+1})}{1 + \frac{2\sigma dt}{dx^2}}\right) . \qquad \text{(III-28)}$$

By using this formula iteratively, we can now compute values for T_j^{n+1} for any values of $\frac{\sigma\, dt}{dx^2}$.

D. Computational Implementation of the Implicit Method

Now that we have developed an implicit method for use in solving the heat-transfer equation, we can implement this method on the computer. We will do this by making small modifications to the heat-transfer code that has already been written.

We begin by defining a constant beta that is set during the initialization procedure. Beta is defined as

$$\beta \equiv \frac{1}{1 + \frac{2\sigma dt}{dx^2}}$$

and is used to avoid successive calculation of the denominator in Eq. (III-28) during iterations of Newton's method. Also in this procedure, we define a constant ftest that is equal to the margin of error to which our iterative procedure will converge. Typically ftest has a value of approximately 0.001 times some maximum value of T in the problem.

Besides these two definitions in the initialization procedure, most of the major modifications to the program occur in the computation section (referred to as Section 4 in the previous chapter). This section in its explicit form should be removed and replaced with an implicit section of code.

This implicit section should consist of a loop that makes the initial guesses for the temperatures at time $n+1$ and a loop that iterates until the values of T^{n+1} have converged to within ftest. The first loop is simply a do loop that defines the initial guesses for the new temperatures as the temperatures at the old time step. Thus,

$$\text{Tnew(j)} = \text{T(j)} \; . \tag{III-29}$$

This loop is then followed by an until loop that is constructed in the following manner. At the beginning of each iteration, a value fmax is set at zero. After this statement, the program moves into another loop that calculates $f(T)$ along every point along the rod and uses these values to calculate the next guess for the new temperatures. Also in this loop, the largest absolute value for $f(T)$ is stored in the variable fmax. After this loop, the program makes a check to see if fmax is less than ftest. If ftest is larger, the until loop ends; if fmax is larger, the loop is repeated. The code for this loop should be similar to the following:

```
100  fmax = 0.
       do 200 j= 1,jmax
         f = Tnew(j) - T(j) - (sig*dt/(dx*dx)) * (Tnew(j+1)+ T (j-1) - 2*T(j))
         fmax = amax1(abs(f),fmax)
         Tnew(j) = Tnew(j) - f * beta
200  continue
        if (fmax.gt.ftest) then goto 100
```

Notice that all the T terms on the right of the equation that sets f are actually temperature values at the present implicit iteration. T's and Tnew's are mixed due to the structure of the loop. Optionally, a counter for the number of iterations of the until loop can be added, terminating this iterative process when a maximum number of iterations is reached, regardless of the values of fmax.

When this loop has finally terminated, the T array is redefined with the values from the Tnew array, and the program moves on to the next time step. All other sections remain in the same form as in our original program. No other modifications are necessary to create a fully-implicit version of our one-dimensional heat-transfer code.

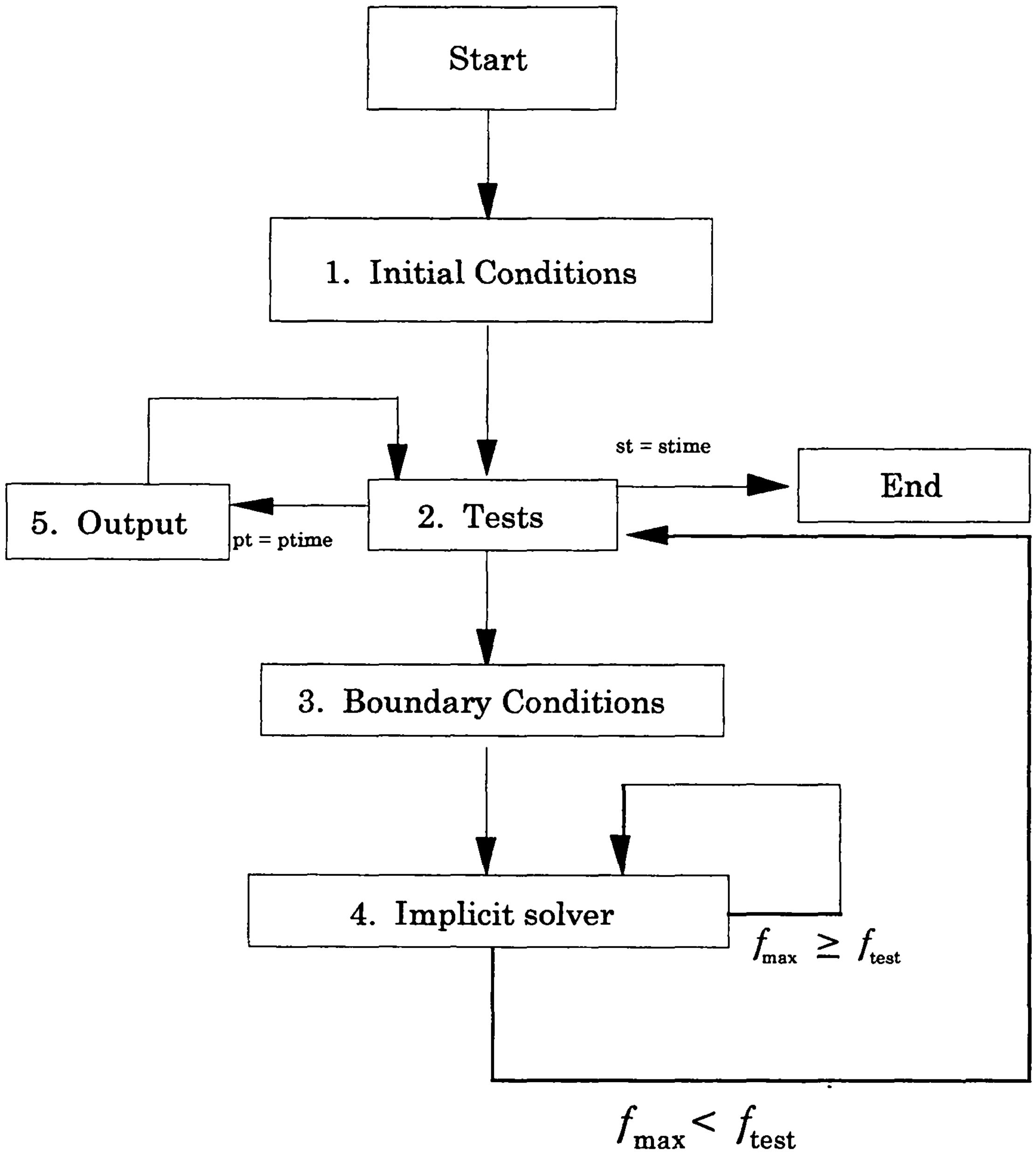

Fig. III-8

By using the implicit code with the same set of parameters as were present in Fig. II-12 ($T_0 = 0°\text{C}$, $T_1 = 400°\text{C}$, $T_r = 0°\text{C}$, $\sigma = 1\,\text{m}^2/\text{sec}$, $dx = 1$ m, $\bar{j} = 50$, $dt = 0.5$ sec, time $= 100$ sec, $\frac{\sigma dt}{\partial x^2} = 0.5$), the results shown in Fig. III-9 are obtained.

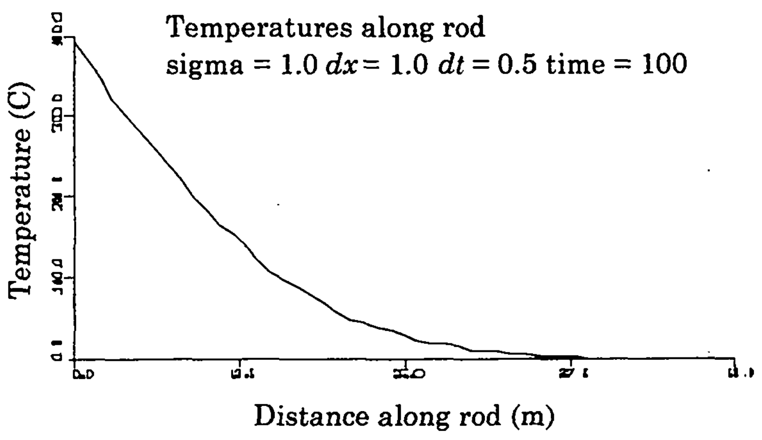

Fig. III-9

This figure helps to illustrate the numerical stability of this method. The system remains stable at a time step of 0.505 $\left(\frac{\sigma dt}{dx^2} = 0.505\right)$, as indicated in Fig. III-10.

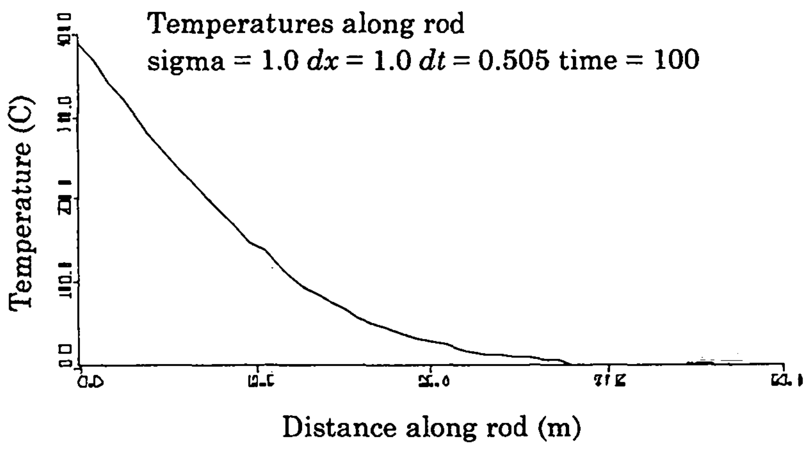

Fig. III-10

Even at a time step of 10 sec, where $\frac{\sigma dt}{dx^2}$ is equal to 10 and only 10 time cycles are computed up to time 100, the system remains numerically stable. The results in Fig. III-11 below appear almost identical to those calculated explicitly in Fig. II-11, yet the time step used is over twenty times as large.

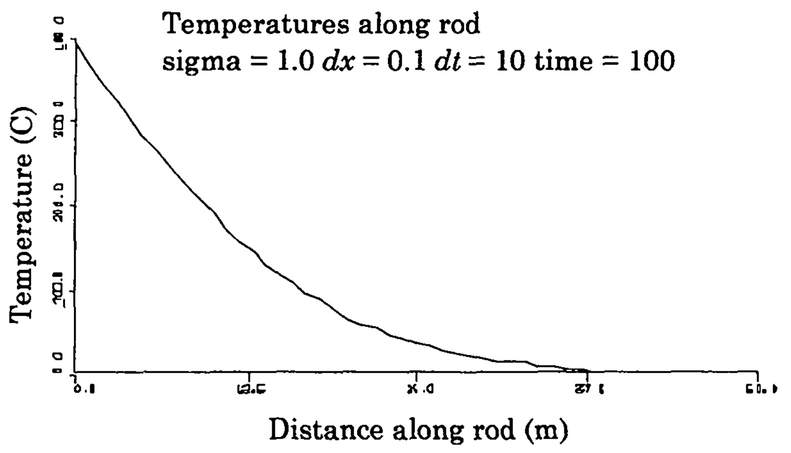

Fig. III-11

We see through example that implicit methods are able to generate results for sets of parameters that are numerically unstable when calculated explicitly. This technique will prove essential in later simulations, preventing the first of two major numerical instabilities that we will examine in our series of exercises.

E. Analytic Solution of the Heat-Flow Equation

In this section, we will be manipulating the heat-flow equation in order to generate an analytic solution that can be used to check the validity of our computational results. Once again, following the manipulation of this partial-differential equation is not essential to making use of the derived solution.

We begin with Eq. (II-19), the one-dimensional heat-flow equation in partial-differential form

$$\frac{\partial T}{\partial t} = \sigma \frac{\partial^2 T}{\partial x^2} \qquad \text{(II-19)}$$

and make the assumption that T is a function of the single dimensionless quantity that includes σ, x, and t:

$$T = T(\xi) \qquad \text{(III-30)}$$

where

$$\xi \equiv \frac{x}{\sqrt{\sigma t}} .$$

By making this definition, we are assuming that the rod is of infinite length, so that the length of the rod does not enter into these parameters. This assumption is made because the derivation of a solution for a finite rod is a much more involved process than a solution for the infinite case. For our purposes an analytic solution to the infinite rod case will prove to be sufficient.

Using Eq. (III-30), we can obtain expressions for its partial-derivatives. Differentiating with respect to t we obtain

$$\frac{\partial T}{\partial t} = \frac{dT}{d\xi}\frac{\partial \xi}{\partial t} = \frac{x}{2\sqrt{\sigma}}\left(-t^{3/2}\right)\frac{dT}{d\xi} . \qquad \text{(III-31)}$$

Differentiating to obtain the second derivative with respect to x gives us

$$\frac{\partial T}{\partial x} = \frac{dT}{d\xi}\frac{\partial \xi}{\partial x} = \frac{1}{\sqrt{\sigma t}}\frac{dT}{d\xi} \qquad \text{(III-32)}$$

and

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{\left(\sqrt{\sigma t}\right)^2}\frac{d^2 T}{d\xi^2} . \qquad \text{(III-33)}$$

Equations (III-31) and (III-33) are substituted into Eq. (II-19) to obtain

$$\frac{x}{2\sqrt{\sigma}}\left(-t^{3/2}\right)\frac{dT}{d\xi} = \frac{\sigma}{\sigma t}\frac{d^2 T}{d\xi^2} \qquad \text{(III-34)}$$

or

$$-\frac{\xi}{2}\frac{dT}{d\xi} = \frac{d}{d\xi}\left(\frac{dT}{d\xi}\right) . \qquad \text{(III-35)}$$

If we define a variable y such that

$$y \equiv \frac{dT}{d\xi} ,$$

Eq. (III-35) becomes

$$-\frac{\xi}{2}y = \frac{dy}{d\xi} \qquad \text{(III-36)}$$

or

$$-\frac{\xi}{2}d\xi = \frac{1}{y}dy \; , \tag{III-37}$$

which can be integrated to obtain

$$-\frac{\xi^2}{4} = \ln y + C \; , \tag{III-38}$$

where C is a constant.

Exponentiating both sides of this equation gives us:

$$e^{\frac{-\xi^2}{4}} = e^{\ln y + C} \tag{III-39}$$

or

$$e^{\frac{-\xi^2}{4}} = yK_1 \; , \tag{III-40}$$

where K_1 is a constant. We then use our definition of y and multiply both sides of the equation by $d\xi$ to obtain

$$e^{\frac{-\xi^2}{4}} d\xi = K_1 dT \; . \tag{III-41}$$

This can be integrated to obtain

$$\int_{\xi_1}^{\xi_2} e^{\frac{-\xi^2}{4}} d\xi = K_1 T + K_2 \; , \tag{III-42}$$

where K_2 is another constant. If we choose ξ_1 to represent ξ at a distance of zero from the end of the rod this equation becomes

$$\int_{0}^{\frac{x}{\sqrt{\sigma t}}} e^{\frac{-\xi^2}{4}} d\xi = K_1 T + K_2 \; . \tag{III-43}$$

which can be simplified by defining a variable z such that

$$z \equiv \frac{\xi}{2} \; .$$

Equation (III-43) then becomes

$$2\int_0^{\frac{x}{2\sqrt{\sigma t}}} e^{-z^2}dz = K_1T + K_2\ . \tag{III-44}$$

To determine the values K_1 and K_2, we examine two test cases. In the case where x is 0, the temperature is equal to that of the wall at the left end of the rod. We then have:

$$2\int_0^0 e^{-z^2}dz = K_1T_L + K_2 \tag{III-45}$$

or

$$K_2 = -(K_1T_L)\ . \tag{III-46}$$

In the case where we are at an infinite distance from the heat source at one end of the rod, the temperature is equal to the initial temperature specified for the rod:

$$2\int_0^\infty e^{-z^2}dz = K_1T_0 + K_2\ , \tag{III-47}$$

which reduces to

$$2\left(\frac{\sqrt{\pi}}{2}\right) = K_1T_0 + K_2 \tag{III-48}$$

or

$$\sqrt{\pi} - K_1T_0 = K_2\ . \tag{III-49}$$

Setting Eqs. (III-46) and (III-49) equal to each other, we obtain

$$-K_1T_L = \sqrt{\pi} - K_1T_0 \tag{III-50}$$

or

$$K_1 = \frac{\sqrt{\pi}}{(T_0 - T_L)}\ . \tag{III-51}$$

K_2 can then be obtain by substituting into Eq. (III-46):

$$K_2 = -\frac{T_L\sqrt{\pi}}{(T_0 - T_L)}\ . \tag{III-52}$$

Substituting both of these values into Eq. (III-44) we obtain

$$2\int_0^{\frac{x}{2\sqrt{\sigma t}}} e^{-z^2} dz = (T - T_L)\left(\frac{\sqrt{\pi}}{(T_0 - T_L)}\right) \tag{III-53}$$

or

$$T = T_L + (T_0 - T_L)\left(\frac{2}{\sqrt{\pi}}\right)\int_0^{\frac{x}{2\sqrt{\sigma t}}} e^{-z^2} dz\,. \tag{III-54}$$

$\left(\frac{2}{\sqrt{\pi}}\int_0^{\frac{x}{2\sqrt{\sigma t}}} e^{-z^2} dz\right)$ in this equation is a form of the PROBABILITY INTEGRAL, also called the ERROR FUNCTION. This term is not integrable in terms of simple polynomials, but it can be "solved" by defining

$$erf(a) \equiv \frac{2}{\sqrt{\pi}}\int_0^{a} e^{-z^2} dz\,,$$

where $erf(a)$ can be determined as in the following table.

$erf(a)$

a	$erf(a)$	a	$erf(a)$
0.00	0.0000	0.9	0.7969
0.05	0.0564	1.0	0.8427
0.10	0.1125	1.1	0.8801
0.15	0.1680	1.2	0.9103
0.20	0.2227	1.3	0.9340
0.25	0.2763	1.4	0.9523
0.30	0.3286	1.5	0.9661
0.35	0.3794	1.6	0.9764
0.40	0.4283	1.7	0.9838
0.45	0.4755	1.8	0.9891
0.50	0.5205	1.9	0.9928
0.55	0.5633	2.0	0.9953
0.60	0.6039	2.1	0.9970
0.65	0.6420	2.2	0.9981
0.70	0.6778	2.3	0.9987
0.75	0.7112	2.4	0.9994
0.80	0.7421	2.5	0.9996

Our final solution to the heat-flow equation is then

$$T = T_L + (T_0 - T_L)\, erf\left(\frac{x}{2\sqrt{\sigma t}}\right) . \qquad \text{(III-55)}$$

This equation can be used to check the accuracy of our numerical results. It represents the infinite rod case in which the temperature wave is not affected by the conditions at the right end of the rod. At early time steps, the temperatures in our finite-rod simulation should approximate those generated by this equation. Solutions at late time steps, as we saw in Chapter II, should approach a straight line. By examining the results generated by our code in both these circumstances, we can verify the validity of our finite-difference calculations.

IV. LAGRANGIAN FLUID DYNAMICS

A. Fluid Flow and Lagrangian Methods

Up to this point our finite-difference codes have dealt strictly with the equation of heat transfer [Eq. (II-19)], but heat flow is only one of many phenomena that can be modeled using the finite-difference method. In the following several chapters, we will be looking at another physical phenomenon that can be simulated in this manner: the motion of FLUIDS.

For our purposes, we will define a fluid as anything that is infinitely deformable or malleable. This means that, while a fluid may resist moving from one shape to another, it resists the same amount in all directions and in all shapes. Fluids can be either COMPRESSIBLE or INCOMPRESSIBLE. An incompressible fluid is one that does not change its density much when pressure is applied to it, meaning that the fluid is moving at a velocity much less than its sound speed. A compressible fluid is one that undergoes a large change in its density as pressure is applied to it, meaning that the fluid is moving at a speed that is comparable to its sound speed. We will be dealing with compressible fluids in this chapter.

Our simulation will be of a system that can be reduced to one dimension: a piston moving in a long cylinder that is filled with gas. The compression of gas in this manner can by dealt with in one of two ways: through LAGRANGIAN or EULERIAN methods.

In an Eulerian code, zones remain fixed in space throughout the simulation. Fluids move in and out of the zone at various rates, causing the mass contained in a particular zone to change as the simulation progresses. All physical quantities are fluxed between cells, but the position of the cells at all time steps remains the same. We will examine this method in Chapter V.

Another method for simulating this situation is the Lagrangian technique. In a Lagrangian code the positions of zones vary between each time step. As fluids are compressed and decompressed, the zones move accordingly, maintaining an equal mass throughout the simulation. In a Lagrangian calculation, the energy, momentum, and position of a given zone vary from time step to time step; only the mass contained by the zone is held fixed. The Lagrangian technique is the one that is used in this chapter.

B. Description of Equations Used in Lagrangian Fluid Flow

In order to derive the equations that are used in a one-dimensional Lagrangian code, we must first define a group of variables and coordinates similar to those used in our first two simulations. We again have a one-dimensional system of zones, each zone representing a certain section of the system being simulated. The system appears as in Fig. II-3, with a series of $\bar{j}$ true zones and ghost zones appearing at 0 and $\bar{j}+1$

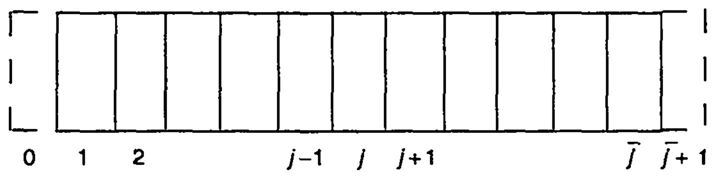

Figure IV-1

The variables that will be applied to this system, however, are quite different from those of the heat-transfer problem. In the fluid-flow case there is no longer a single array of temperatures, but instead a group of arrays that represent position, pressure, velocity, density, internal energy, and viscous pressure. The definition of these variables over a zone j is shown in Fig. IV-2.

In this figure:

$x_{j+1/2} \equiv$ position of cell wall to the right of zone j

$u_{j+1/2} \equiv$ velocity at cell wall to the right of zone j

$p_j \equiv$ pressure of zone j

$I_j \equiv$ internal energy per unit mass of zone j

$q_j \equiv$ viscous pressure of zone j

$\rho \equiv$ density of zone j

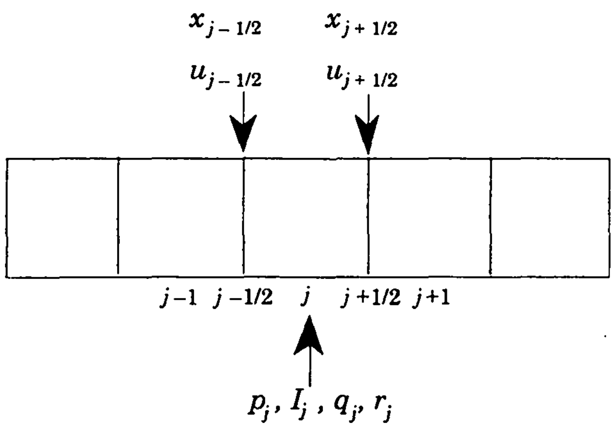

Figure IV-2

Note that u and x are located at the walls of the cells while the rest of the variables are located at the centers. These positions will be important in determining the relationship among these various quantities.

With our variables defined as in Fig. IV-2, the equations that relate them to one another can be derived. Consider the relationship between x and u: x is the array of wall positions and u is the array of the time rate of change of those wall positions, i.e., velocity. From these definitions, we see that

$$\frac{\partial x}{\partial t} = u \, . \tag{IV-1}$$

Finite-differencing this equation gives us

$$\frac{x^{n+1}_{j+1/2} - x^{n}_{j+1/2}}{dt} = u^{n}_{j+1/2} \, , \tag{IV-2}$$

which can be rewritten as an equation for position in terms of velocity:

$$x^{n+1}_{j+1/2} = x^{n}_{j-1/2} + u^{n}_{j+1/2} dt \, . \tag{IV-3}$$

This is the first important equation of our Lagrangian fluid flow code.

The next equation follows from Newton's second law of motion, (Force = Mass × Acceleration), and the definition of pressure, (pressure $\equiv \frac{\text{Force}}{\text{area}}$). In our code we define a

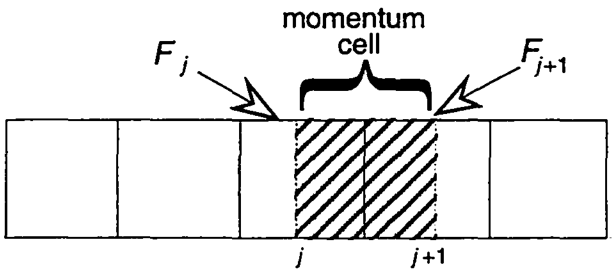

Figure IV-3

momentum cell whose center lies at the boundary between two normal cells and define F_j and F_{j+1} as the force at the right and the left of the j momentum cell. A momentum cell is depicted in Fig. IV-3.

By the definition of pressure, F_j and F_{j+1} are rewritten in terms of variables defined in Fig. IV-2:

$$F_j = (p_j + q_j)\, A \tag{IV-4}$$

$$F_{j+1} = (p_{j+1} + q_{j+1})\, A\ , \tag{IV-5}$$

where A is the surface area of a cell wall. Note that here we use the sum of the physical pressure and the viscous pressure, an additional pressure that is necessary to achieve numerical stability. The viscous pressure will be discussed in more detail in section C.

Using Newton's second law and noting that acceleration is the time rate of change of velocity, we obtain

$$F_{j+1/2} = m \left(\frac{u_{j+1/2}^{n+1} - u_{j+1/2}^{n}}{dt} \right)\ , \tag{IV-6}$$

where $F_{j+1/2}$ is equal to the net force at the cell wall $j + 1/2$. In the case where there are no outside forces such as gravity in the x-direction, the net force at $j + 1/2$ is equal to the force pushing the momentum cell from the left (F_j) minus the force by which it pushes the next momentum cell on the right (F_{j+1}):

$$F_{j+1/2} = F_j - F_{j+1} \ . \tag{IV-7}$$

Combining Eq. (IV-6) and Eq. (IV-7) gives us

$$F_j - F_{j+1} = m \left(\frac{u^{n+1}_{j+1/2} - u^n_{j+1/2}}{dt} \right) , \tag{IV-8}$$

or, by substituting for F_j and F_{j+1},

$$(q_j + p_j - q_{j+1} - p_{j+1}) \, A = m \left(\frac{u^{n+1}_{j+1/2} - u^n_{j+1/2}}{dt} \right) . \tag{IV-9}$$

Defining the quantity M as $\frac{m}{A}$ and solving for $u^{n+1}_{j+1/2}$, we are left with the expression

$$u^{n+1}_{j+1/2} = u^n_{j+1/2} + \frac{dt}{M} \left(p^n_j + q^n_j - p^n_{j+1} - q^n_{j+1} \right) \tag{IV-10}$$

which gives change in u in terms of variables used in Fig. IV-2.

An expression for ρ can be obtained by again using our definition of M. Because density is equal to M divided by the width of a zone, it follows that

$$\rho^n_j = \frac{M}{x^n_{j+1/2} - x^n_{j-1/2}} , \tag{IV-11}$$

which is the Lagrangian density equation.

An expression for I, the internal energy per unit mass of a cell, can be derived by appealing to the definition of internal energy. I can be defined as the difference between the total energy per unit mass and the kinetic energy per unit mass contained in a cell, which is the same as the heat energy per unit mass. We can therefore use the first law of thermodynamics,

$$\Delta I = \Delta E = Q - p \Delta V , \tag{IV-12}$$

where ΔE is the change in heat energy, Q is the heat received from both conduction from an outside source (which will not be important in this simulation) and dissipation of mean flow kinetic energy, p is the pressure, and ΔV is the change in volume.

We now make use of the variable q, which we called the viscous pressure earlier in this chapter. One way of thinking of this pressure is as $-Q/\Delta V$, or the increase in heat energy over the compressive (or negative) change in volume. Using q, Eq. (IV-12) can be written as

$$\Delta I = -(q+p)\Delta V \ . \tag{IV-13}$$

Because the total internal energy is equal to $m \times I$, this equation can be rewritten in the following differential form:

$$m\frac{\partial I}{\partial t} = -(q+p)\frac{\partial V}{\partial t} \ . \tag{IV-14}$$

In finite-difference form, Eq. (IV-14) becomes

$$m\left(\frac{I_j^{n+1} - I_j^n}{dt}\right) = -(q+p)A\left[\frac{\partial x_{j+1/2}}{\partial t} - \frac{\partial x_{j-1/2}}{\partial t}\right] \ . \tag{IV-15}$$

Using our definition of M as $\frac{m}{A}$ and $u_{j+1/2}$ as $\frac{\partial x_{j+1/2}}{\partial t}$, we solve for I_j^{n+1} and obtain the Lagrangian internal energy equation in finite-difference form:

$$I_j^{n+1} = I_j^n + \frac{dt}{M}\left(q_j^n + p_j^n\right)\left(u_{j-1/2}^n - u_{j+1/2}^n\right) \ . \tag{IV-16}$$

To obtain an equation for pressure (p), we employ the ideal gas law, namely

$$pV = n\mathrm{R}T \ , \tag{IV-17}$$

where p is the pressure, V is volume, n the number of moles, T the temperature, and R is the universal gas constant. In SI units R = 8.3145 J/°K mol

In an ideal gas it can be shown that when C_p and C_v are the molar heat capacities at constant pressure and constant volume,

$$C_p - C_v = R \ . \tag{IV-18}$$

Combining this equation with Eq. (IV-17) and solving for p gives

$$p = \frac{n}{V}(C_p - C_v)T \ , \tag{IV-19}$$

which can be rewritten as

$$p = \frac{1}{V}\left(\frac{C_p}{C_v} - 1\right) nC_vT \, . \tag{IV-20}$$

Because C_v is the molar specific heat and I is the internal energy per unit mass,

$$nC_vT = mI \, . \tag{IV-21}$$

By use of this equation, Eq. (IV-20) becomes

$$p = \frac{1}{V}\left(\frac{C_p}{C_v} - 1\right) mI \, . \tag{IV-22}$$

We now define a constant γ such that

$$\gamma \equiv \frac{C_p}{C_v} \, .$$

This variable is called the POLYTROPIC GAS CONSTANT. This constant is always greater than one and represents the ratio of specific heats and is a property of the gas being simulated. Some typical values of γ are

air, $\gamma = 1.4$

helium, $\gamma = 1.66$

CO_2, $\gamma = 1.34$

SF_6, $\gamma = 1.08$

γ and ρ are used in Eq. (IV-22) to obtain the Lagrangian finite-difference equation for pressure:

$$p_j^n = (\gamma - 1)\rho_j^n I_j^n \, . \tag{IV-23}$$

This equation, also known as the POLYTROPIC EQUATION OF STATE, is used to calculate values of p at each time step.

With pressure defined, let us take a closer look at q, called the viscous pressure. This variable accounts for loss of kinetic energy in addition to what is used to compress the gas. It serves as a means by which kinetic energy is dissipated in irreversible processes in the fluid such as the creation of heat through friction. The equation for artificial viscous pressure appears in the following form:

$$q_j^n = q_o \rho_j^n c \left(u_{j-1/2}^n - u_{j+1/2}^n\right) \qquad \text{if positive}$$

$$\text{or if negative} \qquad q_j^n = 0 \, , \tag{IV-24}$$

where q_0 is a constant between 0.1 and 0.25, and c is a characteristic velocity of the system.

This equation will not be derived in this work, but it is important to understand why it appears in this form. The irreversible processes that are modeled through the use of the q equation occur when there is a rapid change in the volume occupied by a gas in a system. In our system this change occurs when there is a large differential between the velocities at the left and right of a given cell. Hence we make q proportional to $u_{j-1/2}^n - u_{j+1/2}^n$, indicating a large amount of kinetic energy dissipated when there is a large velocity differential and a small amount dissipated when there is a lesser difference in velocities. Because dissipated kinetic energy is never returned to the system, this term is said to have a value of zero when its computed value is negative.

In order to be dimensionally correct $u_{j-1/2}^n - u_{j+1/2}^n$ is multiplied by the density of the zone and by c, which is called the characteristic velocity. As c's purpose is simply to make our equation dimensionally correct, we have some leeway in choosing this velocity. Typically it is chosen in one of three ways. The simplest method is to define c as equal to the value of some other major velocity in the simulation. In our simulation this would be the velocity of the piston that compresses the gas in the cylinder. Another way that this velocity can be defined is by using the sound speed of the fluid in question. Namely

$$c = \sqrt{\frac{\gamma p}{\rho}} \, , \tag{IV-25}$$

which can be rewritten using our equation for pressure as

$$c = \sqrt{\gamma(\gamma - 1)I} \, . \tag{IV-26}$$

The third way in which c can be determined is by combining these two approaches by adding the sound speed to a prevalent velocity in the problem. In this case

$$c = \text{piston speed } + \sqrt{\gamma(\gamma - 1)I}\,. \tag{IV-27}$$

This third method is the one used in the model presented in this work.

In this section transport equations for position, velocity, density, internal energy, pressure, and viscous pressure were derived—all the equations necessary to construct a one-dimensional compressible fluid-flow simulation. The equations of fluid flow, particularly those of density (or continuity), internal energy, and velocity (or momentum), are often called the NAVIER-STOKES EQUATIONS. This is a general term that can be used to represent any set of fluid-flow equations in 1, 2, or 3 dimensions. Having derived these equations, we are ready to move our discussion to the construction of the computer code itself; but before we take this step, let us first take a closer look at the artificial viscous pressure. Its relationship with the diffusion equation will help us derive a stability requirement that will be important in this simulation.

C. Viscous Pressure and Diffusion

Let us examine the effect of q on the momentum equation [Eq. (IV-10)]. Substituting our definition of q [Eq. (IV-24)] into the momentum equation, while dropping the p's and the subscripts on ρ, gives us

$$u_{j+1/2}^{n+1} = u_{j+1/2}^{n} + \frac{dt}{M}\left(q_o\,\rho\,c\left(u_{j-1/2}^{n} - u_{j+1/2}^{n}\right) - q_o\,\rho\,c\left(u_{j+1/2}^{n} - u_{j+3/2}^{n}\right)\right)\,, \tag{IV-28}$$

which can be rewritten

$$u_{j+1/2}^{n+1} = u_{j+1/2}^{n} + \frac{q_o\,\rho\,c\,dt}{M}\left(u_{j-1/2}^{n} + u_{j+3/2}^{n} - 2u_{j+1/2}^{n}\right)\,. \tag{IV-29}$$

This equation is in the form of a diffusion equation [Eq. (III-18)] where

$$\sigma = \frac{q_0\,\rho\,c\,dx^2}{M}\,. \tag{IV-30}$$

We can use our definition of ρ

$$\rho = \frac{m}{A dx} = \frac{M}{dx} , \qquad \text{(IV-31)}$$

to rewrite Eq. (IV-30) as

$$\sigma = q_0 \, c \, dx . \qquad \text{(IV-32)}$$

Using the diffusional stability condition on σ [Eq. (III-6)],

$$\frac{\sigma \, dt}{dx^2} < \frac{1}{2} , \qquad \text{(III-6)}$$

we find that

$$\frac{q_0 \, c \, dt}{dx} < \frac{1}{2} . \qquad \text{(IV-33)}$$

This important stability requirement arises from the parallelism between the effect of q on the momentum equation and the equation of heat diffusion. It is the first of two major stability conditions that are found in this code.

The second condition, the COURANT STABILITY CONDITION, is a stability condition that occurs as a result of a numerical instability that will be discussed in Chapter VI. Its presence in a Lagrangian simulation can be explained by using a simple example.

Consider the Fig. IV-4, where the fluid at the left wall of a zone of length dx is moving to the right with velocity v, while the fluid at the right wall is stationary.

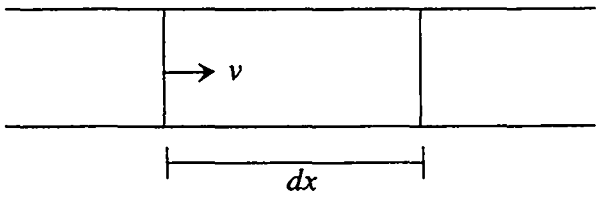

Figure IV-4

In this case, we see that for a given time step (dt), the left wall will move toward the right wall a distance $v \, dt$. In order for our Lagrangian simulation to remain valid, the left wall cannot be allowed to move past the right wall. Mathematically the system must satisfy the following equation:

$$v\,dt < dx\,. \tag{IV-34}$$

Because dx is always positive, this equation can also be written as

$$v\frac{dt}{dx} < 1\,. \tag{IV-35}$$

Since cross-overs can also occur when a right wall moves leftward past a left wall, our final stability condition is

$$|v|\,\frac{dt}{dx} < 1\,. \tag{IV-36}$$

In our code, v is equal to the maximum velocity in the system $|u| + c$. Equation (IV-36) is the Courant stability condition. It will be discussed in greater detail later on in this work. In this chapter, we need only note its restrictions when we represent our system of equations computationally.

D. Computational Lagrangian Fluid Flow

Using the equations derived in Section B, we can begin writing our one-dimensional Lagrangian compressible fluid-flow code. As was true in the heat code, we must define our variables in FORTRAN terms before we discuss their use in the code itself. Our variable names appear in code form as follows: $p_j \equiv$ p(j), $q_j \equiv$ q(j), $I_j \equiv$ sie(j), $\rho_j \equiv$ rho(j), $u_{j+1/2} \equiv$ u (j), $x_{j+1/2} \equiv$ x(j), $u_{j-1/2} \equiv$ u(j-1), and $x_{j-1/2} \equiv$ x(j-1). Note that x and u, while still defined at $j + 1/2$ and $j - 1/2$, are written as x(j), u(j), x(j-1), and u(j-1). For both of these variables an array index of j indicates a value at position $j + 1/2$, the wall directly to the right of cell j.

Our program is structured similarly to the heat flow problem illustrated in Fig. II-5. The code exists in five main sections: an initialization routine, a section for time checks and incrementation of counters, the definition of boundary conditions, the updating of variable values, and an output procedure.

First, let us examine our initialization procedure, which defines all the initial values necessary for the problem. Just as in the last simulation, this procedure is used to initialize time counters and define the length of the system being simulated, but this procedure must also set values which were not present in our last case.

It defines the constants:

q0	as occurs in Eq. (IV-24)
gamma	the ratio of specific heats
M	mass/area
ul	the velocity at the left end of the cylinder
ur	the velocity at the right end of the cylinder

Also initial values must be assigned to the constants:

rho0	the initial zone density
sie0	the initial zone internal energy
u0	the initial zone velocity

A loop such as the one that assigned temperatures in the previous problem must be constructed to initialize all the real elements of arrays rho as rho0, sie as sie0, and u as u0, and to compute initial values for x, p, and q using Eqs. (IV-3), (IV-23), and (IV-24), respectively.

After the initialization procedure, the program moves into the same sort of loop as did the heat program: making checks, incrementing time counters, updating boundary conditions, and updating the arrays. The time check portion of the loop is exactly the same as in our last two codes and can be written by repeating what was discussed in Chapter II. The boundary conditions and array assignments are also quite similar to those of our first program but must now be modified to deal with a group of arrays as opposed to a single array of temperatures.

The boundary conditions must be defined for each variable that is referenced at the j = 0 or j = jbar+1 positions. We can determine which variables are referenced in these positions by referring to the equations that define our variable values: Eqs. (IV-3), (IV-10), (IV-11), (IV-16), (IV-23), and (IV-24).

From Eq. (IV-11), which appears in code form as

$$\text{rho(j)} = \text{M}/(\text{x(j)} - \text{x(j} - 1)) \ , \qquad \text{(IV-37)}$$

we see that x(0) will be referenced, indicating the need for the position at the left of the system to be prescribed in our boundary conditions. By similar analysis of Eqs. (IV-16) and (IV-24), we see that there is also a need for values to be determined for $u_{-1/2}$ and $u_{\bar{j}+1/2}$, and for this reason boundary conditions must be assigned to u(0) and u(jbar), representing $u_{-1/2}$ and $u_{\bar{j}+1/2}$, respectively.

An examination of Eq. (IV-10) might lead the reader to believe that boundary conditions are also required for p and q at position jbar+1. This requirement would be true if the wall at the right of the cell were not prescribed, indicating a u(jbar) that is determined independently of p and q. The only three variables that must be modified to establish our boundary conditions are x(0), u(0), and u(jbar).

These variables should be assigned according to the system we wish to represent. For the piston problem, the wall at the right is stationary, indicating that u(jbar) = ur. The velocity at the leftmost cell wall in this problem is equal to the velocity of our simulated piston, u(0) = ul. The position of the leftmost cell wall is equal to its position at the old time step plus the distance that it moves to the right during the new time step, x(0) = x(0) + dt * ul.

In our boundary conditions, we set a value of u(jbar) and not u(jbar+1). This value may seem strange to the student, as it does not make use of a ghost zone but rather modifies a real value in the array. It is allowed because u(jbar) itself exists at a boundary, representing the velocity at the wall directly to the left of zone jbar. In effect, a ghost zone is being modified in which u(jbar) defines the rightmost wall.

With the boundary conditions updated, the code then moves into the updating portion of the program. This is an explicit procedure, which does not use the nested loop structure that was employed in the previous chapter. We are once again dealing with a single loop that assigns new array values based on the values at the previous time step. However, this time we are not dealing with a single array of temperatures but a series of interdependent arrays.

This change presents a problem that was not present in the previous simulation, namely that of updating the values of the variables in an order such that all terms defined at time n in an equation exist at the same time step. This problem is a more complicated version of the one that caused us to create a *Tnew* array in Chapter II. Now we are not only concerned that the terms of a single array exist at the same time step, but that the values of a group of arrays be updated in an order such that each variable is calculated using values from appropriate time steps.

To understand how this order is determined, let us first list our six equations in pseudo-code format. All variables are expressed as they would be in FORTRAN with the exception of the superscripts which are used to remind the reader of the time step at which each of these terms exists. In this form, Eq. (IV-3) becomes

$$\mathrm{x}^{n+1}(\mathrm{j}) = \mathrm{x}^{n}(\mathrm{j}) + \mathrm{u}^{n}(\mathrm{j})\ \mathrm{dt}. \tag{IV-38}$$

Equation (IV-10) becomes

$$\mathrm{u}^{n+1}(\mathrm{j}) = \mathrm{u}^{n}(\mathrm{j}) + (\mathrm{dt/M}) * (\mathrm{p}^{n}(\mathrm{j}) + \mathrm{q}^{n}(\mathrm{j}) - \mathrm{p}^{n}\ (\mathrm{j+1}) - \mathrm{q}^{n}(\mathrm{j+1})). \tag{IV-39}$$

Equation (IV-11), written at the new time step, is

$$\mathrm{rho}^{n+1}\ (\mathrm{j}) = \mathrm{M}\ (\mathrm{x}^{n+1}(\mathrm{j}) - \mathrm{x}^{n+1}(\mathrm{j-1})\ ). \tag{IV-40}$$

Equation (IV-16) becomes

$$\mathrm{sie}^{n+1} = \mathrm{sie}^{n}(\mathrm{j}) + (\mathrm{dt/M}) * (\mathrm{q}^{n}(\mathrm{j}) + \mathrm{p}^{n}(\mathrm{j})) * (\mathrm{u}^{n}(\mathrm{j-1}) - \mathrm{u}^{n}(\mathrm{j})). \tag{IV-41}$$

Equation (IV-23) at the new time step is

$$\mathrm{p}^{n+1}(\mathrm{j}) = (\mathrm{gamma} - 1) * \mathrm{rho}^{n+1}\ (\mathrm{j}) * \mathrm{sie}^{n+1}(\mathrm{j}). \tag{IV-42}$$

Equation (IV-24) also at the new time step is

$q^{n+1}(j) = q0 * rho^{n+1}(j) * c^{n+1} * (u^{n+1}(j-1) - u^{n+1}(j))$

if ($q^{n+1}(j)$. lt. 0) $q^{n+1}(j) = 0.0$, (IV-43)

where c^{n+1} is computed at cell j as in Eq. (IV-27).

The equations are placed in an order such that each variable exists at an appropriate time step when it is used. For example, both x and u must exist at time step n when Eq. (IV-38) is implemented, so this equation must appear before Eq. (IV-39). By similar argument, Eq. (IV-39), which includes a p term at time n, must appear before Eq. (IV-42), which updates p. Further examination of equations in this manner leaves us with a final order in which these equations must be placed, namely, Eq. (IV-38), Eq. (IV-41), Eq. (IV-39), Eq. (IV-40), Eq. (IV-42), and Eq. (IV-43), where the third and fourth are interchangeable, as well as the fifth and sixth. The variable updating portion of the program is a loop that implements these transport equations in an appropriate order.

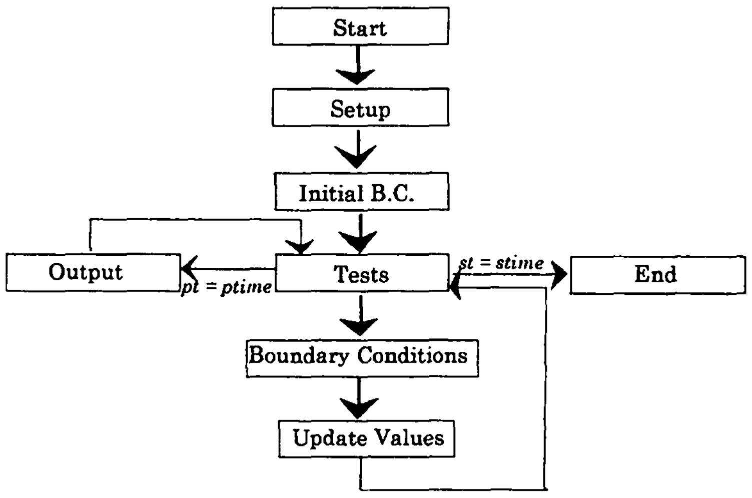

Figure IV-5

With these four sections completed, all that remains is to construct an output procedure desirable to the user, and the one-dimensional Lagrangian fluid code is complete. A graphical representation of this code appears in Figure IV-5.

E. Shocks and Shock Tubes

The following five figures are plots of the density of the fluid in the cylinder as the piston moves in from the left. The parameters chosen for this simulation are: length = 10.(cm), ul = 0.5(cm/s), ur = 0.0(cm/s), gamma = 5/3, jbar=20, rho0 = 1.0 (g/cm^3), sie0 = 0 (cm^2/s^2), q0= 0.3, and dt = 0.05(s). Plots appear at times of 2, 4, 6, 8, and 10 seconds respectively.

Densities at Time 2 (s)

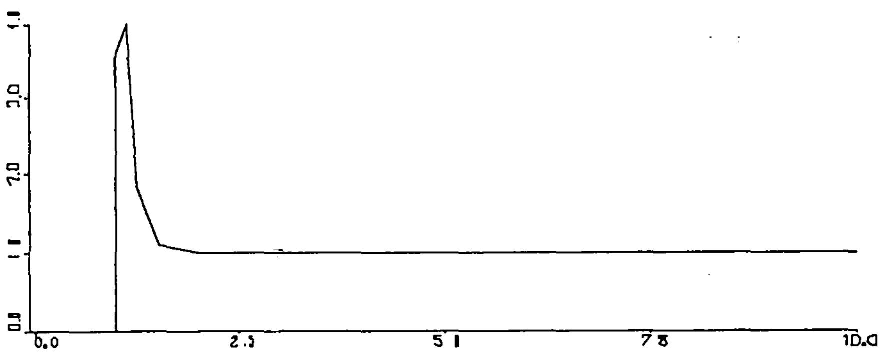

Figure IV-6

Densities at Time 4 (s)

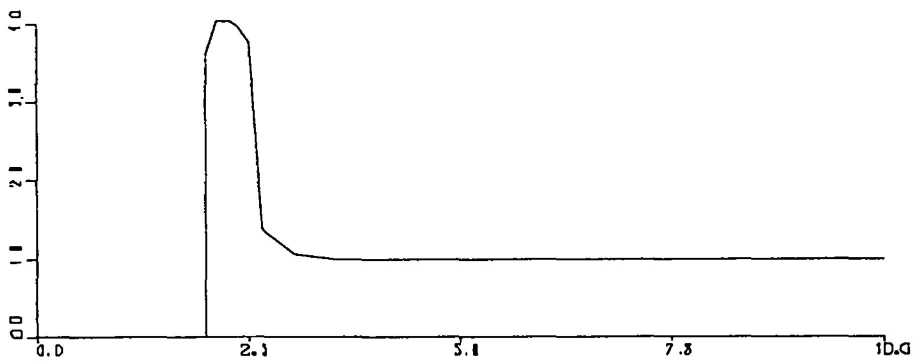

Figure IV-7

Densities at Time 6 (s)

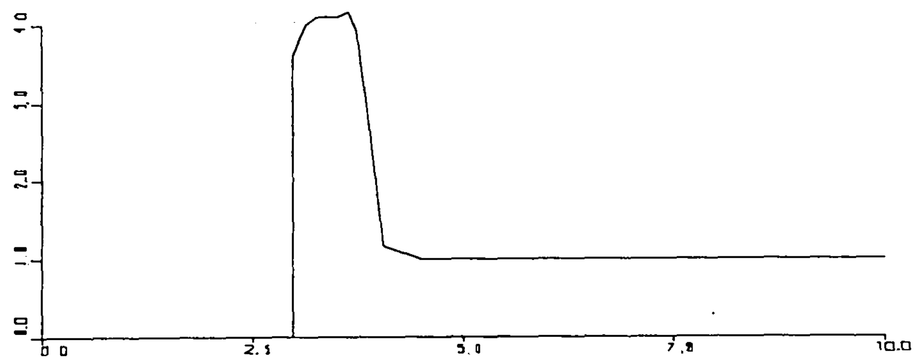

Figure IV-8

Densities at Time 8 (s)

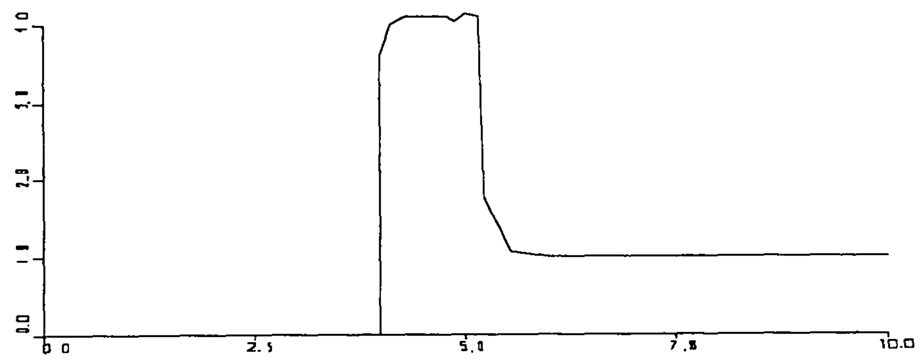

Figure IV-9

Densities at Time 10 (s)

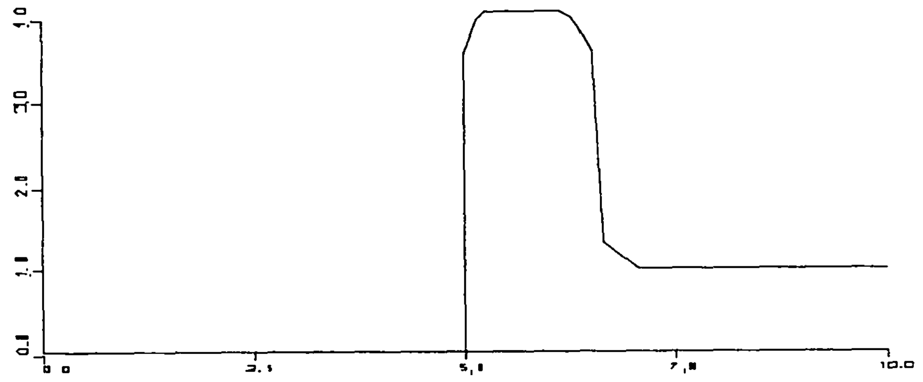

Figure IV-10

The phenomenon that we are examining in these plots is known as a SHOCK, a rapid transition between two states that moves relative to the fluid. Weak shocks occur when fluid is moved at low speeds, but the effects of shocks are most notable when a fluid is moved at a velocity that is near to or greater than the sound speed of the fluid. A shock can be visualized by using the analogy of an evenly spaced line of billiard balls.

Consider the case in which a narrow channel has a piston at one end and is filled with evenly spaced billiard balls, as depicted in Fig. IV-11.

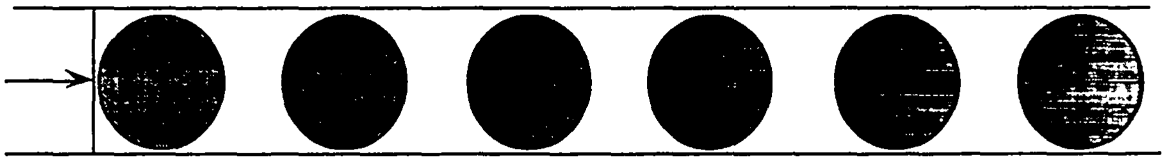

Figure IV-11

The leftmost ball is pushed to the right by the piston and begins to pile up balls in front of it as it moves down the passage. This movement creates a region in which billiard balls exist at a much higher density than in the rest of the passage, because the billiard balls that are moving are touching each other whereas the stationary ones are still evenly spread apart. The front of this compressed region (called the shock or SHOCK FRONT) moves forward faster than the piston itself because billiard balls are constantly piling up in front of the piston as it moves to the right. This system is illustrated in Fig. IV-12. Note that in this figure the transition from the compressed region to the undisturbed surroundings is virtually instantaneous, and that the shock front is not a gradual change in density but rather takes place over a very narrow span.

Our Lagrangian plots demonstrate this sharp contrast between compressed and uncompressed fluid that occurs in a shock. In these plots we can also see the compressed region expanding and the shock front moving at a velocity that is greater than that of the piston. The velocity of the shock front can be predicted, as can other properties,

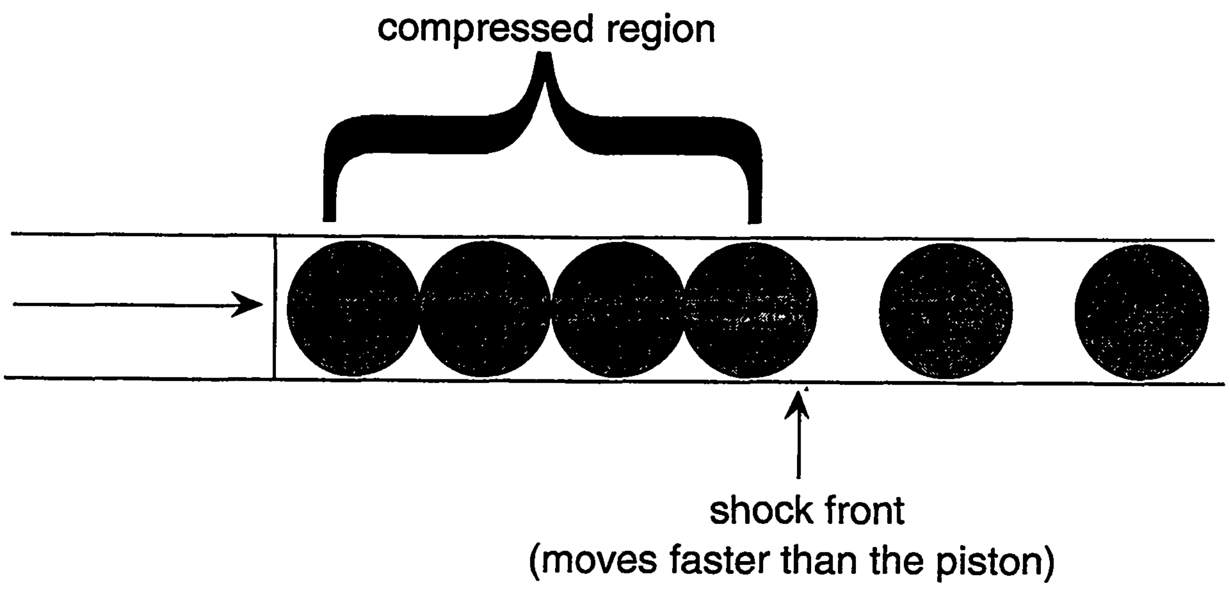

Figure IV-12

by appealing to the equations that describe the theory of shocks. In particular, we will be using the equations of INFINITE STRENGTH SHOCKS, shocks that occur when the shock speed is large compared to the sound speed ahead.

These equations will not be derived in this work, but such derivations are available in various textbooks and monographs, specifically in "Fluid Dynamics—A LASL Monograph" by Francis Harlow and Anthony Amsden, LA-4700. In an infinite strength shock, these equations are

$$u_s = \frac{\gamma + 1}{2} u_p \ , \tag{IV-44}$$

where u_s is the velocity of the shock, and u_p is the velocity of the piston, and

$$\rho_s = \frac{\gamma + 1}{\gamma - 1} \rho_0 \ , \tag{IV-45}$$

where ρ_s the density behind the shock, and ρ_0 is the initial density of the fluid.

Applying these equations to the parameters used in our simulation, we predict that the shock will move forward at a speed of 0.66 (cm/s), and produce a compressed region of density 4 (g/cm^3). These two values can be used to verify the results presented in Figs. IV-6 through IV-10.

The next two graphs illustrate the effect of $q0$ on the accuracy of our numerical simulations. If $q0$ is chosen too low, the answer becomes numerically unstable, as is illustrated in Fig. IV-13.

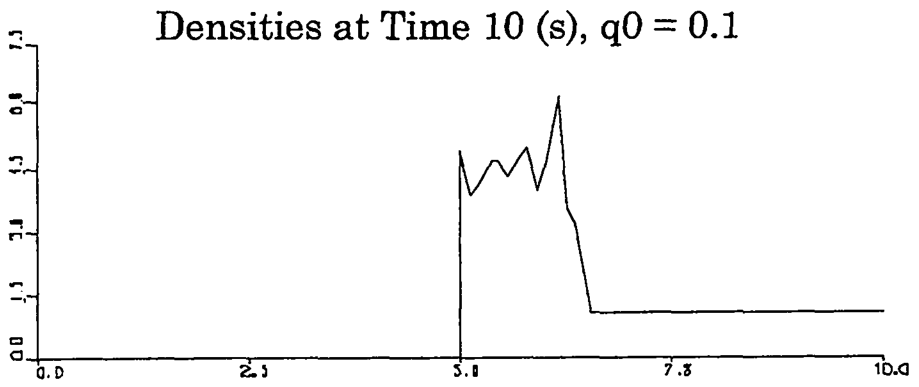

Figure IV-13

If $q0$ is too high, on the other hand, the answer is stable but inaccurate, losing the degree of clarity that was present in the $q0 = 0.3$ graphs.

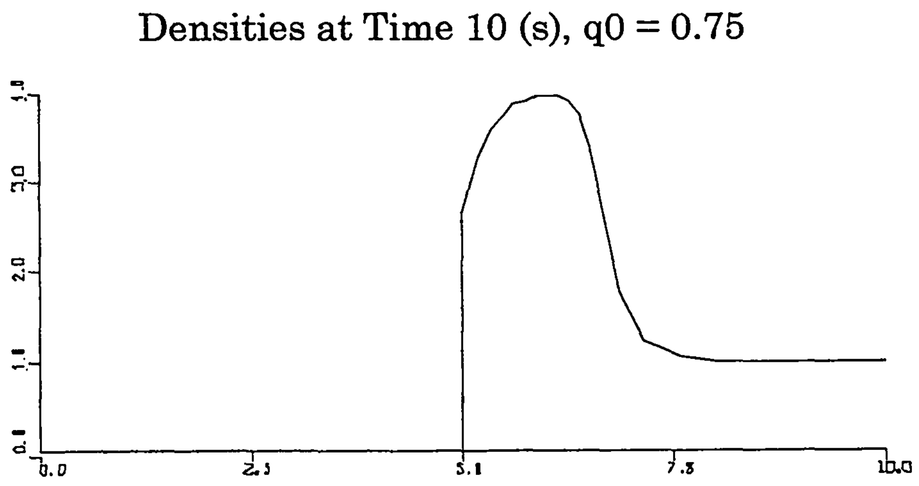

Figure IV-14

At even higher $q0$ values, the diffusional stability condition is violated, resulting in the program being terminated by errors.

A second problem that can be modeled using a one-dimensional Lagrangian code is that of a SHOCK TUBE, a tube that contains two fluids, usually gases, of different densities. Computationally, this problem is set up by setting all velocities to zero and creating an array that is made up of one set of zones at a density ρ_{left} and another set of zones at a different density ρ_{right}. Because our equations assume a constant M, these zones must be MASS MATCHED such that the mass of every zone is a constant. This matching is done by decreasing the initial length of the denser zones relative to the initial length of the less dense zones such that $dx\,\rho$ is a constant.

In the example presented in this work, ρ_{left} is chosen to be 1 (g/cm^3) while ρ_{right} is 4 (g/cm^3). Mass matching is achieved by multiplying the length of the left zones by 8/5 and multiplying the length of the right zones by 2/5 so that $8/5 \times 1 = 8/5 = 2/5 \times 4$. The resulting code appears as the following:

```
      do 100 j = 1,jbar/2
        rho(j) = rho0
        x(j) = x(j−1)+(8./5.)(length/float(jbar))
 100  continue
      do 200 j = (jbar/2)+1,jbar+1
        rho(j) = rho0*4
        x(j) = x(j−1)+(2./5.)*(length/float(jbar))
 200  continue
```

For the results shown in this simulation, the following parameters are used: length = 10.0 (cm), ul = 0.0 (cm/s), ur = 0.0 (cm/s), jbar = 20, rhol = 1.0 (g/cm^3), rhor = 4.0 (g/cm^3), sie0 = 1.0(cm^2/s^2), q0= 0.3, gamma = 5/3, and dt = 0.05 (s). Note that sie0 is not equal to zero in this simulation; there would be no motion of fluids in the shock tube without some initial internal energy being present. Figures IV-15 through IV-20 are graphs of this system at times of 0, 1, 2, 3, 4, and 5 seconds respectively.

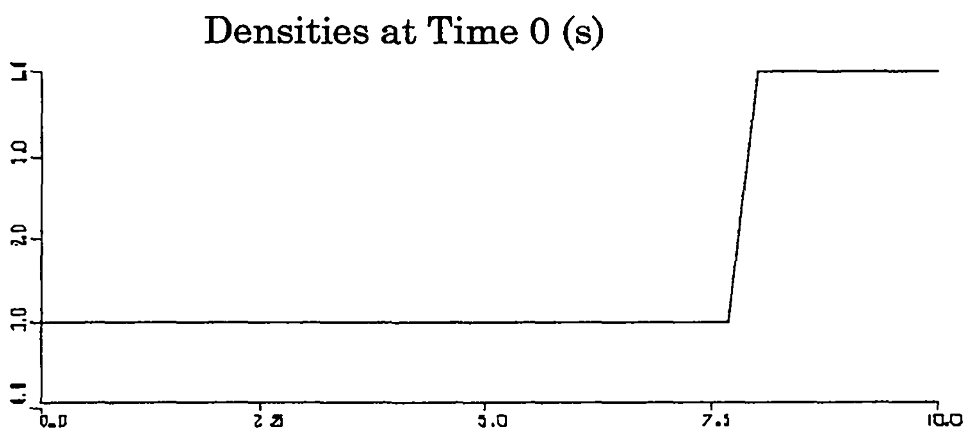

Figure IV-15

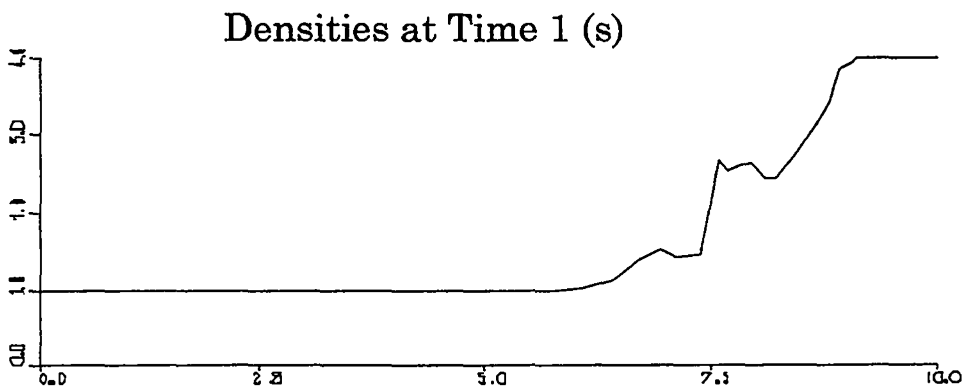

Figure IV-16

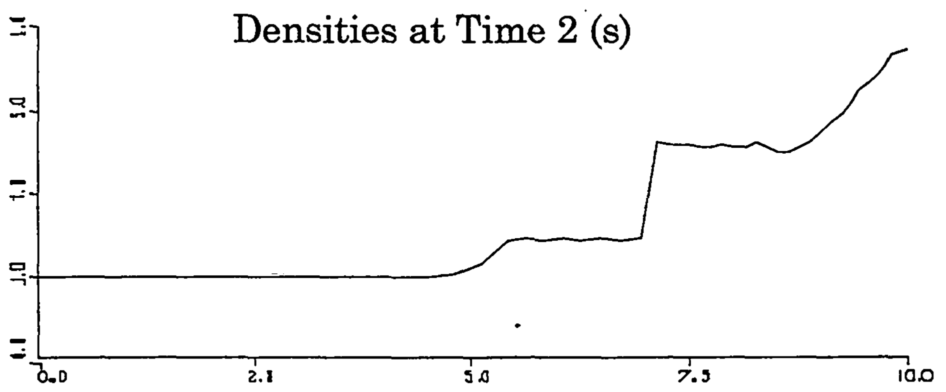

Figure IV-17

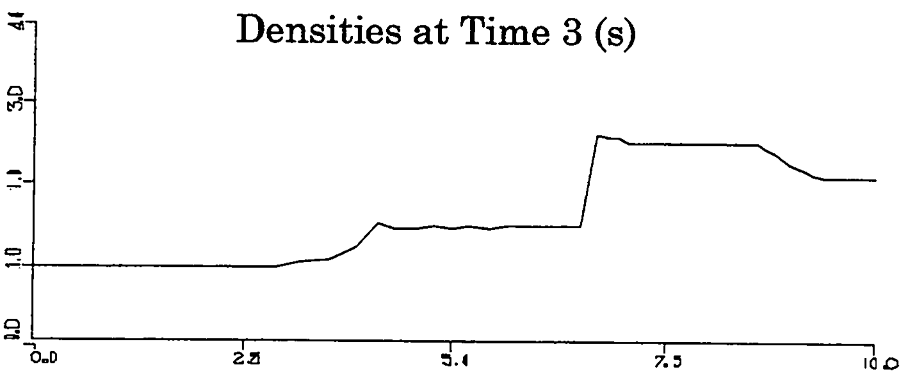
Figure IV-18

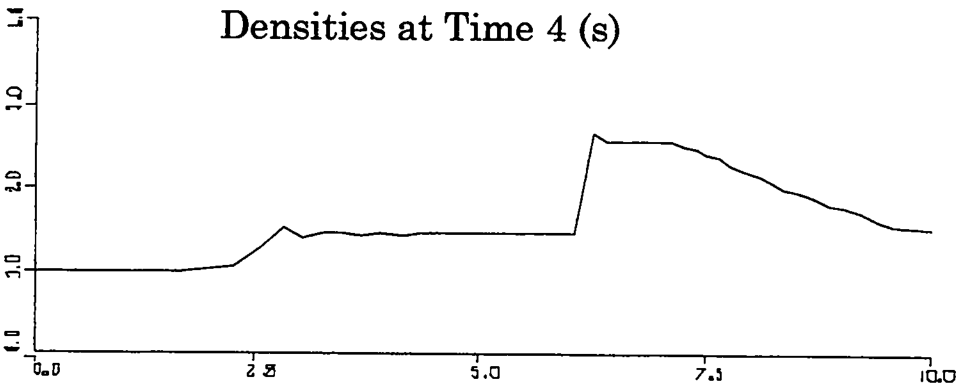
Figure IV-19

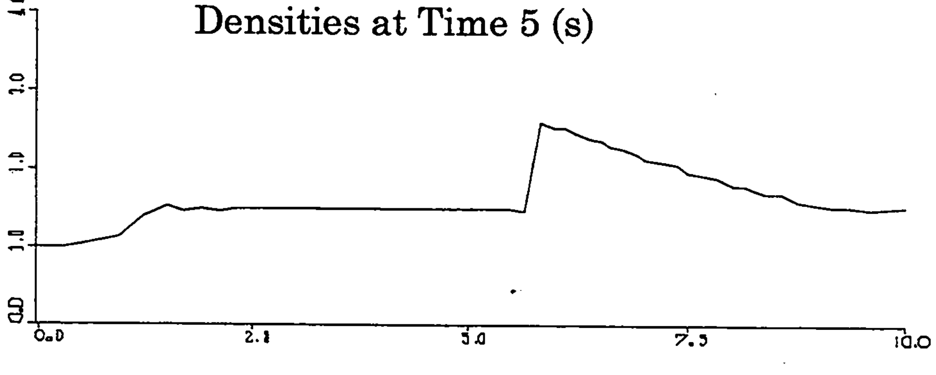
Figure IV-20

In these graphs we see three major features: a shock wave moving to the left, a CONTACT DISCONTINUITY between the two fluids that is also moving to the left, and a RAREFACTION WAVE that is moving to the right and bouncing off of the wall. Each of these elements has been labeled in Fig. IV-21, below.

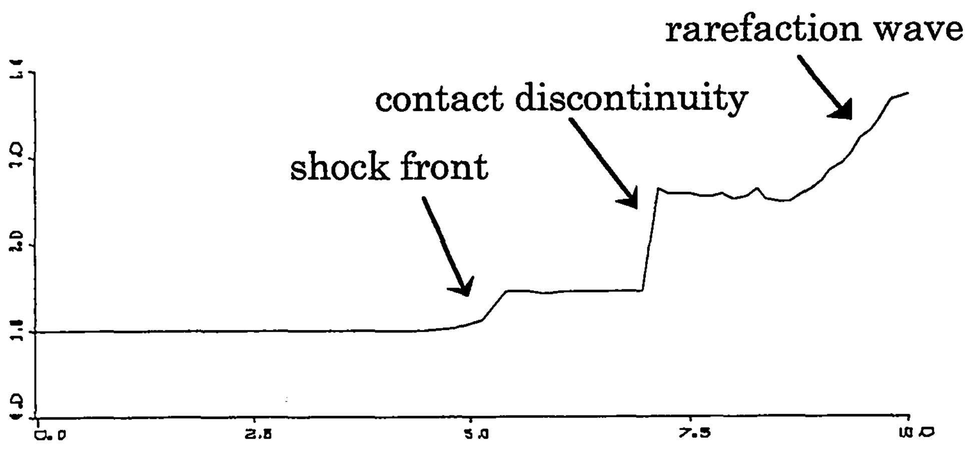

Figure IV-21

The equations that describe the properties of each of these three features of the shock tube problem will not be included in this work. Once again, the reader interested in these equations should refer to LA-4700 or a similar work.

The same sort of instabilities that were present in the piston problem can also be induced in the shock tube problem, as is illustrated by the following plots of density at a time of 2 seconds, each generated by the same parameters as the previous graphs except for $q0$, which is 0.1 in the first graph and 0.75 in the second.

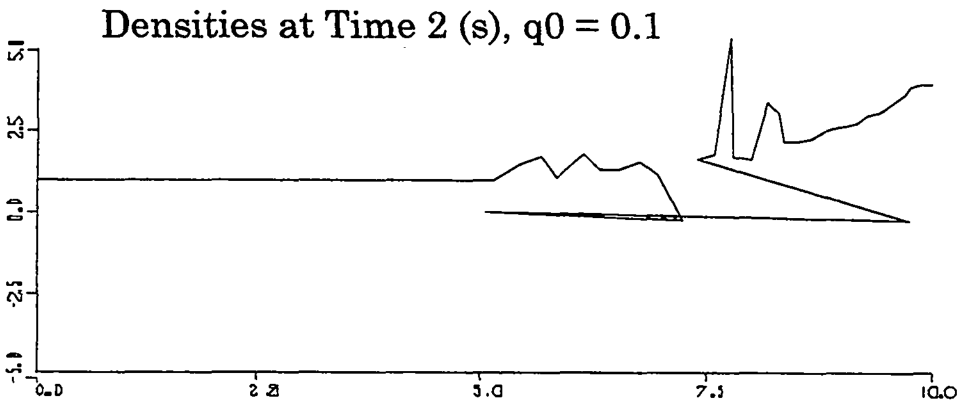

Figure IV-22

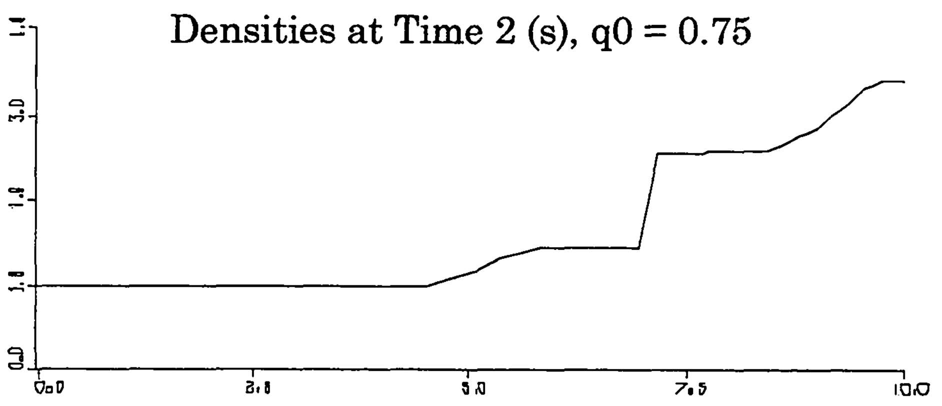

Figure IV-23

Again, if $q0$ is increased to an even higher level, the code will become numerically unstable.

In this chapter we have seen a number of simulations that can be created using the Lagrangian equations for one-dimensional compressible fluid flow. In the following chapter, we will solve the same sorts of problems using an Eulerian method, learning a different technique that can be used to solve the equations of fluid motion computationally.

V. EULERIAN FLUID DYNAMICS

A. Eulerian Methods and Advective Flux

In the previous chapter we examined the use of Lagrangian methods in solving the equations of one-dimensional compressible fluid flow. We are now going to approach the same problem from a different perspective, using an Eulerian technique. In this method the zone positions are held completely fixed, while all quantities are allowed to move between zones. Cell masses are not constant in time, but instead fluid moves between cells; while only the spatial coordinates of the zones remain constant.

This constancy of spatial coordinates is maintained by the calculation of ADVECTIVE FLUXES, fluxes that occur as a result of the motion of fluid from one region to another. An example of this type of flux is the transfer of heat by convection, where heat energy is moved from one region to another by the transfer of the material that contains that energy. The new region is heated not because the material in that region has absorbed the energy from another region, but because a new, hotter material has been moved in to replace the old.

This type of flux is in contrast to the NONADVECTIVE FLUXES that were present in our Lagrangian calculations. Those fluxes occur when the quantities themselves move from one region to another without any motion of material. An example of a nonadvective flux is heat conduction.

While our previous simulation dealt only with nonadvective fluxes, our Eulerian one-dimensional fluid code will include both advective and nonadvective fluxes. In order to accomplish this, we must return to our six equations that describe the interaction of the various physical quantities and add to each a term that describes the advective fluxes that are intrinsic to the Eulerian method.

B. The Equations of Eulerian Fluid Flow

In order to understand the manner that advective flux can be mathematically represented, we must first take a closer look at the situation that it represents. Consider a system such as in Fig. V-1, in which a portion of the material in one zone is being moved into the zone that is adjacent to the right.

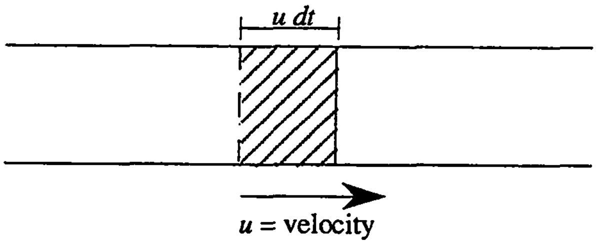

Figure V-1

In this picture we see that when the material in a zone is moving at a velocity u, the material contained in a length $u\,dt$ will be moved into the adjacent cell. Because each zone has an area A, the volume moved from one cell to another is $A\,u\,dt$.

This transfer of volume can be multiplied by ρ, the mass per unit volume, to obtain the following equation for total mass crossing the cell boundary in a given time step:

$$\text{Total Mass Crossing Boundary in a Time Step } (dt) = A\,\rho\,u\,dt \qquad \text{(V-1)}$$

This equation can be used to find the mass flux, the total mass crossing per unit time per unit area:

$$\text{Mass Flux } = \frac{A\,\rho\,u\,dt}{dt\,A} = \rho\,u \qquad \text{(V-2)}$$

Equation (V-2) is a statement of the advective mass flux between two cells. It illustrates a much more general principle that can be shown by replacing ρ by a value Q, the density of any quantity that is being advected. In this general case

$$\text{Advective Flux } = Qu\,. \qquad \text{(V-3)}$$

The density of each of the various physical variables is computed by simply dividing the desired quantity by the volume of a cell. Consider the case of momentum, for example. As was stated in the previous chapter, momentum is mass times velocity:

$$m\,u = \text{momentum}\,. \tag{V-4}$$

Dividing both sides by the volume of a cell, we obtain

$$\frac{m\,u}{\text{volume}} = \frac{\text{momentum}}{\text{volume}}\,. \tag{V-5}$$

Because m/volume is ρ and momentum/volume is the momentum density, this equation can be rewritten:

$$\rho\,u = \text{momentum density}\,. \tag{V-6}$$

By a similar process, we find that

$$\rho\,I = \text{internal energy density}\,, \tag{V-7}$$

and

$$\frac{\rho\,u^2}{2} = \text{kinetic energy density}\,. \tag{V-8}$$

Note that I in Eq. (V-7) is internal energy per unit mass.

Substituting these three density terms into Eq. (V-3), we obtain the following equations of advective flux:

$$\text{Advective flux of mass} = \rho\,u \tag{V-9}$$

$$\text{Advective flux of momentum} = \rho\,u^2 \tag{V-10}$$

$$\text{Advective flux of internal energy} = \rho\,I\,u \tag{V-11}$$

$$\text{Advective flux of kinetic energy} = \frac{\rho u^2}{2} u \tag{V-12}$$

We will use these expressions in deriving the equations of Eulerian fluid flow.

We begin with the expression for density, which was described in our Lagrangian calculations as

$$\rho_j^n = \frac{M}{x_{j+1/2}^n - x_{j-1/2}^n} \,. \tag{IV-11}$$

This expression needs to be modified to reflect the fact that mass is no longer a constant and that distance between cell walls is no longer a variable. To modify this equation, we first substitute dx, the fixed distance between the cell walls, for $x_{j+1/2}^n - x_{j-1/2}^n$:

$$\rho = \frac{M}{dx} \,. \tag{V-13}$$

We must now derive an expression for changes in M, the mass of a cell divided by the area. This derivation is similar to that of the expression for heat in Chapter II. From mass conservation,

$$\text{mass}_j^{n+1} - \text{mass}_j^n = \text{amount in - amount out} \,. \tag{V-14}$$

Because amount in and amount out are simply flux × area × time step, and flux has been defined by Eq. (V-2), numerical expressions for both these terms can be calculated:

$$\text{amount in} = \text{flux}_{\text{left}}\, A\, dt = (\rho\, u)_{j-1/2}\, A\, dt \tag{V-15}$$

$$\text{amount out} = \text{flux}_{\text{right}}\, A\, dt = (\rho\, u)_{j+1/2}\, A\, dt \tag{V-16}$$

By substituting these two values into Eq. (V-14) the change in mass, $\text{mass}_j^{n+1} - \text{mass}_j^n$, can be expressed as follows:

$$\Delta\ \text{mass} = (\rho\, u)_{j-1/2}\, A\, dt - (\rho\, u)_{j+1/2}\, A\, dt \,. \tag{V-17}$$

Using our definition of M as mass divided by area and factoring out like terms, we obtain an equation for change in M:

$$\Delta M = dt\left((\rho\, u)_{j-1/2} - (\rho\, u)_{j+1/2}\right) \,. \tag{V-18}$$

Combining this equation with Eq. (V-13), we find the following:

$$\Delta\rho = \left(\frac{dt}{dx}\right)\left((\rho\, u)_{j-1/2} - (\rho\, u)_{j+1/2}\right) \,. \tag{V-19}$$

Because the new density is equal to the old density plus the change in density, $\rho + \Delta\rho$, we are left with a final equation for the updating of densities that is made up of two parts: an expression for the density at the old time step and an expression for the change due to advective flux:

$$\rho_j^{n+1} = \rho_j^n + \left(\frac{dt}{dx}\right) \left((\rho u)_{j-1/2} - (\rho u)_{j+1/2}\right) . \tag{V-20}$$

This analysis leaves us with an equation that expresses density at the new time step, but also presents us with a problem. Equation (V-20) makes use of the advected $\rho_{j-1/2}$ and $\rho_{j+1/2}$ densities expressed at the left and right wall of cell j. These quantities cannot be referenced directly but instead must be computed using one of two methods: CENTERED or DONOR CELL.

Centered expressions for advected quantities are computed by averaging the values at the cell centers to the right and left of the wall across which fluid is being advected. In our case, centering would lead to an expression for density in the form of Eq. (V-15):

$$\rho_{j-1/2} = \frac{1}{2}(\rho_j + \rho_{j-1}) . \tag{V-21}$$

This value is not acceptable for $\rho_{j-1/2}$, however, because it is UNCONDITIONALLY UNSTABLE, meaning unstable no matter how small we choose our time step. The reason for this instability will be discussed in Chapter VI.

A better method is the donor-cell technique, which uses the upstream value as the value at the advection cell wall. In this technique, the value of a quantity at the cell wall is equal to the value at the left cell center if the flow is from the left or equal to the value at the right cell center if the flow is from the right. This choice of values is mathematically expressed as

$$\begin{gathered} (u\rho)_{j-1/2} = \rho_{j-1} u_{j-1/2} \quad \text{.if} \quad u_{j-1/2} > 0 \\ \text{or} \\ (u\rho)_{j-1/2} = \rho_j u_{j-1/2} \quad \text{if} \quad u_{j-1/2} < 0 \end{gathered} \tag{V-22}$$

and is illustrated visually in Fig. V-2.

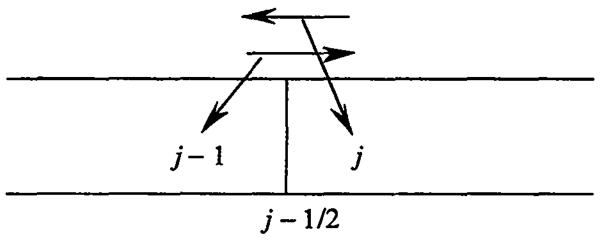

Figure V-2

The donor technique should be employed wherever a quantity is being advected across a cell wall, as is the case with internal energy.

The Eulerian internal energy equation can be calculated beginning with the Lagrangian equation:

$$I_j^{n+1} = I_j^n + \frac{dt}{M}\left(q_j^n + p_j^n\right)\left(u_{j-1/2}^n - u_{j+1/2}^n\right) \tag{IV-16}$$

This transport equation for I is made up of two major terms: the internal energy at the last time step (I_j^n) and the change because of nonadvective flux $\frac{dt}{M}\left(q_j^n + p_j^n\right)$ $\left(u_{j-1/2}^n - u_{j+1/2}^n\right)$. To write this equation in an Eulerian manner, we must add a third term to represent the advective flux. Before this term is added, however, this equation must first be modified. Multiplying by M, we obtain

$$(M\,I)_j^{n+1} = (M\,I)_j^n + dt(p+q)_j^n\left(u_{j-1/2}^n - u_{j+1/2}^n\right) . \tag{V-23}$$

This equation represents the total change in $M\,I$ due to the nonadvective pressure terms.

We saw in Eq. (V-20) that the transport equation for a variable whose value is changed only by advective flux appears in the form

$$\left(\frac{Q}{V}\right)_j^{n+1} = \left(\frac{Q}{V}\right)_j^n + (dt/dx)\left((\text{flux of } Q)_{j-1/2} - (\text{flux of} Q)_{j+1/2}\right) , \tag{V-24}$$

where Q is any variable property of the cells and V is the volume of a single zone. Using this equation to express change due to advective flux in terms of energy density, we obtain

$$(MI)_j^{n+1} = (MI)_j^n - dt\left((\rho uI)_{j+1/2}^n - (\rho uI)_{j-1/2}^n\right) . \tag{V-25}$$

Combining this equation with Eq. (V-23) gives us an expression for change in internal energy that accounts for both advective and nonadvective fluxes:

$$(MI)_j^{n+1} = (MI)_j^n - dt\left[(\rho uI)_{j+1/2} - (\rho uI)_{j-1/2}^n + (p+q)_j^n(u_{j+1/2}^n - u_{j-1/2}^n)\right] \,. \quad \text{(V-26)}$$

Because zones are stationary in an Eulerian simulation, $M = \rho\, dx$. Therefore, this equation can be rewritten as

$$(\rho I)_j^{n+1} = (\rho I)_j^n - \frac{dt}{dx}\left[(\rho uI)_{j+1/2}^n - (\rho uI)_{j-1/2}^n + (p+q)_j^n(u_{j+1/2}^n - u_{j-1/2}^n)\right] \,. \quad \text{(V-27)}$$

This equation is computed using the donor-cell technique for the ρuI terms:

$$(\rho uI)_{j-1/2} = u_{j-1/2}(\rho I)_{j-1} \quad \text{if} \quad u_{j-1/2} > 0$$

$$\text{or} \qquad \text{(V-28)}$$

$$(\rho uI)_{j-1/2} = u_{j-1/2}(\rho I)_j \quad \text{if} \quad u_{j-1/2} < 0 \,.$$

By a process similar to the derivation of Eq. (V-26), Eq. (IV-10) is rewritten as

$$(Mu)_{j+1/2}^{n+1} = (Mu)_{j+1/2}^n - dt\left((p+q)_{j+1}^n - (p+q)_j^n\right) \,. \quad \text{(V-29)}$$

This equation is combined with an advective equation in the form of Eq. (V-25), namely

$$(Mu)_{j+1/2}^{n+1} = (Mu)_{j+1/2}^n + dt\left((\rho u^2)_j^n - (\rho u^2)_{j+1}^n\right) \,, \quad \text{(V-30)}$$

to obtain an equation that accounts for both the advective and nonadvective fluxes that affect momentum:

$$(Mu)_{j+1/2}^{n+1} = (Mu)_{j+1/2}^n - dt\left((\rho u^2)_{j+1}^n - (\rho u^2)_j^n + (p+q)_{j+1}^n - (p+q)_j^n\right) \,. \quad \text{(V-31)}$$

Once again referring to our equation for M given a fixed distance between cells ($M = \rho\, dx$) we obtain

$$(\rho u)_{j+1/2}^{n+1} = (\rho u)_{j+1/2}^n - \frac{dt}{dx}\left((\rho u^2)_{j+1}^n - (\rho u^2)_j^n + (p+q)_{j+1}^n - (p+q)_j^n\right) \,. \quad \text{(V-32)}$$

In this equation, values for u at the cell centers must be computed using the donor-cell technique. These appear in the form

$$(\rho u^2)_j = u_j(\rho u)_{j-1/2} \quad \text{if} \quad u_j > 0$$

or

$$(\rho u^2)_j = u_j(\rho u)_{j+1/2} \quad \text{if} \quad u_j < 0\,, \tag{V-33}$$

where u_j, $\rho_{j-1/2}$, and $\rho_{j+1/2}$ are computed as averages of the values half a cell to the left and half a cell to the right of the point at which these quantities are defined:

$$u_j = \left(\frac{u_{j-1/2} + u_{j+1/2}}{2}\right) \tag{V-34}$$

$$\rho_{j-1/2} = \left(\frac{\rho_{j-1} + \rho_j}{2}\right) \tag{V-35}$$

$$\rho_{j+1/2} = \left(\frac{\rho_j + \rho_{j+1}}{2}\right)\,. \tag{V-36}$$

The student may pose the question of why these averages are used in donor-cell calculations, as they seem to indicate a centered approach that is unconditionally unstable. To explain why these averages are employed, we return to our momentum cell diagram, noting where these various variables are located. In the following figure, the letters in bold indicate quantities at positions where their values are not specified.

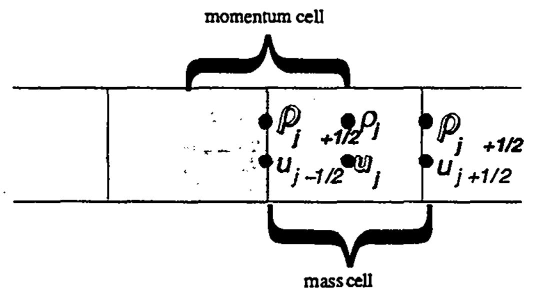

Figure V-3

The terms that employ the donor cell technique are made up of two portions, the donoring velocity and the quantity that is donored. In our original case of density flux, these quantities are $u_{j-1/2}$ and $\rho_{j-1/2}$, respectively. The donoring velocity is always taken at the position at which the flux is taking place, whereas the donored quantity is taken at the center of the cell to the upstream side of the cell wall at which a flux is taking place. For density flux and internal energy flux, all of these values can be taken from positions at which these quantities were directly defined: density and internal energy at the cell centers and velocity at the cell walls. For momentum flux, however, the situation is different.

From Fig. V-3, we see that the donoring velocity at the wall of the momentum cell is u_j, whereas the donored quantities at the center of the momentum cell exist at positions $j - 1/2$ or $j + 1/2$. This configuration forces us to use values that are not directly present in our arrays. These values: u_j, $\rho_{j-1/2}$, and $\rho_{j+1/2}$, are obtained by averaging as in Eqs. (V-34) through (V-36).

We have now derived Eulerian equations for density (ρ), internal energy (I), and momentum (mass $\times\, u$). Values for pressure and viscous pressure (p and q) are determined directly from the values of the other three quantities at each new time step. Thus, the equations for p and q from Chapter IV can be used in our Eulerian simulation. We have completed all the derivation necessary to obtain a set of equations for the simulation of one-dimensional fluid flow in an Eulerian manner and can now begin to implement these equations on the computer. Before we begin this implementation, however, let us first take a look at how our equations appear in partial-differential form and make some observations as to the way that Lagrangian and Eulerian calculations are related.

C. The Partial-Differential Equations of Fluid Flow

Once again, we are going to examine the partial-differential equations that relate to our finite-difference equations. As was the case when we previously examined these equations, this section is not necessary in the writing of our finite-difference code. It is provided only as an additional method of looking at this system.

By a process similar to that used in Section C of Chapter 2, we can rewrite our equations of fluid flow by taking the limits as dx and dt approach zero and generating equations in partial-differential form. In this form Eqs. (V-20) (mass), (V-27) (momentum), and (V-31) (heat energy) appear as follows:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0 \tag{V-37}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u^2}{\partial x} = -\frac{\partial (p+q)}{\partial x} \tag{V-38}$$

$$\frac{\partial \rho I}{\partial t} + \frac{\partial \rho u I}{\partial x} = -(p+q)\frac{\partial u}{\partial x} \, . \tag{V-39}$$

These equations represent the Eulerian form of the transport equations for mass, momentum, and heat energy respectively. They are another form of the Navier-Stokes Equations.

We are going to take a look at these Eulerian equations and relate them to the equations used in the Lagrangian code, trying to gain a better understanding of why these two seemingly dissimilar methods yield computationally similar results.

We will begin with the mass equation, Eq. (V-37). By the chain rule, the second term can be expanded to obtain

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + \rho\frac{\partial u}{\partial x} = 0 \, . \tag{V-40}$$

We now employ the mathematical identity for the total differential of a function of two variables, $f(x,t)$:

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}dx \, . \tag{V-41}$$

This equation states that for arbitrarily slight changes in t and x (denoted by dt and dx) the function f changes by an amount df, as given by the formula. Dividing by dt gives us

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\frac{dx}{dt} \, . \tag{V-42}$$

In the special case when $\frac{dx}{dt}$ follows the motion of a fluid, as in a Lagrangian calculation, then $\frac{dx}{dt} = u$ and

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} . \tag{V-43}$$

This is an expression for the rate of change of f along the motion of a fluid, also known as the LAGRANGIAN DERIVATIVE. It will be denoted in this work as $\frac{df}{dt}$ as opposed to $\frac{\partial f}{\partial t}$. Elsewhere in the literature, the notation $\frac{Df}{Dt}$ is often used to further emphasize the difference between the partial and Lagrangian derivatives. The meaning, however, is equivalent.

Using the Lagrangian derivative to rewrite Eq. (V-40), we obtain

$$\frac{d\rho}{dt} + \rho\frac{\partial u}{\partial x} = 0 . \tag{V-44}$$

This equation is the Lagrangian partial-differential equation for fluid flow; its finite-difference approximation is equivalent Eq. (IV-11). To show this equivalence, we begin with Eq. (V-44) and divide by ρ^2 to obtain

$$\frac{1}{\rho^2}\frac{d\rho}{dt} + \frac{1}{\rho}\frac{\partial u}{\partial x} = 0 \tag{V-45}$$

or

$$-\frac{d\frac{1}{\rho}}{dt} + \frac{1}{\rho}\frac{\partial u}{\partial x} = 0 . \tag{V-46}$$

Finite differencing the second term gives us

$$-\frac{d\frac{1}{\rho}}{dt} + \frac{1}{\rho dx}\left(u_{j+1/2} - u_{j-1/2}\right) = 0 . \tag{V-47}$$

Note that in this equation dx is no longer part of a partial derivative but a finite distance between zones.

From Eq. (V-13) we have $M = \rho\, dx$, and from Eq. (IV-1) we have $u = dx/dt$, so we can write this equation as

$$-\frac{d\frac{1}{\rho}}{dt} + \frac{1}{M}\left(\frac{dx_{j+1/2}}{dt} - \frac{dx_{j-1/2}}{dt}\right) = 0 . \tag{V-48}$$

In this equation, all dt terms are Lagrangian derivatives and can thus be treated in the same manner. We can therefore integrate this equation with respect to dt to obtain

$$\frac{1}{\rho} = \left(\frac{x_{j+1/2} - x_{j-1/2}}{M} \right) . \tag{V-49}$$

This equation is equivalent to our Lagrangian density equation,

$$\rho_j^n = \frac{M}{x_{j+1/2}^n - x_{j-1/2}^n} . \tag{IV-11}$$

We see then, through the use of partial-differential equations, that the Eulerian and the Lagrangian mass equations are equivalent in the properties that they represent.

This equivalence is also true for the momentum equation, which appears in Eulerian form as Eq. (V-38). This equation can be expanded to obtain

$$\rho\frac{\partial u}{\partial t} + u\frac{\partial \rho}{\partial t} + \rho u\frac{\partial u}{\partial x} + u\frac{\partial \rho u}{\partial x} = -\frac{\partial P}{\partial x} , \tag{V-50}$$

where P signifies the total pressure ($P = p + q$).

Returning to the mass equation (V-37), we see that the sum of the second and fourth terms of Eq. (V-50) is equal to zero; this gives us

$$\rho\frac{\partial u}{\partial t} + \rho u\frac{\partial u}{\partial x} = -\frac{\partial P}{\partial x} \tag{V-51}$$

or

$$\rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x}\right) = -\frac{\partial P}{\partial x} . \tag{V-52}$$

Employing the Lagrangian derivative, we obtain

$$\rho\frac{du}{dt} = -\frac{\partial P}{\partial x} . \tag{V-53}$$

This equation is the Lagrangian partial-differential equation for momentum. Dividing both sides by ρ and finite-differencing it gives us

$$\frac{u_j^{n+1} - u_j^n}{dt} = -\frac{1}{\rho dx}\left(P_{j+1/2}^n - P_{j-1/2}^n\right) , \tag{V-54}$$

which, with a shift of indices, is equal to Eq. (IV-10):

$$u_{j+1/2}^{n+1} = u_{j+1/2}^{n} + \frac{dt}{M}\left(p_j^n + q_j^n - p_{j+1}^n - q_{j+1}^n\right) . \tag{IV-10}$$

For internal energy, the process is similar. Eq. (IV-39) is expanded

$$\rho\frac{\partial I}{\partial t} + I\frac{\partial \rho}{dt} + \rho u\frac{\partial I}{\partial x} + I\frac{\partial \rho u}{\partial x} = -\frac{P\partial u}{\partial x} , \tag{V-55}$$

the second and fourth terms are dropped using the mass equation

$$\rho\left(\frac{\partial I}{\partial t} + u\frac{\partial I}{\partial x}\right) = -P\frac{\partial u}{\partial x} , \tag{V-56}$$

and finally the Lagrangian derivative is used to get the Lagrangian equation for change in internal energy:

$$\rho\frac{dI}{dx} = -P\frac{\partial u}{\partial x} . \tag{V-57}$$

Through finite differencing, this equation can be shown to be a partial-differential representation of Eq. (IV-16):

$$I_j^{n+1} = I_j^n + \frac{dt}{M}\left(q_j^n + p_j^n\right)\left(u_{j-1/2}^n - u_{j+1/2}^n\right) . \tag{IV-16}$$

So we see that for an Eulerian simulation, our equations appear as

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0 , \tag{V-37}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u^2}{\partial x} + \frac{\partial P}{\partial x} = 0 , \tag{V-58}$$

$$\frac{\partial \rho I}{\partial t} + \frac{\partial \rho u I}{\partial x} + \frac{P\partial u}{\partial x} = 0 ; \tag{V-59}$$

whereas, in a Lagrangian simulation, our equations are

$$\frac{d\rho}{dt} + \rho\frac{\partial u}{\partial x} = 0 , \tag{V-44}$$

$$\rho\frac{du}{dt} + \frac{\partial P}{\partial x} = 0 , \tag{V-60}$$

$$\rho\frac{dI}{dt} + P\frac{\partial u}{\partial x} = 0 . \tag{V-61}$$

These partial-differential equations provide another way of looking at our one-dimensional fluid-flow equations. They help to explain the Lagrangian and Eulerian finite-difference equations and demonstrate that, although these are seemingly different, their underlying principles are the same.

D. Computational Implementation of Equations

The structure of our Eulerian one-dimensional fluid code is similar to that of the Lagrangian code: it contains the same five sections, its variable declarations are almost the same, and the output procedure is of the same type.

There are, however, some major differences between these two codes. These differences are found in the initialization procedure, in the boundary conditions, and in the equations that are used to update the variable values.

The order in which our variables are given new values is again rho, u, I, p, and q; but rho, u, and I must now be calculated using quantities calculated before the program enters the loop that assigns new values to these arrays. This loop must generate values for rho, u, and I; but the transport equations that were derived in Section B are written in terms of ρ_j, $(\rho u)_{j+1/2}$, and $(\rho I)_j$. We have to obtain array values for the following quantities before calculating the other physical variables:

$$\text{rhon(j)}\text{—}\rho_j^{n+1}$$

$$\text{rhoun(j)}\text{—}\rho u_{j+1/2}^{n+1}$$

$$\text{rhoin(j)}\text{—}\rho I_j^{n+1} \ .$$

Each of these arrays is calculated using the Eulerian equations of transport. The calculations are done for all array values before any updating of rho, u, I, p, or q is done.

Density is computed by simply setting the rho array equal to the rhon array:

rho(j) = rhon(j) .

Velocity, u is computed by dividing the density times velocity array by the density array at position j+1/2:

```
        u(j) = rhou(j)/(.5 * (rhon(j) + rhon(j+1))) .
```

Internal energy is computed by dividing the internal energy times density array by the density array:

```
        sie(j) = rhoin(j)/rhon(j)
```

The p and q equations remain unchanged from the Lagrangian case. This situation leaves us with a loop that assigns values for rho, u, I, p, and q that appears in the following form:

```
        do 300 j =1,jbar
          rho(j) = rhoun(j)
          u(j) = rhoun(j)/(.5*(rhon(j)+rhon(j+1))
          sie(j) = rhoin(j) / rhon(j)
          p(j) = (gamma-1) * rho(j) * sie(j)
          asie = gamma * (gamma-1) * abs(sie(j))
          c = abs(ul) + sqrt(asie)
          q(j) = q0 * rho(j) * c * (u(j-1)-u(j))
          if (q(j).lt.(0.0)) q(j) = 0.0
  300   continue
```

This loop is preceeded by another loop that computes rhon, rhoun, and rhoin arrays using the Eulerian equations for the transport of mass, energy, and momentum:

$$\rho_j^{n+1} = \rho_j^n + \frac{dt}{dx}\left((\rho u)_{j-1/2} - (\rho u)_{j+1/2}\right) \tag{V-20}$$

$$(\rho I)^{n+1} = (\rho I)_j^n - \frac{dt}{dx}\left[(\rho u I)_{j+1/2}^n - (\rho u I)_{j-1/2}^n + (p+q)_j^n(u_{j+1/2}^n - u_{j-1/2}^n)\right] \tag{V-27}$$

$$(\rho u)_{j+1/2}^{n+1} = (\rho u)_{j+1/2}^n - \frac{dt}{dx}\left((\rho u^2)_{j+1}^n - (\rho u^2)_j^n + (p+q)_{j+1}^n - (p+q)_j^n\right) \tag{V-32}$$

Each of these equations requires the use of the donor-cell technique, meaning that donor-cell values must be computed for

$$\rho_{j-1/2} \quad \text{and} \quad \rho_{j+1/2}$$

$$(\rho u I)_{j-1/2} \quad \text{and} \quad (\rho u I)_{j+1/2}$$

$$(\rho u^2)_j \quad \text{and} \quad (\rho u^2)_{j+1} .$$

The problem of having to write our equations in a manner that allows the computation of donor-cell for each of these terms can be approached in at least two ways: with a series of if/then checks or with a double look-up technique.

The first of these methods involves writing a separate if/then check for each of these six terms. This approach uses six different variables, each with values determined according to the direction of the motion of the fluid, with six separate checks being made for the direction of fluid motion at every loop iteration. This method is viable, but it triples the number of if/then checks, calls for the use of additional scalar variables, and unnecessarily complicates our code.

A much easier technique is to carry out all the flow direction checks before any of the arrays are computed. To do this, we create two arrays of variables: idnr and jdnr. In a loop at the beginning of the variable updating portion of the program, all the elements in these arrays are assigned values of either 0, if the flow is from the left to the right, or 1, if the flow is from the right to the left. idnr represents the motion of fluid at the cell walls ($j + 1/2$), while jdnr represents the flow of the fluid at the cell centers (j). The loop in which they are computed is the following:

```
      do 100 j =1,jbar
        idnr(j) = 0
        jdnr(j) = 0
        if (u(j).lt.0.0) idnr(j)=1
        if ((u(j-1)+u(j)).lt.0.0) jdnr(j)=1
 100  continue
```

We can use these integer arrays to determine the positions at which the donor-cell terms are computed. By indexing our variables with j plus an appropriate value of idnr or jdnr, we can rewrite the donor cell terms in the following manner:

$(\rho u)_{j-1/2}$—rho(j-1+idnr(j-1)) * u(j-1)

$(\rho u)_{j+1/2}$—rho(j+idnr(j)) * u(j)

$(\rho u I)_{j-1/2}$—rho(j-1+idnr(j-1)) * u(j-1) * sie(j-1+idnr(j-1))

$(\rho u I)_{j+1/2}$—rho(j+idnr(j)) * u(j) * sie(j+idnr(j))

$(\rho u^2)_j$—(u(j-1)+u(j)) * .5 * u(j-1+jdnr(j-1))

(rho(j-1+jdnr(j-1))+rho(j+jdnr(j))) * .5

$(\rho u^2)_{j+1}$—(u(j)+u(j+1)) * .5 * u(j+jdnr(j)) *

(rho(j+jdnr(j))+rho(j+1+jdnr(j+1))) * .5

The terms on the right of this table are simply computational translations of Eqs. (V-22), (V-28), and (V-33), using a double look-up technique rather than carrying out an if/then statement for each of the equations.

We can compute rhon(j), rhoun(j), and rhoin(j) by constructing a loop that follows the computation of the donor-cell arrays but comes before the computation of the rho-u-sie loop. This loop should appear similar to the following, with the values from the above table being used whenever one of the bold (donor-cell) terms is used.

```
      do 200 j = 1,jbar
c..   density
        rhon(j) = rho(j) + ((dt/dx) * ( (rho u)j-1/2 -(rhou)j+1/2))
c..   internal energy
        rhoin(j) = (rho(j) * sie(j)) - ((dt/dx) * ( (rho u sie)j+1/2-
      &(rho u sie)j-1/2 + (p(j)+q(j)) * (u(j)-u(j-1))))
c..   momentum
        rhoun(j) = (((rho(j)+rho(j+1)) * .5) * u(j)) - ( (dt/dx) *
      &((rho u2)j+1 - (rho u2)j) * (p(j+1)+q(j+1)-p(j)-q(j)))
  200 continue
```

To summarize, the variable updating portion of our program consists of first a donor-cell loop; then a loop to compute the mass, momentum, and internal energy densities; and finally a loop that changes the values of the rho, sie, u, p, and q arrays. The return designations of these loops have been numbered in this order.

Returning to the issue of the boundary conditions: An analysis of our equations indicates a need for specified values of rho(0) and sie(0) in addition to u(0) and u(jbar) as in the Lagrangian case. Positions need no longer be updated because they remain fixed throughout the simulation; instead, the conditions must be added that sie at position 0 is equal to a variable siel, and rho at position 0 is equal to a variable rhol. These boundary conditions are of a different nature than those in the heat-transfer problem. There, a constant temperature was maintained at the wall by the recalculation of the value of T(0) at every time step. The equation was

$$T_0 = 2T_L - T_L \; . \tag{II-21}$$

This averaging is not necessary in the present code, because the value used at the wall is computed using the donor-cell technique. If flow is from the left to the right, as is the case with our piston, the values assigned to sie and rho at position j = 0 will effectively exist at the rightmost wall, $j = 1/2$. We are now faced with the question of what values should be assigned to the variables at these positions.

In the Eulerian case, we do not represent the piston itself but rather a shock that is created by the motion of a piston somewhere upstream. We can therefore appeal to the equations for the fluid dynamics of shocks to determine our boundary conditions at the left: $\rho = \frac{\gamma+1}{\gamma-1}\rho_0$, and $I = \frac{u^2}{2}\rho_0$. By substituting our "gamma" and "ul" for the γ's and u's in these equations, we can generate quantities for rhol and siel that will help to maintain a shock wave. Finally, these two variables are added to the initialization procedure, completing our Eulerian code. A graphical representation of this code appears in Figure V-4.

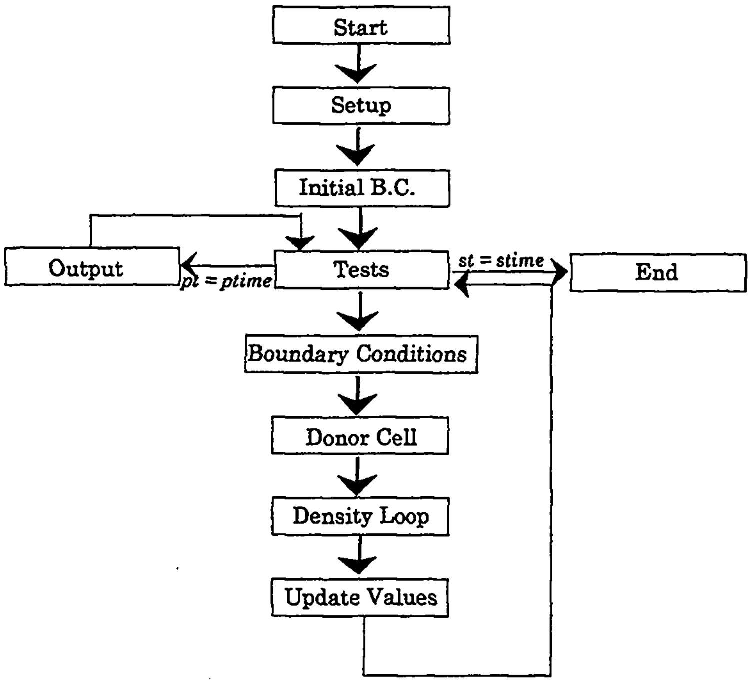

Figure V-4

E. Eulerian Results and Comparison of Eulerian and Lagrangian Simulations

The following five figures are plots of the density of the fluid in the cylinder as the shock moves in from the left. The parameters chosen for this simulation are: length = 10.0 (cm), ul = 0.5 (cm/s), ur = 0.0 (cm/s), jbar=20, rho0 = 1.0 (g/cm^3), sie0 = 0 (cm^2/s^2), q0 = 0.25, gamma = 5/3, and dt = 0.05 (s). Note that these parameters are precisely those used to run the Lagrangian piston problem, with the exception of $q0$, which is lowered from 0.3 to 0.25 for the Eulerian simulation. Plots appear at times of 2, 4, 6, 8, and 10 seconds respectively.

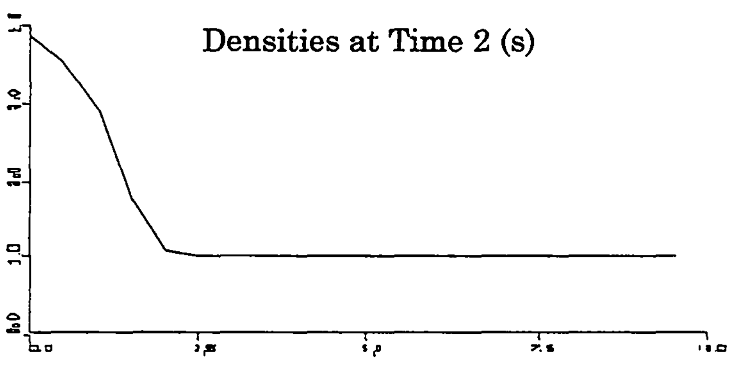

Figure V-5

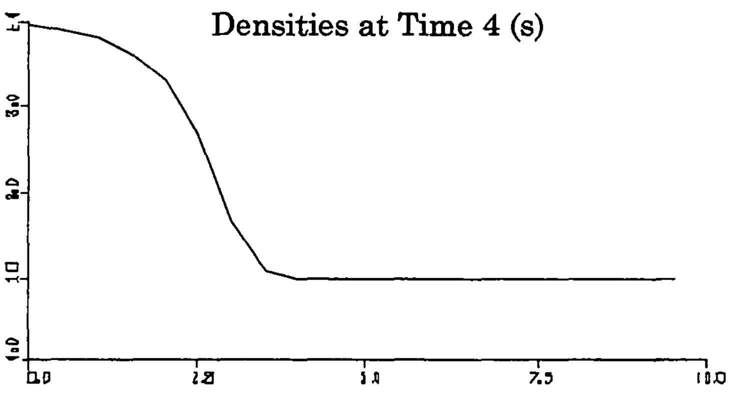

Figure V-6

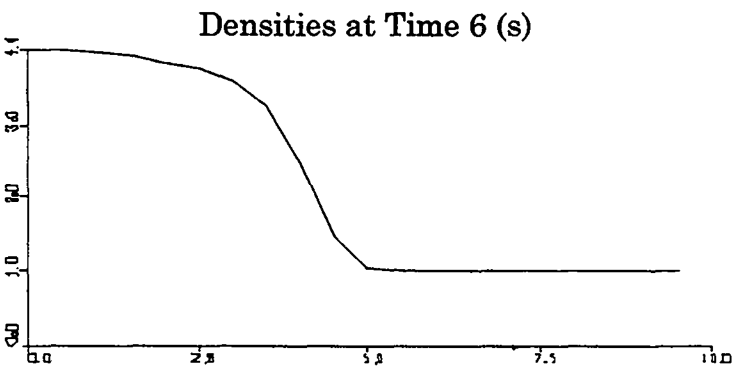

Figure V-7

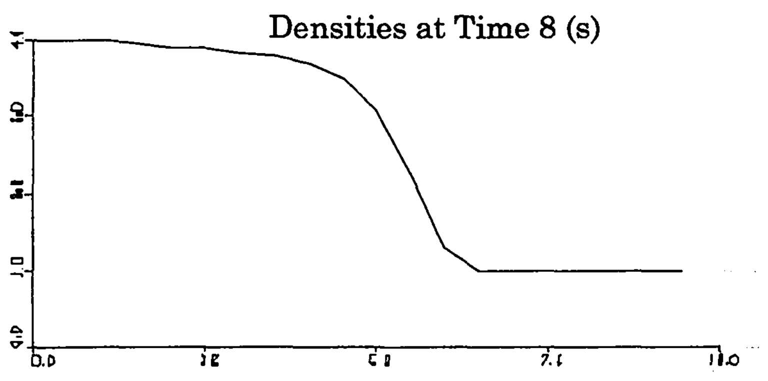

Figure V-8

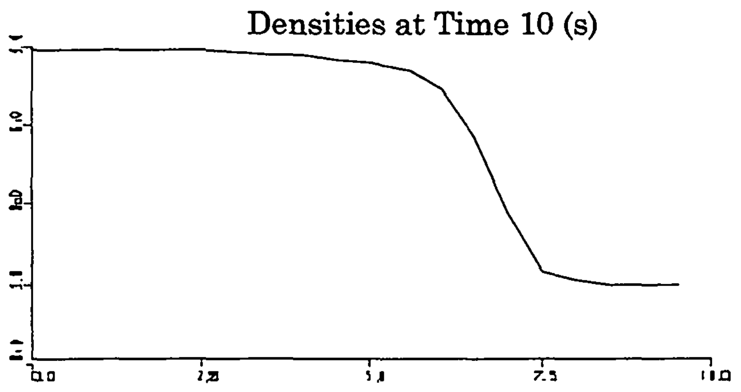

Figure V-9

Note that the Eulerian shock is not as sharp as the Lagrangian shock, even at this lower value of q0. This difference is due to an artificial diffusion that results as an effect of the donor-cell technique. This effect will be discussed in Chapter VI.

Once again applying the equations of shocks [Eqs. (IV-44) and (IV-45)] to the parameters used in our simulation, we predict that our shock will move forward at a speed of 0.66 (cm/s) and produce a compressed region with a density of 4 (g/m^{2}s) . These values verify the results presented in Figs. V-5 through V-9.

The next set of plots demonstrate the results that can be obtained by applying an Eulerian code to the shock tube problem. The following parameters are used: length = 10. (cm), ul = 0.0 (cm/s), ur = 0.0 (cm/s), jbar=20, rhol = 1.0 (g/cm^3), rhor = 4.0

(g/cm^3), sie0 = 1.0 (m^2/s^2), q0= 0.3 and dt = 0.025 (s). Our simulation is set up such that 0.8 of the tube is filled with the less dense fluid and 0.2 is filled with the denser fluid. This set up is necessary to parallel the situation simulated in Chapter IV. Unlike the Lagrangian simulation, however, no mass matching is necessary in the Eulerian case. As was previously stated, masses of zones in an Eulerian simulation are variable; only the positions of zones are constant. Figures V-10 through V-15 are graphs of density within the shock tube system at times of 0, 1, 2, 3, 4, and 5 seconds respectively.

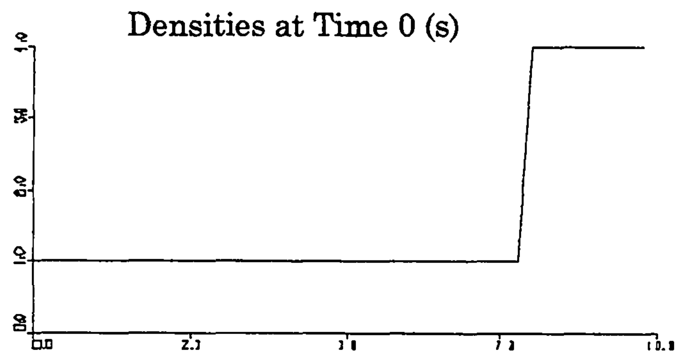

Figure V-10

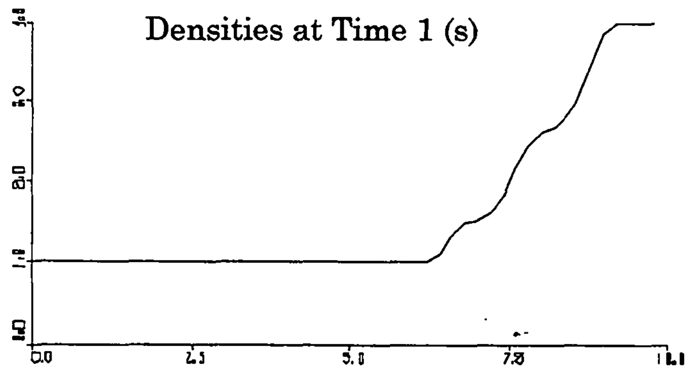

Figure V-11

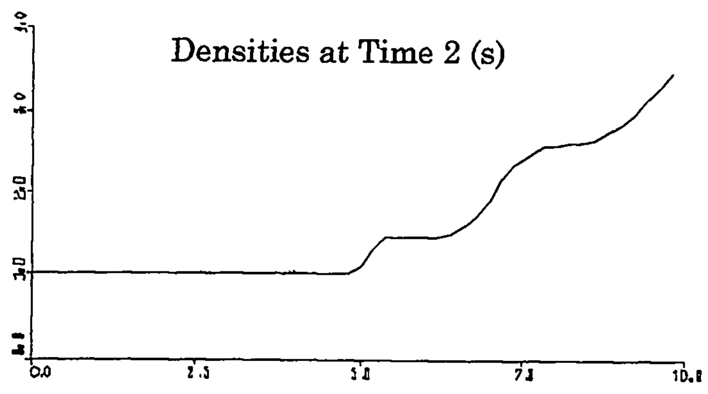

Figure V-12

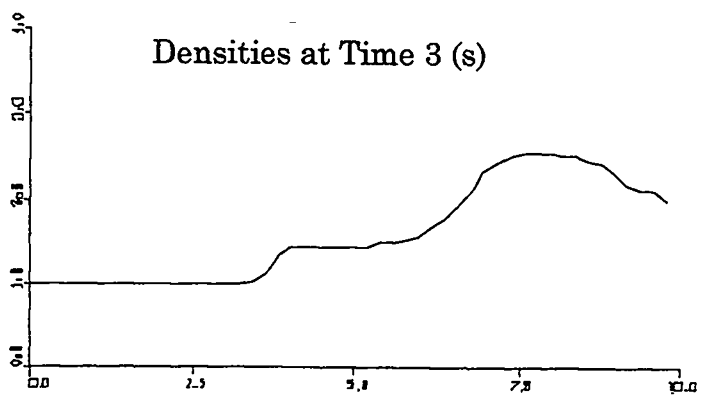

Figure V-13

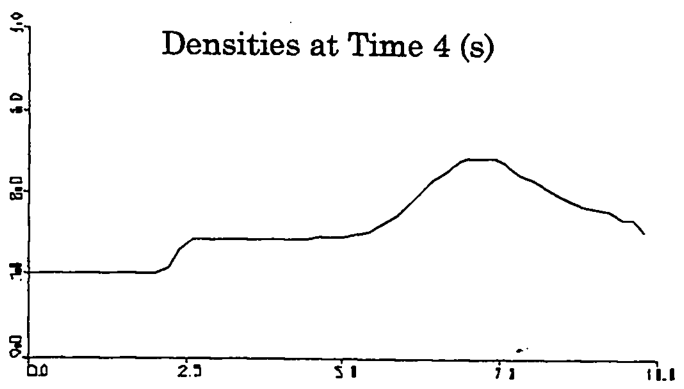

Figure V-14

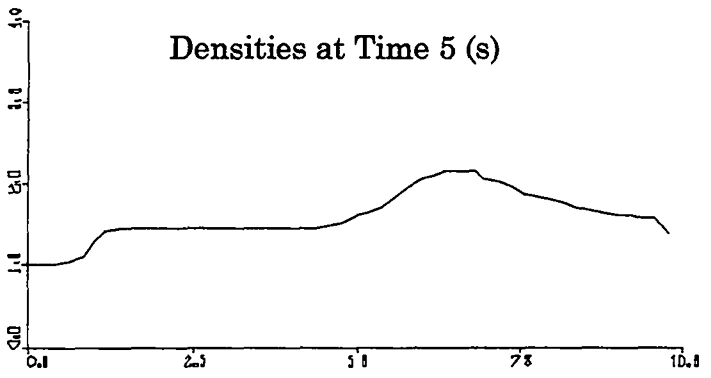

Figure V-15

These graphs contain the same features as the Lagrangian graphs: a shock wave moving to the left, a contact discontinuity between the two fluids that is also moving to the left, and a rarefaction wave that is moving to the right. These features have the same properties as those of the Lagrangian graph and are described by the same set of fluid-dynamics equations.

Lagrangian and Eulerian simulations are also subject to the same stability conditions. At low $q0$ values, the Courant condition is violated, as illustrated in Fig. V-16.

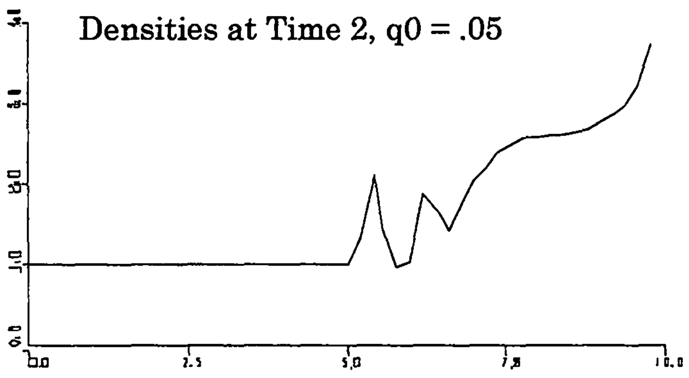

Figure V-16

At high $q0$ values, the features are smeared out:

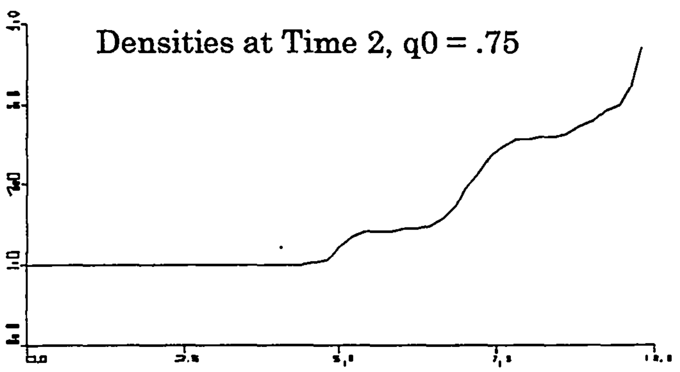

Figure V-17

If $q0$ is raised even higher, the diffusional stability condition is violated and the program terminates.

Although the Lagrangian and Eulerian simulations share the same stability conditions, they are quite different in the sharpness with which they resolve features at a given set of parameters. We see this by comparing graphs of both these simulations at a time step of 2 (s) and a $q0$ of 0.3.

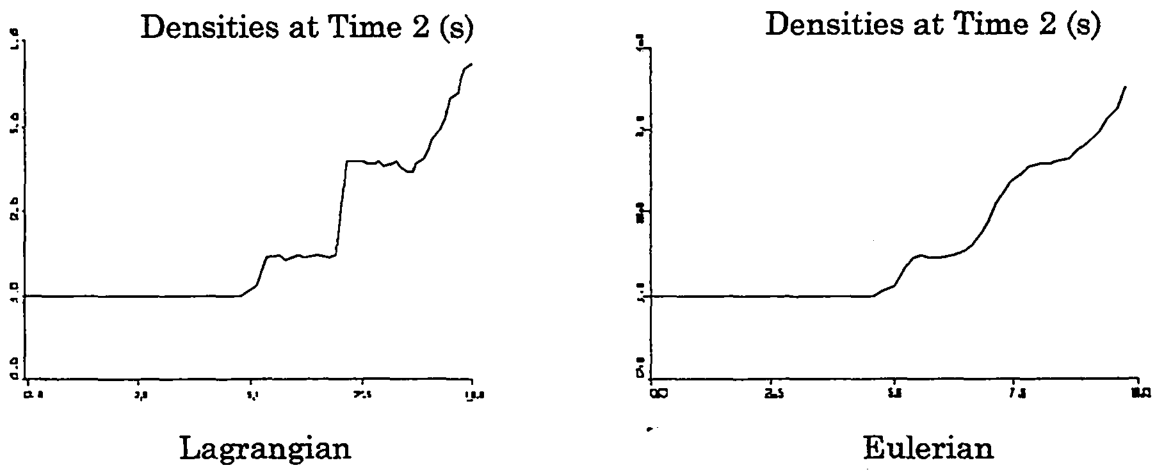

Figure V-18

In this figure, we see that the features of the Lagrangian graph are much sharper than those of the Eulerian simulation. The difference in sharpness is particularly noticeable for the contact discontinuity, which is clear in the Lagrangian simulation but smeared out over several zones in the Eulerian simulation.

The smearing of features in the Eulerian case is a result of the artificial diffusion that is intrinsic to the donor-cell technique. In order to understand why the donor-cell technique causes diffusion in this manner, as well as to understand why the Courant condition is present in an Eulerian simulation, we will have to make use of a method known as truncation error analysis. This method will be discussed in Chapter VI.