Modelling, Analysis and Simulation

Building your own shock tube

J. Naber

REPORT MAS-E0502 JANUARY 2005

ISSN 1386-3703

Building your own shock tube

ABSTRACT
This report treats the development of a shock tube solver for the simulation of flows described by the one-dimensional Euler equations. A well-known one-dimensional flow problem is the initial Riemann problem, which treats the development of a flow due to two initially separated states. Removing the membrane separating these two states results in a characteristic wave pattern consisting in general of three waves, either a shock wave or an expansion fan, and a contact discontinuity. This Riemann problem can be solved exactly using gasdynamics theory. Implementing this theory in a computer program results in the exact Riemann solver. An elegant algorithm is used for the determination of the region containing the time axis. Several tests are performed to check this solver; all results complied with the literature. Solving more complex flow problems than the Riemann problem requires a numerical approach. A Finite-Volume Method is used to discretize the Euler equations. Hancock's predictor-corrector-type MUSCL scheme is used for marching in time, although also the first-order upwind scheme is implemented in the solver. The cell-interface fluxes required for this time stepping procedure are calculated using two different solvers; the exact Riemann solver, also called Godunov's method, and the approximate Riemann solver by Roe. Roe's method is implemented both with and without the entropy fix, which prevents the solution from becoming unphysical in the case of a transonic expansion fan. Further more, several averaging schemes used for the predictor step are implemented in the solver, being the Algebraic average, and the Double Minmod, Superbee and Koren's limiters. The shock tube solver is tested using the five test cases also used for testing the exact Riemann solver. The numerical results compare very well with the exact results. No significant differences between the Godunov and the Roe solver are present. Testing for convergence was done for both the first-order upwind scheme and the second-order Hancock scheme using the less-known fractional error norms such that the influence of the first order errors induced by the contact discontinuity and the shock is reduced. Both these schemes resulted in the expected order of convergence. The averaging schemes are also compared with each other. The limiters perform almost equally well, allowing practically no overshoot or wiggles. This behavior was visible in the case of the linear algebraic average. The entropy fix was tested using an adapted version of Sod's problem, with a transonic expansion fan. The solver of Roe without an entropy fix results in a discontinuity in the expansion fan, while the solver with the fix prevents this from happening. Further the Woodward-Colella double blast wave problem in a closed tube is treated. Two strong initial discontinuities in the pressure result in a complex wave pattern. The solver performs well on this test. Finally the interaction of two non-simple waves is treated. The special case in which the ratio of specific heats is taken equal to 3.0 and in which the characteristics become straight even in the non-simple region is treated too.

2000 Mathematics Subject Classification: 65M20, 65M60, 76N15
Keywords and Phrases: Gas dynamics, Shock tube problems, Riemann problems, Finite Volume discretization
Note: This work was carried out under project MAS2 - Computational Fluid Dynamics and Computational Electromagnetics.

Building your own shock tube

Jorick Naber
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

July 2004

ABSTRACT

This report treats the development of a *shock tube* solver for the simulation of flows described by the one-dimensional Euler equations. A well-known one-dimensional flow problem is the initial Riemann problem, which treats the development of a flow due to two initially separated states. Removing the membrane separating these two states results in a characteristic wave pattern consisting in general of three waves, either a shock wave or an expansion fan, and a contact discontinuity. This Riemann problem can be solved exactly using gasdynamics theory. Implementing this theory in a computer program results in the exact Riemann solver. An elegant algorithm is used for the determination of the region containing the time axis. Several tests are performed to check this solver; all results complied with the literature.

Solving more complex flow problems than the Riemann problem requires a numerical approach. A Finite-Volume Method is used to discretize the Euler equations. Hancock's predictor-corrector-type MUSCL scheme is used for marching in time, although also the first-order upwind scheme is implemented in the solver. The cell-interface fluxes required for this time stepping procedure are calculated using two different solvers; the exact Riemann solver, also called Godunov's method, and the approximate Riemann solver by Roe. Roe's method is implemented both with and without the entropy fix, which prevents the solution from becoming unphysical in the case of a transonic expansion fan. Further more, several averaging schemes used for the predictor step are implemented in the solver, being the Algebraic average, and the Double Minmod, Superbee and Koren's limiters.

The shock tube solver is tested using the five test cases also used for testing the exact Riemann solver. The numerical results compare very well with the exact results. No significant differences between the Godunov and the Roe solver are present. Testing for convergence was done for both the first-order upwind scheme and the second-order Hancock scheme using the less-known *fractional error norms* such that the influence of the first order errors induced by the contact discontinuity and the shock is reduced. Both these schemes resulted in the expected order of convergence. The averaging schemes are also compared with each other. The limiters perform almost equally well, allowing practically no overshoot or wiggles. This behavior was visible in the case of the linear algebraic average. The entropy fix was tested using an adapted version of Sod's problem, with a transonic expansion fan. The solver of Roe without an entropy fix results in a discontinuity in the expansion fan, while the solver with the fix prevents this from happening. Further the Woodward-Colella double blast wave problem in a closed tube is treated. Two strong initial discontinuities in the pressure result in a complex wave pattern. The solver performs well on this test. Finally the interaction of two non-simple waves is treated. The special case in which the ratio of specific heats is taken equal to 3.0 and in which the characteristics become straight even in the non-simple region is treated too.

Acknowledgements

During the months of September through December I will perform my internship at the W.M. Keck Laboratory for Computational Fluid Dynamics (CFD) of the University of Michigan in the United States of America. Here I will, together with Professor Bram van Leer, perform research on the subject of *'multi-fluid simulation'*. To prepare myself in the correct manner for this unique internship I was recommended to become familiar with a programming language (FORTRAN 90/95) and the application of this to several solution techniques for the one-dimensional Euler equations. Especially the application of exact and approximate Riemann solvers in a so-called *shock tube* was of interest.

I was given the opportunity by Professor Barry Koren to perform this preliminary research on the National Research Institute for Mathematics and Computer Science (CWI) in Amsterdam, The Netherlands. Under the supervision of Professor Barry Koren and Phd. student Jeroen Wackers I made an exercise provided by Professor Bram van Leer called *'Building your own shock tube'*. This report treats the development of such a shock tube program, and the results of several calculations using standard test problems.

I would like to thank both Professor Barry Koren and Professor Bram van Leer for giving me the opportunity to work on such an interesting and challenging subject. I believe that the knowledge obtained in this month will be very useful for my further work. Finally I would like to thank Jeroen Wackers for his support; it was a pleasure working with him.

Amsterdam, July 2004,

Jorick Naber

Table of Contents

Chapter 1
Introduction

The flow of fluids and their applications has always interested people; Leonardo da Vinci was fascinated by the complex structures of whirls and eddies in a water pool and studied these extensively, Isaac Newton devoted a whole section of his *Principia* to the mechanics of fluids leading to the first understanding of fluid motion which for example was used in naval applications, the brothers Wright used the principle of lift on a wing to succesfully fly the first airplane ever, Edward Lorentz performed much research on the flows of air in the earth's atmosphere for meteorological purposes and ended up with the famous *chaos theory*. These situations are only a few examples where the flows of fluids play an important role. One can safely conclude that knowledge about fluid flows, both practically and theoretically, is of utmost importance for everyday life.

Ever since Newton, Euler, d'Alembert and many others started their research on the motion of fluids, the knowledge on this matter has increased continuously. While the earliest equations were only coarse approximations of reality, being valid for incompressible, inviscid, irrotational flow, the later Navier-Stokes equations hold for compressible, viscous, rotational flows and are even assumed to describe turbulence. Having these flow equations available means that for specific flow problems solutions can be obtained, which in turn can be used for practical purposes. This is indeed the case for the more simple flow equations such as the Laplace equation for potential flow. But the more complex, and thus realistic, equations can only be solved in certain situations where the flow geometry is simple, as is the case for Poiseuille or Couette flow (Navier-Stokes equations). Due to the complexity of these equations it is until now simply not possible to obtain their exact solutions.

Fortunately the appearance of the computer made another approach possible. The governing flow equations can be approximated in such a way that a computer can numerically solve these. The scientific area that treats these solution techniques is known as *Computational Fluid Dynamics* (CFD). With the ever increasing computer power CFD is able to solve even the most complex flow situations, resulting in many applications both in practical and theoretical areas, making these methods more and more important every day. Using CFD it is now possible to solve complete three-dimensional flow problems, resulting in accurate weather forecasts using simulations of the complete atmosphere, detailed models of the flow around whole airplanes and many more applications like these.

However, there are still many problems left to be solved, which means that the search for more accurate solution techniques should not yet be stopped. But before one is able to increase the complexity of the flow problem, there should first be a thorough understanding of the possibilities of the present techniques, even for less difficult problems. One of the tools that can be used to simulate simple flow situations is the so-called *shock tube*, which shall be the subject of discussion in this report.

1.1. A shock tube

According to Professor Bram van Leer [5] building you own shock tube means: developing a (FORTRAN) computer program, which is able to simulate one-dimensional inviscid compressible flow in a tube. Given an initial flow state in the tube (the density, velocity and pressure are prescribed), which often means that in the tube several regions with different flow states exist, a shock tube solver is able to simulate the development of the flow in the tube in time. When a flow situation changes in time the flow is called *unsteady*. The equations that describe this compressible, inviscid, one-dimensional and unsteady flow are called the one-dimensional Euler equations (see next paragraph).

It is obvious that this flow problem does not quite look like a practical flow situation; the flow is inviscid, which means that drag due to viscosity is not simulated and the flow is only one-dimensional, while real-life is in fact three-dimensional. Fortunately building a shock tube certainly has its use.

The fact that the flow is inviscid seems like an enormous simplification, because drag plays an important role in fluid flows, but it is a fact that for high Reynolds numbers $Re = \rho UL/\mu$, the contribution of viscosity can be neglected to get a good impression of many flow features already.

One-dimensional means that the fluid can only flow in one direction. The radial direction of the tube is thus of no importance in this discussion[1]. Although most practical flow problems need a three-dimensional solution technique, it is wise to first experiment with one-dimensional techniques. Expanding the solver from one to two or even three dimensions is relatively easy.

1.2. The Euler equations

A compressible, inviscid, one-dimensional, unsteady flow can be described using the Euler equations. These equations are an approximation of the Navier-Stokes equations, which can be derived from these by neglecting friction and heat conduction. The Euler equations allow discontinuities in the solution, which means that shocks are a part of the solution. The equations in conservative form are given by:

$$\frac{\partial}{\partial t}\boldsymbol{q} + \frac{\partial}{\partial x}\boldsymbol{f}(\boldsymbol{q}) = 0. \tag{1.1}$$

Here $\boldsymbol{q}$ is the conservative state vector and $\boldsymbol{f}$ the flux vector, both consisting of three components for the one-dimensional version of the Euler equations. These vectors are given by:

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix} \quad \text{and} \quad \boldsymbol{f} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho u H \end{pmatrix}. \tag{1.2}$$

Their components include the three primary state variables being, ρ the density, u the velocity and p the pressure, the total energy E given by:

$$E = \frac{1}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}u^2, \tag{1.3}$$

and the total enthalpy H:

$$H = E + \frac{p}{\rho}. \tag{1.4}$$

Here γ is the ratio of specific heats, which is equal to 1.4 for air.

From the above it is obvious that all components of both the conservative state vector and the flux vector can be written as functions of the primary variables. For convenience's sake therefore the so-called primary state vector is introduced:

$$\boldsymbol{w} = \begin{pmatrix} \rho \\ u \\ p \end{pmatrix}. \tag{1.5}$$

1.3. Geometry and boundary conditions

It has already been mentioned that a shock tube has only one spatial dimension. For the shock tube problem to be relevant a length of the tube has to be specified. In most test problems (see the following chapters) the left end of the tube is taken at location $x = 0$ and the right end at $x = 1$. In this report the same convention shall be used.

Besides the geometry of the problem, one also needs *boundary conditions* that specify what happens with the flow near the boundaries of the problem, which are in this specific case the left and right ends of the tube. Two boundary conditions shall be applied separately:

[1]Although the word tube may be confusing it shall still be used here as long as there is no better description.

Soft boundary: The tube is not closed, such that the fluid can flow out of the tube at both ends.

Hard boundary: The tube is closed, such that waves that hit the end of the tube will be reflected.

Of course there are many more boundary conditions possible, think only of describing the velocity at the end of the tube by moving a piston in or out of the tube, but these fall out of the scope of this exercise.

1.4. The shock tube solver

The shock tube solver that has to be designed must be able, given the flow equations, the initial conditions and the boundary conditions, to simulate the flow at any location in the tube at any moment in time. An advantage of the Euler equations is that they can be solved exactly for certain problems. One of these specifically interesting problems is the so-called *Riemann problem*. A Riemann problem is a one-dimensional problem where initially two flow states are separated by a discontinuity, which will result in a characteristic flow situation when time passes. In the case of a shock tube, the solution of the Riemann problem also gives the solution of the flow in the tube.

Unfortunately, when the problem becomes more difficult than the one treated here, no exact solution to the flow equations can be found. A way out is the application of numerical methods (CFD). These solution methods are based on discretizing the continuous flow domain in discrete elements or volumes (these methods are therefore called Finite Element or Finite Volume Methods). The continuous flow equations can now be simplified by applying them to this discrete grid, leading to the so-called discretized equations. Using computers it is possible to solve this approximate flow problem numerically. The flow problem is called approximate because when discretizing the equations, errors are introduced.

To gain a better understanding of these numerical methods they are often applied to problems of which the exact solution is known, such that the numerical errors introduced by the approximate equations can be obtained and conclusions can be drawn about the accuracy of the numerical method. As mentioned before, the shock tube problem is such a problem with a known exact solution, e.g. the solution to the Riemann problem. In this report a shock tube solver will be designed that is able to both simulate the flow in the tube using the exact solution of the Riemann problem and the approximate solution obtained by the Finite-Volume Method (FVM).

1.5. Outline of the report

In the following chapter the so-called *Riemann problem*, the governing equations and the design of the exact Riemann solver are treated. Several test cases are evaluated using this solver. In chapter 3 the development of the shock tube program is described, focussing on the different elements of the solver. In this same chapter also the test results of several tests are discussed. The final chapter contains the conclusions and further remarks on the work done.

Chapter 2
The Riemann problem

Riemann's initial-value problem is one of the most fundamental problems in gasdynamics. The Riemann problem provides an exact and thus physical solution for the unsteady one-dimensional Euler equations, which makes this problem a frequently used solution method for different gasdynamics problems. Besides its application for solving physical flow problems, it has also been used for numerical solution techniques. Godunov was the first to apply the theory of the Riemann problem for solving fluxes on cell faces in a finite volume method for the one-dimensional Euler equations (see [3]). This Godunov method allows for accurate solutions of more complex flow problems, even in higher dimensions. In chapter 3, Godunov's approach shall be explained further as it will be used in the shock tube solver. But before this method can be applied, first the theory of the Riemann problem should be clear.

2.1. The principle of a Riemann problem

The Riemann problem means finding the solution to the one-dimensional unsteady Euler equations (non-linear), given two initial uniform states 1 and 4 or $left$ and $right$, which are separated by a discontinuity. When the membrane separating the two states is removed, two pressure waves[1] will appear, being either a shock wave (shock) or an expansion fan (expansion), which will start to run into the initial flow states resulting in two uniform states 2 and 3, in literature also called $left^*$ and $right^*$. These final states are separated by a contact discontinuity, which means that the pressure and the velocity of these states are equal, e.g. $p_2 = p_3 = p^*$ and $u_2 = u_3 = u^*$, but that the density jumps accross the discontinuity, $\rho_2 \neq \rho_3$. The general Riemann problem is depicted in figure 2.1:

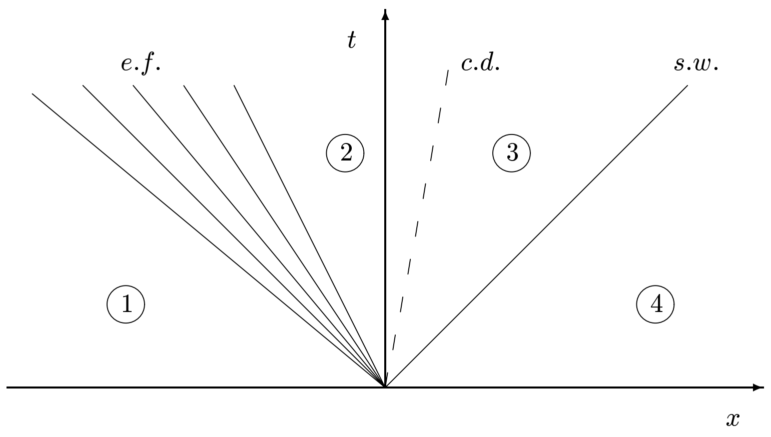

Figure 2.1: A Riemann problem. Two pressure waves, either a shock wave (s.w.) or an expansion fan (e.f.), separate the initial states 1 and 4 and the states 2 and 3. A contact discontinuity (c.d.) separates both final states.

It has already been mentioned that the solution to the Riemann problem is an exact solution to the non-linear,

[1]Do not confuse the term *wave* with the term *shock wave*, while the former is also used for *expansion fans*.

one-dimensional Euler equations. Solving the Riemann problem requires some knowledge about these equations and their so-called characteristic form.

2.2. Characteristic equations

Given the one-dimensional Euler equations (1.1) and (1.2), describing conservation of mass, momentum and energy respectively, it is possible to derive the following non-conservative form (for a detailed derivation see [1]):

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + \rho\frac{\partial u}{\partial x} = 0, \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0, \tag{2.2}$$

$$\frac{\partial s}{\partial t} + u\frac{\partial s}{\partial x} = 0. \tag{2.3}$$

Here the equation for conservation of energy e has been replaced by the equation for conservation of entropy s, stating that the entropy is constant along a particle path. Because the system of equations remains closed, this is a valid procedure.

Recalling from the theory of *Thermodynamics*, one finds that the change in entropy ds for an isentropic process (reversible and adiabatic) is proportional to $dp - a^2 d\rho$. Using this expression it is possible to derive the following three so-called *characteristic equations*:

$$\frac{\partial J^\pm}{\partial t} + (u \pm a)\frac{\partial J^\pm}{\partial x} = 0, \tag{2.4}$$

$$\frac{\partial s}{\partial t} + u\frac{\partial s}{\partial x} = 0, \tag{2.5}$$

where J^- and J^+ are called the *Riemann invariants* given by $dJ^\pm = du \pm \frac{dp}{\rho a}$. Here a is the speed of sound defined as $a = \sqrt{\frac{dp}{d\rho}\big|_s}$ where the s indicates an isentropic process. Further the so-called backward and forward running and entropy *characteristics* are defined as follows:

$$\Gamma^\pm : \frac{dx}{dt} = u \pm a, \tag{2.6}$$

$$\Gamma^0 : \frac{dx}{dt} = u. \tag{2.7}$$

From standard calculus we know that a quantity χ is constant when for that quantity holds that its derivative is equal to zero, $d\chi = 0$. Expanding the derivative in partial derivatives in both t and x direction leads to the following equation:

$$d\chi = \frac{\partial \chi}{\partial t}dt + \frac{\partial \chi}{\partial x}dx = 0, \tag{2.8}$$

from which we find the following expression for the direction along which the quantity χ is constant:

$$\frac{dx}{dt} = -\frac{\frac{\partial \chi}{\partial t}}{\frac{\partial \chi}{\partial x}}. \tag{2.9}$$

This observation and the definitions for the characteristics (2.6) and (2.7), combined with the characteristic equations (2.4) and (2.5) leads to the following conclusions for the one-dimensional Euler equations:

$$dJ^\pm = 0 \quad \text{along} \quad \Gamma^\pm \quad \text{with} \quad \frac{dx}{dt} = u \pm a, \tag{2.10}$$

$$ds = 0 \quad \text{along} \quad \Gamma^0 \quad \text{with} \quad \frac{dx}{dt} = u. \tag{2.11}$$

Because the expressions for the Riemann invariants $dJ^{\pm} = du \pm \frac{dp}{\rho a}$ can not be integrated directly, another simplification is necessary to obtain a more useful form of the derived relations. Therefore *homentropic* flow is assumed, which means that all particles have the same entropy. Some manipulations lead to the following expression for the Riemann invariants:

$$J^{\pm} = u \pm \frac{2}{\gamma - 1}a. \tag{2.12}$$

Using the expressions derived above it is possible to obtain a solution technique for the Riemann problem. But because the Riemann problem consists of two initially separated uniform states, one can make use of a special class of the characteristic equations, called *simple waves*, which shall now be explained in more detail.

2.3. Simple waves

Simple waves occur when one of the Riemann invariants $J^{\pm}$ is constant in a region of a homentropic flow domain. One Riemann invariant being constant leads to a relation between the velocity u and the speed of sound a of the form $J^{\pm} = u \pm \frac{2}{\gamma-1}a =$ constant, which automatically means that along the other characteristic both u and a are constant. Because the slope of the characteristic in the (t, x) plane is a function of u and a, these slopes are constant in the case of a simple wave, meaning that the characteristics are straight lines. Summarized:

- In a simple wave region where J^- is constant, the Γ^+ characteristics are straight lines.
- In a simple wave region where J^+ is constant, the Γ^- characteristics are straight lines.

Using the *Method of Characteristics* as described in [1] (MOC) and the simple wave theory it is now possible to solve the Riemann problem.

2.4. Solving the Riemann problem

In figure 2.1 one of the possible Riemann problems is sketched. Because the so-called *left* and *right-running* pressure waves[2] can be either a shock wave or an expansion fan many different situations are possible, of which a few are depicted in figure 2.2.

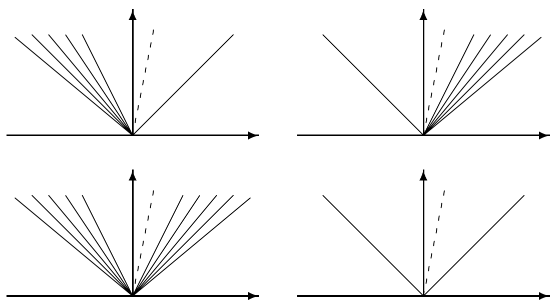

Figure 2.2: Some of the possible Riemann problems. Of course it is also possible that both waves lie on one side of the time axis, or that the wave lies exactly on the time axis.

To be able to solve all these possible problems a general theory is required that describes these different situations. Because the physics of a shock and an expansion are completely different also the solution methods for

[2]The terms left and right-running can be misleading because in certain cases both waves can be directed to one side.

these different waves differ. Therefore for both situations the governing equations will be treated, leading to two solution methods that can be applied when required. Solving the Riemann problem is then just a matter of determining which problem one is confronted with and then using the correct method.

2.4.1 Shocks: the Hugoniot curve

The one-dimensional Euler equations allow for discontinuities in the solution. A shock is an example of such a discontinuity, as is the contact discontinuity mentioned earlier. Using the Euler equations it is possible to derive the so-called *Rankine-Hugoniot* jump relations for mass, momentum and energy for a steady discontinuity:

$$\rho_1 u_1 = \rho_2 u_2, \tag{2.13}$$
$$p_1 + \rho_1 u_1^2 = p_2 + \rho_2 u_2^2, \tag{2.14}$$
$$\rho_1 u_1 H_1 = \rho_2 u_2 H_2. \tag{2.15}$$

These jump equations can be applied to a shock wave moving into a gas with state 1, also called the pre state, leading to a state 2 or $post$ state. In the *laboratory frame* of reference this situation is depicted in figure 2.3. But because the Rankine-Hugoniot equations only hold for steady shocks one has to transform the problem to the *shock frame* (see also figure 2.3).

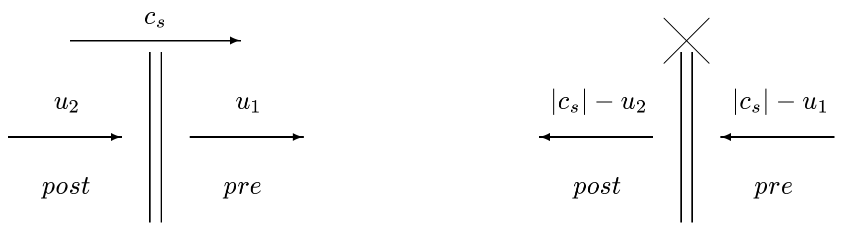

Figure 2.3: A shock moving into a pre state, leading to a $post$ state. *Left:* Laboratory frame of reference. *Right:* Shock frame of reference

Applying equations (2.13), (2.14) and (2.15) to this situation leads, after some manipulations, to the following relation between the change in pressure $\Delta p = p_{post} - p_{pre}$ and the change in velocity $\Delta u = u_{post} - u_{pre}$ over the shock:

$$\Delta p = \pm m \Delta u, \tag{2.16}$$

Where the $+$ is for a right-running shock and the $-$ for a left-running shock. The mass flux m can be obtained from the shock velocity c_s and the pressure rise Δp, using the following equations:

$$m = \rho_{pre}|c_s| = \rho_{pre} a_{pre} \sqrt{1 + \frac{\gamma+1}{2\gamma}\frac{\Delta p}{p_{pre}}}. \tag{2.17}$$

Because the mass flux can not become complex, we find the well-known physical property that the entropy always increases across a shock. Equations (2.16) and (2.17) describe all possible $post$ states that can be reached via a shock. The curve that connects the pre and $post$ states in the (p, u) diagram is called the *Hugoniot curve*. Solving for a $post$ state in the case of a shock thus means looking for that state that lies on the Hugoniot curve.

2.4.2 Expansions: the Poisson curve

Besides shocks also expansions can occur in a Riemann problem. Expansions are no discontinuities and can thus be solved using *regular* isentropic theory, described by the Euler equations. Using the simple wave theory given in the previous paragraph, the possible states after an expansion can be easily obtained. To derive the governing equations a right-running expansion fan is treated, running into the by now well-known pre state, while leaving a $post$ state behind (see figure 2.4).

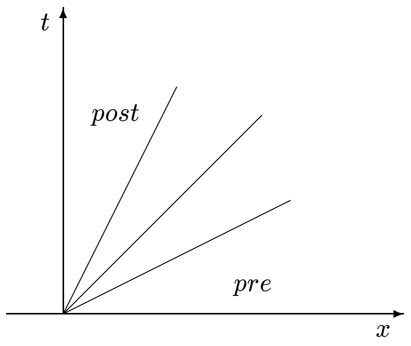

Figure 2.4: An expansion moving into a pre state, leading to a $post$ state.

Along a Γ^- characteristic, which moves from the pre to the $post$ state, holds that the Riemann invariant $J^- = u - \frac{2}{\gamma-1}a$ is constant. Using this condition and the isentropic relations, something similar to (2.16) can be derived:

$$\Delta p = \pm m \Delta u, \tag{2.18}$$

where the $+$ stands for a right-running expansion and the $-$ for a left-running expansion. The mass flux m is equal to:

$$m = \rho_{pre} a_{pre} \frac{\gamma - 1}{2\gamma} \frac{1 - \frac{p_{post}}{p_{pre}}}{1 - \left(\frac{p_{post}}{p_{pre}}\right)^{\frac{\gamma-1}{2\gamma}}}. \tag{2.19}$$

The curve in the p, u diagram connecting all possible states given by equations (2.18) and (2.19) is called the *Poisson curve*. Solving for the $post$ state in case of an expansion means finding that state that lies on the Poisson curve.

Further it can be mentioned that the Poisson curve lies above the Hugoniot curve everywhere in the (p, u) diagram.

2.4.3 Solving for the final states

Given a Riemann problem with a certain combination of two waves, the theory of the previous two sections can be used to determine the flow conditions in the final states 2 and 3. Because these states are separated by a contact discontinuity, the pressures and velocities in both regions are the same. This means that both the $left$ and the $right\ post$ states are the same, when only looking at the (p, u) diagram. Solving a Riemann problem thus means solving for the final state (p^*, u^*) in this diagram. Because all possible states lie either on a *Hugoniot* (for a shock) or on a *Poisson* (for an expansion) curve, solving means finding the intersection of the two curves that come from the two initial states. An example is given below:

Suppose that the $left\ pre$ state, indicated by the number 1, has conditions p_1 and u_1 which are different from the conditions in the $right\ pre$ state, p_4 and u_4. Assume for instance that the p_1 is higher than p_4 but that the velocities u_1 and u_4 are equal. Because state 1 is the left state, from here a left-running wave departs, while from state 4 a right-running wave departs. Because the pressure in state 1 is higher than in state 4 the left-running curve will be a Poisson curve, meaning an expansion, and the right-running curve will be a Hugoniot curve, meaning a shock. The *post* state conditions lie on the intersection of the Poisson and the Hugoniot curve. This example is depicted in figure 2.5.

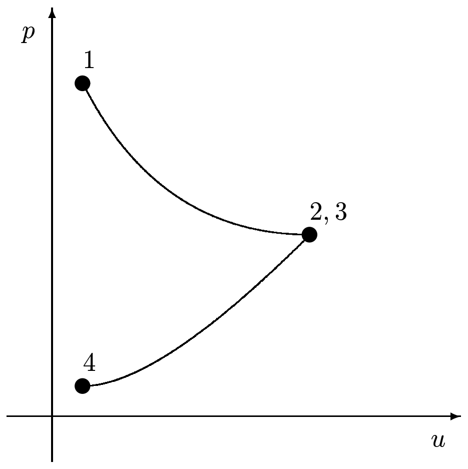

Figure 2.5: An example of a (p, u) diagram of a Riemann problem.

The density in the final states 2 and 3 can be obtained from either the shock jump relations in the case of a shock or from the isentropic relations in case of an expansion. These equations are respectively:

$$\rho_{post} = \rho_{pre} \frac{1 + \frac{\gamma+1}{\gamma-1} \frac{p^*}{p_{pre}}}{\frac{\gamma+1}{\gamma-1} + \frac{p^*}{p_{pre}}}, \tag{2.20}$$

$$\rho_{post} = \rho_{pre} \left(\frac{p^*}{p_{pre}} \right)^{1/\gamma}. \tag{2.21}$$

Given the density and the pressure also the speed of sound can be determined using the relation:

$$a = \sqrt{\frac{\gamma p}{\rho}}. \tag{2.22}$$

In this way all the conditions in the uniform states 2 and 3 have been determined. The only thing that remains is the determination of the conditions in an expansion fan. While a shock is infinitely thin (in reality a shock does have a certain thickness, but this shall be neglected in this discussion), an expansion is obviously not. The region lying between the first and the last characteristic of an expansion is governed by the isentropic relations, such that the following relations hold for the pressure and the density in the expansion fan:

$$p = p_{pre} \left(\frac{a}{a_{pre}} \right)^{\frac{2\gamma}{\gamma-1}}, \tag{2.23}$$

$$\rho = \rho_{pre} \left(\frac{a}{a_{pre}} \right)^{\frac{2}{\gamma-1}}, \tag{2.24}$$

which means that the speed of sound in the expansion fan should be known. Using the Method of Characteristics expressions can be derived for the velocity and the speed of sound in a fan at a certain location and time.

The derivation will not be shown here for levity's sake. The resulting equations for a left-running expansion are:

$$u = \frac{2}{\gamma+1}\left(\frac{x}{t} \mp a_{pre}\right) + \frac{\gamma-1}{\gamma+1}u_{pre}, \tag{2.25}$$

$$a = \pm\frac{\gamma-1}{\gamma+1}\left(\frac{x}{t} - u_{pre}\right) + \frac{2}{\gamma+1}a_{pre}, \tag{2.26}$$

where the $-$ means a left-running expansion and a $+$ a right-running expansion. The $\mp$ indicates that in this case the $-$ and the $+$ should be switched.

2.4.4 Solving for the wave locations

The only thing that remains is determining where the waves and the contact discontinuity are at a certain moment in time because these separate the different regions (1, 2, 3, 4 and if present the inner region of an expansion fan). Knowing the location of the different waves means that the complete distribution of all primary variables in space at any moment in time is known.

Shock The shock speed follows from the expression for c_s. But because this is the shock velocity in the shock frame of reference, the velocity of the flow should be added:

$$\left.\frac{dx}{dt}\right|_s = u_{pre} \pm |c_s| = u_{pre} \pm a_{pre}\sqrt{1 + \frac{\gamma+1}{2\gamma}\frac{\Delta p}{p_{pre}}}, \tag{2.27}$$

where the $\pm$ speaks for itself.

Expansion The direction of the first and the last characteristic of an expansion follows from characteristic theory, being respectively:

$$\left.\frac{dx}{dt}\right|_{first} = u_{pre} \pm a_{pre}, \tag{2.28}$$

$$\left.\frac{dx}{dt}\right|_{last} = u^* \pm a_{post}. \tag{2.29}$$

Contact discontinuity Also from characteristic theory follows that the direction of the entropy wave or contact discontuinity is:

$$\left.\frac{dx}{dt}\right|_0 = u^*. \tag{2.30}$$

Because all waves emanate from the origin of the (x, t) diagram, solving for the location of a wave at a given time means multiplying the slopes $\frac{dx}{dt}$ of the different waves with the time t of interest. In this way the Riemann problem can be solved completely.

2.5. The exact Riemann solver

Using the theory of the previous sections and the solution method described, it is now possible to develop a program that is able to solve, given the two initial Riemann states, for the final state and thus the distribution of the primary variables at a certain moment in time. Such a program has to consist of several elements, which shall be discussed here.

2.5.1 Solving for the final states

The main goal of a Riemann problem is solving for the final states behind the waves. As shown in the previous sections, the way to start this procedure is determining where the two curves (Hugoniot or Poisson), going through the initial states, intersect. Graphically this approach is straightforward and easy to perform, but on a computer, e.g. numerically, another approach is required. In equations (2.16) and (2.18) it was shown that the pressure p^* for both a shock and an expansion is given as a function of the initial conditions and the final velocity u^*. In the case of a Riemann problem consisting of two of such waves, it is thus possible to obtain an expression for the pressure p^* independent of u^*, being:

$$p^* = \frac{m_1 p_4 + m_4 p_1 - m_1 m_4 (u_4 - u_1)}{m_1 + m_4}. \tag{2.31}$$

In the same manner the equation for the velocity u^* as a function of the initial conditions becomes:

$$u^* = \frac{m_1 u_1 + m_4 u_4 - (p_4 - p_1)}{m_1 + m_4}. \tag{2.32}$$

Because m_1 and m_4, as defined in (2.17) and (2.19), contain the final pressure p^* these expressions still give no explicit equation for this pressure. But, because expression (2.31) consists only of the initial conditions and p^*, it is possible to obtain the final state by using an iteration method. When an accurate enough approximation of the pressure is found in this way, the velocity follows immediately from (2.32).

And when the pressure and the velocity in the regions 2 and 3 are known, the density in these regions follows directly from the relations (2.20) and (2.21). In the same way, the conditions in an expansion fan can be determined.

Before discussing the iteration procedures, it is wise to treat two special situations. The first situation is when the flow in both initial regions is supersonic in the same direction. In this case no information can be transferred upstream because information (waves) travel always with the speed of sound, which is in this special case lower than the flow velocity. When this occurs the final conditions can be taken equal to the $left$ initial states in the case of a supersonic flow to the $right$, and to the $right$ initial conditions in the case of a supersonic flow to the $left$.

The second special situation is called *cavitation*, which means that the density becomes negative. Of course this is an unphysical situation and should therefore always be avoided. To do so, a check should be implemented in the Riemann solver that stops the calculations when:

$$u_1 + \frac{2}{\gamma - 1} a_1 < u_4 + \frac{2}{\gamma - 1} a_4. \tag{2.33}$$

2.5.2 Iterating for the pressure

For obtaining the pressure p^* several iteration methods can be used. Well-known iteration methods are the *Fixed-Point* method, the *Secant* method or the *Newton-Raphson* method (for a detailed discussion of these methods the reader is referred to [2]). All these methods are so-called *Root-finding* methods. Given a function $f(x)$, they find that value of x for which $f(x) = 0$. Because the expression for p^* according to (2.31) can be written in this form, solving for the pressure means finding the root of:

$$g(p^*) = f(p^*) - p^* = 0, \tag{2.34}$$

where the function $f(p^*)$ is the righthand-side of (2.31). The fastest of these is the method of Newton-Raphson, which uses the following iteration scheme:

$$p^*_n = p^*_{n-1} - \frac{g(p^*_{n-1})}{g'(p^*_{n-1})}, \tag{2.35}$$

where n indicates the number of iterations performed. The iteration starts with an initial guess for the pressure p_0^*. Because this method uses derivatives, which are difficult to determine exactly, it is better to use the approximate version of this method, the Secant method. In this method the derivative $g'(p_{n-1}^*)$ is approximated by its discrete value, leading to the following expression:

$$p_n^* = p_{n-1}^* - \frac{g(p_{n-1}^*)(p_{n-1}^* - p_{n-2}^*)}{g(p_{n-1}^*) - g(p_{n-2}^*)}. \tag{2.36}$$

The Secant method requires two initial values for the pressure instead of one as was the case for Newton-Raphson. A good choice for one of these values would be the value of the final pressure in the case of two expansion fans. It has already been mentioned that the solution given by the Poisson curves always lies above the Hugoniot curves, such that the pressure that follows in this case can be seen as an upper-bound of the iteration domain. Because a Poisson curve can also be used for compression (compression fan), the solution of this approach gives an accurate initial guess for the iteration procedure. Taking two expansions allows us to write explicitly for p^*:

$$p^* = \left(\frac{\frac{\gamma-1}{2}(u_1 - u_4) + a_1 + a_4}{a_1 p_1^{\frac{2\gamma}{\gamma-1}} + a_4 p_4^{\frac{2\gamma}{\gamma-1}}} \right)^{\frac{2\gamma}{\gamma-1}}. \tag{2.37}$$

In the case of two expansion fans, the solution of this equation gives immediately the exact solution, requiring no further iteration. Thus it is wise to implement a check into the Riemann solver which determines whether two expansion fans are present. For the second initial guess the lower-bound of the iteration domain can be taken. This lower bound follows from linear theory. Here the mass flows do not depend on the pressure in the final states. These mass flows are defined as:

$$m_1 = \rho_1 a_1, \tag{2.38}$$

$$m_4 = \rho_4 a_4. \tag{2.39}$$

Using the equations (2.16) and (2.18) the conditions in the regions 2 and 3 can be determined. Because the solution of linear theory differs quite a bit from non-linear theory, it is wise to take the second initial guess closer to the first initial guess, but this depends on the Riemann problem treated.

2.5.3 Determining the complete solution

As mentioned in the previous sections, the following step is determining what the speeds of the different waves of the Riemann problem are. Knowing the location of these waves means knowing where the different regions of the Riemann problem lie, such that the complete solution at a given moment in time can be determined. The most general representation of a Riemann problem and its waves is depicted in figure 2.6.
In the case of an expansion, the speed of the first and last characteristics are given by (2.28) and (2.29), enclosing a region (1/2 or 3/4) governed by the isentropic relations. In the case of a shock, the two waves indicated are actually the same, with a speed given by (2.27). The contact discontinuity moves with a speed given by (2.30).

Knowing the locations of the waves and the conditions inside the different regions allows to determine the conditions over the whole length of the tube. This means that all Riemann problems can be solved using the exact Riemann solver developed here.

2.6. Test cases

To test the exact Riemann solver for its performance several test cases are available. Here the test cases from [7] are used. These test cases treat a one-dimensional tube with length 1, with both ends at respectively $x = 0$ and $x = 1$. The two initial states 1 and 4 are separated in the center of the tube at $x = 0.5$. Running the calculations in time results in different Riemann problems for the different initial situations.

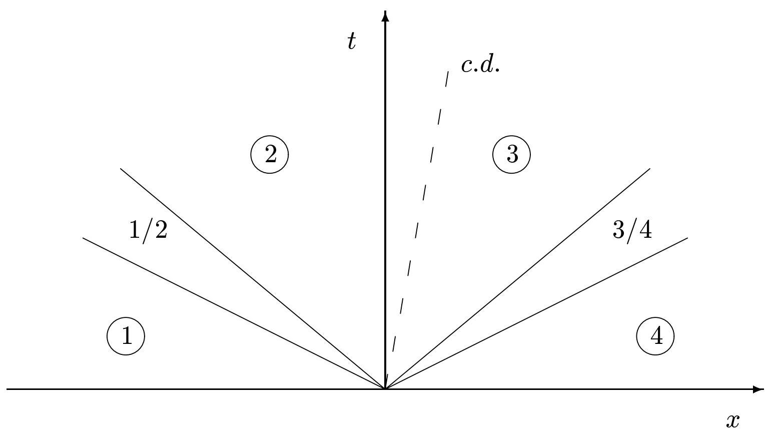

Figure 2.6: The different waves in a Riemann problem.

For all tests the conditions of the final states 2 and 3 are calculated. Also the density ρ, velocity u and pressure p distributions in the tube are plotted. Further the, for each Riemann problem characteristic, distribution of waves is plotted. The results obtained will be checked with the results of [7] (pages 129-133). Because others excessively used these tests, the results that should be obtained are trustworthy. If the program developed here complies with these expected results, it is assumed to be error-free and accurate enough for further applications.

2.6.1 Geometry

For the geometry of the exact solver the same tube is used as for the shock tube program still to be developed. This because the numerical results of this shock tube solver can in this way be compared to the exact results of the exact Riemann solver. A tube of length 1, with both ends at respectively $x = 0$ and $x = 1$, is used for the calculations. The exact solver described here will be able to give at the required moment in time the distribution of primary variables ρ, u and p in the tube. Making the solver as general as possible, allows it to be easily implemented in the shock tube program.

2.6.2 Test 1, Sod's problem

The initial conditions for this well-known Riemann test problem are given in table 2.1. This problem starts with both a density and a pressure jump over the initial discontinuity. Given these conditions one expects a left-running expansion and a right-running shock.

	Left (1)	Right (4)
ρ	1.0	0.125
u	0.0	0.0
p	1.0	0.1

Table 2.1: Initial data for test 1.

Applying the initial conditions and running the program leads to the final conditions as given in table 2.2. These are exactly the same as the ones given in [7] on page 133.
In figure 2.7 the distributions of the primary variables are plotted at time $t = 0.25$. One clearly sees the expansion fan moving to the left, while the shock and the contact discontinuity move to the right. In the density distribution the contact discontinuity is visible while, as it should be, it is invisible in the plots of the velocity and pressure. As expected, the velocity varies linearly in the expansion fan.

p^*	u^*	ρ_2	ρ_3
0.30313	0.92745	0.42632	0.26557

Table 2.2: Results for test 1.

2.6.3 Test 2, two expansions

The initial conditions for this Riemann test problem are given in table 2.3. This problem starts with a velocity jump over the initial discontinuity. Given these conditions one expects two expansions; one running to the left and one to the right.

	Left (1)	Right (4)
ρ	1.0	1.0
u	-2.0	2.0
p	0.4	0.4

Table 2.3: Initial data for test 2.

Applying the initial conditions and running the program leads to the final conditions as given in table 2.4. The results of test 2 are also the same as the ones given in [7].

p^*	u^*	ρ_2	ρ_3
0.00189	0.00000	0.02185	0.02185

Table 2.4: Results for test 2.

In figure 2.8 the distributions of the primary variables are plotted at time $t = 0.15$. One clearly sees the expansion fans moving to the left and to the right. The pressure becomes zero in the middle of the tube indicating that vacuum occurs. Because the density also approaches zero, the solver has to be cavitation proof. To avoid the calculations being stopped, the program does not allow the density and the pressure to become zero or negative, but gives it a small non-zero value in these cases.

2.6.4 Test 3, expansion and shock

The initial conditions for this Riemann test problem are given in table 2.5. This problem starts with a very strong pressure jump over the initial discontinuity. One expects an expansion to the left and a shock to the right, the same as in test 1. Only now the shock and expansion are much stronger due to the larger pressure jump. The wave speeds are thus expected to be much larger.
Applying the initial conditions and running the program leads to the final conditions as given in table 2.6.
In figure 2.9 the distributions of the primary variables are plotted at time $t = 0.012$. A very strong shock moves to the right and an expansion to the left. Also the contact discontinuity is extremely strong in this case. Another interesting thing to mention is that the contact discontinuity and the shock wave lie very close to each other.

2.6.5 Test 4, shock and expansion

The initial conditions for this Riemann test problem are given in table 2.7. This problem also starts with a very strong pressure jump over the initial discontinuity. An expansion to the right and a shock to the left are expected, which is exactly the opposite of test 3.
Applying the initial conditions and running the program leads to the final conditions as given in table 2.8. Again these results agree with the ones in [7].
In figure 2.10 the distributions of the primary variables are plotted at time $t = 0.035$. This test is the opposite of test 3 as was already mentioned. A strong shock and contact discontinuity move to the left while an expansion moves to the right.

	Left (1)	Right (4)
ρ	1.0	1.0
u	0.0	0.0
p	1000.0	0.01

Table 2.5: Initial data for test 3.

p^*	u^*	ρ_2	ρ_3
460.894	19.5975	0.57506	5.99924

Table 2.6: Results for test 3.

	Left (1)	Right (4)
ρ	1.0	1.0
u	0.0	0.0
p	0.01	100.0

Table 2.7: Initial data for test 4.

p^*	u^*	ρ_2	ρ_3
46.0950	-6.19633	5.99242	0.57511

Table 2.8: Results for test 4.

2.6.6 *Test 5, two shocks*

The initial conditions for this Riemann test problem are given in table 2.9.

	Left (1)	Right (4)
ρ	5.99924	5.99242
u	19.5975	-6.19633
p	460.894	46.0950

Table 2.9: Initial data for test 5.

Applying the initial conditions and running the program leads to the final conditions as given in table 2.8.

p^*	u^*	ρ_2	ρ_3
1691.64	8.68975	14.2823	31.0426

Table 2.10: Results for test 5.

In figure 2.11 the distributions of the primary variables are plotted at time $t = 0.035$. In the case of test 5, two shocks appear that both run to the right. Characteristic for this test is that the conditions at the time-axis are always constant and equal to the conditions in domain 1. This is due to the fact that the *left-running* shock is actually moving to the *right*, such that the time-axis lies in the left initial region. The same situation occurs in the case of a supersonic flow to the right in both initial regions.

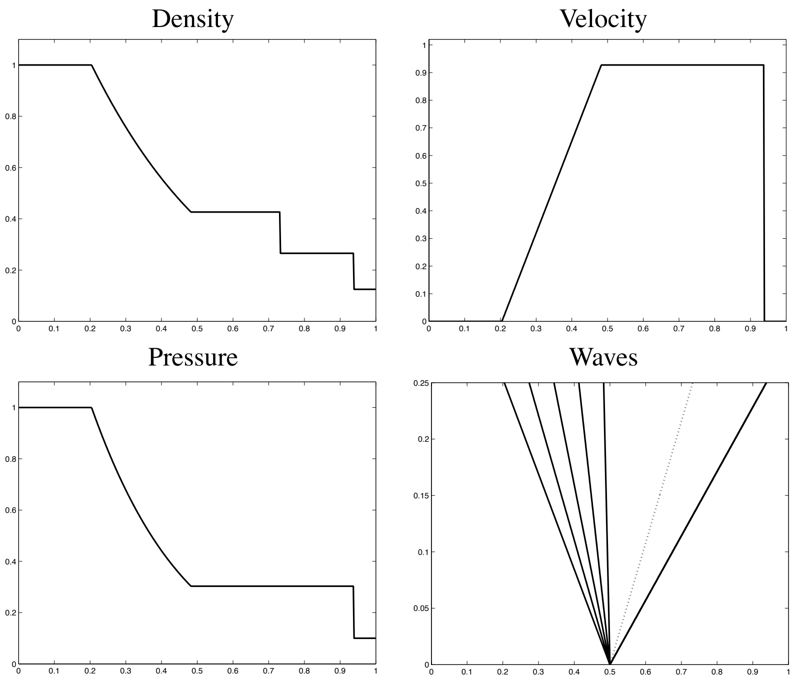

Figure 2.7: Results of test 1. Primary variables and wave propagation against the x location in the tube at $t = 0.25$.

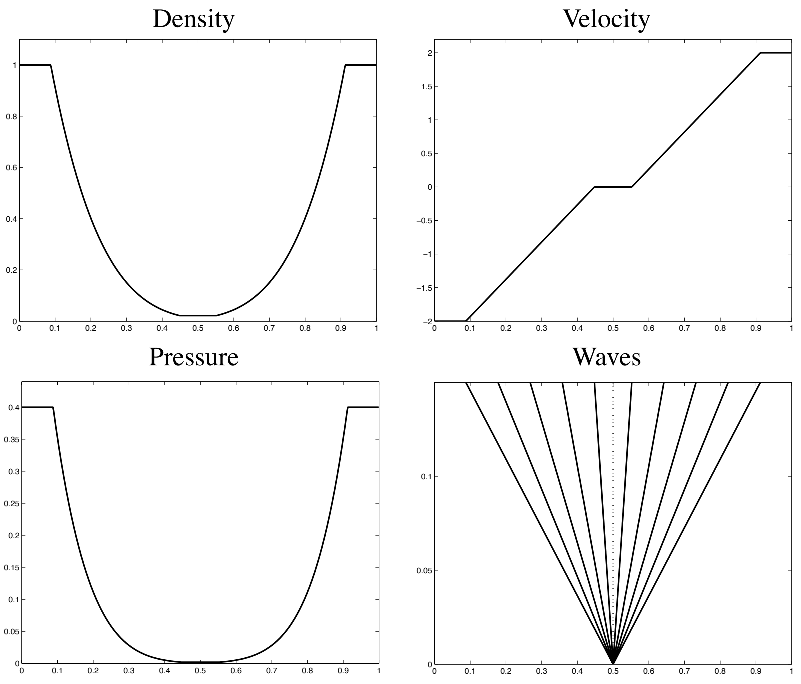

Figure 2.8: Results of test 2. Primary variables and wave propagation against the x location in the tube at $t = 0.15$.

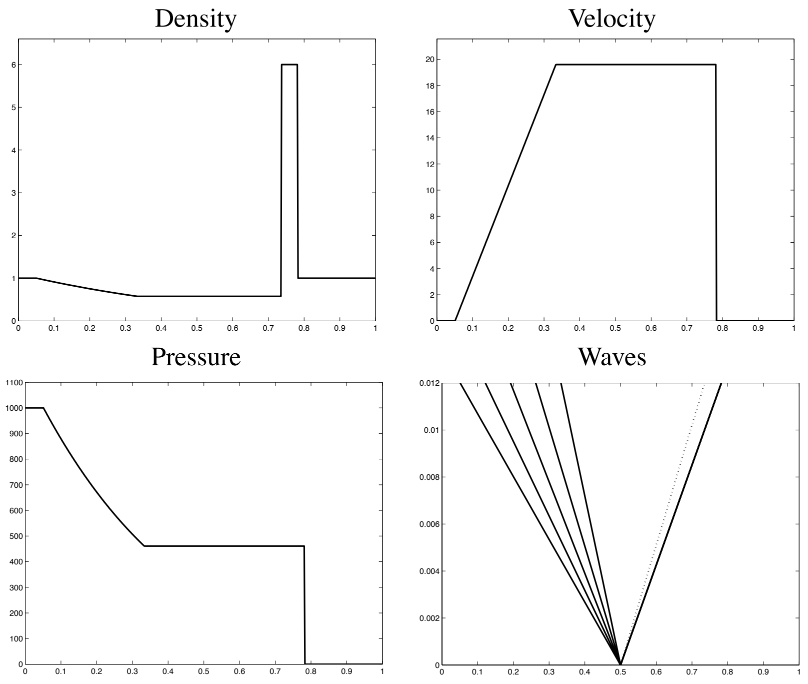

Figure 2.9: Results of test 3. Primary variables and wave propagation against the x location in the tube at $t = 0.012$.

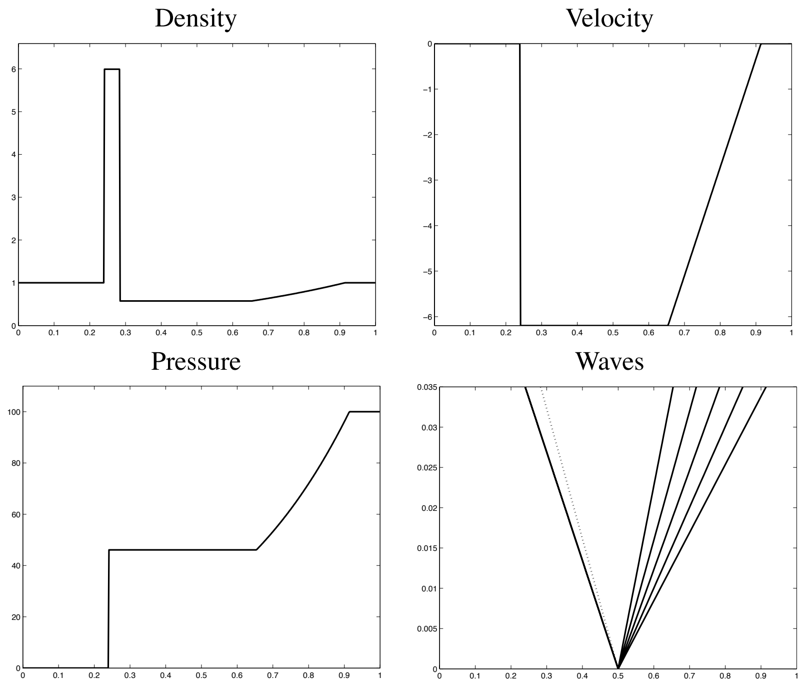

Figure 2.10: Results of test 4. Primary variables and wave propagation against the x location in the tube at $t = 0.035$.

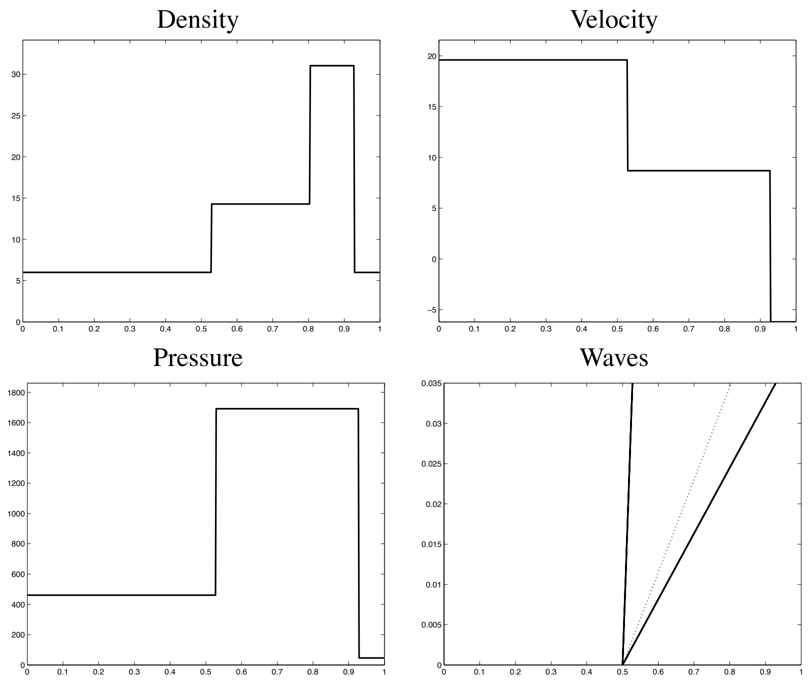

Figure 2.11: Results of test 5. Primary variables and wave propagation against the x location in the tube at $t = 0.035$.

Chapter 3
The shock tube solver

A shock tube solver is able to solve the inviscid, compressible, unsteady, one-dimensional Euler equations for a tube with given initial conditions. Although in this specific situation the exact solution is known, in many cases this is not the case, such that the use of approximate methods is inevitable. A suitable method to discretize the Euler equations is a Finite Volume Method (FVM). This method requires knowledge about the fluxes over the cell-interfaces and thus about the flow conditions on these interfaces. To obtain these interface conditions, given the conditions in the cell-centers, several methods are available, of which two shall be implemented in the shock tube solver developed here. The first is Godunov's method based on the exact solutions of the Riemann problem as discussed in the previous chapter. The second is Roe's method, which is based on an approximate version of a Riemann solver. Because the unsteady Euler equations are treated here, a method for time marching should be used that advances the solution at a certain time level to the next level. For this time-marching, Hancock's scheme will be used. Many others are possible but this falls out of the scope of this research.

Because the exact solutions to several shock tube test cases are known (chapter 2) the approximate shock tube solver developed here can easily be tested for its performance. A way of measuring the performance is testing for convergence of the error (the difference between the exact and the approximate solution) with decreasing size of the discrete volumes, by calculating the order of convergence. Also the double blast wave in a close tube problem is treated. As a final test the interaction of two simple waves is discussed.

3.1. Discretizing the equations

The discretization of the Euler equations given by (1.1) is done using a FVM. The tube running from $x \in [0, 1]$, is divided in J uniform volumes with length $\Delta x = 1/J$, as is depicted in figure 3.1. Two extra *ghost-cells* are added at both ends to ease the implementation of the boundary conditions. The flow variables, $\boldsymbol{w} = (\rho, u, p)^T$, are defined in the cell-centres with location $x_j = (j - 0.5)\Delta x$ with $j = 0, 1, \ldots, J + 1$.

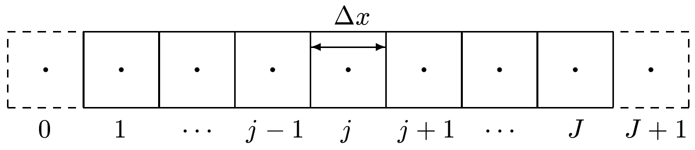

Figure 3.1: The discretized shock tube consisting of J uniform volumes and two ghost-cells at the ends of the tube. The flow variables are defined in the cell-centers.

Because the Euler equations allow discontinuities such as shocks, the discretization method used should be a conservative numerical method. Non-conservative methods do not converge when shocks are present. Therefore the derivation of the discretized equations starts with the integral form of the Euler equations, obtained after integration of (1.1) over one cell of the discretized tube $\Omega_j = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}]$ and application of Gauss' divergence theorem for a scalar:

$$\int_{\Omega_j} \frac{\partial \boldsymbol{q}(x,t)}{\partial t} dx = \boldsymbol{f}\left(\boldsymbol{q}(x_{j-\frac{1}{2}}, t)\right) - \boldsymbol{f}\left(\boldsymbol{q}(x_{j+\frac{1}{2}}, t)\right). \tag{3.1}$$

Because the cell does not change size in time, it is allowed to interchange the time-differentiation and the integration, such that:

$$\frac{\partial}{\partial t}\int_{\Omega_j} \boldsymbol{q}(x,t)dx = \boldsymbol{f}\left(\boldsymbol{q}(x_{j-\frac{1}{2}},t)\right) - \boldsymbol{f}\left(\boldsymbol{q}(x_{j+\frac{1}{2}},t)\right). \tag{3.2}$$

The value of $\boldsymbol{q}_j$ in the cell-center is taken equal to the average of $\boldsymbol{q}$ over the cell:

$$\boldsymbol{q}_j = \frac{1}{\Delta x}\int_{\Omega_j} \boldsymbol{q} dx, \tag{3.3}$$

such that one finally ends up with the following Finite Volume discretization:

$$\frac{\partial \boldsymbol{q}_j}{\partial t}\Delta x = \boldsymbol{f}\left(\boldsymbol{q}(x_{j-\frac{1}{2}},t)\right) - \boldsymbol{f}\left(\boldsymbol{q}(x_{j+\frac{1}{2}},t)\right). \tag{3.4}$$

3.2. Hancock's scheme

Hancock's scheme is used here for both the temporal and spatial discretisation of equation (3.4). This scheme is a *predictor-corrector* type of MUSCL scheme, which advances the solution $\boldsymbol{q}_j$ at time-level n to time-level $n+1$ using an estimate of the fluxes at time-level $n+\frac{1}{2}$. Starting from the Finite-Volume equation, the time-derivative is discretised using a time-centered difference scheme:

$$\frac{\partial \boldsymbol{q}_j}{\partial t} = \frac{\boldsymbol{q}_j^{n+1} - \boldsymbol{q}_j^n}{\Delta t}. \tag{3.5}$$

This time-centering implies that the physical fluxes have to be discretised by the numerical fluxes at time-level $n+\frac{1}{2}$, which can be written as:

$$\boldsymbol{f}\left(\boldsymbol{q}(x_{j-\frac{1}{2}},t_{n+\frac{1}{2}})\right) = \boldsymbol{f}^{n+\frac{1}{2}}_{j-\frac{1}{2}}, \qquad \boldsymbol{f}\left(\boldsymbol{q}(x_{j+\frac{1}{2}},t_{n+\frac{1}{2}})\right) = \boldsymbol{f}^{n+\frac{1}{2}}_{j+\frac{1}{2}}. \tag{3.6}$$

Inserting both the temporal and the spatial discretizations into the FV discretization leads to the final form of the discretized Euler equations:

$$\boldsymbol{q}_j^{n+1} = \boldsymbol{q}_j^n + \frac{\Delta t}{\Delta x}\left[\boldsymbol{f}^{n+\frac{1}{2}}_{j-\frac{1}{2}} - \boldsymbol{f}^{n+\frac{1}{2}}_{j+\frac{1}{2}}\right]. \tag{3.7}$$

Marching in time thus requires knowledge of the fluxes at the intermediate time-level. The key lies in chosing the correct method for solving for these interface fluxes.

3.2.1 Flux calculations

Using equations (1.2) the interface fluxes at $n+\frac{1}{2}$ can be calculated from the interface conditions at the same time-level. The approach that shall be taken here for determining these interface conditions is using either an exact (Godunov's) or an approximate (Roe's) Riemann solver. In the previous chapter the Riemann problem was treated extensively. The main idea of a Riemann problem was that using both a $left$ and a $right$ initial state, the final state could be obtained. Applied to the discretized Euler equations of the shock tube this means that the interface fluxes at time-level $n+\frac{1}{2}$ can be calculated from both the $left$ and $right$ interface conditions at the same time level, indicated here with a *tilde*:

$$\boldsymbol{f}^{n+\frac{1}{2}}_{j-\frac{1}{2}} = \boldsymbol{f}\left(\tilde{\boldsymbol{q}}_{j-\frac{1}{2},L}, \tilde{\boldsymbol{q}}_{j-\frac{1}{2},R}\right), \qquad \boldsymbol{f}^{n+\frac{1}{2}}_{j+\frac{1}{2}} = \boldsymbol{f}\left(\tilde{\boldsymbol{q}}_{j+\frac{1}{2},L}, \tilde{\boldsymbol{q}}_{j+\frac{1}{2},R}\right). \tag{3.8}$$

Determination of the $left$ and $right$ initial conditions at the intermediate time-level is done using both a predictor and corrector step.

3.2.2 Predictor step

The method of Hancock uses the non-conservative equations in terms of the primary state vector $\boldsymbol{w}$:

$$\frac{\partial}{\partial t}\boldsymbol{w} + (\boldsymbol{A})_j \frac{\partial}{\partial x}\boldsymbol{w} = 0, \tag{3.9}$$

where the coefficient matrix reads:

$$\boldsymbol{A} = \begin{pmatrix} u & \rho & 0 \\ 0 & u & \frac{1}{\rho} \\ 0 & \rho a^2 & u \end{pmatrix}. \tag{3.10}$$

The *predictor step*, which can be seen as an interpolation for $\boldsymbol{w}$ at half a time step further, can be obtained by taking the forward Euler discretisation for the time derivative and a central discretisation for the spatial derivative:

$$\tilde{\boldsymbol{w}}_j = \boldsymbol{w}_j - \frac{\Delta t}{2\Delta x}\boldsymbol{A}\delta\boldsymbol{w}_j. \tag{3.11}$$

Here $\delta\boldsymbol{w}$ is defined as the *step size*, which can be written as:

$$\delta\boldsymbol{w}_j = ave(\boldsymbol{w}_j - \boldsymbol{w}_{j-1}, \boldsymbol{w}_{j+1} - \boldsymbol{w}_j) = ave(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2). \tag{3.12}$$

Taking an average based on the value of the gradients allows us to limit the value of $\delta\boldsymbol{w}$ and thus reduce sharp gradients. In other words, it allows us to limit the maximum values of the state variables. That is why some of these averages are called limiters. In the following section a few averages are discussed. Note that taking the step equal to zero results in the first-order upwind method, which is implemented in the solver but shall not be elaborated here.

3.2.3 Averaging

The following averaging schemes have been implemented in the program:

Algebraic average The value of the step size is equal to the algebraic average of the two gradients::

$$ave(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) = \frac{\Delta\boldsymbol{w}_1 + \Delta\boldsymbol{w}_2}{2}. \tag{3.13}$$

Different from the other three averages used, the Algebraic average is linear.

Double Minmod Limiter The step size is equal to zero, when the two gradients have an opposite sign, e.g. when a maximum occurs. And when the two gradients have equal sign the step size is equal to the minimum of the following three possibilities: the algebraic average, two times $\Delta\boldsymbol{w}_1$ or two times $\Delta\boldsymbol{w}_2$. This can be expressed as:

$$ave(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) = \begin{cases} \text{minmod}\left(\frac{\Delta\boldsymbol{w}_1+\Delta\boldsymbol{w}_2}{2}, 2\Delta\boldsymbol{w}_1, 2\Delta\boldsymbol{w}_2\right) & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 > 0, \\ 0 & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 \leq 0. \end{cases} \tag{3.14}$$

Closely related to this limiter is the normal *Minmod* limiter. This limiter is more strict than the Double Minmod, because the factors 2 in front of the gradients are not used. For the calculations performed, the Double Minmod suffices.

Superbee Limiter The Superbee limiter also sets the step size equal to zero when a maximum occurs. But when the gradients have equal signs, the minimum value of the following is taken: the maximum of the gradients or the minimum of two times the gradients. Thus:

$$ave(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) = \begin{cases} \text{minmod}\left(\text{maxmod}\left(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2\right), \text{minmod}\left(2\Delta\boldsymbol{w}_1, 2\Delta\boldsymbol{w}_2\right)\right) & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 > 0, \\ 0 & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 \leq 0. \end{cases} \tag{3.15}$$

Koren's Limiter This limiter has been developed by Koren [4]. This limiter chooses the minimum of the following three possibilities: two times the first gradient, the sum of one third the first and two third the second gradient or two times the second gradient:

$$ave(\Delta\boldsymbol{w}_1, \Delta\boldsymbol{w}_2) = \begin{cases} \text{minmod}\left(2\Delta\boldsymbol{w}_1, \frac{1}{3}\Delta\boldsymbol{w}_1 + \frac{2}{3}\Delta\boldsymbol{w}_2, 2\Delta\boldsymbol{w}_2\right) & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 > 0, \\ 0 & \Delta\boldsymbol{w}_1\Delta\boldsymbol{w}_2 \leq 0. \end{cases} \tag{3.16}$$

This limiter follows the so-called $\kappa = \frac{1}{3}$ scheme. Koren's limiter should perform somewhat better than the other two limiters. Therefore, when stated otherwise, Koren's limiter is used in the calculations. At the end of this chapter a comparison is made between the different limiters.

3.2.4 Corrector step

Using the predictor step and the value of the step size it is possible to determine the time-centered interface values at each interface, e.g., the initial states for the Riemann solver can be calculated. Each cell Ω_j delivers a $right$ interface value for the interface to the left of this cell and a $left$ interface value for the interface to the right. These can be written as:

$$\tilde{\boldsymbol{w}}_{j-\frac{1}{2},R} = \tilde{\boldsymbol{w}}_j - \frac{1}{2}\delta\boldsymbol{w}_j, \qquad \tilde{\boldsymbol{w}}_{j+\frac{1}{2},L} = \tilde{\boldsymbol{w}}_j + \frac{1}{2}\delta\boldsymbol{w}_j. \tag{3.17}$$

Using the primary state variables at the interfaces allows for the calculation of the interface fluxes. The conversion between primary and conservative state variables is not shown here but is rather straightforward. The solver developed here uses primary state variables in its Riemann solvers and converts their results to conservative variables for the flux calculations.

3.3. Geometry

Using the definitions obtained from the Hancock scheme it is now possible to derive mathematical expressions for the boundary conditions and the conditions that should be imposed on the spatial and temporal grid size.

3.3.1 Boundary conditions

It has already been mentioned that two boundary conditions are implemented in the shock tube code.

Soft boundary The first is the *soft* boundary, which allows waves to run out of the tube such that no reflections occur. Using the virtual cells Ω_0 and Ω_{J+1} it is possible to write explicitly for the boundary conditions:

$$\begin{aligned} \boldsymbol{w}_0 &= \boldsymbol{w}_1 \quad \text{and} \quad \delta\boldsymbol{w}_0 = 0, \\ \boldsymbol{w}_{J+1} &= \boldsymbol{w}_J \quad \text{and} \quad \delta\boldsymbol{w}_{J+1} = 0. \end{aligned} \tag{3.18}$$

Hard boundary The second boundary condition is the *hard* boundary. The hard boundary means that the ends of the tube are closed such that waves are reflected. This can be expressed as:

$$\begin{array}{llllll} \rho_0 = \rho_1 & u_0 = -u_1 & p_0 = p_1 & \delta\rho_0 = -\delta\rho_1 & \delta u_0 = \delta u_1 & \delta p_0 = -\delta p_1, \\ \rho_{J+1} = \rho_J & u_{J+1} = -u_J & p_{J+1} = p_J & \delta\rho_{J+1} = -\delta\rho_J & \delta u_{J+1} = \delta u_J & \delta p_{J+1} = -\delta p_J. \end{array} \tag{3.19}$$

In the following calculations always the soft boundary is used. Only in the section treating the double blast wave problem, the hard boundary shall be applied.

3.3.2 CFL number

The stability of a numerical scheme plays an important role. Almost every explicit scheme has some kind of condition on the size of the grid elements that prevents the solution from becoming unstable and growing out-of-bounds. In the case of both a temporal and a spatial grid, the two grid-size parameters Δx and Δt are linked. This means that only one of the parameters can be chosen freely, because the other has to fulfill the condition that relates the two. This condition is often called the *CFL condition*, named after its inventors Courant, Friedrichs and Lewy. The CFL condition for the Hancock scheme used here has the following form:

$$\frac{\max_k \left|\frac{x}{t}\right| \Delta t}{\Delta x} \leq 1. \tag{3.20}$$

Thus the CFL condition depends on the value of the strongest shock or the first characteristic of an expansion fan. This condition requires knowledge of the solution of the Riemann problem before a time step can be chosen. Unfortunately, the Hancock scheme requires the time step in advance, such that it is more convenient to use an approximate form of the CFL condition:

$$\frac{(|u| + a)\, \Delta t}{\Delta x} \leq 1. \tag{3.21}$$

The approach used in the shock tube solver discussed here is that a convenient (sufficiently fine) temporal grid size is estimated based on the CFL-condition, which is kept constant during the calculations.

3.4. Godunov's exact Riemann solver

The Riemann problem not only plays an important role in obtaining the exact solution to the one-dimensional Euler equations, it can also be used in the numerical solution methods described here. Godunov was the first to apply the theory of the Riemann problem to discretized solution methods. The task of Godunov's exact Riemann solver lies in solving for the flow conditions on the cell interfaces, as depicted in equation (3.8). Given the initial conditions in both the $left$ and the $right$ interface regions ($\tilde{\boldsymbol{w}}_L$ and $\tilde{\boldsymbol{w}}_R$) the Riemann solver is able to determine the conditions in the final regions $left^*$ and $right^*$. Because these particular conditions are needed to calculate the fluxes on the cell interfaces, the Riemann approach is ideally suited for this application. The main advantage is that all physical properties are conserved. No approximations whatsoever are made.

Because Godunov's approach uses exactly the same approach as described in the previous chapter, the Riemann solver developed here can be copied completely into the shock tube solver. Insert the distribution of the $left$ and $right$ interface conditions into the Riemann solver and the interface conditions are returned. The only thing that requires some further attention is the fact that the Godunov approach requires the final solution of the Riemann problem on the time axis ($\frac{x}{t} = 0$). Therefore knowledge is required about the region in which the time axis lies. When the final conditions of the Riemann problem are known, also the locations of the different waves can be determined as was shown in the section treating the exact Riemann solver. Using this information it is possible to program a simple algorithm that determines which region contains the time axis.

Given the wave distribution as depicted in figure 2.6, the point is determining which wave is located closest to the time axis. This can be done by creating a vector containing the slopes (actually the wave speeds) of the five waves present in the Riemann problem; a left- and a right running wave being either a shock (in this case both waves are the same and move with the shock speed) or an expansion (in this case the waves are the first and the last characteristics respectively) and the contact discontinuity. The absolute minimum value of this vector is the wave that lies closest to the time axis (having slope equal to zero). Because each wave separates two regions of the six possible regions of the problem, knowing the wave closest to the time axis means knowing which region contains the time axis. If the wave closest to the time axis has a positive slope, thus lies right of the time axis, it automatically follows that the correct region lies left of this wave. Because the numbering of the waves and the regions is from the left to the right (1 to 5 for the waves and 1 to 6 for the regions), this holds for both a shock and an expansion. In the case that the wave closest to the time axis has a negative slope, thus lies to the left of the time axis, two possibilities occur: in case of a shock the region containing the time axis is equal to the region to the right of the shock (because all waves are numbered, even in the case of a shock where the two waves have equal speeds, this means taking the second wave describing the shock instead of the first) and in case of an expansion this is the region between the first and the last characteristics. For the details of this elegant algorithm see the program code in the appendix.

The test results of this exact Riemann solver applied to the shock tube can be found in a following section.

3.5. Roe's approximate Riemann solver

Another method to determine the interface fluxes is Roe's approximate Riemann solver. Roe's solver is one of the many Riemann solvers that exist. A linearised solver means that the governing equations of the Riemann problem have been approximated. Obviously this implies that the solution to the Riemann problem will not be exact anymore, but Roe's approach has shown that despite the approximations good results can be obtained.

3.5.1 The interface flux

Roe's Riemann solver calculates the interface fluxes $\boldsymbol{f}_I$ using the following approximation:

$$\boldsymbol{f}_I = \frac{1}{2}\left(\boldsymbol{f}_L + \boldsymbol{f}_R\right) - \frac{1}{2}\sum_{k=1}^{3}\left|\hat{\lambda}_k\right|^* \Delta v_k \hat{\boldsymbol{r}}_k, \tag{3.22}$$

where $\boldsymbol{f}_L$ and $\boldsymbol{f}_R$ are the fluxes calculated using the $left$ and $right$ initial conditions as calculated in the previous steps of the Hancock scheme. The eigenvalues of the Euler equations λ_k, being the wave speeds $u-a$, u and $u+a$, are used here in their modified absolute form $\left|\hat{\lambda}_k\right|^*$. This modified form uses the so-called *Roe averaged state quantities* given by:

$$\begin{aligned}
\hat{\rho} &= \omega\rho_L, && (3.23)\\
\hat{u} &= \frac{u_L + \omega u_R}{1+\omega}, && (3.24)\\
\hat{H} &= \frac{H_L + \omega H_R}{1+\omega}, && (3.25)\\
\hat{a} &= \sqrt{(\gamma-1)\left(\hat{H} - \frac{1}{2}\hat{u}^2\right)}. && (3.26)
\end{aligned}$$

where the ratio ω is used, which is defined as:

$$\omega = \sqrt{\frac{\rho_R}{\rho_L}}. \tag{3.27}$$

Further the following parameter is introduced:

$$\begin{aligned}
\frac{\delta\lambda_k}{2} &= 0 && \text{for} \quad k=2,\\
\frac{\delta\lambda_k}{2} &= \min\left[\hat{a}, \max\left(0, 2\left(\lambda_{kR} - \lambda_{kL}\right)\right)\right] && \text{for} \quad k=1,3.
\end{aligned} \tag{3.28}$$

Such that the modified absolute eigenvalues can finally be expressed as:

$$\begin{aligned}
\left|\hat{\lambda}_k\right|^* &= \left|\hat{\lambda}_k\right| && \text{for} \quad \left|\hat{\lambda}_k\right| \geq \tfrac{\delta\lambda_k}{2},\\
\left|\hat{\lambda}_k\right|^* &= \frac{\left(\hat{\lambda_k}\right)^2}{\delta\lambda_k} + \frac{\delta\lambda_k}{4} && \text{for} \quad \left|\hat{\lambda}_k\right| < \tfrac{\delta\lambda_k}{2}.
\end{aligned} \tag{3.29}$$

The jump relations Δv_k in (3.22) are obtained from the differential relation $d\boldsymbol{v} = \boldsymbol{R}^{-1}d\boldsymbol{q}$ where $\boldsymbol{R}$ is the matrix containing the right eigenvectors of the Euler equations. This matrix is equal to:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 1 & 1 \\ u-a & u & u+a \\ H-ua & \frac{1}{2}u^2 & H+ua \end{pmatrix}, \tag{3.30}$$

such that the jump relations become:

$$\Delta\boldsymbol{v} = \begin{pmatrix} \frac{\Delta p - \hat{\rho}\hat{a}\Delta u}{2\hat{a}^2} \\ -\frac{\Delta p - \hat{a}^2\Delta\rho}{\hat{a}^2} \\ \frac{\Delta p + \hat{\rho}\hat{a}\Delta u}{2\hat{a}^2} \end{pmatrix}. \tag{3.31}$$

Here $\Delta\rho$, Δu and Δp are respectively the density, velocity and pressure jumps over the cell-interface:

$$\begin{aligned}\Delta\rho &= \rho_{right} - \rho_{left},\\ \Delta u &= u_{right} - u_{left},\\ \Delta p &= p_{right} - p_{left}.\end{aligned} \tag{3.32}$$

The modified eigenvectors $\hat{\boldsymbol{r}}_k$ follow by replacing the *normal* state variables in the eigenvectors (3.30) with Roe's averaged state quantities.

With the above, all necessary information for calculating the interface fluxes is known. The procedure is as follows; given the $left$ and $right$ initial interface conditions, it is possible to calculate the fluxes left and right of the interface $\boldsymbol{f}_L$ and $\boldsymbol{f}_R$. The initial conditions can further be used to calculate Roe's averaged state quantities $\hat{\rho}$, $\hat{u}$, $\hat{H}$ and $\hat{a}$. This allows for the calculation of the modified eigenvectors $\hat{\boldsymbol{r}}_k$ and the modified absolute eigenvalues $\left|\hat{\lambda}_k\right|^*$. Also the jump relations $\Delta\boldsymbol{v}$ can be obtained from these averaged quantities together with the jumps in density, velocity and pressure, which allows for the calculation of the interface fluxes using (3.22).

3.5.2 One term equation

To reduce the work it is possible to transform the above to a more simple expression. The sum in (3.22) can be written as one term. To do so two situations are distinguished:

$$\begin{aligned}&\hat{u} \geq 0 \quad \text{then} \quad \boldsymbol{f}_I = \boldsymbol{f}_L + \hat{\lambda}_1^{-*}\Delta v_1 \hat{\boldsymbol{r}}_1,\\ &\hat{u} < 0 \quad \text{then} \quad \boldsymbol{f}_I = \boldsymbol{f}_R - \hat{\lambda}_3^{+*}\Delta v_3 \hat{\boldsymbol{r}}_3.\end{aligned} \tag{3.33}$$

where the *new* modified eigenvalues are defined as follows:

$$\hat{\lambda}_k^{-*} = \begin{cases} \hat{\lambda}_k & \text{if} \quad \hat{\lambda}_k \leq \frac{-\delta\lambda_k}{2},\\ -\frac{\left(\hat{\lambda}_k - \frac{\delta\lambda_k}{2}\right)^2}{2\delta\lambda_k} & \text{if} \quad \frac{-\delta\lambda_k}{2} < \hat{\lambda}_k < \frac{\delta\lambda_k}{2},\\ 0 & \text{if} \quad \hat{\lambda}_k \geq \frac{\delta\lambda_k}{2}, \end{cases} \tag{3.34}$$

$$\hat{\lambda}_k^{+*} = \begin{cases} \hat{\lambda}_k & \text{if} \quad \hat{\lambda}_k \leq \frac{-\delta\lambda_k}{2},\\ \frac{\left(\hat{\lambda}_k + \frac{\delta\lambda_k}{2}\right)^2}{2\delta\lambda_k} & \text{if} \quad \frac{-\delta\lambda_k}{2} < \hat{\lambda}_k < \frac{\delta\lambda_k}{2},\\ 0 & \text{if} \quad \hat{\lambda}_k \geq \frac{\delta\lambda_k}{2}. \end{cases} \tag{3.35}$$

The procedure is for the remaining part exactly the same as the general method described above.

3.5.3 Entropy fix

The method described above contains the so-called *entropy fix*. This means that the unphysical situation of compression fans (reversed expansion fans) are eliminated from the solution of the approximate Riemann solver. Especially in the case of transonic expansion fan and a first-order upwind scheme (instead of the Hancock scheme) this problem occurs. Using the Hancock scheme allows most of the times for a much easier version of Roe's approximate solver, which does not have this entropy fix. Defining the following two *switches*:

$$\begin{aligned}\sigma_1 &= \operatorname{sign}(\hat{u}),\\ \sigma_2 &= \operatorname{sign}\left(\hat{u}^2 - \hat{a}^2\right).\end{aligned} \tag{3.36}$$

allows for the following expression for the interface flux:

$$\boldsymbol{f}_I = \frac{1+\sigma_1}{2}\boldsymbol{f}_L + \frac{1-\sigma_1}{2}\boldsymbol{f}_R - \frac{1-\sigma_2}{2}\left(\hat{u} - \sigma_1\hat{a}\right)\frac{\left(\hat{\rho}\hat{a}\Delta u - \sigma_1\Delta p\right)}{2\hat{a}^2}\begin{pmatrix}1\\ \hat{u} - \sigma_1\hat{a}\\ \hat{H} - \sigma_1\hat{u}\hat{a}\end{pmatrix}. \tag{3.37}$$

This method will only be used to indicate the effect of the entropy fix. Further calculations with Roe's approximate Riemann solver will be done with the one-term version with entropy fix.

3.6. Test cases

The shock tube solver as developed here (see appendix I for the program code) shall be tested for its performance using the test described in the previous chapter. Because the exact results are known for these tests, conclusions can be drawn regarding the accuracy of the numerical methods. The results of the numerical calculations with the shock tube solver using both Godunov's solver and Roe's solver are plotted in the same graphs as the exact solution. The calculations are performed using Koren's limiter and the second-order Hancock scheme. Unless otherwise indicated the number of cells is taken equal to 100, such that $\Delta x = 1/100$. For the temporal discretization a step size is taken based on the CFL-number specific for each test.

3.6.1 Test 1

The initial data for this test case can be found in table 2.1. The maximum wave speed $|u| + a$ is estimated to be approximately 2.5 such that a comfortable temporal step size of $1/500$ suffices. The results of both the Godunov and the Roe method are given in figure 3.2. From these figures it becomes clear that both the exact and approximate solvers give very good results. This is for a large part the contribution of the limiter, which prevents the solution from having *wiggles* (in a following section some tests shall be performed with the linear *algebraic average* such that the difference becomes clear). The expansion running to the left is captured almost exactly by the solvers. The shock running to the right is somewhat *smeared out* over approximately four cells, but remains well-visible. The contact discontinuity however is smeared to a much larger extent. This phenomenon occurs with all numerical solvers. The difference between the shock and the discontinuity has to do with the fact that a shock is formed by characteristics that meet at the shock, but that in the case of a contact discontinuity the characteristics are parallel, such that the information is not *pushed* towards the discontinuity.

The exact and the approximate solver produce results that are almost exactly the same. A small difference occurs near the last ray of the expansion in the velocity profile. Here the exact solver has a small overshoot, while Roe's solver *cuts-off* the sharp corner, but this difference is of no significance. Although Roe's solver is a linearised version of the exact solver, its results are almost similar.

3.6.2 Test 2

The initial data for this test case are given in table 2.3. The maximum wave speed is approximately 3, such that a step size of $1/500$ will suffice. The results of both the Godunov and the Roe method are given in figure 3.3. Unfortunately the program was not able to finish the calculations using the approximate Riemann solver of Roe. During the calculations cavitation occurred. Probably this can be avoided by fine-tuning the program, but this falls out of the scope of this exercise. The exact solver gives fairly good results.

Interesting are the small overshoots near the center of the tube, where the last rays of both the left- and right-running expansions divide the inner expansion and final regions. This behaviour can be explained by the fact that the density and the pressure are almost equal to zero here. A small numerical error can cause cavitiation or vacuum. The program has a check that determines if one of these cases happens. If so, the values of both the density and pressure are adjusted. This can cause small oscillations and errors.

3.6.3 Test 3

The initial data for this test case are given in table 2.5. A fairly small step size equal to $1/5000$ is required due to the relatively large wave speeds. The results of both the Godunov and the Roe method are given in figure 3.4. Again both the results of the exact and the approximate solver compare well to the exact results. The very sharp peak in the left part of the density distribution is due to the large jumps over the contact discontinuity and the shock. It is not strange that the numerical solvers have difficulties capturing these.

Again the difference between both solvers is minimal. As was seen in the results of test 1, the exact solver has somewhat more overshoot near the last ray of the expansion than the approximate solver.

3.6.4 Test 4

The initial data for this test case are given in table 2.7. Again a small time step is required. This time a step size of $1/2500$ is taken. The results of both the Godunov and the Roe method are given in figure 3.5. This test is qualitatively the mirror version of test 3. The behaviour should therefore be comparable. It is obvious that this is the case. Again both solvers have difficulties capturing the large density peak near the shock and the contact discontinuity. The expansion is captured very accurately.

3.6.5 Test 5

The initial data for this test case are given in table 2.9. The maximum wave speed is extimated to be equal to 30 such that a step size equal to $1/4000$ suffices. The results of both the Godunov and the Roe method are given in figure 3.6. This Riemann problem consists of two shocks, both running to the right. While shocks and contact discontinuities are difficult to capture using numerical solvers, the results obtained for this test are less accurate than was the case for the previous tests, although the results are still quite good. The two shocks cause oscillations of the pressure in the final region. And due to the contact discontinuity the density distribution is smeared out. Especially near the right end of the tube the errors become large.

3.6.6 Conclusions

Both Godunov's exact Riemann approach and the approximate solver of Roe show accurate results. Expansions are captured almost exactly. Shocks and expansions are smeared out to some extent, but their positions remain exact. In case of large density or pressure jumps, the solvers have difficulties reaching the appropriate values. The Superbee limiter is responsible for the smooth behaviour of the solutions, although this sometimes results in cut-off corners and less sharp peaks.

No large differences between the two solvers occur. Only some small details are visible near expansion fans. The exact solver has a tendency to overshoot, while Roe's solver smooths the solution somewhat more.

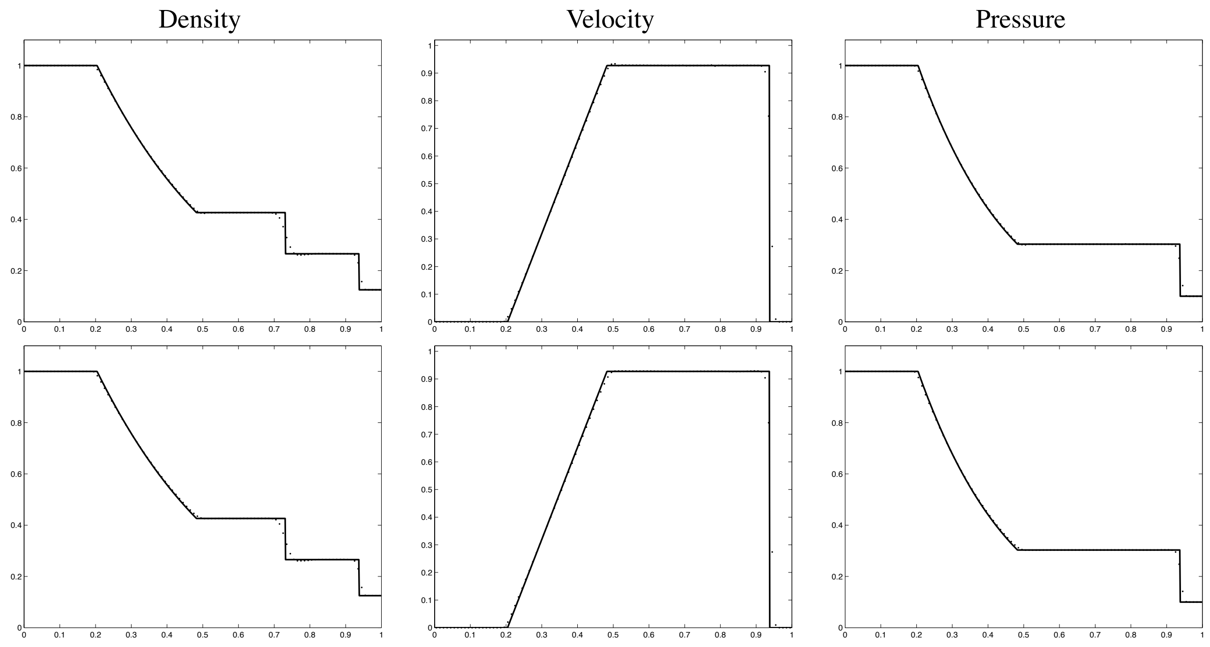

Figure 3.2: Results of test 1. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution. *Top:* Godunov's exact Riemann solver. *Bottom:* Roe's approximate Riemann solver.

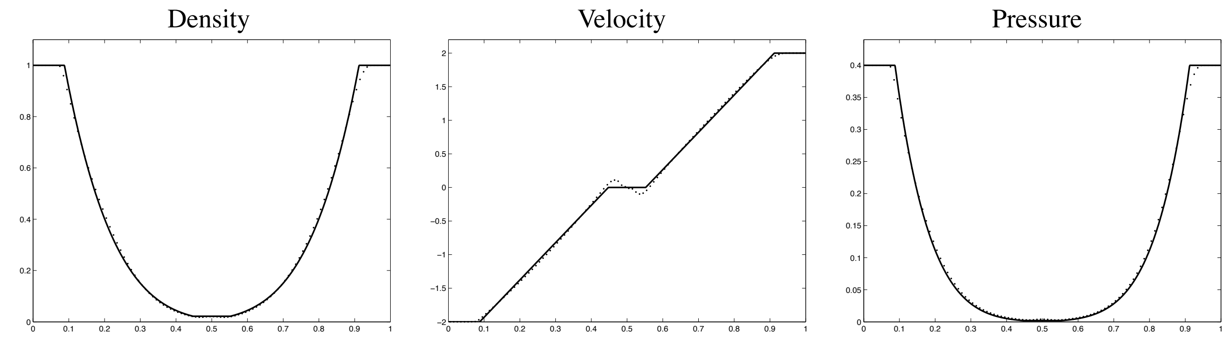

Figure 3.3: Results of test 2. Primary variables against the x location in the tube at $t = 0.15$. $-$ line is exact solution, $\cdots$ line is numerical solution. Godunov's exact Riemann solver. *Not shown:* Roe's approximate Riemann solver; cavitiation occurred.

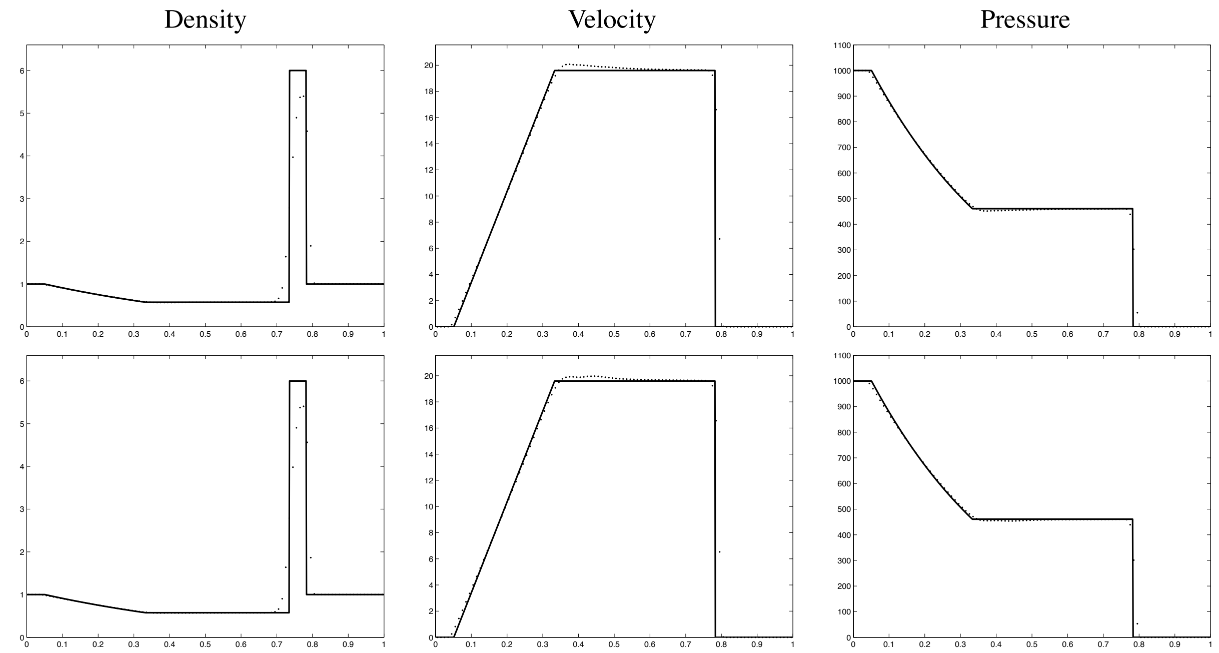

Figure 3.4: Results of test 3. Primary variables against the x location in the tube at $t = 0.012$. $-$ line is exact solution, $\cdots$ line is numerical solution. *Top:* Godunov's exact Riemann solver. *Bottom:* Roe's approximate Riemann solver.

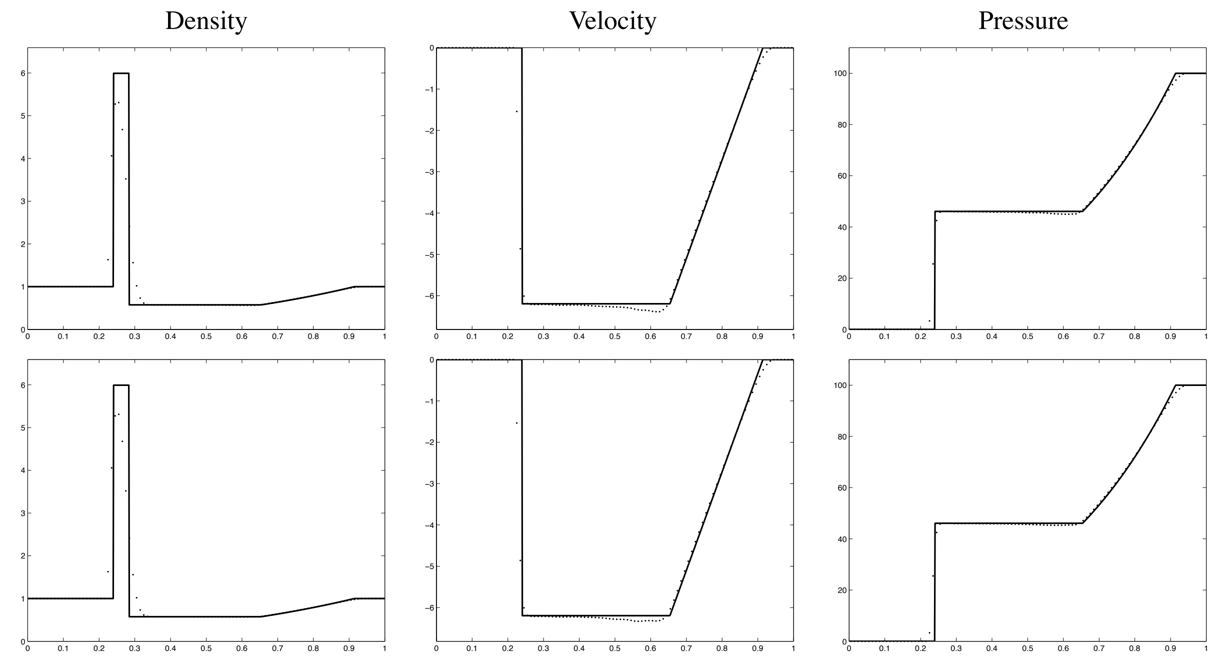

Figure 3.5: Results of test 4. Primary variables against the x location in the tube at $t = 0.035$. $-$ line is exact solution, $\cdots$ line is numerical solution. *Top:* Godunov's exact Riemann solver. *Bottom* Roe's approximate Riemann solver.

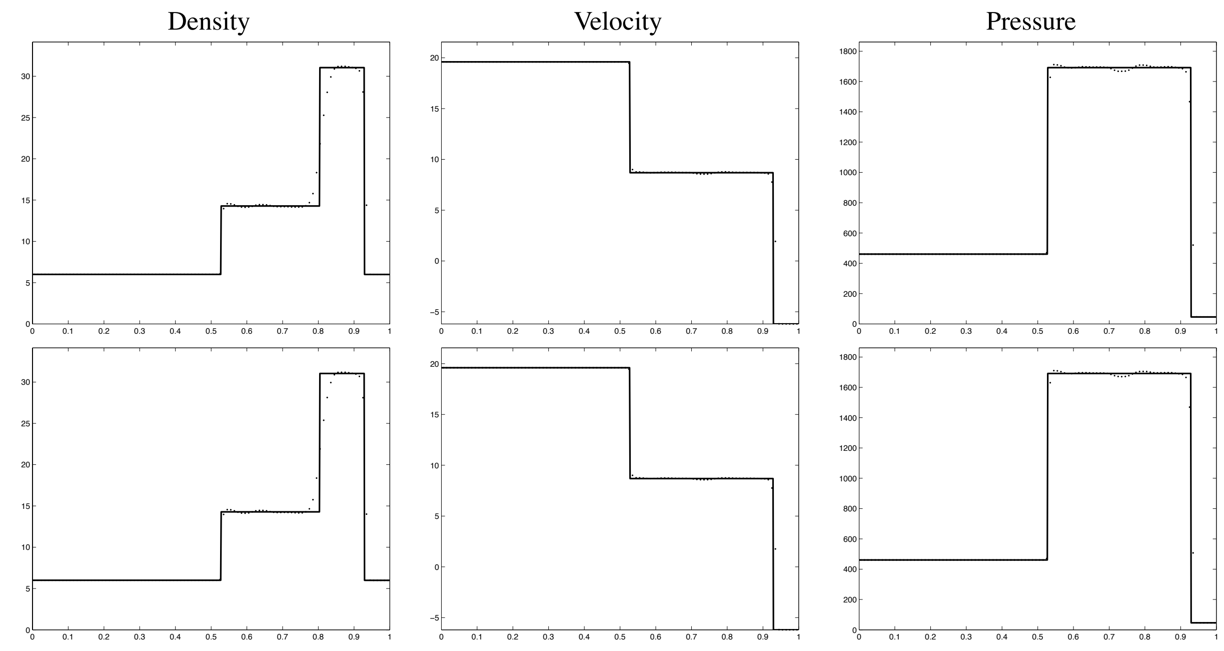

Figure 3.6: Results of test 5. Primary variables against the x location in the tube at $t = 0.035$. $-$ line is exact solution, $\cdots$ line is numerical solution. *Top:* Godunov's exact Riemann solver. *Bottom:* Roe's approximate Riemann solver.

3.7. Convergence tests

Decreasing the size of a grid cell Δx increases the accuracy of the solution and thus decreases the error. The relation between the error and the spatial grid size is indicated by the *order* of the scheme. When a numerical scheme is of the first-order this means that the error and the grid size relate as follows: $\epsilon \sim \Delta x^1$. Halving the size of a cell, halves the error. The same holds for a second-order scheme: $\epsilon \sim \Delta x^2$, only now halving the grid size means reducing the error with a factor four. If this behaviour is the case, the solution is called *convergent*. When the error at different grid sizes is known it is possible to determine the order of the scheme using the following expression:

$$\epsilon \sim \Delta x^p \qquad \Rightarrow \qquad p = \frac{\log(\epsilon)}{\log(\Delta x)} + c, \tag{3.38}$$

or expressed as a function of the number of cells J:

$$\epsilon \sim \left(\frac{1}{J}\right)^p \qquad \Rightarrow \qquad p = -\frac{\log(\epsilon)}{\log(J)} + c. \tag{3.39}$$

The Hancock scheme used in the calculations above is of the second order. Setting $\delta \boldsymbol{w}$ in this scheme equal to zero gives the first-order upwind scheme. To test the solver developed here for convergence several calculations shall be performed using the initial data from test 1. Also the first-order scheme is used because for this scheme the required convergence rates are known. The number of grid cells J is taken equal to 50, 100, 200, 400 and 800 and the CFL-number is held constant such that $\frac{\Delta t}{\Delta x} = 0.2$. The general definition of a numerical error is:

$$\epsilon_{L_n} = \left(\frac{\sum_1^J |\epsilon|^n}{J}\right)^{\frac{1}{n}}. \tag{3.40}$$

Depending on the problem one is treating an error order n is chosen. To avoid the errors from being dominated by the shocks, which in the case of $n \geq 1$ always leads to an order $p \leq 1$, for the second-order scheme the $L_{\frac{1}{4}}$*-error* shall be used and for the first-order scheme the $L_{\frac{1}{2}}$*-error* (see appendix I for a discussion of these fractional error norms). The error ϵ is taken equal to the difference between the exact solution and the numerical solution. Given the error ϵ_{L_n} for different grid sizes J it is possible to determine the order p. A *least-squares method* is used for this. The time step size is the same as for the five test cases discussed earlier.

3.7.1 First-order upwind

Performing the calculations for the first-order upwind scheme results in the data given in table 3.1. The errors have been determined for the five different numbers of cells. Using a least-squares method the order of the scheme could then be calculated.

For the first-order upwind scheme, the obtained convergence order lies around $p = 0.9$, which corresponds to the expectations. It is a fact however, that although the scheme is called first-order, this order is often not reached when treating non-smooth problems (such as problems with shocks). The reason why this is the case, has to do with the chosen error norm. The $L_{\frac{1}{2}}$*-error* weighs the errors due to the contact discontinuity and the shocks less heavily than the other errors. This thus leads to a somewhat distorted order. Though, taking the L_1*-error* norm would lead to results dominated by the discontinuities, which isn't a required situation either. So the results obtained here indicate that the program works as required.

3.7.2 Second-order Hancock

The same convergence test has been performed for the second-order Hancock scheme with the different averaging schemes. The results are shown in tables 3.2, 3.3, 3.4 and 3.5.

The convergence is of the order $p = 1.6$ to $p = 1.8$ for all limiters, while the algebraic average lies somewhat higher. These values are far from equal to the value two, which is expected from a second-order scheme. But the same argument can be used here, namely that values much higher than these are unique due to the large influence of the errors induced by the contact discontinuity and the shocks. Even with the use of the more convenient lower-order error norms the order stays below the required value 2.

3.7.3 Conclusions

Given the above-mentioned results of the convergence tests it can safely be said that the results of the program compare quite well with the expectations. The order of the schemes lies in the expected regions. A sideremark that can be made here is that the errors for the second-order scheme are a factor 5 till 10 lower than for the first-order scheme. This is not so strange since the first-order scheme is not able to capture the sharp discontinuities, but smears everything out instead, even the expansions.

J	ϵ_ρ	ϵ_u	ϵ_p
50	0.0255746	0.037459	0.018790
100	0.015366	0.018623	0.010117
200	0.008325	0.008701	0.004887
400	0.004470	0.004205	0.002515
800	0.002458	0.002147	0.001349
p	-0.854007	-1.017936	-0.960808

Table 3.1: Results for convergence test for first-order upwind scheme.

J	ϵ_ρ	ϵ_u	ϵ_p
50	0.008036	0.009102	0.005060
100	0.002615	0.003177	0.001630
200	0.000682	0.000845	0.000415
400	0.000157	0.000188	0.000089
800	0.000057	0.000069	0.000028
p	-1.831682	-1.814645	-1.922161

Table 3.2: Results for convergence test for second-order Hancock scheme, using the Algebraic Average.

J	ϵ_ρ	ϵ_u	ϵ_p
50	0.002689	0.003544	0.001575
100	0.000929	0.001082	0.000505
200	0.000261	0.000324	0.000151
400	0.000059	0.000072	0.000028
800	0.000029	0.000036	0.000014
p	-1.695444	-1.717639	-1.774338

Table 3.3: Results for convergence test for second-order Hancock scheme, using the Double Minmod limiter.

3.8. The averaging schemes compared

The shock tube solver contains several averaging schemes, being: the algebraic average and the Double Minmod, Superbee and Koren limiters. To compare these with each other they shall be applied to the Sod problem (test 1). Godunov's exact Riemann solver is used for the calculations. The time step is again taken equal to $\Delta t = 1/1000$ and the cell size is taken equal to $\Delta x = 1/100$ ($J = 100$). For each test the density, velocity and pressure are plotted against the x location in the tube at time level $t = 0.25$. Further the data obtained from the convergence tests are used.

J	ϵ_ρ	ϵ_u	ϵ_p
50	0.003701	0.004015	0.001791
100	0.001285	0.001310	0.000672
200	0.000417	0.000585	0.000302
400	0.000140	0.000211	0.000109
800	0.000026	0.000032	0.000015
p	-1.751698	-1.654497	-1.648838

Table 3.4: Results for convergence test for second-order Hancock scheme, using the Superbee limiter.

J	ϵ_ρ	ϵ_u	ϵ_p
50	0.002760	0.003299	0.001507
100	0.000918	0.000919	0.000443
200	0.000240	0.000261	0.000124
400	0.000053	0.000060	0.000024
800	0.000021	0.000029	0.000009
p	-1.824881	-1.757232	-1.881861

Table 3.5: Results for convergence test for second-order Hancock scheme, using Koren's limiter.

3.8.1 Algebraic average

Running the calculations using the algebraic average results in the plots in figure 3.7 and the earlier found errors in table 3.2. The main difference between the algebraic average and the other three limiters is that the former is linear. This is the reason that it allows for maximum values to occur. This is clearly visible in the results. Especially near sharp gradients the solver has difficulties following the exact solution.

3.8.2 Double Minmod

The results of the calculations with the Double Minmod limiter are plotted in figure 3.8. Further the errors for the Sod problem are given in table 3.3. This limiter does cut off peaks, which results in a more smooth behaviour of the state variables. Almost no peaks occur near the expansion fan, and only small ones near the discontinuities. A result of this is that the shock and the contact discontinuity are smeared out to some extent.

3.8.3 Superbee

The results of the Superbee limiter are given in figure 3.9 and table 3.4. The results of the Superbee limiter show somewhat more oscillations in the state variable distributions than was the case for the Double Minmod limiter. This can also be seen in the errors, which are higher for this limiter than for the other two limiters in the region of small numbers of grid cells. Though due to the higher order, for higher numbers of grid cells this limiter starts to perform better than the other two. The shock and the contact discontinuity are smeared out, but the centres of these are in the right locations.

3.8.4 Koren

The Sod problem with the Superbee limiter has already been treated in a previous section. The results can be found in figure 3.2 and table 3.5. The errors of this limiter are of the same order as those of the Double Minmod limiter, though the distribution seems somewhat more smooth.

3.8.5 Conclusions

Of the three limiters discussed, Koren's limiter shows the best results, although the Double Minmod performs very well too. The Superbee limiter performs best for a larger amount of grid cells. But because the tendency is to use as less cells as possible, this is not a desirable advantage.

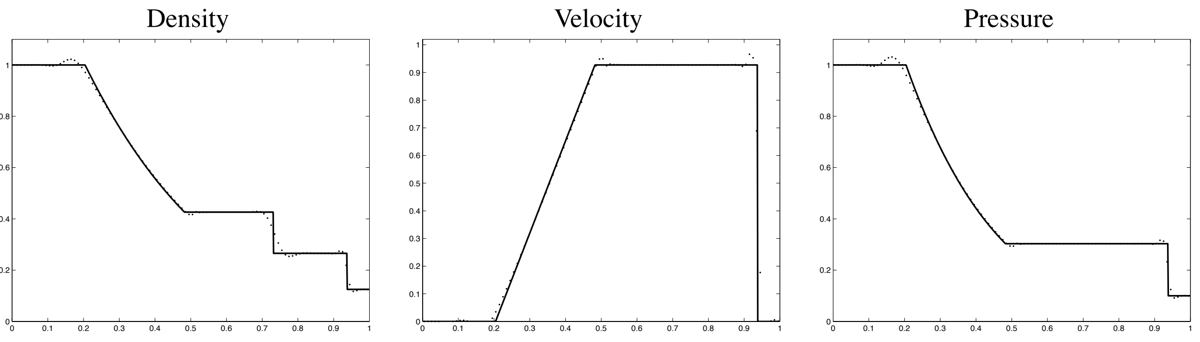

Figure 3.7: Results of Sod's problem with the Algebraic Average. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution.

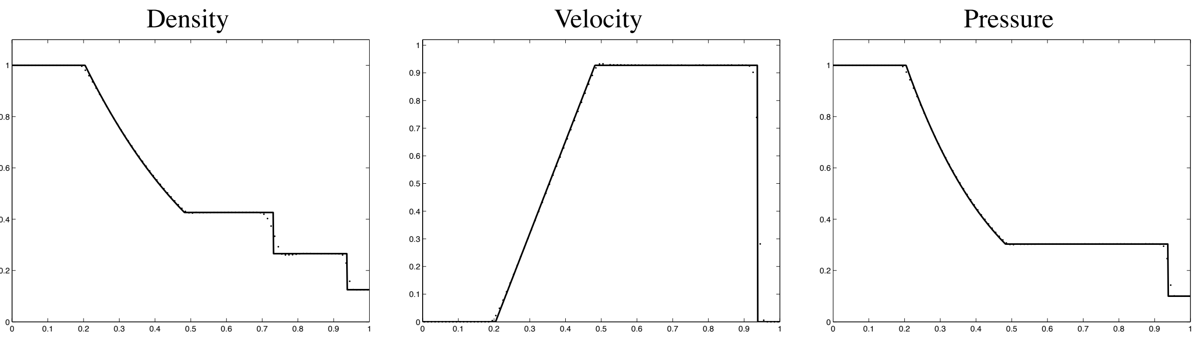

Figure 3.8: Results of Sod's problem with the Double Minmod limiter. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution.

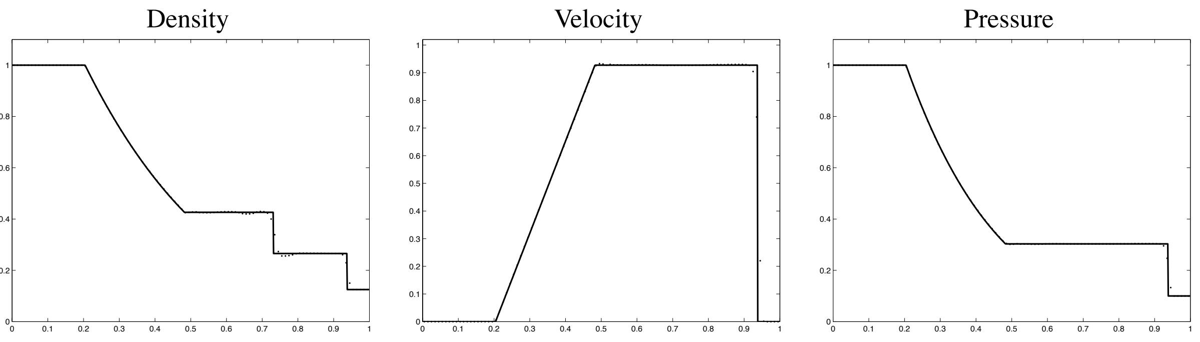

Figure 3.9: Results of Sod's problem with the Superbee limiter. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution.

3.9. THE ENTROPY FIX

In a previous chapter the entropy fix in Roe's average Riemann solver has been explained. It was mentioned that in most cases this fix is not needed. Only in the special case of a transonic expansion fan unphysical compression fans occur. Fortunately most second-order schemes have no troubles with this phenomenon. To show the influence of the entropy fix a test is performed with and without this fix using the first-order scheme, such that the difference becomes clear. The test case used is almost equal to the frequently used Sod problem, only the velocity is now taken equal to $u_L = u_R = 0.5$ such that the expansion fan indeed becomes transonic.

Performing the calculations without the entropy fix leads to the results depicted in figure 3.10. It is obvious that something goes wrong here in the expansion fan. Here a discontinuous jump occurs in the density, velocity and pressure distributions. As mentioned this is due to the unphysical compression fan. Fortunately this can be solved with the entropy fix. This fix restores the physics in the solver, such that the discontinuities dissappear. Performing the calculations with the entropy fix leads to the plots in figure 3.11. Immediately one sees that the obtained solution is the correct one.

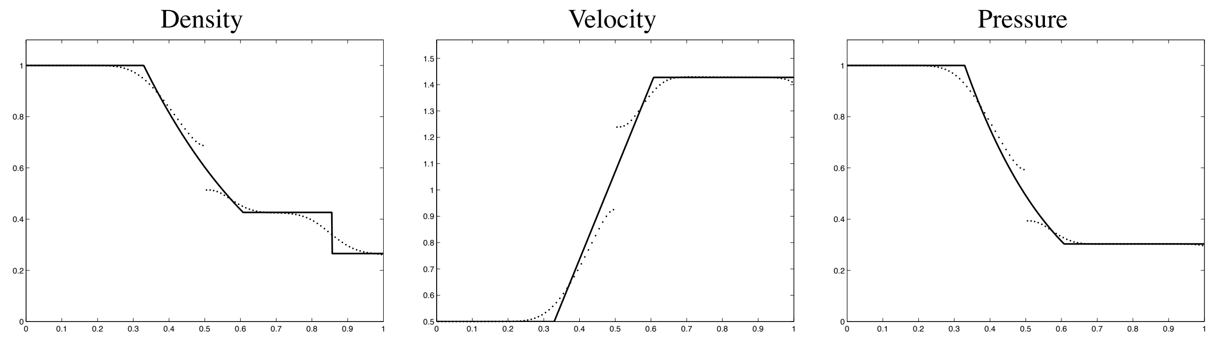

Figure 3.10: Results of the special Sod problem without the entropy fix in Roe's approximate Riemann solver. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution.

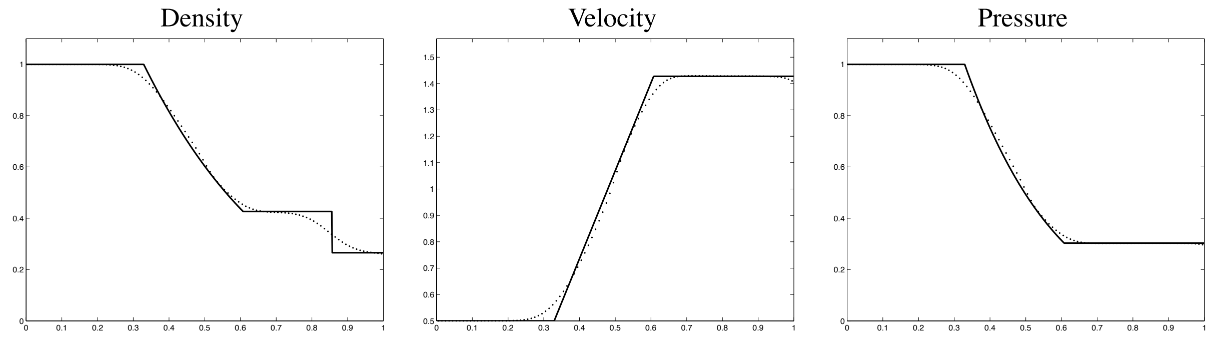

Figure 3.11: Results of the special Sod problem with the entropy fix in Roe's approximate Riemann solver. Primary variables against the x location in the tube at $t = 0.25$. $-$ line is exact solution, $\cdots$ line is numerical solution.

3.10. Double blast wave in a closed tube

For more elaborate testing of the Riemann solver another test shall be performed here. This is the *double blast wave in a closed tube* as described in [8]. This problem consists of three regions in the tube with the same initial densities ($\rho = 1.0$) and velocities ($u = 0.0$), but with different pressures. The left region, running from $x = 0$ to $x = 0.1$ has a pressure equal to $p = 1000.0$. The inner region, $x = 0.1$ to $x = 0.9$, has a pressure equal to $p = 0.01$ and the right region, $x = 0.9$ to $x = 1$, has a pressure equal to $p = 100.0$. These initial conditions will result in a very complex distribution of shocks and expansions. Due to the pressure jump on the right, a shock will travel towards the left and an expansion to the right. The expansion will be reflected due to the closed end of the tube. At the left side of the tube, the opposite occurs. After a while the expansions will meet near the center of the tube.

A fine grid is used for the calculations, the time step is chosen equal to $\Delta t = 1/2000$ and the cell size equal to $\Delta x = 1/1000$.

The whole process is indicated in figure 3.12. This is a contourplot of the (t, x) domain. Here one can clearly see the expansions that are reflected by the tube-ends, the shocks that are curved when they meet an expansion and the contact discontinuities which are somewhat smeared out. Besides the contourplot, at several moments in time the density, velocity and pressure distributions are given. The results compare very well with those given in [8].

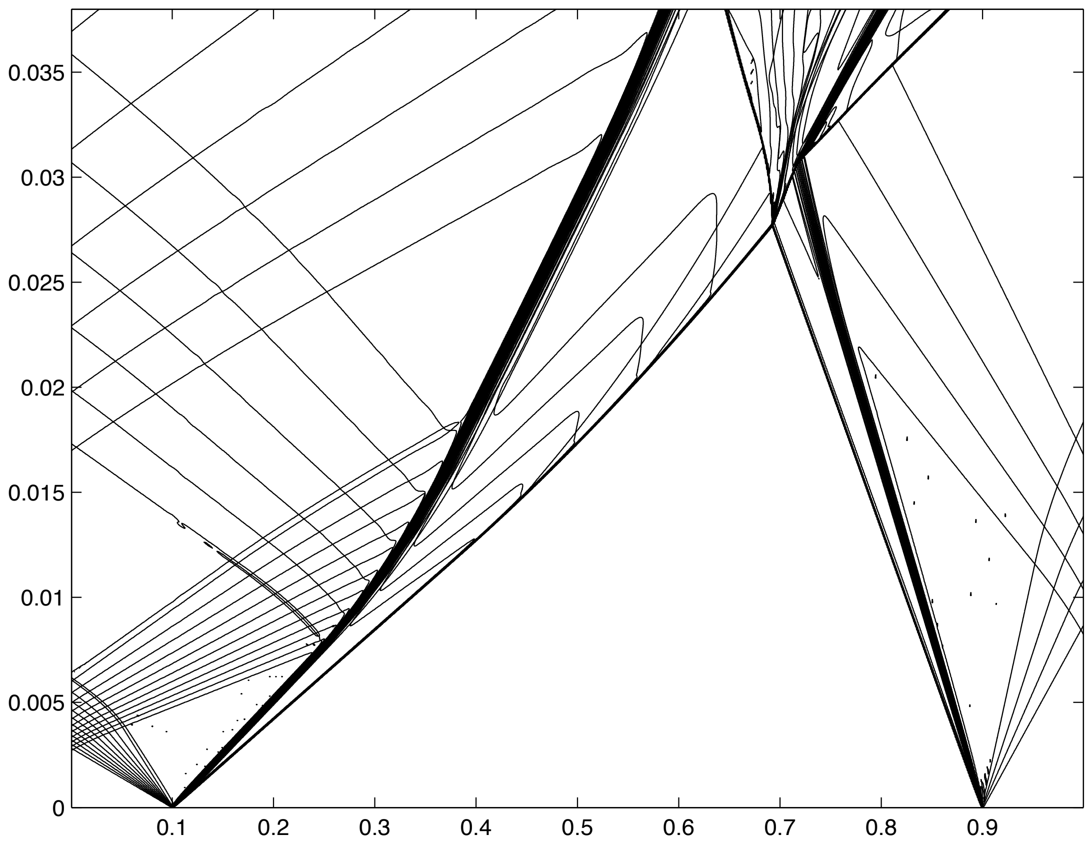

Figure 3.12: Results of the double blast wave problem using Godunov's exact Riemann solver. Contour plot of the Riemann invariants $J^{\pm}$. Clearly visible are the shocks, expansions and contact discontinuities.

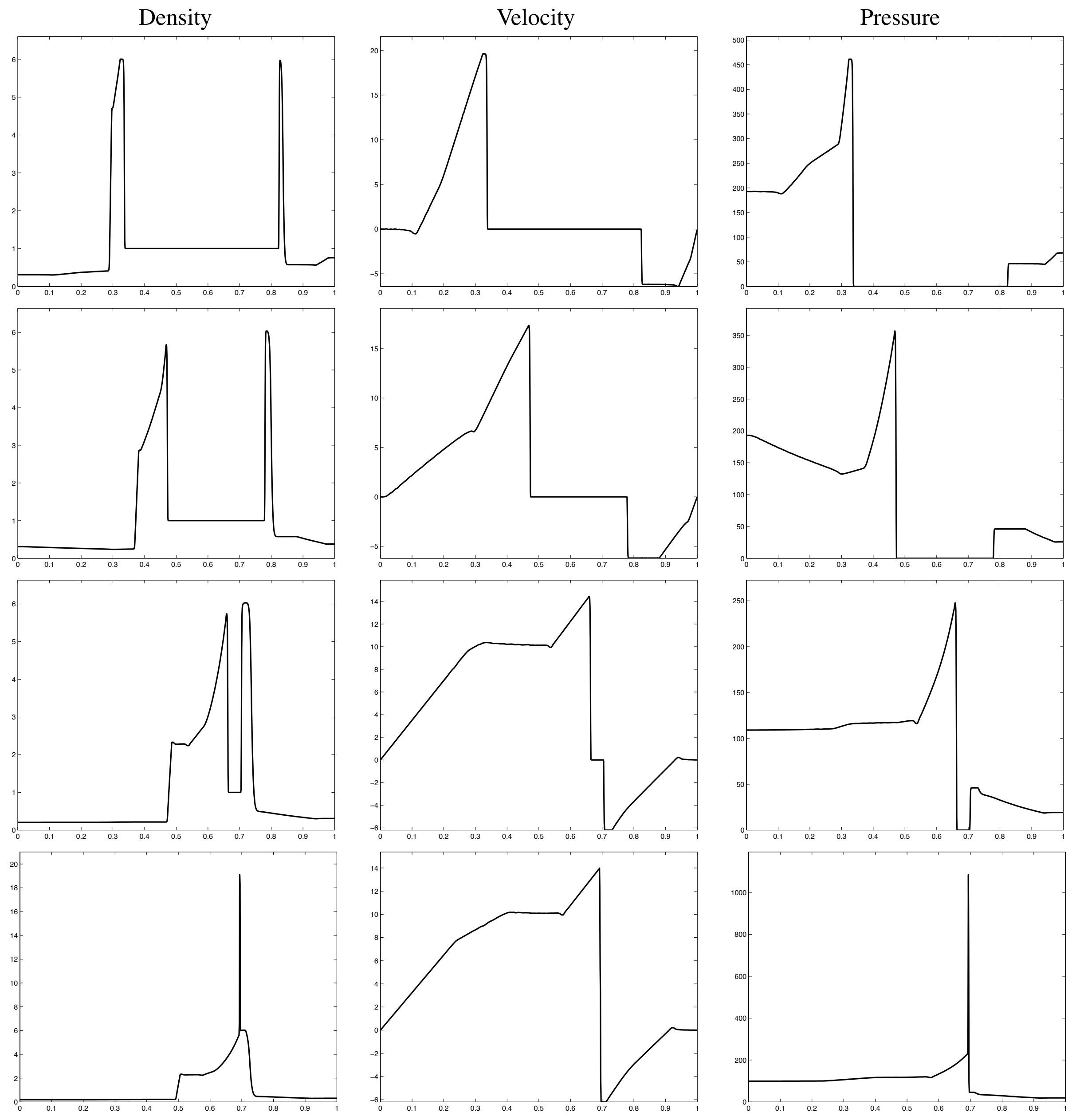

Figure 3.13: Results of the double blast wave problem using Godunov's exact Riemann solver. Primary variables against the x location in the tube at different time levels $t = 0.01$, $t = 0.016$, $t = 0.026$, $t = 0.028$. Continued at next page.

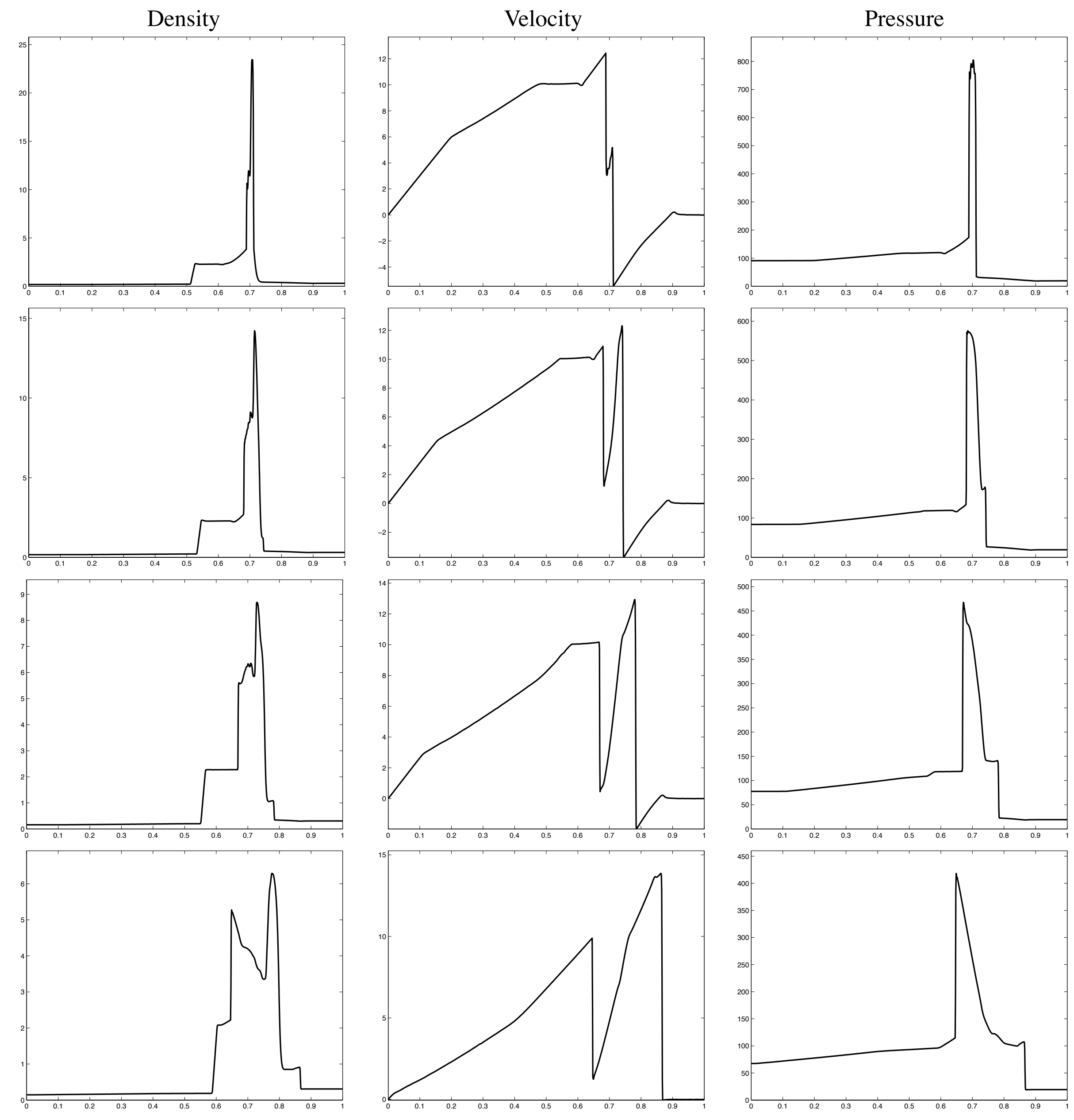

Figure 3.14: Results of the double blast wave problem using Godunov's exact Riemann solver. Primary variables against the x location in the tube at different time levels $t = 0.030$, $t = 0.032$, $t = 0.034$ and $t = 0.038$.

3.11. Interaction of simple waves

As a final test for the solver developed here the interaction of simple waves shall be treated here. When two simple waves interact a *non-simple* region is formed. This region is relatively hard to solve exactly such that the application of the numerical solver is useful here. The test problem treated consists of two expansion fans of different strength both moving towards the center of the tube where they meet. In the first test the ratio of specific heats γ is equal to 1.4, while in the second test, the ratio is taken equal to 3. This second situation leads to fully uncoupled flow equations, such that the $left-$ and $right-$running characteristics completely ignore each other, as shall be proven later in this section. To obtain these interacting simple waves three regions, (L), (M) and (R), with different initial state variable distributions are defined. Because the governing equations used here depend on γ, two different initial distributions shall be used.

The grid-sizes are chosen based on the CFL number, such that $\Delta t = 1/3750$ and $\Delta x = 1/500$. The limiter of Koren is used in the calculations. A contour-plot of the (t, x) domain is used to visualize the interaction of the waves.

3.11.1 $\gamma = 1.4$

The initial conditions for this test are given in table 3.6. Choosing the distribution of state variables in this specific way leads to only two expansion fans. No contact discontinuity or shocks occur.

$Region$	L	M	R
ρ	0.462664366	1.0	0.69036153
u	-5.0	0.0	2.5
p	11.8970837	35.0	20.83412477

Table 3.6: Initial state variable distribution for two interacting simple waves.

The test results of the two interacting waves are displayed in figure 3.15. One clearly sees the two expansion fans emanating from both the left and right ends of the tube. Because the characteristic speeds of the first ray of each fan are equal the two fans meet each other exactly in the middle. When two simple waves interact a non-simple region occurs. In this region there is no class (left- or right-running) of characteristics that is straight. The two expansion fans accelerate each other while *bending* each other's characteristics. When the characteristics leave the non-simple region and enter a simple wave region again, the characteristics become straight again.

3.11.2 $\gamma = 3$

In the special case that the ratio of specific heats is taken equal to 3.0 the set of equations describing the flow becomes uncoupled. This can be shown using the expressions for the Riemann invariants and the characteristic speeds. Taking $\gamma = 3.0$ in (2.12) leads to the following expression:

$$J^{\pm} = u \pm \frac{2}{3-1} a = u \pm a, \tag{3.41}$$

which is exactly equal to the expression for the characteristic speeds along which the Riemann invariants are constant:

$$\Gamma^{\pm} : \frac{dx}{dt} = u \pm a. \tag{3.42}$$

Thus taking γ equal to 3.0 means that the characteristics are always straight lines. Taking the initial conditions as given in 3.7 results in figure 3.16. Immediately one notices the characteristics to be straight in the non-simple region. The expansion fans are not affected by the presence of the other. Besides the fans some other contours are drawn. These are most likely the result of some small numerical errors and the choice of the program that plots the contours.

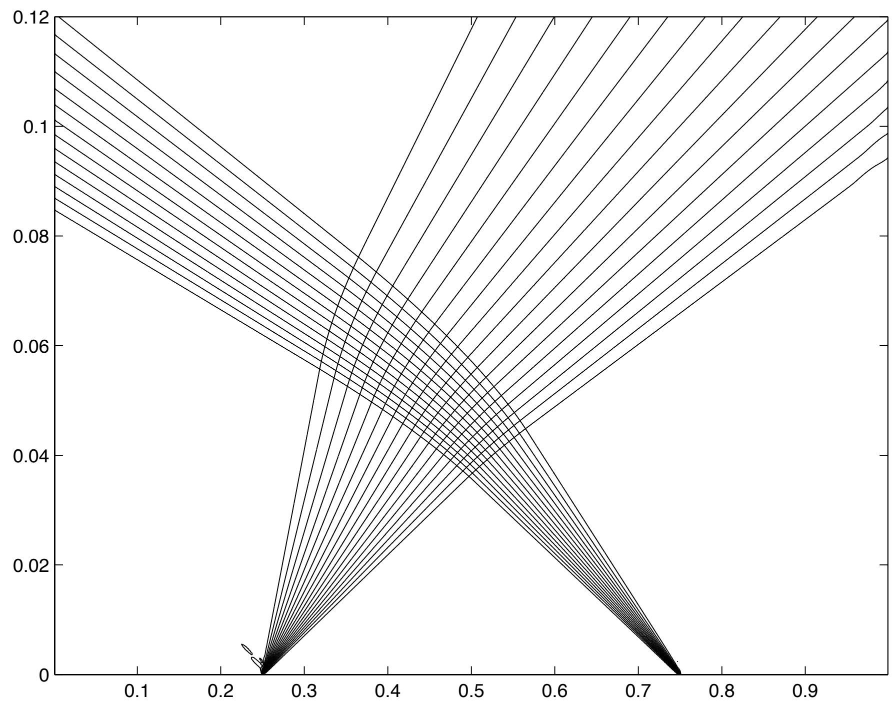

Figure 3.15: Two interacting simple waves with $\gamma = 1.4$. Contour plot of the Riemann invariants $J^{\pm}$.

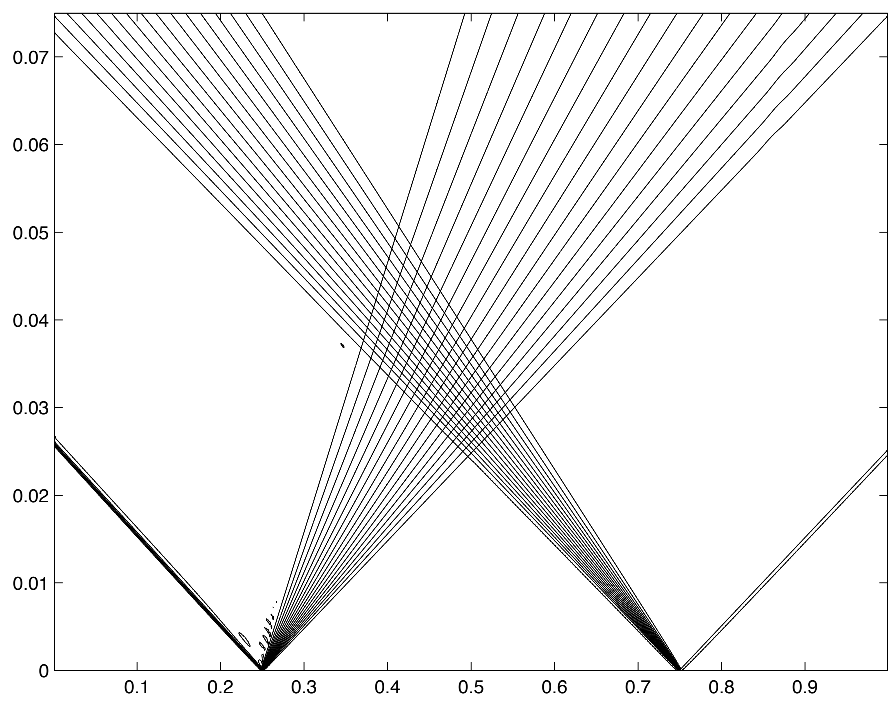

Figure 3.16: Two interacting simple waves with $\gamma = 3.0$. Contour plot of the Riemann invariants $J^{\pm}$.

$Region$	L	M	R
ρ	0.571428571	1.0	0.785714285
u	-3.0	0.0	1.5
p	6.530612245	35.0	16.97704082

Table 3.7: Initial state variable distribution for two interacting simple waves.

3.11.3 Convergence of entropy error

As a final test the convergence of the entropy error is determined. The entropy does not change when a flow is influence by an expansion fan (isentropic process), which means that the entropy in the whole tube at a certain time level has to be constant and equal to the initial entropy. From thermodynamic theory the following expression for the entropy can be derived:

$$s - s_0 = c_v \ln\left(\frac{\frac{p}{\rho^\gamma}}{\frac{p_0}{\rho_0^\gamma}}\right). \tag{3.43}$$

Such that one finally ends up with:

$$s = c_v \ln\left(\frac{p}{\rho^\gamma}\right). \tag{3.44}$$

This expression can be used to calculate the initial value of the entropy, resulting in $s_0 = 2551.406653$, which holds for all three regions. Running the calculations and taking the state variable distribution at time-level $t = 0.15$ allows for the calculation of the entropy distribution in the tube, indicated here with s_j. This specific time-level is used because the few errors occurring at the initial discontinuity have moved out of the tube at this moment. The error can now easily be obtained by taking the difference between the calculated value and the initial value of the entropy: Such that one finally ends up with:

$$\epsilon_s = s_j - s_0 \tag{3.45}$$

Because no discontinuities are present the L_1-error suffices, such that the total error is the mean of all errors. Performing these calculations results in the errors as depicted in table 3.8. A least-squares calculation of the order results in exactly the value 2.0.

J	ϵ_s
50	0.284939
100	0.081219
200	0.018508
400	0.004127
800	0.001133
p	-2.024807

Table 3.8: Results for convergence test for second-order Hancock scheme, using Koren's limiter.

Chapter 4
Conclusions

This report treats the development of a *shock tube* solver. This program is able to solve flows described by the one-dimensional Euler equations in a unit length tube, with either open or closed ends. Often such a shock tube is used for flows due to a discontinuous initial distribution of the state variables (density, velocity and pressure). In these problems shocks, expansions and contact discontinuities occur that travel through the tube and interact with each other.

A well-known problem of this type is the initial Riemann problem. Two initial states are separated by a contact discontinuity. When the states are released (a membrane is removed for example) the two gases start to interact, resulting in the characteristic distribution of two waves and a contact discontinuity. These Riemann problems can be solved exactly, making it a powerful tool in one-dimensional flow problems. The report describes the development of an exact Riemann solver, able to solve these initial Riemann problems. Five test cases have been used to test the solver for its performance. All tests delivered results that compared exactly with the literature.

Besides this exact approach, also a numerical one has been chosen. A Finite Volume Method has been used to discretize the Euler equations on the spatial (the tube) and temporal grid. The discretized equations are solved using the second-order Hancock method, which is a predictor-corrector type of method. Several limiters have been implemented in the shock tube solver, being the algebraic average, the Double Minmod and the Superbee limiter. Hancock's method is based upon determining the cell-interface fluxes, given a $left$ and $right$ state. For calculating the flux on the interface, the state variables on this interface should be known. This problem is exactly the same as a Riemann problem where the initial discontinuity is actually the cell-interface. Solving these interface fluxes is done using two different methods. The first is Godunov's exact Riemann solver and the second Roe's approximate Riemann solver. Both methods have been implemented in the schocktube code. Besides the general solver of Roe also the more simple version without an entropy fix is included. This version works fine in most cases, but fails when a transsonic expansion is present.

Several tests have been performed using the shock tube solver. The numerical results compared very well with the exact results. No significant differences between the Godunov and the Roe solver were present. The program was also tested for convergence using the less-known *fractional error norms*. Both the first-order upwind scheme and the second-order Hancock scheme returned their expected order with these error norms, which was not the case for the more common-used norms. This is because the contact discontinuity and the shock always produce errors of the first order, which dominate the rest of the error distribution. The limiters were also compared with each other. The order of convergence was the same for all limiters. Koren's limiter produced the smallest errors of the three limiters. The algebraic average, being linear, lacked behind. The entropy fix was tested using an adapted version of Sod's problem, with a transsonic expansion fan. The solver of Roe without an entropy fix resulted in a discontinuity in the expansion fan, while the solver with the fix prevented this from happening. Finally the double blast wave problem in a closed tube was treated. Two strong initial discontinuities in the pressure resulted in a complex wave pattern. The solver performed well on this test.

Further research should focus on the expanion of the one-dimensional solver to a two or three-dimensional version. Both the exact and approximate Riemann solver developed here can be implemented in these higher dimension solvers. Making the shock tube solver able to simulate two- or multi-fluid flows is another interesting possibility for further research. This will probably be the subject of the internship at the University of Michigan.

References

1. P.G. Bakker: *Lecture notes on Gasdynamics*, AE4-140, November 2001
2. R.L. Burden, J.D. Faires: *Numerical Analysis*, 7^{th} edition, 2001, Brooks/Cole
3. S.K. Godunov: *A Finite Difference Method for the Computation of Discontinuous Solutions of the Equations of Fluid Dynamics*, 1959 , Mat. Sb. 47 , pages 357-393
4. B. Koren. A robust upwind discretization method for advection, diffusion and source terms. *Numerical Methods for Advection-Diffusion Problems* (C.B. Vreugdenhil and B. Koren, eds.), Notes on Numerical Fluid Mechanics, 45, pages 117-138, Vieweg, Braunschweig, 1993.
5. B. van Leer: *Computer Problem 1*, Building you own shocktube, Winter 2004
6. M. Metcalf, J. Reid: *FORTRAN 90/95 explained*, 2^{nd} edition,1999, Oxford University Press
7. E.F. Toro: *Riemann Solvers and Numerical Methods for Fluid Dynamics*, A practical Introduction, 2^{nd} edition, 1999, Springer
8. P. Woodward, P. Colella: *The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks*, Review Article, 1984, Journal of Computational Physics 54, pages 115-173

Appendix I
Fractional error norms

It is a known fact that the numerical errors in non-smooth problems are dominated by the order one errors introduced in the few cells at the discontinuities in the solution. Determination of the order of convergence p of the numerical scheme used will therefore always return a value much lower than the expected order when using the more common error norms. A possible solution for this is the application of fractional error norms. This can be shown mathematically using the expression for the error norm of order n:

$$\epsilon_{L_n} = \left(\frac{\sum_1^J |\epsilon|^n}{J} \right)^{\frac{1}{n}}. \tag{I.1}$$

Assuming that the discontinuities are captured by k cells each having errors of order one $O(1)$. The other $(J-k)$ cells have all errors depending on the order of the chosen scheme p and the size of a cell h, indicated here as $O(h^p)$. Writing out the sum and assuming that the total error ϵ_{L_n} has to be of the same order as the scheme used gives:

$$\epsilon_{L_n} \sim \left(\frac{kO(1^n) + (J-k)O(h^{np})}{J} \right)^{\frac{1}{n}} \sim O(h^p). \tag{I.2}$$

Omitting the orders and rewriting the expression slightly gives:

$$\frac{k + (J-k)h^{np}}{J} \sim h^{np}. \tag{I.3}$$

Replacing J by $1/h$ and assuming that the total number of cells is much larger than the number of cells that capture the discontinuities, $J >> k$, and that $k \sim 1$, results in:

$$h + h^{np} \sim h^{np}, \tag{I.4}$$

which is the expression that has to be fulfilled for the error to be of the same order as the scheme [1]. This expression can be evaluated for different choices of n and p. This shall be done for both a first order numerical scheme ($p = 1$) and a second order scheme ($p = 2$).

I.1. First order scheme, $p = 1$

Insertion of $p = 1$ into (I.4) gives:

$$h + h^p \sim h^p. \tag{I.5}$$

Evaluating this expression for $n = 2$, $n = 1$ and $n = \frac{1}{2}$ results in the expressions as given in table I.1. Because h is very small ($h << 1$) for large numbers of cells, the lowest power of h dominates the left-hand side. For $n = 2$ this means that the errors of the discontinuities dominate the total error and make it an order lower than required. In the case of $n = 1$ the discontinuities deliver the same order as the other cells such that theoretically the required order will be obtained. But one could question if the few cells should have an equal influence on the total error as all the other cells. Therefore a third possible norm is the fractional error norm $n = \frac{1}{2}$, which results in the required order only now the errors in the cells at the discontinuities have a higher order compared to the other cells such that their influence has decreased.

[1]Note that the assumption $J >> k$ is justified because the power of the term $-kh^{np+1}$ in the final expression is always higher than the power of the other two terms

$p=2$	$h+h^2 \sim h^2$
$p=1$	$h+h \sim h$
$p=\frac{1}{2}$	$h+\sqrt{h} \sim \sqrt{h}$

Table I.1: Expressions for the total error for different error norms. First order numerical scheme.

I.2. SECOND ORDER SCHEME, $p = 2$

Insertion of $p = 2$ into (I.4) gives:

$$h + h^{2p} \sim h^{2p}. \tag{I.6}$$

Evaluating this expression for $n = 2$, $n = 1$, $n = \frac{1}{2}$ and $n = \frac{1}{4}$ results in the expressions as given in table I.2. As was the case for the first order scheme, the total error is dominated by the discontinuities in the case of the second order error norm. For $p = 1$ this is also the case. Only for $p = \frac{1}{2}$ and $p = \frac{1}{4}$ the total error reaches the required order. Whether to choose the $\frac{1}{2}$ or $\frac{1}{4}$ norm is still a question. Giving the discontinuous cells a less important role means choosing the latter of the two norms. Whether this choice is justified is still unknown and needs further investigation.

$p=2$	$h+h^4 \sim h^4$
$p=1$	$h+h^2 \sim h^2$
$p=\frac{1}{2}$	$h+h \sim h$
$p=\frac{1}{4}$	$h+\sqrt{h} \sim \sqrt{h}$

Table I.2: Expressions for the total error for different error norms. Second order numerical scheme.

Appendix II
Shocktube program code

```
 PROGRAM Shocktube

!---------------------------------------------------------------------!
!---------------------------------------------------------------------!
! Program: Shocktube                                                  !
!                                                                     !
! This program is able to solve Shocktube problems, and has been used !
! as an exercise to obtain more programming skills and to get used to !
! these kind of solvers. The 1D Euler equations have been used.       !
!                                                                     !
! It uses an Exact Riemann solver, where the pressure is              !
! obtained using a 'secant' iteration (approximate Newton-Raphson).   !
! Roe's approximation scheme is implemented to be compared with       !
! the exact solution                                                  !
!                                                                     !
! A Hancock Predictor-Corrector (MUSCL) scheme is used for the time   !
! marching. Setting dW = 0 everywhere leads to the first order scheme !
!                                                                     !
! This program contains the following elements:                       !
!                                                                     !
! - Muscl solver (with different limiters or 1st order approximation) !
! - Exact Riemann Solver (using secant iteration)                     !
! - Roe's approximate Riemann Solver                                  !
! - Roe's approximate Riemann Solver without entropy fix              !
!                                                                     !
! By:  Jorick Naber                                                   !
!      j.naber@student.tudelft.nl                                     !
!                                                                     !
! Written: 06-07-2004                                                 !
! Changed: 15-09-2004                                                 !
!---------------------------------------------------------------------!
!---------------------------------------------------------------------!


  USE Vars
  USE Solvers
  IMPLICIT NONE

!---------------------------------------------------------------------!
! Local variables                                                     !
!---------------------------------------------------------------------!
```

```
! Grid generation parameters
    INTEGER,                 PARAMETER :: NC = 200       ! # cells
    DOUBLE PRECISION,        PARAMETER :: L  = 1        ! Length
    DOUBLE PRECISION,        PARAMETER :: dx = L/NC     ! Stepsize
    DOUBLE PRECISION, DIMENSION(1:NC) :: x             ! Cell centres

! Grid generation variables
    INTEGER,              PARAMETER :: NT = 1500       ! # timesteps
    DOUBLE PRECISION,     PARAMETER :: T  = 0.01      ! End Time
    DOUBLE PRECISION,     PARAMETER :: dt = T/NT      ! Stepsize

! Boundary conditions (B = 1 for soft & B = 2 for hard)
    INTEGER, PARAMETER :: B = 1

! Counters
    INTEGER :: i,j,n

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1,0:NT) :: W ! Primary
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1,0:NT) :: V ! Conservative
    DOUBLE PRECISION, DIMENSION(1:3,1:2*(NC+2))  :: Wf! Cell face st.
    DOUBLE PRECISION, DIMENSION(1:3,1:NC+1,0:NT) :: F ! Flux vector

! Riemann invariants
    DOUBLE PRECISION, DIMENSION(1:NC,0:NT) :: Jplus  ! Riemann inv.
    DOUBLE PRECISION, DIMENSION(1:NC,0:NT) :: Jmin   ! Riemann inv.
    DOUBLE PRECISION, DIMENSION(1:NC,0:NT) :: a      ! Speed of sound

! Entropy
    DOUBLE PRECISION, DIMENSION(1:NC) :: s   ! Entropy
    DOUBLE PRECISION                  :: s0  ! Initial entropy
    DOUBLE PRECISION, DIMENSION(1:NC) :: es  ! Entropy error
    DOUBLE PRECISION                  :: Tes ! Total entropy error

!----------------------------------------------------------------------!
! Start calculations                                                   !
!----------------------------------------------------------------------!

  WRITE(*,*) 'Shocktube, 1D Euler'
  WRITE(*,*) '  '
  WRITE(*,*) 'Calculations have started'
  WRITE(*,*) '...'
  WRITE(*,*) '...'
  WRITE(*,*) '...'

!----------------------------------------------------------------------!
! Generate grid                                                        !
!----------------------------------------------------------------------!

! Coordinates of cell centres
```

```
    DO j = 1,NC
      x(j) = (j - 0.5)*dx
    END DO

!----------------------------------------------------------------------!
! Initial condition                                                    !
!----------------------------------------------------------------------!

! Conditions at t=0

!      W(1,1:NC/2,0) = 1.0
!      W(2,1:NC/2,0) = 0.0
!      W(3,1:NC/2,0) = 1.0

!       W(1,NC/2+1:NC,0) = 0.125
!       W(2,NC/2+1:NC,0) = 0.0
!       W(3,NC/2+1:NC,0) = 0.1

      W(1,1:NC/4,0) = 0.462664366
      W(2,1:NC/4,0) = -5.0
      W(3,1:NC/4,0) = 11.8970837

      W(1,NC/4+1:3*NC/4,0) = 1.0
      W(2,NC/4+1:3*NC/4,0) = 0.0
      W(3,NC/4+1:3*NC/4,0) = 35.0

      W(1,3*NC/4+1:NC,0) = 0.69036153
      W(2,3*NC/4+1:NC,0) = 2.5
      W(3,3*NC/4+1:NC,0) = 20.83412477

!----------------------------------------------------------------------!
! March in time                                                        !
!----------------------------------------------------------------------!

  DO n = 1,NT

!----------------------------------------------------------------------!
! Boundary conditions                                                  !
!----------------------------------------------------------------------!

    SELECT CASE (B)

! Soft boundary
    CASE (1)
      W(:,0,n-1) = W(:,1,n-1)
      W(:,NC+1,n-1) = W(:,NC,n-1)

! Hard boundary
    CASE (2)
      W(1,0,n-1) = W(1,1,n-1)
      W(1,NC+1,n-1) = W(1,NC,n-1)
      W(2,0,n-1) = -W(2,1,n-1)
      W(2,NC+1,n-1) = -W(2,NC,n-1)

      W(3,0,n-1) = W(3,1,n-1)
      W(3,NC+1,n-1) = W(3,NC,n-1)

    END SELECT

!----------------------------------------------------------------------!
! Call MUSCL scheme                                                    !
!----------------------------------------------------------------------!

  CALL Muscl(NC,dx,dt,B,W(:,:,n-1),Wf(:,:))

!----------------------------------------------------------------------!
! Call Exact Riemann solver                                            !
!----------------------------------------------------------------------!

 CALL Exact(NC,Wf(:,:),F(:,:,n-1))

!----------------------------------------------------------------------!
! Call Roe's approximate Riemann solver                                !
!----------------------------------------------------------------------!

! CALL Roe(NC,Wf(:,:),F(:,:,n-1))

!----------------------------------------------------------------------!
! Call Roe's approximate Riemann solver without entropy fix           !
!----------------------------------------------------------------------!

! CALL RoeNoFix(NC,Wf(:,:),F(:,:,n-1))

!----------------------------------------------------------------------!
! Solve for new timestep using fluxes                                  !
!----------------------------------------------------------------------!

  DO j = 1,NC

! Convert from primary (W) to conservation variables (U)

    V(1,j,n-1) = W(1,j,n-1)
    V(2,j,n-1) = W(1,j,n-1)*W(2,j,n-1)
    V(3,j,n-1) = W(3,j,n-1)/(gamma-1) + 0.5*W(1,j,n-1)*W(2,j,n-1)**(2)

! Time step

    V(:,j,n) = V(:,j,n-1) - dt*(F(:,j+1,n-1) - F(:,j,n-1))/dx

! Convert from conservation (U) to primary (W) variables
```

```
    W(1,j,n) = V(1,j,n)
    W(2,j,n) = V(2,j,n)/V(1,j,n)
    W(3,j,n) = (gamma-1)*(V(3,j,n) - 0.5*V(2,j,n)**(2)/V(1,j,n))

  END DO

  END DO


!----------------------------------------------------------------------!
! Calculate Riemann invariants for 2D contourplot                      !
!----------------------------------------------------------------------!

  DO n = 0,NT
    DO j = 1,NC

      a(j,n) = sqrt(gamma*W(3,j,n)/W(1,j,n))

      Jplus(j,n) = W(2,j,n) + gamma6*a(j,n)
      Jmin(j,n) = W(2,j,n) - gamma6*a(j,n)

    END DO
  END DO

!----------------------------------------------------------------------!
! Calculate Entropy                                                    !
!----------------------------------------------------------------------!

! Initial entropy
    s0 = log(W(3,NC/2,0)/(W(1,NC/2,0)**(gamma)))

! Entropy distribution
    DO j = 1,NC

      s(j) = log(W(3,j,NT)/(W(1,j,NT)**(gamma)))

    END DO

! Entropy error

    Tes = 0.0

    DO j = 1,NC

      es(j) = s(j) - s0

      Tes = Tes + es(j)/NC

    END DO

!----------------------------------------------------------------------!
```

```
! Print results to file                                              !
!--------------------------------------------------------------------!

! Results at t=n+1 (at timestep NT)
    OPEN (1, file='Output1.txt', access='sequential')
    DO j = 1,NC
      WRITE (1,'(4f15.5)') x(j), W(1,j,NT), W(2,j,NT), W(3,j,NT)
    END DO
    CLOSE (1)

! Results at t=n+1 (at timestep NT)
!    OPEN (2, file='Output2.txt', access='sequential')
!  DO n = 0,NT
!    DO j = 1,NC
!      WRITE (2,'(4f15.5)') x(j), (dt*n), Jplus(j,n), Jmin(j,n)
!    END DO
!  END DO
!   CLOSE (2)

!--------------------------------------------------------------------!
! End of calculations                                                !
!--------------------------------------------------------------------!

WRITE (*,*) 'Calculations have finished'
WRITE (*,*) '   '
WRITE (*,*) 'See files "Output*.txt" for results'
WRITE (*,*) '   '

END PROGRAM Shocktube
```

```
MODULE Solvers

!--------------------------------------------------------------------!
! Select the packages for the definitions and subroutines           !
!--------------------------------------------------------------------!

   USE Vars
   IMPLICIT NONE

! Some grid routines
   PUBLIC :: Exact
   PUBLIC :: Secant
   PUBLIC :: Roe
   PUBLIC :: RoeNoFix
   PUBLIC :: Muscl
   PUBLIC :: Average
   PUBLIC :: DMinmod
   PUBLIC :: Superbee
   PUBLIC :: Koren

CONTAINS

  SUBROUTINE Exact(NC,Wf,F)

!--------------------------------------------------------------------!
!--------------------------------------------------------------------!
! Subroutine: Exact                                                  !
!                                                                    !
! Exact Riemann solver for shocktube program                         !
!                                                                    !
! Written: 06-07-2004                                                !
! Changed:                                                           !
!--------------------------------------------------------------------!
!--------------------------------------------------------------------!

!--------------------------------------------------------------------!
! Global variables                                                   !
!--------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!--------------------------------------------------------------------!
! Local variables                                                    !
!--------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
```

```
    INTEGER :: j

! Initial states
    DOUBLE PRECISION, DIMENSION(1:3,1:2*(NC+2)) :: Wf
    DOUBLE PRECISION :: rho1,rho4,p1,p4,u1,u4
    DOUBLE PRECISION :: a1,a4,M1,M4

! Tuning of solver
    DOUBLE PRECISION, PARAMETER :: eps = 1e-7

! Final states
    DOUBLE PRECISION :: rho2,rho3,p23,u23,a2,a3
    DOUBLE PRECISION :: rhof,pf,uf,af

! Flux vector
    DOUBLE PRECISION, DIMENSION(1:3,1:NC+1) :: F

! Wave speeds
    DOUBLE PRECISION :: G0,G1,G2,G3,G4,w0,w1,w2,w3,w4
    DOUBLE PRECISION, DIMENSION(1:5) :: w

! Solving for time axis
    INTEGER :: K,N
    INTEGER, DIMENSION(1) :: M

!--------------------------------------------------------------------!
! Loop for every cell face                                           !
!--------------------------------------------------------------------!

  DO j = 1,(NC+1)

!--------------------------------------------------------------------!
! Initial states                                                     !
!--------------------------------------------------------------------!

! Density
    rho1 = Wf(1,2*j)
    rho4 = Wf(1,2*j+1)

! Velocity
    u1 = Wf(2,2*j)
    u4 = Wf(2,2*j+1)

! Pressure
    p1 = Wf(3,2*j)
    p4 = Wf(3,2*j+1)

!--------------------------------------------------------------------!
! Test for negative pressure and density                             !
!--------------------------------------------------------------------!
```

```
! Density
    IF (rho1 <= 0.0) THEN
      rho1 = eps
    END IF
    IF (rho4 <= 0.0) THEN
      rho4 = eps
    END IF

! Pressure
    IF (p1 <= 0.0) THEN
      p1 = eps
    END IF
    IF (p4 <= 0.0) THEN
      p4 = eps
    END IF

!----------------------------------------------------------------------!
! Speed of sound and Mach number                                       !
!----------------------------------------------------------------------!

! Speed of sound
    a1 = sqrt(gamma*p1/rho1)
    a4 = sqrt(gamma*p4/rho4)

! Mach numbers
    M1 = u1/a1
    M4 = u4/a4

!----------------------------------------------------------------------!
! Test for supersonic flow                                             !
!----------------------------------------------------------------------!

 !Flow is supersonic to the right
    IF (M1 >= 1.0 .AND. M4 >= 1.0) THEN
      rhof = rho1
      uf = u1
      pf = p1
      af = a1

 !Flow is supersonic to the left
    ELSE IF (M1 <= -1.0 .AND. M4 <= -1.0) THEN
      rhof = rho4
      uf = u4
      pf = p4
      af = a4

 !Flow is not supersonic, continue with program
    ELSE

!----------------------------------------------------------------------!
! Test for cavity                                                      !
```

```
!----------------------------------------------------------------------!

! Cavity occurs - Calculations have been stopped
    IF ((u1 + gamma6*a1) < (u4 - gamma6*a4)) THEN

        WRITE(*,*) 'Cavity Occurs'
        WRITE(*,*) 'Program has stopped'

        STOP

! No cavity occurs, continue with program
     ELSE

!----------------------------------------------------------------------!
! Call iterative solver for p23                                        !
!----------------------------------------------------------------------!

    CALL Secant(rho1,rho4,p1,p4,u1,u4,a1,a4,p23,u23)

!----------------------------------------------------------------------!
! Calculate final states                                               !
!----------------------------------------------------------------------!

! Density

! Left running wave
    IF (p23 >= p1) THEN
      rho2 = rho1*(1.0 + gamma9*p23/p1)/(gamma9 + p23/p1)
    ELSE IF (p23 < p1) THEN
      rho2 = rho1*(p23/p1)**(1.0/gamma)
    END IF

! Right running wave
    IF (p23 >= p4) THEN
      rho3 = rho4*(1.0 + gamma9*p23/p4)/(gamma9 + p23/p4)
    ELSE IF (p23 < p4) THEN
      rho3 = rho4*(p23/p4)**(1.0/gamma)
    END IF

! Speed of sound
    a2 = sqrt(gamma*p23/rho2)
    a3 = sqrt(gamma*p23/rho3)

!----------------------------------------------------------------------!
! Determine characteristics                                            !
!----------------------------------------------------------------------!

    G1 = u1 - a1 ! 1st LEFT RUNNING CHARACTERISTIC
    G2 = u23 - a2 ! 2nd LEFT RUNNING CHARACTERISTIC
    G3 = u23 + a3 ! 1st RIGHT RUNNING CHARACTERISTIC
    G4 = u4 + a4 ! 2nd RIGHT RUNNING CHARACTERISTIC
    G0 = u23 ! ENTROPY WAVE

!----------------------------------------------------------------------!
! Determine wave speeds                                                !
!----------------------------------------------------------------------!

! Entropy wave
     w0 = G0

! Left running wave
    IF (p23 >= p1) THEN
      w1 = u1-a1*sqrt(1.0+gamma3*((p23/p1)-1.0))
      w2 = w1
    ELSE IF (p23 < p1) THEN
      w1 = G1
      w2 = G2
    END IF

! Right running wave
    IF (p23 >= p4) THEN
      w4 = u4+a4*sqrt(1.0+gamma3*((p23/p4)-1.0))
      w3 = w4
    ELSE IF (p23 < p4) THEN
      w4 = G4
      w3 = G3
    END IF

! Wave vector:
    w(1) = w1
    w(2) = w2
    w(3) = w0
    w(4) = w3
    w(5) = w4

!----------------------------------------------------------------------!
! Determine the conditions on time axis                                !
!----------------------------------------------------------------------!

! Wave closest to time axis
    M = minloc(abs(w))
    K = M(1)

! Numbering of areas
!   -inf|w1  -> N=1    LEFT INITIAL STATE
!     w1|w2  -> N=2    LEFT EXPANSION FAN
!     w2|w0  -> N=3    LEFT FINAL STATE
!     w0|w3  -> N=4    RIGHT FINAL STATE
!     w3|w4  -> N=5    RIGHT EXPANSIION FAN
!     w4|inf -> N=6    RIGHT INITIAL STATE

! Region of time axis
```

```
    IF (w(K) >= 0) THEN
      N = K
    ELSE IF (w(K) < 0 .AND. w(K) /= w(K+1)) THEN
      N = K+1
    ELSE IF (w(K) < 0 .AND. w(K) == w(K+1)) THEN
      N = K+2
    END IF

! Determining final conditions on time axis
    SELECT CASE (N)
    CASE (1)
      uf = u1
      af = a1
      pf = p1
      rhof = rho1
    CASE (2)
      uf = gamma8*a1 + gamma10*u1
      af = gamma8*a1 + gamma10*u1
      pf = p1*(af/a1)**(gamma2)
      rhof = rho1*(af/a1)**(gamma6)
    CASE (3)
      uf = u23
      af = a2
      pf = p23
      rhof = rho2
    CASE (4)
      uf = u23
      af = a3
      pf = p23
      rhof = rho3
    CASE (5)
      uf = -gamma8*a4 + gamma10*u4
      af = gamma8*a4 - gamma10*u4
      pf = p4*(af/a4)**(gamma2)
      rhof = rho4*(af/a4)**(gamma6)
    CASE(6)
      uf = u4
      af = a4
      pf = p4
      rhof = rho4
    END SELECT

! End for cavity test loop
    END IF

! End for supersonic test loop
    END IF

!----------------------------------------------------------------------!
! Calculate fluxes on cell faces                                       !
!----------------------------------------------------------------------!
```

```
     F(1,j) = rhof*uf
     F(2,j) = rhof*uf**(2) + pf
     F(3,j) = rhof*uf*(pf/(rhof*(gamma-1.0)) + pf/rhof + 0.5*uf**(2))

!----------------------------------------------------------------------!
! End of loop for cell faces                                           !
!----------------------------------------------------------------------!

  END DO
  RETURN
  END SUBROUTINE Exact




  SUBROUTINE Secant(rho1,rho4,p1,p4,u1,u4,a1,a4,p23,u23)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Secant                                                   !
!                                                                      !
! Iterative solver for p23 for shocktube program                       !
!                                                                      !
! Written: 07-07-2004                                                  !
! Changed:                                                             !
!----------------------------------------------------------------------!
!----------------------------------------------------------------------!

!----------------------------------------------------------------------!
! Global variables                                                     !
!----------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!----------------------------------------------------------------------!
! Local variables                                                      !
!----------------------------------------------------------------------!

! Errors
    DOUBLE PRECISION, PARAMETER :: emax = 10e-6
    DOUBLE PRECISION :: e
    DOUBLE PRECISION, PARAMETER :: small = 10e-15

! Initial states
    DOUBLE PRECISION :: rho1,rho4,p1,p4,u1,u4,a1,a4

! Final states
    DOUBLE PRECISION :: p23,u23,m1,m4
```

```
! Pressure iteration
    INTEGER :: n
    INTEGER, PARAMETER :: nmax = 20
    DOUBLE PRECISION :: p_up,m1_up,m4_up
    DOUBLE PRECISION :: p_low,m1_low,m4_low

! Iteration function
    DOUBLE PRECISION :: pa,pb,pc,fa,fb,m1a,m1b,m4a,m4b

!----------------------------------------------------------------------!
! Determine upper bound for p23                                        !
!----------------------------------------------------------------------!

    p_up = (((u1-u4)*gamma5 + (a1+a4))/&
           &(a1/(p1)**(gamma1)+a4/(p4)**(gamma1)))**(gamma2)

!----------------------------------------------------------------------!
! Test for double expansion                                            !
!----------------------------------------------------------------------!

! No shock is present - Double expansion - p23 = p_up
    IF (p_up < p1 .AND. p_up < p4) THEN
      p23 = p_up
      m1_up  = rho1*a1*gamma1*(1.0-p23/p1)/(1.0-(p23/p1)**(gamma1)+small)
      m4_up  = rho4*a4*gamma1*(1.0-p23/p4)/(1.0-(p23/p4)**(gamma1)+small)
      u23 = (m1_up*u1 + m4_up*u4 - (p4-p1))/(m1_up+m4_up+small)

! At least one shock is present, continue with program
    ELSE

!----------------------------------------------------------------------!
! Determine lower bound for p23                                        !
!----------------------------------------------------------------------!

!    m1_low = rho1*a1
!    m4_low = rho4*a4

!    p_low = (m1_low*p4 + m4_low*p1 - m1_low*m4_low*(u4-u1))/&
!           &(m1_low+m4_low)


!----------------------------------------------------------------------!
! Initial guesses for p23                                              !
!----------------------------------------------------------------------!

    pa = 0.95*p_up
    pb = p_up

!----------------------------------------------------------------------!
! Calculation of iteration step a!                                     !
!----------------------------------------------------------------------!
```

```
! Left running wave
    IF (pa >= p1) THEN
        m1a = rho1*a1*sqrt(1+gamma3*(pa/p1 - 1))
    ELSE IF (pa < p1) THEN
        m1a = rho1*a1*gamma1*(1-pa/p1)/(1-(pa/p1)**(gamma1)+small)
    END IF

! Right running wave
    IF (pa >= p4) THEN
        m4a = rho4*a4*sqrt(1+gamma3*(pa/p4 - 1))
    ELSE IF (pa < p4) THEN
        m4a = rho4*a4*gamma1*(1-pa/p4)/(1-(pa/p4)**(gamma1)+small)
    END IF

    fa = pa - (m1a*p4 + m4a*p1 - m1a*m4a*(u4-u1))/(m1a+m4a+small)

!--------------------------------------------------------------------!
! Calculation of iteration step c, using a and b                     !
!--------------------------------------------------------------------!

! Count iterations
    n = 0.0

    DO n = 1,nmax

! Left running wave
    IF (pb >= p1) THEN
        m1b = rho1*a1*sqrt(1.0 + gamma3*(pb/p1 - 1.0))
    ELSE IF (pb < p1) THEN
        m1b = rho1*a1*gamma1*(1.0 - pb/p1)/(1.0 - (pb/p1)**(gamma1)+small)
    END IF

! Right running wave
    IF (pb >= p4) THEN
        m4b = rho4*a4*sqrt(1.0 + gamma3*(pb/p4 - 1.0))
    ELSE IF (pb < p4) THEN
        m4b = rho4*a4*gamma1*(1.0 - pb/p4)/(1.0 - (pb/p4)**(gamma1)+small)
    END IF

    fb = pb - (m1b*p4 + m4b*p1 - m1b*m4b*(u4 - u1))/(m1b + m4b+small)

! Iterated pressure
    pc = pb - fb*(pb - pa)/(fb - fa+small)

! Error (absolute)
    e = abs(pc - pb)/(0.5*(pc + pb)+small)

! Test for error
    IF (e <= emax) EXIT
```

```
! New iterates
    pa = pb
    pb = pc
    fa = fb

! Test for convergence
    IF (n == nmax) THEN
        WRITE(*,*) 'No convergence of pressure iteration'
        WRITE(*,*) 'Program has stopped'
        STOP
    END IF

    END DO

!----------------------------------------------------------------------!
! Calculation of final states                                          !
!----------------------------------------------------------------------!

! Pressure
    p23 = pc

! Mass flow through left running wave
    IF (p23 >= p1) THEN
        m1 = rho1*a1*sqrt(1.0 + gamma3*(p23/p1 - 1.0))
    ELSE IF (p23 < p1) THEN
        m1 = rho1*a1*gamma1*(1.0 - p23/p1)/(1.0 - (p23/p1)**(gamma1)+small)
    END IF
! Mass flow through right running wave
    IF (p23 >= p4) THEN
        m4 = rho4*a4*sqrt(1.0 + gamma3*(p23/p4 - 1.0))
    ELSE IF (p23 < p4) THEN
        m4 = rho4*a4*gamma1*(1.0 - p23/p4)/(1.0 - (p23/p4)**(gamma1)+small)
    END IF

! Velocity
    u23 = (m1*u1 + m4*u4 - (p4-p1))/(m1+m4+small)

! End for double expansion test loop
    END IF

  RETURN
  END SUBROUTINE Secant


  SUBROUTINE Roe(NC,Wf,F)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Roe                                                      !
!                                                                      !
! Roe's approximate Riemann solver for shocktube program       !
!                                                                    !
! Written: 19-07-2004                                                !
! Changed:                                                           !
!--------------------------------------------------------------------!
!--------------------------------------------------------------------!

!--------------------------------------------------------------------!
! Global variables                                                   !
!--------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!--------------------------------------------------------------------!
! Local variables                                                    !
!--------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
    INTEGER :: i,j

! Initial states
    DOUBLE PRECISION, DIMENSION(1:3,1:2*(NC+2)) :: Wf
    DOUBLE PRECISION :: rho1,rho4,p1,p4,u1,u4
    DOUBLE PRECISION :: a1,a4,M1,M4,H1,H4

! Tuning of solver
    DOUBLE PRECISION, PARAMETER :: eps = 1e-15

! Roe-averaged states
    DOUBLE PRECISION :: w
    DOUBLE PRECISION :: rhoh,uh,Hh,ah

! Final states
    DOUBLE PRECISION :: rho2,rho3,p23,u23,a2,a3
    DOUBLE PRECISION :: rhof,pf,uf,af

! Flux vector
    DOUBLE PRECISION, DIMENSION(1:3,1:NC+1) :: F
    DOUBLE PRECISION, DIMENSION(1:3)        :: F1,F4
    DOUBLE PRECISION, PARAMETER             :: one = 1.0

! Eigenvalues and vectors
    DOUBLE PRECISION, DIMENSION(1:3)        :: lambda
    DOUBLE PRECISION                        :: lambda14,lambda11,lambda34,lambda31
    DOUBLE PRECISION, DIMENSION(1:2)        :: dlam1,dlam2,dlam3,dlam4
    DOUBLE PRECISION, DIMENSION(1:3)        :: dlambda
    DOUBLE PRECISION, DIMENSION(1:3)        :: lambdamin,lambdaplus
    DOUBLE PRECISION, DIMENSION(1:3)        :: R1,R2,R3
```

```
! Jump Relations
    DOUBLE PRECISION                        :: dp,du,drho
    DOUBLE PRECISION, DIMENSION(1:3)        :: dV
!-----------------------------------------------------------------------!
! Loop for every cell face                                              !
!-----------------------------------------------------------------------!

  DO j = 1,(NC+1)

!-----------------------------------------------------------------------!
! Initial states                                                        !
!-----------------------------------------------------------------------!

! Density
    rho1 = Wf(1,2*j)
    rho4 = Wf(1,2*j+1)

! Velocity
    u1 = Wf(2,2*j)
    u4 = Wf(2,2*j+1)

! Pressure
    p1 = Wf(3,2*j)
    p4 = Wf(3,2*j+1)

!-----------------------------------------------------------------------!
! Test for negative pressure and density                                !
!-----------------------------------------------------------------------!

! Density
    IF (rho1 <= 0.0) THEN
      rho1 = eps
    END IF
    IF (rho4 <= 0.0) THEN
      rho4 = eps
    END IF

! Pressure
    IF (p1 <= 0.0) THEN
      p1 = eps
    END IF
    IF (p4 <= 0.0) THEN
      p4 = eps
    END IF

!-----------------------------------------------------------------------!
! Speed of sound and Enthalpy                                           !
!-----------------------------------------------------------------------!

! Speed of sound
```

```
    a1 = sqrt(gamma*p1/rho1)
    a4 = sqrt(gamma*p4/rho4)

! Enthalpy
    H1 = p1/(rho1*(gamma-1.0)) + p1/rho1 + 0.5*u1**(2)
    H4 = p4/(rho4*(gamma-1.0)) + p4/rho4 + 0.5*u4**(2)

!----------------------------------------------------------------------!
! Test for cavity                                                      !
!----------------------------------------------------------------------!

! Cavity occurs - Calculations have been stopped
    IF ((u1 + gamma6*a1) < (u4 - gamma6*a4)) THEN

        WRITE(*,*) 'Cavity Occurs'
        WRITE(*,*) 'Program has stopped'

        STOP

! No cavity occurs, continue with program
     ELSE

!----------------------------------------------------------------------!
! Roe-averaged state quantities                                        !
!----------------------------------------------------------------------!

! Ratio
    w = sqrt(rho4/rho1)

! Density
    rhoh = w*rho1

! Velocity
    uh = (u1 + w*u4)/(1.0 + w)

! Enthalpy
    Hh = (H1 + w*H4)/(1.0 + w)

! Speed of sound
    ah = sqrt((gamma - 1.0)*(Hh - 0.5*uh**(2)))

!----------------------------------------------------------------------!
! Initial flux vectors                                                 !
!----------------------------------------------------------------------!

! Flux vector left of cell face
    F1(1) = rho1*u1
    F1(2) = rho1*u1**(2) + p1
    F1(3) = rho1*u1*(p1/rho1/(gamma-1.0) + p1/rho1 + 0.5*u1**(2))

! Flux vector left of cell face
```

```
    F4(1) = rho4*u4
    F4(2) = rho4*u4**(2) + p4
    F4(3) = rho4*u4*(p4/rho4/(gamma-1.0) + p4/rho4 + 0.5*u4**(2))

!----------------------------------------------------------------------!
! Eigenvalues and vectors                                              !
!----------------------------------------------------------------------!

! Eigenvalues
    lambda(1) = uh - ah
    lambda(2) = uh
    lambda(3) = uh + ah

! Eigenvectors
    R1(1) = 1.0
    R1(2) = uh - ah
    R1(3) = Hh - uh*ah

    R2(1) = 1.0
    R2(2) = uh
    R2(3) = 0.5*uh**(2)

    R3(1) = 1.0
    R3(2) = uh + ah
    R3(3) = Hh + uh*ah

! Modified absolute eigenvalues

    lambda14 = u4 - a4
    lambda11 = u1 - a1
    lambda34 = u4 + a4
    lambda31 = u1 + a1

    dlam3(1) = 0.0
    dlam3(2) = 2*(lambda14 - lambda11)
    dlam4(1) = 0.0
    dlam4(2) = 2*(lambda34 - lambda31)

    dlam1(1) = ah
    dlam1(2) = maxval(dlam3)
    dlam2(1) = ah
    dlam2(2) = maxval(dlam4)

    dlambda(1) = minval(dlam1)
    dlambda(2) = 0.0
    dlambda(3) = minval(dlam2)

    DO i = 1,3
      IF (lambda(i) <= -dlambda(i)) THEN
        lambdamin(i) = lambda(i)
        lambdaplus(i) = 0.0
      ELSE IF (lambda(i) > -dlambda(i) .AND. lambda(i) < dlambda(i)) THEN
        lambdamin(i) = -(lambda(i) - dlambda(i))**(2)/(4*dlambda(i))
        lambdaplus(i) = (lambda(i) + dlambda(i))**(2)/(4*dlambda(i))
      ELSE IF (lambda(i) >= dlambda(i)) THEN
        lambdamin(i) = 0.0
        lambdaplus(i) = lambda(i)
      END IF
    END DO

!----------------------------------------------------------------------!
! Jump relations                                                        !
!----------------------------------------------------------------------!

! Pressure and velocity jump
    drho = rho4 - rho1
    du = u4 - u1
    dp = p4 - p1

! Jump vector

    dV(1) = (dp - rhoh*ah*du)/(2*ah**(2))
    dV(2) = -(dp - ah**(2)*drho)/(ah**(2))
    dV(3) = (dp + rhoh*ah*du)/(2*ah**(2))

!----------------------------------------------------------------------!
! Flux vector at interface                                              !
!----------------------------------------------------------------------!

! Flux vector
    DO i = 1,3
      IF (uh >= 0.0) THEN
        F(i,j) = F1(i) + lambdamin(1)*dV(1)*R1(i)
      ELSE IF (uh < 0.0) THEN
         F(i,j) = F4(i) - lambdaplus(3)*dV(3)*R3(i)
      END IF
    END DO

!----------------------------------------------------------------------!
! End of loop for cavity test                                           !
!----------------------------------------------------------------------!

  END IF

!----------------------------------------------------------------------!
! End of loop for cell faces                                            !
!----------------------------------------------------------------------!

  ENDDO
  RETURN
  END SUBROUTINE Roe
  SUBROUTINE RoeNoFix(NC,Wf,F)

!--------------------------------------------------------------------!
!--------------------------------------------------------------------!
! Subroutine: RoeNoFix                                               !
!                                                                    !
! Roe’s approximate Riemann solver for shocktube program       !
! No enthalpy fix is present                                         !
!                                                                    !
! Written: 19-07-2004                                                !
! Changed:                                                           !
!--------------------------------------------------------------------!
!--------------------------------------------------------------------!

!--------------------------------------------------------------------!
! Global variables                                                   !
!--------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!--------------------------------------------------------------------!
! Local variables                                                    !
!--------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
    INTEGER :: i,j

! Initial states
    DOUBLE PRECISION, DIMENSION(1:3,1:2*(NC+2)) :: Wf
    DOUBLE PRECISION :: rho1,rho4,p1,p4,u1,u4
    DOUBLE PRECISION :: dp,du
    DOUBLE PRECISION :: a1,a4,M1,M4,H1,H4

! Tuning of solver
    DOUBLE PRECISION, PARAMETER :: eps1 = 0.01D0
    DOUBLE PRECISION, PARAMETER :: eps2 = 0.01D0
    DOUBLE PRECISION, PARAMETER :: eps3 = 0.01D0
    DOUBLE PRECISION, PARAMETER :: eps4 = 0.01D0

! Roe-averaged states
    DOUBLE PRECISION :: w
    DOUBLE PRECISION :: rhoh,uh,Hh,ah

! Final states
```

```
    DOUBLE PRECISION :: rho2,rho3,p23,u23,a2,a3
    DOUBLE PRECISION :: rhof,pf,uf,af

! Flux vector
    DOUBLE PRECISION, DIMENSION(1:3,1:NC+1) :: F
    DOUBLE PRECISION, DIMENSION(1:3)        :: F1,F4
    DOUBLE PRECISION                        :: sigma1,sigma2
    DOUBLE PRECISION, PARAMETER             :: one = 1.0
    DOUBLE PRECISION, DIMENSION(1:3)        :: R

!----------------------------------------------------------------------!
! Loop for every cell face                                             !
!----------------------------------------------------------------------!

  DO j = 1,(NC+1)

!----------------------------------------------------------------------!
! Initial states                                                       !
!----------------------------------------------------------------------!

! Density
    rho1 = Wf(1,2*j)
    rho4 = Wf(1,2*j+1)

! Velocity
    u1 = Wf(2,2*j)
    u4 = Wf(2,2*j+1)

! Pressure
    p1 = Wf(3,2*j)
    p4 = Wf(3,2*j+1)

!----------------------------------------------------------------------!
! Test for negative pressure and density                               !
!----------------------------------------------------------------------!

! Density
    IF (rho1 <= 0.0) THEN
      rho1 = eps1
    END IF
    IF (rho4 <= 0.0) THEN
      rho4 = eps2
    END IF

! Pressure
    IF (p1 <= 0.0) THEN
      p1 = eps3
    END IF
    IF (p4 <= 0.0) THEN
      p4 = eps4
    END IF
```

```
!----------------------------------------------------------------------!
! Speed of sound and Enthalpy                                          !
!----------------------------------------------------------------------!

! Speed of sound
    a1 = sqrt(gamma*p1/rho1)
    a4 = sqrt(gamma*p4/rho4)

! Enthalpy
    H1 = p1/(rho1*(gamma-1.0)) + p1/rho1 + 0.5*u1**(2)
    H4 = p4/(rho4*(gamma-1.0)) + p4/rho4 + 0.5*u4**(2)

!----------------------------------------------------------------------!
! Test for cavity                                                      !
!----------------------------------------------------------------------!

! Cavity occurs - Calculations have been stopped
    IF ((u1 + gamma6*a1) < (u4 - gamma6*a4)) THEN

        WRITE(*,*) 'Cavity Occurs'
        WRITE(*,*) 'Program has stopped'

        STOP

! No cavity occurs, continue with program
     ELSE

!----------------------------------------------------------------------!
! Roe-averaged state quantities                                        !
!----------------------------------------------------------------------!

! Ratio
    w = sqrt(rho4/rho1)

! Density
    rhoh = w*rho1

! Velocity
    uh = (u1 + w*u4)/(1.0 + w)

! Enthalpy
    Hh = (H1 + w*H4)/(1.0 + w)

! Speed of sound
    ah = sqrt((gamma - 1.0)*(Hh - 0.5*uh**(2)))

!----------------------------------------------------------------------!
! Initial flux vectors                                                 !
!----------------------------------------------------------------------!
```

```
! Flux vector left of cell face
    F1(1) = rho1*u1
    F1(2) = rho1*u1**(2) + p1
    F1(3) = rho1*u1*(p1/rho1/(gamma-1.0) + p1/rho1 + 0.5*u1**(2))

! Flux vector left of cell face
    F4(1) = rho4*u4
    F4(2) = rho4*u4**(2) + p4
    F4(3) = rho4*u4*(p4/rho4/(gamma-1.0) + p4/rho4 + 0.5*u4**(2))

!----------------------------------------------------------------------!
! Flux vector at interface                                             !
!----------------------------------------------------------------------!

! Pressure and velocity jump
    dp = p4 - p1
    du = u4 - u1

! Switches

    IF (uh == 0.0) THEN
       sigma1 = 0.0
    ELSE
       sigma1 = sign(one, uh)
    END IF

    IF ((uh**(2) - ah**(2)) == 0.0) THEN
       sigma2 = 0.0
    ELSE
       sigma2 = sign(one, (uh**(2) - ah**(2)))
    END IF


! Vector
    R(1) = 1.0
    R(2) = uh - sigma1*ah
    R(3) = Hh - sigma1*uh*ah

! Flux vector
    DO i = 1,3
      F(i,j) = (1.0+sigma1)*F1(i)/2.0 + (1.0-sigma1)*F4(i)/2.0 -&
               &(1.0-sigma2)*(uh - sigma1*ah)/2*&
               &(rhoh*ah*du-sigma1*dp)*R(i)/(2*ah**(2))
    END DO

!----------------------------------------------------------------------!
! End of loop for cavity test                                          !
!----------------------------------------------------------------------!

    END IF
```

```
!----------------------------------------------------------------------!
! End of loop for cell faces                                           !
!----------------------------------------------------------------------!

  ENDDO
  RETURN
  END SUBROUTINE RoeNoFix




  SUBROUTINE Muscl(NC,dx,dt,B,Wi,Wf)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Muscl                                                    !
!                                                                      !
! Hankcock's Predictor-Corrector method for intial states       !
!                                                                      !
! Written: 09-07-2004                                                  !
! Changed:                                                             !
!----------------------------------------------------------------------!
!----------------------------------------------------------------------!

!----------------------------------------------------------------------!
! Global variables                                                     !
!----------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!----------------------------------------------------------------------!
! Local variables                                                      !
!----------------------------------------------------------------------!

! Geometry and time
    DOUBLE PRECISION :: dx,dt
    INTEGER :: NC

! Counters
    INTEGER :: i,j

! Boundary conditions
    INTEGER :: B

! Limiter selection
    INTEGER, PARAMETER :: LIM = 1

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)     :: Wi
    DOUBLE PRECISION, DIMENSION(1:3,1:2*(NC+2)) :: Wf
    DOUBLE PRECISION, DIMENSION(1:3)            :: Wp
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)     :: dW
    DOUBLE PRECISION, DIMENSION(1:3,1:3)        :: Aw

!----------------------------------------------------------------------!
! Calculate States at cell walls                                       !
!----------------------------------------------------------------------!

! Calculate the predictor step value using the chosen limiter

    SELECT CASE (LIM)

! 0. First order
    CASE (0)

      DO j = 1,NC
        dW(:,j) = 0.0
      END DO

! 1. Algebraic Average
    CASE (1)

      CALL Average(NC,Wi,dW(:,1:NC))

! 2. Double Minmod
    CASE (2)

      CALL DMinmod(NC,Wi,dW(:,1:NC))

! 3. Superbee
    CASE (3)

      CALL Superbee(NC,Wi,dW(:,1:NC))

! 4. Koren
    CASE (4)

      CALL Koren(NC,Wi,dW(:,1:NC))


    END SELECT


! Calculate precictor step for virtual cells

  SELECT CASE (B)

! Soft boundary
  CASE (1)
    dW(:,0) = 0.0
    dW(:,NC+1) = 0.0
```

```
! Hard boundary
  CASE (2)
    dW(1,0) = -dW(1,1)
    dW(2,0) =  dW(2,1)
    dW(3,0) = -dW(3,1)

    dW(1,NC+1) = -dW(1,NC)
    dW(2,NC+1) =  dW(2,NC)
    dW(3,NC+1) = -dW(3,NC)

  END SELECT


! Calculate the states left and right of cell faces

    DO j = 0,NC+1

! Calculate Matrix Aw
      Aw(1,1) = Wi(2,j)
      Aw(1,2) = Wi(1,j)
      Aw(1,3) = 0.0
      Aw(2,1) = 0.0
      Aw(2,2) = Wi(2,j)
      Aw(2,3) = 1/Wi(1,j)
      Aw(3,1) = 0.0
      Aw(3,2) = gamma*Wi(3,j)
      Aw(3,3) = Wi(2,j)

! Calculate the predictor value
      DO i = 1,3
        Wp(i) = Wi(i,j) - (dt/(2*dx))*(Aw(i,1)*dW(1,j) + Aw(i,2)*dW(2,j) + Aw(i,3)*dW(3,

        Wf(i,2*j+1) = Wp(i) - 0.5*dW(i,j)
        Wf(i,2*j+2) = Wp(i) + 0.5*dW(i,j)
      END DO

    END DO

  RETURN
  END SUBROUTINE Muscl


  SUBROUTINE Average(NC,Wi,dW)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Average                                                  !
!                                                                      !
! Algebraic average limiter for MUSCL scheme                           !
!                                                                      !
! Written: 16-07-2004                                                     !
! Changed:                                                                !
!-------------------------------------------------------------------------!
!-------------------------------------------------------------------------!

!-------------------------------------------------------------------------!
! Global variables                                                        !
!-------------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!-------------------------------------------------------------------------!
! Local variables                                                         !
!-------------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
    INTEGER :: j

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)     :: Wi
    DOUBLE PRECISION, DIMENSION(1:3,1:NC)   :: dW
    DOUBLE PRECISION, DIMENSION(1:3)            :: AVG,dW1,dW2

!-------------------------------------------------------------------------!
! Calculate predictor step                                                !
!-------------------------------------------------------------------------!

! Calculate predictor step for non-virtual cells

  DO j = 1,NC

    dW1(:) = Wi(:,j) - Wi(:,j-1)
    dW2(:) = Wi(:,j+1) - Wi(:,j)
    AVG(:) = (dW1(:) + dW2(:))/2

    dW(:,j) = AVG(:)

  END DO


  RETURN
  END SUBROUTINE Average


  SUBROUTINE DMinmod(NC,Wi,dW)

!-------------------------------------------------------------------------!
```

```
!-------------------------------------------------------------------!
! Subroutine: DMinmod                                               !
!                                                                   !
! Double Minmod limiter for MUSCL scheme                            !
!                                                                   !
! Written: 16-07-2004                                               !
! Changed:                                                          !
!-------------------------------------------------------------------!
!-------------------------------------------------------------------!

!-------------------------------------------------------------------!
! Global variables                                                  !
!-------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!-------------------------------------------------------------------!
! Local variables                                                   !
!-------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
    INTEGER :: i,j

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)   :: Wi
    DOUBLE PRECISION, DIMENSION(1:3,1:NC)     :: dW
    DOUBLE PRECISION, DIMENSION(1:3)          :: AVG,dW1,dW2
    DOUBLE PRECISION, DIMENSION(1:3)          :: AB
    INTEGER, DIMENSION(1)         :: MAB
    INTEGER                       :: K

!-------------------------------------------------------------------!
! Calculate predictor step                                          !
!-------------------------------------------------------------------!

! Calculate predictor step for non-virtual cells

  DO j = 1,NC

    dW1(:) = Wi(:,j) - Wi(:,j-1)
    dW2(:) = Wi(:,j+1) - Wi(:,j)
    AVG(:) = 0.5*(dW1(:) + dW2(:))

    DO i = 1,3

      AB(1) = AVG(i)
      AB(2) = 2*dW1(i)
      AB(3) = 2*dW2(i)

      MAB = minloc(abs(AB))
      K = MAB(1)

      IF (dW1(i)*dW2(i) <= 0.0) THEN

dW(i,j) = 0.0

      ELSE

        dW(i,j) = AB(K)

      END IF
    END DO
  END DO


  RETURN
  END SUBROUTINE DMinmod


  SUBROUTINE Superbee(NC,Wi,dW)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Superbee                                                 !
!                                                                      !
! Superbee limiter for MUSCL scheme                                    !
!                                                                      !
! Written: 16-07-2004                                                  !
! Changed:                                                             !
!----------------------------------------------------------------------!
!----------------------------------------------------------------------!

!----------------------------------------------------------------------!
! Global variables                                                     !
!----------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!----------------------------------------------------------------------!
! Local variables                                                      !
!----------------------------------------------------------------------!


! Geometry
    INTEGER :: NC

! Counters
```

```
    INTEGER :: i,j

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)   :: Wi
    DOUBLE PRECISION, DIMENSION(1:3,1:NC)     :: dW
    DOUBLE PRECISION, DIMENSION(1:3)   :: AVG,dW1,dW2

! Limiter specific
    DOUBLE PRECISION, DIMENSION(1:2)   :: AB,AB2,AB3
    INTEGER, DIMENSION(1)   :: MAB,MAB2,MAB3
    INTEGER                       :: K,K2,K3

!---------------------------------------------------------------------!
! Calculate predictor step                                            !
!---------------------------------------------------------------------!

! Calculate predictor step for non-virtual cells

    DO j = 1,NC

    dW1(:) = Wi(:,j) - Wi(:,j-1)
    dW2(:) = Wi(:,j+1) - Wi(:,j)
    AVG(:) = 0.5*(dW1(:) + dW2(:))

    DO i = 1,3

      AB(1) = dW1(i)
      AB(2) = dW2(i)
      AB2(1) = 2*dW1(i)
      AB2(2) = 2*dW2(i)

      MAB = maxloc(abs(AB))
      MAB2 = minloc(abs(AB2))

      K = MAB(1)
      K2 = MAB2(1)

      AB3(1) = AB(K)
      AB3(2) = AB2(K2)

      MAB3 = minloc(abs(AB3))
      K3 = MAB3(1)

      IF (dW1(i)*dW2(i) <= 0.0) THEN

dW(i,j) = 0.0

      ELSE

        dW(i,j) = AB3(K3)
      END IF
    END DO
  END DO


  RETURN
  END SUBROUTINE Superbee

  SUBROUTINE Koren(NC,Wi,dW)

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Subroutine: Koren                                                    !
!                                                                      !
! Korens limiter for MUSCL scheme                                      !
!                                                                      !
! Written: 20-07-2004                                                  !
! Changed:                                                             !
!----------------------------------------------------------------------!
!----------------------------------------------------------------------!

!----------------------------------------------------------------------!
! Global variables                                                     !
!----------------------------------------------------------------------!

  USE Vars
  IMPLICIT NONE

!----------------------------------------------------------------------!
! Local variables                                                      !
!----------------------------------------------------------------------!

! Geometry
    INTEGER :: NC

! Counters
    INTEGER :: i,j

! State vectors
    DOUBLE PRECISION, DIMENSION(1:3,0:NC+1)   :: Wi
    DOUBLE PRECISION, DIMENSION(1:3,1:NC)     :: dW
    DOUBLE PRECISION, DIMENSION(1:3)   :: AVG,dW1,dW2
    DOUBLE PRECISION, DIMENSION(1:3)   :: AB
    DOUBLE PRECISION, DIMENSION(1)            :: MAB
    INTEGER                        :: K

!----------------------------------------------------------------------!
! Calculate predictor step                                             !
!----------------------------------------------------------------------!

! Calculate predictor step for non-virtual cells
```

```
  DO j = 1,NC

    dW1(:) = Wi(:,j) - Wi(:,j-1)
    dW2(:) = Wi(:,j+1) - Wi(:,j)

    DO i = 1,3

      AB(1) = 2*dW2(i)
      AB(2) = 2*dW2(i)/3 + dW1(i)/3
      AB(3) = 2*dW1(i)

      MAB = minloc(abs(AB))
      K = MAB(1)

      IF (dW1(i)*dW2(i) <= 0.0) THEN

dW(i,j) = 0.0

      ELSE

  dW(i,j) = AB(K)

      END IF
    END DO
  END DO


  RETURN
  END SUBROUTINE Koren


END MODULE Solvers
```

```
MODULE Vars

!----------------------------------------------------------------------!
!----------------------------------------------------------------------!
! Module: Vars                                                         !
!                                                                      !
! Variables for shocktube program                                      !
!                                                                      !
! Written: 06-07-2004                                                  !
! Changed:                                                             !
!----------------------------------------------------------------------!
!----------------------------------------------------------------------!

  IMPLICIT NONE
  PUBLIC

!----------------------------------------------------------------------!
! define the parameters                                                !
!----------------------------------------------------------------------!

! Gamma factors
    DOUBLE PRECISION, PARAMETER :: gamma = 1.4D0
    DOUBLE PRECISION, PARAMETER :: gamma1 = (gamma-1.D0)/(2.D0*gamma)
    DOUBLE PRECISION, PARAMETER :: gamma2 = 1.D0/gamma1
    DOUBLE PRECISION, PARAMETER :: gamma3 = (gamma+1.D0)/(2.D0*gamma)
    DOUBLE PRECISION, PARAMETER :: gamma4 = (gamma+1.D0)/(2.D0*gamma)
    DOUBLE PRECISION, PARAMETER :: gamma5 = (gamma-1.D0)/2.D0
    DOUBLE PRECISION, PARAMETER :: gamma6 = 1.D0/gamma5
    DOUBLE PRECISION, PARAMETER :: gamma7 = (gamma+1.D0)/2.D0
    DOUBLE PRECISION, PARAMETER :: gamma8 = 1.D0/gamma7
    DOUBLE PRECISION, PARAMETER :: gamma9 = (gamma+1.D0)/(gamma-1.D0)
    DOUBLE PRECISION, PARAMETER :: gamma10= 1.D0/gamma9

END MODULE Vars
```