Project 2: 1D Ultrarelativistic Fluid.

March 1, 2011

1 Introduction

In this project we study the evolution of a relativistic fluid with an ultrarelativistic equation of state, in slab symmetry. We will use one of the so called HRSC (High Resolution Shock Capturing) methods that are suited for the evolution of discontinuities. These methods have been proven to be useful in both the special relativistic and general relativistic cases (see Martí, et al. [1] and Font [2], respectively).

In this handout we first describe how one obtains the equations of motion for such a fluid, focusing on casting the equations in a form appropriate for discretization. In section 3 we describe the discretization *per se*, as well as the numerical method that you will use to solve the discrete equations. In 4 we describe a simpler system—Burger's equation—and provide a code that solves it using the same HRSC method described for the fluid. You will perform some simple numerical experiments with this code, before moving onto the main task of implementing the fluid solver. The last section contains notes concerning how Burger's equation code can be modified to produce a code to solve the fluid equations.

2 Equations of Motion

The equations of motion can be derived from the following conservation laws:

$$\left(\rho_o u^a\right)_{;a} = 0, \tag{1}$$

$$\left(T^{ab}\right)_{;b} = 0, \tag{2}$$

where ρ_o is the proper rest mass density in a local inertial frame, u^a is the four velocity of the fluid, and T^{ab} is the fluid's energy momentum tensor. Equation (1) expresses the conservation of baryons in the system while (2) represents the conservation of the energy and momentum.

For a perfect fluid (we will only consider an adiabatic fluid, i.e. we do not consider heat exchange or viscous terms) the stress energy tensor can be written as (see MTW [3], 22.3)

$$T^{ab} = \left(\rho + P\right) u^a u^b + P g^{ab}\,, \tag{3}$$

where $\rho = \rho_o\left(\epsilon + 1\right)$ is the energy density of the fluid, g^{ab} is the inverse metric and ϵ is the specific internal energy. In our case we consider a fluid on Minkowski spacetime (i.e. non-self-gravitating), and work in Cartesian coordinates; therefore $g^{\mu\nu} = \eta^{\mu\nu} \equiv \text{diag}\{-1, 1, 1, 1\}$.

In order to completely describe the fluid we need to augment the equations of motion with an equation of state (EOS) that relates the pressure to the rest of the fluid variables. In this project we will consider an *ultrarelativistic* fluid (i.e. we assume that the internal energy density of the fluid is much larger than the rest mass density—$\rho_o\epsilon \gg \rho_o$) for which the equation of state is:

$$P = (\Gamma - 1)\,\rho\,. \tag{4}$$

Here, Γ is the adiabatic index that we will be taken to be a constant in the range $(1, 2]$. With such an EOS, the rest mass density becomes dynamically irrelevant, and we can drop equation (1) from the system. Therefore the only equations that we need to consider are (2) which we can now write as:

$$\left(T^{\mu\nu}\right)_{,\nu} = 0. \tag{5}$$

The method of solution that we will use requires that our equations of motion be cast in conservation law form, i.e. in the form

$$\frac{\partial\,\mathbf{q}}{\partial t} + \frac{\partial\,\mathbf{f}(\mathbf{q})}{\partial x^i} = \boldsymbol{\psi}\,. \tag{6}$$

Here, $\mathbf{q}$ is a vector formed by the so called the *conservative* variables, $\mathbf{f}(\mathbf{q})$ is a vector of fluxes which depend on these variables (in general not explicitly) [1] and $\boldsymbol{\psi}$ are source functions.

We now impose the condition of *slab symmetry*, i.e we demand that our solutions are invariant under translations in the y and z directions. With this assumption, the dynamical variables, as well as their time derivatives, become functions of x and t alone. (We also work in a coordinate system in which the fluid velocity components, v^y and v^z, vanish).. In order to write equations (5) in conservation form it will be useful to introduce the following conservative variables, following Neilsen and Choptuik[4] (hereafter NC):

$$\begin{aligned} \tau &= (\rho + P)\,W^2 - P & (7)\\ S &= v\,(\tau + P) & (8)\end{aligned}$$

where $W = \left(1 - v^2\right)^{-1/2} = u^t$ and $v = u^x/u^t$. Using these variables, is easy to express the relevant components of the energy momentum tensor:

$$T^{tt} = \tau, \qquad T^{tx} = T^{xt} = S, \qquad T^{xx} = Sv + P. \tag{9}$$

The non-trivial equations of motion derived from (5) are:

$$\begin{aligned} \dot\tau + S' &= 0, & (10)\\ \dot S + (Sv + P)' &= 0, & (11)\end{aligned}$$

where the dot means $\partial/\partial t$ and the prime $\partial/\partial x$. At this point it is important to stress that these *conservative* variables are not *extra* variables needed to describe the state of the fluid but a different set of state variables. A crucial part of the algorithm will be the prescription of how to transform from the conservative variables $\{\tau, S\}$ to the *primitive* variables $\{\rho, v\}$ and vice versa.

[1]Actually at some points it will be more interesting to consider the fluxes to be function of the primitive variables $\mathbf{p} = \{v, \rho\}$.

Equations (10-11) are in conservation law form where the vector $\mathbf{q}$, the physical fluxes $\mathbf{f}(\mathbf{q})$, and the source terms $\boldsymbol{\psi}$ are given by

$$\mathbf{q} = \begin{bmatrix} \tau \\ S \end{bmatrix} , \quad \mathbf{f}(\mathbf{q}) = \begin{bmatrix} S \\ (Sv + P) \end{bmatrix} , \quad \boldsymbol{\psi} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} . \tag{12}$$

Notice that the fluxes not only depend on $\mathbf{q}$ directly but also implicitly through v and P.

3 Discretization

3.1 Finite Volumes

The system of equations (10-11) generically give rise to shocks (discontinuities on the variables describing the state of the fluid) even when the initial data is smooth. This implies that naïve discretizations based on the continuity of the functions (like some of the finite difference methods used on Project 1) are doomed to fail. There are different approaches we could take to solve this system. Here we will take a finite volume approach, meaning that we will assume that we have a mesh of grid points that define a cell structure on our spacetime (see Figure 1). In the presence of discontinuities the only way to make sense of our system

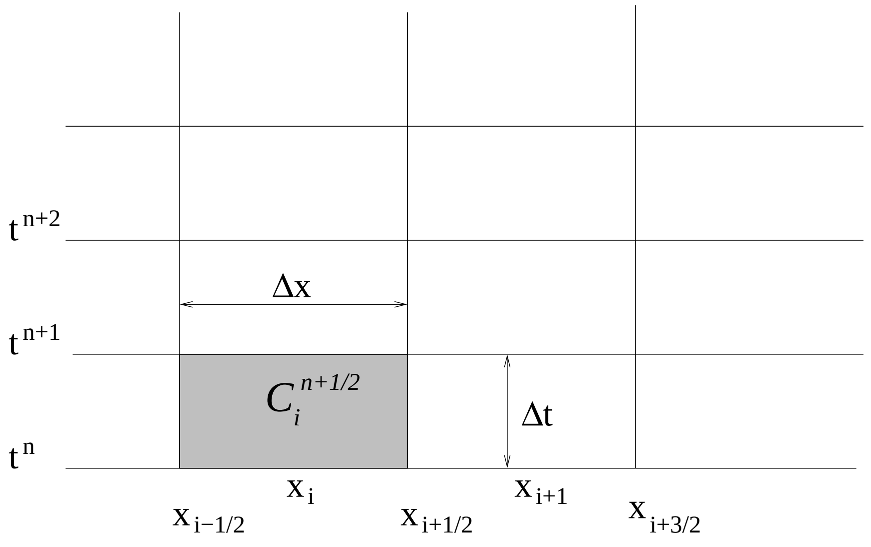

Figure 1: Cell structure of the spacetime for a finite volume discretization in one dimension. The spacetime cells $C_i^{n+1/2}$ are centred at positions $(t^{n+1/2}, x_i)$ and their volumes are $\Delta t \Delta x$.

of equations is to consider averages over a finite volume of the spacetime. Therefore to find the discretization we take the average of equation (6) over a spacetime cell $\mathcal{C}_i^{n+1/2}$:

$$\frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \frac{\partial \mathbf{q}}{\partial t} + \frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \frac{\partial \mathbf{f}}{\partial x} = \frac{1}{V_{\mathcal{C}_i^{n+1/2}}} \int_{\mathcal{C}_i^{n+1/2}} \boldsymbol{\psi}, \tag{13}$$

where $\mathcal{C}_i^{n+1/2}$ is the region of spacetime defined by $(t^n, t^{n+1}) \times (x_{i-1/2}, x_{i+1/2})$, and $V_{\mathcal{C}_i^{n+1/2}} = \Delta t \Delta x$ is its volume. The resulting equation can be written as:

$$\frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \frac{\partial \mathbf{q}}{\partial t} dx dt + \frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \frac{\partial \mathbf{f}}{\partial x} dx dt = \frac{1}{\Delta t \Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \boldsymbol{\psi} dx dt. \tag{14}$$

We can partially integrate the different terms of the equation using Gauss' theorem:

$$\frac{\bar{\mathbf{q}}_i^{n+1} - \bar{\mathbf{q}}_i^n}{\Delta t} + \frac{\mathbf{F}_{i+1/2}^{n+1/2} - \mathbf{F}_{i-1/2}^{n+1/2}}{\Delta x} = \widehat{\boldsymbol{\psi}}_i^{n+1/2}. \tag{15}$$

Here we have used the following definitions: the spatial averages of the conservative variables,

$$\bar{\mathbf{q}}_i^n \equiv \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{q}(t^n, x) dx, \tag{16}$$

the temporal averages for the fluxes, also referred as the numerical fluxes,

$$\mathbf{F}_{i+1/2}^{n+1/2} \equiv \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} \mathbf{f}\left(\mathbf{q}(t, x_{i+1/2})\right) dt, \tag{17}$$

and the total averages over the spacetime cell for the sources,

$$\widehat{\boldsymbol{\psi}}_i^{n+1/2} \equiv \frac{1}{\Delta x \Delta t} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{t^n}^{t^{n+1}} \boldsymbol{\psi}(t, x) dx dt. \tag{18}$$

The idea now is to use equation (15) to calculate $\{\bar{\mathbf{q}}_i^{n+1}\}$ assuming we know the values $\{\bar{\mathbf{q}}_i^n\}$. However the calculation of the numerical fluxes $\{\mathbf{F}_{i+1/2}^{n+1/2}\}$ is not as straightforward as one may think—these fluxes are averages in time, so in order to explicitly calculate them we need to already know the solution. More importantly, the values for the fluid quantities on the left side of the cell boundary $\{\bar{\mathbf{q}}_i^n\}$ and on the right side $\{\bar{\mathbf{q}}_{i+1}^n\}$ won't agree in general. The values have discontinuities and *a priori* is not clear which values to use in order to compute the numerical fluxes. One way to solve these problems is to use an idea due to Godunov that involves solving a Riemann problem at every cell boundary in order to calculate $\{\mathbf{F}_{i+1/2}^{n+1/2}\}$. For more information about Godunov methods see LeVeque [5]. In the following section we explain one such method.

3.2 Roe Solver

The solution of the full Riemann problem at every cell boundary is usually not very efficient. In most cases the overall time step to update the variables to the future time will involve some kind of iterative process, and thus exactly solving the Riemann problem at each iteration will not imply that the overall process will be solved more rapidly or accurately. The Roe solver is a solver that uses modified Riemann problems in order to compute the numerical fluxes. For a more extensive explanation of this and other approximate Riemann solvers see LeVeque [5]. The main idea is to linearize the fluxes in equation (6) as functions of $\mathbf{q}$, also assuming that the sources vanish:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial x} = 0. \tag{19}$$

Considering $\partial \mathbf{f}/\partial \mathbf{q}$ to have constant coefficients linearizes the above equation, and a solution can be obtained by diagonalizing the Jacobian matrix (see appendix for the solution of the scalar linear equation and its generalization to a system of linear equations). The numerical flux can then be written as a function of the solution to this problem. Here we write the resulting numerical fluxes directly as:

$$\mathbf{F}^{\mathrm{Roe}}_{i+1/2} = \frac{1}{2}\left[\mathbf{f}\left(\tilde{\mathbf{p}}^R_{i+1/2}\right) + \mathbf{f}\left(\tilde{\mathbf{p}}^L_{i+1/2}\right) - \sum_\alpha |\lambda_\alpha| \omega_\alpha \mathbf{r}_\alpha\right]. \tag{20}$$

Some explanation of the different terms that appear in the above equation are in order. First, $(\tilde{\mathbf{p}}^R,\ \tilde{\mathbf{p}}^L)$, known as right and left *reconstructed variables*, are the values of the primitive variables at the boundary, $x_{i+1/2}$ calculated via some specific reconstruction (interpolation) scheme. Special care is taken in calculating the reconstructed variables in order to reduce spurious oscillations close to discontinuities. Here we use a *slope limiter* interpolation to compute the reconstructed values (see Martí and Mueller [1] and LeVeque [5] for alternate reconstruction algorithms):

$$\begin{aligned}
\tilde{\mathbf{p}}^L_{i+1/2} &= \bar{\mathbf{p}}_i + \boldsymbol{\sigma}_i \left(x_{i+1/2} - x_i\right), & (21)\\
\tilde{\mathbf{p}}^R_{i+1/2} &= \bar{\mathbf{p}}_{i+1} + \boldsymbol{\sigma}_{i+1} \left(x_{i+1/2} - x_{i+1}\right), & (22)
\end{aligned}$$

where $\boldsymbol{\sigma}_i$ is given by

$$\boldsymbol{\sigma}_i = \mathrm{minmod}\left(\mathbf{s}_{i-1/2}, \mathbf{s}_{i+1/2}\right). \tag{23}$$

Here:

$$\mathbf{s}_{i+1/2} = \frac{\bar{\mathbf{p}}_{i+1} - \bar{\mathbf{p}}_i}{x_{i+1} - x_i}, \tag{24}$$

and the minmod function is defined by

$$\mathrm{minmod}(a,b) = \begin{cases} 0 & \text{if} \quad ab < 0 \\ a & \text{if} \quad |a| < |b| \ \text{and} \ ab > 0 \\ b & \text{if} \quad |a| > |b| \ \text{and} \ ab > 0. \end{cases} \tag{25}$$

In equation (20) we also use the characteristic structure of the Riemann problem at the $x_{i+1/2}$ interface (λ_α, ω_α, $\mathbf{r}_\alpha$). Given the Jacobian matrix:

$$\mathbf{A}|_{i+1/2} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right|_{\mathbf{q}=1/2\,(\tilde{\mathbf{q}}^L_{i+1/2}+\tilde{\mathbf{q}}^R_{i+1/2})}, \tag{26}$$

λ_α are the eigenvalues[2] of $\mathbf{A}$, $\mathbf{r}_\alpha$ are the right eigenvectors associated with the eigenvalues λ_α and ω_α are the jumps in the characteristic variables defined by

$$\tilde{\mathbf{q}}^R_{i+1/2} - \tilde{\mathbf{q}}^L_{i+1/2} = \sum_\alpha \omega_\alpha \boldsymbol{r}_\alpha. \tag{27}$$

[2]Here α (and later on also β) labels the equation number, in the fluid case since we have two equations it takes values on $\{1, 2\}$.

Here $(\tilde{\mathbf{q}}^R, \tilde{\mathbf{q}}^L)$ are the values of the conservative variables calculated from the reconstructed primitive variables $(\tilde{\mathbf{p}}^R, \tilde{\mathbf{p}}^L)$. Reconstruction of the primitive variables followed by transformation to conservative variables generally yields more stable results than direct reconstruction of the conservative variables.

The final part of the Roe solver involves the update of $\{\bar{\mathbf{q}}_i^n\}$. The fact that we use an approximate Riemann solver to calculate the numerical flux makes (20), when evaluated using the spatial averages at time t^n, a first order approximation to the real numerical flux defined by (17). This is usually the case also when calculating the numerical sources (18). In order to make the time evolution second order (in the temporal discretization scale), we use a second order Runge-Kutta method to advance the solution in time:

$$\bar{\mathbf{q}}^{n+1/2} = \bar{\mathbf{q}}^n + \frac{\Delta t}{2} L(\bar{\mathbf{q}}^n) \tag{28}$$

$$\bar{\mathbf{q}}^{n+1} = \bar{\mathbf{q}}^n + \Delta t L(\bar{\mathbf{q}}^{n+1/2})\,. \tag{29}$$

Here L is defined by:

$$L(\bar{\mathbf{q}}^n) = -\frac{\mathbf{F}^{Roe}_{i+1/2}(\bar{\mathbf{q}}^n) - \mathbf{F}^{Roe}_{i-1/2}(\bar{\mathbf{q}}^n)}{\Delta x} + \widehat{\boldsymbol{\psi}}_i(\bar{\mathbf{q}}^n)\,. \tag{30}$$

Characteristic Structure for the ultrarelativistic fluid

For the case of our system of fluid equations, with $\mathbf{q}$ and $\mathbf{f}(\mathbf{q})$ are given by (12) the matrix $\mathbf{A} = (A^\alpha{}_\beta)$ has the following components:

$$\begin{array}{ll} A^1{}_1 = 0, & A^1{}_2 = 1, \\ A^2{}_1 = -v^2 + (1-v^2)\,(\partial P/\partial \tau)\,, & A^2{}_2 = 2v + (1-v^2)\,(\partial P/\partial S)\,, \end{array} \tag{31}$$

where

$$\frac{\partial P}{\partial \tau} = -2\beta + \frac{(4\beta^2 + \Gamma - 1)\,\tau}{[4\beta^2\tau^2 + (\Gamma-1)\,(\tau^2 - S^2)]^{1/2}}, \tag{32}$$

$$\frac{\partial P}{\partial S} = -\frac{(\Gamma-1)\,S}{[4\beta^2\tau^2 + (\Gamma-1)\,(\tau^2 - S^2)]^{1/2}}, \tag{33}$$

and $\beta = 1/4\,(2-\Gamma)$.

In terms of these matrix components, the eigenvalues λ_α are

$$\lambda_\pm = \frac{1}{2}\left[A^1{}_1 + A^2{}_2 \pm \sqrt{(A^1{}_1 - A^2{}_2)^2 + 4A^1{}_2 A^2{}_1}\right], \tag{34}$$

and the right eigenvectors, $\mathbf{r}_\pm$, are

$$\mathbf{r}_\pm = \begin{bmatrix} 1 \\ Y_\pm \end{bmatrix}, \qquad Y_\pm = \frac{\lambda_\pm - A^1{}_1}{A^1{}_2}\,. \tag{35}$$

Accordingly, the jumps, ω_α, are given by

$$\begin{aligned}
\omega_+ &= \frac{1}{d}\left[r_-[2]\left(\tilde{q}^R[1]-\tilde{q}^L[1]\right)+r_-[1]\left(\tilde{q}^L[2]-\tilde{q}^R[2]\right)\right], && (36)\\
\omega_- &= \frac{1}{d}\left[r_+[1]\left(\tilde{q}^R[2]-\tilde{q}^L[2]\right)+r_+[2]\left(\tilde{q}^L[1]-\tilde{q}^R[1]\right)\right], && (37)\\
& && (38)
\end{aligned}$$

where

$$d = r_+[1]r_-[2] - r_-[1]r_+[2]. \tag{39}$$

In the previous equations $\tilde{q}^R[1]$ stands for the first component of the vector $\tilde{\mathbf{q}}^R$ and similarly for the rest of the analogous expressions.

Calculation of the conservative/primitive variables

We have explained how the primitive variables are reconstructed at the boundaries of the cells using a slope limiter. However, in order to compute the physical fluxes, $\mathbf{f}(\tilde{\mathbf{p}}^L)$ and $\mathbf{f}(\tilde{\mathbf{p}}^R)$, that appear in equation (20), we need to compute the conservative variables as well. The conservative variables can be calculated from the primitive variables via equations (7), (8), (4) and the definition of $W = (1-v^2)^{-1/2}$.

Conversely at every half and full step in our update procedure we need to calculate the primitive variables after the conservative variables have been evolved. It is not difficult to invert the equations that define the conservative variables in order to get the primitive ones (see NC [4]):

$$\begin{aligned}
P &= -2\beta\tau + \sqrt{4\beta^2\tau^2 + (\Gamma-1)\left(\tau^2 - S^2\right)} && (40)\\
\rho &= P/(\Gamma-1) && (41)\\
v &= \frac{S}{\tau+P} && (42)
\end{aligned}$$

where $\beta = (2-\Gamma)/4$.

Floor

Due to numerical errors (truncation error, roundoff error) the quantities that describe the fluid can sometimes take unphysical values (i.e. negative pressures, negative densities, speeds larger than one, etc,...) (see NC [4]). Effects of such errors become particularly important in "evacuated" regions, where densities are low and velocities can be very large. In order to circumvent problems associated with these errors, we force certain values to be above some threshold, that we call a *floor*. For the ultrarelativistic fluid it is convenient to floor the conservative variable τ in the following way:

$$\tau = \max\{\tau, \mathit{floor} + |S|\}, \tag{43}$$

where $|S|$ is the absolute value of S and *floor* is a small value, typically several orders of magnitude (usually 13 or 14 orders of magnitude) smaller than typical values of τ. Generally, a flooring procedure of this sort will not have an important dynamical effect (although this is something that needs to be verified empirically), and ameliorates the problems described above. We recommend application of this algorithm every time the conservative variables are updated or calculated from reconstructed primitive variables at the cell boundaries.

4 Burger's Equation

Burger's equations is an example of a non-linear scalar equation that produces shocks, even from smooth initial data (see LeVeque [5]). One form of the equation is

$$\dot{q} + q\frac{\partial q}{\partial x} = 0. \tag{44}$$

which is easy to cast into conservative form

$$\dot{q} + \left(\frac{1}{2}q^2\right)' = 0. \tag{45}$$

Using the notation introduced previously, we have

$$\mathbf{q} = q, \qquad \mathbf{f} = \frac{1}{2}q^2, \qquad \boldsymbol{\psi} = 0. \tag{46}$$

We solve this equation with a finite volume discretization and a Roe solver, as outlined in the previous section. The finite volume discretization of the equation is

$$\frac{\bar{q}_i^{n+1} - \bar{q}_i^n}{\Delta t} + \frac{F_{i+1/2}^{n+1/2} - F_{i-1/2}^{n+1/2}}{\Delta x} = 0\,, \tag{47}$$

where $\bar{q}_i^n$ is the spatial average defined by equation (16) and $F_{i+1/2}^{n+1/2}$ is the numerical flux. We now focus on a description of the characteristic structure of the equation that will allow us to compute the Roe flux. Since (44) is a scalar equation the Jacobian matrix $\mathbf{A}$ is also a scalar

$$\mathbf{A} = \frac{1}{2}\left(q^L + q^R\right), \tag{48}$$

the eigenvalue is the same scalar, $\lambda = 1/2\left(q^L + q^R\right)$, and we can take the right eigenvector to be 1. Finally, the jump ω is just the difference of q across the cell boundary,

$$\omega = q^R - q^L\,. \tag{49}$$

Thus, we can write the Roe numerical flux as:

$$F_{i+1/2}^{\mathrm{Roe}} = \frac{1}{2}\left[f\left(q^L\right) + f\left(q^R\right) - |\lambda|\, r\, \omega\right]_{i+1/2}, \tag{50}$$

where $[\cdots]_{i+1/2}$ means that the quantities within the bracket are evaluated at $x_{i+1/2}$. We then solve (47) using the following time-stepping procedure:

```
1) Calculate the Roe numerical fluxes at the cell boundaries.
2) Update the variables to the half time step using equation (47) with
   Delta t=Delta t/2.
3) Use the quantities at the half time step to compute the Roe numerical fluxes.
4) Do a full step to update to the future time step using the numerical fluxes
   calculated in 3).
```

The previous pseudo code describes the use of the Roe solver within a second order Runge-Kutta time stepping scheme. The overall method should be second order if no shocks are developed, except in the vicinity of extrema of the dynamical variable, where the slope limiting interpolation will generally degrade the solution to first order.

The calculation of the numerical fluxes involve the following steps:

```
1) Calculate the left and right reconstructed variables at the cell boundary.
2) Calculate the characteristic structure: eigenvalues, right eigenvectors
   and jumps in the characteristic variables.
4) Calculate the physical fluxes F for the left and right reconstructed
   variables.
3) Calculate the Roe numerical flux using equation (50).
```

Boundary Conditions and Cell Structure

In order to impose boundary conditions we make use of *ghost cells.* These cells are *not* updated using the equations of motion, but rather are set according to the specific boundary conditions that we wish to impose. The boundary conditions that we impose are a first order

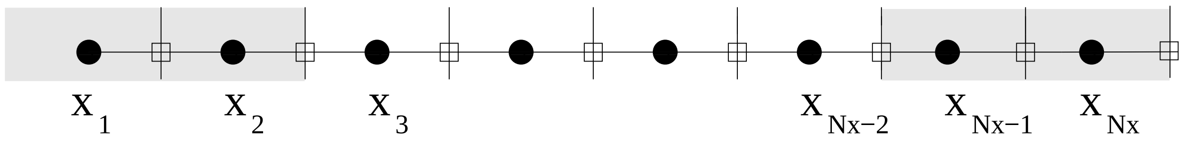

Figure 2: Spatial cell structure for a grid with N_x cells and two ghost cells ($N_g = 2$) per boundary. The solid circles lie at the spatial locations of the grid cell centres, and coincide with the grid points generated by the RNPL code. Shaded areas represent ghost cells where dynamical variables are updated according to the boundary conditions we impose. The squares and vertical lines denote the cell boundaries, and are the locations at which the reconstructed variables and numerical fluxes are calculated.

approximation to outgoing boundary conditions (often called *outflow conditions* in the fluid literature). We implement these conditions simply by setting the ghost cell values to the value in the last regular cell:

$$q_1 = q_3 \tag{51}$$

$$q_2 = q_3 \tag{52}$$

$$q_{N_x-1} = q_{N_x-2} \tag{53}$$

$$q_{N_x} = q_{N_x-2} \tag{54}$$

Here N_x is the number of cells in the entire grid (including ghost cells).

Note that in order to update the interior points, i.e. the x_i with $i = N_g + 1, ..., N_x - N_g$ we need to calculate the numerical fluxes at positions $x_{i+1/2}$ with $i = N_g, ..., N_x - N_g$.

Finite Difference

Due to the simplicity of equation (44), it is also straightforward to solve using a finite difference approximation that uses an *upwind stencil* (an upwind stencil is one which uses information only a specific characteristic direction, relative to the point at which the approximation is applied). Interestingly, however, if we discretize Burger's equation in the form (44) using such a technique

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x} q_i^{n+1/2} \left(q_i^{n+1/2} - q_{i-1}^{n+1/2} \right), \tag{55}$$

we find that the shock speeds obtained are erroneous, even in the continuum limit. In this instance, the problem can be solved by discretizing the conservative form of the equation, (45):

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x} \left[\frac{1}{2} \left(q_i^{n+1/2} \right)^2 - \frac{1}{2} \left(q_{i-1}^{n+1/2} \right)^2 \right]. \tag{56}$$

This latter discretization gives the proper shock velocities in the continuum limit. We also note that both (55) and 56) have first order spatial truncation error.

Finally, the use of a second order centred spatial discretization such as

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{4\Delta x} \left[3 \left(q_i^{n+1/2} \right)^2 - 4 \left(q_{i-1}^{n+1/2} \right)^2 + \left(q_{i-2}^{n+1/2} \right)^2 \right]. \tag{57}$$

generically introduces oscillations close to the shock front.

Problem 2a) Download the `burgers` code from the Computational Lab web page. This code solves equation (44) using the following methods:

1. upwind 1st order finite difference discretization in non-conservation form (discretization (55)): grid function `q_nc`
2. upwind 1st order finite difference discretization in conservation form (discretization (56)): grid function `q_c`,
3. centred 2nd order finite difference discretization in conservation form (discretization (57)): grid function `q_2c`
4. a finite volume approximation using the Roe flux: grid function `q`.

Scan through the `RNPL` source code, as well as the various `Fortran` source files, to get a sense for how the program implements the pseudo-code given in Sec. 4

Run the code for discontinuous initial data, `ini_type=1` in `id0`, with $q^L > q^R$, and using `xvs`, compare the solutions computed using the different discretizations. (Note: The parameter file `id0` contains two lines that initialize `ini_type`. Ensure that whichever line is not needed for the desired initialization is "commented out", i.e. has a `#` character at the beginning of the line. Also be careful to remove any initial state files (e.g. `in0.sdf`) prior to each run of the program.)

Download the `burgers_exact` code from the lab page. This code generates exact solutions of the Riemann problem for Burger's equation that can be used to check the numerical results generated in Problems 2b) and 2c).

Problem 2b) In the rest of the exercises on Burger's equation, we will focus on solutions computed using the finite volume approach, i.e. on the grid function `q`. We will first consider initial conditions for q, such that `ini_type=1` and $q^L > q^R$. Evolve and compute the shock speed for different values of q^L in the range, $0.5 \leq q^L \leq 1.0$, maintaining $q^R = \text{const.} = 0.1$. Explain how the the shock speed varies as a function of q^L. In order to do this, you may find it useful to plot the shock speeds as a function of q^L. Your plot should be approximately a straight line, with slope 1/2 and y intercept $q^R/2$. This is an example of a more general result, called the Rankine-Hugoniot jump condition (see LeVeque [5]):

$$s = \frac{[F]}{[q]} \tag{58}$$

Here, s is the speed of the shock, and $[F]$ and $[q]$ are the jumps in the physical flux and fundamental dynamical variable, respectively, across the discontinuity.

Problem 2c) When setting initial conditions such that $q^L < q^R$, the evolution is quite different. Assuming that at $t = 0$ the discontinuity is at $x = 0$, the exact solution is (see LeVeque [5]):

$$q(t,x) = \begin{cases} q^L & x < q^L\,t \\ x/t & q^L\,t < x < q^R\,t \\ q^R & x > q^R\,t \end{cases} \tag{59}$$

Run the code with `ini_type=1` and $q^L < q^R$ and compare with the exact solution generated by `burgers_exact`. This type of solution is called a rarefaction wave.

Problem 2d) Even if one starts with smooth initial data, evolution via Burger's equation generically produces discontinuities. Setting `ini_type=2`, which produces gaussian initial data of the following form:

$$q(t=0,x) = A \exp\left(-(x - x0)^2/\Delta^2\right), \tag{60}$$

try to estimate the time at which a shock is produced. By studying the characteristics of this equation it can be shown (see LeVeque [5]) that the time at which the shock forms is

$$T = \frac{-1}{\min\{q(t=0,x)'\}}, \tag{61}$$

where the prime denotes $\partial/\partial x$.

Perform a convergence test with the code using gaussian initial data, and observe how convergence is only first order in the vicinity of the maximum of `q`. As noted previously, this behaviour is a result of the reconstruction algorithm, which is only first order accurate where the dynamical variable has an extremum. *Note that you will have to use the "zoom" facility of* `xvs` *in order to see the convergence behaviour clearly.*

PROBLEM 2e) Write a code that solves the equations of motion (10) and (11) for a fluid with an ultrarelativistic equation of state using a finite volume approximation and a Roe solver to calculate the fluxes. We recommend use of the `burgers` code as a template. In the following we explain some of the modifications needed to each subroutine of `burgers` in the treatment of the ultrarelativistic fluid.

ultra_rnpl: Copy the file `burgers_rnpl` to `ultra_rnpl`. The following parameters will need to be added to the RNPL code in order to describe the initial data:

```
rho_xc, rho_R,    rho_L
v_xc,   v_R,      v_L
rho_0,  rho_amp,  rho_delta
v_0,    v_amp,    v_delta
```

The meaning of the above should be clear via comparison with the corresponding variables for the Burger's equation code. In addition, new floating-point parameters that characterize the fluid are needed:

```
Gamma, floor
```

as are the following grid functions:

```
rho,  v,   P,    tau,   S  defined on 2 time steps
FF1,  FF2                  defined on 1 time step.
```

The code in `step.inc` does the UPDATE of *all* the variables, i.e. it updates `rho`, `v`, `P`, `tau` and `S`. Modify the `UPDATES` statement in `ultra_rnpl` to reflect the addition of the new grid functions and parameters, noting in particular that in addition to the functions being evolved we need to pass the flux functions `FF1`, `FF2` as well as the fluid parameters `Gamma`, `floor`.

Similarly, modify the `INITIALIZE` statement to properly interface with the initialization code fragment `init_fluid.inc`, explained next.

init_fluid.inc: Implement code in `init_fluid.inc` that does the initialization of the fluid variables as `init_q.inc` does for `burgers`, and note that you will now need to initialize 5 grid functions. Your code should first initialize the primitive variables as follows

```
      if( ini_type .eq. 1 ) then
!        Initialize rho and v to Riemann problem
      else if( ini_type .eq. 2 ) then
!        Initialize rho and v to Gaussian pulse
      end if
```

Following this initialization, the code should then use equations (4), (7) and (8) to compute `P`, `tau` and `S`. Note that since `P` and `rho` are trivially related via the EOS (4), it would be possible to write code that eliminates one or the other. However, it is convenient, particularly in view of extending the code to more realistic EOSs, to keep both.

step.inc: Several modifications need to be made here. The calls to `calc_flux`, `update_q` and `update_boundary` will need to be modified to use the appropriate argument calling sequences (i.e. the routines will generally have different numbers and types of arguments). Also, *whenever* conservative variables are computed (even at the half step) it is useful to floor them using equation (43). Finally, the primitive variables also need to be re-calculated whenever the conservative variables are updated, and *after* the latter have been floored.

calc_flux.f: In order to calculate the numerical fluxes, we must compute the Jacobian, $\mathbf{A}$, which is now a 2×2 matrix. $\mathbf{A}$ has 2 eigenvalues λ_+ and λ_- and 2 two-component eigenvectors, $\mathbf{r}_+$ and $\mathbf{r}_-$. The numerical flux vector also has two components that are stored in the grid functions `FF1` and `FF2`.

First, reconstruct the primitive variables, `p1` and `p2`, on both the right and left sides of $x_{i+1/2}$. Then call the routine `calc_cons` (explained below), to compute the conservative variables and the pressure. Remember to floor the conservative variables following this call.

As with the code in `step.inc`, the argument lists of most of the routines called in `calc_flux.f` will need to be modified. Note that `recos_qL.f` and `recos_qR.f` do *not* need to be altered. However, they each will need to be called twice, once for each primitive variable.

When the numerical fluxes have been calculated at point $x_{i+1/2}$, store the values in the grid functions `FF1` and `FF2`.

calc_cons.f: Calculates the conservative variables $\{\tau, S\}$ from equations (7) and (8), and computes the pressure P from the EOS (4).

calc_prim.f: This is a new subroutine that computes the primitive variables ρ and v, and the pressure P from the conservative variables and Γ. Use equation (40), (41) and (42) for these calculations.

calc_A.f: Modify this subroutine to evaluate the Jacobian matrix using equations (31). Take into account that $A^\alpha{}_\beta$ is now a 2×2 matrix.

calc_lambda.f: Modify to use equations (34) to compute the 2 eigenvalues from the values $A^\alpha{}_\beta$.

calc_eta.f: Modify to use equations (35) to compute the 2 right eigenvectors from the values $A^\alpha{}_\beta$ and the eigenvalues.

calc_f.f: Modify to calculate the physical fluxes given by equation (12) from the reconstructed values of the primitive and conservative variables.

calc_omega.f: Modify to use equations (36) to calculate the jumps in the characteristic variables. Note that there are now two distinct jumps.

calc_FF.f: Modify to use the characteristic values and the physical fluxes calculated from the right and left reconstructed values to calculate the numerical fluxes.

update_q.f: Modify to update both conservative variables.

update_boundary.f: Modify to update both conservative variables.

id0: The parameter file needs to be modified to include the new parameters used in initialization of the fluid, and should also set values for `floor` and `Gamma`. For development purposes we suggest that you use `Gamma` $= 1.5$, and start with `floor` $= 10^{-11}$. Note, however, that some experimentation with the floor value may be necessary to achieve stability. Also note that it is not advisable to use initial data for a Riemann problem that has `rho_L = 0` or `rho_R = 0`.

In order to test your implementation's solution of a Riemann problem, you can download the code `ultra_exact` from the Computational Lab web page—`ultra_exact` provides an exact solution of the Riemann problem for an ultrarelativistic fluid, with a few restrictions. Specifically, the program can only solve a single Riemann problem problem for the range $x \in (-1, 1)$, with the discontinuity initially at $x = 0$ and with `rho_L` $>$ `rho_R`.

At a minimum, you should complete the following calculations:

1. **Convergence test:** Choosing smooth initial data (e.g. a Gaussian profile in `rho`, zero initial velocity), demonstrate that your code is second order accurate, except in the vicinity of extrema in the dynamical variables, and until a shock forms.

2. **Shock tube:** Set the following initial conditions

   - `rho_L = 1.0`
   - `rho_R = 0.1`
   - `v_R = v_L = 0.0`

   Run the code and notice the generic features that develop in a shock tube test: the left state, the right state, the intermediate state, the shock wave, and the rarefaction fan. Compare your solution with that generated using `ultra_exact` at different resolutions. What can you say about the convergence of your code? What can you say about the resolution of the shock front?

   Using the Rankine-Huginot conditions (58) for both conservative variables, τ and S, and the corresponding fluxes, calculate the expected shock speed, and compare to the speed deduced from your numerical results.

   For the current problem, the conservative variables in any rarefaction wave have self-similar profiles, i.e. they can be expressed as a function of the single variable x/t. Note that if $f \equiv f(x/t)$, then

$$x\frac{\partial f}{\partial x} + t\frac{\partial f}{\partial t} = 0 \tag{62}$$

Add to your RNPL code a grid function that stores the residual of the above quantity (where f is any of the dynamical variables), and attempt to verify that this residual tends to 0, in the continuum limit, in the rarefaction region.

3. **How relativistic is relativistic?:** Using shock tube data of the form

   - `rho_R = 0.1`
   - `v_R = v_L = 0.0`

   vary `rho_L` in an attempt to determine the maximum Lorentz factor, W, that your code can attain. Describe what ultimately happens to your code in this process (i.e. does it "break", and if so, what appears to be the difficulty, or difficulties).

Give yourself a BIG pat on the back!

Appendix

We consider a scalar, linear equation of the form:

$$\dot{q} + v q' = 0, \tag{63}$$

where the velocity, v, is now a *constant* (i.e. independent of q). We take initial data for a a Riemann problem, analagous to the ones we must solve in order to calculate the Roe numerical flux (20) (see Fig. 3). The solution of this particular Riemann problem is simple

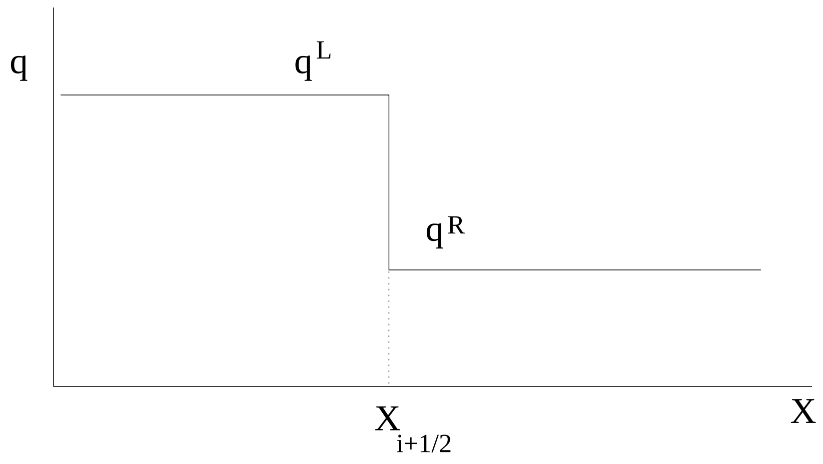

Figure 3: Initial data.

since we know that the characteristics of equation (63) are $x - vt = k$, where k is a constant. Therefore the solution $q^\star$ of the problem at $x = x_{i+1/2}$ can be written as:

$$q^* = \begin{cases} q_L + \omega & \text{if} \quad v < 0 \\ q_R - \omega & \text{if} \quad v > 0 \end{cases} \tag{64}$$

where $\omega = q_R - q_L$. Now if we consider (63) as a linearization of a non-linear scalar equation then the numerical flux is $F_{i+1/2} = vq^\star$, and using (64):

$$F_{i+1/2} = \frac{1}{2}\left[f\left(q^L\right) + f\left(q^R\right) - |v|\,\omega\right]\,. \tag{65}$$

In the above $f(q)$ is the physical flux for which vq is the linearization. In the case we do have a *system* of equations, we perform a rotation in variable space that diagonalizes the Jacobian. Once the Jacobian has been diagonalized, the problem reduces to a system of decoupled scalar equations, each of which can can solved independently as above. It is for this reason that the eigenvectors appear in formula (20).

References

[1] J.M. Marti, and E. Mueller, "Numerical Hydrodynamics in Special Relativity", Living Rev. Relativity 2, (1999), 3. [Online article]: cited on 25 Jun 2003 http://www.livingreviews.org/lrr-1999-3

[2] J. Font, "Numerical Hydrodynamics in General Relativity", Living Rev. Relativity 3, (2000), 2. [Online article]: cited on 25 Jun 2003 http://www.livingreviews.org/lrr-2000-2

[3] C.W. Misner, K.S. Thorne, J.A. Wheeler, "Gravitation". W.H. Freeman, San Francisco, 1973.

[4] D.W. Neilsen, and M.W. Choptuik, Class. Quant. Grav. **17**, 733 (2000) [arXiv:gr-qc/9904052].

[5] R.J. LeVeque "Numerical Methods for Conservation Laws". Lecture in Mathematics: ETH Zürich. Birkhäuser Verlag, Boston, 1992. (see also http://www.amath.washington.edu/˜rjl/booksnotes.html)