Mesh Based Numerical Hydrodynamics of Ideal Gases

The Used and Implemented Equations

Mladen Ivkovic (mladen.ivkovic@hotmail.com)

Contents

1 General Implementation Details

This section gives an overview on how this code works. To read about how to use the code, see the `/README.md` file.

1.1 How the program works

- All the source files are in the `/program/src/` directory.
- The main program files are `/program/src/main.c` if you want to use the code as a hydro/hyperbolic conservation law solver, or `/program/src/main-riemann.c` if you want to use the code as a Riemann solver.
- The program starts off by reading in the initial conditions (IC) file and the parameter file, which are both required command line arguments when executing the program. All functions related to reading and writing files are in `/program/src/io.c`
- Then a few checks are done to make sure no contradictory or missing parameters are given, and the grid on which the code will work on is initialized, as well as some global variables like the step number and current in-code time.
- There are two global variables used throughout the code:
  - `struct params param`: A parameter struct (defined in `/program/src/params.h`) that stores global parameters that are useful to have everywhere throughout the code (even if it isn't optimal coding practice...) All parameter related functions are in `/program/src/params.c`.
  - `struct cell* grid` or `struct cell** grid`, depending on whether you run the code in 1D or 2D, respectively. It's an array of `struct cell`, defined in `/program/src/cell.h`, which is meant to store all cell related quantities: primitive states, `prim`, as `struct pstate`, and conserved states, `cons`, as `struct cstate`. Furthermore they have `struct pstate pflux` and `struct cstate cflux` to store fluxes of primitive and conserved variables. All cell/grid related functions are written in `/program/src/cell.c`.
- If the code is used as a hydro/conservation law solver:
  - The main loop starts now:

  - Advance the solver for a time step.

  - Write output data if you need to (when to dump output is specified in the parameter file). All functions related to reading and writing files are in `/program/src/io.c`

  - Write a message to the screen after the time step is done. This includes the current step number, in-code time, the ratio of initial mass to the current mass on the entire grid, and the wall-clock time that the step needed to finish.

- If the code is used as a Riemann solver:

  - Only do a loop over all cells and solve the Riemann problem given by the IC at the cell's position at given time t. Some functions for the Riemann problem that are used by all implemented Riemann solvers are given in `/program/src/riemann.h` and `/program/src/riemann.c`. When a specific Riemann solver is set, `/program/src/riemann.h` includes that file from the directory `/program/src/riemann/`. The file names in that directory should be obvious.

  - Write output data. All functions related to reading and writing files are in `/program/src/io.c`

1.2 Contents of specific files and directories in `/program/src/`

- `/program/src/cell.c`, `/program/src/cell.h`:

  All cell/grid related functions. The grid is used as `struct cell* grid` or `struct cell** grid`, depending on whether you run the code in 1D or 2D, respectively.

  It's an array of `struct cell`, defined in `/program/src/cell.h`, which is meant to store all cell related quantities: primitive states, `prim`, as `struct pstate`, and conserved states, `cons`, as `struct cstate`. Furthermore they have `struct pstate pflux` and `struct cstate cflux` to store fluxes of primitive and conserved variables. All cell/grid related functions are written in `/program/src/cell.c`.

- `/program/src/defines.h`:

  Contains all macro definitions, like iteration tolerance, the box size, the adiabatic coefficient γ etc, as well as some physical constants (mostly related to γ).

- `/program/src/io.h`, `/program/src/io.c`:

  All input/output related functions, i.e. anything related to reading and writing files.

- `/program/src/limiter.h`, `/program/src/limiter.c`:

  Slope and flux limiter related functions (section 7) that are used regardless of the choice of the limiter. For a specific choice of slope limiter, `/program/src/limiter.h` includes a specific file from `/program/src/limiter/`. The file name in `/program/src/limiter/` should be obvious.

- `/program/src/limiter/`:

  Slope limiter functions (section 7) for specific limiters. They will be included by `/program/src/limiter.h` during compile time by setting the corresponding variable name in the `/program/bin/Makefile`.

  Essentially, these files only contain the actual computation of $\phi(r)$ and $\xi(r)$.

- `/program/src/main.c`:

  The main function of the program when the program is utilized as a hydro/hyperbolic conservation law solver.

- `/program/src/main-riemann.c`:

  The main function of the program when the program is utilized as a Riemann solver.

- `/program/src/riemann.h`, `/program/src/riemann.c`:

  Riemann solver related functions (section 4) that are used regardless of the choice of the Riemann solver. For a specific choice of Riemann solver, `/program/src/riemann.h` includes a specific file from `/program/src/riemann/`. The file name in `program/src/riemann/` should be obvious.

- `/program/src/riemann/`:

  Riemann solver functions (section 4) for specific Riemann solvers. They will be included by `/program/src/riemann.h` during compile time by setting the corresponding variable name in the `/program/bin/Makefile`.

  Essentially, these files contain only the specific function to get the star state pres-

sure and contact wave velocity. The only exception is the HLLC solver, which works a bit differently than the other implemented solvers. There, essentially everything needs to be done in a special way, so the solver contains its own routines, with a "`HLLC`" added to the function names.

- `/program/src/solver.h`, `/program/src/solver.c`:

  Hydro and advection solver related functions (section 5, 6) that are used regardless of the choice of the hydro solver. For a specific choice of solver, `/program/src/solver.h` includes a specific file from `/program/src/solver/`. The file name in `/program/src/solver/` should be obvious. For implementation details of each solver, look up the implementation details in their respective section 5, 6.

- `/program/src/solver/`:

  Hydro and advection solver functions (section 5, 6) for specific solvers. They will be included by `/program/src/solver.h` during compile time by setting the corresponding variable name in the `/program/bin/Makefile`. For implementation details of each solver, look up the implementation details in their respective section 5, 6.

- `/program/src/utils.h`, `/program/src/utils.c`:

  Miscellaneous small utilities that are irrelevant for the actual hydro or hyperbolic conservation law solving, like printing a banner every time the code starts, standardized functions to print outputs to screen or throw errors, etc.

1.3 Compilation Flags

1.3.1 Intended to be set by the user

variable	options	description
EXEC	some_name	resulting executable name. Can be set as you wish.
NDIM	1	Number of dimensions
	2	
SOLVER	*Which conservation law solver to use. Two classes are implemented: To solve advection (section 5) and Euler equations (section 6).*	
	ADVECTION_PWCONST	Piecewise constant advection (section 5.2)
	ADVECTION_PWLIN	Piecewise linear advection (section 5.3)
	ADVECTION_WAF	Weighted Average Flux advection (section 5.4)
	GODUNOV	Godunov Upwind first order hydrodynamics (section 6.1)
	WAF	Weighted Average Flux hydrodynamics (section 6.2)
	MUSCL	MUSCL-Hancock hydrodynamics (section 6.3)
RIEMANN	*Riemann solvers (section 4) are used in combination with hydro solvers for the Euler equations. Advection solvers don't need them.*	
	EXACT	Exact Riemann solver (section 4.3)
	HLLC	HLLC (section 4.4.3)
	TRRS	Two Rarefaction Riemann solver (section 4.4.1)
	TSRS	Two Shock Riemann solver (section 4.4.2)
LIMITER	*Flux/slope limiters (section 7) are needed for all methods higher than first order to avoid unphysical oscillations that arise from the numerical schemes.*	
	MC	Monotoniced Center Difference limiter (section 7.6)
	MINMOD	Minmod limiter (section 7.6)
	SUPERBEE	Superbee limiter (section 7.6)
	VANLEER	Van Leer limiter (section 7.6)
SOURCES	*Source Terms (section 10) from external forces. Can be left undefined.*	
	NONE	No source terms
	CONSTANT	Constant source terms (section 10.1)
	RADIAL	Radial constant source terms (section 10.1)
INTEGRATOR	*Which integrator to use to integrate the source terms (section 10) from external forces in time. Can (and should) be left undefined if sources are NONE or undefined.*	
	NONE	No source terms
	RK2	Runge-Kutta 2 integrator (section 10.2)
	RK4	Runge-Kutta 4 integrator (section 10.2)

1.3.2 Behind the Scenes

It's simpler to pass on an integer as a definition `-Dsomedefinition` than any kind of string, in particular if you want to do comparisons like `#if SOLVER == GODUNOV` inside the code. Hence all choices are being translated to integers behind the scenes. Here is the list of translations:

variable	option	integer
SOLVER	ADVECTION_PWCONST	11
	ADVECTION_PWLIN	12
	ADVECTION_WAF	13
	GODUNOV	21
	WAF	22
	MUSCL	23
RIEMANN	NONE	0
	EXACT	1
	HLLC	2
	TRRS	3
	TSRS	4
LIMITER	NONE	0
	MINMOD	1
	SUPERBEE	2
	VANLEER	3
	MC	4
SOURCES	NONE	0
	CONSTANT	1
	RADIAL	2
INTEGRATOR	NONE	-
	RK2	-
	RK4	-

The `INTEGRATOR` definition requires no checks, it only determines which files will be included and compiled, so no integers are assigned.

Some other definitions for checks are required. These definitions are automatically identified and passed to the compiler, so no additional action from the user's side is required.

- If any advection solver is used, the code needs an additional `-DADVECTION` flag
- If any source terms are used, the code needs an additional `-DWITH_SOURCES` flag

All these things are handled in the `/program/bin/processing.mk` file, as well as checks whether all required variables are defined, and if not, default values are set.

The default values are:

variable	default value
SOLVER	GODUNOV
RIEMANN	EXACT
LIMITER	NONE
SOURCES	NONE

1.4 Tests

Check whether the code works as intended

Code Coverage

2 Ideal Gases

2.1 Governing Equations

We are mostly going to concern ourselves with ideal gases, which are described by the Euler equations:

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho\mathbf{v}\\ E\end{pmatrix} + \nabla\cdot\begin{pmatrix}\rho\mathbf{v}\\ \rho\mathbf{v}\otimes\mathbf{v}+p\\ (E+p)\mathbf{v}\end{pmatrix} = \begin{pmatrix}0\\ \rho\mathbf{a}\\ \rho\mathbf{a}\mathbf{v}\end{pmatrix} \tag{1}$$

Where

- ρ: fluid density
- $\mathbf{v}$: fluid (mean/bulk) velocity at a given point. I use the notation $\mathbf{v} = (u, v, w)$, or when indices are useful, $\mathbf{v} = (v_1, v_2, v_3)$
- p: pressure
- E: specific energy. $E = \frac{1}{2}\rho\mathbf{v}^2 + \rho\varepsilon$, with ε = specific internal thermal energy
- $\mathbf{a}$: acceleration due to some external force.

The outer product $\cdot\otimes\cdot$ gives the following tensor:

$$(\mathbf{v}\otimes\mathbf{v})_{ij} = \mathbf{v}_i\mathbf{v}_j \tag{2}$$

Furthermore, we have the following relations for ideal gasses:

$$p = nkT \tag{3}$$

$$p = C\rho^\gamma \qquad \text{entropy relation for smooth flow, i.e. no shocks} \tag{4}$$

$$s = c_V \ln\left(\frac{p}{\rho^\gamma}\right) + s_0 \qquad \text{entropy} \tag{5}$$

$$a = \sqrt{\left.\frac{\partial p}{\partial\rho}\right|_s} = \sqrt{\frac{\gamma p}{\rho}} \qquad \text{sound speed} \tag{6}$$

with

- n : number density
- k : Boltzmann constant
- T : temperature
- s : entropy
- γ: adiabatic index
- c_V: specific heat

and the Equation of State

$$\varepsilon = \frac{1}{\gamma - 1}\frac{p}{\rho} \tag{7}$$

The Euler equations can be written as a conservation law as

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0 \tag{8}$$

where we neglect any outer forces, i.e. $\mathbf{a} = 0$, and

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho\mathbf{v} \\ E \end{pmatrix}, \qquad \mathbf{F}(\mathbf{U}) = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v} \otimes \mathbf{v} + p \\ (E + p)\mathbf{v} \end{pmatrix} \tag{9}$$

2.1.1 Euler equations in 1D

In 1D, we can write the Euler equations without source terms ($\mathbf{a} = 0$) as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} = 0 \tag{10}$$

or explicitly (remember $\mathbf{v} = (u, v, w)$)

$$\frac{\partial}{\partial t}\begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E+p)u \end{pmatrix} = 0 \tag{11}$$

2.1.2 Euler equations in 2D

In 2D, we have without source terms ($\mathbf{a} = 0$)

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = 0 \tag{12}$$

or explicitly (remember $\mathbf{v} = (u, v, w)$)

$$\frac{\partial}{\partial t}\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E+p)u \end{pmatrix} + \frac{\partial}{\partial y}\begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E+p)v \end{pmatrix} = 0 \tag{13}$$

2.2 Conserved and primitive variables

For now, we have described the Euler equation as a hyperbolic conservation law using the (conserved) state vector $\mathbf{U}$:

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho \mathbf{v} \\ E \end{pmatrix} \tag{14}$$

for this reason, the variables ρ, $\rho\mathbf{v}$, and E are referred to as "conserved variables", as they obey conservation laws.

However, this is not the only set of variables that allows us to describe the fluid dynamics. In particular, the solution of the Riemann problem (section 4) will give us a set of with so called "primitive variables" (or "physical variables") with the "primitive" state vector $\mathbf{W}$

$$\mathbf{W} = \begin{pmatrix} \rho \\ \mathbf{v} \\ p \end{pmatrix} \tag{15}$$

Using the ideal gas equations, these are the equations to translate between primitive and conservative variables:

Primitive to conservative:

$$(\rho) = (\rho) \tag{16}$$
$$(\rho\mathbf{v}) = (\rho) \cdot (\mathbf{v}) \tag{17}$$
$$(E) = \frac{1}{2}(\rho)(\mathbf{v})^2 + \frac{(p)}{\gamma - 1} \tag{18}$$

Conservative to primitive:

$$(\rho) = (\rho) \tag{19}$$
$$(\mathbf{v}) = \frac{(\rho\mathbf{v})}{(\rho)} \tag{20}$$
$$(p) = (\gamma - 1)\left((E) - \frac{1}{2}\frac{(\rho\mathbf{v})^2}{(\rho)}\right) \tag{21}$$

2.3 Implementation Details

All the functions for computing gas related quantities are written in `gas.h` and `gas.c`. Every cell is represented by a struct `struct cell` written in `cell.h`. It stores both the

primitive variables/states and the conservative states in the `pstate` and `cstate` structs, respectively.

The adiabatic index γ is hardcoded as a macro in `defines.h`. If you change it, all the derived quantities stored in macros (e.g. $\gamma-1$, $\frac{1}{\gamma-1}$) should be computed automatically.

3 Notation

We are working on numerical methods. Both space and time will be discretized.

Space will be discretized in cells which will have integer indices to describe their position. Time will be discretized in fixed time steps, which may have variable lengths. Nevertheless the time progresses step by step.

The lower left corner has indices $(0, 0)$ in 2D. In 1D, index 0 also represents the leftmost cell.

We have:

- integer subscript: Value of a quantity at the cell, i.e. the center of the cell. Example: $\mathbf{U}_i$, $\mathbf{U}_{i-2}$ or $\mathbf{U}_{i,j+1}$ for 2D.
- non-integer subscript: Value at the cell faces, e.g. $\mathbf{F}_{i-1/2}$ is the flux at the interface between cell i and $i-1$, i.e. the left cell as seen from cell i.
- integer superscript: Indication of the time step. E.g. $\mathbf{U}^n$: State at timestep n
- non-integer superscript: (Estimated) value of a quantity in between timesteps. E.g. $\mathbf{F}^{n+1/2}$: The flux at the middle of the time between steps n and $n+1$.

4 Riemann Problem and Solvers

4.1 The Riemann Problem and Solution Strategy

At the heart of Eulerian (not-comoving) numerical fluid dynamics is the solution to a specific initial value problem (IVP) called the "Riemann problem". For a hyperbolic system of conservation laws of the form

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U}))}{\partial x} = 0 \tag{22}$$

the Riemann problem is defined as

$$\mathbf{U}(x, t = 0) = \begin{cases} \mathbf{U}_L & \text{if } x < 0 \\ \mathbf{U}_R & \text{if } x > 0 \end{cases} \tag{23}$$

see also fig. 1.

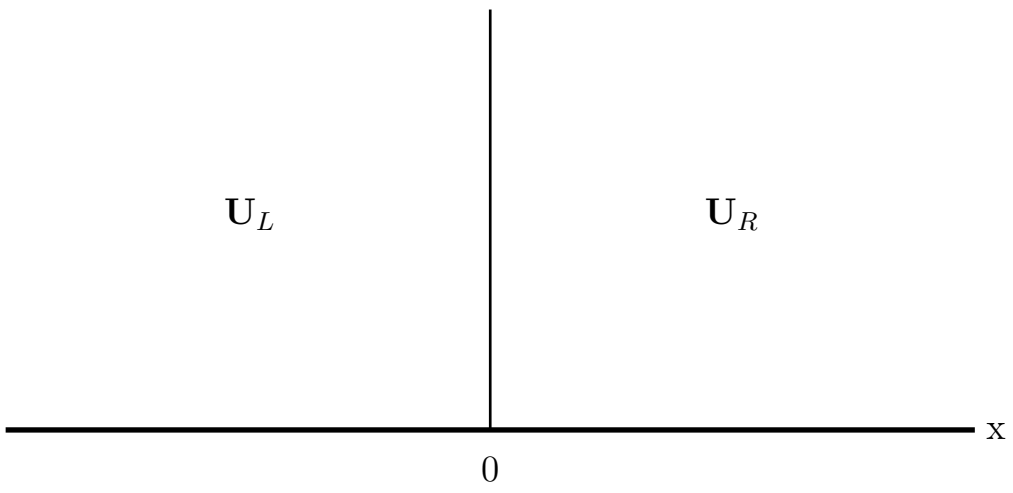

Figure 1: The Riemann Initial Value Problem in 1D.

Unfortunately, there is no exact closed-form solution to the Riemann problem for the

Euler equations. However, it is possible to devise iterative schemes whereby the solution can be computed numerically. To solve full fluid dynamics problems, this calculation needs to be repeated many many times, making the solution quite expensive. For that reason, people have developed approximate Riemann solvers, which we also will have a look at.

For a full derivation of how to solve the Riemann problem for the Euler equations, see e.g. Toro [1999]. For our purposes, it suffices to accept that (assuming we have no vacuum) as time progresses, three waves will form which will separate the two initial states $\mathbf{U}_L$ and $\mathbf{U}_R$. This results in two new states, $\mathbf{U}_L^*$ and $\mathbf{U}_R^*$ between the initial states, because $\mathbf{U}_L^*$ and $\mathbf{U}_R^*$ themselves will be separated by a wave. This is shown in figure 2

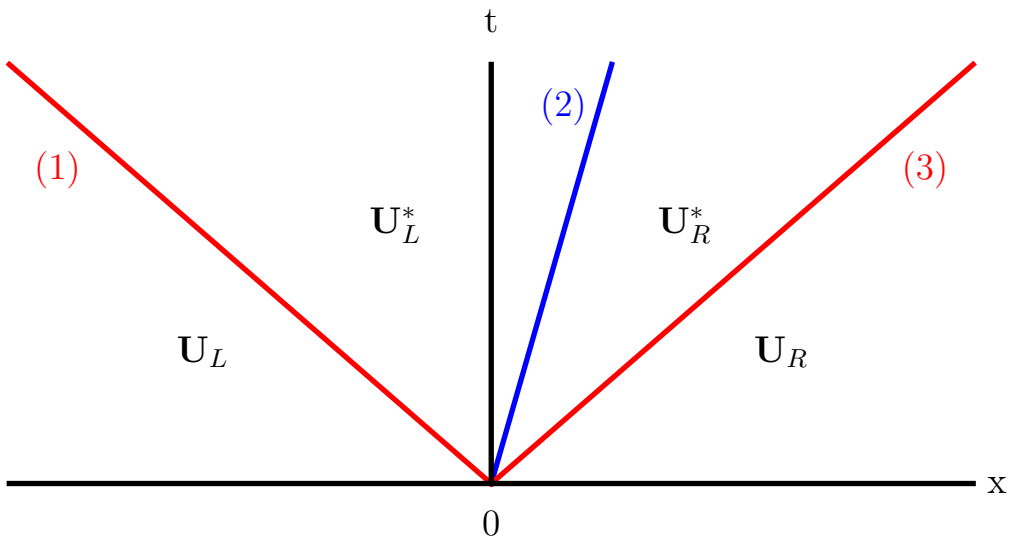

Figure 2: The solution to the Riemann problem for Euler equations: Three waves, (1), (2), and (3), arise from the origin as time evolves. (2) is always a contact wave, (1) and (3) can be either rarefaction or shock waves in each case, depending on the initial conditions. The initial states $\mathbf{U}_L$ and $\mathbf{U}_R$ are separated through the waves (1) and (3) from the two new arising "star states" $\mathbf{U}_L^*$ and $\mathbf{U}_R^*$, which themselves are separated by the contact wave (2).

4.2 Wave types and relations

It turns out that we get three types of waves: A contact wave, a shock wave, and a rarefaction wave. The middle wave is always a contact wave. The left and right waves can be any combination of shock and/or rarefaction wave, depending on the initial conditions. A model problem containing all three waves is shown in figure 3. These are the wave properties:

4.2.1 Contact wave

The contact wave is a jump discontinuity in the density ρ only. Pressure and velocity remain constant across the contact wave. This gives us the relations

$$p_L^* = p_R^* = p^*$$
$$u_L^* = u_R^* = u^*$$

for this reason, the star state pressure and velocity will have no index indicating whether they are the left or right star state, and will be referred to as p^* and u^*, respectively.

4.2.2 Shock wave

All three primitive variables ρ, p, and u change across a shock wave. A shock wave is a jump discontinuity too. If the **leftmost** wave (wave (1) in fig. 2) is a shock wave, we have

$$\begin{aligned}
\rho_L^* &= \frac{\frac{p^*}{p_L} + \frac{\gamma-1}{\gamma+1}}{\frac{\gamma-1}{\gamma+1}\frac{p^*}{p_L} + 1}\rho_L \\
u^* &= u_L - \frac{p^* - p_L}{\sqrt{\frac{p^*+B_L}{A_L}}} \\
&= u_L - f_L(p^*) \\
A_L &= \frac{2}{(\gamma+1)\rho_L} \\
B_L &= \frac{\gamma-1}{\gamma+1}p_L
\end{aligned}$$

f_L is given in eq. 35, and the shock speed is

$$S_L = u_L - a_L\left[\frac{\gamma+1}{2\gamma}\frac{p^*}{p_L} + \frac{\gamma-1}{2\gamma}\right]^{1/2}$$

where a_L is the sound speed in the left state U_L.

For a **right shock wave**, i.e. when wave (3) is a shock wave, we have the relations

$$\begin{aligned}
\rho_R^* &= \frac{\frac{p^*}{p_R} + \frac{\gamma-1}{\gamma+1}}{\frac{\gamma-1}{\gamma+1}\frac{p^*}{p_R} + 1}\rho_R \\
u^* &= u_R + \frac{p^* - p_R}{\sqrt{\frac{p^*+B_R}{A_R}}} \\
&= u_R + f_R(p^*) \\
A_R &= \frac{2}{(\gamma+1)\rho_R} \\
B_R &= \frac{\gamma-1}{\gamma+1}p_R
\end{aligned}$$

and the shock speed is

$$S_R = u_R + a_R\left[\frac{\gamma+1}{2\gamma}\frac{p^*}{p_R} + \frac{\gamma-1}{2\gamma}\right]^{1/2}$$

where a_R is the sound speed in the left state U_L. f_R is given in equation 35.

4.2.3 Rarefaction wave

Rarefaction waves are smooth transitions, not infinitesimally thin jump discontinuities. This makes them really easy to spot in the solutions of Riemann problems, see fig. 3. Across rarefactions, entropy is conserved.

The rarefaction waves are enclosed by the head and the tail of the wave, between which we have a smooth transition which is called the "fan". The head is the "front" of the wave, i.e. the part of the wave that gets furthest away from the origin as time progresses. The tail is the "back" of the wave, i.e. the part of the wave that stays closest to the origin as time progresses. The wave speeds of the head, S_H, and of the tail, S_T, are both given below.

If we have a **left-facing** rarefaction, i.e. if wave (1) is a rarefaction wave, we have

$$
\begin{aligned}
\rho_L^* &= \rho_L \left(\frac{p^*}{p_L}\right)^{\frac{1}{\gamma}} \\
u^* &= u_L - \frac{2a_L}{\gamma - 1}\left[\left(\frac{p^*}{p_L}\right)^{\frac{\gamma-1}{2\gamma}} - 1\right] \\
&= u_L - f_L(p^*)
\end{aligned}
$$

f_L is given in eq. 35, and a_L is the sound speed in the left state U_L.

The wave speeds of the head, S_H, and the tail, S_T, for the left facing wave are

$$
\begin{aligned}
S_{HL} &= u_L - a_L \\
S_{TL} &= u^* - a_L^* \\
a_L^* &= a_L \left(\frac{p^*}{p_L}\right)^{\frac{\gamma-1}{2\gamma}}
\end{aligned}
$$

Finally, the solution inside the rarefaction fan, i.e. in regions where $S_{HL} \leq \frac{x}{t} \leq S_{TL}$, is

$$
\rho_{\text{fan},L} = \rho_L \left[\frac{2}{\gamma+1} + \frac{\gamma-1}{\gamma+1}\frac{1}{a_L}\left(u_L - \frac{x}{t}\right)\right]^{\frac{2}{\gamma-1}} \tag{24}
$$

$$
u_{\text{fan},L} = \frac{2}{\gamma+1}\left[\frac{\gamma-1}{2}u_L + a_L + \frac{x}{t}\right] \tag{25}
$$

$$
p_{\text{fan},L} = p_L \left[\frac{2}{\gamma+1} + \frac{\gamma-1}{\gamma+1}\frac{1}{a_L}\left(u_L - \frac{x}{t}\right)\right]^{\frac{2\gamma}{\gamma-1}} \tag{26}
$$

If we have a **right-facing rarefaction**, i.e. if wave (1) is a rarefaction wave, we have

$$
\begin{aligned}
\rho_R^* &= \rho_R \left(\frac{p^*}{p_R}\right)^{\frac{1}{\gamma}} \\
u^* &= u_R - \frac{2a_R}{\gamma - 1}\left[1 - \left(\frac{p^*}{p_R}\right)^{\frac{\gamma-1}{2\gamma}}\right]
\end{aligned}
$$

$$= u_R + f_R(p^*)$$

where a_R is the sound speed in the left state U_R. f_R is given in equation 35.

The wave speeds of the head, S_H, and the tail, S_T, for the left facing wave are

$$\begin{aligned} S_{HR} &= u_R + a_R \\ S_{TR} &= u^* + a_R^* \\ a_R^* &= a_R \left(\frac{p^*}{p_R}\right)^{\frac{\gamma-1}{2\gamma}} \end{aligned}$$

Finally, the solution inside the rarefaction fan, i.e. in regions where $S_{HL} \leq \frac{x}{t} \leq S_{TL}$, is

$$\rho_{\text{fan},R} = \rho_R \left[\frac{2}{\gamma+1} - \frac{\gamma-1}{\gamma+1}\frac{1}{a_R}\left(u_R - \frac{x}{t}\right)\right]^{\frac{2}{\gamma-1}} \tag{27}$$

$$u_{\text{fan},R} = \frac{2}{\gamma+1}\left[\frac{\gamma-1}{2}u_R - a_R + \frac{x}{t}\right] \tag{28}$$

$$p_{\text{fan},R} = p_R \left[\frac{2}{\gamma+1} - \frac{\gamma-1}{\gamma+1}\frac{1}{a_R}\left(u_R - \frac{x}{t}\right)\right]^{\frac{2\gamma}{\gamma-1}} \tag{29}$$

4.2.4 Which wave type do we have?

As written before, the middle wave (wave (2) in fig. 2) is always a contact wave, while the other two waves are any combination of rarefaction and/or shock wave. It turns out that the condition for a rarefaction or shock wave is remarkably simple.

For the left wave (wave (1)):

$$p^* > p_L : \qquad \text{(1) is a shock wave} \tag{30}$$

$$p^* \leq p_L : \qquad \text{(1) is a rarefaction wave} \tag{31}$$

and for the right wave (wave (3)):

$$p^* > p_R : \qquad \text{(3) is a shock wave} \tag{32}$$
$$p^* \leq p_R : \qquad \text{(3) is a rarefaction wave} \tag{33}$$

See Toro [1999] for more details.

4.2.5 Solution for p^*

The only thing missing to have a complete solution to the Riemann problem for the Euler equations is an expression how to obtain p^*, the pressure in the star region, depending on the initial conditions U_L and U_R. We make use of the fact that p^* and u^* are constant across the star region to relate U_L and U_R, or more precisely the primitive states $\mathbf{W}_L$ and $\mathbf{W}_R$ which can easily be derived from the conservative ones. For both shock and rarefaction waves on either side, we have equations for u^* depending on the outer states $\mathbf{W}_L$ and $\mathbf{W}_R$ and p^*. By setting $u_L^* - u_R^* = 0$, which must hold, we get the equation

$$f(p, \mathbf{W}_L, \mathbf{W}_R) \equiv f_L(p, \mathbf{W}_L) + f_R(p, \mathbf{W}_R) + (u_R - u_L) = 0 \tag{34}$$

with

$$f_{L,R} = \begin{cases} (p - p_{L,R}) \left[\frac{A_{L,R}}{p + B_{L,R}} \right]^{\frac{1}{2}} & \text{if } p > p_{L,R} \quad \text{(shock)} \\ \frac{2a_{L,R}}{\gamma - 1} \left[\left(\frac{p}{p_{L,R}} \right)^{\frac{\gamma-1}{2\gamma}} - 1 \right] & \text{if } p \leq p_{L,R} \quad \text{(rarefaction)} \end{cases} \tag{35}$$

$$A_{L,R} = \frac{2}{(\gamma + 1)\rho_{L,R}} \tag{36}$$

$$B_{L,R} = \frac{\gamma - 1}{\gamma + 1} p_{L,R} \tag{37}$$

4.3 Exact Solver

The equation for the pressure in the star region 35 can't be solved analytically, but it can be solved iteratively.

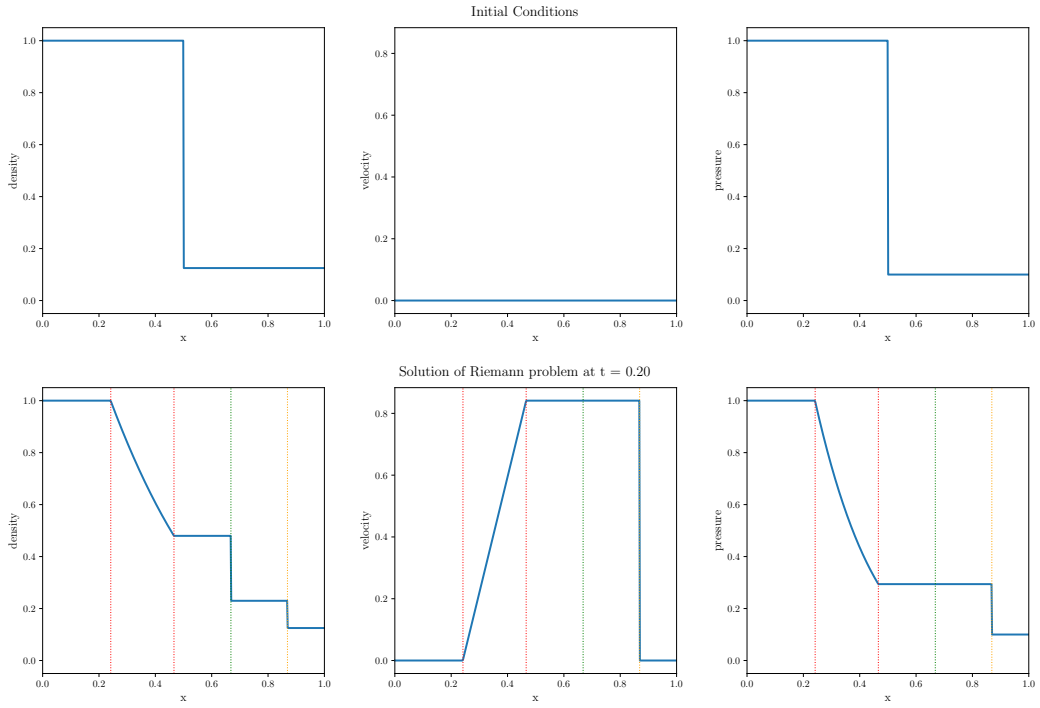

Figure 3: Top row: The initial conditions to a classical Riemann problem, called the Sod shock.

Bottom row: The exact solution of the problem at $t = 0.2$. The solution consists of a left facing rarefaction wave (between the two red dotted lines), easily recognisable through its non-linear shape. To the right (orange dotted line) is a shock wave, across which all three primitive quantities (density, pressure, bulk velocity) change as a jump discontinuity. The two waves enclose the third middle wave (green dotted line), which is a contact wave. The contact wave is a jump discontinuity just like a shock wave, but only the density changes; Velocity and pressure remain constant.

Since we have the analytic function and the first derivative w.r.t. p can be computed, i.e.

$$\frac{\partial f_{L,R}}{\partial p} = \begin{cases} \left[\frac{A_{L,R}}{p+B_{L,R}}\right]^{\frac{1}{2}} \left(1 - \frac{1}{2}\frac{p-p_{L,R}}{p+B_{L,R}}\right) & \text{if } p > p_{L,R} \quad \text{(shock)} \\ \frac{a_{L,R}}{\gamma p_{L,R}} \left(\frac{p}{p_{L,R}}\right)^{\frac{-(\gamma+1)}{2\gamma}} & \text{if } p \leq p_{L,R} \quad \text{(rarefaction)} \end{cases}$$

$$A_{L,R} = \frac{2}{(\gamma+1)\rho_{L,R}}$$

$$B_{L,R} = \frac{\gamma-1}{\gamma+1} p_{L,R}$$

Then, using the Newton-Raphson iteration, we can find the solution using the prescription

$$p_{n+1} = p_n - \frac{f(p_n)}{\frac{\partial f(p_n)}{\partial p}} \tag{38}$$

we re-iterate until it converges, i.e. when the relative pressure change

$$\frac{|p_k - p_{k+1}|}{\frac{1}{2}|p_k + p_(k+1)|} < \epsilon \tag{39}$$

where ϵ is some tolerance. Default value is set in `defines.h` as `#define EPSILON_ITER 1e-6`.

We need to find a first guess for the pressure. An ok way to do it is to take the average:

$$p_0^* = \frac{1}{2}(p_L + p_R)$$

The implemented way is based on a linearised solution based on primitive variables:

$$p_{PV} = \frac{1}{2}(p_L + p_R) - \frac{1}{8}(u_R - u_L)(\rho_L + \rho_R)(a_L + a_R)$$

$$p_0^* = \max(\epsilon, p_{PV})$$

Note that every step along the iteration, we must make sure that we didn't accidentally get negative pressures, and limit it to zero (or the tolerance ϵ). If it drops below zero, it might get stuck there, and then all hell breaks loose. (Seriously, you will get NANs because you're trying to take fractal powers of negative stuff.)

And that's it for the exact solver. All that is left to do is sample the solution, which is described in section 4.6.

4.4 Approximate Solvers

4.4.1 Two-Rarefaction Riemann Solver (TRRS)

The big idea is to assume a priori that both the left and right waves are going to be rarefaction waves, and to use that assumption to get an expression for p^* and u^*, the pressure and velocity in the star region, respectively.

We get

$$\beta \equiv \frac{\gamma - 1}{2\gamma} \tag{40}$$

$$u^* = \frac{\frac{2}{\gamma-1}\left[\left(\frac{p_L}{p_R}\right)^\beta - 1\right] + \frac{u_L}{a_L}\left(\frac{p_L}{p_R}\right)^\beta + \frac{u_R}{a_R}}{\frac{1}{a_R} + \frac{1}{a_L}\left(\frac{p_L}{p_R}\right)^\beta} \tag{41}$$

$$p^* = \frac{1}{2}\left[p_R\left[\frac{\gamma-1}{2a_R}(u^* - u_R) + 1\right]^{\frac{1}{\beta}} + p_L\left[\frac{\gamma-1}{2a_L}(u_L - u^*) + 1\right]^{\frac{1}{\beta}}\right] \tag{42}$$

Note that we may also write p^* independently of u^*:

$$p^* = \left[\frac{a_L + a_R - \frac{\gamma-1}{2}(u_R - u_L)}{\frac{a_L}{p_L^\beta} + \frac{a_R}{p_R^\beta}}\right]^{\frac{1}{\beta}} \tag{43}$$

But eq. 42 is computationally more efficient if we compute u^* first. (Way fewer uses of fractional powers.)

4.4.2 Two-Shock Riemann Solver (TSRS)

The big idea is to assume a priori that both the left and right waves are going to be shock waves, and to use that assumption to get an expression for p^* and u^*, the pressure and velocity in the star region, respectively.

The equation for the pressure in the star region (eq. 34) then is given by

$$f(p) = (p - p_L)g_L(p) + (p - p_R)g_R(p) + u_R - u_L = 0 \tag{44}$$

$$g_{L,R}(p) = \left[\frac{A_{L,R}}{p + B_{L,R}}\right]^{1/2} \tag{45}$$

$$A_{L,R} = \frac{2}{(\gamma + 1)\rho_{L,R}} \tag{46}$$

$$B_{L,R} = \frac{\gamma - 1}{\gamma + 1}p_{L,R} \tag{47}$$

Unfortunately, this approximation does not lead to a closed form solution. So we can either use an iterative method again, or use further approximations. So the idea is to find p_0, an estimate for the pressure to use in eq. 44, and to use that to get a better approximation for p^*:

$$p^* = \frac{g_L(p_0)p_L + g_R(p_0)p_R - (u_R - u_L)}{g_L(p_0) + g_R(p_0)} \tag{48}$$

and

$$u^* = \frac{1}{2}(u_L + u_R) + \frac{1}{2}\left[(p^* - p_R)g_R(p_0) - (p^* - p_L)g_L(p_0)\right] \tag{49}$$

A good choice for p_0 is coming from the linearised solution of the primitive variables approach:

$$p_{PV} = \frac{1}{2}(p_L + p_R) - \frac{1}{8}(u_R - u_L)(\rho_L + \rho_R)(a_L + a_R)$$

$$p_0 = \max(\epsilon, p_{PV})$$

4.4.3 HLLC Solver

The HLLC solver is based on the approximation that we have 3 waves which are jump discontinuities, travelling with the speeds S_L, S^*, and S_R, respectively. Using integral relations and Rankine-Hugeniot relations, we directly find an expression for the fluxes $\mathbf{F}$:

$$\mathbf{F}_{i+1/2} = \begin{cases} \mathbf{F}_L & \text{if} \quad \frac{x}{t} \leq S_L \\ \mathbf{F}_L^* & \text{if} \quad S_L \leq \frac{x}{t} \leq S^* \\ \mathbf{F}_R^* & \text{if} \quad S^* \leq \frac{x}{t} \leq S_R \\ \mathbf{F}_R & \text{if} \quad S_R \leq \frac{x}{t} \end{cases} \tag{50}$$

where $x/t = 0$ at the boundary of a cell, $i + 1/2$, and with

$$S^* = \frac{p_R - p_L + \rho_L u_L (S_L - u_L) - \rho_R u_R (S_R - u_R)}{\rho_L (S_L - u_L) - \rho_R (S_R - u_R)} \tag{51}$$

$$\mathbf{U}_{L,R}^* = \rho_{L,R} \frac{S_{L,R} - u_{L,R}}{S_{L,R} - S^*} \begin{pmatrix} 1 \\ S^* \\ v_{L,R} \\ w_{L,R} \\ \frac{E_{L,R}}{\rho_{L,R}} + (S^* - u_{L,R}) \left(S^* + \frac{p_{L,R}}{\rho_{L,R}(S_{L,R} - u_{L,R})} \right) \end{pmatrix} \tag{52}$$

$$\mathbf{F}_{L,R}^* = \mathbf{F}_{L,R} + S_{L,R}(\mathbf{U}_{L,R}^* - \mathbf{U}_{L,R}) \tag{53}$$

for the flux in x - direction; For y and z direction, you'll need to exchange the velocity components with S^* appropriately. $\mathbf{U}_{L,R}$ are the given initial states, and $\mathbf{F}_{L,R} = \mathbf{F}(\mathbf{U}_{L,R})$ the corresponding initial states.

What we still lack is estimates for the left and right wave speeds, S_L and S_R. There are multiple ways to get good and robust estimates. A very simple, implemented one is:

$$S_L = u_L - a_L q_L \tag{54}$$

$$S_R = u_R + a_R q_R \tag{55}$$

$$q_{L,R} = \begin{cases} 1 & \text{if } p^* \leq p_{L,R} \quad \text{(rarefaction)} \\ \sqrt{1 + \frac{\gamma+1}{2\gamma}\left(\frac{p^*}{p_{L,R}} - 1\right)} & \text{if } p^* > p_{L,R} \quad \text{(shock)} \end{cases} \tag{56}$$

$$p^* = \max(0, p_{PV}) \tag{57}$$

$$p_{PV} = \frac{1}{2}(p_L + p_R) - \frac{1}{8}(u_R - u_L)(\rho_L + \rho_R)(a_L + a_R). \tag{58}$$

You can get a better estimate using an adaptive wave speed estimate method. We also start off by computing the primitive variable solution for the star state pressure p^* following eq. 58. Then we assume that the primitive variable estimate is ok as long as

- The ratio $\frac{p_{max}}{p_{min}}$ is small enough, where $p_{max} = \max\{p_L, p_R\}$ and $p_{min} = \min\{p_L, p_R\}$. In the code, the threshold ratio is hardcoded as 2.
- $p_{min} \leq p_{PV} \leq p_{max}$

If that is not the case, then we switch to the star state solutions of other approximate Riemann solvers: If $p_{PV} \leq p_{min}$, then we will certainly expect two rarefactions to form, so we use the star state estimates of the Two Rarefaction Riemann Solver (TRRS, eq. 42). Otherwise, we have at least one shock, so use the star state estimates of the Two Shock Riemann Solver (TSRS, eq. 48).

The adaptive wave speed estimate is used by default in the code, but can be turned off by removing the line `#define HLLC_USE_ADAPTIVE_SPEED_ESTIMATE` from `defines.h`.

Note that we don't need to compute the contact wave speed S^* according to eq. 51 in case we are using the TSRS or TRRS estimates, since this equation follows from the Rankine Hugeniot relations of three infinitely thin waves. Instead, you can directly compute it using the approximation that you are employing at that moment.

4.5 Dealing with Vacuum

Vacuum is characterized by the condition $\rho = 0$. With out equation of state, we also have $p = 0$ and $E = 0$ following from $\rho = 0$. The structure of the solution to the Riemann problem is different, there is no more star region. It can be shown that a shock wave can't be adjacent to a vacuum. Instead, we have a rarefaction wave and a contact wave which coalesces with the tail of the rarefaction. So we have a jump discontinuity next to the vacuum, which makes perfect sense, and this discontinuity will travel with

some "escape velocity" u_{vac}. Hence it makes sense to characterize the vacuum state as $\mathbf{W}_{vac} = (0, u_0, 0)$.

There are three cases to consider:

1. **The right state is a vacuum:**

   The vacuum front travels with the velocity

   $$S_{vac,L} = u_L + \frac{2a_L}{\gamma - 1} \tag{59}$$

   and left of it we have a left going rarefaction wave, i.e.

   $$\mathbf{W}_{L,\text{ with vacuum}} = \begin{cases} \mathbf{W_L} & \text{if } \frac{x}{t} \leq u_L - a_L \\ \mathbf{W}_{\mathbf{L},\text{inside fan}} & \text{if } u_L - a_L < \frac{x}{t} < S_{vac,L} \\ \mathbf{W_{vac}} & \text{if } \frac{x}{t} \geq S_{vac,L} \end{cases} \tag{60}$$

2. **The left state is a vacuum:**

   The vacuum front travels with the velocity

   $$S_{vac,R} = u_R - \frac{2a_R}{\gamma - 1} \tag{61}$$

   and right of it we have a right going rarefaction wave, i.e.

   $$\mathbf{W}_{R,\text{ with vacuum}} = \begin{cases} \mathbf{W_{vac}} & \text{if } \frac{x}{t} \leq S_{vac,R} \\ \mathbf{W}_{\mathbf{R},\text{inside fan}} & \text{if } S_{vac,R} < \frac{x}{t} < u_R + a_R \\ \mathbf{W_R} & \text{if } \frac{x}{t} \geq u_R + a_R \end{cases} \tag{62}$$

3. **Vacuum is being generated**

   In certain cases, with both the left and the right state being non-vacuum states, vacuum can be generated in regions of the solution. Just think what might happen if the left state has high velocity towards the left, and the right state having a high velocity towards the right, leaving the center region empty. The result is that we have a vacuum state emerging around the center, bounded by two vacuum fronts

$S_{vac,L}$ on the left side and $S_{vac,R}$ on the right side. The full solution is

$$\mathbf{W} = \begin{cases} \mathbf{W}_{L,\text{ with vacuum}} & \text{if } \frac{x}{t} \leq S_{vac,L} \\ \mathbf{W}_{vac} & \text{if } S_{vac,L} < \frac{x}{t} < S_{vac,R} \\ \mathbf{W}_{R,\text{ with vacuum}} & \text{if } \frac{x}{t} \geq S_{vac,R} \end{cases} \tag{63}$$

When do we have a vacuum generating condition? Well, $S_{vac,L} \leq S_{vac,R}$ must hold, hence

$$\Delta u_{crit} \equiv \frac{2a_L}{\gamma - 1} + \frac{2a_R}{\gamma - 1} \leq u_R - u_L \tag{64}$$

4.6 Sampling the Solution

With the solvers readily available, the final task is to sample the solution at some given point (x, t). Assuming we have computed all the star region state variables, what is left to do is to determine in which case the point (x, t) is located. The flow chart of decision making and finally which relations to use is shown in figure 4.

Initially all we need to compute is the star states, then we sample the solution at the given point that we're interested in, and the flowchart will tell us which states we need to compute using which relations.

4.7 Implementation Details

The specific Riemann solvers are `/program/src/riemann` directory. If we have a vacuum condition, we always use the vacuum solver, which is given above. It's not iterative, so no reason to do approximate solutions there. Note however that for the Godunov method (section 6.1), we may have problems with the vacuum solution. The vacuum solution of the Riemann problem gives non-zero velocities in regions where $\rho = p = E = 0$. So if we have for example a left vacuum, until the initial state boundary we will typically have $\mathbf{U} = 0$, while the Riemann problem will give $u \neq 0$ for the flux; So jump discontinuities will form.

A way to deal with this is to not set the density and pressure to zero, but to a very small number, I employ `SMALLRHO` and `SMALLP` in the code, which are defined as macros in `/program/src/defines.h`. Also, instead of returning a nonzero-velocity in the vacuum, return also something very small, `SMALLU`. This makes the code a bit more stable, but it will deviate from the solution of the exact Riemann solver.

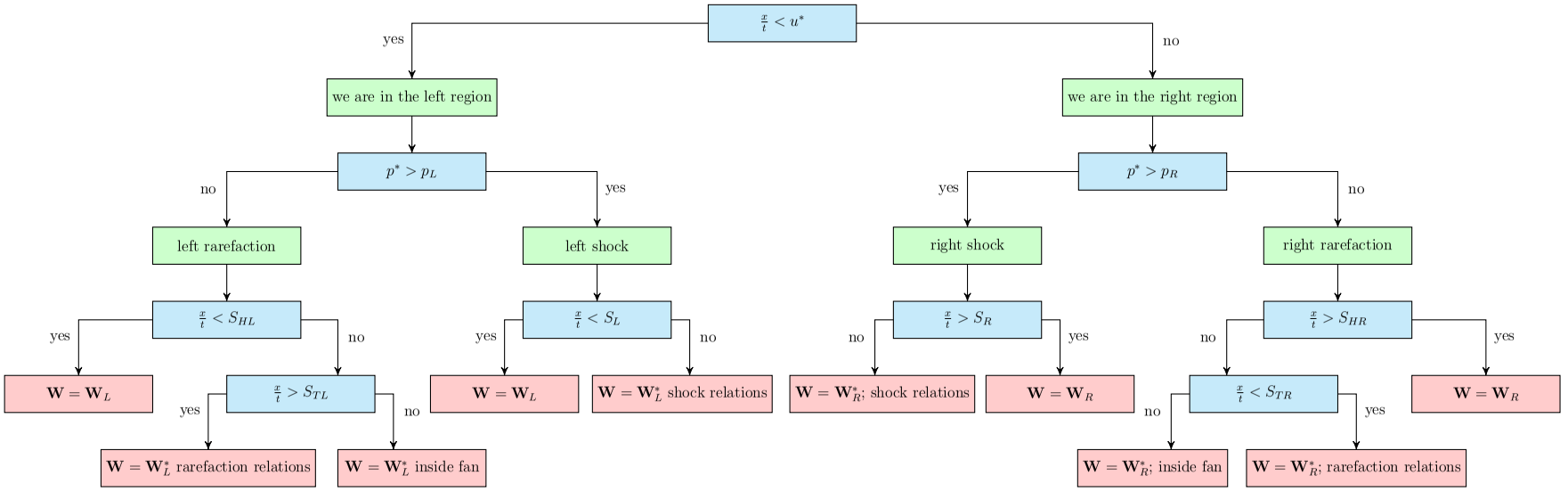

Figure 4: Flow chart to sample the solution of the Riemann problem for the Euler equations at a given point (x, t).

However, this exception handling is only used when the code isn't employed as a Riemann solver only. To differentiate between the cases, the `-DUSE_AS_RIEMANN_SOLVER` flag should be passed to the compiler if you want to run the code as a Riemann solver only.

5 Advection

5.1 Analytical Equation

Advection is a bit of an exception as a hydrodynamics method because we're not actually solving the (ideal) gas equations, but these instead:

$$\frac{\partial \mathbf{U}}{\partial t} + v \cdot \frac{\partial \mathbf{U}}{\partial x} = 0 \tag{65}$$

Which is still a conservation law of the form

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = 0 \tag{66}$$

with the flux tensor

$$\mathbf{F} = v \cdot \mathbf{U} \tag{67}$$

We assume the advection velocity $v = \text{const}$.

Note that in the formalism used, we only solve the 1D advection, but for every component of the state vector $\mathbf{U}$ and flux tensor $\mathbf{F}$.

The analytical solution is given by any function $q(x)$ with $\mathbf{U}(x,t) = q(x - vt)$, which is just $q(x)$ translated by vt.

Why do we even bother with linear advection? It's the simplest imaginable hyperbolic conservation law, and we can learn many things from considering this simple case. Furthermore, a lot of mathematical background can't be (or at least isn't yet) proven for more complex conservation laws, so often the approach is to adapt the results of the linear advection to more complex situations, like for the Euler equations.

5.2 Piecewise Constant Method

We assume that the cell state within a cell is constant (fig. 5). Furthermore, we also assume that the velocity v is constant and positive.

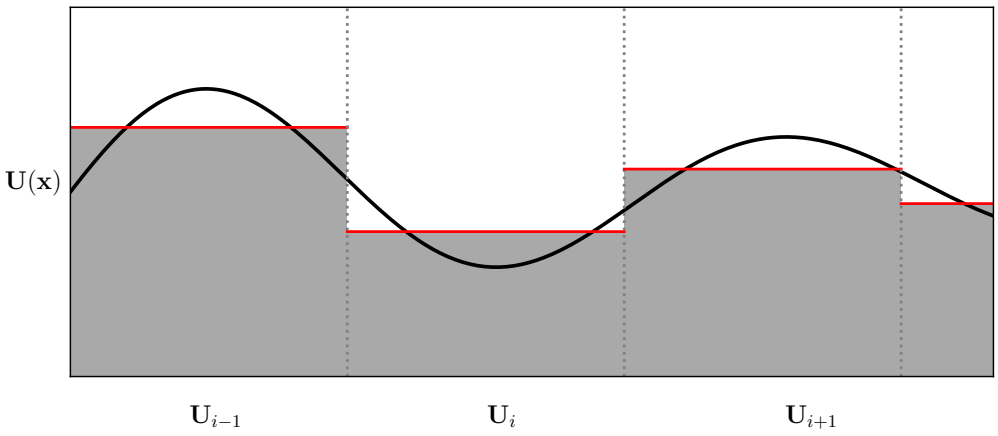

Figure 5: Piecewise constant reconstruction of the field

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2}\right) \tag{68}$$

$$\mathbf{F}_{i\pm 1/2}^{n+1/2} = v_{i\pm 1/2} \cdot \mathbf{U}_{i-1/2\pm 1/2} \tag{69}$$

The method is first order accurate in time and space.

We assumed that the velocity is positive and constant. What if it's negative?

The important point is that we always do **upwind differencing**. To obtain a finite difference, as we do here, you must never use the value that is downstream, i.e. that is in the direction of the flux. Doing this means taking a value for your computation that won't be valid as soon as an infinitesimal time interval passes, because the ingoing flux will change the downwind state. This is unphysical and leads to violent instabilities.

So if we have negative velocity, all we need to do is change the expression 69 to

$$\mathbf{F}_{i\pm 1/2}^{n+1/2} = v_{i\pm 1/2} \cdot \mathbf{U}_{i+1/2\pm 1/2} \tag{70}$$

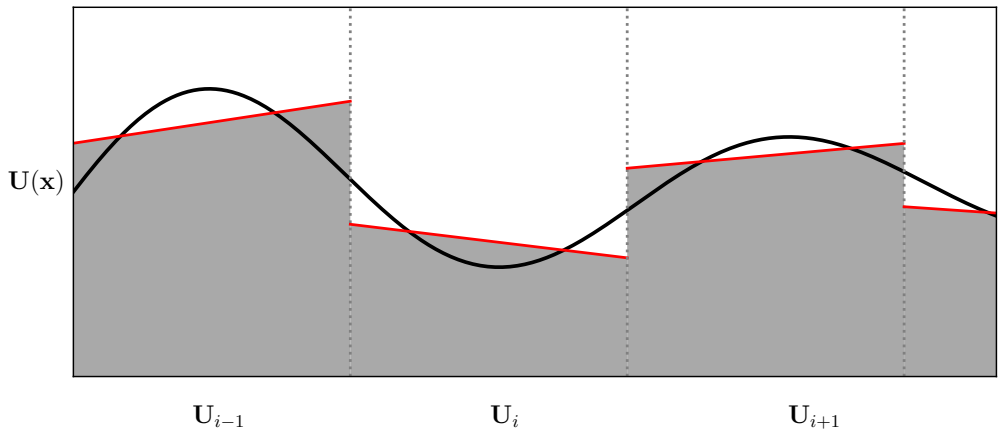

Figure 6: Piecewise linear reconstruction of the field

5.3 Piecewise Linear Method

The equation is still solved using the discretisation

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2}\right) \tag{71}$$

but his time, we assume that the state is not constant within a cell, but follows a piecewise linear profile with some slope $\mathbf{s}$ (fig. 6):

$$\begin{aligned}
\text{For } x_{i-1/2} < x < x_{i+1/2}: \quad & \mathbf{U}(x, t = t_n) = \mathbf{U}_i^n + \mathbf{s}_i^n (x - x_i) \\
\text{Centered slope: (Fromm)} \quad & \mathbf{s_i}^n = \frac{\mathbf{U}_{i+1}^n - \mathbf{U}_{i-1}^n}{2\Delta x} \\
\text{Upwind slope: (Beam-Warming)} \quad & \mathbf{s_i}^n = \mathbf{U}_i^n - \mathbf{U}_{i-1}^n \\
\text{Downwind slope: (Lax Wendroff)} \quad & \mathbf{s_i}^n = \mathbf{U}_{i+1}^n - \mathbf{U}_i^n
\end{aligned}$$

The "upwind" and "downwind" slope namings are assuming that the velocity $v > 0$. The implemented slope in the code is a centered slope. The decision to go with a centered slope has no particular reason, I just felt like it.

Assuming a positive constant velocity v, we derive the flux $\mathbf{F}$ at the time $t^n < t < t^{n+1}$ at the interface position $i-1/2$. At time t, the cell will have been advected by a distance $v(t-t^n)$, and the current state at the interface will be

$$\begin{aligned}\mathbf{U}(x = x_{i-1/2}, t) &= \mathbf{U}^n_{i-1} + \mathbf{s}_{i-1}(x_{i-1/2} - v(t-t^n) - x_{i-1}) \\ &= \mathbf{U}^n_{i-1} + \mathbf{s}_{i-1}(\frac{1}{2}\Delta x - v(t-t^n))\end{aligned}$$

To understand how the $x_{i-1/2} - v(t-t^n)$ comes into play, imagine the state doesn't change (i.e. isn't advected), but you move the boundaries to the left instead over a distance $v(t-t^n)$.

So if we have a **negative** constant velocity, the term changes to

$$\begin{aligned}\mathbf{U}(x = x_{i-1/2}, t) &= \mathbf{U}^n_{i} + \mathbf{s}_{i}(x_{i-1/2} - v(t-t^n) - x_{i}) \\ &= \mathbf{U}^n_{i} + \mathbf{s}_{i}(-v(t-t^n) - \Delta x)\end{aligned}$$

Note that the minus sign remains, and that the indices changed by one because we need to always make sure to do upwind differencing, i.e. take only values where the flow comes from, not from the direction where it's going.

Finally, we can compute the average flux over the time step $\Delta t = t^{n+1} - t^n$:

$$\mathbf{F}^{n+1/2}_{i-1/2} = \langle \mathbf{F}_{i+1/2}(t) \rangle^{t^{n+1}}_{t^n} = \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} v\mathbf{U}(x = x_{i-1/2}, t) \tag{72}$$

$$= \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} v\left(\mathbf{U}^n_{i-1} + \mathbf{s}_{i-1}(\frac{1}{2}\Delta x - v(t-t^n))\right) \tag{73}$$

$$= v\left(\mathbf{U}^n_{i-1} + \mathbf{s}_{i-1}\left(\frac{1}{2}\Delta x - v\left(\left[\frac{1}{2\Delta t}t^2\right]^{t^{n+1}}_{t^n} - t^n\right)\right)\right) \tag{74}$$

$$= v\left(\mathbf{U}^n_{i-1} + \mathbf{s}_{i-1}\left(\frac{1}{2}\Delta x - v\left[\frac{1}{2}(t^{n+1}+t^n) - t^n\right]\right)\right) \tag{75}$$

$$= v\left(\mathbf{U}^n_{i-1} + \frac{1}{2}\mathbf{s}_{i-1}\left(\Delta x - v\Delta t\right)\right) \tag{76}$$

Finally averaging the fluxes over a time step gives:

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - v \cdot \frac{\Delta t}{\Delta x}(\mathbf{U}_i^n - \mathbf{U}_{i-1}^n) - v \cdot \frac{\Delta t}{\Delta x}\frac{1}{2}(\mathbf{s}_i^n - \mathbf{s}_{i-1}^n)(\Delta x - v\Delta t) \tag{77}$$

This is the same as eq. 68 where we used

$$\begin{aligned}
\mathbf{F}_{i+1/2}^{n+1/2} &= v_{i+1/2} \cdot \mathbf{U}_{i+1/2}^{n+1/2} \\
&= v \cdot \mathbf{U}(x_{i+1/2} - \frac{1}{2}v\Delta t) \\
&= v \cdot \left(\mathbf{U}_i^n + \mathbf{s}_i^n[(x_{i+1/2} - \frac{1}{2}v\Delta t) - x_i] \right) \\
&= v \cdot \left(\mathbf{U}_i^n + \frac{1}{2}\mathbf{s}_i^n(\Delta x - v\Delta t) \right)
\end{aligned}$$

and analoguely

$$\mathbf{F}_{i-1/2}^{n+1/2} = v \cdot \left(\mathbf{U}_{i-1}^n + \frac{1}{2}\mathbf{s}_{i-1}^n(\Delta x - v\Delta t) \right)$$

To summarize the formulae:

$$\begin{aligned}
\mathbf{U}_i^{n+1} &= \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2} \right) \\
\mathbf{F}_{i-1/2}^{n+1/2} &= \begin{cases} v_{i-1/2} \cdot \mathbf{U}_{i-1}^n + \frac{1}{2}v_{i-1/2} \cdot \mathbf{s}_{i-1}^n(\Delta x - v_{i-1/2}\Delta t) & \text{for } v \geq 0 \\ v_{i-1/2} \cdot \mathbf{U}_i^n - \frac{1}{2}v_{i-1/2} \cdot \mathbf{s}_i^n(\Delta x + v_{i-1/2}\Delta t) & \text{for } v \leq 0 \end{cases} \\
\mathbf{F}_{i+1/2}^{n+1/2} &= \begin{cases} v_{i+1/2} \cdot \mathbf{U}_i^n + \frac{1}{2}v_{i+1/2} \cdot \mathbf{s}_i^n(\Delta x - v_{i+1/2}\Delta t) & \text{for } v \geq 0 \\ v_{i+1/2} \cdot \mathbf{U}_{i+1}^n - \frac{1}{2}v_{i+1/2} \cdot \mathbf{s}_{i+1}^n(\Delta x + v_{i+1/2}\Delta t) & \text{for } v \leq 0 \end{cases}
\end{aligned}$$

Let us introduce the unitless function $\phi(r)$ as

$$\phi(r_{i+1/2}) = \phi_{i+1/2} = \begin{cases} \frac{\Delta x}{\mathbf{U}_{i+1} - \mathbf{U}_i} \cdot \mathbf{s}_i & \text{for } v \geq 0 \\ \frac{\Delta x}{\mathbf{U}_{i+1} - \mathbf{U}_i} \cdot \mathbf{s}_{i+1} & \text{for } v \leq 0 \end{cases} \tag{78}$$

$$\phi(r_{i-1/2}) = \phi_{i-1/2} = \begin{cases} \frac{\Delta x}{\mathbf{U}_i - \mathbf{U}_{i-1}} \cdot \mathbf{s}_{i-1} & \text{for } v \geq 0 \\ \frac{\Delta x}{\mathbf{U}_i - \mathbf{U}_{i-1}} \cdot \mathbf{s}_i & \text{for } v \leq 0 \end{cases} \tag{79}$$

Such that

$$\mathbf{F}_{i-1/2}^{n+1/2} = \begin{cases} v_{i-1/2} \cdot \mathbf{U}_{i-1}^n + \frac{1}{2}|v_{i-1/2}| \left(1 - \frac{|v_{i-1/2}|\Delta t}{\Delta x}\right) \phi_{i-1/2}(\mathbf{U}_i - \mathbf{U}_{i-1}) & \text{for } v \geq 0 \\ v_{i-1/2} \cdot \mathbf{U}_i^n + \frac{1}{2}|v_{i-1/2}| \left(1 - \frac{|v_{i-1/2}|\Delta t}{\Delta x}\right) \phi_{i-1/2}(\mathbf{U}_i - \mathbf{U}_{i-1}) & \text{for } v \leq 0 \end{cases}$$

$$\mathbf{F}_{i+1/2}^{n+1/2} = \begin{cases} v_{i+1/2} \cdot \mathbf{U}_i^n + \frac{1}{2}|v_{i+1/2}| \left(1 - \frac{|v_{i+1/2}|\Delta t}{\Delta x}\right) \phi_{i+1/2}(\mathbf{U}_{i+1} - \mathbf{U}_i) & \text{for } v \geq 0 \\ v_{i+1/2} \cdot \mathbf{U}_{i+1}^n + \frac{1}{2}|v_{i+1/2}| \left(1 - \frac{|v_{i+1/2}|\Delta t}{\Delta x}\right) \phi_{i+1/2}(\mathbf{U}_{i+1} - \mathbf{U}_i) & \text{for } v \geq 0 \end{cases}$$

Depending on our choice of ϕ, we can get different slopes. Here for positive velocity only, and for $r = r_{i-1/2}$:

$$\phi(r) = 0 \rightarrow \mathbf{s}_i = 0 \qquad \text{No slopes; Piecewise constant method.}$$

$$\phi(r) = 1 \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_i - \mathbf{U}_{i-1}}{\Delta x} \qquad \text{Downwind slope (Lax-Wendroff)}$$

$$\phi(r) = r \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_{i-1} - \mathbf{U}_{i-2}}{\Delta x} \qquad \text{Upwind slope (Beam-Warming)}$$

$$\phi(r) = \frac{1}{2}(1 + r) \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_i - \mathbf{U}_{i-2}}{2\Delta x} \qquad \text{Centered slope (Fromm)}$$

More elaborate expressions for $\phi(r)$ are going to be used to construct monotone high order schemes using a slope limiter, see section 7.

5.4 Weighted Average Flux (WAF) Method

For the WAF method, we again assume piece-wise constant data (see fig. 5), i.e.

$$\mathbf{U}_i^n = \frac{1}{\Delta \mathbf{x}} \int_{\mathbf{x}_{i-1/2}}^{\mathbf{x}_{i+1/2}} \mathbf{U}(\mathbf{x}, t^n) \mathrm{d}\mathbf{x} \tag{80}$$

The scheme is again based on the explicit conservative formula

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x} \left[\mathbf{F}_{i-1/2} - \mathbf{F}_{i+1/2} \right] \tag{81}$$

The intercell flux $\mathbf{F}_{i+1/2}$ is defined as an integral average of the flux function:

$$\mathbf{F}_{i+1/2} = \frac{1}{\Delta x} \int_{-\frac{1}{2}\Delta x}^{\frac{1}{2}\Delta x} \mathbf{F}(\mathbf{U}_{i+1/2}(x, \frac{1}{2}\Delta t)) \mathrm{d}x \tag{82}$$

The integration range goes from the middle of the cell to the middle of the neighbouring cell.

The solution for $\mathbf{U}_{i+1/2}(x, \frac{1}{2}\Delta t)$ for linear advection is simple; it is

$$\mathbf{U}_{i+1/2}(x,t) = \begin{cases} \mathbf{U}_i & \frac{x}{t} < v \\ \mathbf{U}_{i+1} & \frac{x}{t} > v \end{cases} \tag{83}$$

and the evaluation of the flux integral is trivial since the solution consists of constant states: See figure 7. For $v > 0$, the state, and hence the flux, is constant over the intervals $\overline{AB^+}$ and $\overline{B^+C}$; For $v < 0$, the state, and hence the flux, is constant over the intervals $\overline{AB^-}$ and $\overline{B^-C}$.

It's easy to show that

$$\overline{AB^+} = \frac{1}{2}\Delta x + v \cdot \frac{1}{2}\Delta t$$
$$\overline{B^+C} = \frac{1}{2}\Delta x - v \cdot \frac{1}{2}\Delta t$$

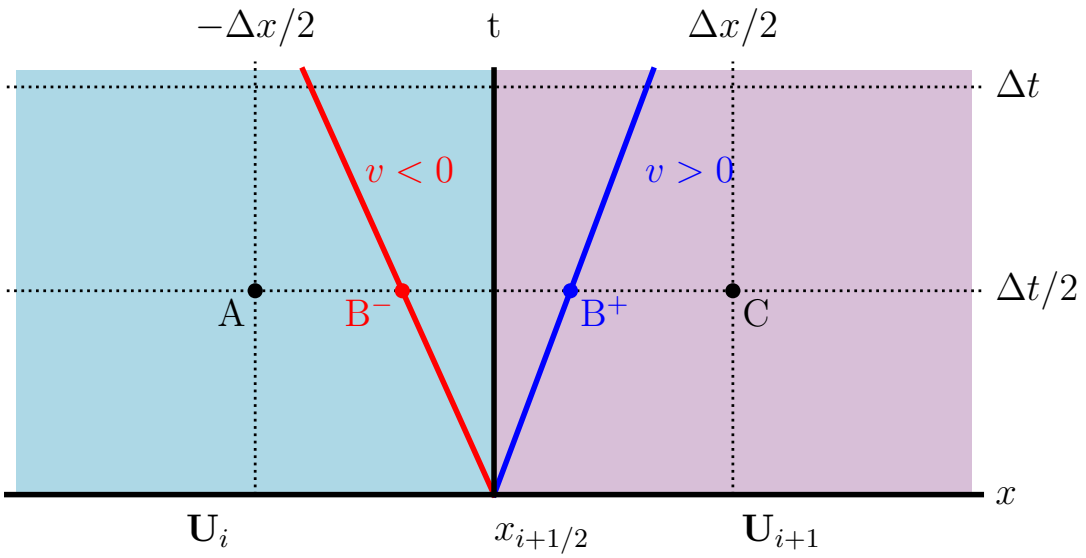

Figure 7: Figure to show the derivation of the WAF intercell flux (eq. 82) for linear advection. We have two piecewise constant states, $\mathbf{U}_i$ and $\mathbf{U}_{i+1}$, separated at the position $x_{i+1/2}$. For $v > 0$, the state, and hence the flux, is constant over the intervals $\overline{AB^+}$ and $\overline{B^+C}$. For $v < 0$, it is constant over the intervals $\overline{AB^-}$ and $\overline{B^-C}$.

$$\overline{AB^-} = \frac{1}{2}\Delta x - |v| \cdot \frac{1}{2}\Delta t$$
$$\overline{B^-C} = \frac{1}{2}\Delta x + |v| \cdot \frac{1}{2}\Delta t$$

So in either case for $v > 0$ and $v < 0$ we obtain the final expression for the flux

$$\mathbf{F}_{i+1/2} = \frac{1}{2}(1+c)v\mathbf{U}_i^n + \frac{1}{2}(1-c)v\mathbf{U}_{i+1} \tag{84}$$

where

$$c = \frac{v\Delta t}{\Delta x} \tag{85}$$

and is allowed to be negative.

This gives us a second order accurate method despite having piecewise constant data.

Second order methods will have spurious oscillations near discontinuities and steep gradients, which we try to handle with flux limiters. To this end, a flux limited WAF flux can be written as

$$\mathbf{F}_{i+1/2}^{n+1/2} = \frac{1}{2}(1 + sign(v)\psi_{i+1/2})\ v\mathbf{U}_i^n + \frac{1}{2}(1 - sign(v)\psi_{i+1/2})\ v\mathbf{U}_{i+1}^n \tag{86}$$

possible choices for flux limiters ψ are discussed in section 7. The choice

$$\psi = |c|$$

recovers the original expression.

5.5 CFL Condition

To keep things stable and physical, we must not allow any flux in the simulation to go further than one single cell size. Otherwise, you're skipping interactions between fluxes on cells. This time restriction is known as the CFL condition.

In 1D, it's straightforward:

$$\Delta t_{max} = C_{cfl}\frac{\Delta x}{v_{max}} \tag{87}$$

$C_{cfl} \in [0, 1)$ is a user-set factor. The lower it is, the more precise the results, but the more computations you need to do.

In 2D, it is:

$$\Delta t_{max} = C_{cfl}\left(\frac{|v_{x,max}|}{\Delta x} + \frac{|v_{y,max}|}{\Delta y}\right)^{-1} \tag{88}$$

This condition is more strict than what one would expect from the restriction based on physical arguments, i.e. not allowing the flux to pass more than one cell, which would be $\Delta t_{max} = C_{cfl}\min\left\{\frac{\Delta x}{|v_{x,max}|}, \frac{\Delta y}{|v_{y,max}|}\right\}$. It follows from a convergence condition in (von Neumann) stability analysis of the method.

For N dimensions, the condition translates to

$$\Delta t_{max} = C_{cfl} \left(\sum_{i=1}^{N} \frac{|v_{i,max}|}{\Delta x_i} \right)^{-1} \tag{89}$$

5.6 Implementation Details

5.6.1 General

What is implemented is the equation

$$\frac{\partial \mathbf{U}}{\partial t} + v \cdot \frac{\partial \mathbf{U}}{\partial x} = 0 \tag{90}$$

where we assume that the velocity v is constant.

In every case, this equation is solved through the discretization

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x} \left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2} \right) \tag{91}$$

and the expressions for the fluxes $\mathbf{F}_{i\pm1/2}$ vary between the methods.

Therefore, the fluid velocity is never updated, but kept identical to the initial conditions. You can change that behaviour by removing the `ADVECTION_KEEP_VELOCITY_CONSTANT` macro definition in `defines.h`

Even though usually $\mathbf{U}$ is taken to be a conserved state, advection only updates the primitive states, as those are the one that are read in and written out.

The advection related functions are called in the main loop in `/program/src/main.c` when `solver_step(...)` is called. The specifics for each solver are described in the following subsection.

5.6.2 Piecewise Constant Advection

The related functions are written in `/program/src/solver/advection_pwconst.c` and `/program/src/solver/advection_pwconst.h`.

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero
- Impose boundary conditions (section 8)
- Find the maximal timestep that you can do by applying the CFL condition 87.
- Compute fluxes:
  - For every cell, find the upwind and the downwind cell for the interaction.
  - Compute the net flux, i.e.

$$\mathbf{F}_{i,net}^{n+1/2} = \mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2} \tag{92}$$

    and store it in the `struct pstate pflux` struct of the cell i. `struct pstate` is a struct that contains the primitive state, i.e. density ρ, velocity u_x, u_y, and pressure p.
- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$ and the computed net flux $\mathbf{F}_{i,net}^{n+1/2}$ according to eqn. 91.

For 2D, we just do the 1D step twice, once in each direction while using the intermediate result coming from the first sweep as the IC for the second step in the other dimension. See section 9 for details.

5.6.3 Piecewise Linear Advection

The related functions are written in `/program/src/solver/advection_pwlin.c` and `/program/src/solver/advection_pwlin.h`.

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero
- Impose boundary conditions (section 8)
- Find the maximal timestep that you can do by applying the CFL condition 87.
- Compute fluxes:

- For every cell, find the upwind and the downwind cell for the interaction.

- Compute the net flux, i.e.

$$\mathbf{F}_{i,net}^{n+1/2} = \mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2} \tag{93}$$

  and store it in the `struct pstate pflux` struct of the cell i. `struct pstate` is a struct that contains the primitive state, i.e. density ρ, velocity u_x, u_y, and pressure p.

- When computing the fluxes $\mathbf{F}_{i-1/2}^{n+1/2}$ and $\mathbf{F}_{i+1/2}^{n+1/2}$, first just compute the left and right slope by calling the `limiter_get_advection_slope_left(...)` and `limiter_get_advection_slope_right(...)` functions, and only determine later which one is the upwind one and which one is the downwind one. The `limiter_get_advection_slope_left/right` functions will figure out which limiter to use, even if you chose the "no limiter" option. If the "no limiter" option is chosen, you get a centered slope.

- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$ and the computed net flux $\mathbf{F}_{i,net}^{n+1/2}$ according to eqn. 91.

For 2D, we just do the 1D step twice, once in each direction while using the intermediate result coming from the first sweep as the IC for the second step in the other dimension. See section 9 for details.

5.6.4 WAF Advection

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero

- Impose boundary conditions (section 8)

- Find the maximal timestep that you can do by applying the CFL condition 87.

- Compute fluxes:

  - For every cell, find the pass it and it's neighbour with index $i+1$ to compute the flux.

  - Compute the flux, i.e. $\mathbf{F}_{i+1/2}^{n+1/2}$, and subtract it from the `struct pstate pflux` struct of the cell i. `struct pstate` is a struct that contains the prim-

itive state, i.e. density ρ, velocity u_x, u_y, and pressure p. But also add it to the cell with index $i+1$, since $\mathbf{F}_{i+1/2}$ will the the flux $\mathbf{F}_{i-1/2}$ for the cell with $i = i+1$. This way, once the iteration is over, the net flux $\mathbf{F}_{i-1/2}^{n+1/2}$ and $\mathbf{F}_{i+1/2}^{n+1/2}$ will be stored in `struct pstate pflux` of the cell i.

- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$ and the computed net flux $\mathbf{F}_{i,net}^{n+1/2}$ according to eqn. 91.

For 2D, we just do the 1D step twice, once in each direction while using the intermediate result coming from the first sweep as the IC for the second step in the other dimension. See section 9 for details.

6 Hydrodynamics Methods

6.1 Godunov's Method

6.1.1 Method

Godunov's method arises from the integral form of the conservation law so that discontinuous solutions are allowable.

For the one dimensional method, we discretize the spatial domain into M computing cells of regular size, and assume that the initially continuous data is represented by piecewise constant distribution of data, see fig. 8.

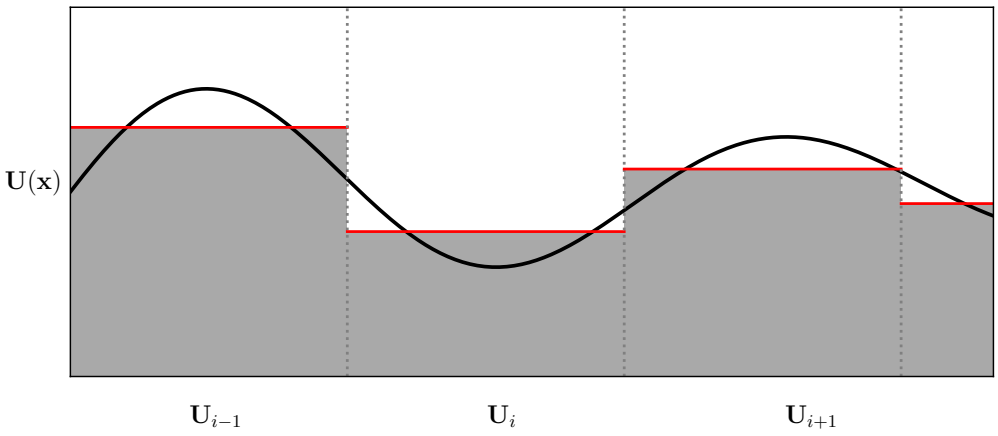

Figure 8: A piecewise constant representation of continuous data among cells.

Having a collection of piecewise constant states, we effectively have to solve local Riemann problems with data $\mathbf{U}_i$ as $\mathbf{U}_L$ and $\mathbf{U}_{i+1}$ as $\mathbf{U}_R$, centered at the intercell boundary positions $x_{i+1/2}$. The solution of the Riemann problem will depend on $\frac{\bar{x}}{\bar{t}}$, where $\bar{x}$ and $\bar{t}$ are in local coordinates to the specific Riemann problem under consideration. $\bar{x}$ is zero at $x_{i+1/2}$ and increasingly negative with decreasing i. $\bar{t}$ is zero at the current timestep.

Now suppose that we have solved the Riemann problem at the position $x_{i+1/2}$ with left state $\mathbf{U}_L = \mathbf{U}_i$ and right state $\mathbf{U}_R = \mathbf{U}_{i+1}$. Then, as time evolves, how will the state at $x_{i+1/2}$ change?

Recall that the elementary waves travel along characteristics, and that the characteristics are straight lines on the $x-t$ - diagram (see fig. 2. Then the state at $x_{i+1/2}$, which is where the dividing line between the two initial states is, will be given by the solution of the Riemann problem at the position $\bar{x}=0$, and will remain the same for all $\bar{t}>0$. (This is assuming there is nothing else that might disturb the current situation.)

Using that fact, it can be derived that (see Toro [1999])

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left[\mathbf{F}(\mathbf{U}_{i-1/2}) - \mathbf{F}(\mathbf{U}_{i+1/2})\right] \tag{94}$$

where $\mathbf{U}_{i-1/2}$ and $\mathbf{U}_{i+1/2}$ are the solutions to the Riemann problems at $x_{i-1/2}$ and $x_{i+1/2}$, respectively.

It is noteworthy that this is an exact solution to a piecewise constant initial state.

Lastly, we need to limit the time step size. We mustn't allow for a wave to be able to travel further than one cell length between two timesteps, otherwise we get bogus results. Remember that we assumed that the state at $x_{i+1/2}$ doesn't change after $\bar{t}>0$. This is only satisfied if the wave doesn't reach the boundary of the neighbouring cell.

This time step restriction is imposed by the CFL condition:

$$\Delta t_{max} \leq \frac{C_{cfl}\Delta x}{|S_{max}^n|} \tag{95}$$

where S_{max}^n is the highest wave propagation speed at the current time, and $C_{cfl} \in [0,1($ is the Courant number.

However, this is not the most practical way of doing things. In 2D, using dimensional splitting, we'd have to first solve everything in one direction to find the wave speeds, then advance the time step of the sweep, then do the other sweep and hope that the maximal wave velocity won't be greater than the one of the previous sweep. Or re-do the first sweep iteratively until we get decent time steps.

Instead, we use the estimate

$$S^n_{max} = \max\{|u^n_i| + a^n_i\} \tag{96}$$

This is not always accurate, and the wave speed can be underestimated, leading to instabilities. To combat this, we just need to choose a lower C_{cfl}. Toro recommends to use $C_{cfl} < 0.8 - 0.9$.

6.1.2 Implementation Details

The cells are stored as an array of `struct cell` that stores both primitive states, `prim`, as `struct pstate`, and conserved states, `cons`, as `struct cstate`. Furthermore they have `struct pstate pflux` and `struct cstate cflux` to store fluxes of primitive and conserved variables.

The grid is set up as follows: In 1D, it is a 1D array of `struct cell`. In 2D, it is a 2D array. Indices (0, 0) represent the lower left corner of the simulation domain. First index is in x direction, i.e. (nx - 1, 0) is at the coordinates (x = xmax, y = 0).

The flux at $x_{i+1/2,j}$ and $y_{i,j+1/2}$ are stored in `cflux` or `pflux` of cell `grid[i, j]`, depending whether you're storing primitive or conserved variables. For the Godunov scheme, we need conserved variables. For advection, we only deal with primitive variables. Because we're doing dimensional splitting, it suffices to have only one storage place, as they will be used in successive order. See section 9 for details.

We can afford to store $x_{i+1/2}$ at cell i because we have at least 1 extra virtual boundary cell which is used to apply boundary conditions, so the flux at $x_{-1/2}$ will be stored in `grid[BC-1]`, where `BC` is the number of boundary cells used, defined in `defines.h`.

If the grid is only in 1D, then all the above definitions still apply as if y didn't exist.

The related functions are written in `/program/src/solver/godunov.c` and `/program/src/solver/godunov.h`. The hydro related functions are called in the main loop in `/program/src/main.c` when `solver_step(...)` is called.

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero
- Compute the primitive states for all cells from the updated conserved states

- Impose boundary conditions (section 8)
- Find the maximal timestep that you can do by applying the CFL condition 95.
- Compute fluxes:
  - For every cell pair $(i, i+1)$, solve the Riemann problem (see section 4) to find the flux $\mathbf{F}_{i+1/2}$.
  - Store the flux $\mathbf{F}_{i+1/2}$ in the `struct pstate pflux` struct of the cell i. `struct pstate` is a struct that contains the primitive state, i.e. density ρ, velocity u_x, u_y, and pressure p.
- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$, the flux $\mathbf{F}_{i+1/2}$ stored in every cell i, and the flux $\mathbf{F}_{i-1/2}$ stored in every cell $i-1$ following eq. 94.

6.2 Weighted Average Flux (WAF) Method

6.2.1 Method

For the WAF method, we again assume piece-wise constant data (see fig. 8), i.e.

$$\mathbf{U}_i^n = \frac{1}{\Delta \mathbf{x}} \int_{\mathbf{x}_{i-1/2}}^{\mathbf{x}_{i+1/2}} \mathbf{U}(\mathbf{x}, t^n) \mathrm{d}\mathbf{x} \tag{97}$$

The scheme is again based on the explicit conservative formula

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x} \left[\mathbf{F}_{i-1/2} - \mathbf{F}_{i+1/2} \right] \tag{98}$$

The intercell flux $\mathbf{F}_{i+1/2}$ is defined as an integral average of the flux function:

$$\mathbf{F}_{i+1/2} = \frac{1}{\Delta x} \int_{-\frac{1}{2}\Delta x}^{\frac{1}{2}\Delta x} \mathbf{F}(\mathbf{U}_{i+1/2}(x, \frac{1}{2}\Delta t)) \mathrm{d}x \tag{99}$$

The integration range goes from the middle of the cell to the middle of the neighbouring

cell.

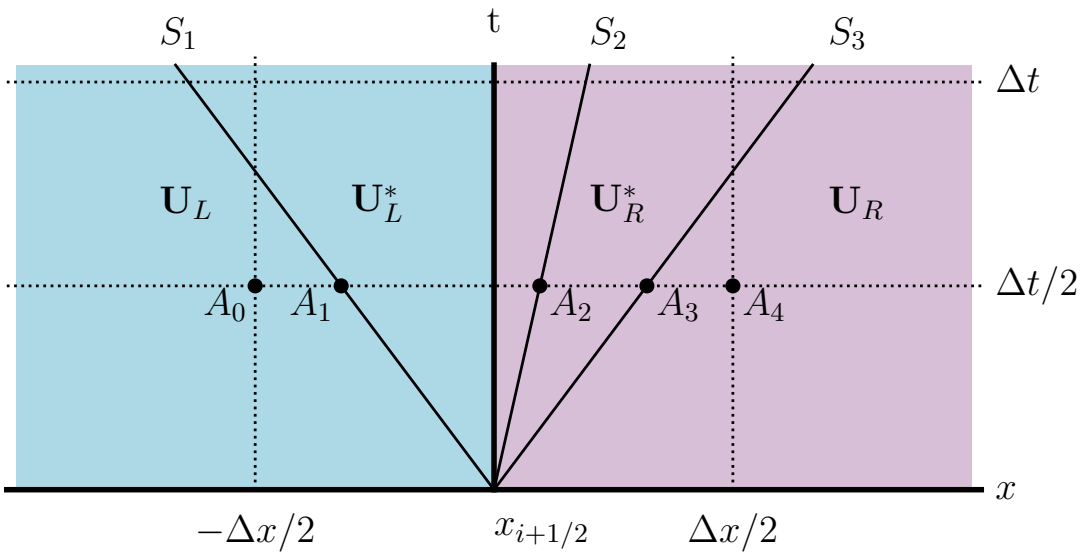

Figure 9: Figure to show the derivation of the WAF intercell flux (eq. 99) for the 1D Euler equations. We have initially two piecewise constant states, $\mathbf{U}_i$ and $\mathbf{U}_{i+1}$, separated at the position $x_{i+1/2}$. As time evolves, three waves will emerge, with respective speeds S_1, S_2, and S_3. The states between the points A_k, A_{k+1} are assumed constant.

The solution of the Riemann problem separates the two initial states $\mathbf{U}_L = \mathbf{U}_i$, $U_R = \mathbf{U}_{i+1}$ into four states

$$\mathbf{U}^{(1)} = \mathbf{U}_L, \quad \mathbf{U}^{(2)} = \mathbf{U}_L^*, \quad \mathbf{U}^{(3)} = \mathbf{U}_R^*, \quad \mathbf{U}^{(4)} = \mathbf{U}_R$$

that are separated by three waves with the speeds S_1, S_2, and S_3. At $t = \frac{1}{2}\Delta t$, we can separate the interval $[-\Delta x/2, \Delta x/2]$ by introducing 5 points along the x axis:

$$\begin{aligned} A_0 &= -\Delta x/2 \\ A_1 &= S_1 \Delta t/2 \\ A_2 &= S_2 \Delta t/2 \\ A_3 &= S_3 \Delta t/2 \\ A_4 &= \Delta x/2 \end{aligned}$$

and separate the integral 99 into the sum

$$\mathbf{F}_{i+1/2} = \frac{1}{\Delta x} \sum_{k=1}^{N+1} \int_{A_{k-1}}^{A_k} \mathbf{F}(\mathbf{U}(x, \Delta t/2)) \mathrm{d}x \tag{100}$$

where N is the number of occuring waves in the solution.

Note that with $|C_{cfl}| \leq 1$ we should always have all the waves within $[-\Delta x/2, \Delta x/2]$ at $t = \Delta t/2$ if the C_{cfl} is chosen properly using actual wave speeds. However, we use approximate wave speed estimates, so we need to check whether the actual waves are still inside $[-\Delta x/2, \Delta x/2]$ at $t = \Delta t/2$.

Since we assume constant states between these points (which they will be unless we have a rarefaction present), the fluxes $\mathbf{F}$ between the points A_k will be constant too, and the integral is trivial. We only need expressions for the distances $\overline{A_k A_{k+1}}$. It's easy to show that regardless of the sign of the wave speeds S_k, we obtain

$$\begin{aligned}
\overline{A_0 A_1} &= \frac{\Delta x}{2}(1 + c_1) \\
\overline{A_1 A_2} &= \frac{\Delta x}{2}(c_2 - c_1) \\
\overline{A_2 A_3} &= \frac{\Delta x}{2}(c_3 - c_2) \\
\overline{A_3 A_4} &= \frac{\Delta x}{2}(1 - c_3) \\
\text{with } c_k &= \frac{S_k \Delta t}{\Delta x}
\end{aligned}$$

If we define

$$\begin{aligned}
\beta_k &= \frac{\overline{A_{k-1} A_k}}{\Delta x} \\
c_0 &= -1, \quad c_5 = c_{N+1} = 1
\end{aligned}$$

we obtain

$$\beta_k = \frac{1}{2}(c_k - c_{k-1})$$

and we can write the WAF flux as

$$\mathbf{F}_{i+1/2} = \sum_{k=1}^{N+1} \beta_k \mathbf{F}^{(k)} \tag{101}$$

$$= \frac{1}{2}(\mathbf{F}_i + \mathbf{F}_{i+1}) - \frac{1}{2}\sum_{k=1}^{N} c_k \left(\mathbf{F}^{(k+1)} - \mathbf{F}^{(k)}\right) \tag{102}$$

The TVD modification of the WAF flux is

$$\mathbf{F}_{i+1/2} = \frac{1}{2}(\mathbf{F}_i + \mathbf{F}_{i+1}) - \frac{1}{2}\sum_{k=1}^{N} sign(c_k)\psi^{(k)}_{i+1/2} \left(\mathbf{F}^{(k+1)} - \mathbf{F}^{(k)}\right) \tag{103}$$

$$\psi^{(k)}_{i+1/2} = \psi_{i+1/2}(r^{(k)}) \tag{104}$$

$$r^{(k)} = \begin{cases} \frac{\Delta q^{(k)}_{i-1/2}}{\Delta q^{(k)}_{i+1/2}} & \text{if } c_k > 0 \\ \\ \frac{\Delta q^{(k)}_{i+3/2}}{\Delta q^{(k)}_{i+1/2}} & \text{if } c_k < 0 \end{cases} \tag{105}$$

Where q is one single quantity which is known to change across every wave. Options are density ρ and internal energy ϵ. I implemented the choice ρ.

The notation here is a bit tricky: k signifies which wave in the solution of Riemann problems (not only one Riemann problem!!) we look at:

- $\Delta q^{(k)}_{i+1/2}$ is the jump in e.g. density between wave k and $k+1$ of the solution of the Riemann problem with initial conditions $\mathbf{U}_L = \mathbf{U}_i$, $\mathbf{U}_R = \mathbf{U}_{i+1}$

- $\Delta q^{(k)}_{i-1/2}$ is the jump between wave k and $k+1$ of the solution of the Riemann problem with initial conditions $\mathbf{U}_L = \mathbf{U}_{i-1}$, $\mathbf{U}_R = \mathbf{U}_i$

- $\Delta q^{(k)}_{i+3/2}$ is the jump between wave k and $k+1$ of the solution of the Riemann

problem with initial conditions $\mathbf{U}_L = \mathbf{U}_{i+1}$, $\mathbf{U}_R = \mathbf{U}_{i+2}$

See figure 10 for visual clarification.

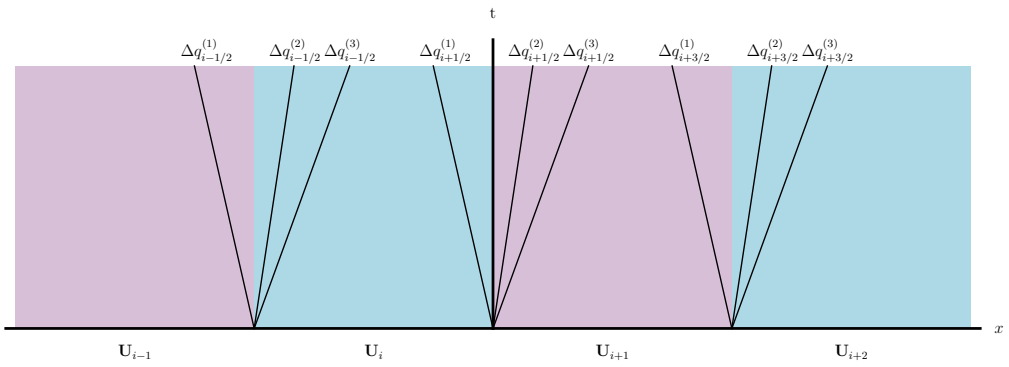

Figure 10: Clarification of the value differences $\Delta q_i^{(k)}$ needed to compute the flow parameter r for the WAF method.

The limiters ψ are related to conventional flux limiters ϕ via

$$\psi_{i+1/2} = 1 - (1 - |c|)\phi_{i+1/2}(r) \tag{106}$$

Some implemented options are given in section 7.6.

6.2.2 Dealing with Vacuum

Vacuum needs some special attention because it doesn't follow the solution structure of a typical Riemann problem. It can easily crash your program: Consider for example the computation of the local sound speed $a = \sqrt{\gamma p/\rho}$ for $\rho = 0$. There is no more star region present, instead next to the vacuum ($\rho = 0$) we have a contact wave coinciding with the tail of a rarefaction wave - a wave that we also handle only approximately.

I wasn't able to find literature of how to best deal with this, but I was able to find an approximate solution on my own following the idea of how the WAF scheme works.

In the end, what we want is a solution that we can use to get WAF fluxes, namely we need

- 4 fluxes $\mathbf{F}_i^{(k)}$ corresponding to the four states $\mathbf{U}_i^{(k)}$: U_L, U_L^*, U_R^*, U_R
- 3 wave speeds S_L, S^*, S_R

- 3 differences/jumps in density $q^{(k)}$: $\rho_L^* - \rho_L$, $\rho_R^* - \rho_L^*$, $\rho_R - \rho_R^*$

For vacuum, we have three cases to consider, where $\mathbf{U}_{vac}$ describes the vacuum state:

1. **The right state is a vacuum**:

   Choose

$$\begin{aligned}
\mathbf{F}^{(1)} &= \mathbf{F}(\mathbf{U}_L) \\
\mathbf{F}^{(2)} &= \begin{cases} \mathbf{F}(\mathbf{U}(t = \Delta t/2,\ x = 0) & \text{for a sonic rarefaction} \\ \mathbf{F}(\mathbf{U}_L) & \text{for a non-sonic rarefaction} \end{cases} \\
\mathbf{F}^{(3)} &= \mathbf{F}(\mathbf{U}_{vac}) \\
\mathbf{F}^{(4)} &= \mathbf{F}(\mathbf{U}_R)
\end{aligned}$$

$$\begin{aligned}
S^{(1)} &= 0 \\
S^{(2)} &= \text{ rarefaction tail speed} \\
S^{(3)} &= \text{ rarefaction head speed}
\end{aligned}$$

$$\begin{aligned}
q^{(1)} &= \rho_L^* - \rho_L \\
q^{(2)} &= 0 \\
q^{(3)} &= 0
\end{aligned}$$

   Where ρ_L^* is either the density of $\mathbf{U}_L$ or of $\mathbf{U}(x = 0, t = \Delta t/2)$, depending on whether we have a sonic or non-sonic rarefaction.

2. **The left state is a vacuum**:

   Choose

$$\begin{aligned}
\mathbf{F}^{(1)} &= \mathbf{F}(\mathbf{U}_L) \\
\mathbf{F}^{(2)} &= \mathbf{F}(\mathbf{U}_{vac}) \\
\mathbf{F}^{(3)} &= \begin{cases} \mathbf{F}(\mathbf{U}(t = \Delta t/2,\ x = 0) & \text{for a sonic rarefaction} \\ \mathbf{F}(\mathbf{U}_R) & \text{for a non-sonic rarefaction} \end{cases} \\
\mathbf{F}^{(4)} &= \mathbf{F}(\mathbf{U}_R)
\end{aligned}$$

$$\begin{aligned}
S^{(1)} &= 0 \\
S^{(2)} &= \text{ rarefaction tail speed} \\
S^{(3)} &= \text{ rarefaction head speed}
\end{aligned}$$

$$q^{(1)} = 0$$
$$q^{(2)} = 0$$
$$q^{(3)} = \rho_R - \rho_R^*$$

Where ρ_R^* is either the density of $\mathbf{U}_R$ or of $\mathbf{U}(x = 0, t = \Delta t/2)$, depending on whether we have a sonic or non-sonic rarefaction.

3. **A vacuum is being generated**:

   This may occur when condition 64 is satisfied. The solution structure contains two rarefaction fans, to the left and to the right, enclosed by the initial states U_L and U_R, and separated by a vacuum state U_{vac}.

   My solution strategy is as follows: Compute an approximate solution to describe the average state inside each rarefaction fan, and then "stretch" the states out such that we pretend that there is no vacuum state in between.

   For example, for the left star state, we can compute the rarefaction head speed $S_{L,H}$ and tail speed $S_{L,T}$ and define the average speed $\overline{S}_L = \frac{S_{H,L}+S_{T,L}}{2}$.

   Now we approximate the entire star state as the solution inside a rarefaction fan, eqns. 24-26 and 27-29, at $x/t = \overline{S}_L$. The same computation can be done for the right rarefaction.

   Then we say that we have no vacuum state in between, but instead "stretch out" the star states to meet at a contact wave with speed $S^* = \frac{S_{T,L}+S_{T,R}}{2}$. However, we want the total mass, momentum, and energy to be conserved. Hence we need to reduce the density, momentum, and energy by a factor

$$f_{L,R} = \frac{S_{HL,R} - S_{L,R}}{S_{HL,R} - \frac{S_L+S_R}{2}}$$

   which is simply the "stretch factor" based on the distances the utilized waves would travel at any given time $t > 0$.

   In terms of primitive variables, it suffices to apply this factor to density and pressure such that energy, momentum, and mass conservation are satisfied. With this, we have expressions for the two star states and a contact speed S^*, and can now easily compute the four necessary fluxes.

   This solution is not very good, because even when applying flux limiters, it will

introduce new peaks at the position of the vacuum. However, due to the lack of a better idea, this is what we have. At least it provides a somewhat usable solution, such that the WAF method can be used in actual hydrodynamics applications without worrying of code crashes should such a condition occur even for very small time steps.

6.2.3 Implementation Details

The cells are stored just like in the Godunov method. Additionally, we need to store

- the three wave speeds $S^{(1)}$, $S^{(2)}$, $S^{(3)}$ as `float`s
- the four fluxes in the four constant regions $\mathbf{F}_L$, $\mathbf{F}_L^*$, $\mathbf{F}_R^*$, $\mathbf{F}_R$ (in that order) as `cstate`s
- the three jumps in density $\Delta q^{(1)}$, $\Delta q^{(2)}$, $\Delta q^{(3)}$ between each wave

for each cell.

The flux at $x_{i+1/2,j}$ and $y_{i,j+1/2}$ are stored in `cflux`of cell `grid[i, j]`. Because we're doing dimensional splitting, it suffices to have only one storage place, as they will be used in successive order. See section 9 for details.

We can afford to store $x_{i+1/2}$ at cell i because we have at least 2 extra virtual boundary cell which is used to apply boundary conditions, so the flux at $x_{-1/2}$ will be stored in `grid[BC-1]`, where `BC` is the number of boundary cells used, defined in `defines.h`.

The related functions are written in `/program/src/solver/waf.c` and `/program/src/solver/waf.h`. The hydro related functions are called in the main loop in `/program/src/main.c` when `solver_step(...)` is called.

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero
- Compute the primitive states for all cells from the updated conserved states
- Impose boundary conditions (section 8)
- Find the maximal timestep that you can do by applying the CFL condition 95.
- Compute fluxes:

- For every cell, first solve the Riemann problem and find the three wave speeds, four fluxes, and three jumps in density. We need all the jumps in density present before we can compute the fluxes, because for the limiting procedure, the k-th wave of cell i is compared to the k-th wave of cells $i \pm 1$. Hence all the waves between all pairs of cells need to be known before the actual intercell flux can be computed. This is done by first solving the Riemann problems between each cell pair in a single loop over all cells, and then the rest of the solution is done in a second loop over all cells.

- For every cell pair, compute the WAF flux using the wave speeds, jumps over density, and fluxes found in the previous loop.

- Store the flux $\mathbf{F}_{i+1/2}$ in the `struct cstate cflux` struct of the cell i.

- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$, the flux $\mathbf{F}_{i+1/2}$ stored in every cell i, and the flux $\mathbf{F}_{i-1/2}$ stored in every cell $i-1$.

6.3 The MUSCL-Hancock Method

6.3.1 Method

The MUSCL-Hackock method is a method of the class of **M**onotone **U**pwind **S**chemes for **C**onservation **L**aws, which try to achieve higher order accuracy by describing the states $\mathbf{U}_i$ of each cell i not by a constant state, but by some higher order interpolation. The MUSCL-Hanckock scheme in particular assumes a piecewise linear state reconstruction, see fig. 11 for an example.

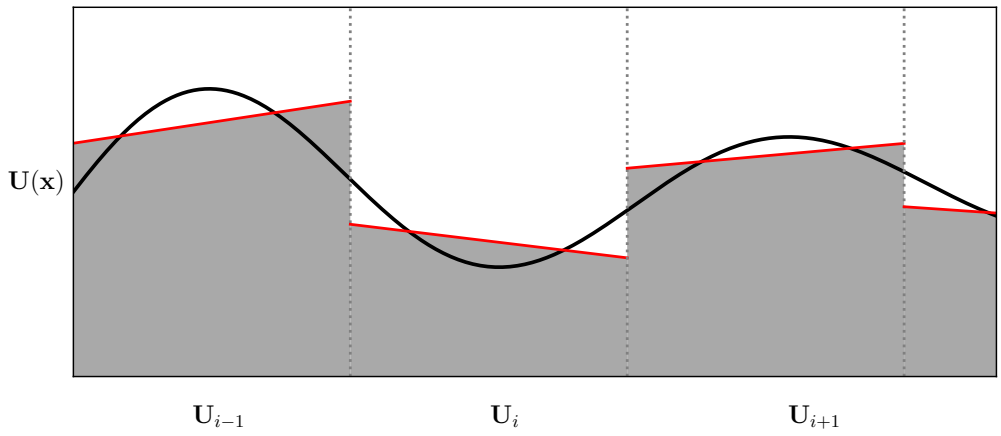

Figure 11: A piecewise linear representation of continuous data among cells.

It solves the hyperbolic conservation law of the form

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U}))}{\partial x} = 0$$

just like all other schemes so far using the explicit conservative formula

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2} - \mathbf{F}_{i+1/2}\right) \tag{107}$$

where $\mathbf{U}_i$ is the volume average of a state in a cell, which is however not constant throughout the cell, but is described by

$$\mathbf{U}_i(x) = \mathbf{U}_i^n + \frac{x - x_i}{\Delta x}\mathbf{s}_i; \quad x \in [0, \Delta x] \tag{108}$$

$\mathbf{s}_i$ is a suitably chosen slope vector of $\mathbf{U}_i(x)$ in cell i. A general slope can be written as

$$\mathbf{s}_i = \frac{1}{2}(1+\omega)(\mathbf{U}_i - \mathbf{U}_{i-1}) + \frac{1}{2}(1-\omega)(\mathbf{U}_{i+1} - \mathbf{U}_i) \tag{109}$$

with $\omega \in [-1, 1]$. For $\omega = 0$, we retrieve the centered scheme. $\omega = 1$ gives us the upwind slope, $\omega = -1$ gives us the downwind slope. In the code, ω can be set as a compile time macro `OMEGA` in `defines.h`.

However, having non-constant states creates a problem for Riemann solvers. We now need to solve the so called *Generalised Riemann Problem* with

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U}))}{\partial x} = 0 \tag{110}$$

$$\mathbf{U}(x, 0) = \begin{cases} \mathbf{U}_i(x), & x < 0 \\ \mathbf{U}_{i+1}(x), & x > 0 \end{cases} \tag{111}$$

As the left and right states change with x, the characteristics are no longer straight lines, which makes things tricky. So instead of dealing with that analytically, the MUSCL-Hancock method tries to compute some sort of "intermediate state" such that in the end, we get an approximate flux that is good enough for our purposes.

In each cell, the extreme values are located at the cell boundaries. They are referred to as boundary extrapolated values, and are given by

$$\mathbf{U}_i^L = \mathbf{U}_i^n - \frac{1}{2}\mathbf{s}_i; \quad \mathbf{U}_i^R = \mathbf{U}_i^n + \frac{1}{2}\mathbf{s}_i; \tag{112}$$

To obtain a second order intercell flux, we first try and find intermediate extrapolated boundary values by applying the same conservative explicit formula on every cell separately over half the timestep. We denote the intermediate boundary extrapolated values as $\overline{\mathbf{U}}_i^L$ and $\overline{\mathbf{U}}_i^R$, and obtain them by computing

$$\overline{\mathbf{U}}_i^L = \mathbf{U}_i^L + \frac{1}{2}\frac{\Delta t}{\Delta x}\left(\mathbf{F}(\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)\right) \tag{113}$$

$$\overline{\mathbf{U}}_i^R = \mathbf{U}_i^R + \frac{1}{2}\frac{\Delta t}{\Delta x}\left(\mathbf{F}(\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)\right) \tag{114}$$

Note that this update depends only on the values inside a cell, and can be computed for every cell individually.

Finally, with the evolved boundary extrapolated values, we can compute the fluxes $\mathbf{F}_{i+1/2} = \mathbf{F}(\mathbf{U}_{i+1/2}(x = 0))$ by solving the Riemann problem at every cell interface and

using the initial values

$$\mathbf{U}_L = \overline{\mathbf{U}}_i^R \tag{115}$$
$$\mathbf{U}_R = \overline{\mathbf{U}}_{i+1}^L \tag{116}$$

and by sampling the solution at $x = 0$.

Note that we only use the intermediate evolved state to estimate the evolved boundary extrapolated values, which in their turn are used to obtain left and right initial values for the Riemann problem. We don't use the evolved state when finally evolving the state!

So in short, what we need to do is

- Compute slope $\mathbf{s}_i$ for each cell
- find evolved boundary extrapolated values using

$$\overline{\mathbf{U}}_i^L = \mathbf{U}_i^L + \frac{1}{2}\frac{\Delta t}{\Delta x}\left(\mathbf{F}(\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)\right)$$
$$\overline{\mathbf{U}}_i^R = \mathbf{U}_i^R + \frac{1}{2}\frac{\Delta t}{\Delta x}\left(\mathbf{F}(\mathbf{U}_i^L) - \mathbf{F}(\mathbf{U}_i^R)\right)$$

- find fluxes $\mathbf{F}_{i+1/2}$ as $\mathbf{F}(\mathbf{U}_{i+1/2})$ with $\mathbf{U}_{i+1/2}$ being the solution to the Riemann problem

$$\mathbf{U}_L = \overline{\mathbf{U}}_i^R \tag{117}$$
$$\mathbf{U}_R = \overline{\mathbf{U}}_{i+1}^L \tag{118}$$

  at $x = 0$

- update the state using

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2} - \mathbf{F}_{i+1/2}\right) \tag{119}$$

A TVD version is obtained by using slope limiters, i.e. replacing the slope $\mathbf{s}_i$ by a limited slope $\bar{\mathbf{s}}_i$ with

$$\overline{\mathbf{s}}_i = \xi(r)\mathbf{s}_i$$
$$r = \frac{\mathbf{U}_i - \mathbf{U}_{i-1}}{\mathbf{U}_{i+1} - \mathbf{U}_i} \qquad \text{for each component of } \mathbf{U}$$

some possible and implemented slope limiters $\xi(r)$ are given in section 7.6.

6.3.2 Implementation Details

The cells are stored just like in the Godunov method. Additionally, we need to store the evolved extrapolated boundary states $\overline{\mathbf{U}}_i^L$ and $\overline{\mathbf{U}}_i^R$ for each cell.

The flux at $x_{i+1/2,j}$ and $y_{i,j+1/2}$ are stored in `cflux`of cell `grid[i, j]`. Because we're doing dimensional splitting, it suffices to have only one storage place, as they will be used in successive order. See section 9 for details.

We can afford to store $x_{i+1/2}$ at cell i because we have at least 2 extra virtual boundary cell which is used to apply boundary conditions, so the flux at $x_{-1/2}$ will be stored in `grid[BC-1]`, where `BC` is the number of boundary cells used, defined in `defines.h`.

The related functions are written in `/program/src/solver/muscl.c` and `/program/src/solver/muscl.h`. The hydro related functions are called in the main loop in `/program/src/main.c` when `solver_step(...)` is called.

The `solver_step(...)` function does the following for the 1D case:

- Reset the stored fluxes from the previous timestep to zero
- Compute the primitive states for all cells from the updated conserved states
- Impose boundary conditions (section 8)
- Find the maximal timestep that you can do by applying the CFL condition 95.
- Compute fluxes:
  - For every cell, find the (limited) slope $\mathbf{s}_i$ and then compute the evolved extrapolated boundary states $\overline{\mathbf{U}}_i^L$ and $\overline{\mathbf{U}}_i^R$ for each cell. We need to solve the Riemann problem with $\mathbf{U}_L = \overline{\mathbf{U}}_i^R$, $\mathbf{U}_R = \overline{\mathbf{U}}_{i+1}^L$ later, so those values need to be present for the neighbouring cells before we can compute the Riemann

problem. This is done by first computing the evolved boundary extrapolated states in a single loop over all cells, and then the rest of the solution is done in a second loop over all cells.

  - For every cell pair $(i, i+1)$, solve the Riemann problem (see section 4) to find the flux $\mathbf{F}_{i+1/2}$ with $\mathbf{U}_L = \overline{\mathbf{U}}_i^R$, $\mathbf{U}_R = \overline{\mathbf{U}}_{i+1}^L$.

  - Store the flux $\mathbf{F}_{i+1/2}$ in the `struct cstate cflux` struct of the cell i. `struct cstate` is a struct that contains the conserved states, i.e. density ρ, momentum ρu_x, ρu_y, and energy E.

- Update the states: Effectively compute $\mathbf{U}^{n+1}$ at this point using $\mathbf{U}_i^n$, the flux $\mathbf{F}_{i+1/2}$ stored in every cell i, and the flux $\mathbf{F}_{i-1/2}$ stored in every cell $i-1$.

7 Slope and Flux Limiters

7.1 Why Limiters?

Limiters are employed because issues arise around numerical schemes due to their discrete nature. For example, a non-limited piecewise linear advection scheme will produce oscillations around jump discontinuities. See **Godunov's Theorem**:

> Linear numerical schemes for solving partial differential equations (PDE's), having the property of not generating new extrema (monotone scheme), can be at most first-order accurate.

So if we want to employ linear higher order schemes, we will generate new extrema, which induce non-physical extrema around steep gradients, in particular around discontinuities. An example of the arising oscillations for linear advection is shown in fig. 12. Obviously, ideally we want both: High order accuracy and no unphysical oscillations. So how do we avoid this problem? This is where limiters enter the game.

7.2 How do they work?

First of all, let's try and express our criterion for the scheme not to introduce spurious oscillations mathematically. We are looking for so called monotone schemes, which don't introduce new extrema into our data. It can be shown[1] that monotone schemes are **Total Variation Diminishing (TVD)**. A method is TVD if:

$$TV(\mathbf{U}^n) \equiv \sum_j |\mathbf{U}_{j+1} - \mathbf{U}_j| \qquad \text{Definition: Total Variation of a state} \tag{120}$$

$$TV(\mathbf{U}^{n+1}) \leq TV(\mathbf{U}^n) \quad \Leftrightarrow \qquad \text{method is TVD} \tag{121}$$

How do we magic up TVD methods? A very popular approach, perhaps the most well known one, relies on the fact that some schemes are in fact TVD, but only for certain cases, which depend on the dataset. Commonly the dependence on the dataset is expressed through the ratio of the upwind difference to the local difference r:

[1] at least for the case of linear advection, it can be done rigorously. For other conservation laws, an empirical approach is usually necessary, as proof becomes exponentially more difficult, or even impossible.

$$r^n_{i-1/2} = \begin{cases} \frac{\mathbf{U}^n_{i-1} - \mathbf{U}^n_{i-2}}{\mathbf{U}^n_i - \mathbf{U}^n_{i-1}} & \text{for } v \geq 0 \\ \frac{\mathbf{U}^n_{i+1} - \mathbf{U}^n_i}{\mathbf{U}^n_i - \mathbf{U}^n_{i-1}} & \text{for } v \leq 0 \end{cases}$$

.

r is a measure of the "curvature", or "monotonicity" in that place, and called the "flow parameter".

Different schemes cover various ranges of values for r for which they are TVD. So the idea is as follows: *Let's use different schemes to solve our problem depending on what states we have currently present in our dataset, such that our method is always TVD.* Which scheme we employ depends on the value of r, and the expressions for fluxes become non-linear because from this point on, we choose the coefficients for the fluxes[2] depending on r. Also remember that first-order schemes are monotone, but very diffusive and have low accuracy. With that in mind, if nothing else works, we can still switch back to a first-order scheme only at the point where we need it.

7.3 Constructing Limiters

In general, the limiters need to be constructed for every method differently, depending on which methods you want to switch between, how you're treating the fluxes, etc. In any case, you start off by introducing coefficients $\phi(r)$ which depend on r, and then apply the TVD constraint, giving you a region in the $r - \phi(r)$ plot: The TVD constraint gives you inequalities, thus some upper and lower limits for ϕ at given r. Within this TVD region, a multitude of possible expressions for ϕ are possible. See for example fig. 13, where some popular limiters are plotted within the grey TVD region.

Depending on the way you're trying to get high-order accuracy, you can have various approaches to introduce a limiter function $\phi(r)$ in your scheme:

- If the high-order approach is to treat the states as not constant, but some higher order interpolation, e.g. piecewise linear advection, or any MUSCL scheme, then we use *slope limiters* to modify the expression for the slope of the piecewise linear states such that the resulting method is TVD.

- If we leave the states constant, but use a more clever way to get fluxes, e.g. WAF schemes, then we need *flux limiters* directly, as no slopes are available.

[2] the schemes will always be of the form $\mathbf{U}^{n+1}_i = \sum_{k=-k_L}^{k_R} \beta_{i+k} \mathbf{U}_{i+k}$, but from now on, $\beta = \beta(r)$

7.4 Slope Limiters

Let's start with slope limiters. The idea is to compute the slope of the states in way that is useful for us based on the current situation of the gas state that we're solving for.

7.4.1 Slope Limiters for Linear Advection

The choice of the slope can be expressed via a function $\phi(r)$. For linear advection, we are solving the equation:

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2}^{n+1/2} - \mathbf{F}_{i+1/2}^{n+1/2}\right) \tag{122}$$

Recall that we're working for a scalar conservation law here, and just treat every component of the state $\mathbf{U}$ independently.

If we assume that the states $\mathbf{U}_i$ are piecewise linear, i.e.

$$\mathbf{U}(x) = \mathbf{U}_i(x_i) + \mathbf{s}\cdot(x - x_i) \quad \text{for } x_{i-1/2} \le x \le x_{i+1/2} \tag{123}$$

then the expression for the fluxes is given by

$$\mathbf{F}_{i-1/2}^{n+1/2} = \begin{cases} v_{i-1/2}\cdot\mathbf{U}_{i-1}^n + \frac{1}{2}v_{i-1/2}\cdot\mathbf{s}_{i-1}^n(\Delta x - v_{i-1/2}\Delta t) & \text{for } v \ge 0 \\ v_{i-1/2}\cdot\mathbf{U}_i^n - \frac{1}{2}v_{i-1/2}\cdot\mathbf{s}_i^n(\Delta x + v_{i-1/2}\Delta t) & \text{for } v \le 0 \end{cases}$$

$$\mathbf{F}_{i+1/2}^{n+1/2} = \begin{cases} v_{i+1/2}\cdot\mathbf{U}_i^n + \frac{1}{2}v_{i+1/2}\cdot\mathbf{s}_i^n(\Delta x - v_{i+1/2}\Delta t) & \text{for } v \ge 0 \\ v_{i+1/2}\cdot\mathbf{U}_{i+1}^n - \frac{1}{2}v_{i+1/2}\cdot\mathbf{s}_{i+1}^n(\Delta x + v_{i+1/2}\Delta t) & \text{for } v \le 0 \end{cases}$$

Let us introduce the unitless function $\phi(r)$ as

$$\phi(r_{i+1/2}) = \phi_{i+1/2} = \begin{cases} \frac{\Delta x}{\mathbf{U}_{i+1}-\mathbf{U}_i}\cdot\mathbf{s}_i & \text{for } v \ge 0 \\ \frac{\Delta x}{\mathbf{U}_{i+1}-\mathbf{U}_i}\cdot\mathbf{s}_{i+1} & \text{for } v \le 0 \end{cases} \tag{124}$$

$$\phi(r_{i-1/2}) = \phi_{i-1/2} = \begin{cases} \frac{\Delta x}{\mathbf{U}_i - \mathbf{U}_{i-1}} \cdot \mathbf{s}_{i-1} & \text{for } v \geq 0 \\ \frac{\Delta x}{\mathbf{U}_i - \mathbf{U}_{i-1}} \cdot \mathbf{s}_i & \text{for } v \leq 0 \end{cases} \tag{125}$$

Such that

$$\mathbf{F}_{i-1/2}^{n+1/2} = \begin{cases} v_{i-1/2} \cdot \mathbf{U}_{i-1}^n + \frac{1}{2}|v_{i-1/2}| \left(1 - \frac{|v_{i-1/2}|\Delta t}{\Delta x}\right) \phi_{i-1/2}(\mathbf{U}_i - \mathbf{U}_{i-1}) & \text{for } v \geq 0 \\ v_{i-1/2} \cdot \mathbf{U}_i^n + \frac{1}{2}|v_{i-1/2}| \left(1 - \frac{|v_{i-1/2}|\Delta t}{\Delta x}\right) \phi_{i-1/2}(\mathbf{U}_i - \mathbf{U}_{i-1}) & \text{for } v \leq 0 \end{cases}$$

$$\mathbf{F}_{i+1/2}^{n+1/2} = \begin{cases} v_{i+1/2} \cdot \mathbf{U}_i^n + \frac{1}{2}|v_{i+1/2}| \left(1 - \frac{|v_{i+1/2}|\Delta t}{\Delta x}\right) \phi_{i+1/2}(\mathbf{U}_{i+1} - \mathbf{U}_i) & \text{for } v \geq 0 \\ v_{i+1/2} \cdot \mathbf{U}_{i+1}^n + \frac{1}{2}|v_{i+1/2}| \left(1 - \frac{|v_{i+1/2}|\Delta t}{\Delta x}\right) \phi_{i+1/2}(\mathbf{U}_{i+1} - \mathbf{U}_i) & \text{for } v \geq 0 \end{cases}$$

Depending on our choice of ϕ, we can get different slopes, which are not limited, but well known schemes: Here for positive velocity only, and for $r = r_{i-1/2}$:

$$\phi(r) = 0 \rightarrow \mathbf{s}_i = 0 \qquad \text{No slopes; Piecewise constant method.} \tag{126}$$

$$\phi(r) = 1 \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_i - \mathbf{U}_{i-1}}{\Delta x} \qquad \text{Downwind slope (Lax-Wendroff)} \tag{127}$$

$$\phi(r) = r \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_{i-1} - \mathbf{U}_{i-2}}{\Delta x} \qquad \text{Upwind slope (Beam-Warming)} \tag{128}$$

$$\phi(r) = \frac{1}{2}(1 + r) \rightarrow \mathbf{s}_i = \frac{\mathbf{U}_i - \mathbf{U}_{i-2}}{2\Delta x} \qquad \text{Centered slope (Fromm)} \tag{129}$$

$$\tag{130}$$

Note that taking the downwind slope is very different from doing downwind differencing! We only use the downwind value to estimate the state inside the cell, not to compute derivatives.

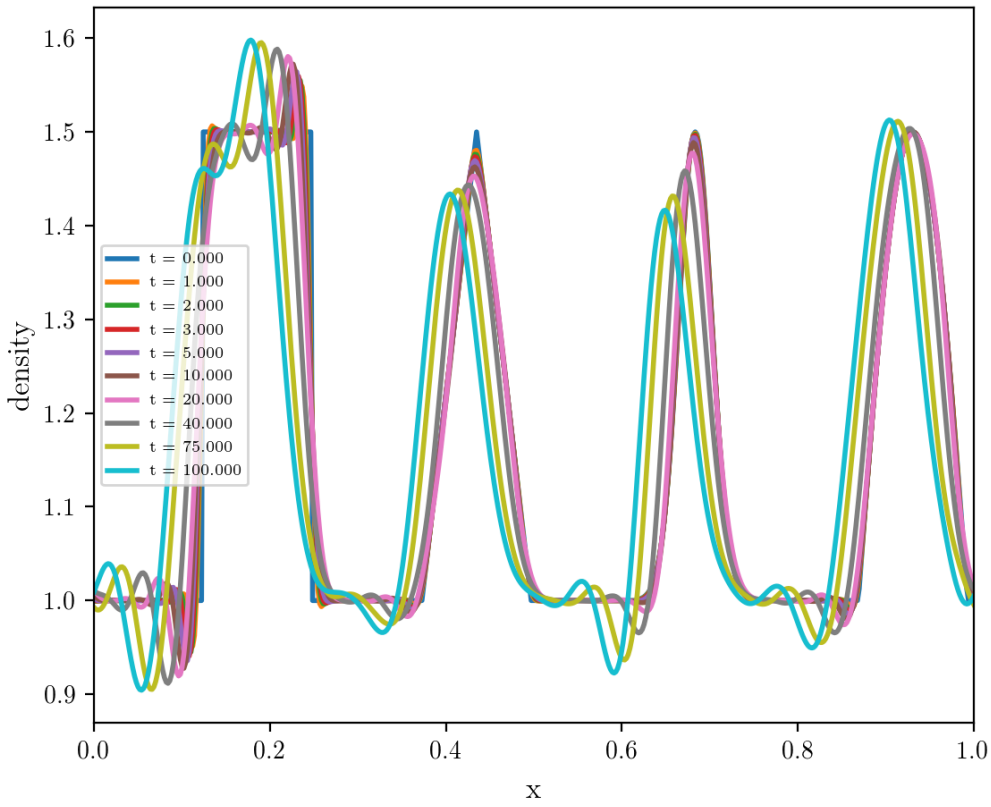

Figure 12: Piecewise linear advection with positive fixed global velocity $v_x = 1$ at different times. $C_{CFL} = 0.9$, $nx = 100$. The oscillations are a natural consequence according to Godunov's theorem.

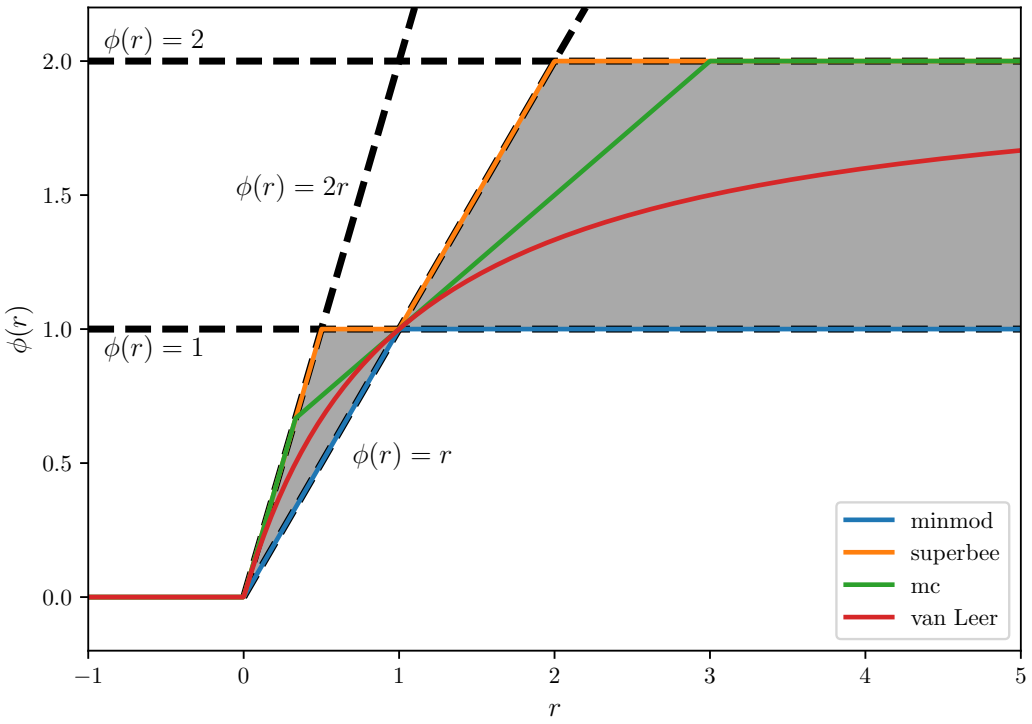

Figure 13: The behaviour for different slope limiters. The grey zone is the zone allowed by the conditions 131 - 134, and is the region in which $\phi(r)$ yields a TVD method.

To remove the oscillations that we see in fig. 12, we want to go back to a first order expression (piecewise constant expression) when we find an oscillation, i.e. when the numerator and denominator have different signs. We get the piecewise constant expression for $\phi(r) = 0$.

$\Rightarrow$ For slope limiters, we must have $r < 0 \Rightarrow \phi = 0$

Other restrictions follow from the constraint that the method should be TVD and continuous (see Sweby [1984]):

$$r \leq \phi(r) \leq 2r \qquad 0 \leq r \leq 1 \tag{131}$$

$$1 \leq \phi(r) \leq r \qquad 1 \leq r \leq 2 \tag{132}$$

$$1 \leq \phi(r) \leq 2 \qquad r > 2 \tag{133}$$

$$\phi(1) = 1 \tag{134}$$

Effectively, this defines regions in the $r - \phi(r)$ diagram through which the limiters are allowed to pass such that they are still TVD (fig 13). The implemented limiters are given in section 7.6.

The functions $\phi(r)$ are typically called *flux limiters*, even though we are using them here to limit slopes. Because who needs consistency in their life, right? In any case, in principle you need to derive your limiters separately for every method that you are trying to use. Sometimes you get lucky and can find relations to well known functions like $\phi(r)$, see e.g. 7.5.1.

However, this is not always the case. For the MUSCL-Hancock method, we need slope limiters $\xi(r)$ that are different from $\phi(r)$. But what we can do is define *analogous* limiters based on how they explore the respective TVD region. For example, see how the superbee limiter in fig. 13 follows the upper boundary of the TVD region? So we name the $\xi(r)$ that follows the upper boundary of the TVD region for the MUSCL-Hancock method also superbee. Recall that we defined the slope for the MUSCL-Hancock method as

$$\mathbf{s}_i = \frac{1}{2}(1+\omega)(\mathbf{U}_i - \mathbf{U}_{i-1}) + \frac{1}{2}(1-\omega)(\mathbf{U}_{i+1} - \mathbf{U}_i) \tag{135}$$

The superbee slope limiter $\xi(r)$ and the superbee flux limiter $\phi(r)$ are plotted w.r.t. r in figure 14 for the cases $\omega = 1$ (upwind slope), $\omega = 0$ (centered slope), and $\omega = -1$ (downwind slope). You can clearly see that they give different values for $r > 1$. The implemented slope limiters $\xi(r)$ are given in section 7.6.

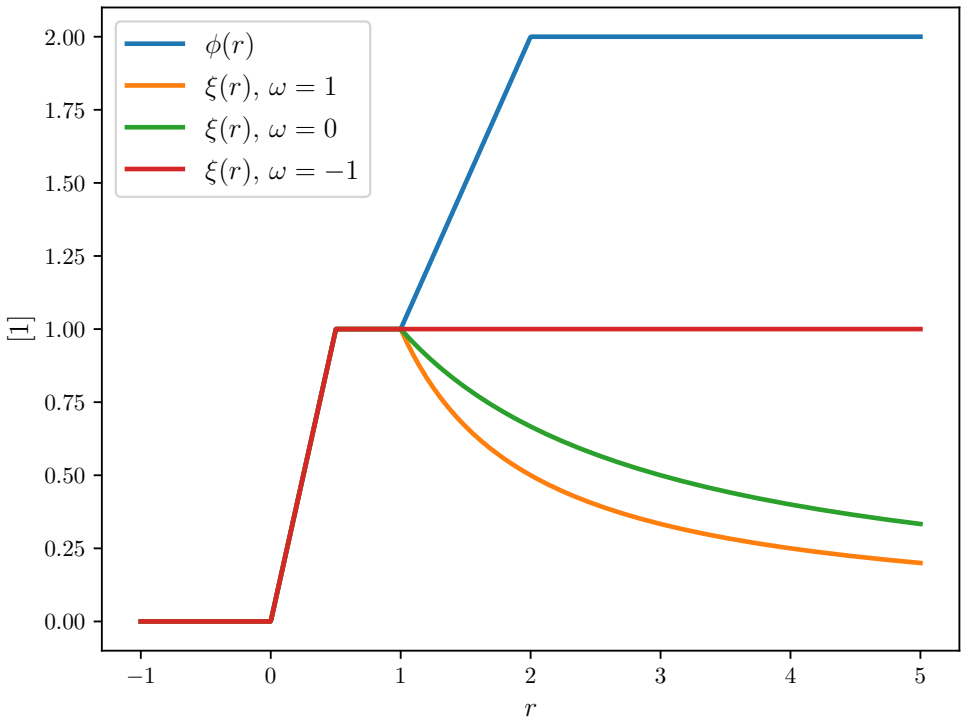

Figure 14: Comparison of the superbee slope limiter $\xi(r)$ for $\omega = 1$ (upwind slope), $\omega = 0$ (centered slope), and $\omega = -1$ (downwind slope) and the flux limiter $\phi(r)$. They are equivalent, but not equal.

7.5 Flux Limiters

In general, flux limiters need to be constructed for each method individually. There are some general approaches to help you get started, see e.g. Toro [1999].

Lucky for us, for linear advection, we can find relations between the slope limiters $\phi(r)$ described in section 7.4, while the full expressions for $\phi(r)$ are given in section 7.6.

7.5.1 The limited WAF flux

For WAF advection, we have the expression for a limited flux

$$\mathbf{F}^{n+1/2}_{i+1/2} = \frac{1}{2}(1 + sign(v)\psi_{i+1/2})\ v\mathbf{U}^n_i + \frac{1}{2}(1 - sign(v)\psi_{i+1/2})\ v\mathbf{U}^n_{i+1} \tag{136}$$

The choice

$$\psi = |c|$$

recovers the non-limited expression, where

$$c = \frac{\Delta t v}{\Delta x}$$

It can be shown that

$$\psi_{i+1/2} = 1 - (1 - |c|)\phi_{i+1/2}(r) \tag{137}$$

Some possible choices for the limiter function $\phi(r)$ are given in section 7.6.

7.6 Implemented Limiters

7.6.1 Flux limiters $\phi(r)$

Some popular (and implemented) limiters are:

$$\text{Minmod} \qquad \phi(r) = \text{minmod}(1, r) \tag{138}$$

$$\text{Superbee} \qquad \phi(r) = \max(0, \min(1, 2r), \min(2, r)) \tag{139}$$

$$\text{MC (monotonized cenral-difference)} \qquad \phi(r) = \max(0, \min((1 + r)/2, 2, 2r)) \tag{140}$$

$$\text{van Leer} \qquad \phi(r) = \frac{r + |r|}{1 + |r|} \tag{141}$$

where

$$\text{minmod}(a, b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } ab > 0 \\ b & \text{if } |a| > |b| \text{ and } ab > 0 \\ 0 & \text{if } ab \leq 0 \end{cases} \tag{142}$$

The functions $\phi(r)$ are shown in fig. 13.

7.6.2 Slope limiters $\xi(r)$

Some popular (and implemented) limiters are:

Minmod
$$\xi(r) = \begin{cases} 0, & r \le 0 \\ r, & 0 \le r \le 1 \\ \min\{1, \xi_R(r)\}, & r \ge 1 \end{cases} \tag{143}$$

Superbee
$$\xi(r) = \begin{cases} 0, & r \le 0 \\ 2r, & 0 \le r \le \frac{1}{2} \\ 1, & \frac{1}{2} \le r \le 1 \\ \min\{r, \xi_R(r), 2\}, & r \ge 1 \end{cases} \tag{144}$$

van Leer
$$\xi(r) = \begin{cases} 0, & r \le 0 \\ \min\{\frac{2r}{1+r}, \xi_R(r)\}, & r \ge 0 \end{cases} \tag{145}$$

$$\tag{146}$$

where

$$\xi_R(r) = \frac{2}{1 - \omega + (1+\omega)r} \tag{147}$$

$$r = \frac{\mathbf{U}_i - \mathbf{U}_{i-1}}{\mathbf{U}_{i+1} - \mathbf{U}_i} \qquad \text{for each component of } \mathbf{U} \tag{148}$$

7.7 Implementation Details

All the main functions for limiters are written in `/program/src/limiter.c` and `/program/src/limiter.h`.

If we are using slope limiters, it first finds the relevant states $\mathbf{U}_i$ that enter the equation, and then call a function to compute $\phi(r)$. The specific way how $\phi(r)$ is computed is then defined in the specific files in `/program/src/limiter/`, depending on what limiter you want.

8 Boundary Conditions

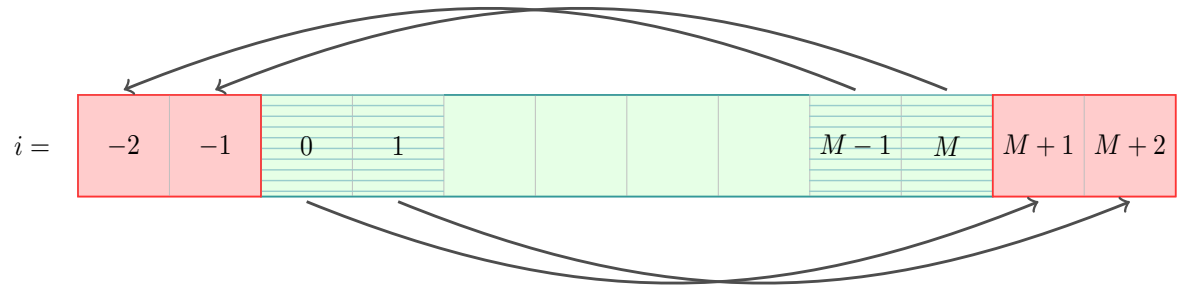

Figure 15: Method to obtain periodic boundary conditions. The ghost cells are red, the arrows show what will be copied where.

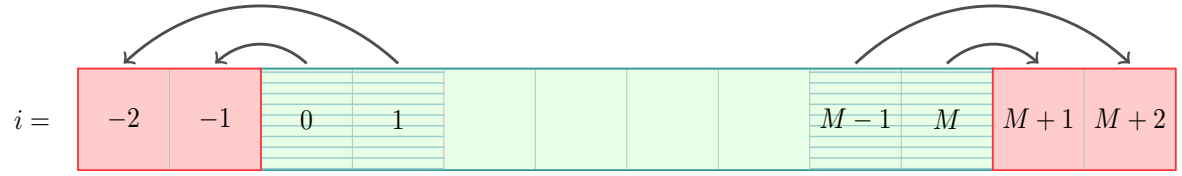

Figure 16: Method to obtain wall boundary conditions. The ghost cells are red, the arrows show what will be copied where.

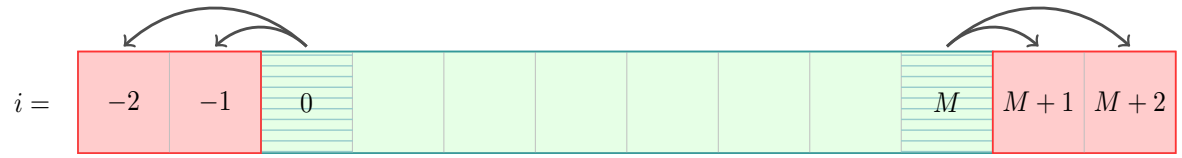

Figure 17: Method to obtain transmissive boundary conditions. The ghost cells are red, the arrows show what will be copied where.

There are tricks how to obtain different kinds of boundary conditions. In every case, we add additional cells ("*ghost cells*") in every dimension so we can simulate the desired behaviour. How many cells you need to add depends on the methods (and mostly stencils) you use. If you only take into account one neighbouring cell, then one ghost cell on every boundary suffices. In figures 15, 16, and 17, two ghost cells for a 1D grid are drawn.

Suppose we have 1D grid with M cells and require 2 ghost cells each, which will have indices -2, -1, $M+1$, and $M+2$. Then we can get:

- **periodic boundary conditions:**

what goes over the right edge, comes back in over the left edge, and vice versa. We achieve this behaviour by enforcing (fig. 15)

$$\begin{aligned}\mathbf{U}_{-2} &= \mathbf{U}_{M-1}\\ \mathbf{U}_{-1} &= \mathbf{U}_{M}\\ \mathbf{U}_{M+1} &= \mathbf{U}_{0}\\ \mathbf{U}_{M+2} &= \mathbf{U}_{1}\end{aligned}$$

- **reflective boundary conditions:**

  pretend there is a wall at the boundary. We achieve that by "mirroring" the cells next to the boundary (fig 16):

$$\begin{aligned}\mathbf{U}_{-2} &= \mathbf{U}_{1}\\ \mathbf{U}_{-1} &= \mathbf{U}_{0}\\ \mathbf{U}_{M+1} &= \mathbf{U}_{M}\\ \mathbf{U}_{M+2} &= \mathbf{U}_{M-1}\end{aligned}$$

  However, every directional component (i.e. velocities/momentum) needs to have the negative value in the ghost cell compared to the real cell.

  This is valid for the Euler equations, but not for linear advection. In fact, having reflecting boundary conditions for constant linear advection doesn't really make sense.

- **transmissive boundary conditions**:

  Just let things flow out however they want. We achieve this by copying the last boundary cell over and over again, such that it looks that the fluid appears to have that state infinitely, and there are no net fluxes to interfere with the hydrodynamics inside the actual grid (fig. 17)

$$\begin{aligned}\mathbf{U}_{-2} &= \mathbf{U}_{0}\\ \mathbf{U}_{-1} &= \mathbf{U}_{0}\end{aligned}$$

$$\mathbf{U}_{M+1} = \mathbf{U}_M$$
$$\mathbf{U}_{M+2} = \mathbf{U}_M$$

9 Solving Multidimensional Problems: Dimensional Splitting

To go from one to multiple dimensions, it is tempting to just extend the one dimensional discretisation. For example, starting with the conservation law

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F}(\mathbf{U}) + \frac{\partial}{\partial y}\mathbf{G}(\mathbf{U}) = 0 \tag{149}$$

and just apply Godunov's finite volume method:

$$\mathbf{U}_{i,j}^{n+1} = \mathbf{U}_{i,j}^{n} + \frac{\Delta t}{\Delta x}\left(\mathbf{F}_{i-1/2,j} - \mathbf{F}_{i+1/2,j}\right) + \frac{\Delta t}{\Delta x}\left(\mathbf{G}_{i,j-1/2} - \mathbf{G}_{i,j+1/2}\right) \tag{150}$$

However, this is a bit of a problem. The upwinding here is not complete. Consider the 2D advection equation

$$\frac{\partial}{\partial t}q + u\frac{\partial}{\partial x}q + v\frac{\partial}{\partial y}q = 0 \tag{151}$$

Now suppose we have advecting velocities $u = v = 1$, i.e. the advection velocity is along the diagonal. Then our method reads

$$q_{i,j}^{n+1} = q_{i,j}^{n} + u\frac{\Delta t}{\Delta x}\left(q_{i-1/2,j} - q_{i+1/2,j}\right) + v\frac{\Delta t}{\Delta x}\left(q_{i,j-1/2} - q_{i,j+1/2}\right) \tag{152}$$

This expression doesn't involve $q_{i-1,j-1}$ at all, but that's actually the value that should be advected to $q_{i,j}$ in the next timestep!

One way of doing things is to actually formulate more sophisticated unsplit methods that involve the appropriate stencils. We shan't do that here though. We make use of dimensional splitting. Instead of solving

$$\begin{cases} \text{PDE:} & \frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F}(\mathbf{U}) + \frac{\partial}{\partial y}\mathbf{G}(\mathbf{U}) = 0 \\ \text{IC:} & \mathbf{U}(x, y, t^n) = \mathbf{U}^n \end{cases} \tag{153}$$

we do it in 2 (3 for 3D) steps:

Step 1: We obtain a intermediate result $\mathbf{U}^{n+1/2}$ by solving the "x - sweep" over the full time interval Δt :

$$\begin{cases} \text{PDE:} & \frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F}(\mathbf{U}) = 0 \\ \text{IC:} & \mathbf{U}^n \end{cases} \tag{154}$$

and then we evolve the solution to the final $\mathbf{U}^{n+1}$ by solving the "y - sweep" over the full time interval Δt :

$$\begin{cases} \text{PDE:} & \frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial y}\mathbf{G}(\mathbf{U}) = 0 \\ \text{IC:} & \mathbf{U}^{n+1/2} \end{cases} \tag{155}$$

using the 1D methods that are described. This is called "Strang splitting" and gives us a first order accurate method.

For a second order accurate method, we need to do one more step in 2D:

1. Get $\mathbf{U}^{n+1/4}$ by doing an x - sweep over the time interval $\Delta t/2$

2. Get $\mathbf{U}^{n+3/4}$ by doing an y - sweep over the time interval Δt and using the intermediate solution $\mathbf{U}^{n+1/4}$

3. Get $\mathbf{U}^{n+1}$ by doing an x - sweep over the time interval $\Delta t/2$ and using the intermediate solution $\mathbf{U}^{n+3/4}$

However, if Δt is kept constant over multiple time steps, which is often the case for linear advection, we can get the first order method described above to be second order accurate by switching the order of the sweeps every time step (see LeVeque [2002]), i.e. if for one time step we did the x-sweep first, in the next one we do the y-sweep first, then in the next time step we do the x-sweep first again, and so on. The order of the sweeps doesn't matter as long as all sweeps have been done in the end, and by alternating the order we recover the second order method formalism, but with a twice as big time step.

10 Source Terms: Dealing with External Forces

10.1 Implemented Sources

10.2 Implemented Integrators

References

Eleuterio F Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction; 2nd ed.* Springer, Berlin, 1999. URL `http://cds.cern.ch/record/404378`.

P. K. Sweby. High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws. *SIAM Journal on Numerical Analysis*, 21(5):995–1011, October 1984. ISSN 0036-1429, 1095-7170. doi: 10.1137/0721062.

Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. doi: 10.1017/CBO9780511791253.