PHYS 555 Project:

Finite-Volume Methods

Michael Kinach

December 11^{th}, 2017

Table of Contents

List of Figures

Outline

In this report, I study high-resolution shock-capturing (HRSC) finite-volume schemes in computational hydrodynamics. Two systems are studied:

1. Burgers' equation in non-conservative and conservative form
2. Relativistic fluids with an ultrarelativistic equation of state in slab symmetry

In Chapter 1, Burgers' equation is studied in detail. Burgers' equation is perhaps the simplest system that exhibits the formation of shocks. As such the usual discretizations based on finite-difference methods will often fail near the discontinuities. One approach to deal with this problem is to use finite-volume techniques. A finite-volume Roe solver is used to compute the evolution of the system. The system is also cast in both conservative and non-conservative forms to observe the breakdown of naive finite-difference discretizations near the discontinuities. The results are compared to the analytic solution.

In Chapter 2, I study the evolution of an ultrarelativistic fluid. The equations of motion of the fluid are also prone to shocks so HRSC methods are required. Building on the code of Chapter 1, a finite-volume Roe solver is implemented for a fluid with an ultrarelativistic equation of state. We perform a convergence test to confirm that the code is second-order accurate. I also investigate the 'shock tube problem' and study the rarefaction waves.

This report is based on Project 2 [1] from the Graduate Summer School on General Relativistic Hydrodynamics that took place from July 21 to August 1, 2003 at the University of British Columbia. I have made an attempt to solve most of the problems from the Project 2 handout. Note that I reference the handout heavily in Section 1.2 and Sections 2.1/2.2 because the mathematics is quite complicated.

Chapter 1

Burgers' Equation

One of the simplest fluid models is the one-dimensional Burgers' equation. The inviscid Burgers' equation is a non-linear scalar equation given by

$$\dot{q} + q\frac{\partial q}{\partial x} = 0. \tag{1.1}$$

This equation is a simplified model of fluid flow and turbulence that is found in the Navier-Stokes equations. More importantly it is a nonlinear equation for which the exact solution is known. Thus it serves as a good benchmark for testing different numerical methods.

1.1 Finite-Difference Method

It is notable that the solution to (1.1) exhibits shocks (i.e. discontinuities) even for smooth initial data. The computation of fluid flow containing shocks is very difficult because these flows contain sharp, discontinuous changes in the state variables. In this section we will show that naïve discretizations based on basic finite-difference approximations will usually fail.

1.1.1 Upwind Non-Conservative Scheme

Due to the simplicity of (1.1), at first glance one might try to solve the equation using an upwind finite-difference discretization. This leads to

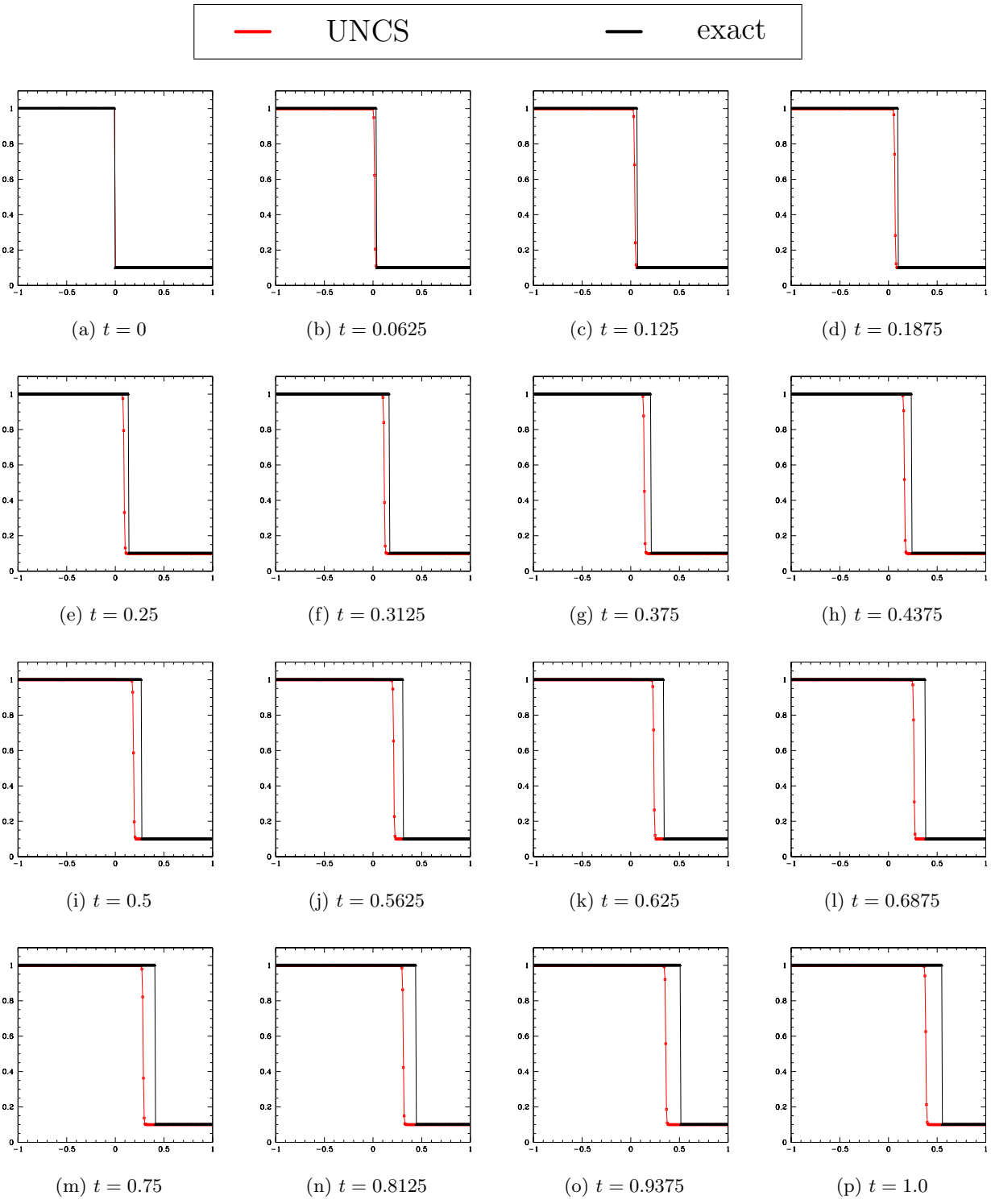
$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x} q_i^n \left(q_{i+1}^n - q_i^n\right) \tag{1.2}$$

We will call (1.2) the upwind non-conservative scheme (UNCS). Unfortunately the evolution of the solution using UNCS gives an erroneous shock speed. This is illustrated in Figure 1.1 where we have used piecewise initial data of the general form

$$q(x,0) = \begin{cases} q_L & \text{if } x < 0 \\ q_R & \text{if } x > 0 \end{cases} \tag{1.3}$$

For case where $q_L > q_R$, the exact solution is simply

$$q(x,t) = \begin{cases} q_L & \text{if } x < st \\ q_R & \text{if } x > st \end{cases} \tag{1.4}$$



(a) $t = 0$ (b) $t = 0.0625$ (c) $t = 0.125$ (d) $t = 0.1875$

(e) $t = 0.25$ (f) $t = 0.3125$ (g) $t = 0.375$ (h) $t = 0.4375$

(i) $t = 0.5$ (j) $t = 0.5625$ (k) $t = 0.625$ (l) $t = 0.6875$

(m) $t = 0.75$ (n) $t = 0.8125$ (o) $t = 0.9375$ (p) $t = 1.0$

Figure 1.1: Numerical solution of Burgers' equation using the Upwind Non-Conservative Scheme (UNCS) with initial data $q_L = 1$, $q_R = 0.1$. The UNCS gives a shock speed ($s \approx 0.38$) that is slower than the shock speed for the exact solution ($s = 0.55$). This error remains even in the continuum limit.

where

$$s = \frac{f(q_L) - f(q_R)}{q_L - q_R} \tag{1.5}$$

is the shock speed. Note that this scheme has first-order spatial truncation error.

1.1.2 Upwind Conservative Scheme

The results of the simple experiment in Figure 1.1 illustrate that naïve discretizations can give incorrect results. These errors can be remedied by casting (1.1) in conservative form. Conservative form refers to a form of a hyperbolic system that resembles a continuity equation. The general structure is

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{q})}{\partial x} = \boldsymbol{\psi} \tag{1.6}$$

where $\mathbf{q}$ is a vector of conservative variables (i.e. fluid variables), $\mathbf{f}(\mathbf{q})$ is a vector of fluxes that depend on these variables, and $\boldsymbol{\psi}$ is a source term. For the specific case of (1.1) we can recognize

$$\mathbf{q} = q \tag{1.7}$$

$$\mathbf{f} = \frac{1}{2}q^2 \tag{1.8}$$

$$\boldsymbol{\psi} = 0. \tag{1.9}$$

The conservative form of (1.1) is therefore

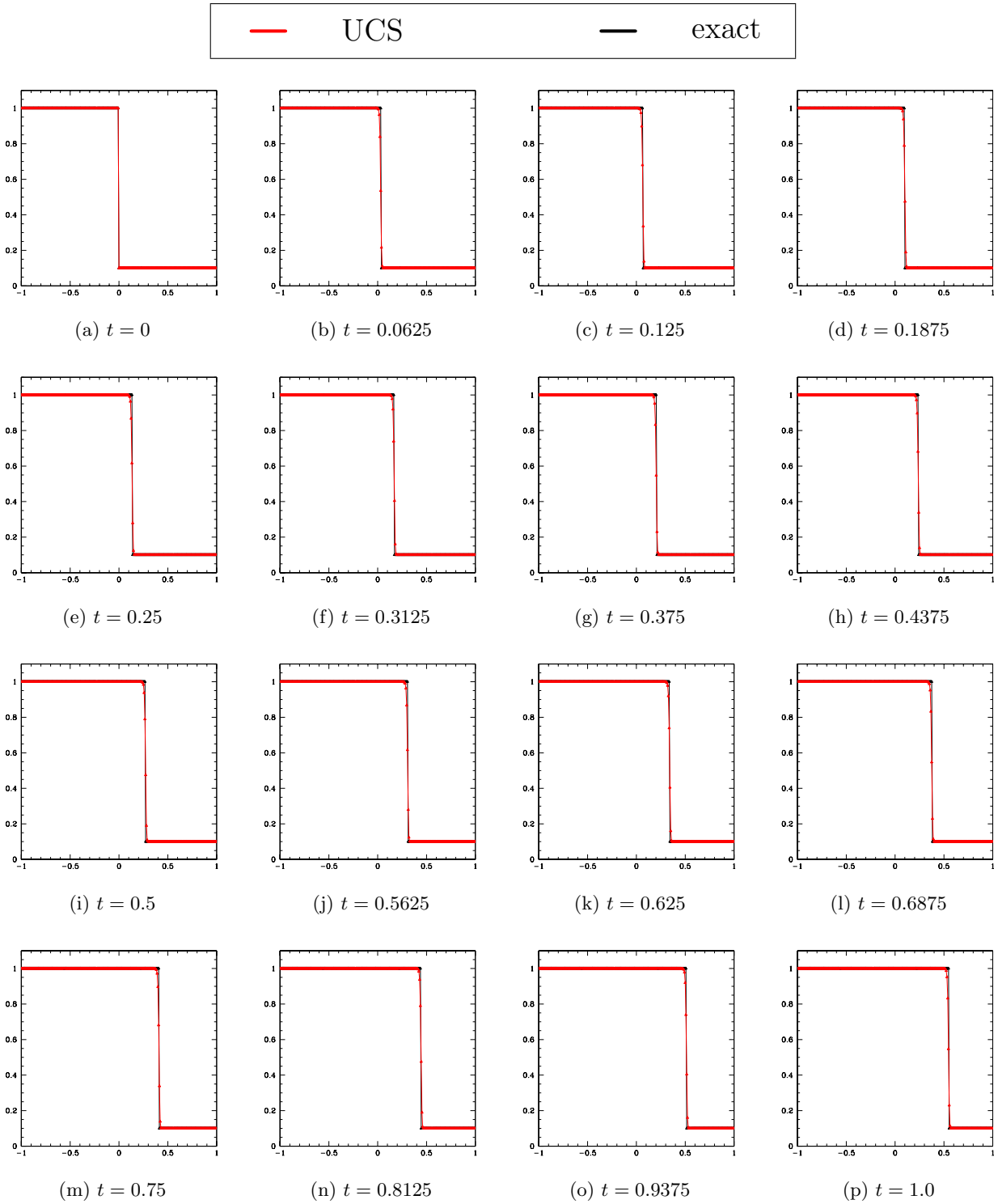
$$\dot{q} + \left(\frac{1}{2}q^2\right)' = 0. \tag{1.10}$$

The motivation for writing Burgers' equation in conservation form is subtle. It turns out that conservative methods will yield more accurate results for systems that exhibit shocks and non-conservative methods are generally restricted to systems where smooth solutions are expected.

Although (1.1) and (1.10) are mathematically equivalent they will yield different finite-difference discretizations. The upwind discretization of (1.10) is

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x}\left(\frac{1}{2}\left(q_{i+1}^n\right)^2 - \frac{1}{2}\left(q_i^n\right)^2\right) \tag{1.11}$$

We call this the Upwind Conservative Scheme (UCS). Note that this scheme also has first-order spatial truncation error. The computed solution is shown in Figure 1.2. Unlike the UNCS, the UCS gives proper shock velocities in the continuum limit and seems to be a very good approximation to the exact solution. This illustrates the power of writing the nonlinear equation in conservative form.



(a) $t = 0$ (b) $t = 0.0625$ (c) $t = 0.125$ (d) $t = 0.1875$

(e) $t = 0.25$ (f) $t = 0.3125$ (g) $t = 0.375$ (h) $t = 0.4375$

(i) $t = 0.5$ (j) $t = 0.5625$ (k) $t = 0.625$ (l) $t = 0.6875$

(m) $t = 0.75$ (n) $t = 0.8125$ (o) $t = 0.9375$ (p) $t = 1.0$

Figure 1.2: Numerical solution of Burgers' equation using the Upwind Conservative Scheme (UCS) with initial data $q_L = 1$, $q_R = 0.1$. The solution closely approximates the exact solution except for some rounding of the sharp corners. The shock speed is corrected when using the UCS.

1.1.3 Second-Order Centred Scheme

As a final experiment we will consider a second-order discretization of (1.1). For example, a second-order centred difference:

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{4\Delta x}\left(3\left(q_i^n\right)^2 - 4\left(q_{i-1}^n\right)^2 + \left(q_{i-2}^n\right)^2\right) \tag{1.12}$$

We observe that the computed solution has the correct shock speed. However it also develops oscillatory behaviour around the base of the shock. This shows that it is not well suited to our problem.

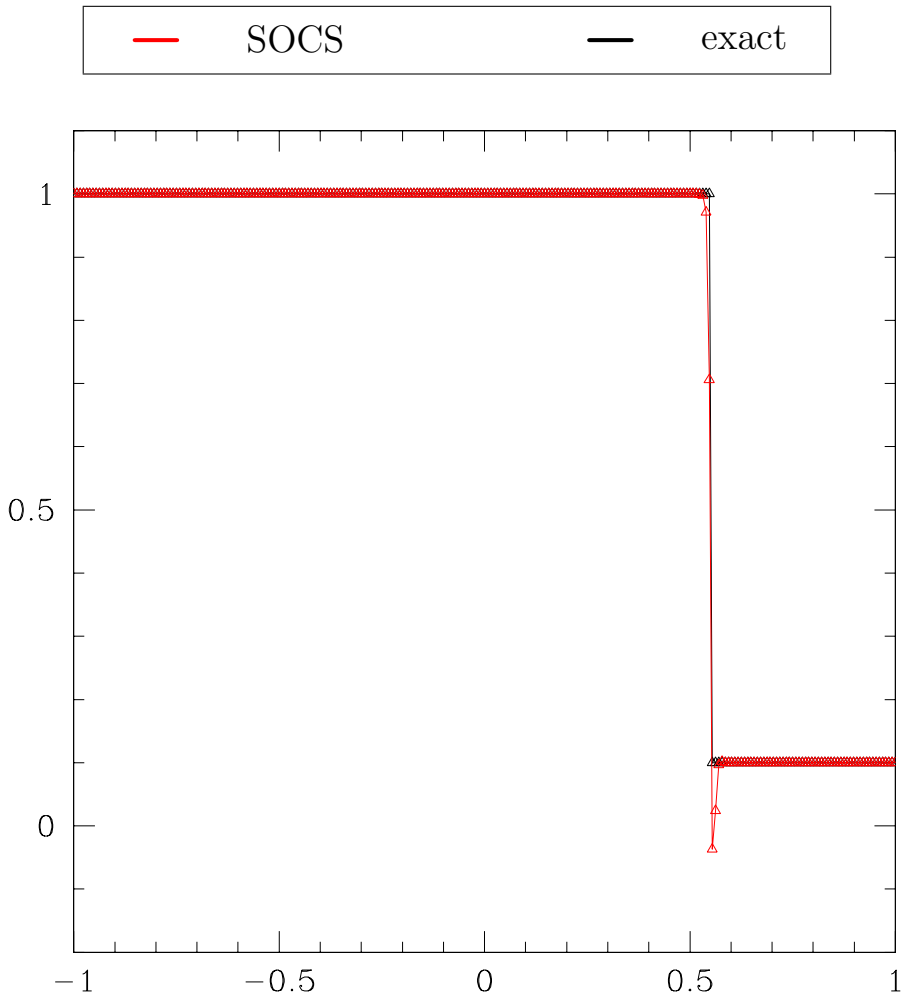
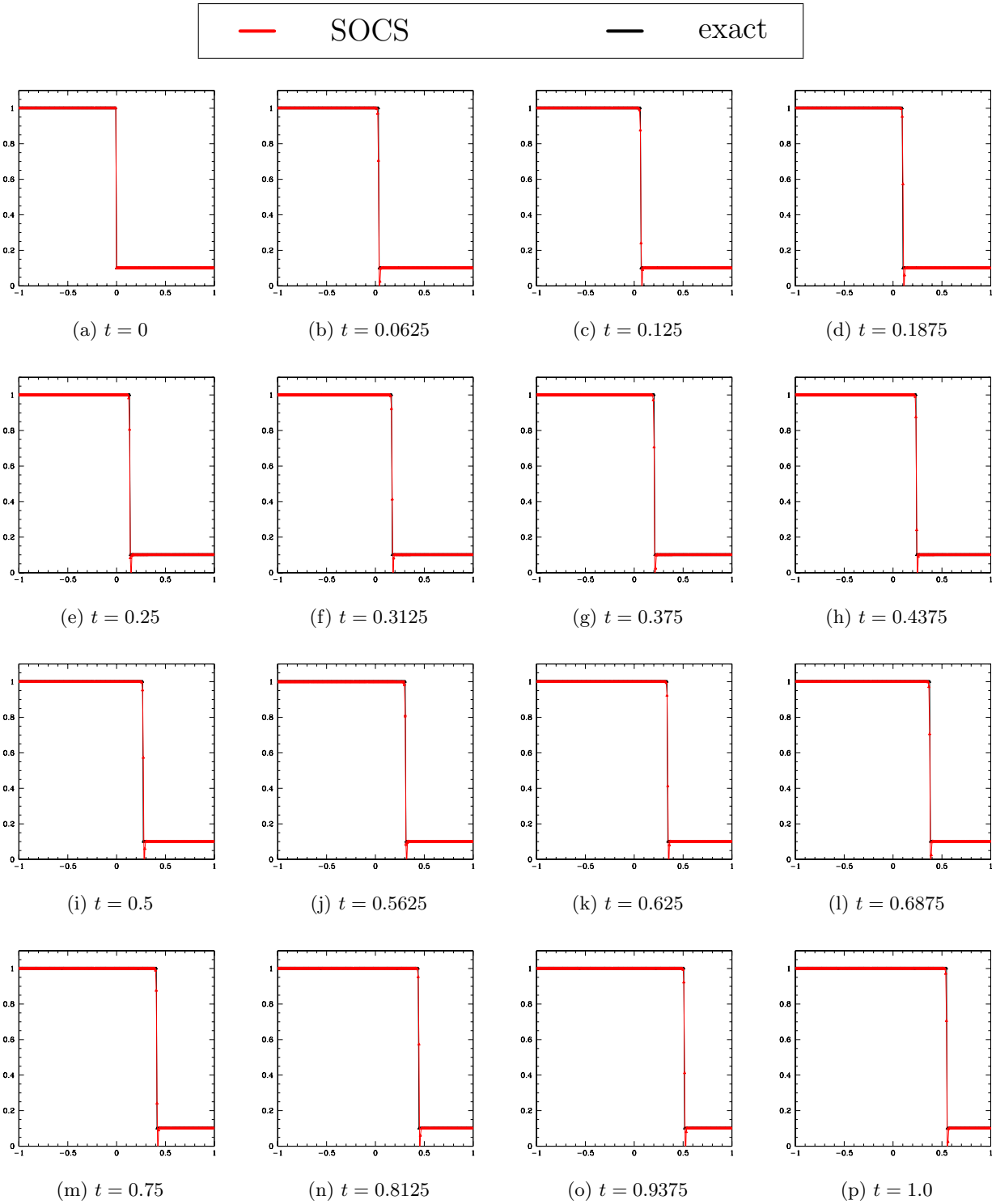


Figure 1.3: Snapshot of the solution to Burgers' equation using the Second-Order Centred Scheme (SOCS) at $t = 1$ with initial data $q_L = 1$, $q_R = 0.1$. The solution has the correct shock speed but oscillates near the base of the shock. The full evolution is given in Figure 1.4.



(a) $t = 0$ (b) $t = 0.0625$ (c) $t = 0.125$ (d) $t = 0.1875$

(e) $t = 0.25$ (f) $t = 0.3125$ (g) $t = 0.375$ (h) $t = 0.4375$

(i) $t = 0.5$ (j) $t = 0.5625$ (k) $t = 0.625$ (l) $t = 0.6875$

(m) $t = 0.75$ (n) $t = 0.8125$ (o) $t = 0.9375$ (p) $t = 1.0$

Figure 1.4: Numerical solution of Burgers' equation using the Second-Order Centred Scheme (SOCS) with initial data $q_L = 1$, $q_R = 0.1$. The evolution has the correct shock speed but oscillates near the base of the shock.

1.2 Finite-Volume Method

As we saw in the previous section, using basic finite-difference discretizations gives incorrect results for shock evolution. There are several different ways to fix this problem. One approach is the finite-volume method.

1.2.1 Theory

The basic idea is to define a series of cells on a grid. Each cell has a finite volume of $V_{C_i^{n+1/2}} = \Delta x \Delta t$. As time evolves, the fluid quantities will be conserved as they travel between adjacent cells. We can therefore calculate the solution based on averages within each cell. Specifically, we take the average of the conservative equation (1.6) over a cell $C_i^{n+1/2}$:

$$\frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \frac{\partial \mathbf{q}}{\partial t} + \frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \frac{\partial \mathbf{f}}{\partial x} = \frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \boldsymbol{\psi}, \tag{1.13}$$

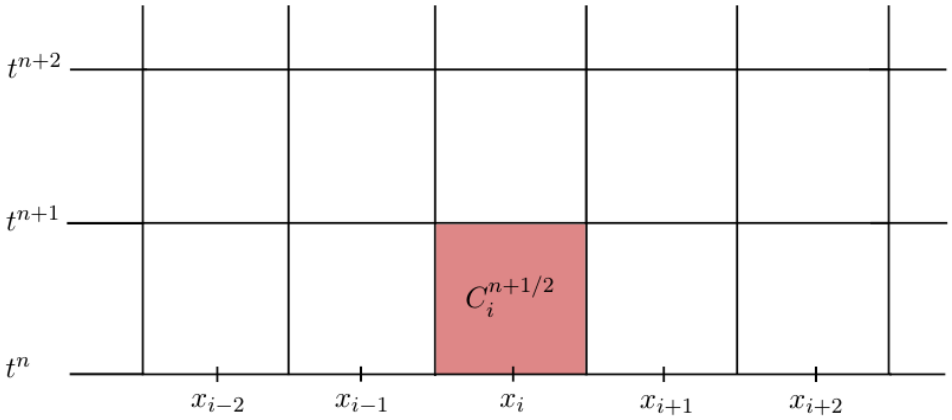


Figure 1.5: The basic cell structure for a finite-volume discretization. We define the cells $C_i^{n+1/2} = (t^n, t^{n+1/2}) \times (x_{i-1/2}, x_{i+1/2})$ to be centred at $(t^{n+1/2}, x_i)$. The red cell represents just one such cell on the grid with a volume $V_{C_i^{n+1/2}}$.

Plugging in the bounds of integration using $C_i^{n+1/2} = (t^n, t^{n+1/2}) \times (x_{i-1/2}, x_{i+1/2})$ and replacing $V_{C_i^{n+1/2}} = \Delta t \Delta x$ we arrive at

$$\frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \frac{\partial \mathbf{q}}{\partial t} dx dt + \frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \frac{\partial \mathbf{f}}{\partial x} dx dt = \frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \boldsymbol{\psi} dx dt. \tag{1.14}$$

We note that this equation involves the integral of a vector field over a volume. Thus we can

invoke Gauss' theorem to simplify the expression:

$$\frac{\bar{\mathbf{q}}_i^{n+1} - \bar{\mathbf{q}}_i^n}{\Delta t} + \frac{\mathbf{F}_{i+1/2}^{n+1/2} - \mathbf{F}_{i-1/2}^{n+1/2}}{\Delta x} = \widehat{\boldsymbol{\psi}}_i^{n+1/2}. \tag{1.15}$$

Note that we have defined

$$\bar{\mathbf{q}}_i^n \equiv \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{q}(t^n, x) dx \tag{1.16}$$

$$\mathbf{F}_{i+1/2}^{n+1/2} \equiv \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} \mathbf{f}\left(\mathbf{q}(t, x_{i+1/2})\right) dt \tag{1.17}$$

$$\widehat{\boldsymbol{\psi}}_i^{n+1/2} \equiv \frac{1}{\Delta x \Delta t} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \boldsymbol{\psi}(t, x)\, dx\, dt \tag{1.18}$$

We will call $\bar{\mathbf{q}}_i^n$ the spatial averages of the conservative variables, $\mathbf{F}_{i+1/2}^{n+1/2}$ the temporal averages of the fluxes of the conservative variables (the so-called numerical flux), and $\widehat{\boldsymbol{\psi}}_i^{n+1/2}$ the source averages. We can then, in principle, calculate $\bar{\mathbf{q}}_i^{n+1}$ assuming that $\bar{\mathbf{q}}_i^n$, $\mathbf{F}_{i\pm1/2}^{n+1/2}$ and $\widehat{\boldsymbol{\psi}}_i^{n+1/2}$ are known. However, there are still technical difficulties in calculating the numerical fluxes because they are averages in time. It is also problematic that the fluid quantities $\bar{\mathbf{q}}_i^n$ and $\bar{\mathbf{q}}_{i+1}^n$ on the left and right side of each cell will be discontinuous.

The way to overcome these difficulties is to use a Godunov scheme to calculate $\mathbf{F}_{i+1/2}^{n+1/2}$. A detailed description of the Godunov scheme is given in [2]. The basic idea of the Godunov scheme is to specify the initial data $\mathbf{q}_i^n$ to be a piecewise constant function over the grid of cells:

$$\mathbf{q}_i^n \approx \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{q}(t^n, x)\, dx \tag{1.19}$$

The initial data then looks as in Figure 1.6. Notice that the initial data resembles a Riemann problem at each cell interface. In general, a Riemann problem consists of a conservation law (1.6) together with the initial data (1.3). If we can solve this Riemann problem at every cell interface then we will have the numerical fluxes.

It turns out that solving a full Riemann problem for every cell boundary is not very efficient. Instead we will use an approximate Riemann solver called the Roe solver [3]. The basic idea is to linearize the $\mathbf{f}(\mathbf{q})$ terms in (1.6) and assume that $\boldsymbol{\psi} = 0$. The linearization can be done by assuming $\partial\mathbf{f}/\partial\mathbf{q}$ has constant coefficients so that a solution can be obtained by diagonalizing the Jacobian [1]. The Roe numerical flux $\mathbf{F}_{i+1/2}^{\text{Roe}}$ can then be expressed as a function of the solution variables. The fluxes are found to be

$$\mathbf{F}_{i+1/2}^{\text{Roe}} = \frac{1}{2}\left[\mathbf{f}\left(\tilde{\mathbf{p}}_{i+1/2}^R\right) + \mathbf{f}\left(\tilde{\mathbf{p}}_{i+1/2}^L\right) - \sum_\alpha |\lambda_\alpha| \omega_\alpha \mathbf{r}_\alpha\right]. \tag{1.20}$$

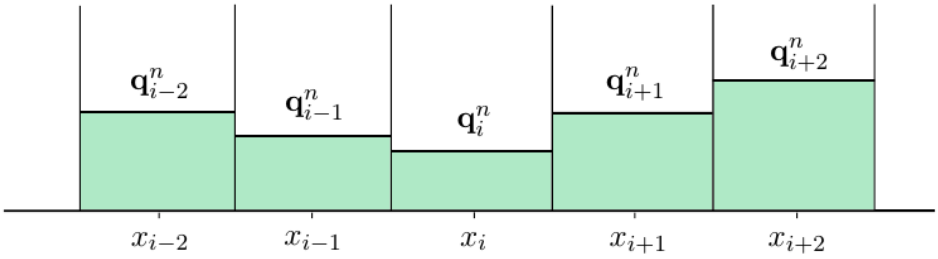


Figure 1.6: Initial data for the Godunov scheme. We define a piecewise constant function to set up a Riemann problem at each cell interface. This allows us to approximately compute the numerical fluxes $\mathbf{F}^{n+1/2}_{i+1/2}$.

The quantities $(\tilde{\mathbf{p}}^R_{i+1/2}, \tilde{\mathbf{p}}^L_{i+1/2})$ are the values of the original variables $\mathbf{q}(t^n, x_i)$ at the boundary $x_{i+1/2}$ which can be calculated via interpolation. We will call $(\tilde{\mathbf{p}}^R, \tilde{\mathbf{p}}^L)$ the reconstructed variables and define

$$\tilde{\mathbf{p}}^L_{i+1/2} = \bar{\mathbf{p}}_i + \boldsymbol{\sigma}_i \left(x_{i+1/2} - x_i\right), \tag{1.21}$$

$$\tilde{\mathbf{p}}^R_{i+1/2} = \bar{\mathbf{p}}_{i+1} + \boldsymbol{\sigma}_{i+1} \left(x_{i+1/2} - x_{i+1}\right), \tag{1.22}$$

where $\boldsymbol{\sigma}_i$ is defined as

$$\boldsymbol{\sigma}_i = \text{minmod}\left(\mathbf{s}_{i-1/2}, \mathbf{s}_{i+1/2}\right) \tag{1.23}$$

with

$$\text{minmod}(a,b) = \begin{cases} 0 & \text{if} \quad ab < 0 \\ a & \text{if} \quad |a| < |b| \ \text{ and } \ ab > 0 \\ b & \text{if} \quad |a| > |b| \ \text{ and } \ ab > 0. \end{cases} \tag{1.24}$$

The function $\boldsymbol{\sigma}_i$ is called a slope limiter. The slope limiter is used to reduce spurious oscillations that would otherwise arise at the discontinuities. There are many choices for slope limiters and here we will use the minmod limiter suggested in [1]. Also note that we have defined

$$\mathbf{s}_{i+1/2} = \frac{\bar{\mathbf{p}}_{i+1} - \bar{\mathbf{p}}_i}{x_{i+1} - x_i}, \tag{1.25}$$

and λ_α, ω_α and $\mathbf{r}_\alpha$ in (1.20) are characteristics of the Jacobian matrix

$$\mathbf{A}|_{i+1/2} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right|_{\mathbf{q}=1/2\,(\tilde{\mathbf{q}}^L_{i+1/2}+\tilde{\mathbf{q}}^R_{i+1/2})}. \tag{1.26}$$

Specifically, λ_α are the eigenvalues of $\mathbf{A}$, $\mathbf{r}_\alpha$ are eigenvectors associated with the eigenvalues λ_α and ω_α are the jumps in the original variables $(\tilde{\mathbf{q}}^R, \tilde{\mathbf{q}}^L)$ that have been calculated from the

reconstructed variables ($\tilde{\mathbf{p}}^R$, $\tilde{\mathbf{p}}^L$) according to

$$\tilde{\mathbf{q}}^R_{i+1/2} - \tilde{\mathbf{q}}^L_{i+1/2} = \sum_\alpha \omega_\alpha \mathbf{r}_\alpha. \tag{1.27}$$

The final step of the Roe solver is to update $\bar{\mathbf{q}}^n_i$. We use a second-order Runge-Kutta method to advance the solution in time:

$$\bar{\mathbf{q}}^{n+1/2} = \bar{\mathbf{q}}^n + \frac{\Delta t}{2} L(\bar{\mathbf{q}}^n) \tag{1.28}$$

$$\bar{\mathbf{q}}^{n+1} = \bar{\mathbf{q}}^n + \Delta t L(\bar{\mathbf{q}}^{n+1/2}) \,. \tag{1.29}$$

where

$$L(\bar{\mathbf{q}}^n) = -\frac{\mathbf{F}^{Roe}_{i+1/2}(\bar{\mathbf{q}}^n) - \mathbf{F}^{Roe}_{i-1/2}(\bar{\mathbf{q}}^n)}{\Delta x} + \widehat{\boldsymbol{\psi}}_i(\bar{\mathbf{q}}^n) \tag{1.30}$$

as suggested in [1].

1.2.2 Implementation

Now that we have developed the general method we can implement the finite-volume method for Burgers' equation (1.10). The finite-volume discretization of Burgers' equation is

$$\frac{\bar{q}^{n+1}_i - \bar{q}^n_i}{\Delta t} + \frac{F^{n+1/2}_{i+1/2} - F^{n+1/2}_{i-1/2}}{\Delta x} = 0 \,. \tag{1.31}$$

(a) Boundary Conditions

We use the suggestion of [1] and use ghost cells to impose boundary conditions. A ghost cell is a cell that lies next to the boundaries. These cells are not updated according to the equations of motion but are instead set according to our choice of boundary conditions. In this case we will use so-called "outflow boundary conditions" which is a first-order approximation to outgoing boundary conditions.

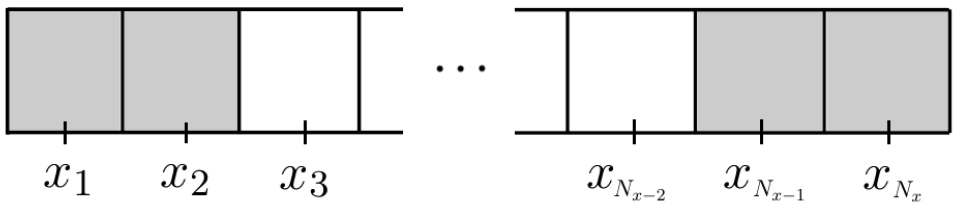


Figure 1.7: An illustration of ghost cells for outflow boundary conditions. Our grid contains N_x cells. There are two ghost cells (gray) per boundary. In these cells the dynamical variables are updated according the outgoing boundary conditions we impose.

If we have a grid of N_x cells, the outgoing boundary conditions are given by

$$q_1 = q_2 \tag{1.32}$$
$$q_2 = q_3 \tag{1.33}$$
$$q_{N_x} = q_{N_{x-1}} \tag{1.34}$$
$$q_{N_{x-1}} = q_{N_{x-2}} \tag{1.35}$$

In effect we set the ghost cell values to the value of the closest regular cell.

(b) Roe Numerical Flux

We are interested in computing the Roe numerical flux according to (1.20) so we need the Jacobian matrix **A**. Since Burgers' equation is a scalar the Jacobian is also a scalar with A = $\lambda = \frac{1}{2}\left(q^L + q^R\right)$ and r = 1. The jump quantity ω is simply $\omega = q^R - q^L$. The Roe numerical flux then simplfies to

$$F^{\text{Roe}}_{i+1/2} = \frac{1}{2}\left[f\left(q^L\right) + f\left(q^R\right) - |\lambda|\, r\, \omega\right]_{i+1/2} \tag{1.36}$$

(c) Second-Order Runge-Kutta Scheme

With the Roe numerical fluxes we can update the solution to a future time step. The procedure is

1. Calculate numerical fluxes at each cell boundary
2. Update the variables with a half time step using (1.36)
3. Use the quantities at the half time step to compute the new numerical fluxes
4. Do a full time step using the numerical fluxes from step 3

This describes the use of the Roe solver in a second-order Runge-Kutta scheme. This method is second-order except near the shocks where the slope limiter will generally reduce the accuracy to first order.

1.2.3 Results

The finite-volume method is implemented in RNPL and Fortran. We will first analyze the solution for piecewise initial data (1.3). We will then move on to Gaussian initial data.

Rankine-Hugoniot Jump Condition

The Rankine-Hugoniot jump condition describes the relationship between the state variables on both sides of a shock in one-dimensional fluid flow [2]. The condition is given by

$$s = \frac{f(q_L) - f(q_R)}{q_L - q_R} = \frac{[F]}{[q]} \tag{1.37}$$

where s is the speed of the shock and $[F]$ and $[q]$ are jumps in the physical flux and the fluid variable, respectively. We investigate this condition for the piecewise initial data with $0.5 \leq q_L \leq 1.0$ and $q_R = 0.1$. The result is given in Figure 1.8.

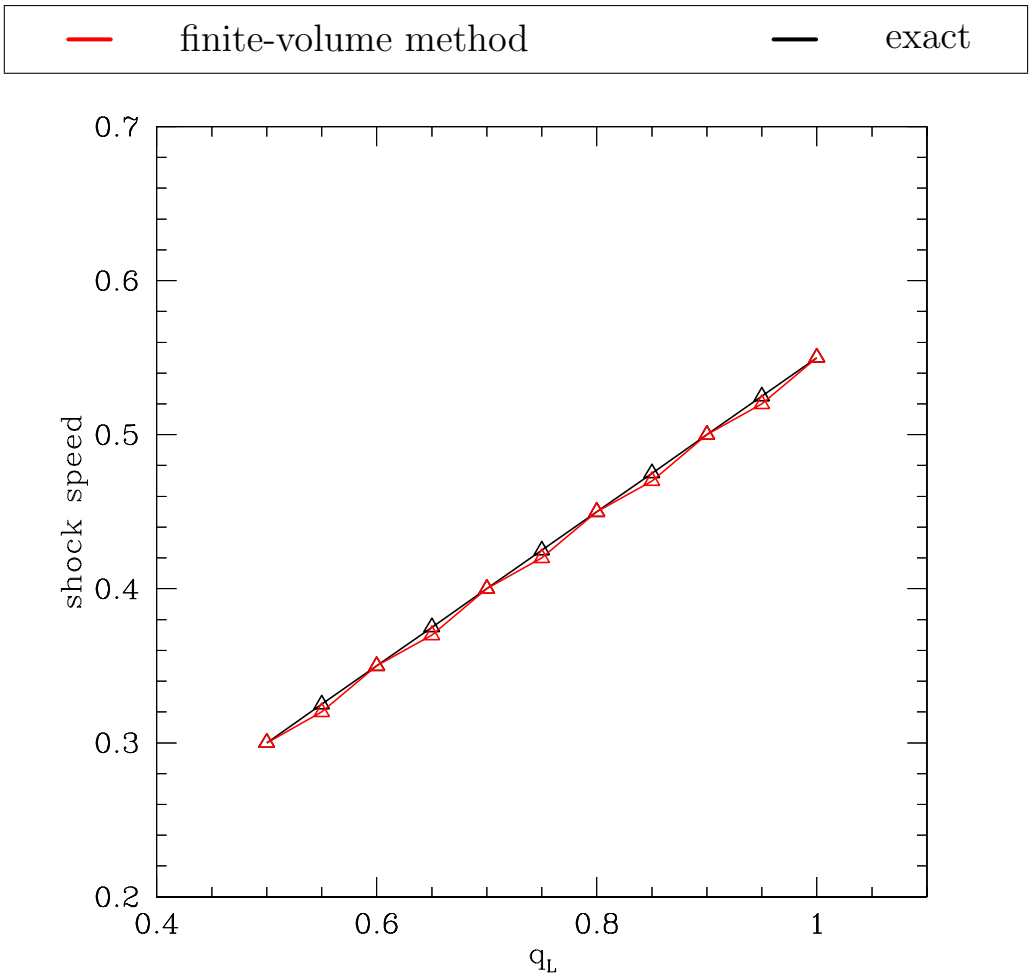


Figure 1.8: Rankine-Hugoniot jump condition for Burgers' equation with the piecewise initial data (1.3). The computed shock speed closely resembles the exact shock speed $(q_L + q_R)/2$. The slight deviation is due to the slope limiter σ_i; this smooths out the shock so that the speed is not precisely defined.

Rarefaction Wave

If we change the initial conditions so that $q_L < q_R$ then we observe the formation of a rarefaction wave. Rarefaction waves are common in fluid dynamics when the fluid variables have a small gradient. This corresponds to a gradual decrease in density or pressure, for example. For the case of Burgers' equation with piecewise initial data the exact solution is

$$q(t,x) = \begin{cases} q_L & x < q_L\, t \\ x/t & q_L\, t < x < q_R\, t \\ q_R & x > q_R\, t \end{cases} \tag{1.38}$$

Let's choose $q_L = 0.2$ and $q_R = 0.7$. Then the solution becomes

$$q(t,x) = \begin{cases} 0.2 & x < 0.2\, t \\ x/t & 0.2\, t < x < 0.7\, t \\ 0.7 & x > 0.7\, t \end{cases} \tag{1.39}$$

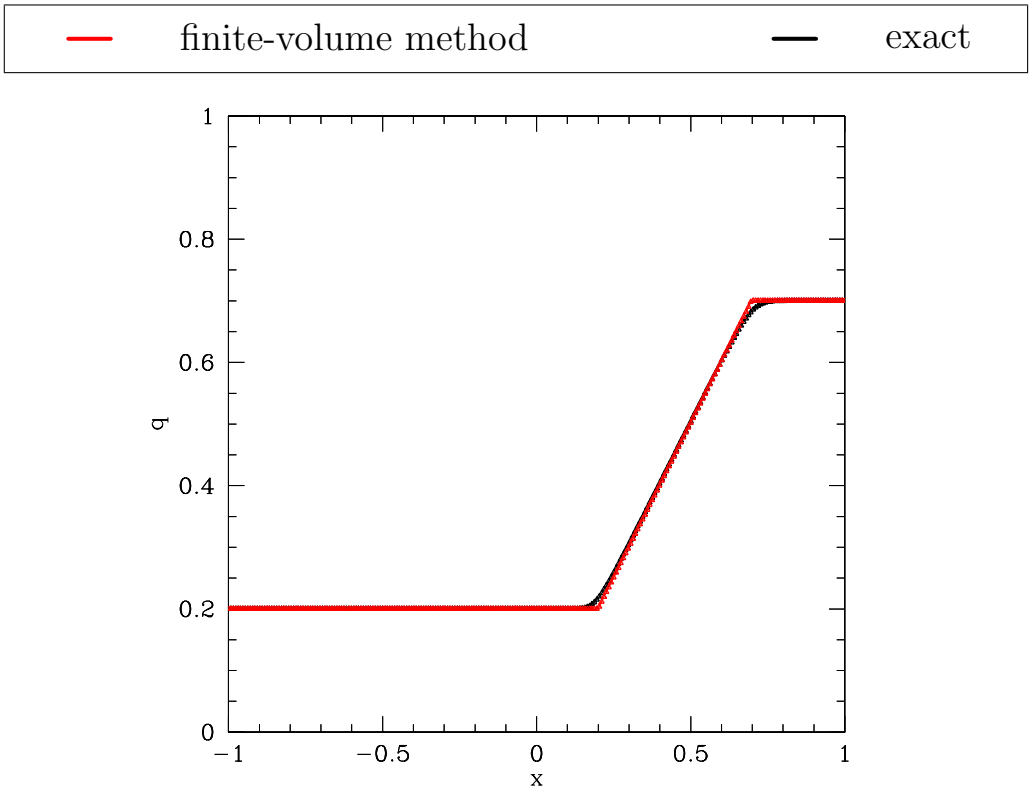
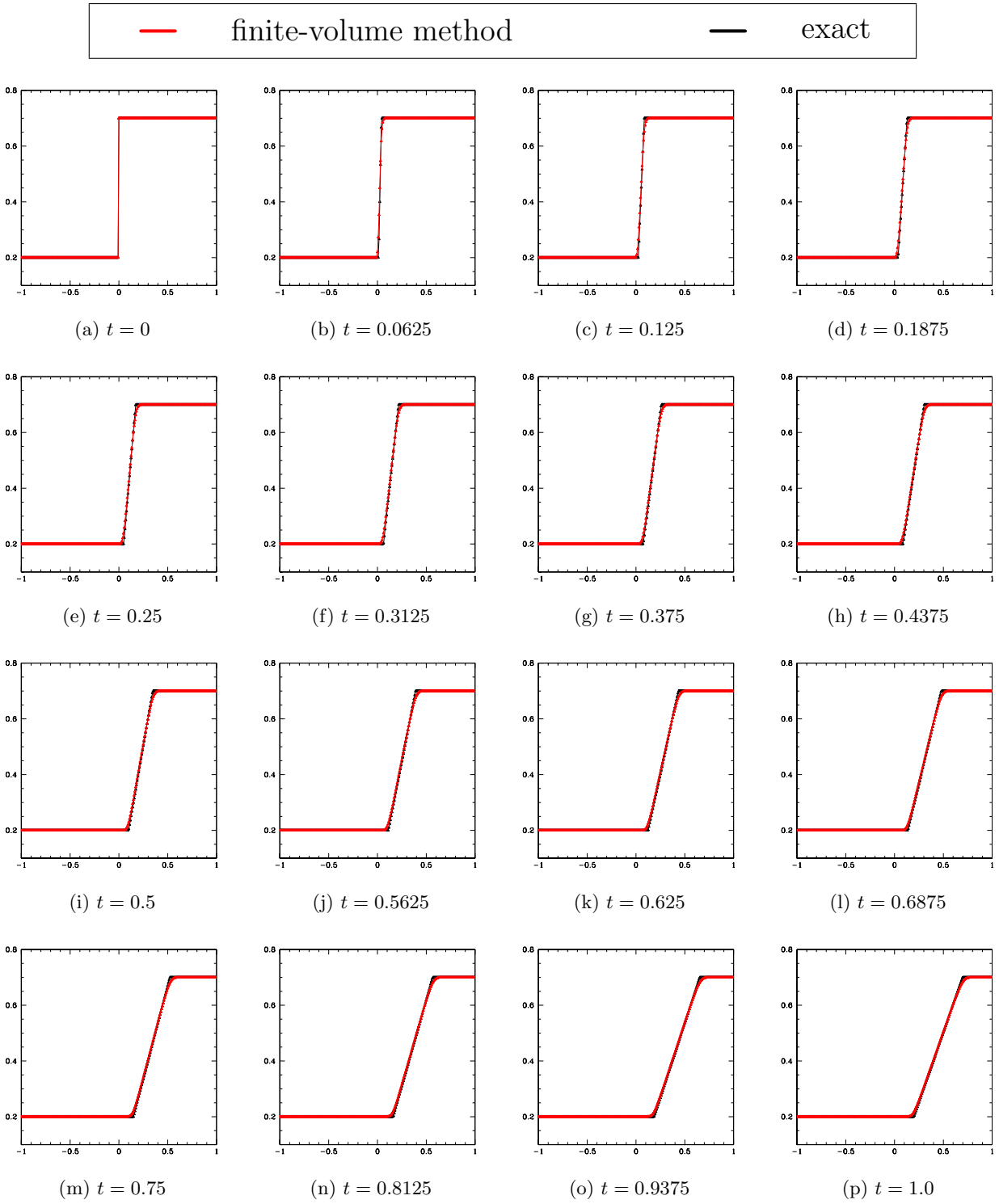


Figure 1.9: Formation of a rarefaction wave when $q_L < q_R$. Here we have taken a snapshot at $t = 1$ for $q_L = 0.2$, $q_R = 0.7$. The computed solution is in excellent agreement with the exact solution (1.39).



(a) $t = 0$ (b) $t = 0.0625$ (c) $t = 0.125$ (d) $t = 0.1875$

(e) $t = 0.25$ (f) $t = 0.3125$ (g) $t = 0.375$ (h) $t = 0.4375$

(i) $t = 0.5$ (j) $t = 0.5625$ (k) $t = 0.625$ (l) $t = 0.6875$

(m) $t = 0.75$ (n) $t = 0.8125$ (o) $t = 0.9375$ (p) $t = 1.0$

Figure 1.10: Rarefaction solution of Burgers' equation using $q_L = 0.2$, $q_R = 0.7$. The discontinuity in the initial data evolves into a continuous solution. The numerical solution using the finite-volume method closely models the exact solution (1.39).

Evolution for Gaussian Initial Data

We now consider Gaussian initial data of the form

$$q(0,x) = A\exp\left(-(x-x_0)^2/\Delta^2\right). \tag{1.40}$$

It is interesting that this system develops shocks even from smooth initial data. This is apparent in Figure 1.12. It can be shown from the characteristics of the equation [2] that the time it takes for the shock to form is

$$T = \frac{-1}{\min\{\partial_x q(0,x)\}} \tag{1.41}$$

For Gaussian initial data

$$\partial_x q(0,x) = \frac{-2A(x-x_0)}{\Delta^2}\exp\left(-(x-x_0)^2/\Delta^2\right) \tag{1.42}$$

and for the choices $A=1$, $x_0=0$, $\Delta=0.1$ the time of shock formation is $T\approx 0.118$. We observe that a discontinuity forms in the solution at approximately $t=0.125$ in Figure 1.12. This helps to confirm that our results match the analytic predictions.

We must also make sure that the numerical solution converges for this initial data because there is no exact solution to compare with. We expect our numerical solution to be of the form

$$\mathrm{q}_{numerical}(t) = \mathrm{q}_{true}(t) + h^n\epsilon(t) \tag{1.43}$$

where $\epsilon(t)$ is an error term due to the truncation error of our method. We can confirm that we have converged to the true solution by using different step sizes:

$$\mathrm{q}_h(t) = \mathrm{q}(t) + h^n\epsilon(t) \tag{1.44}$$

$$\mathrm{q}_{h/2}(t) = \mathrm{q}(t) + \left(\frac{h}{2}\right)^n\epsilon(t) \tag{1.45}$$

$$\mathrm{q}_{h/4}(t) = \mathrm{q}(t) + \left(\frac{h}{4}\right)^n\epsilon(t) \tag{1.46}$$

It follows that

$$\frac{\mathrm{q}_h(t) - \mathrm{q}_{h/2}(t)}{\mathrm{q}_{h/2}(t) - \mathrm{q}_{h/4}(t)} = \frac{h^n\epsilon(t) - \left(\frac{h}{2}\right)^n\epsilon(t)}{\left(\frac{h}{2}\right)^n\epsilon(t) - \left(\frac{h}{4}\right)^n\epsilon(t)} = 2^n \tag{1.47}$$

This means that if we expect convergence of $O(h)$ then each halving of the step size will yield a decrease in the difference by $2^1 = 2$. This is precisely what we observe near the vicinity of the shock in Figure 1.11. Away from the shock the convergence is at least $O(h^2)$.

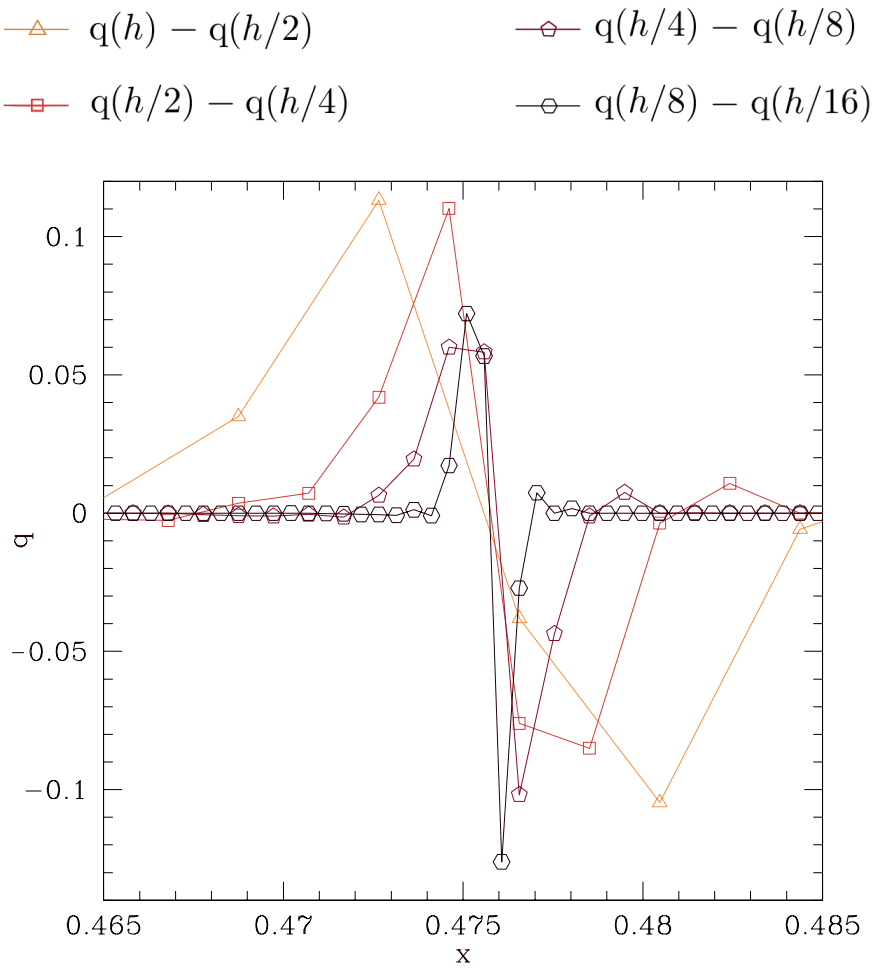
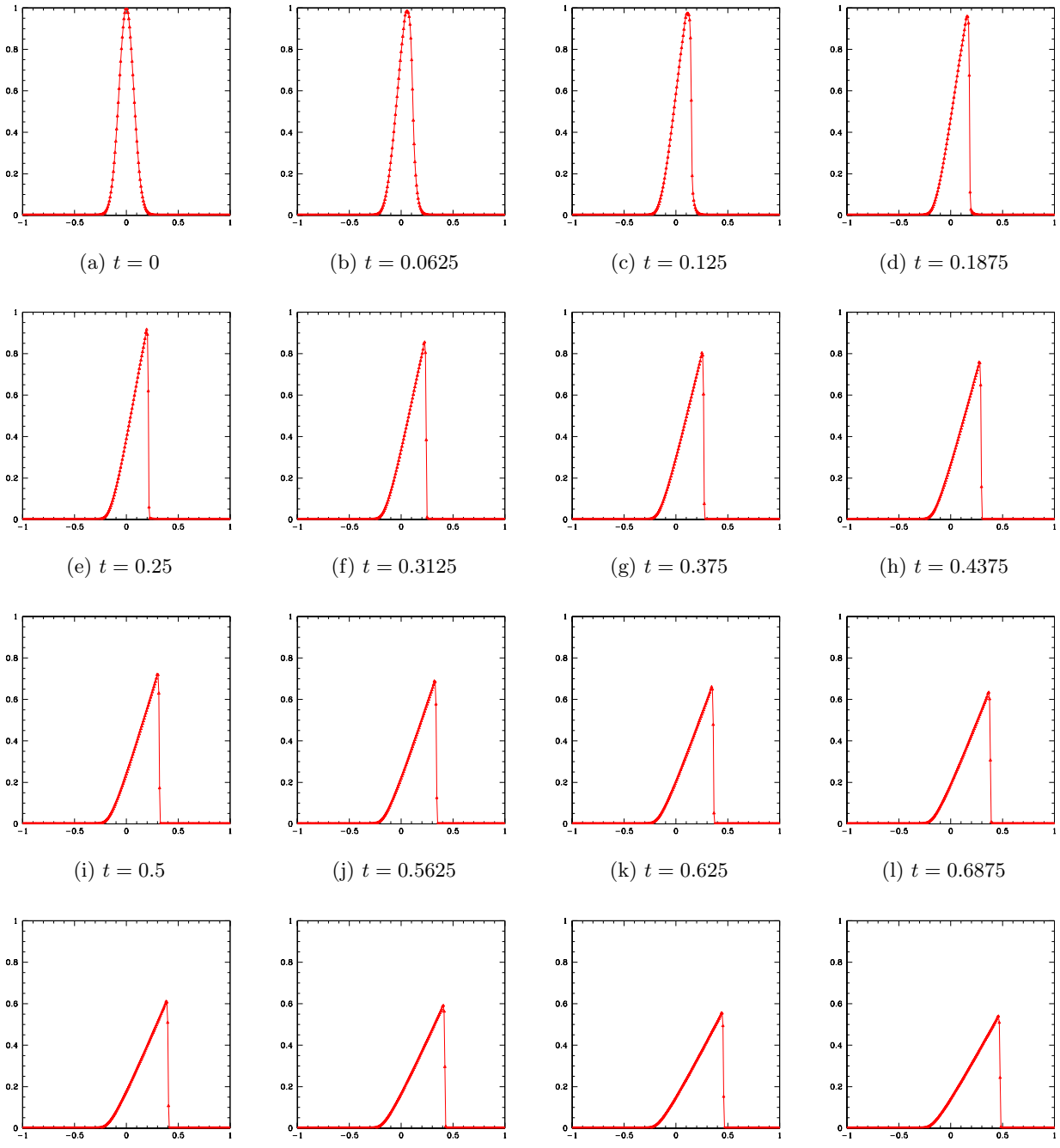


Figure 1.11: Convergence test for Burgers' equation with Gaussian initial data (1.40) with $A = 1$, $x_0 = 0$, $\Delta = 0.1$. This snapshot is taken in the vicinity of the shock at $t = 1$. We observe that the convergence is $O(h^2)$ everywhere except near the shock. Near the shock we have convergence of $O(h)$ as depicted in the figure.



(a) $t = 0$ (b) $t = 0.0625$ (c) $t = 0.125$ (d) $t = 0.1875$

(e) $t = 0.25$ (f) $t = 0.3125$ (g) $t = 0.375$ (h) $t = 0.4375$

(i) $t = 0.5$ (j) $t = 0.5625$ (k) $t = 0.625$ (l) $t = 0.6875$

(m) $t = 0.75$ (n) $t = 0.8125$ (o) $t = 0.9375$ (p) $t = 1.0$

Figure 1.12: Shock formation in Burgers' equation using Gaussian initial data (1.40) with $A = 1$, $x_0 = 0$, $\Delta = 0.1$. The smooth initial data develops a shock starting at around $t = 0.125$. This is in agreement with the predicted value of $T \approx 0.12$ from (1.41).

Chapter 2

Ultrarelativistic Fluids

We now apply the finite-volume method to the evolution of a relativistic fluid with an ultrarelativistic equation of state. We will first explain how to obtain the equations of motion for the fluid before casting them in an appropriate form for discretization. The theory of ultrarelativistic fluids follows the papers by Martí [4] and Font [5] as well as the project handout [1].

2.1 Theory

For an ultrarelativistic fluid there are two conservation laws corresponding to conservation of baryons and conservation of energy/momentum:

$$\left(\rho_o u^a\right)_{;a} = 0, \tag{2.1}$$

$$\left(T^{ab}\right)_{;b} = 0, \tag{2.2}$$

Here, ρ_o is the proper rest mass density in a local inertial frame and u^a is the four-velocity of the fluid. As usual, T^{ab} is the fluid's stress-energy tensor. The stress-energy tensor can be written

$$T^{ab} = (\rho + P)\, u^a u^b + P g^{ab}\,, \tag{2.3}$$

for a perfect adiabatic fluid where $\rho = \rho_o\,(\epsilon + 1)$ is the energy density of the fluid, P is the pressure and ϵ is the specific internal energy. We will work in Minkowski space so that $g_{ab} = \eta_{ab}$. To fully describe the fluid we need an equation of state that relates P to the other fluid variables. This equation of state is

$$P = (\Gamma - 1)\,\rho\,. \tag{2.4}$$

Note that we are only considering ultrarelativistic fluids so the internal energy density of the fluid greatly exceeds the rest mass energy density: $\rho_o\epsilon \gg \rho_o$. In (2.4), Γ is an adiabatic index that we will be taken to be a constant in $(1, 2]$. We can drop (2.1) from the system if we assume the rest mass density becomes irrelevant when considering an ultrarelativistic fluid. Thus the system of equations reduces to

$$\left(T^{ab}\right)_{;b} = 0. \tag{2.5}$$

We can simplify the problem considerably if we consider a 1D fluid and impose slab symmetry. In slab symmetry, we require our solutions to be invariant in the y and z directions. This simplifies

the fluid variables to be functions of t and x only.

As in Chapter 1 we will solve the system using a finite-volume method. This requires us to cast (2.5) as a conservation law (i.e. in the form of (1.6)). To do so we will introduce the following conservative variables suggested by Neilsen and Choptuik [6]:

$$\tau = (\rho + P) W^2 - P \tag{2.6}$$
$$S = v(\tau + P) \tag{2.7}$$

where $W = \left(1 - v^2\right)^{-1/2} = u^t$ and $v = u^x/u^t$. Note that we also use a system of units where $c = 1$. With these conservative variables the stress-energy tensor becomes

$$T^{tt} = \tau \tag{2.8}$$
$$T^{tx} = T^{xt} = S \tag{2.9}$$
$$T^{xx} = Sv + P \tag{2.10}$$

so that (2.5) simplifies to

$$\partial_t \tau + \partial_x S = 0, \tag{2.11}$$
$$\partial_t S + \partial_x (Sv + P) = 0. \tag{2.12}$$

It follows that the necessary quantities for the conservation equation (1.6) are

$$\mathbf{q} = \begin{bmatrix} \tau \\ S \end{bmatrix} \tag{2.13}$$
$$\mathbf{f}(\mathbf{q}) = \begin{bmatrix} S \\ (Sv + P) \end{bmatrix} \tag{2.14}$$
$$\psi = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{2.15}$$

2.2 Implementation

Following the framework presented in Section 1.2 we will use a Roe solver to approximately solve a Riemann problem at every cell boundary. This means we need the Jacobian matrix (1.26) which has the following components:

$$\begin{aligned} A^1{}_1 &= 0, & A^1{}_2 &= 1, \\ A^2{}_1 &= -v^2 + \left(1 - v^2\right)(\partial P/\partial \tau), & A^2{}_2 &= 2v + \left(1 - v^2\right)(\partial P/\partial S), \end{aligned} \tag{2.16}$$

where

$$\frac{\partial P}{\partial \tau} = -2\beta + \frac{\left(4\beta^2 + \Gamma - 1\right)\tau}{\left[4\beta^2\tau^2 + (\Gamma - 1)\left(\tau^2 - S^2\right)\right]^{1/2}}, \tag{2.17}$$

$$\frac{\partial P}{\partial S} = -\frac{(\Gamma - 1)\, S}{\left[4\beta^2\tau^2 + (\Gamma - 1)\left(\tau^2 - S^2\right)\right]^{1/2}}, \tag{2.18}$$

and where we have defined $\beta = 1/4\,(2 - \Gamma)$ for simplicity. The eigenvalues and eigenvectors of the matrix are

$$\lambda_{\pm} = \frac{1}{2}\left[A^1{}_1 + A^2{}_2 \pm \sqrt{\left(A^1{}_1 - A^2{}_2\right)^2 + 4A^1{}_2A^2{}_1}\right] \tag{2.19}$$

$$\mathbf{r}_{\pm} = \begin{bmatrix} 1 \\ Y_{\pm} \end{bmatrix} \tag{2.20}$$

where $Y_{\pm} = \frac{\lambda_{\pm} - A^1{}_1}{A^1{}_2}$. Meanwhile the jumps in the fluid variables are

$$\omega_+ = \frac{1}{d}\left[r_-[2]\left(\tilde{q}^R[1] - \tilde{q}^L[1]\right) + r_-[1]\left(\tilde{q}^L[2] - \tilde{q}^R[2]\right)\right], \tag{2.21}$$

$$\omega_- = \frac{1}{d}\left[r_+[1]\left(\tilde{q}^R[2] - \tilde{q}^L[2]\right) + r_+[2]\left(\tilde{q}^L[1] - \tilde{q}^R[1]\right)\right], \tag{2.22}$$

$$\tag{2.23}$$

where we have defined

$$d = r_+[1]r_-[2] - r_-[1]r_+[2]. \tag{2.24}$$

Lastly, it follows from Neilsen and Choptuik [6] that the fluid variables in terms of the conservative variables are

$$P = -2\beta\tau + \sqrt{4\beta^2\tau^2 + (\Gamma - 1)\left(\tau^2 - S^2\right)} \tag{2.25}$$

$$\rho = P/(\Gamma - 1) \tag{2.26}$$

$$v = \frac{S}{\tau + P} \tag{2.27}$$

where $\beta = (2 - \Gamma)/4$. This tells us how to convert back to the original fluid variables from our conservative variables.

2.3 Results

The Shock Tube Problem

The shock tube problem is a common test for checking the accuracy of fluid dynamics codes. The test consists of evolving piecewise initial data and observing the formation of several wavefronts in the fluid variables. The most interesting is the rarefaction wave (already discussed in Section 1.2.3 for Burgers' equation) and the shock discontinuity. We will use the initial data $\rho_L = 1$, $\rho_R = 0.1$, $v_R = v_L = 0$ with a grid spacing of $\Delta x = 1/128$.

We observe the formation of both of these wavefronts (Figure 2.1). The rarefaction wave is spread over many grid points and propagates to the left. This is expected; rarefaction is chracterized by all fluid variables being continuous. The shock wavefront is characterized by being discontinuous and propagates to the right. It is notable that our shock front is spread out over several points. Although the shock narrows for higher grid spacings (see Figure 2.4) it is not as sharp as one might expect. I hypothesize that this is due to the slope limiter; the trade-off of reducing oscillations is less accuracy near the shock front.

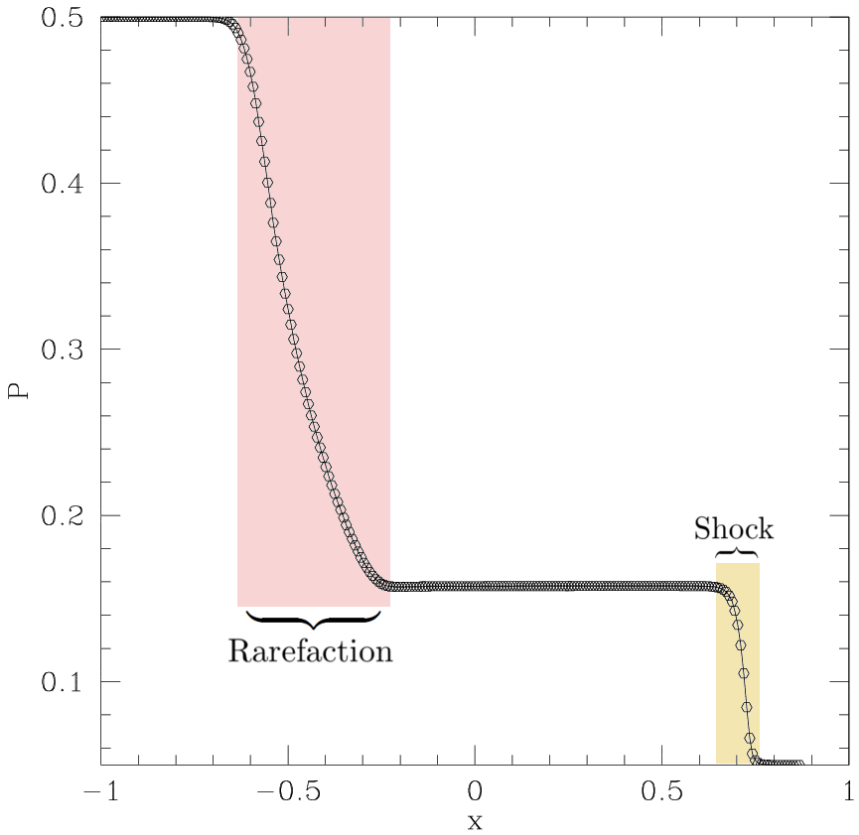
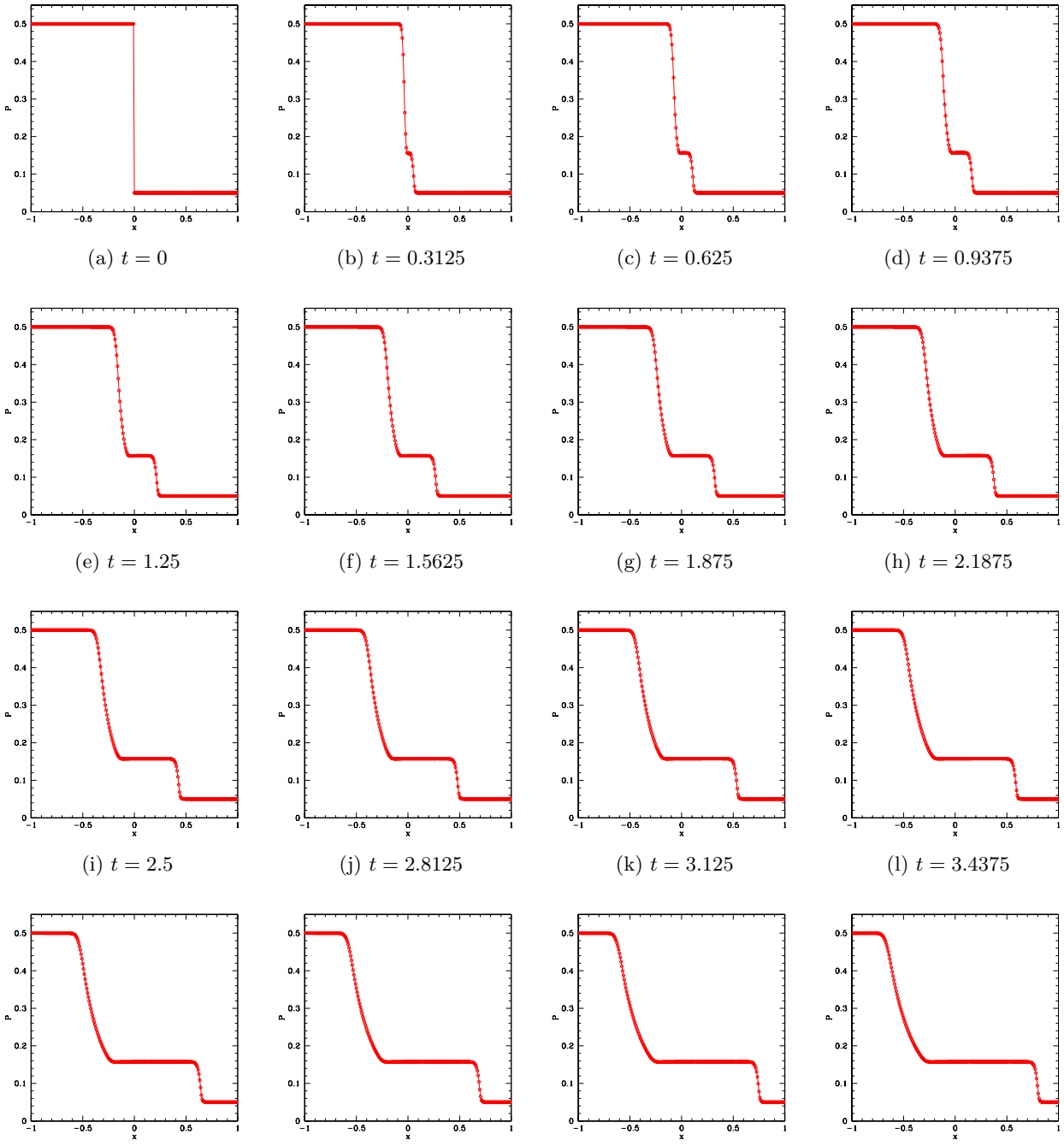


Figure 2.1: The shock tube problem for an ultrarelativistic fluid. This snapshot of the pressure P was taken with piecewise initial data $\rho_L = 1$, $\rho_R = 0.1$, $v_R = v_L = 0$, $\Delta x = 1/128$.



(a) $t = 0$ (b) $t = 0.3125$ (c) $t = 0.625$ (d) $t = 0.9375$

(e) $t = 1.25$ (f) $t = 1.5625$ (g) $t = 1.875$ (h) $t = 2.1875$

(i) $t = 2.5$ (j) $t = 2.8125$ (k) $t = 3.125$ (l) $t = 3.4375$

(m) $t = 3.75$ (n) $t = 4.0625$ (o) $t = 4.375$ (p) $t = 4.6875$

Figure 2.2: Evolution of an ultrarelativistic fluid in a shock tube with initial data $\rho_L = 1$, $\rho_R = 0.1$, $v_R = v_L = 0$. The rarefaction wave and the shock quickly form. The rarefaction wave propagates to the left and the shock propagates to the right. Notice that the shock has a greater speed than the rarefaction wave.

Convergence Test

We can confirm that our solution is second-order accurate by plotting the fluid variables for a variety of grid spacings and observing that they converge. We choose Gaussian initial data of the form

$$\rho(0, x) = A \exp\left(-(x - x_0)^2/\Delta^2\right). \tag{2.28}$$

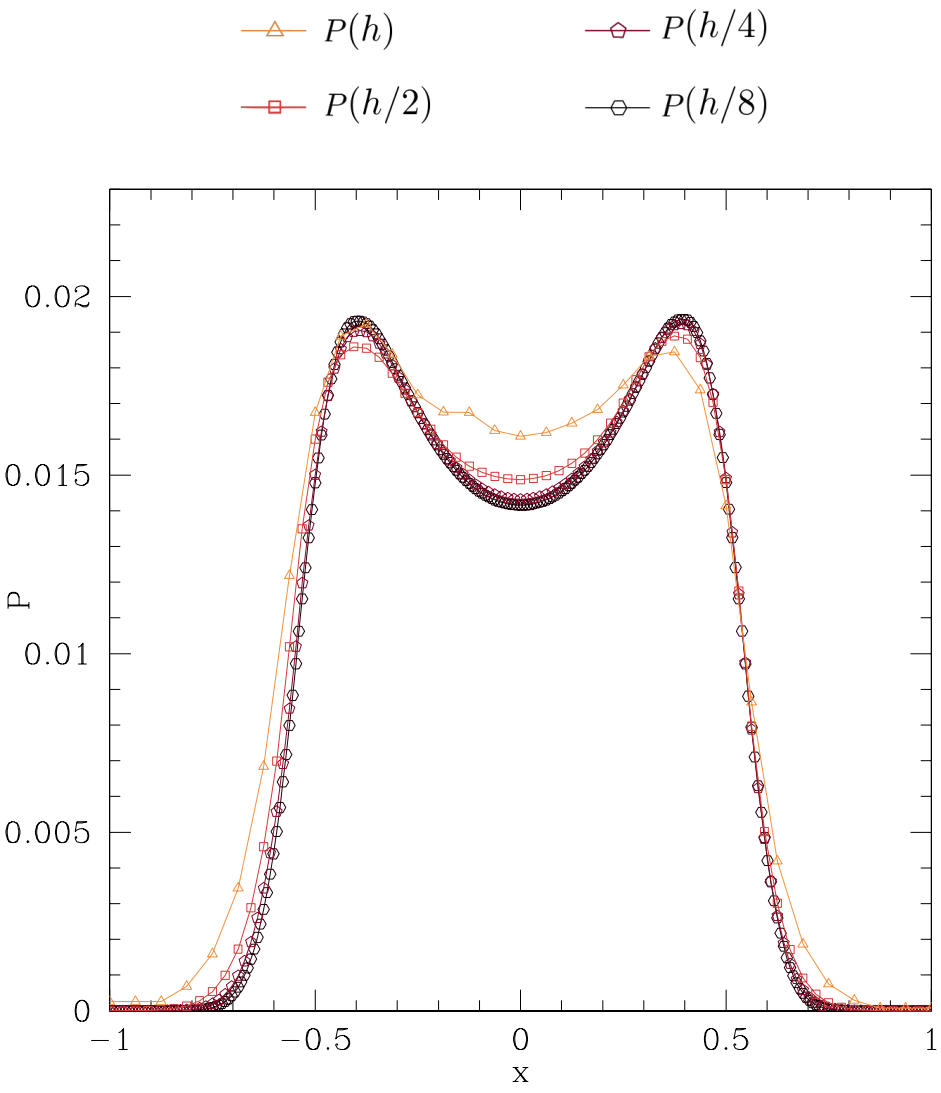


Figure 2.3: Convergence test for the pressure P of ultrarelativistic fluid with Gaussian initial data (2.28) with $A = 1$, $x_0 = 0$, $\Delta = 0.1$. Similar to Burgers' equation, the finite-volume method is second-order accurate except near the shocks where it is $O(h)$.

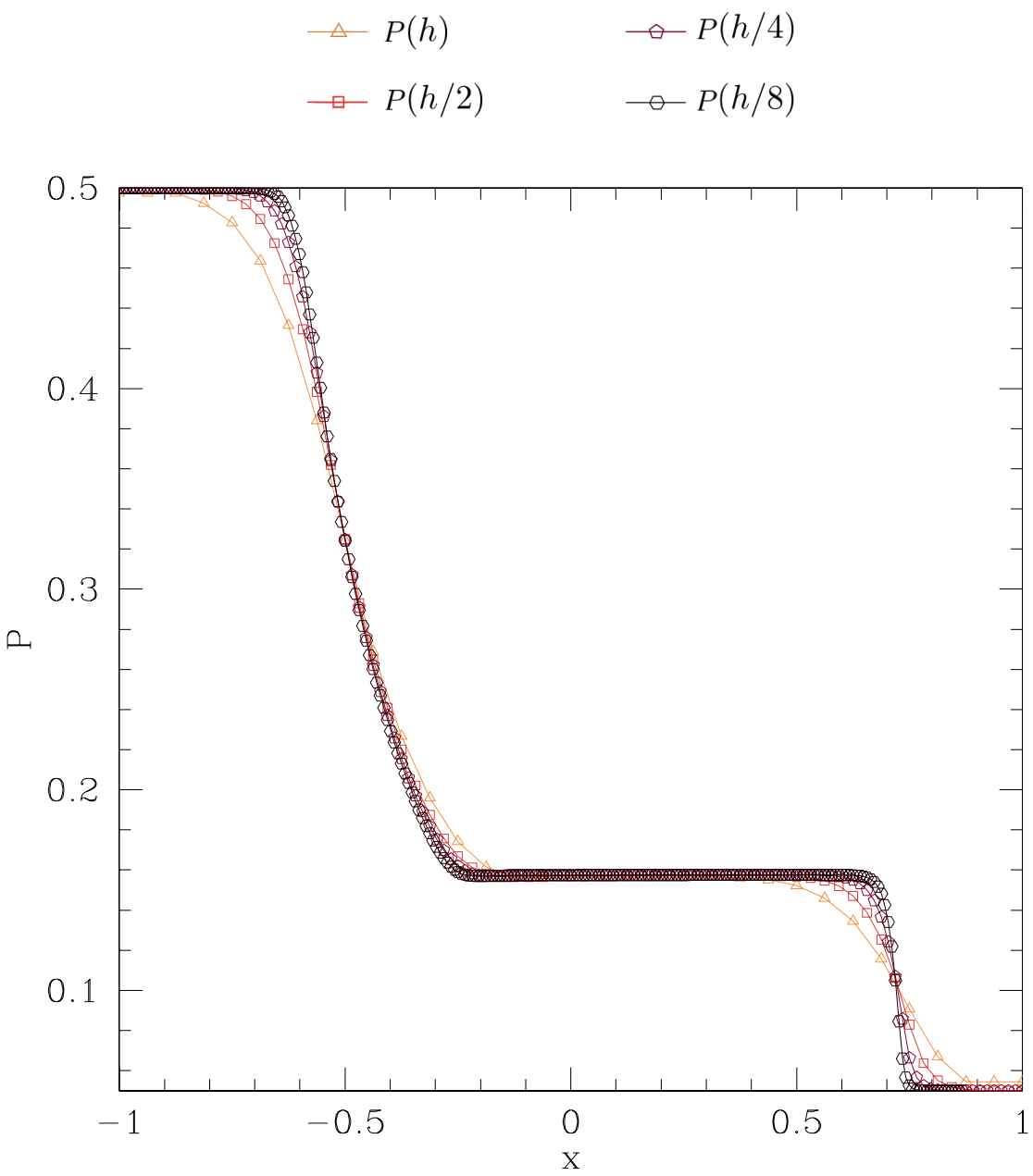


Figure 2.4: Convergence test for the pressure P of ultrarelativistic fluid with piecewise initial data $\rho_L = 1$, $\rho_R = 0.1$, $v_R = v_L = 0$. As the grid spacing is decreased the wavefronts become more vertical. Similar to Burgers' equation, the finite-volume method is second-order accurate except near the shocks where it is $O(h)$.

Conclusion

In Chapter 1, we studied Burgers' equation using finite-difference methods and a finite-volume Roe solver. In Chapter 2, we extended the finite-volume code to study the evolution of an ultrarelativistic fluid in slab symmetry. We performed convergence tests and studied the 'shock tube problem' and rarefaction waves.

The scope of the project was quite advanced and so time constraints limited the analysis of the ultrarelativistic fluid. Thus there is still much that can be done. For instance, it would be interesting to study the ultrarelativistic limit where the Lorentz factor $W \to c$ and investigate the dynamics. It would also be worthwhile to investigate the speed of shock propagation for the fluid and confirm that it obeys the Rankine-Hugoniot condition. These will be done in the future as they are good training exercises for my research in numerical relativity.

As a final note I would like to thank Matt Choptuik and Graham Reid for their guidance.

Bibliography

[1] M. Choptuik, Project 2: 1D Ultrarelativistic Fluid, webpage, http://laplace.physics.ubc.ca/People/matt/Teaching/03Vancouver/p2.pdf, accessed 2017-11-17.

[2] R. J. LeVeque, *Numerical Methods for Conservation Laws*, Birkhäuser Basel, second edition, 1992.

[3] P. Roe, J. Comput. Phys. **43**, 357 (1981).

[4] J. M. Martí and E. Müller, Living Rev. Relativ. **6**, 7 (2003).

[5] J. A. Font, Living Rev. Relativ. **3**, 2 (2000).

[6] D. W. Neilsen and M. W. Choptuik, Class. Quantum Grav. **17**, 761 (2000).

[7] R. Marsa and M. Choptuik, The RNPL Reference Manual, webpage, http://laplace.physics.ubc.ca/People/marsa/rnpl/refman/refman.html, accessed 2017-11-17.

[8] R. Marsa and M. Choptuik, The RNPL User's Guide, webpage, http://laplace.physics.ubc.ca/People/marsa/rnpl/users_guide/users_guide.html, accessed 2017-11-17.

Appendix A

Code Listings

Code for this project was written in Fortran and RNPL (Rapid Numerical Prototyping Language). RNPL is a special-purpose language designed for solving time-dependent systems of PDEs [7] [8]. The user specifies the essential structure of the discretization and the Fortran programs are automatically generated. This is an essential tool for this project because writing the Fortran routines for memory management, input/output control, parameter reading, etc. are otherwise extremely time-consuming and tedious.

The basic code was provided on the summer school webpage. The basic code was then modified to answer each of the questions in the handout.

**Note that you must have RNPL installed in order to run the code.* It can be downloaded here: http://laplace.physics.ubc.ca/Doc/rnpletal/

A.1 burgers_exact

To run the code, please type in the terminal: `burgers_exact id0`
The output data will be generated as .sdf files

burgers_exact_rnpl: RNPL code for exact solution of Burgers' equation. Allows the user to specify the domain of solution and the initial conditions for piecewise initial data.

id0: initial parameter file for burgers_exact_rnpl. Allows the user to specify additional parameters and change the initial conditions.

update.inc: a routine for updating the solution based on the exact solution of Burgers' equation.

init.inc: an initialization routine based on the parameters supplied to id0.

burgers_exact.f: a complete Fortran program generated by RNPL for memory management. When compiled the executable will accept the parameter file id0 and output a .sdf file containing the solution data.

burgers_exact_init.f: another memory management program that is invoked if no parameter file is supplied to burgers_exact.

gfuni0.inc: a routine that declares the arrays needed for specifying grid functions in burgers_exact.

globals.inc/other_glbs.inc/sys_param.inc: a routine that declares various variables, strings and arrays for RNPL.

initializers.f: a routine that initializes the grid function i.e. defines the function qxct over the grid.

updates.f: updates the grid function to evolve the solution i.e. updates qxct for each time step.

A.2 burgers

To run the code, please type in the terminal: `burgers id0`
The output data will be generated as .sdf files

burgers_rnpl: implements an approximate Riemann solver (Roe solver) for the finite-volume method. Allows the user to specify the initial conditions and domain of integration. Also implements the naive finite-difference discretizations to show that they give incorrect results.

id0: initial parameter file for burgers_rnpl. Allows the user to specify whether they want piecewise initial data or Gaussian initial data.

burgers.f: a complete Fortran program generated by RNPL for memory management. When compiled the executable will accept the parameter file id0 and output a .sdf file containing the solution data.

burgers_init.f: another memory management program that is invoked if no parameter file is supplied to burgers_exact.

calc_A.f: calculates the Jacobian matrix **A** for the Roe numerical flux.

calc_eta.f/calc_lambda.f: calculates the eigenvalues of the Jacobian matrix **A**.

calc_FF.f/calc_flux.f: calculates the Roe numerical flux and physical flux for Burgers' equation.

calc_f.f: calculates the jumps ω in the fluid variables.

gfuni0.inc: a routine that declares the arrays needed for specifying grid functions in burgers_exact.

globals.inc/other_glbs.inc/sys_param.inc: a routine that declares various variables, strings and arrays for RNPL.

init_q.inc: an initialization routine that supplies the user-specified parameters to burgers.f.

initializers.f: initializes the grid functions q_nc, q_2c, q_c, q_0 for the UNCS, SOCS, UCS and finite-volume method.

minmod.f: implements the minmod slope limiter.

recos_qL.f/recos_qR.f: calculates the reconstructed variables using the minmod slope limiter

update_q.f/update_boundary.f/step.inc : implements the outflowing boundary conditions and uses a second-order Runge-Kutta method to evolve the solution

updates.f: updates the grid functions q_nc, q_2c and q_c for the UNCS, SOCS, UCS

A.3 ultra

To run the code, please type in the terminal: `ultra id0`
The output data will be generated as .sdf files

ultra_rnpl: implements an approximate Riemann solver (Roe solver) to solve the equations of motion for an ultrarelativistic fluid with the finite-volume method. Initializes parameters, ghost cells, update routines, etc. for other Fortran routines.

id0: initial parameter file for ultra_rnpl. Allows the user to specify whether they want piecewise initial data or Gaussian initial data. Also allows them to specify other parameters like step size, RNPL parameters, etc.

init_fluid.inc: initializes the fluid variables just as q.inc does for the Burgers' code above. Initializes 5 grid functions and implements both piecewise and Gaussian initial data.

step.inc: update code that is interfaced to the RNPL-generated update routine via the UPDATES statement in ultra_rnpl. Implements a second-order Runge-Kutta method for updating.

calc_flux.f: routine that calculates the numerical fluxes via the 2×2 Jacobian with components A11, A12, A21, A22. Used in conjunction with calc_cons.f, calc_A.f and other routines to compute the conservative variables.

calc_cons.f: computes τ, S, P.

calc_prim.f: computes ρ, v, P. Calls calc_cons.f.

calc_A.f: evaluates the Jacobian matrix $\mathbf{A}$ (which is now 2×2).

calc_lambda.f: computes the eigenvalues of the Jacobian matrix $\mathbf{A}$.

calc_lambda.f: computes the eigenvectors of the Jacobian matrix $\mathbf{A}$.

calc_f.f: computes the physical fluxes (2.14).

calc_omega.f: computes the jumps ω in the original variables. There are two such jumps since there are two variables.

calc_FF.f: calculates the numerical fluxes $\mathbf{F}^{\text{Roe}}_{i+1/2}$.

dvdump.f: debugging script that dumps into fort.80.

minmod.f: implements the minmod slope limiter.

recos_qL.f/recos_qR.f: calculates the reconstructed variables using the minmod slope limiter

update_q.f: updates the conservative variables.

update_boundary.f: updates the original fluid variables. Implements outflow boundary conditions using ghost cells.