

# **Lecture Numerical Fluid Dynamics:**

Lecture given by

C.P. Dullemond  
and  
R. Kuiper

Course given at Heidelberg University  
Summer semester 2008



# About the lecture

## Introduction

The topic of *hydrodynamics* or *fluid dynamics* describes the time-dependent or stationary flow of fluids or gases. The name originates from fluid flow (hence the names 'hydro' and 'fluid'), but in physics this name is also used to describe *gas* flows, and in astrophysics it is nearly exclusively used in that way.

Hydrodynamics is a topic that is of fundamental importance to many applications in industry. For instance, it is used to model the flow around airfoils and entire airplanes, the aerodynamics of sports cars, how water flows through pipes and ducts, the aerodynamic stability of constructions, complex combustion processes in chemical reactors, etc. It also plays an important role in many scientific studies, for instance in meteorology and climate research (modeling ocean and air currents and the formation of weather fronts) and in astrophysics where it is used for practically all topics: star- and planet formation, the formation of galaxies, the formation of structure in the early universe, the accretion of gases onto black holes and neutron stars, the colliding winds of evolved stars etc.

In practically all applications the problems are already so complex that analytical methods are inadequate. The solution is almost always sought using *numerical methods*, which, implemented in computer programs, can give numerical solutions for given initial- and or boundary conditions.

This lecture aims to give an introduction to these numerical methods. We will focus on:

- Gaining an understanding of the mathematical properties of the equations of hydrodynamics, which underly many of the algorithms to solve them.
- Gaining an understanding of the methods, how and why they work and what their limitations are.
- Hands-on experiments (in the practicum) by self-made programs for simple problems
- Hands-on experiments with some off-the-shelf programs for more complex and realistic problems
- Example applications of the methods, and their background.

## Various methods

There is no single numerical recipe for all hydrodynamic problems. For instance, for modeling weather patterns on Earth one requires methods that can handle the strong contrast of distance scales in the atmosphere, where the vertical height (roughly 20 km) is much less than the size of a typical depression (1000 km). The problem of flow around airfoils (=wings) typically require methods that have their grids adapted to the wing-shape. In astronomy one typically has 3-D flows of interstellar gas which can be extremely supersonic (involving very strong shocks) and can have extreme density contrasts (factors of a million) and spatial scales differing by similar

factors. In other applications one may have more interest in detailed steady-state flows which are best solved using special-purpose potential solvers. In some cases a detailed treatment of turbulence is required, which again puts different constraints on the numerical methods used.

Also, each problem may involve different additional physics. For instance, astrophysical hydrodynamics usually involves the interaction between gas and a radiation field. This requires the combined hydrodynamic and radiative transfer equations to be solved. This is called *radiation-hydrodynamics*, and is so complex that in many applications it is still in its infancy. Also in astrophysics magnetic fields play a crucial role: gas motions can generate (or at least amplify) magnetic fields through dynamo action and these fields can strongly affect the motion of the gas. An example of this is the magnetosphere of the sun, where solar eruptions are clearly an interplay between gas and magnetic fields. This is called *magneto-hydrodynamics*, and is a field of applied mathematics that is still under development. In meteorology, in addition to the interaction with radiation from the sun, the air currents are affected by the evaporation and condensation of water, and the precipitation that follows from this. Moreover, in this case the coriolis forces by the Earth's rotation are of crucial importance, too.

In this lecture we put the main focus on normal hydrodynamics with simple source terms such as gravity. We will focus mostly on *grid-based methods for time-dependent fluid flows*. However, there will be a number of excursions away from that main line, so as to give a taste of the more complex versions of hydrodynamic study.

## Literature

This lecture is inspired by, and partially follows, the following books:

- Randall J. LeVeque, "Finite Volume Methods for Hyperbolic Problems" (Cambridge Texts in Applied Mathematics)
- Bodenheimer, Laughlin, Rozyczka and Yorke, "Numerical Methods in Astrophysics: An Introduction" (Series in Astronomy and Astrophysics, Taylor and Francis)
- Toro, E.F. "Riemann Solvers and Numerical Methods for Fluid Dynamics" (Springer Verlag)
- Collela and Puckett, "Modern Numerical Methods for Fluid Flow"
- Liepmann, H. W. and Roshko, A., "Elements of Gas Dynamics" (Dover Publications)
- Ferziger, J. H. and Peric, M., "Computational Methods for Fluid Dynamics" (Springer Verlag)

# Chapter 1

## Equations of hydrodynamics

In this chapter we will be concerned with compressible gas flows. This forms the main focus of this lecture.

### 1.1 Basic quantities

The basic quantities that describe the gas are:

Name	Symbol	Unit (CGS)	Unit (SI)
Gas density	$\rho$	g/cm <sup>3</sup>	kg/m <sup>3</sup>
Particle number density	$N$	1/cm <sup>3</sup>	1/m <sup>3</sup>
Velocity	$\vec{u}$	cm/s	m/s
Temperature	$T$	K	K
Sound speed	$C_s$	cm/s	m/s
Isothermal sound speed	$c_s$	cm/s	m/s
Pressure	$P$	dyne/cm <sup>2</sup>	N/m <sup>2</sup>
Internal energy density	$E$	erg/cm <sup>3</sup>	J/m <sup>3</sup>
Internal specific energy	$e$	erg/g	J/kg
Internal specific enthalpy	$h$	erg/g	J/kg
Total specific energy	$e_{\text{tot}}$	erg/g	J/kg
Total specific enthalpy	$h_{\text{tot}}$	erg/g	J/kg

This is a large set of variables. But only 3 of these are independent:

$$\rho, \quad \vec{u}, \quad e \quad (1.1)$$

Since the velocity  $\vec{u}$  consists of 3 components, this means that there are in fact 5 independent variables:

$$\rho, \quad u_x, \quad u_y, \quad u_z, \quad e \quad (1.2)$$

These are the 5 quantities that we will model in the computer programs we will discuss in this lecture. Usually it is unpleasant to write  $u_x$ ,  $u_y$  and  $u_z$ , so we will, from now on, write:

$$\vec{u} \equiv (u, v, w) \quad (1.3)$$

where  $u = u_x$ ,  $v = u_y$  and  $w = u_z$ .

All other quantities are linked to the above 5 basic quantities. Most of the relations are fundamental. The density and number density are related by:

$$\rho = N\mu \quad (1.4)$$

The  $\mu$  is the mean weight of the gas particles (in gram). For atomic hydrogen, for example,  $\mu = 1.67 \times 10^{-24}$  g, and for a typical cosmic mixture of molecular hydrogen and atomic helium one has roughly  $\mu = 3.8 \times 10^{-24}$ . The internal energy density is linked to the specific internal energy:

$$E \equiv \rho e \quad (1.5)$$

The specific enthalpy is linked to the specific energy, the density and the pressure as:

$$h \equiv e + \frac{P}{\rho} \quad (1.6)$$

The total specific energy is the internal (heat) energy plus the kinetic energy:

$$e_{\text{tot}} \equiv e + \frac{1}{2}|\vec{u}|^2 \quad (1.7)$$

and the same for the enthalpy:

$$h_{\text{tot}} \equiv h + \frac{1}{2}|\vec{u}|^2 \quad (1.8)$$

How the internal specific energy  $T$ , the adiabatic sound speed  $C_s$ , the isothermal sound speed  $c_s$ , the pressure  $P$  and the internal energy  $E$  depend on each other is dependent on the properties of the gas. This is described by the so-called *equations of state* for the gas.

## 1.2 Ideal gases

In this lecture we will be mainly concerned with an *ideal gas*. Often this is also called a *perfect fluid*. This is the simplest kind of gas, and of course this is an idealization of real gases. No gas is perfect. But if densities are low enough and temperatures are high enough, a gas typically starts to behave more and more as an ideal gas. For astrophysical purposes the validity of this simple equation of state is mixed: For modeling the interiors of stars or gaseous planets this equation of state is often inadequate. But when modeling gas flows around stars or in interstellar space, the ideal gas equation of state is very accurate! Therefore in astrophysics (with the exception of stellar and planetary interiors) the ideal gas law is almost always used.

In ideal gases the elementary particles (atoms or molecules) are considered to be freely moving particles, moving in straight lines and once in a while colliding with another particle and thereby changing direction. The collision events are always two-body processes, perfectly elastic, and the particles are so small that they are many orders of magnitude smaller than the mean-free path between two collisions. Also the particles are assumed to have only interactions with each other through localized collisions, so with the exception of these collisions they move along straight lines.

For the ideal gas law to work, and indeed for any *fluid-description* of a gas to work, the mean free path between consecutive collisions must be orders of magnitude smaller than the typical scales at which we study our flows. For all of the examples we shall see this condition is guaranteed.

The ideal equation of state links the temperature, pressure and number density  $N$  of the gas particles:

$$P = NkT \quad \leftrightarrow \quad P = \frac{\rho kT}{\mu} \quad (1.9)$$

where  $k = 1.38 \times 10^{-16}$  erg/K is the Boltzmann constant. Another aspect of the ideal gas is the equation of state relating the pressure to the internal specific energy  $e$

$$P = (\gamma - 1)\rho e \quad (1.10)$$

where  $\gamma$  is the adiabatic index of the gas. In numerical hydrodynamics this equation for the state of the gas is more relevant, and we will use this one typically, instead of Eq. (1.9). The adiabatic index  $\gamma$  of an ideal gas is derived from the number of degrees of freedom of each gas particle:

$$\gamma = \frac{f + 2}{f} \quad (1.11)$$

In case of an atomic hydrogen gas each particle has merely 3 degrees of freedom: the three translational degrees of freedom. So for atomic gas one has  $\gamma = 5/3$ . For diatomic molecular gas, such as most of the gas in the Earth's atmosphere (mainly  $N_2$  and  $O_2$ ), as well as for molecular hydrogen gas in interstellar molecular clouds ( $H_2$ ) the gas particles have, in addition to the three translational degrees of freedom also 2 rotational ones, yielding  $f = 5$  and thereby  $\gamma = 7/5$ . For adiabatic compression/expansion of an ideal gas with adiabatic index  $\gamma$  one can relate the pressure to the density at all times by

$$P = K\rho^\gamma \quad (1.12)$$

where  $K$  is a constant during the adiabatic process. The  $K$  is in some sense a form of entropy. In fluids without viscosity, nor shocks or heating or cooling processes  $K$  remains constant for any fluid package. The adiabatic sound speed  $C_s$  is by definition:

$$C_s^2 \equiv \frac{\partial P}{\partial \rho} = \gamma \frac{P}{\rho} = \gamma(\gamma - 1)e \quad (1.13)$$

For isothermal sound waves (in which  $K$  is not constant, but  $e$  is), the sound speed is:

$$c_s^2 = \frac{P}{\rho} = (\gamma - 1)e \quad (1.14)$$

### 1.2.1 Most used variables in numerical hydrodynamics

Of all the above quantities we typically use only a few. The most used symbols are:

$$\rho, \quad e, \quad h, \quad P \quad \text{and} \quad \vec{u} \quad (1.15)$$

## 1.3 The first law of thermodynamics

In compressible hydrodynamics a fluid or gas parcel undergoes many compression and decompression events. When a gas parcel is compressed, its temperature tends to rise and when it is decompressed the temperature drops. In the absense of any heat transfer to or from the parcel we know from the first law of thermodynamics that

$$dU = -PdV \quad (1.16)$$

$V$  is the volume occupied by the gas parcel,  $U = \int_V \rho e dV$  is the total thermal energy in the volume  $V$  and  $P$  is the pressure. Eq.(1.16) is valid for *adiabatic (de-)compression*, i.e. (de-)compression of the gas parcel without any heat exchange. With this expression we can derive Eqs. (1.10,1.12) from the assumption that  $e \propto \rho^\alpha$ , where  $\alpha$  is for now some arbitrary constant which we shall constrain later. If we assume that the density in the parcel is constant and if we define the mass of the parcel  $M = \rho V$  (which is constant) then we can rewrite Eq.(1.16):

$$Me d \lg e = M \frac{P}{\rho} d \lg \rho \quad (1.17)$$

yielding:

$$P = \rho e \frac{d \lg e}{d \lg \rho} = \alpha \rho e \quad (1.18)$$

Since  $e \propto \rho^\alpha$ , we see that  $P \propto \rho^{\alpha+1}$ . Since we define  $\gamma$  in Eq.(1.12) such that  $P \propto \rho^\gamma$ , we see that  $\alpha = \gamma - 1$ , which means we can write

$$P = (\gamma - 1) \rho e \quad (1.19)$$

proving Eq.(1.10), which is the equation of state for a polytropic gas.

The equation of state tells us what the pressure  $P$  is, given the density  $\rho$  and the internal heat energy  $e$ . We have seen here that upon adiabatic compression or decompression the  $\rho$  and  $e$  are related to each other. However, the precise proportionality of the relation is fixed by the constant  $K$  in Eq.(1.12). So, given the constant  $K$ , each density  $\rho$  has its own well-defined thermal energy  $e$ . Two gas parcels with the same  $K$  lie on the same adiabat. Upon compression and decompression of a parcel the  $K$  does not change, unless heat is transferred to or from the parcel.

The constant  $K$  is related to the *entropy* of the gas. The full first law of thermodynamics reads:

$$dU = dQ - PdV = TdS - PdV \quad (1.20)$$

where  $dQ$  is the added thermal heat to the system, and  $S$  is the entropy of the volume  $V$  which can be expressed in terms of the specific entropy  $s$  as  $S = Ms$ . Let us consider  $K$  now to be a variable instead of a constant, and work out Eq.(1.20). We use  $P = K\rho^\gamma = (\gamma - 1)\rho e = \rho kT/\mu$  (note:  $k \neq K$ ),  $e = K\rho^{\gamma-1}/(\gamma - 1)$ ,  $U = Me$  and  $M = \rho V$  and we obtain

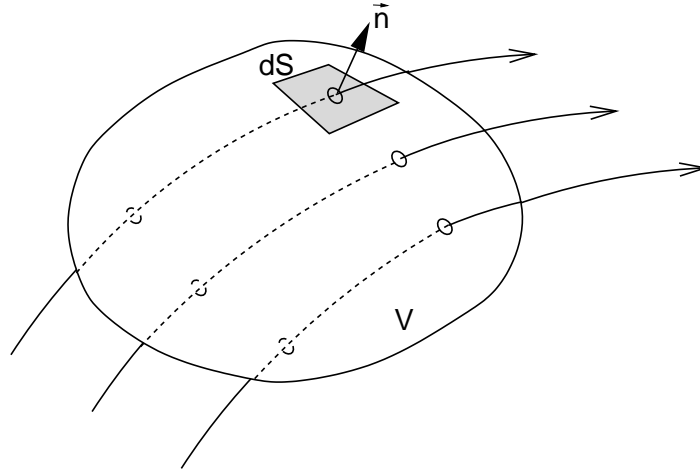
$$\frac{\mu}{k} ds = \frac{1}{\gamma - 1} d \lg e - d \lg \rho = \frac{1}{\gamma - 1} d \lg K \quad (1.21)$$

where we used  $d \lg e = (\gamma - 1) d \lg \rho + d \lg K$ . We can therefore write:

$$s = s_0 + \frac{k}{\mu(\gamma - 1)} \lg K \quad (1.22)$$

where  $s_0$  is an arbitrary offset constant. This shows that the number  $K$  in Eq.(1.12) is another way of writing the entropy of the gas. The entropy of a gas parcel is an important quantity. We will see below that normal gas flow does not change the entropy of a gas parcel. Shock fronts, however, will increase the entropy, and heating/cooling through radiative processes does this also. Heat conduction can also do this. But these are all special conditions. Under normal conditions a parcel of ideal gas keeps a constant entropy. This is a property that can be both useful and problematic for the development of numerical hydrodynamic schemes, as we shall see later.





**Figure 1.1.** Flow through a volume  $V$  with surface  $\partial V \equiv S$ . The diagram shows Gauss's theorem by which the change of the total content within the volume equals the integral of the flux through its surface.

## 1.4 The Euler equations: the equations of motion of the gas

The motion of a gas is governed entirely by conservation laws: the conservation of matter, the conservation of momentum and the conservation of energy. These conservation laws can be written in the form of partial differential equations (PDEs) as well as in the form of integral equations. Both forms will prove to be useful for the numerical methods outlined in this lecture.

### 1.4.1 Conservation of mass

Consider an arbitrary volume  $V$  in the space in which our gas flow takes place. Its surface we denote as  $\partial V \equiv S$  with the (outward pointing) normal unit vector at each location on the surface denoted as  $\vec{n}$  and differential surface element as  $dS$  (Fig. 1.1). The conservation of mass says that the variation of the mass in the volume must be entirely due to the in- or outflow of mass through  $\partial V$ :

$$\frac{\partial}{\partial t} \int \rho dV = - \int_{\partial V} \rho \vec{u} \cdot \vec{n} dS \quad (1.23)$$

Using Gauss's theorem, we can write this as:

$$\frac{\partial}{\partial t} \int \rho dV = - \int_V \nabla \cdot (\rho \vec{u}) dV \quad (1.24)$$

Since this has to be true for any volume  $V$  one chooses, we arrive at the following PDE:

$$\partial_t \rho + \nabla \cdot (\rho \vec{u}) = 0 \quad (1.25)$$

where  $\rho \vec{u}$  is the mass flux. This is also called the *continuity equation*.

Often it is useful to write such equations in *tensor form* (see appendix A):

$$\partial_t \rho + \partial_i (\rho u_i) = 0 \quad (1.26)$$

### 1.4.2 Conservation of momentum

The momentum density of the gas  $\rho \vec{u}$  (which is equal to the mass flux). The total momentum in a volume  $V$  is therefore the volume integral over  $\rho \vec{u}$ . In principle we can do the same trick as

above, with a volume integral over  $\rho \vec{u}$  and a surface integral over  $\rho \vec{u} \vec{u} \cdot \vec{n}$ . But here we must also take into account the forces that act on the surface by the gas surrounding the volume. At any position on the surface, the force acting by the gas outside the volume onto the gas inside the volume is  $-P\vec{n}$ . We can therefore write:

$$\frac{\partial}{\partial t} \int \rho \vec{u} dV = - \int_{\partial V} \rho \vec{u} \vec{u} \cdot \vec{n} dS - \int_{\partial V} P \vec{n} dS \quad (1.27)$$

Unfortunately, to use Gauss's theorem, one must have an inner product of something with  $\vec{n}$  at the surface, and  $P\vec{n}$  is not. To solve this problem we need to introduce the unit tensor  $\mathbf{I}$ , which is in index notation the Kronecker-delta (see Appendix A)<sup>1</sup>. With this we can write  $P\vec{n}$  as  $P\mathbf{I} \cdot \vec{n}$ . Using Gauss's theorem we get:

$$\frac{\partial}{\partial t} \int \rho \vec{u} dV = - \int_V \nabla \cdot (\rho \vec{u} \vec{u} + \mathbf{I}P) dV \quad (1.28)$$

and arrive thus at the PDE:

$$\partial_t(\rho \vec{u}) + \nabla \cdot (\rho \vec{u} \vec{u} + \mathbf{I}P) = 0 \quad (1.29)$$

which is the same as

$$\partial_t(\rho \vec{u}) + \nabla \cdot (\rho \vec{u} \vec{u}) + \nabla P = 0 \quad (1.30)$$

The quantity  $\rho \vec{u} \vec{u} + \mathbf{I}P$  is the *stress tensor* of the fluid.

Again here we can put this in index notation, which is particularly practical in the momentum equation as we are dealing with the stress tensor here:

$$\partial_t(\rho u_i) + \partial_k(\rho u_i u_k + \delta_{ik}P) = 0 \quad (1.31)$$

If we include a volume force on the gas, such as gravity, then we must add this as a source term. For gravity the force is the divergence of the gravitational potential  $\Phi$ , so we obtain:

$$\partial_t(\rho u_i) + \partial_k(\rho u_i u_k + \delta_{ik}P) = -\rho \partial_i \Phi \quad (1.32)$$

### 1.4.3 Conservation of energy

Energy exists in many forms. Here we concentrate on the two most basic ones: the thermal (internal) specific energy  $e$  and the kinetic specific energy  $e_{\text{kin}} = u^2/2$ . So the total energy is the volume integral of  $\rho(e + u^2/2)$ , and the advection of energy through the control volume surface is the surface integral of  $\rho(e + u^2/2)\vec{u} \cdot \vec{n}$ . But in addition to this we also have the work that the exterior acts on the control volume according to the first law of thermodynamics ( $dU = TdS - PdV$ ), which is the surface integral of  $P\vec{u} \cdot \vec{n}$ . So the conservation equation of energy is:

$$\frac{\partial}{\partial t} \int \rho \left( e + \frac{1}{2} u^2 \right) dV = - \int_{\partial V} \rho \left( e + \frac{1}{2} u^2 \right) \vec{u} \cdot \vec{n} dS - \int_{\partial V} P \vec{u} \cdot \vec{n} dS \quad (1.33)$$

Using Gauss theorem we then get:

$$\frac{\partial}{\partial t} \int \rho \left( e + \frac{1}{2} u^2 \right) dV + \int \nabla \cdot \left[ \left( \rho e + \frac{1}{2} \rho u^2 + P \right) \vec{u} \right] = 0 \quad (1.34)$$

---

<sup>1</sup>For those who are very familiar with tensor mathematics, this is in fact the contravariant metric.

Since this must be valid for all control volumes  $V$  we get the differential form of the energy conservation equation:

$$\partial_t(\rho e_{\text{tot}}) + \nabla \cdot [(\rho e_{\text{tot}} + P)\vec{u}] = 0 \quad (1.35)$$

or in index notation:

$$\partial_t(\rho e_{\text{tot}}) + \partial_k[(\rho e_{\text{tot}} + P)u_k] = 0 \quad (1.36)$$

This is the energy conservation equation. With the definition of  $h$  this can also be written as:

$$\partial_t(\rho e_{\text{tot}}) + \partial_k[\rho h_{\text{tot}} u_k] = 0 \quad (1.37)$$

## 1.5 Lagrange form of the hydrodynamics equations

The equations derived in Section 1.4 are the hydrodynamics equations in the form which we shall later numerically solve. But there exists another form of these equations which is a bit more intuitive, and also has some applications in numerical schemes. The idea is to follow a gas element along its path and see how it changes its direction of motion and how its density and pressure change along its way. This is called the *Lagrange form* of the equations. To derive this form of the equations we need to introduce the *comoving derivative*  $D_t$  as

$$D_t \equiv \partial_t + \vec{u} \cdot \vec{\nabla} \quad (1.38)$$

### 1.5.1 Continuity equation

With this definition the continuity equation then becomes:

$$D_t \rho = -\rho \vec{\nabla} \cdot \vec{u} \quad (1.39)$$

This form of the continuity equation has a physical meaning. It says that a gas parcel changes its density when the gas motion converges. In other words: when the gas motion is such that the parcel gets compressed. This compression is expressed by  $-\vec{\nabla} \cdot \vec{u}$ .

### 1.5.2 Momentum conservation equation

The momentum equation can be written as

$$\vec{\partial}_t \rho + \rho \partial_t \vec{u} + \vec{u} \vec{\nabla} \cdot [\rho \vec{u}] + \rho \vec{u} \cdot \vec{\nabla} \vec{u} + \vec{\nabla} P = 0 \quad (1.40)$$

We now use Eq. (1.25), i.e. the continuity equation, to remove two of the terms, and with the definition of the comoving derivative we obtain

$$D_t \vec{u} = -\frac{\vec{\nabla} P}{\rho} \quad (1.41)$$

This form of the equation of momentum conservation also has a physical interpretation. It says that a gas parcel will be accelerated due to a force, which is the pressure gradient. Any other body force, such as gravity, can be easily added as a term on the right-hand-side. This is the advantage of the Lagrangian form of the equations.

### 1.5.3 Energy conservation equation

Finally, the energy equation can be manipulated in a similar manner, also using Eq. (1.25) and the definition of the comoving derivative, and we obtain

$$D_t e_{\text{tot}} = -\frac{P}{\rho} \vec{\nabla} \cdot \vec{u} - \frac{1}{\rho} \vec{u} \cdot \nabla P \quad (1.42)$$

Now with  $e_{\text{tot}} = e + |\vec{u}|^2/2$  we can write (in index notation)

$$D_t e_{\text{tot}} = D_t e + u_i \partial_t u_i + u_k u_i \partial_k u_i = D_t e + u_i (\partial_t u_i + u_k \partial_k u_i) = D_t e + u_i D_t u_i \quad (1.43)$$

With the momentum equation (Eq. 1.41) we can replace the part in brackets, which yields  $-\vec{u} \cdot \vec{\nabla} P / \rho$ . Therefore we obtain for the energy equation:

$$D_t e = -\frac{P}{\rho} \vec{\nabla} \cdot \vec{u} \quad (1.44)$$

This also has physical meaning: the thermal energy of a gas parcel changes only as a result of adiabatic compression. Recall that the first law of thermodynamics, when expressed in  $\rho$  and  $e$ , reads

$$de = Tds - Pd\left(\frac{1}{\rho}\right) \quad (1.45)$$

where  $s$  is the specific entropy. If we replace  $d$  with  $D_t$ , and we use the continuity equation we obtain

$$D_t e = T D_t s - \frac{P}{\rho} \vec{\nabla} \cdot \vec{u} \quad (1.46)$$

So this means that another way of writing the Lagrange form of the energy equation is:

$$T D_t s = 0 \quad (1.47)$$

or to say: the entropy of a gas parcel does not change along its path of motion! The equations of hydrodynamics, at least the simplified forms we wrote down until now, conserve entropy. After its journey, a fluid parcel lies on the same adiabat as it started out.

We should, however, be careful with this statement. It is only true in regions of smooth flow. As we shall see below, gas flows tend to form shocks, which do not conserve entropy.

## 1.6 Properties of the hydrodynamic equations

### 1.6.1 Isentropic flow

In Section 1.4.3 we have derived the energy equation for a generic gas flow. But there are idealized situation where this equation becomes redundant. We have seen in Section 1.5.3 that a gas parcel does not change its entropy as it flows through some region. If we follow the motion of a gas parcel, then we do not need the energy equation: we can simply remember the initial entropy  $s$  (as represented, for instance, by the parameter  $K$  in the equation  $P = \rho^\gamma$ ) and at the end of its journey we can compute the temperature and pressure immediately using this same  $s$  (or  $K$ ) that we started out with. This is indeed true for Lagrangian systems, but if we use (as we shall do often below) the original form of the equations (Section 1.4), then we cannot keep track of which gas parcel is which. Therefore usually the energy equation is retained in numerical hydrodynamics.

However, if *all* of the gas in this region of interest has the same specific entropy, i.e. if the system is *isentropic*, then we do not need to keep track of parcels, since all gas lies on the same adiabat. In that case we can, like in the Lagrangian case, drop the energy equation and use the globally constant  $s$  or  $K$  to compute the pressure  $P$  from  $\rho$  whenever this is needed. This is the advantage of isentropic flow.

### 1.6.2 Isothermal flow

So far we have always assumed that the gas cannot radiate away its heat, nor get heated externally by radiation. Let us here consider the other extreme: the case of extreme heating/cooling: the situation where the gas is at any given time immediately cooled/heated to some ambient temperature. This can be true if the heating/cooling time scale of the gas is much shorter than the dynamic time scales, which is sometimes the case for astrophysical flows. In such a case we can also drop the energy equation, since we always know what  $e$  is: it is a global constant. In that case we can always say (for ideal gases):  $P = (\gamma - 1)e\rho$ , where  $(\gamma - 1)e$  is a global constant. We see that  $P$  is then linearly proportional to  $\rho$ .

Note that this linear relation between  $P$  and  $\rho$  is similar as taking  $\gamma = 1$  in the polytropic gas equation of state  $P = K\rho^\gamma$ . However, for  $\gamma = 1$  we have  $P = 0$ , as  $(\gamma - 1) = 0$  in the equation  $P = (\gamma - 1)e\rho$ . Some hydrodynamicists have in the past, however, modeled isothermal flows with  $\gamma = 1.01$  or thereabout, so as to simulate roughly a linear relation between pressure and density, without killing the equations.

### 1.6.3 Pressureless flow

Virtually all the properties and peculiarities of the motion of gases arises because of the effect that the pressure  $P$  has. If we were to assume that the pressure is small, what would then happen? First of all, we need to define what is small. The pressure appears as a second term next to  $\rho u_i u_k$  in the momentum equation. So if  $P \ll \rho |\vec{u}|^2$  we can reasonably safely assume  $P \simeq 0$ . The momentum conservation equation then becomes

$$\partial_t(\rho u_i) + \partial_k(\rho u_i u_k) = 0 \quad (1.48)$$

The continuity equation remains unchanged. The energy equation becomes irrelevant because  $e \simeq 0$ . So the continuity equation together with Eq. (1.48) form the equations for a pressureless gas. It is more precise to say: for a extremely supersonic gas as no gas is perfectly pressureless. We will later examine the properties of *Burger's equations* (Section 2.13) which are mathematically equivalent to these equations.

## 1.7 Sound waves

As we will show rigorously lateron, the dynamics of inviscid fluids, as described by the Euler equations, is in fact purely a matter of the propagation of signals. There are two kinds of signals: sound waves and fluid movements. One can describe the entire dynamics of the fluid in terms of these kinds of signals. For 1-D fluids one can make diagrams of these signals in the  $(x, t)$  plane, and the signals propagate along lines called *characteristics*. In this section we will derive the propagation of sound waves in a static and in a moving gas with constant (background-) density, and we will therewith demonstrate the concept of characteristics. Since this example is for infinitesimal perturbations on a constant density medium, it will only give an impression of the principle, but it will be very important for what follows. In fact, the entire applied mathematics

of numerical hydrodynamics is based on these concepts, and they therefore stand at the basis of this lecture.

### 1.7.1 Derivation of wave equation

Consider a constant density medium with a small space- and time-dependent perturbation. Let us restrict ourselves to 1-D. The density is then

$$\rho(x, t) = \rho_0 + \rho_1(x, t) \quad (1.49)$$

with  $\rho_1 \ll \rho_0$ . The fluid is assumed to move with a speed  $u(x, t)$  which is given by

$$u(x, t) = u_0 + u_1(x, t) \quad (1.50)$$

The equations of motion in 1-D are:

$$\partial_t \rho + \partial_x(\rho u) = 0 \quad (1.51)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + P) = 0 \quad (1.52)$$

With  $P = K\rho^\gamma$  the second equation becomes

$$\partial_t(\rho u) + \partial_x(\rho u^2) + \gamma \frac{P}{\rho} \partial_x \rho = 0 \quad (1.53)$$

With Eqs.(1.49,1.50) we get to first order in the perturbations:

$$\partial_t \rho_1 + \rho_0 \partial_x u_1 + u_0 \partial_x \rho_1 = 0 \quad (1.54)$$

$$u_0 \partial_t \rho_1 + \rho_0 \partial_t u_1 + 2\rho_0 u_0 \partial_x u_1 + u_0^2 \partial_x \rho_1 + \gamma \frac{P_0}{\rho_0} \partial_x \rho_1 = 0 \quad (1.55)$$

If we insert Eq.(1.54) into Eq. (1.55) then we get:

$$\partial_t \rho_1 + \rho_0 \partial_x u_1 + u_0 \partial_x \rho_1 = 0 \quad (1.56)$$

$$\rho_0 \partial_t u_1 + \rho_0 u_0 \partial_x u_1 + \gamma \frac{P_0}{\rho_0} \partial_x \rho_1 = 0 \quad (1.57)$$

If, just for now, we set  $u_0 = 0$  (static background density), then we obtain:

$$\partial_t \rho_1 + \rho_0 \partial_x u_1 = 0 \quad (1.58)$$

$$\rho_0 \partial_t u_1 + \gamma \frac{P_0}{\rho_0} \partial_x \rho_1 = 0 \quad (1.59)$$

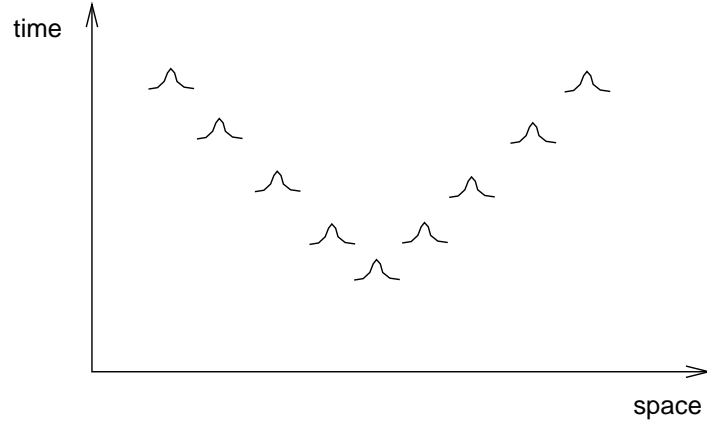
These two equations can be combined to

$$\partial_t^2 \rho_1 - \gamma \frac{P_0}{\rho_0} \partial_x^2 \rho_1 = 0 \quad (1.60)$$

This is a wave equation with waves travelling at velocity  $+\sqrt{\gamma P_0/\rho_0}$  and  $-\sqrt{\gamma P_0/\rho_0}$ . Hence the definition of  $C_s^2 = \gamma P/\rho$  in Section 1.2.

Now let us go back to the full equations with  $u_0 \neq 0$ . Let us define a *comoving derivative*  $D_t$  of some function  $q(x, t)$  as:

$$D_t q(x, t) \equiv \partial_t q(x, t) + u_0 \partial_x q(x, t) \quad (1.61)$$



**Figure 1.2.** The movement of the signal of a perturbation imposed at some point  $x_0$  and time  $t_0$  in a 1-D spacetime, for the case of zero background velocity, i.e.  $u_0 = 0$ .

With this definition we can write Eqs. (1.56,1.56) as:

$$D_t \rho_1 + \rho_0 \partial_x u_1 = 0 \quad (1.62)$$

$$\rho_0 D_t u_1 + \gamma \frac{P_0}{\rho_0} \partial_x \rho_1 = 0 \quad (1.63)$$

As before, these two equations can be combined to

$$D_t^2 \rho_1 - \gamma \frac{P_0}{\rho_0} \partial_x^2 \rho_1 = 0 \quad (1.64)$$

This can be interpreted as a wave equation *in the comoving frame of the background medium*. So the two waves propagate with velocities  $u_0 + \sqrt{\gamma P_0 / \rho_0}$  and  $u_0 - \sqrt{\gamma P_0 / \rho_0}$ .

The solution is:

$$\rho_1(x, t) = A e^{ikx - i\omega t} \quad (1.65)$$

$$u_1(x, t) = \frac{A}{\rho_0} \sqrt{\gamma \frac{P_0}{\rho_0}} e^{ikx - i\omega t} \quad (1.66)$$

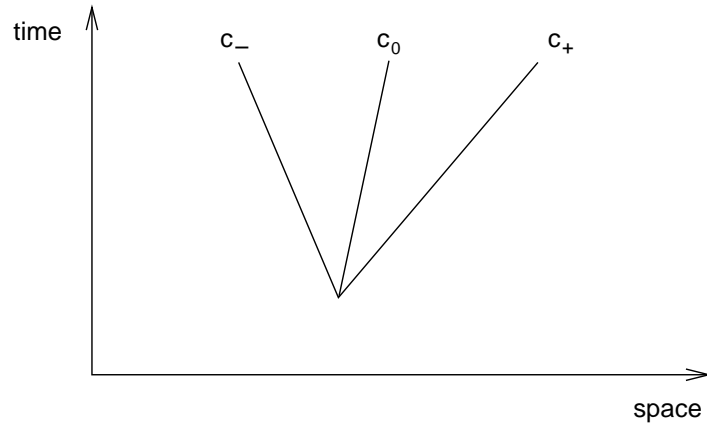
with the dispersion relation

$$\frac{\omega}{k} = u_0 \pm \sqrt{\gamma \frac{P_0}{\rho_0}} \quad (1.67)$$

### 1.7.2 Characteristics

The fact that linear perturbations move as waves can be depicted in the  $(x, t)$  plane. Consider a sudden point-like perturbation at some place  $x_0$  at time  $t_0$ . For  $t < t_0$  there were no perturbations, and at time  $t_0$  this point-like perturbation is initiated. What will happen? One wave will propagate to the left at velocity  $u_0 - \sqrt{\gamma P_0 / \rho_0}$ , and one wave will propagate to the right at velocity  $u_0 + \sqrt{\gamma P_0 / \rho_0}$ . Of course, if  $u_0 > \sqrt{\gamma P_0 / \rho_0}$  then the leftward moving wave is also moving to the right as it is dragged along. In this case the background velocity  $u_0$  is *supersonic*. In any case, we can plot the exact location of the perturbation at any time  $t > t_0$  in the  $(x, t)$  diagram (Fig. 1.2).

One sees that this forms a wedge and, dependent on  $u_0$ , this wedge is slanted or not. In case of supersonic background flow the wedge tops over. The lines shown in these diagrams



**Figure 1.3.** The three characteristics  $c_-$ ,  $c_0$ ,  $c_+$  of the hydrodynamics equations in 1-D, belonging to the characteristic velocities  $\lambda_{-1}$ ,  $\lambda_0$ ,  $\lambda_{+1}$ .

are called *characteristics*. They do not necessarily have to be defined as belonging to the origin  $(x_0, t_0)$ . Any line in the  $(x, t)$  diagram following the possible trajectory of a signal is called a characteristic. In this case we have plotted the sound-characteristics.

There is also another set of characteristics. This is not seen in the above analysis. However, it can be shown if we give the fluid a ‘color’. Suppose that at  $t = t_0$  we dye all gas left of  $x_0$  blue and all gas right of  $x_0$  as red. We now introduce a special function  $\varphi(x, t)$  which gives the color. If it is 0, it means blue, if 1 it means red. The equation of this *passive tracer* is:

$$\partial_t \varphi + u \partial_x \varphi = 0 \quad (1.68)$$

We now insert  $u = u_0 + u_1$ :

$$\partial_t \varphi + u_0 \partial_x \varphi + u_1 \partial_x \varphi = 0 \quad (1.69)$$

We immediately see that the last term is negligible compared to the first two, so we obtain approximately:

$$\partial_t \varphi + u_0 \partial_x \varphi = 0 \quad (1.70)$$

This signal evidently propagates with velocity  $u_0$ . This gives the third set of characteristics, describing the movement of the fluid itself. This shows that in total we have, for this 1-D example, three sets of characteristics, moving at speeds:

$$\lambda_{-1} = u_0 - \sqrt{\gamma P_0 / \rho_0} \quad (1.71)$$

$$\lambda_0 = u_0 \quad (1.72)$$

$$\lambda_{+1} = u_0 + \sqrt{\gamma P_0 / \rho_0} \quad (1.73)$$

This example shows that, at least for linear perturbations of an otherwise steady constant-density background, the hydrodynamics equations amount to the propagation of signals at three different speeds. Two signals are sound signals, while a third signal is the movement of mass (Fig. 1.3). This third signal may sound a bit as a cheat, since it is simply the passive co-movement with the background fluid, and has no dynamical character of its own as the sound waves do. However, in the non-linear evolution of hydrodynamic flows this third characteristic plays an essential role and is no longer a passive tracer. We will see this later, when we view the hydrodynamics equations as a hyperbolic set of equations.



## 1.8 Viscosity

### 1.8.1 Bulk and shear viscosity

So far we have assumed that the only force on a fluid parcel is the force due to a pressure gradient. There are also other forces that can be involved in the exchange of momentum between fluid parcels. Here we focus on the force of viscosity. A force between two adjacent fluid parcels A and B can be seen as a flux of momentum from fluid parcel A to B, or the reverse flux from B to A. Let us recall Eq.(1.31) and write it in a special way:

$$\partial_t(\rho u_i) + \partial_k(\rho u_i u_k + \delta_{ik} P) = 0 \quad (1.74)$$

$$= \partial_t(\rho u_i) + \partial_k(\rho u_i u_k + \Pi_{ki}) = 0 \quad (1.75)$$

$$= \partial_t(\rho u_i) + \partial_k T_{ki} = 0 \quad (1.76)$$

where  $T_{ki}$  is the momentum flux density tensor and  $\Pi_{ki}$  the pressure tensor<sup>2</sup>. The  $T_{ki}$  tensor is the flux of  $i$  momentum in  $k$  direction. It is the spatial part of the stress-energy tensor familiar from relativity theory ( $T^{\mu\nu} = \rho u^\mu u^\nu + g^{\mu\nu} P$ , for those who are familiar with it). Both  $T_{ki}$  and  $\Pi_{ki}$  are always a symmetric tensor, i.e. it could also describe the flux of  $k$  momentum in  $i$  direction. For the particular case at hand, where  $\Pi_{ki}$  represents merely the pressure force, this tensor has two special properties:

1. Momentum is exchanged only in the direction where the transported momentum points. In other words:  $\Pi_{ki}$  is a diagonal tensor. X-momentum is transported in X-direction, Y-momentum in Y-direction etc.
2. The forces are always perfectly ‘elastic’ in the sense that the force does not depend on the speed of compression or decompression of a fluid element. These (de-)compressions are therefore reversible processes.

If we have viscosity, then these properties do no longer hold. Viscosity is a force that acts whenever there are velocity gradients in the flow, or to be precise, whenever there is shear flow. In the presence of viscosity,  $\Pi_{ki}$  reads:

$$\Pi_{ki} = \delta_{ki} P - \sigma'_{ki} \quad (1.77)$$

where  $\sigma'_{ki}$  is the viscous stress tensor:

$$\sigma'_{ki} = \eta \left( \partial_i u_k + \partial_k u_i - \frac{2}{3} \delta_{ki} \partial_l u_l \right) + \zeta \delta_{ki} \partial_l u_l \quad (1.78)$$

where  $\eta$  is the coefficient of shear viscosity and  $\zeta$  is the coefficient of bulk viscosity. These are also called *the* coefficient of viscosity and the *second* viscosity respectively. The *kinematic viscosity* coefficient  $\nu$  is defined as

$$\nu = \eta / \rho \quad (1.79)$$

The equations of motion are then:

$$\partial_t(\rho u_i) + \partial_k(\rho u_i u_k + \delta_{ik} P - \sigma'_{ik}) = 0 \quad (1.80)$$

or, in Lagrange form:

$$\rho D_t u_i = -\partial_i P + \partial_k \sigma'_{ki} \quad (1.81)$$

---

<sup>2</sup>Here we deviate from the Landau & Lifshitz notation.

This equation is called the *Navier-Stokes equation*.

To understand what these equations mean we can do two experiments, which we do in 3-D Cartesian coordinates  $(x, y, z)$ :

1. Take a velocity field  $u_i = (y, 0, 0)$ . This is a typical shear flow without compression. We obtain:

$$\sigma'_{ki} = \begin{pmatrix} 0 & \eta & 0 \\ \eta & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}_{ki} \quad (1.82)$$

When inserting this into Eq. (1.81) we see that this transports negative  $x$ -momentum in positive  $y$ -direction. This is what we are familiar with as friction.

2. Take a velocity field  $u_i = (-x, 0, 0)$ . This is a typical compressive flow without shear. We obtain:

$$\sigma'_{ki} = \begin{pmatrix} -\zeta & 0 & 0 \\ 0 & 0 & -\zeta \\ 0 & 0 & -\zeta \end{pmatrix}_{ki} \quad (1.83)$$

When inserting this into Eq. (1.81) we see that this transports positive  $x$ -momentum in positive  $x$ -direction. But if we would flip the sign of  $u_i$  (i.e. have decompression), then we have the force also flip sign: we it will transports negative  $x$ -momentum in positive  $x$ -direction. In other words: bulk viscosity acts against both compression and decompression. Or we can say: whatever one does (compression or decompression), it always costs energy. This is what bulk viscosity does.

- **Exercise:** Show that the viscous force vanishes for solid-body rotation, and explain why this must be so.
- **Exercise:** Assume a constant density flow with a velocity field of  $u_i = (\sin(y), 0)$  at time  $t = 0$ . Compute the  $\partial_t u_i$  at time  $t = 0$ .

### 1.8.2 How important is viscosity?

Some flows are very little affected by viscosity, for instance the flow of air around obstacles, or the flow of water through a river. Other flows are very much affected by viscosity, for instance the flow of sirup or Nutella that one smears on bread, or the flow of a glacier. A quantity to tell the difference is the *Reynolds number*:

$$Re = \frac{\rho u L}{\eta} = \frac{u L}{\nu} = \frac{\text{inertial forces}}{\text{viscous forces}} \quad (1.84)$$

where  $u$  is the typical velocity in the fluid flow and  $L$  is a typical spatial scale corresponding to the size of the fluid patterns we are interested in. One sees that the Reynolds number is not a very strictly defined quantity, because  $u$  may vary in the flow and  $L$  is just a ‘typical’ length scale. But it does an indication of the importance of viscosity in the flow:

- $Re \gg 1$  → Very inviscid flows. These flows can often easily become turbulent.
- $Re \ll 1$  → Very viscous flows. These flows tend to be very laminar.

We will see in later chapters that:

1. High  $Re$  flows are difficult to model numerically because the numerical algorithm artificially lowers  $Re$  ('numerical viscosity')
2. High  $Re$  flows often become turbulent (though not always). The flow patterns in turbulent flows have a range of scales  $L$ , going from the largest turbulent eddies, down to very small scales. This is called a *turbulent cascade*. The smallest scale of turbulence is the scale  $L$  where  $Re = uL/\nu$  starts to exceed unity. Therefore, the Reynolds number of turbulent flows is always a matter of definition which  $L$  to use.
3. Turbulence can act as an 'effective viscosity' in itself.
4. Even laminar flows of high  $Re$  are sometimes affected by viscosity through *boundary layers*: thin layers between a high- $Re$  flow and some solid surface with a thickness such that in this layer  $Re \gtrsim 1$ .

## 1.9 Shock waves

Under normal conditions the equations of hydrodynamics ensure that a fluid parcel keeps its entropy constant, as we have seen in Section 1.5.3. However, even in the case of inviscid fluids ( $Re \rightarrow \infty$ ) there are conditions under which this is no longer true: when a *shock wave* occurs. The formation of a shock wave can be understood in various ways. One way is to consider a very strong sound wave. At the top of the wave the temperature is higher than at the valley of the wave. Since the sound speed is proportional to the square root of the temperature, it is to be expected that the top of the wave moves faster than the valley. The wave therefore has the tendency to steepen like waves on the beach. This will necessarily lead to the formation of a discontinuity called a shock wave. One sees that formally even normal sound waves eventually develop shock waves. These will, however, be very very weak ones, and they behave very much like a normal sound wave. In other words: a very weak shock wave is like a sound wave. The shock wave emerging from an explosion will, as it spherically expands, weaken in strength and eventually be a strong jump-like sound wave. Also the shock wave emerging from a supersonic airplane eventually is heard by people on the ground as a 'sonic boom', i.e. a sound wave.

We will deal with shock waves in chapter ?? in much more detail than here. But some of the basics are necessary at this point. The nature of a shock wave is given by its *jump conditions*. It gives which state the gas is in after going through a shock wave.

Let us assume that we move along with the shock, so that in our laboratory frame the shock is standing still and the gas (before and after the shock) is moving. We assume a steady state in this frame and we define  $\rho_1$  and  $\rho_2$  (and idemdito for all other variables) such that the gas moves from region 1 to region 2. This means that mass flux, momentum flux and energy flux through the shock must be constant:

$$\rho_2 u_2 = \rho_1 u_1 \quad (1.85)$$

$$\rho_2 u_2^2 + P_2 = \rho_1 u_1^2 + P_1 \quad (1.86)$$

$$\rho_2 h_{\text{tot}} u_2 = \rho_1 h_{\text{tot}} u_1 \quad (1.87)$$

Now let us define the specific volumes  $V_2 = 1/\rho_2$  and  $V_1 = 1/\rho_1$  and the mass flux  $j = \rho_2 u_2 = \rho_1 u_1$ . We then get

$$u_2 = jV_2 \quad u_1 = jV_1 \quad (1.88)$$

and

$$P_2 + j^2 V_2 = P_1 + j^2 V_1 \quad (1.89)$$

which yields

$$j^2 = (P_1 - P_2)/(V_2 - V_1) \quad (1.90)$$

In principle this equation allows for two solutions, one in which the post-shock medium has a higher pressure than the pre-shock medium ( $P_2 > P_1$ ), and one in which the post-shock medium has a lower pressure than the pre-shock medium ( $P_2 < P_1$ ). The latter will turn out to be an unphysical solution. In practise such a solution will quickly smear out into a *rarefaction wave*, but this topic will be discussed in chapter ???. We focus here on the case  $P_2 > P_1$  which is what we call a shock wave.

We now substitute  $u_2 - u_1 = j(V_2 - V_1)$  into Eq.(1.90) and obtain

$$u_1 - u_2 = \sqrt{(P_2 - P_1)(V_1 - V_2)} \quad (1.91)$$

where we took only the positive root because that is the physical one. Now let us turn to the energy equation

$$\rho_2(h_2 + \frac{1}{2}u_2^2)u_2 = \rho_1(h_1 + \frac{1}{2}u_1^2)u_1 \quad (1.92)$$

Because of mass conservation this turns into

$$h_2 + \frac{1}{2}u_2^2 = h_1 + \frac{1}{2}u_1^2 \quad (1.93)$$

and further into

$$h_2 - h_1 = \frac{1}{2}j^2(V_1^2 - V_2^2) \quad (1.94)$$

and with Eq. (1.90) into

$$h_2 - h_1 = \frac{1}{2}(V_1 + V_2)(P_2 - P_1) \quad (1.95)$$

Now replace  $h = e + PV$  and we obtain

$$e_2 - e_1 = \frac{1}{2}(V_1 - V_2)(P_2 + P_1) \quad (1.96)$$

This is the relation between the state of the gas before and after the shock. It is called the *Rankine-Hugoniot shock adiabetic*.

For polytropic gases we can derive the following expressions (see e.g. Landau & Lifshitz):

$$\frac{\rho_2}{\rho_1} = \frac{u_1}{u_2} = \frac{(\gamma + 1)M_1^2}{(\gamma - 1)M_1^2 + 2} \quad (1.97)$$

$$\frac{P_2}{P_1} = \frac{2\gamma M_1^2}{\gamma + 1} - \frac{\gamma - 1}{\gamma + 1} \quad (1.98)$$

$$M_2^2 = \frac{2 + (\gamma - 1)M_1^2}{2\gamma M_1^2 - (\gamma - 1)} \quad (1.99)$$

where  $M_1 \equiv |u_1|/C_s$  and  $M_2 \equiv |u_2|/C_s$  are the *Mach numbers* on the pre- and post-shock regions.

Some properties of shocks:

- For mono-atomic gas ( $\gamma = 5/3$ ) the maximum compression of the gas (for  $M_1 \rightarrow \infty$ ) is  $\rho_2/\rho_1 \rightarrow 4$ , and for diatomic gas ( $\gamma = 7/5$ ) it is  $\rho_2/\rho_1 \rightarrow 6$ .

- The Mach number before the shock is always  $M_1 > 1$  and the Mach number behind the shock is always  $M_2 < 1$ .
- When a gas parcel goes through a shock, its entropy is increased. For the integral form of the hydrodynamics equations without viscosity, shocks are the only regions in space where gas parcels increase their entropy.



# Chapter 2

## Hyperbolic equations

As we have seen, and will further argue below, the hydrodynamics equations are nothing more than signal-propagation equations. Equations of this kind are called *hyperbolic equations*. The equations of hydrodynamics are only a member of the more general class of hyperbolic equations, and there are many more examples of hyperbolic equations than just the equations of hydrodynamics. But of course we shall focus mostly on the application to hydrodynamics. In this chapter we shall study the mathematical properties of hyperbolic equations, and analytic methods how to solve them for simple linear problems. This background knowledge is not only important for understanding the nature of hydrodynamic flows, it also lies at the basis of numerical hydrodynamics algorithms. Hence we go into quite some detail here.

### 2.1 The simplest form of a hyperbolic equation: advection

Consider the following equation:

$$\partial_t q + u \partial_x q = 0 \quad (2.1)$$

where  $q = q(x, t)$  is a function of one spatial dimension and time, and  $u$  is a velocity that is constant in space and time. This is called an *advection equation*, as it describes the time-dependent shifting of the function  $q(x)$  along  $x$  with a velocity  $u$  (Fig. 2.1-left). The solution at any time  $t > t_0$  can be described as a function of the state at time  $t_0$ :

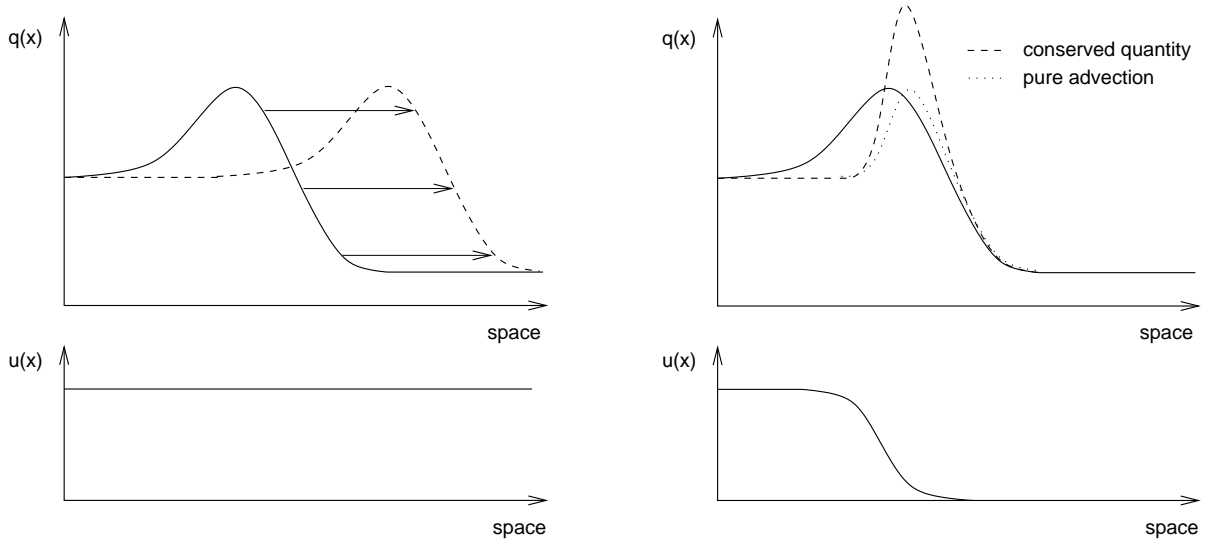
$$q(x, t) = q(x - ut, 0) \quad (2.2)$$

This is a so-called *initial value problem* in which the state at any time  $t > t_0$  can be uniquely found when the state at time  $t = t_0$  is fully given. The characteristics of this problem are straight lines:

$$x_{\text{char}}(t) = x_{\text{char}}^{(0)} + ut \quad (2.3)$$

This is a family of lines in the  $(x, t)$  plane, each of which is labeled by its own unique value of  $x_{\text{char}}^{(0)}$ .

In this example the initial value problem is quite trivial, yet, as we will see below, this problem stands at the basis of numerical methods of hydrodynamics and is numerically surprisingly challenging to solve!



**Figure 2.1.** Advection of a function  $q(x, t)$  with constant velocity  $u$  (left) and space-varying velocity  $u(x)$  (right). The space-varying velocity problem comes in two versions: the conserved form (dashed) and the non-conserved simple advection form (dotted).

## 2.2 Advection with space-dependent velocity

Consider now that  $u$  is a function of space:  $u = u(x)$ . We get:

$$\partial_t q + u(x) \partial_x q = 0 \quad (2.4)$$

This is still an advection problem, but the velocity is not constant, and therefore the solution is somewhat more complex. Define the following variable:

$$\xi = \int_{x_0}^x \frac{dx'}{u(x')} \quad (2.5)$$

for some arbitrary  $x_0$ . Now the solution is given as:

$$q(x, t) = q(x(\xi), t) = q(x(\xi - t), 0) \quad (2.6)$$

where  $x(\xi)$  is the value of  $x$  belonging to the value of  $\xi$ . We see that in regions of low  $u(x)$  the function will be squeezed in  $x$ -direction while in regions of high  $u(x)$  the function will be stretched. Fig. 2.1-right, dotted line, shows the squeezing effect for non-constant advection velocity.

The characteristics are:

$$x_{\text{char}}(t) = x(\xi_{\text{char}}^{(0)} + t) \quad (2.7)$$

In this case the labeling is done with  $\xi_{\text{char}}^{(0)}$  and, by definition, the curves follow the flow in the  $(x, t)$  plane.

If we define the comoving derivative  $D_t$  as  $D_t = \partial_t + u(x) \partial_x$ , Eq.(2.4) translates into:

$$D_t q(x, t) = 0 \quad (2.8)$$

This simply tells that along each flow line (characteristic) the function  $q$  remains constant, i.e. the *comoving* derivative is zero.



### 2.3 Advection of a conserved quantity with space-dependent velocity

Now consider yet another version of the advection equation:

$$\partial_t q + \partial_x [qu(x)] = 0 \quad (2.9)$$

This is very similar to Eq. (2.4), but this time with  $u(x)$  inside the  $\partial_x$  operator. This is a *conservation equation*. It has the following property of conservation: if  $u(x)$  is zero at  $x_0$  and  $x_1 > x_0$ , then the integral  $\int_{x_0}^{x_1} q dx$  is constant in time.

This equation can be written in the previous form, with  $u(x)$  outside the operator:

$$\partial_t q + u(x) \partial_x q = -q \partial_x u(x) \quad (2.10)$$

This is an advection equation of the type of Eq. (2.4), but this time with a non-zero right-hand-side. This can be regarded as some kind of source term in the equation (even though strictly speaking it is not, as the system is conserved). The characteristics are the same as for the case without this term, but in this case the value of  $q$  is no longer constant along these characteristics:

$$D_t q(x, t) = -q(x, t) \partial_x u(x) \quad (2.11)$$

When the characteristics converge,  $q$  goes up, and when they diverge,  $q$  goes down. This is depicted in Fig. 2.1-right, dashed line. This is precisely what one expects to happen if  $q$  represents some kind of density.

### 2.4 Flux-form of conservation equation and the Jacobian

Let us stick with the equation (2.9), but write it in the following form:

$$\partial_t q + \partial_x f(q, x) = 0 \quad (2.12)$$

where  $f(q, x)$  is called the flux of the quantity  $q$ . The flux  $f$  is entirely determined for any given value of the function  $q$  and position  $x$ . We can then use the chain rule to arrive at

$$\partial_t q + \left. \frac{\partial f}{\partial q} \right|_{x=\text{const}} \partial_x q = - \left. \frac{\partial f(q, x)}{\partial x} \right|_{q=\text{const}} \quad (2.13)$$

In our case we have:

$$\left. \frac{\partial f}{\partial q} \right|_{x=\text{const}} = u(x) \quad \text{and} \quad \left. \frac{\partial f}{\partial x} \right|_{q=\text{const}} = q \partial_x u(x) \quad (2.14)$$

In this way we also arrive at Eq. (2.10). It is this route to Eq. (2.10) that we will take in the cases of the more complex non-linear systems of equations to follow. The  $\partial f / \partial q$  is the Jacobian of the system.

### 2.5 Coupled set of equations

All of the above can also be formulated for coupled sets of PDEs. Instead of a state scalar  $q$  we defined a state vector  $Q = (q_1, \dots, q_m)$ . The advection equation of the non-conservative type is then

$$\partial_t Q + A \partial_x Q = 0 \quad (2.15)$$

where  $A$  is an  $m \times m$  matrix. The advection equation of conservative type is

$$\partial_t Q + \partial_x(AQ) = 0 \quad (2.16)$$

The more general conservation equation is:

$$\partial_t Q + \partial_x F = 0 \quad (2.17)$$

where  $F = F(Q, x)$  is the flux. Like in the scalar case we have

$$\partial_t Q + \frac{\partial F}{\partial Q} \Big|_{x=\text{const}} \partial_x Q = - \frac{\partial F(q, x)}{\partial x} \Big|_{Q=\text{const}} \quad (2.18)$$

where the  $J \equiv \partial F / \partial Q$  is the Jacobian of the system, which is an  $m \times m$  matrix. If  $F$  is a linear function of  $Q$ , then one can write  $F = AQ$  where  $A$  is then the Jacobian, and we arrive at Eq. (2.16). If  $A$  is then also independent of  $x$ , then we arrive at Eq. (2.16).

## 2.6 The wave equation in vector notation: eigenvalues and eigenvectors

In Section 1.7.1 we derived the wave equation for perturbations in an otherwise steady constant-density constant-velocity background medium. Let us now define:

$$Q \equiv \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_0 u_1 \end{pmatrix} \quad (2.19)$$

We can then write Eqs. (1.56,1.57) as:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} u_0 & 1 \\ C_s^2 & u_0 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (2.20)$$

We can also write this in flux conservative form:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \partial_x \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = 0 \quad (2.21)$$

with the flux  $F = (f_1, f_2)$  given as:

$$F \equiv \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} u_0 & 1 \\ C_s^2 & u_0 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} u_0 q_1 + q_2 \\ C_s^2 q_1 + u_0 q_2 \end{pmatrix} \quad (2.22)$$

or in other words, the Jacobian matrix is:

$$J_{ik} = \frac{\partial f_i}{\partial q_k} = \begin{pmatrix} u_0 & 1 \\ C_s^2 & u_0 \end{pmatrix} \quad (2.23)$$

One of the advantages of writing the wave equation, and later on many other equations, in the form of a matrix equation like Eq. (2.20), is that we can use some aspects of linear algebra to solve the equations in an elegant way, which will later turn out to have a very powerful application in numerical methods of hydrodynamics.

let us look again at the Jacobian matrix of the wave equation, Eq. (2.23). This matrix has the following eigenvectors and eigenvalues:

$$e_{-1} = \begin{pmatrix} 1 \\ -C_s \end{pmatrix} \quad \text{with} \quad \lambda_{-1} = u_0 - C_s \quad (2.24)$$

$$e_{+1} = \begin{pmatrix} 1 \\ +C_s \end{pmatrix} \quad \text{with} \quad \lambda_{+1} = u_0 + C_s \quad (2.25)$$

If we include the passive tracer of Eq. (1.7.2) we have:

$$Q \equiv \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_0 u_1 \\ \varphi \end{pmatrix} \quad (2.26)$$

and the matrix form of the equation of motion becomes:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \begin{pmatrix} u_0 & 1 & 0 \\ C_s^2 & u_0 & 0 \\ 0 & 0 & u_0 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = 0 \quad (2.27)$$

This has the following set of eigenvectors and eigenvalues:

$$e_{-1} = \begin{pmatrix} 1 \\ -C_s \\ 0 \end{pmatrix} \quad \text{with} \quad \lambda_{-1} = u_0 - C_s \quad (2.28)$$

$$e_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \text{with} \quad \lambda_0 = u_0 \quad (2.29)$$

$$e_{+1} = \begin{pmatrix} 1 \\ +C_s \\ 0 \end{pmatrix} \quad \text{with} \quad \lambda_{+1} = u_0 + C_s \quad (2.30)$$

We can now decompose any vector  $q$  into eigenvectors:

$$Q = \tilde{q}_{-1} e_{-1} + \tilde{q}_0 e_0 + \tilde{q}_{+1} e_{+1} \quad (2.31)$$

Then the equation of motion becomes:

$$\partial_t \begin{pmatrix} \tilde{q}_{-1} \\ \tilde{q}_0 \\ \tilde{q}_{+1} \end{pmatrix} + \begin{pmatrix} u_0 - C_s & 0 & 0 \\ 0 & u_0 & 0 \\ 0 & 0 & u_0 + C_s \end{pmatrix} \partial_x \begin{pmatrix} \tilde{q}_{-1} \\ \tilde{q}_0 \\ \tilde{q}_{+1} \end{pmatrix} = 0 \quad (2.32)$$

The matrix is here diagonal. This has the advantage that we have now decomposed the problem into three *scalar advection equations*:

$$\partial_t \tilde{q}_i + \lambda_i \partial_x \tilde{q}_i = 0 \quad (2.33)$$

for any  $i = -1, 0, +1$ . For these equations we know the solution: they are simply shifts of an initial function (see Sections 2.1 and 2.2). In the case at hand here we are fortunate that the  $C_s$  and  $u_0$  are constant, so we get:

$$\tilde{q}_i(x, t) = \tilde{q}_i(x - \lambda_i t, 0) \quad (2.34)$$

for any  $i = -1, 0, +1$ . The vector-notation and the decomposition into eigenvectors and eigenvalues stands at the basis of much of the theory on numerical algorithms to follow.

The system of equations described here is a *hyperbolic set of equations*, which is another way of saying that they describe the motion of signals. We will define hyperbolicity more rigorously below.

## 2.7 Hyperbolic sets of equations: the linear case with constant Jacobian

Let us consider a set of linear equations that can be written in the form:

$$\partial_t Q + A \partial_x Q = 0 \quad (2.35)$$

where  $Q$  is a vector of  $m$  components and  $A$  is an  $m \times m$  matrix.

This system is called *hyperbolic* if the matrix  $A$  is diagonalizable with real eigenvalues. The matrix is diagonalizable if there exists a complete set of eigenvectors  $e_i$ , i.e. if any vector can be written as:

$$Q = \sum_{i=1}^m \tilde{q}_i e_i \quad (2.36)$$

In this case one can write

$$AQ = \sum_{i=1}^m \lambda_i \tilde{q}_i e_i \quad (2.37)$$

We can define a matrix in which each column is one of the eigenvectors:

$$R = (e_1, \dots, e_m) \quad (2.38)$$

Then we can transform Eq. (2.35) into:

$$R^{-1} \partial_t Q + R^{-1} A R R^{-1} \partial_x Q = 0 \quad (2.39)$$

which with  $\tilde{Q} = R^{-1} Q$  then becomes:

$$\partial_t \tilde{Q} + \tilde{A} \partial_x \tilde{Q} = 0 \quad (2.40)$$

where  $\tilde{A} = \text{diag}(\lambda_1, \dots, \lambda_m)$ . Not all  $\lambda_i$  must be different from each other.

This system of equations has in principle  $m$  sets of characteristics. But any set of characteristics that has the same characteristic velocity as another set is usually called the same set of characteristics. So in the case of 5 eigenvalues, three of which are identical, one typically says that there are three sets of characteristics.

## 2.8 Boundary conditions (I)

So far we have always assumed that space is infinite. In real-life applications the domain of interest is always bound. In some cases these boundaries are real (like a wall or a piston) but in other cases they have to be somewhat artificially imposed because computing power is not as infinite as space is and one is limited to a finite volume. It is therefore important to know how spatial boundary conditions are set. In the numerical chapters we will go into this in far more detail than here, often going into more practical matters. Here we are concerned with the mathematical issue.

We look at the general hyperbolic equation

$$\partial_t Q(x, t) + A \partial_x Q(x, t) = 0 \quad (2.41)$$

where  $Q$  is a vector with  $m$  components and  $A$  is an  $m \times m$  matrix. Let us define as our domain of interest the space between  $x_0$  and  $x_1 > x_0$ . At these boundaries we wish to impose the necessary and sufficient boundary conditions. Since this is a coupled set of  $m$  linear equations we need  $m$  boundary conditions. In systems of hyperbolic equations one usually sets *Dirichlet boundary conditions*, i.e. one specifies the values of the components of  $Q$  at these boundaries. The best way to do this is to decompose into the eigenvalues of  $A$ :

$$\partial_t \tilde{q}_i(x, t) + \lambda_i \partial_x \tilde{q}_i(x, t) = 0 \quad (2.42)$$

For each  $i \in [1, \dots, m]$  one can then set:

$$\tilde{q}_i(x = x_{0|1}, t) = \tilde{q}_i^{(bc)}(t) \quad (2.43)$$

The  $x_{0|1}$  means that one can *either* specify that condition for  $\tilde{q}_i$  at  $x = x_0$  *or* at  $x = x_1$ . For instance, for  $m = 2$  one could specify one condition at  $x_0$  for  $\tilde{q}_1$  and one condition at  $x_1$  for  $\tilde{q}_2$ . In total one can impose  $m$  conditions, divided over the left and right boundaries. Note, by the way, that the boundary conditions may be time-dependent!

The question now is: for some  $\tilde{q}_i$ , at which boundary should we impose the Dirichlet boundary condition? Suppose  $\lambda_i > 0$ . It is then clear that signals propagate from left to right. Therefore it is clear that the boundary condition, for  $\tilde{q}_i$ , must be set at the left boundary, at  $x = x_0$ . The condition specifies the *inflow* of the signal into the system through the left boundary. If one were to specify it at the other boundary (which in a numerical simulation one could simply try to do, and see what happens), then the installed boundary value is immediately advected off the domain again and will not influence the solution within the domain at all. In contrast, setting  $\tilde{q}_i$  at the left boundary means that this value is advected into the domain and affects the solution there.

In general one can say that at boundary  $x = x_0$  one specifies  $\tilde{q}_i(x = x_0, t) = \tilde{q}_i^{(bc)}(t)$  for all  $i$  for which  $\lambda_i > 0$  and at boundary  $x = x_1$  one specifies  $\tilde{q}_i(x = x_1, t) = \tilde{q}_i^{(bc)}(t)$  for all  $i$  for which  $\lambda_i < 0$ .

Another possible kind of boundary conditions are *periodic boundary conditions*. Here we set, at every time  $t$ , for  $\lambda_i > 0$ :

$$\tilde{q}_i(x = x_0, t) := \tilde{q}_i(x = x_1, t) \quad (2.44)$$

In this way, the information flowing out of the right boundary flows back into the left boundary. The system is in this way closed and depends now only on the initial conditions.

## 2.9 Hyperbolic sets of equations: the linear case with variable Jacobian

So what happens if the matrix  $A$  depends on  $x$ , i.e.  $A \rightarrow A(x)$ ? In this case also the eigenvalues change with position, like described in Section 2.2. But in addition to this also the eigenvectors can depend on spatial position  $x$ , and hence the diagonalization of the matrix. In other words, the transformation matrix becomes a local matrix:  $R \rightarrow R(x)$ . This means that one cannot decompose  $Q$  into characteristic modes globally, but this has to be done locally. This also means that the modes get mixed. Consider the following example:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} \cos x & \sin x \\ \sin x & -\cos x \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (2.45)$$

on a domain limited by  $x_0 = 0$  and  $x_1 = \pi$ . The eigenvalues of the matrix are always  $\pm 1$ , but the eigenvectors change with  $x$ . At both  $x = x_0 = 0$  and  $x = x_1 = \pi$  the eigenvectors can be written as  $e_1 = (1, 0)$  and  $e_2 = (0, 1)$ , while at, for example,  $x = \pi/2$  we have  $e_1 = (1, 1)/\sqrt{2}$  and  $e_2 = (-1, 1)/\sqrt{2}$ . Or more general:

$$e_1 = \begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \quad (2.46)$$

$$e_2 = \begin{pmatrix} -\sin x \\ \cos x \end{pmatrix} \quad (2.47)$$

(the eigenvectors at  $x_1$  are now minus those of  $x_0$ , but there is no difference in the meaning as the norm of the eigenvalues are irrelevant except in the definition of the norm of  $\tilde{q}_i$ ).

If we set  $\tilde{q}_1(x_0) = q_1(x_0) = f(t)$  where  $f(t)$  is some function of time  $t$ , and we set  $\tilde{q}_2(x_1) = 0$  and the initial value of  $\tilde{q}$  at  $\tilde{q}_i(x, t = t_0)$ , then the signal put into mode  $\tilde{q}_1$  at the left boundary initially propagates from left to right, but as it gets to larger  $x$  it starts to mix with the  $\tilde{q}_2$  mode, which moves in opposite direction.

This example shows that although hyperbolic equations are about signal propagation, this does not mean that the signals are simply pure waves moving across the domain, but can instead interact with each other even if the equations are linear. However, *locally* one can always uniquely divide the state vector  $Q$  up into the characteristic modes moving each at their own characteristic speeds.

## 2.10 Boundary conditions (II)

When the Jacobian  $m \times m$  matrix  $A$  depends on  $x$ , the question of how and where to impose boundary conditions can become, in some circumstances, a bit more difficult. The limitation that one must impose precisely  $m$  boundary conditions is no longer strictly true. It turns out to depend on the number of inward-pointing characteristics at each of the boundaries. Consider the simple example of scalar advection problem between  $x_0 = -1$  and  $x_1 = 1$  with advection speed  $A(x) = u(x) = x$ :

$$\partial_t q + x \partial_x q = 0 \quad (2.48)$$

In this case, at  $x = x_0 = -1$  the characteristic is pointing out of the domain. But the same is true at  $x = x_1 = 1$ ! So neither at  $x_0$  nor at  $x_1$  must one determine boundary conditions. The opposite example

$$\partial_t q - x \partial_x q = 0 \quad (2.49)$$

requires a Dirichlet boundary condition to be set at *both* boundaries. Here the information (signal) flows into the domain on both sides and piles up near  $x = 0$ .

In general when the sign of an eigenvalue flips one has the risk that the number of boundary conditions deviates from  $m$ . The physical interpretation of sign-flips of eigenvalues can be many. For instance, as we shall see later, the sign of one of the eigenvalues flips in the case of a standing shock in hydrodynamics. Indeed, signals pile up on both sides of the shock and a catastrophe can only be avoided by the minute, but essential viscosity in the shock front which (only very locally) ‘erases’ these converging signals again in a *non-hyperbolic* manner. But these issues will be discussed in the chapter on supersonic flows and shocks, chapter ??

## 2.11 Hyperbolic equations versus elliptic equations

As mentioned above, if the Jacobian matrix can be diagonalized and the eigenvalues are real, then the system is hyperbolic. So what if the eigenvalues are imaginary? Consider the following system, similar to the equations for waves in hydrodynamics:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (2.50)$$

defined on a domain  $x \in [x_0, x_1]$ . This clearly can be written as:

$$\partial_t^2 q_1 - \partial_x^2 q_1 = 0 \quad (2.51)$$

which is a wave equation with eigenvalue  $\lambda_{-1} = -1$  and  $\lambda_{+1} = 1$ . If the state  $Q$  is given at time  $t = t_0$ , and the boundary conditions are specified at  $x = x_0$  and  $x = x_1$  in the way explained in Section 2.8, then the function  $Q(x, t)$  within the domain can be computed for all  $t > t_0$  (in fact, also backward in time).

Now consider the following equation:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (2.52)$$

defined on a domain  $x \in [x_0, x_1]$ . This can be written as:

$$\partial_t^2 q_1 + \partial_x^2 q_1 = 0 \quad (2.53)$$

However, this is not a hyperbolic equation. The eigenvalues of the Jacobian matrix are  $\pm i$ . It is clear that we cannot do the same trick with eigenvectors and eigenvalues here, because it does not make sense to move something with a speed  $\pm i$ . The nature of this equation is therefore entirely different even though it is merely one minus sign in the Jacobian matrix. In fact, Eq. (2.53) can be recognized as the Laplace equation. It needs the specification of boundary conditions at  $x = x_0, x = x_1, t = t_0$  and  $t = t_1$ . In other words: the state at some time  $t$  depends not only on the past but also on the future. Clearly this makes not much sense, and the Laplace equation is usually more used in two (or more) spatial directions instead of space and time.

## 2.12 Hyperbolic equations: the non-linear case

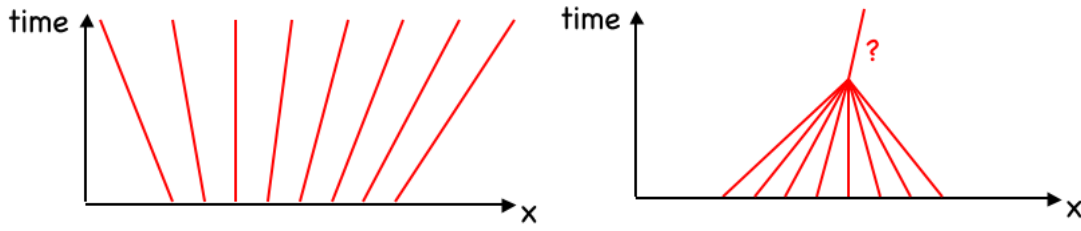
The above definition for linear hyperbolic sets of equations can be generalized to non-linear sets of equations. Let us focus on the general conservation equation:

$$\partial_t Q + \partial_x F = 0 \quad (2.54)$$

where, as ever,  $Q = (q_1, \dots, q_m)$  and  $F = (f_1, \dots, f_m)$ . In general,  $F$  is not always a linear function of  $Q$ , i.e. it cannot always be formulated as a matrix  $A$  times the vector  $Q$  (except if  $A$  is allowed to also depend on  $Q$ , but then the usefulness of writing  $F = AQ$  is a bit gone). So let us assume that  $F$  is some non-linear function of  $Q$ . Let us, for the moment, assume that  $F = F(Q, x) = F(Q)$ , i.e. we assume that there is no explicit dependence of  $F$  on  $x$ , except through  $Q$ . Then according to Eq. (2.18) we get

$$\partial_t Q + \frac{\partial F}{\partial Q} \partial_x Q = 0 \quad (2.55)$$





**Figure 2.2.** Characteristics of Burger's equation. Left: case of diverging characteristics. Right: case of converging characteristics with the formation of a singularity. Beyond the time of the creation of the singularity the solution is ill defined unless a recipe is given how to treat the singularity.

where  $\frac{\partial F}{\partial Q}$  is the Jacobian matrix, which depends, in the non-linear case, on  $Q$  itself. We can nevertheless decompose this matrix in eigenvectors (which depend on  $Q$ ) and we obtain

$$\partial_t \tilde{Q} + \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{pmatrix} \partial_x \tilde{Q} = 0 \quad (2.56)$$

Here the eigenvalues  $\lambda_1, \dots, \lambda_m$  and eigenvectors (and hence the meaning of  $\tilde{Q}$ ) depends on  $Q$ . In principle this is not a problem. The characteristics are now simply given by the state vector  $Q$  itself. The state is, so to speak, self-propagating. We are now getting into the kind of hyperbolic equations like the hydrodynamics equations, which are also non-linear self-propagating.

### 2.13 Example of non-linear conservation equation: Burger's equation

Consider

$$\partial_t q + \frac{1}{2} \partial_x (q^2) = 0 \quad (2.57)$$

This is called *Burger's equation*. It is a conservation equation in  $q$ , with flux  $f(q) = q^2/2$ . The flux is only dependent on  $x$  through  $q$ . We have  $\partial f / \partial q = q$ , so we can write the above equation as

$$\partial_t q + q \partial_x q = 0 \quad (2.58)$$

So the advection velocity is, in this important example, the to-be-advectioned quantity  $q$  itself! The quantity propagates itself with  $u = q$ . If we use the comoving derivative, then we obtain the following equation:

$$D_t q(x, t) = 0 \quad (2.59)$$

which appears to be identical to Eq. (2.8). The difference lies in the definition of  $D_t$  which is defined with respect to a given function  $u(x)$  in Eq. (2.8) and with respect to  $q(x, t)$  in Eq. (2.59).

Eq. (2.59) shows that along a characteristic the value of  $q$  does not change, or in other words: the characteristic speed (slope in the  $x, t$ -plane) does not change along a characteristic. This means that the characteristics are straight lines in the  $(x, t)$ -plane, as in the case of the example of Section 2.3. The difference to that example lies in the fact that in this case not all straight line characteristics are parallel.

The interpretation of Burger's equation is that of the *motion of a pressureless fluid*. Often it is said to be the equation describing the motion of dust in space, as dust clouds do not have



pressure and each dust particle moves along a straight line. This is only partially correct, as we shall show below, but it does describe roughly the point.

Since any non-parallel straight lines in  $(x, t)$  *must* have a crossing at some point in space we can immediately derive that for converging flows there will be a point at which Burger's equations break down.

Now to come back at the difference of Burger's equation with the motion of dust. In Burger's equation the assumption is that at any time and any position there exists only one velocity. In case of dust flows this is not necessary: since the particles do not interact, at any given time and position one can have dust particles flowing left, right at various velocities. If two dust cloud approach, in Burger's equation the equations produce shocks (i.e. a breakdown of the pure inviscid equation). In the case of dust the particles would not feel each other and the cloud simply go through each other.

Perhaps a more adequate physical interpretation of Burger's equation is that of a pressure-less fluid. If the characteristics converge a shock will happen and Burger's equation will no longer be valid and will be replaced by the more generally valid hydrodynamics equation *with* pressure.

→ **Exercise:** As we have shown in Section 2.3 a conserved quantity tends to grow in regions of converging flow and it tends to diminish in regions of diverging flow. In case of Burger's equation it appears that this is not the case, as the equation amounts to  $D_t q = 0$ . Give a modified version of Burger's equation that does have this property of increasing value for converging flow, and note which is the characteristic speed.

## 2.14 Isothermal hydrodynamic equations

The topic of this lecture is hydrodynamics, so let's express the equations of hydrodynamics in the above form. Let's take the isothermal equations for simplicity. We have

$$\partial_t \rho + \partial_x(\rho u) = 0 \quad (2.60)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + \rho c_s^2) = 0 \quad (2.61)$$

Let us define

$$q_1 \equiv \rho \quad , \quad q_2 \equiv \rho u \quad (2.62)$$

Then we can write the above equations as

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \partial_x \begin{pmatrix} q_2 \\ \frac{q_2^2}{q_1} + q_1 c_s^2 \end{pmatrix} = 0 \quad (2.63)$$

or in other words:  $Q = (q_1, q_2)$  and  $F = (q_2, q_2^2/q_1 + q_1 c_s^2)$ . This can be written with the Jacobian:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ c_s^2 - \frac{q_2^2}{q_1^2} & 2\frac{q_2}{q_1} \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (2.64)$$

The eigenvalues are

$$\lambda_{\pm} = \frac{q_2}{q_1} \pm c_s = u \pm c_s \quad (2.65)$$

and the eigenvectors are:

$$e_{\pm} = \begin{pmatrix} 1 \\ \lambda_{\pm} \end{pmatrix} \quad (2.66)$$

One sees that both the eigenvalues and the eigenvectors depend on the state vector  $(q_1, q_2)$  itself and are therefore space- and time-dependent. The state vector determines for itself how it should be decomposed. Modes mix in two different ways: a) the eigenvectors change in space and time, and b) each mode influences the other mode due to the non-linearity.

→ **Exercise:** In Section 2.6 we derived the set of eigenvectors and eigenvalues for the perturbation equation of hydrodynamics with an adiabatic equation of state. If we replace  $\gamma P_0/\rho_0$  with the isothermal sound speed  $c_s^2$ , then we obtain the results for isothermal waves. The funny thing is, however, that the Jacobian matrix in that case (Eq. 2.23) does not appear to be the linearized form of the Jacobian matrix derived in the present section (the one used in Eq. 2.64). Explain this in terms of how the linearization is done in Section 2.6.

## 2.15 Non-isothermal hydrodynamic equations

Now let us turn to the generalization of the isothermal hydrodynamics equations: the non-isothermal hydrodynamics equations. Note that this is *not* equal to the adiabatic hydrodynamics equations, because the adiabatic hydrodynamics equations assume that all of the gas lies on the same adiabat, or in other words: that the gas is isentropic. In contrast, we would now like to relax any assumption of the entropy of the gas, and allow the entropy of the gas to vary arbitrarily in space. This means necessarily that we must include a third equation: the energy equation. Now the equations become significantly more complex.

We have now:

$$\partial_t \rho + \partial_x(\rho u) = 0 \quad (2.67)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + P) = 0 \quad (2.68)$$

$$\partial_t(\rho e_{\text{tot}}) + \partial_x[(\rho e_{\text{tot}} + P)u] = 0 \quad (2.69)$$

Let us define

$$q_1 \equiv \rho, \quad q_2 \equiv \rho u, \quad q_3 \equiv \rho e_{\text{tot}} \quad (2.70)$$

Then we can write the above equations as

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \partial_x \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = 0 \quad (2.71)$$

in which

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ (\rho e_{\text{tot}} + P)u \end{pmatrix} = \begin{pmatrix} q_2 \\ (\gamma - 1)q_3 + \left(\frac{3-\gamma}{2}\right) \frac{q_2^2}{q_1} \\ \gamma \frac{q_3 q_2}{q_1} + \left(\frac{1-\gamma}{2}\right) \frac{q_2^3}{q_1^2} \end{pmatrix} \quad (2.72)$$

where we used

$$u = q_2/q_1 \quad (2.73)$$

$$P = (\gamma - 1) \left( q_3 - \frac{1}{2} \frac{q_2^2}{q_1} \right) \quad (2.74)$$

Eq. 2.71 with the above expressions for  $(f_1, f_2, f_3)$  can be written with the Jacobian:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2} \frac{q_2^2}{q_1} & (3-\gamma) \frac{q_2}{q_1} & (\gamma-1) \\ -\left\{ \gamma \frac{q_3 q_2}{q_1} + (\gamma-1) \frac{q_2^3}{q_1^2} \right\} & \left\{ \gamma \frac{q_3}{q_1} + \frac{3}{2}(1-\gamma) \frac{q_2^2}{q_1^2} \right\} & \gamma \frac{q_2}{q_1} \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = 0 \quad (2.75)$$

It can be useful to rewrite the Jacobian using the primitive variables:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2}\rho u^2 & (3-\gamma)u & (\gamma-1) \\ -\{\gamma e_{\text{tot}}u + (\gamma-1)u^3\} & \{\gamma e_{\text{tot}} + \frac{3}{2}(1-\gamma)u^2\} & \gamma u \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = 0 \quad (2.76)$$

The eigenvalues are

$$\lambda_- = u - C_s \quad (2.77)$$

$$\lambda_0 = u \quad (2.78)$$

$$\lambda_+ = u + C_s \quad (2.79)$$

$$(2.80)$$

with eigenvectors:

$$e_- = \begin{pmatrix} 1 \\ u - C_s \\ h_{\text{tot}} - C_s u \end{pmatrix} \quad (2.81)$$

$$e_0 = \begin{pmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{pmatrix} \quad (2.82)$$

$$e_+ = \begin{pmatrix} 1 \\ u + C_s \\ h_{\text{tot}} + C_s u \end{pmatrix} \quad (2.83)$$

where  $h_{\text{tot}} = e_{\text{tot}} + P/\rho$  is the total specific enthalpy and  $C_s = \sqrt{\gamma P/\rho}$  is the adiabatic sound speed.

## 2.16 Traffic flow equations

As already mentioned, hyperbolic equations are more general than only the equations of hydrodynamics. Here is an example of a model of traffic flow, as was first discussed in papers by Lighthill, Whitham and Richards ( for the references, see book LeVeque from which this example is taken). Let us assume a single lane road with a density of cars  $q$  (the number of cars per car length). We assume that  $0 \leq q \leq 1$  (because we cannot have more than 1 car per car length) and we verify a-posteriori if this condition is satisfied. The conservation equation is:

$$\partial_t q + \partial_x (qu) = 0 \quad (2.84)$$

where  $u$  is the speed of the cars at time  $t$  and position  $x$ . Suppose that there is a speed limit of  $u_{\text{max}}$  and that if the road is nearly empty, the cars drive at the speed limit  $u = u_{\text{max}}$ . If this was all, then the advection equation simply moves the density of cars linearly toward larger  $x$ . However, if the road gets more crowded drivers naturally slow down. Let us for simplicity assume that

$$u(q) = u_{\text{max}}(1 - q) \quad (2.85)$$

The flux of cars is then

$$f = q(1 - q)u_{\text{max}} \quad (2.86)$$

The flux of cars is greatest when  $q = 1/2$ . For lower  $q$  the flux is lower because the density of cars is lower, while for higher  $q$  the flux is lower because the speed of the cars goes down (congestion).

We can now write the traffic flow equation as

$$\partial_t q + u_{\max}(1 - 2q)\partial_x q = 0 \quad (2.87)$$

This shows that the characteristic velocity of the system is:

$$\lambda = u_{\max}(1 - 2q) \quad (2.88)$$

which can range from  $-u_{\max}$  to  $u_{\max}$ . It is very important to note here that the characteristic speed is not equal to the speed of propagation of the cars! It is the speed at which information is propagating, not the speed of the advected quantity  $q$  itself. This is one of the peculiar features of non-linear hyperbolic equations, and it is very similar to the peculiarities of the Burger's equation. The traffic flow equation is a particularly nice example of a non-linear hyperbolic equation because it is a very simple equation, yet has very interesting solution properties.

## 2.17 Hyperbolic equations in 2-D and 3-D

So far we have done everything only in 1-D. But what we learned can also be generalized to higher dimensions, by use of the concept of *operator splitting*. In 2-D we get

$$\partial_t Q + \partial_x F(Q) + \partial_y G(Q) \quad (2.89)$$

which can be written as:

$$\partial_t q_i + \frac{\partial f_i}{\partial q_k} \partial_x q_k + \frac{\partial g_i}{\partial q_k} \partial_y q_k \quad (2.90)$$

By operator splitting we can focus our attention to one of the space dimensions only. If we split it as

$$\partial_t q_i + \frac{\partial f_i}{\partial q_k} \partial_x q_k = -\frac{\partial f_i}{\partial q_k} \partial_y q_k \quad (2.91)$$

then we focus on the advection in x-direction, and consider the y-advection as a source term, while if we write

$$\partial_t q_i + \frac{\partial f_i}{\partial q_k} \partial_y q_k = -\frac{\partial f_i}{\partial q_k} \partial_x q_k \quad (2.92)$$

we focus on advection in y-direction and consider the x-direction as a source term. In numerical methods this operator splitting is often done to reduce the full 2-D or 3-D problem into consecutive 1-D problems which are much easier to handle.

# Chapter 3

## Advection algorithms I. The basics

Numerical solutions to (partial) differential equations always require *discretization* of the problem. This means that instead of a continuous space dimension  $x$  or time dimension  $t$  we now have:

$$x \rightarrow x_i \in \{x_1, \dots, x_{N_x}\} \quad (3.1)$$

$$t \rightarrow t_n \in \{t_1, \dots, t_{N_t}\} \quad (3.2)$$

In other words: we have replaced spacetime with a discrete set of points. This is called a *grid* or a *mesh*. The numerical solution is solved on these discrete grid points. So we must replace functions such as  $q(x)$  or  $q(x, t)$  by their discrete counterparts  $q(x_i)$  or  $q(x_i, t_n)$ . From now on we will write this as:

$$q(x_i, t_n) =: q_i^n \quad (3.3)$$

*NOTE: The upper index of  $q^n$  is not a powerlaw index, but just the time index.* We must now replace the partial differential equation also with a discretized form, with  $q_i^n$  as the quantities we wish to solve. In general we wish to find  $q_i^{n+1}$  for given  $q_i^n$ , so the equations must be formulated in the way to yield  $q_i^{n+1}$  for given  $q_i^n$ . There are infinite ways to formulate the discretized form of the PDEs of hydrodynamics, and some formulations are better than others. In fact, as we shall find out, the simplest formulations tend to be even numerically unstable. And many stable algorithms turn out to be very *diffusive*. The art of numerical hydrodynamics, or of numerical solutions to PDEs in general, is to formulate a discretized form of the PDEs such that the solutions for  $q_i^n$  for given initial condition  $q_i^1$  are numerically stable and are as close as possible to the true  $q(x_i, t_n)$ . Over the past 40 years this has turned out to be a very challenging problem, and even to this day research is on-going to design even better methods than before. One of the main problem is that there is no ideal and universally good method. Some problems require entirely different methods than others, even though they solve exactly the same equations. For example: hydrodynamics of very subsonic flows usually requires different methods than hydrodynamics of supersonic flows with shock waves. In the first case we require higher-order precision algorithms while in the second case we need so-called shock-capturing methods. We will discuss such methods in later chapters.

In this chapter we will focus our attention to the fundamental and easy to formulate problem of *numerical advection on a grid*. As we have seen in Chapter 2, the equations of hydrodynamics can be reduced to signals propagating with three different speeds: the two sound waves and the gas motion. These ‘signals’ are nothing else than the eigenvectors of the Jacobian matrix, and are therefore combinations of  $\rho$ ,  $\rho u$  and  $\rho e_{\text{tot}}$ , or in other words the eigenvector decomposition

coefficients  $(\tilde{q}_1, \tilde{q}_2, \tilde{q}_3)$ . So it should be possible to formulate numerical hydrodynamics as a numerical advection of these signals over a grid.

To simplify things we will not focus on the full set of signals. Instead we focus entirely on how a *scalar* function  $q(x, t)$  can be numerically advected over a grid. The equation is simply:

$$\partial_t q(x, t) + \partial_x [q(x, t)u(x, t)] = 0 \quad (3.4)$$

which is the conserved advection equation. This problem sounds nearly trivial, but it is far from trivial in practice. In fact, finding a proper algorithm for numerical advection of scalar functions over a grid has been one of the main challenges for numerical hydrodynamics in the early years of hydrodynamics. We will in fact reduce the complexity of the problem even further and study simply:

$$\partial_t q(x, t) + u \partial_x [q(x, t)] = 0 \quad (3.5)$$

with  $u$  is a constant. In this case the equation is automatically a conservation equation in spite of the fact that  $u$  is outside of the differential operator.

Since not all readers may be familiar with numerical methods, we will start this chapter with a recapitulation of some basic methods for the integration of functions.

### 3.1 *Prelude: Numerical integration of an ordinary differential equation*

In spite of its simplicity, the advection equation is a 2-D problem (in  $x, t$ ) which therefore already naturally has some level of complexity. To introduce some of the basic concepts that we can use later, we first turn our attention to a simple problem: the solution to an ordinary differential equation (ODE) such as

$$\frac{dq(t)}{dt} = F(t, q(t)) \quad (3.6)$$

where  $F(t, q)$  is a function of both  $t$  and  $q$ . We assume that at some time  $t = t_0$  we know the value of  $q$  and we wish to integrate this equation in time  $t$  using a numerical method. To do this we must discretize time  $t$  in discrete steps  $t_0, t_1, t_2$  etc, and the values of  $q(t)$  belonging to these time nodes can be denoted as  $q_0, q_1, q_2$  etc. So, given  $q_0$ , how can we compute  $q_i$  with  $i > 0$ ? The most straightforward method is the *forward Euler method*:

$$\frac{q_{n+1} - q_n}{\Delta t} = F(t_n, q_n) \quad (3.7)$$

which can be written as an expression for  $q_{n+1}$ :

$$q_{n+1} = q_n + F(t_n, q_n) \Delta t \quad (3.8)$$

This method is also called *explicit integration*, because the new value of  $q$  is explicitly given in terms of the old values. This is the easiest method, but it has several drawbacks. One of these drawbacks is that it becomes numerically unstable if the time step  $\Delta t$  is taken too large.

→ **Exercise:** Consider the equation

$$\frac{dq(t)}{dt} = -q(t)^2 \quad (3.9)$$

Write down the analytic solution (to later compare with). Assume  $q(t = 0) = 1$  and numerically integrate this equation using the forward Euler method to time  $t = 10$ . Plot

the numerical resulting function  $q(t)$  over the analytic solution. Experiment with different time steps  $\Delta t$ , and find out what happens when  $\Delta t$  is taken too large. Derive a reasonable condition for the maximum  $\Delta t$  that can be allowed. Find out which  $\Delta t$  is needed to get reasonable accuracy.

A way to stabilize the integration even for very large time steps is to use the *backward Euler method*:

$$q_{n+1} = q_n + F(t_{n+1}, q_{n+1})\Delta t \quad (3.10)$$

which is also often called *implicit integration*. This equation may seem like magic, because to calculate  $q_{n+1}$  we need  $q_{n+1}$  itself! The trick here is that one can often manipulate the equation such that in the end  $q_{n+1}$  is written in explicit form again. For instance:

$$\frac{dq(t)}{dt} = -q \quad (3.11)$$

discretizes implicitly to

$$q_{n+1} = q_n - q_{n+1}\Delta t \quad (3.12)$$

While this is an implicit equation, one can rewrite it to:

$$q_{n+1} = \frac{q_n}{1 + \Delta t} \quad (3.13)$$

which is stable for all  $\Delta t > 0$ . However, in many cases  $F(t, q)$  is non-linear, and this simple manipulation is not possible. In that case one is either forced to linearize the equations about the current value of  $q_n$  and perform the manipulations with  $\delta q_{n+1/2} \equiv q_{n+1} - q_n$ , or one uses iteration, in which a first guess is made for  $q_{n+1}$  and this is re-computed iteratively until convergence. The latter method is, however, rather time-consuming.

Whether implicit methods produce accurate results for large  $\Delta t$  is another issue. In fact, such implicit integration, while being numerically extraordinarily stable, is about as inaccurate as explicit integration.

An alternative method, that combines the ideas of both forward and backward Euler integration is the *midpoint rule*:

$$q_{n+1} = q_n + F(t_{n+1/2}, q_{n+1/2})\Delta t \quad (3.14)$$

where  $n + 1/2$  stands for the position between  $t_n$  and  $t_{n+1}$ . Here the problem is that we do not know  $q_{n+1/2}$ , neither currently, nor once we know  $q_{n+1}$ . For problems of integrating Hamiltonian systems this method can nevertheless work and turns out to be very useful. This is because the 'coordinates'  $q_i$  are located at  $t_n$  and the conjugate momenta  $p_i$  are located at  $t_{n+1/2}$ , and their time derivatives only depend on their conjugate ( $q$  on  $p$  and  $p$  on  $q$ ). This forms the basis of *symplectic integrators* such as the leapfrog method.

An integration method very akin to the midpoint rule, but more readily applicable is the *trapezoid rule*:

$$q_{n+1} = q_n + \frac{1}{2}[F(t_n, q_n) + F(t_{n+1}, q_{n+1})]\Delta t \quad (3.15)$$

It is half-implicit. For the above simple example (Eq. 3.11) we can thus write:

$$q_{n+1} = q_n - \frac{1}{2}q_n\Delta t - \frac{1}{2}q_{n+1}\Delta t \quad (3.16)$$



which results in:

$$q_{n+1} = \frac{1 - \Delta t/2}{1 + \Delta t/2} q_n \quad (3.17)$$

This method, when generalized to multiple dimensional partial differential equations, is called the *Crank-Nicholson* method. It has the advantage that it is usually numerically stable (though not as stable as fully implicit methods) and since the midpoint is used, it is also naturally more accurate. A variant of this method is the *predictor-corrector* method (or better: the most well-known of the predictor-corrector methods) in which a temporary prediction of  $q_{n+1}$  is made with the explicit forward Euler method, which is then used as the  $q_{n+1}$  in the trapezoid rule.

So far we have always calculated  $q_{n+1}$  on the basis of the known  $q_n$  (and, in case of implicit schemes, on  $q_{n+1}$  as well). Such methods are either first order accurate (such as the forward and backward Euler methods) or second order accurate (such as the trapezoid rule), but they can never be of higher order. However, there exist higher-order methods for integration. They either make predictor-steps in between  $t_n$  and  $t_{n+1}$  (these are the Runge-Kutta type methods) or they fit a Lagrange polynomial through  $q_n, q_{n-1}$  (or even  $q_{n-2}$  and further) to compute the integral of the ODE in the interval  $[t_n, t_{n+1}]$  for finding  $q_{n+1}$  (these are Adams-Bashforth methods). The first order Adams method is equal to the forward Euler method. The second and third order ones are:

$$q_{n+1} = q_n + \frac{\Delta t}{2} [3F(t_n, q_n) - F(t_{n-1}, q_{n-1})] \quad (3.18)$$

$$q_{n+1} = q_n + \frac{\Delta t}{12} [23F(t_n, q_n) - 16F(t_{n-1}, q_{n-1}) + 5F(t_{n-2}, q_{n-2})] \quad (3.19)$$

However, when generalized to multi-dimensional systems (such as hydrodynamics equations) such higher order multi-point schemes in time are not very often used. In astrophysics the **PENCIL** hydrodynamics code is a code that uses Runge-Kutta type integration in time, but most astrophysical codes are first order in time.

## 3.2 Numerical spatial derivatives

### 3.2.1 Second order expressions for numerical spatial derivatives

To convert the PDEs of Eqs. (3.4,3.5) into a discrete form we need to formulate the derivatives in discrete form. A derivative is defined as:

$$\frac{\partial q}{\partial x} = \lim_{\Delta x \rightarrow 0} \frac{q(x + \Delta x) - q(x)}{\Delta x} \quad (3.20)$$

For  $\Delta x$  we could take  $x_{i+1} - x_i$ , but in a numerical calculation we cannot do the  $\lim_{\Delta x \rightarrow 0}$  because it would require infinite number of grid points. So the best we can do is write:

$$\left. \frac{\partial q}{\partial x} \right|_{i+1/2} = \frac{q_{i+1} - q_i}{x_{i+1} - x_i} + \mathcal{O}(\Delta x^2) \simeq \frac{q_{i+1} - q_i}{x_{i+1} - x_i} \quad (3.21)$$

where  $\Delta x \equiv (x_{i+1} - x_i)$ , and we assume for the moment that the grid is constantly spaced. Note that this is an approximate expression of the derivative defined *in between* the grid points  $x_{i+1}$  and  $x_i$ . For this reason we denote this as the derivative at  $i + 1/2$ , which is just a way to index locations in-between grid points. Often one needs the derivative not in between two grid points ( $i + 1/2$ ), but precisely at a grid point ( $i$ ). This can be written as:

$$\left. \frac{\partial q}{\partial x} \right|_i = \frac{q_{i+1} - q_{i-1}}{x_{i+1} - x_{i-1}} + \mathcal{O}(\Delta x^2) \simeq \frac{q_{i+1} - q_{i-1}}{x_{i+1} - x_{i-1}} \quad (3.22)$$



So depending where we need the derivative, we have different expressions for them.

The  $\mathcal{O}(\Delta x^2)$  in Eqs. (3.21, 3.22) is a way of writing the deviation between the ‘true’ derivative and the approximation. Of course, the true derivative is only defined as long as  $q(x)$  is smoothly defined. In our numerical algorithm we do not have  $q(x)$ : we only have  $q_i$ , and hence the ‘true’ derivative is not defined. But it does show that if we re-express the whole problem on a finer grid, i.e. with smaller  $\Delta x$ , then the approximate answer approaches the true one as  $\Delta x^2$ .

To see this, we assume we know the smooth function  $q(x)$ . We say that  $q_i = q(x_i)$  and  $q_{i+1} = q(x_{i+1}) = q(x_i + \Delta x)$  and we express  $q(x_i + \Delta x)$  as a Taylor series:

$$q(x_i + \Delta x) = q(x_i) + q'(x_i)\Delta x + \frac{1}{2}q''(x_i)\Delta x^2 + \frac{1}{6}q'''(x_i)\Delta x^3 + \mathcal{O}(\Delta x^4) \quad (3.23)$$

where the ' (prime) denotes the derivative to  $x$ . Then we insert this into the numerical derivative at  $i$ :

$$\begin{aligned} \frac{q_{i+1} - q_{i-1}}{2\Delta x} &= \frac{q_i + q'_i\Delta x + \frac{1}{2}q''_i\Delta x^2 + \frac{1}{6}q'''_i\Delta x^3 + \dots - q_i + q'_i\Delta x - \frac{1}{2}q''_i\Delta x^2 + \frac{1}{6}q'''_i\Delta x^3 + \dots}{2\Delta x} \\ &= q'_i + \frac{1}{6}q'''_i\Delta x^2 + \dots \end{aligned} \quad (3.24)$$

This shows that the deviations are of order  $\mathcal{O}(\Delta x^2)$ .

### 3.2.2 Higher-order expressions for numerical derivatives

The  $\mathcal{O}(\Delta x^2)$  also shows that there must be various other expressions possible for the derivative that are equally valid. Eq. 3.21 is an example of a *two-point* derivative at position  $i + 1/2$ . Eq. 3.22 is also a two-point derivative, this time at position  $i$ , but it is defined on a *three-point stencil*. A *stencil* around point  $i$  is a mini-grid of points that contribute to the things we wish to evaluate at  $i$ . We can also define a derivative at  $i$  on a *five-point stencil*:

$$\left. \frac{\partial q}{\partial x} \right|_i = \frac{-q_{i+2} + 8q_{i+1} - 8q_{i-1} + q_{i-2}}{12\Delta x} + \mathcal{O}(\Delta x^4) \quad (3.25)$$

*Note: this expression is only valid for a constantly-spaced grid!*

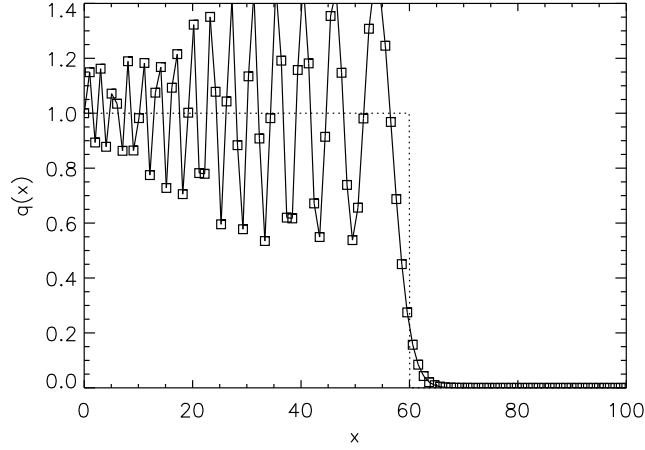
→ **Exercise:** Show that this expression indeed reproduces the derivative to order  $\mathcal{O}(\Delta x^4)$ .

Some hydrodynamics codes are based on such higher-order numerical derivatives. Colloquially it is usually said that higher-order schemes are more accurate than lower-order schemes. However, this is only true if the function  $q(x)$  is reasonably smooth over length scales of order  $\Delta x$ . In other words: the  $\mathcal{O}(\Delta x^4)$  is only significantly smaller than  $\mathcal{O}(\Delta x^2)$  if  $\partial_x^5 q(x)\Delta x^4 \ll \partial_x^3 q(x)\Delta x^2 \ll \partial_x q(x)$ . Higher-order schemes are therefore useful for flows that have no strong discontinuities in them. This is often true for *subsonic* flows, i.e. flows for which the sound speed is much larger than the typical flow speeds. For problems involving shock waves and other types of discontinuities such higher-order schemes turn out to be worse than lower order ones, as we will show below.

## 3.3 Some first advection algorithms

In this section we will try out a few algorithms and find out what properties they have. We focus again on the advection equation

$$\partial_t q + u \partial_x q = 0 \quad (3.26)$$



**Figure 3.1.** Result of center-difference algorithm for advection of a step-function from left to right. Solid line and symbols: numerical result. Dotted line: true answer (produced analytically).

with constant  $u > 0$ . The domain of interest is  $[x_0, x_1]$ . We assume that the initial state  $q(x, t = t_0)$  is given on this domain, and we wish to solve for  $q(x, t > t_0)$ . A boundary condition has to be specified at  $x = x_0$ .

### 3.3.1 Centered-differencing scheme

The simplest discretization of the equation is:

$$\frac{q_i^{n+1} - q_i^n}{t_{n+1} - t_n} + u \frac{q_{i+1}^n - q_{i-1}^n}{x_{i+1} - x_{i-1}} = 0 \quad (3.27)$$

in which we use the  $n + 1/2$  derivative in the time direction and the  $i$  derivative in space. Let us assume that the space grid is equally-spaced so that we can always write  $x_{i+1} - x_{i-1} = 2\Delta x$ . We can then rewrite the above equation as:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x} u (q_{i+1}^n - q_{i-1}^n) \quad (3.28)$$

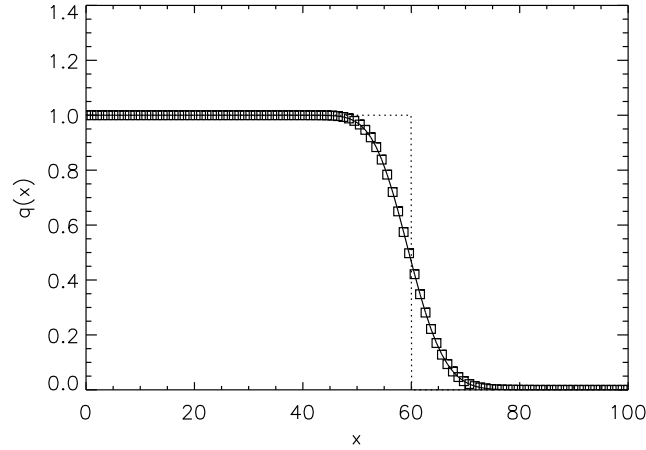
This is one of the simplest advection algorithms possible.

Let us test it by applying it to a simple example problem. We take  $x_0 = 0, x_1 = 100$  and:

$$q(x, t = t_0) = \begin{cases} 1 & \text{for } x \leq 30 \\ 0 & \text{for } x > 30 \end{cases} \quad (3.29)$$

As boundary condition at  $x = 0$  we set  $q(x = 0, t) = 0$ . Let us use an  $x$ -grid with spacing  $\Delta x = 1$ , i.e. we have 100 grid points located at  $i + 1/2$  for  $i \in [0, 99]$ . Let us choose  $\Delta t \equiv t_{n+1} - t_n$  to be  $\Delta t = 0.1\Delta x$  for now. If we do 300 time steps, then we expect the jump in  $q$  to be located at  $x = 60$ . In Fig. 3.1 we see the result that the numerical algorithm produces.

One can see that this algorithm is numerically unstable. It produces strong oscillations in the downstream region. For larger  $\Delta t$  these oscillations become even stronger. For smaller  $\Delta t$  they may become weaker, but they are always present. Clearly this algorithm is of no use. We should find a better method



**Figure 3.2.** Result of center-difference algorithm for advection of a step-function from left to right. Solid line and symbols: numerical result. Dotted line: true answer (produced analytically).

### 3.3.2 Upstream(Upwind) differencing

One reason for the failure of the above centered-difference method is the fact that the information needed to update  $q_i^n$  in time is derived from values of  $q$  in both *upstream* and *downstream* directions<sup>1</sup>. The upstream direction at some point  $x_i$  is the direction  $x < x_i$  (for  $u > 0$ ) since that is the direction from which the flow comes. The downstream direction is  $x > x_i$ , i.e. the direction where the stream goes. Anything that happens to the flow downstream from  $x_i$  should never affect the value of  $q(x_i)$ , because that information *should* flow further away from  $x_i$ . This is because, by definition, information flows downstream. Unfortunately, for the centered differencing scheme, information can flow upstream: the value of  $q_i^{n+1}$  depends as much on  $q_{i+1}^n$  (downstream direction) as it does on  $q_{i-1}^n$  (upstream direction). Clearly this is unphysical, and this is one of the reasons that the algorithm fails.

A better method is, however, easily generated:

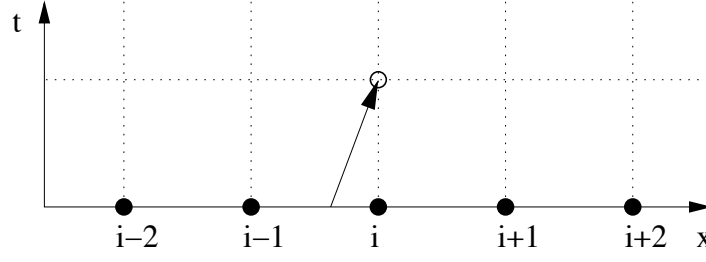
$$\frac{q_i^{n+1} - q_i^n}{t_{n+1} - t_n} + u \frac{q_i^n - q_{i-1}^n}{x_i - x_{i-1}} = 0 \quad (3.30)$$

which is, by the way, only valid for  $u > 0$ . In this equation the information is clearly only from upstream to downstream. This is called *upstream differencing*, often also called *upwind differencing*. The update for the state is then:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x} u (q_i^n - q_{i-1}^n) \quad (3.31)$$

If we now do the same experiment as before we obtain the results shown in Figure 3.2. This looks already much better. It is not unstable, and it does not produce values larger than the maximum value of the initial state, nor values smaller than the minimal value of the initial state. It is also *monotonicity preserving*, meaning that it does not produce new local minima or maxima. However, the step function is smeared out considerably, which is an undesirable property.

<sup>1</sup>These are also often called *upwind* and *downwind* directions.



**Figure 3.3.** Graphical representation of the back-tracing method of the upwind scheme.

In Fig. 3.3 shows the ‘physical’ interpretation of the upstream algorithm. It shows that if we want to know what the value of  $q_i$  is at time  $n + 1$ , then we can trace back the flow with a speed  $-u$  from time  $n + 1$  to time  $n$ . We land somewhere in the middle between grid point  $i$  and grid point  $i - 1$  (for positive  $u$ ). We say that  $q_i^{n+1} = q^n(x = x_i - u\Delta t)$ , and we find  $q^n(x = x_i - u\Delta t)$  by linear interpolation between  $i - 1$  and  $i$ . If we do so, we arrive precisely at Eq. (3.31).

### 3.4 Numerical diffusion

The smearing-out of the solution in Section 3.3.2 is a result of *numerical diffusion*. To understand numerical diffusion, we first have to understand how true diffusion is modeled in numerical methods. This is what we will do in Subsection ???. Then we will analyze numerical diffusion in more detail.

#### 3.4.1 *Intermezzo: The diffusion equation*

Let us, for now, forget about the advection equation and concentrate on another type of equation:

$$\partial_t q - D \partial_x^2 q = 0 \quad (3.32)$$

This is the diffusion equation for constant diffusion coefficient  $D$ . A delta function  $q(x, 0) = \delta(x)$  will smear out as a Gaussian:

$$q(x, t) = \frac{1}{\sqrt{\pi h}} \exp(-x^2/h^2) \quad (3.33)$$

with

$$h(t) = \sqrt{4Dt} \quad (3.34)$$

In discretized form we obtain:

$$\frac{q_i^{n+1} - q_i}{t_{n+1} - t_n} - D \frac{2}{x_{i+1} - x_{i-1}} \left( \frac{q_{i+1}^n - q_i^n}{x_{i+1} - x_i} - \frac{q_i^n - q_{i-1}^n}{x_i - x_{i-1}} \right) = 0 \quad (3.35)$$

For constant grid spacing we obtain:

$$\frac{q_i^{n+1} - q_i}{\Delta t} - D \frac{q_{i+1}^n - 2q_i^n + q_{i-1}^n}{\Delta x^2} = 0 \quad (3.36)$$

This shows that the combination  $q_{i+1}^n - 2q_i^n + q_{i-1}^n$  is a discrete way of writing a diffusion term.

### 3.4.2 Numerical diffusion of advection algorithms

So now let us go back to the pure advection equation. Even though this equation does not have any diffusion in it, the numerical algorithm to solve this advection equation intrinsically has some diffusion. In fact, there exists no numerical method without numerical diffusion. Some algorithms have more of it, some have less, and there exist method to constrain the smearing-out of discontinuities (see Section 4.5 on *flux limiters*). But in principle numerical diffusion is unavoidable.

One way of seeing this is by doing the following exercise. Consider the upwind scheme. It uses the derivative  $i - 1/2$  for the update of the state at  $i$ . In principle this is a bit cheating, since one ‘should’ use the  $i$  derivative for the update at  $i$ . So let us write the derivative at  $i - 1/2$  as:

$$\frac{q_i - q_{i-1}}{\Delta x} = \frac{q_{i+1} - q_{i-1}}{2\Delta x} - \Delta x \frac{q_{i+1} - 2q_i + q_{i-1}}{2\Delta x^2} \quad (3.37)$$

The left-hand-side is the upstream difference, the first term on the right-hand-side is the centered difference and the second term on the right-hand-side can be recognized as a diffusion term with diffusion constant

$$D = \frac{\Delta x u}{2} \quad (3.38)$$

This shows that the upstream difference scheme can be regarded to be the same as the centered difference scheme supplemented with a diffusion term. The pure centered difference scheme is unstable, but once a bit of diffusion is added, the algorithm stabilizes. The drawback is, however, that the diffusion smears any features out. If one would *define* the centered difference formulation of the  $x$ -derivative as the ‘true’ derivative (which is of course merely a definition), then the numerical diffusion of the upstream differencing scheme is quantified by  $D$  as given in Eq. (3.38).

In practice it is not possible to perfectly define the diffusivity of an algorithm since the centered difference formulation of the derivative is also merely an approximation of the true derivative. But it is nevertheless a useful way of looking at the concept of numerical diffusion. In principle one could say that it is as if we are solving

$$\partial_t q + u \partial_x q - \frac{\Delta x u}{2} \partial_x^2 q = 0 \quad (3.39)$$

Clearly, for  $\Delta x \rightarrow 0$  the diffusion vanishes. This is obviously a necessary condition, otherwise we would be modeling the true diffusion equation, which is not what we want. The diffusion that we see here is merely a by-product of the numerical algorithm we used.

Note that sometimes (as we shall see below) it is useful to add some viscosity on purpose to an algorithm. This is called *artificial viscosity*. One could therefore say that the upstream differencing is equal to centered differencing plus artificial viscosity.

## 3.5 Courant-Friedrichs-Lewy condition

No matter how stable an explicit numerical algorithm is, it cannot work for arbitrarily large time step  $\Delta t$ . If, in the above example (with  $\Delta x = 1$  and  $u = 1$ ), we were to use the upstream differencing method but we would take  $\Delta t = 2$ , then the algorithm would produce completely unstable results. The reason is the following: The function  $q$  is advected over a distance of  $u\Delta t$  in a time step  $\Delta t$ . If  $u\Delta t > \Delta x$ , then within a single time step the function is advected over a larger distance than the grid spacing. However, with the above upstream differencing method the

new  $q_i^{n+1}$  depends *only* on the old  $q_{i-1}^n$  and  $q_i^n$  values. The algorithm does not include information about the value of  $q_{i-2}^n$ , but with such a large  $\Delta t$  it should have included it. The algorithm does not know (at least within a single time step) about  $q_{i-2}^n$  and therefore it produces something that is clearly not a solution.

To keep a numerical algorithm stable the time step has to obey the *Courant-Friedrichs-Lewy condition* (CFL condition) which states that the domain of dependence of  $q_i^{n+1}$  of the algorithm at time  $t = t_n$  should include the true domain of dependence at time  $t = t_n$ . Or in other words: nothing is allowed to flow more than 1 grid spacing within one time step. This means quantitatively

$$\Delta t \leq \frac{\Delta x}{u} \quad (3.40)$$

So the higher the velocity  $u$ , the smaller the maximum allowed time step.

For the case that  $u \rightarrow u(x)$  (i.e. space-dependent velocity) this gives different time step constraints at different locations. The allowed global time step is then the smallest of these.

Not always one wants to take this maximum allowed time step. Typically one takes:

$$\Delta t = C \min(\Delta x/u) \quad (3.41)$$

where  $C$  is the *Courant number*. If one takes this 1, then one takes the maximum allowed time step. If it is 0.5 then one takes half of it.

The CFL condition is a necessary (but not sufficient) condition for the stability of any *explicit differencing method*. All the methods we have discussed here, and most of the methods we will discuss lateron, are explicit differencing methods. The word ‘explicit’ points to the fact that the updated state  $q_i^{n+1}$  is explicitly formulated in terms of  $q_{i\pm k}^n$ . There exist also so called ‘implicit differencing’ methods, but they are often too complex and therefore less often used.

### 3.6 Local truncation error and order of the algorithm

Now that we have seen some of the basics of numerical advection algorithms, let us analyze how accurate such algorithms are. Let us define  $q_e(x, t)$  to be an exact solution to the advection equation and  $q_i^n$  a discrete solution to the numerical advection equation. The numerical algorithm will be represented by a *transport operator*  $T$ :

$$q_i^{n+1} = T[q_i^n] \quad (3.42)$$

which is another way of writing the discretized PDE. In case of the upstream differencing method we have  $T[q_i^n] = q_i^n - \frac{\Delta t}{\Delta x} u (q_i^n - q_{i-1}^n)$  (cf. Eq. 3.31). For this method the  $T$  operator is a *linear operator*. Note, incidently, that if the PDE is a linear PDE, that does not guarantee that the transport operator  $T$  is also necessarily linear. Lateron in this chapter we will get to know non-linear operators that represent linear PDEs.

We can also define the discrete values of the exact solution:

$$q_{e,i}^n \equiv q_e(x_i, t_n) \quad (3.43)$$

The values  $q_{e,i}^n$  do not in general strictly obey Eq. (3.42). But we can apply the operator  $T$  to  $q_{e,i}^n$  and compare to  $q_{e,i}^{n+1}$ . In other words: we can see which error the discrete operator  $T$  introduces in one single time step compared to the true solution  $q_{e,i}^{n+1}$ . So let us apply  $T$  to  $q_{e,i}^n$  and define the *one step error (OSE)* as follows:

$$OSE = T[q_{e,i}^n] - q_{e,i}^{n+1} \quad (3.44)$$

By using a Taylor expansion we can write  $q_{e,i}^{n+1}$  as:

$$q_{e,i}^{n+1} = q_{e,i}^n + \left( \frac{\partial q(x_i, t)}{\partial t} \right)_{t=t_n} \Delta t + \frac{1}{2} \left( \frac{\partial^2 q(x_i, t)}{\partial t^2} \right)_{t=t_n} \Delta t^2 + \mathcal{O}(\Delta t^3) \quad (3.45)$$

If we use the upstream difference scheme, we can write:

$$T[q_{e,i}^n] = q_{e,i}^n - \frac{\Delta t}{\Delta x} u (q_{e,i}^n - q_{e,i-1}^n) \quad (3.46)$$

where we can write:

$$q_{e,i-1}^n = q_{e,i}^n - \left( \frac{\partial q(x, t_n)}{\partial x} \right)_{x=x_i} \Delta x + \frac{1}{2} \left( \frac{\partial^2 q(x, t_n)}{\partial x^2} \right)_{x=x_i} \Delta x^2 + \mathcal{O}(\Delta x^3) \quad (3.47)$$

So the OSE of this scheme becomes:

$$\begin{aligned} OSE = & -u \left( \frac{\partial q(x, t_n)}{\partial x} \right)_{x=x_i} \Delta t + \frac{1}{2} u \left( \frac{\partial^2 q(x, t_n)}{\partial x^2} \right)_{x=x_i} \Delta t \Delta x + u \Delta t \mathcal{O}(\Delta x^2) - \\ & \left( \frac{\partial q(x_i, t)}{\partial t} \right)_{t=t_n} \Delta t - \frac{1}{2} \left( \frac{\partial^2 q(x_i, t)}{\partial t^2} \right)_{t=t_n} \Delta t^2 + \mathcal{O}(\Delta t^3) \end{aligned} \quad (3.48)$$

If we ignore all terms of higher order we obtain:

$$OSE = -\Delta t \left[ \left( \frac{\partial q(x_i, t)}{\partial t} \right)_{t=t_n} + u \left( \frac{\partial q(x, t_n)}{\partial x} \right)_{x=x_i} - \mathcal{O}(\Delta t) - \mathcal{O}(\Delta x) \right] \quad (3.49)$$

From the PDE we know that the first two terms between brackets cancel identically, so we obtain:

$$OSE = \Delta t [\mathcal{O}(\Delta t) + \mathcal{O}(\Delta x)] \quad (3.50)$$

So what happens when we make  $\Delta t$  smaller: the OSE gets smaller by a factor of  $\Delta t^2$ . However, one must keep in mind that one now has to do more time steps to arrive at the same simulation end-time. Therefore the final error goes down only as  $\Delta t$ , i.e. linear instead of quadratic. That is why it is convenient if we define the so-called *local truncation error (LTE)* as:

$$LTE \equiv \frac{1}{\Delta t} (T[q_{e,i}^n] - q_{e,i}^{n+1}) \quad (3.51)$$

which for this particular scheme is:

$$LTE = \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x) \quad (3.52)$$

In general an algorithm is called *consistent* with the partial differential equation when the LTE behaves as:

$$LTE = \sum_{k=0, l} \mathcal{O}(\Delta t^k \Delta x^{l-k}) \quad (3.53)$$

with  $l \leq 1$ . The LTE is of order  $l$ , and the algorithm is said to be  $l$ -th order.

The upstream differencing algorithm is clearly a first order scheme.

### 3.7 Lax-Richtmyer stability analysis of numerical advection schemes

The mere fact that the LTE goes with  $\mathcal{O}(\Delta t)$  and  $\mathcal{O}(\Delta x)$  is not a sufficient condition for stability. It says that the operator  $T$  truly describes the discrete version of the PDE under consideration. But upon multiple successive actions of the operator  $T$ , representing the time sequence of the function  $q$ , tiny errors could conceivably grow exponentially and eventually dominate the solution. We want to find out under which conditions this happens.

To analyze stability we need to first define what we mean by stability. To do this we must define a *norm*  $||\cdot||$  by which we can measure the magnitude of the error. In general we define the  $p$ -norm of a function  $E(x)$  as:

$$||E||_p = \left( \int_{-\infty}^{\infty} |E(x)|^p dx \right)^{1/p} \quad (3.54)$$

For the discretized function  $E_i$  this becomes:

$$||E||_p = \left( \Delta x \sum_{i=-\infty}^{\infty} |E_i|^p \right)^{1/p} \quad (3.55)$$

The most commonly used are the 1-norm and the 2-norm. For conservation laws the 1-norm is attractive because this norm can be directly used to represent the conservation of this quantity. However, the 2-norm is useful for linear problems because it is compatible with a Fourier analysis of the problem (see Section 3.8).

Now suppose we start at  $t = 0$  with a function  $q_e(x, t = 0)$  and we set the discrete numerical solution at that time to these values:  $q_i^0 = q_{e,i}^0 \equiv q_e(x_i, 0)$ . The evolution in time is now given by a successive application of the operator  $T$ , such that at time  $t = t_n$  the discrete solution is:

$$q_i^n = T^n[q_i^0] \quad (3.56)$$

In each of these time steps the discrete solution acquires an error. We can write the total accumulated global error at time  $t = t_n$  as  $E_i^n$  defined as:

$$E_i^n = q_i^n - q_{e,i}^n \quad (3.57)$$

i.e. the difference between the discrete solution and the true solution. So when we apply the operator  $T$  to  $q_i^n$  we can write:

$$q_i^{n+1} \equiv T[q_i^n] = T[q_{e,i}^n + E_i^n] \quad (3.58)$$

So we can now write the global error at time  $t_{n+1}$ ,  $E_i^{n+1}$  as

$$\begin{aligned} E_i^{n+1} &= q_i^{n+1} - q_{e,i}^{n+1} \\ &= T[q_{e,i}^n + E_i^n] - q_{e,i}^{n+1} \\ &= T[q_{e,i}^n + E_i^n] - T[q_{e,i}^n] + T[q_{e,i}^n] - q_{e,i}^{n+1} \\ &= T[q_{e,i}^n + E_i^n] - T[q_{e,i}^n] + \Delta t L T E[q_{e,i}^n] \end{aligned} \quad (3.59)$$

Now in the next few paragraphs we will show that the numerical method is stable in some norm  $||\cdot||$  if the operator  $T[\cdot]$  is a *contractive operator*, defined as an operator for which

$$||T[P] - T[Q]|| \leq ||P - Q|| \quad (3.60)$$



for any functions  $P$  and  $Q$ . To show this we write

$$\begin{aligned} \|E^{n+1}\| &\leq \|T[q_e^n + E^n] - T[q_e^n]\| + \Delta t \|LTE[q_e^n]\| \\ &\leq \|E^n\| + \Delta t \|LTE[q_e^n]\| \end{aligned} \quad (3.61)$$

If we apply this recursively we get

$$\|E^N\| \leq \Delta t \sum_{n=1}^N \|LTE[q_e^n]\| \quad (3.62)$$

where we assume that the error at  $t = t_0 = 0$  is zero. Now, the LTE is defined always on the true solution  $q_{e,i}^n$ . So since the true solution is for sure well-behaved (numerical instabilities are only expected to arise in the numerical solution  $q_i^n$ ), we expect that the  $\|LTE[q_e^n]\|$  is a number that is bounded. If we define  $M_{LTE}$  to be

$$M_{LTE} = \max_{1 \leq n \leq N} \|LTE[q_e^n]\| \quad (3.63)$$

which is therefore also a bound number (not subject to exponential growth), then we can write:

$$\|E^N\| \leq N \Delta t M_{LTE} \quad (3.64)$$

or with  $t = N \Delta t$ :

$$\|E^N\| \leq t M_{LTE} \quad (3.65)$$

This shows that the final global error is bound, i.e. not subject to run-away growth, if the operator  $T$  is contractive. Note that this analysis, so far, holds both for linear operators  $T[\cdot]$  as well as for non-linear operators  $T[\cdot]$ . We will cover non-linear operators in the next chapter.

If  $T[\cdot]$  is linear one can write  $T[q_e^n + E^n] - T[q_e^n] = T[E^n]$ . In this case the stability requirement is:

$$\|T[E^n]\| \leq \|E^n\| \quad (3.66)$$

which must be true for any function  $E_i^n$ . In Section 3.8 we will verify this for some simple algorithms.

Sometimes the above stability requirement is loosened a bit. The idea behind this is that we are not concerned if modes grow very slowly, as long as these modes do not start to dominate the solution. If the operator  $T[\cdot]$  obeys

$$\|T[P] - T[Q]\| \leq (1 + \alpha \Delta t) \|P - Q\| \quad (3.67)$$

where  $\alpha$  is some constant (which we will constrain later), then Eq.(3.61) becomes:

$$\|E^{n+1}\| \leq (1 + \alpha \Delta t) \|E^n\| + \Delta t \|LTE[q_e^n]\| \quad (3.68)$$

and thereby Eq.(3.65) becomes:

$$\|E^N\| \leq t M_{LTE} e^{\alpha t} \quad (3.69)$$

One sees that the error growth exponentially in time, which one would in principle consider an instability. But  $e^{\alpha t}$  is a constant that does not depend on  $\Delta t$ . So no matter how large  $e^{\alpha t}$  is, as long as the LTE is linear or higher in  $\Delta t$  (a requirement for *consistency*) one can always find a  $\Delta t$  small enough such that  $\|E^N\| \ll \|q_e^N\|$  even though this might require a very large number of time steps for the integration. This means that for an operator  $T[\cdot]$  obeying Eq.(3.67) the

algorithm is formally stable. In practice, of course, the  $\alpha$  cannot be too large, or else one would require too many time steps (i.e. too small  $\Delta t$ ) to be of any practical use.

This leads us to a fundamental theorem of numerical integration methods: *Lax Equivalence Theorem* which says that:

$$\text{Consistency} + \text{Stability} \rightarrow \text{Convergence}$$

In other words: if an algorithm is *consistent* (see Eq.3.53) and *stable* (see Eq.3.69), then one can be assured that one can find a small enough  $\Delta t$  such that at time  $t$  the right answer is reached down to an accuracy of choice.

### 3.8 Von Neumann stability analysis of numerical advection schemes

The theoretical stability analysis outlined in the previous section has only reformulated the condition for stability as a condition on the operator  $T[\cdot]$ . We now analyze whether a certain algorithm in fact satisfies this condition. For linear operators the Von Neumann analysis does this in Fourier space using the 2-norm. Also we will require strong stability, in the sense that we want to show that  $\|T[E^n]\| \leq \|E^n\|$  (i.e. without the  $(1 + \alpha\Delta t)$  factor).

Any function can be expressed as a sum of complex wave functions. For an infinite space one can therefore write the initial condition function  $q(x)$  as

$$q(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{q}(k) e^{ikx} dk \quad (3.70)$$

Since the advection equation

$$\partial_t q + u \partial_x q = 0 \quad (3.71)$$

merely moves this function with velocity  $u$ , the solution  $q(x, t) = q(x - ut)$  translates in a  $\tilde{q}(k, t)$  given by

$$\tilde{q}(k, t) = \tilde{q}(k) e^{-iuk t} \quad (3.72)$$

which is just a phase rotation. In Fourier space, the true operator  $T_e[\cdot]$  is therefore merely a complex number:  $T_e = e^{-iuk\Delta t}$ . As we shall see below, the numerical operator in Fourier space is also a complex number, though in general a slightly different one. So we need to compare the numerical operator  $T[\cdot]$  with the true operator  $T_e[\cdot]$  to find out what the local truncation error (LTE) is.

Formally, when we follow the logic of Section 3.7, we need to let the operator  $T[\cdot]$  act on the Fourier transforms of  $q_{e,i}^n + E_i^n$  and  $q_{e,i}^n$  and subtract them. Or since the operator is linear, we must apply it to the Fourier transform of  $E_i^n$ . However, since we have assumed that the operator is linear, we can also do the analysis directly on  $\tilde{q}_e^n(k)$  (the Fourier transform of  $q_{e,i}^n$ ) and check if the resulting amplitude is  $\leq 1$ . The advantage is that we can then directly derive the LTE in terms of an amplitude error and a phase error. If the amplitude of the operator  $T$  is  $\leq 1$  for all values of  $k$ ,  $\Delta x$  and  $\Delta t \leq C\Delta x/|u|$  (where  $u$  is the advection speed and  $C$  is the Courant number) then we know that the function  $\tilde{q}(k, t)$  is not growing exponentially, and therefore also the error is not growing exponentially. Moreover, we know that if this amplitude is much smaller than 1, then the algorithm is very diffusive. We can also analyze the phase error to see if the algorithm transports each mode with the correct speed.

### 3.8.1 Analyzing the centered differencing scheme

Let us descretize this function as:

$$q_i^0 := q(x = x_i, 0) = e^{ikx_i} \quad (3.73)$$

Now insert this into the centered difference scheme:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x} u(q_{i+1}^n - q_{i-1}^n) \quad (3.74)$$

We obtain

$$q_i^1 = e^{ikx_i} - \frac{\Delta t}{2\Delta x} u(e^{ik(x_i+\Delta x)} - e^{ik(x_i-\Delta x)}) = e^{ikx_i} \left[ 1 - \frac{\Delta t}{2\Delta x} u(e^{ik\Delta x} - e^{-ik\Delta x}) \right] \quad (3.75)$$

Let us define:

$$\epsilon \equiv u \frac{\Delta t}{\Delta x} \quad (3.76)$$

and

$$\beta \equiv k\Delta x \quad (3.77)$$

then we can write

$$q_i^{n+1} = q_i^n T^C \quad (3.78)$$

with  $T^T$  the transfer function, which for centered differencing is apparently

$$T^C = 1 - \frac{\epsilon}{2}(e^{i\beta} - e^{-i\beta}) \quad (3.79)$$

The  $C$  in the transfer function stands for ‘centered differencing’. We can write  $T^C$  as

$$T^C = 1 - i\epsilon \sin \beta \quad (3.80)$$

This transfer function is most easily analyzed by computing the squared magnitude  $R$

$$R = T^* T = (\text{Re}T)^2 + (\text{Im}T)^2 \quad (3.81)$$

and the phase  $\Phi$

$$\tan \Phi = \frac{\text{Im}T}{\text{Re}T} \quad (3.82)$$

which for this algorithm are:

$$R^T = 1 + \epsilon^2 \sin^2 \beta, \quad \tan \Phi^T = -\epsilon \sin \beta \quad (3.83)$$

We can now compare this to our analytic solution (the solution that should have been produced):  $q(x, t) = e^{ik(x-u\Delta t)}$ . Clearly this analytic solution has  $R = 1$  and a phase of  $\Phi = uk\Delta t$  (if phase is measured negatively). Compared to this solution we see that:

1. The centered differencing scheme diverges: the amplitude of the solution always gets bigger. For very small time steps this happens with a factor  $1 + (uk\Delta t)^2$ . So clearly it gets better for smaller time steps (even if we have to take more of them), but it still remains *unconditionally unstable*.
2. The phase also has an error:  $u\delta t[\sin(k\Delta x) - k\Delta x]/\Delta x$ . For very small time steps the phase error becomes:  $-k^3 u \Delta t \Delta x^2 / 6$ , which means that the phase error grows linearly in time, independent of  $\Delta t$ .

These results confirm our numerical experience that the centered differencing method is unconditionally unstable.

### 3.8.2 Now adding artificial viscosity

We have seen in the numerical experiments of Section 3.3.2 that adding *artificial viscosity* (see Section 3.4.2) can stabilize an algorithm. So let us now consider the following advection scheme:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x} u(q_{i+1}^n - q_{i-1}^n) + D \frac{\Delta t}{\Delta x^2} (q_{i+1}^n - 2q_i^n + q_{i-1}^n) \quad (3.84)$$

Let us define

$$\nu = D \frac{\Delta t}{\Delta x^2} \quad (3.85)$$

so that we get

$$q_i^{n+1} = q_i^n - \frac{\epsilon}{2} (q_{i+1}^n - q_{i-1}^n) + \nu (q_{i+1}^n - 2q_i^n + q_{i-1}^n) \quad (3.86)$$

Let us again insert  $q_i^0 := q(x = x_i, 0) = e^{ikx_i}$  so that we obtain

$$\begin{aligned} q_i^1 &= e^{ikx_i} - \frac{\epsilon}{2} (e^{ik(x_i+\Delta x)} - e^{ik(x_i-\Delta x)}) + \nu (e^{ik(x_i+\Delta x)} - 2e^{ikx_i} + e^{ik(x_i-\Delta x)}) \\ &= e^{ikx_i} \left[ 1 - \frac{\epsilon}{2} (e^{ik\Delta x} - e^{-ik\Delta x}) + \nu (e^{ik\Delta x} - 2 + e^{-ik\Delta x}) \right] \end{aligned} \quad (3.87)$$

so the transfer function becomes:

$$T^{CD} = 1 - \frac{\epsilon}{2} (e^{ik\Delta x} - e^{-ik\Delta x}) + \nu (e^{ik\Delta x} - 2 + e^{-ik\Delta x}) \quad (3.88)$$

or in other terms:

$$T^{CD} = 1 - i\epsilon \sin \beta + 2\nu (\cos \beta - 1) \quad (3.89)$$

The  $R$  and  $\Phi$  are:

$$R^{CD} = \epsilon^2 \sin^2 \beta + (1 + 2\nu (\cos \beta - 1))^2 \quad (3.90)$$

and

$$\tan \Phi^{CD} = -\frac{\epsilon \sin \beta}{1 + 2\nu (\cos \beta - 1)} \quad (3.91)$$

Figure 3.4 shows the transfer function in the complex plane. Whenever the transfer function exceeds the unit circle, the mode grows and the algorithm is unstable. Each of the ellipses shows the complete set of modes (wavelengths) for a given  $\epsilon$  and  $\nu$ . None of the points along the ellipse is allowed to be beyond the unit circle, because if any point exceeds the unit circle, then there exists an unstable mode that will grow exponentially and, sooner or later, will dominate the solution.

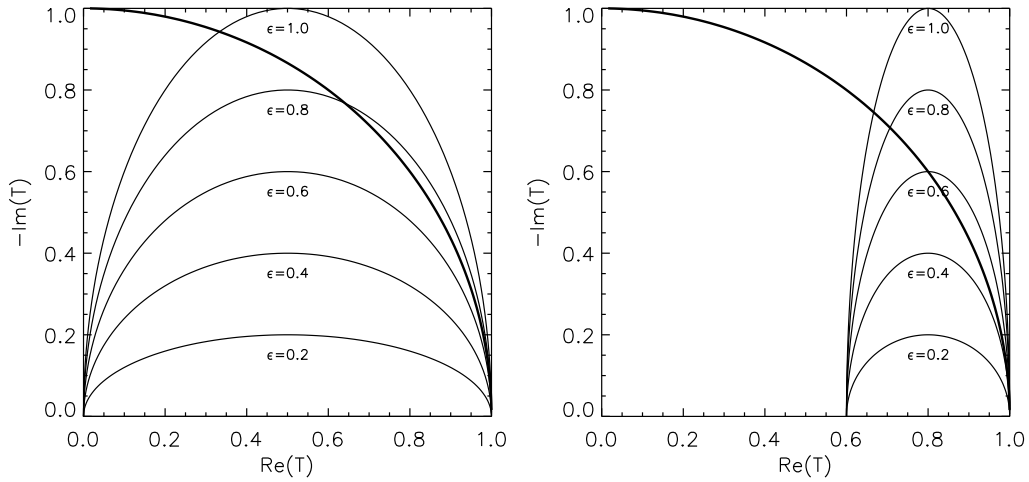
From these figures one can guess that whether an ellipse exceeds the unit circle or not is already decided for very small  $\beta$  (i.e. very close to  $T = 1$ ). So if we expand Eq. (3.90) in  $\beta$  we obtain

$$R^{CD} \simeq 1 + \beta^2 (\epsilon^2 - 2\nu) + \mathcal{O}(\beta^3) \quad (3.92)$$

We see that for the centered-differencing-and-diffusion algorithm to be stable one must have

$$\nu \geq \epsilon^2/2 \quad (3.93)$$

→ **Exercise:** Argue what is the largest allowed  $\epsilon$  for which a  $\nu$  can be found for which all modes are stable.



**Figure 3.4.** The complex transfer function for the centered differencing advection scheme with added artificial viscosity (imaginary axis is flipped). The ellipses are curves of constant advection parameter  $\epsilon = u\Delta t/\Delta x$  (marked on the curves) and varying  $\beta$  (i.e. varying wavelength). The thick line is the unit circle. Left:  $\nu = 0.25$ , right:  $\nu = 0.1$ . Whenever the transfer function exceeds the unit circle, the algorithm is unstable since the mode grows.

→ **Exercise:** Analyze the upstream differencing scheme of Section 3.3.2 with the above method, show that it is stable, and derive whether (and if so, how much) this algorithm is more diffusive than strictly necessary for stability.

Clearly for  $\nu \geq \epsilon^2/2$  the scheme is stable, but does it produce reasonable results? Let us first study the phase  $\Phi^{CD}$  and compare to the expected value. The expected value is:

$$\Phi = -uk\Delta t = -\epsilon\beta \quad (3.94)$$

Comparing this to Eq. (3.91) we see that to first order the phase is OK, at least for small  $\beta$ . The error appears when  $\beta$  gets non-small, i.e. for short wavelength modes. This is not a surprise since short wavelength modes are clearly badly sampled by a numerical scheme.

What about damping? From the phase diagram one can see that if we choose  $\nu$  larger than strictly required, then the ellipse moves more to the left, and thereby generally toward smaller  $R^{CD}$ , i.e. strong damping. For small  $\epsilon$  (assuming we choose  $\nu = \epsilon^2/2$ ) we see that the ellipses flatten. This means that short-wavelength (i.e. large  $\beta$ ) modes are strongly damped, and even longer wavelength modes (the ones that we should be able to model properly) are damped. Since this damping is an exponential process (happening after each time step again, and thereby creating a cumulative effect), even a damping of 10% per time step will result in a damping of a factor of  $10^{-3}$  after only 65 time steps. Clearly such a damping, even if it seems not too large for an individual time step, leads to totally smeared out results in a short time. This is not what we would like to have. The solution should, ideally, move precisely along the unit circle, but realistically this is never attainable. The thing we should aim for is an algorithm that comes as close as possible to the unit circle, and has an as small as possible error in the phase.

### 3.8.3 Lax-Wendroff scheme

If we choose, in the above algorithm, precisely enough artificial viscosity to keep the algorithm stable, i.e.

$$\nu = \frac{1}{2}\epsilon^2 \quad (3.95)$$

then the algorithm is called the *Lax-Wendroff* algorithm. The update for the Lax-Wendroff scheme is evidently:

$$q_i^{n+1} = q_i^n - \frac{\epsilon}{2}(q_{i+1}^n - q_{i-1}^n) + \frac{\epsilon^2}{2}(q_{i+1}^n - 2q_i^n + q_{i-1}^n) \quad (3.96)$$

Interestingly, the Lax-Wendroff scheme also has another interpretation: that of a *predictor-corrector method*. In this method we first calculate the  $q_{i-1/2}^{n+1/2}$  and  $q_{i+1/2}^{n+1/2}$ , i.e. the mid-point fluxes at half-time:

$$q_{i-1/2}^{n+1/2} = \frac{1}{2}(q_i^n + q_{i-1}^n) + \frac{1}{2}\epsilon(q_{i-1}^n - q_i^n) \quad (3.97)$$

$$q_{i+1/2}^{n+1/2} = \frac{1}{2}(q_i^n + q_{i+1}^n) + \frac{1}{2}\epsilon(q_i^n - q_{i+1}^n) \quad (3.98)$$

Then we write for the desired  $q_i^{n+1}$ :

$$q_i^{n+1} = q_i^n + \epsilon(q_{i-1/2}^{n+1/2} - q_{i+1/2}^{n+1/2}) \quad (3.99)$$

Working this out will directly yield Eq. (3.96).

### 3.9 Phase errors and Godunov's Theorem

The Lax-Wendroff scheme we derived in the previous section is the prototype of second order advection algorithms. There are many more types of second order algorithms, and in the next chapter we will encounter them in a natural way when we discuss piecewise linear advection schemes. But most of the qualitative mathematical properties of second order linear schemes in general can be demonstrated with the Lax-Wendroff scheme.

One very important feature of second order schemes is the nature of their phase errors. Using the following Taylor expansions

$$\text{atan}x = x - \frac{1}{3}x^3 + O(x^4) \quad \cos x = 1 - \frac{1}{2}x^2 + O(x^4) \quad \sin x = x - \frac{1}{6}x^3 + O(x^4) \quad (3.100)$$

we can write the difference  $\Phi^{CD} - \Phi_e$ :

$$\begin{aligned} \delta\Phi^{CD} \equiv \Phi^{CD} - \Phi_e &= -\text{atan} \left[ \frac{\epsilon \sin \beta}{1 + 2\nu(\cos \beta - 1)} \right] + \epsilon\beta \\ &\simeq \epsilon\beta^3 \left[ \frac{1}{6} - \nu + \frac{1}{3}\epsilon^2 \right] \end{aligned} \quad (3.101)$$

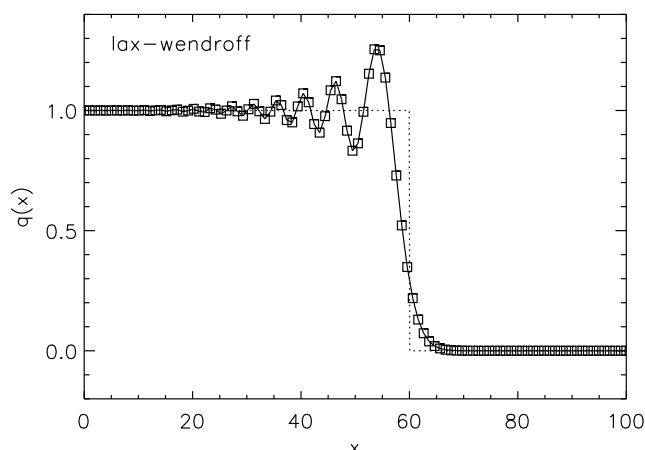
A phase error can be associated with a *lag* in space: the wave apparently moves too fast or too slow. The lag corresponding to the particular phase error is:  $\delta x = \delta\Phi/k \propto \delta\Phi/\beta$ . The lag per unit time is then  $d(\delta x)/dt = \delta x/\Delta t \propto \delta\Phi/(\beta\epsilon)$ . So this becomes

$$\frac{d(\delta x)}{dt} \propto \beta^2 \left[ \frac{1}{6} - \nu + \frac{1}{3}\epsilon^2 \right] \quad (3.102)$$

One sees that the spatial lag is clearly dramatically rising when  $\beta \rightarrow 1$ , i.e. for wavelength that approach the grid size. In other words: the shortest wavelength modes have the largest error in their propagation velocity.

Now suppose we wish to advect a block function:

$$q(x) = \begin{cases} 1 & \text{for } x < x_0 \\ 0 & \text{for } x > x_0 \end{cases} \quad (3.103)$$



**Figure 3.5.** The spurious oscillations which spontaneously arise if the Lax-Wendroff method is applied to a jump solution.

Such a block function can be regarded as a Fourier sum of waves of varying wavelength. In such a sharp jump all wavelengths are represented. If we now advect this function over the grid, then we see that the shortest wavelength components will lag most behind while the larger wavelength modes do relatively fine.

For the upwind scheme ( $\nu = 0$ ) these phase errors simply smear out any jumps like jelly. For the Lax-Wendroff scheme these phase errors produce spurious oscillations (see Fig. 3.5). Note that in contrast to the unstable centered-difference scheme these oscillations remain bound and cause no major damage. Yet, they are clearly unwanted.

Can one find a second order scheme that does not have these spurious oscillations? There are methods that at least produce less strong oscillations, such as the Fromm method (see Chapter 4). But there is a theorem, due to Sergei K. Godunov, that states that *any linear algorithm for solving partial differential equations, with the property of not producing new extrema, can be at most first order*. This is known as *Godunov Theorem*. There is therefore no hope of finding a linear second order accurate scheme that does not produce these unwanted wiggles. In Chapter 4 we will discuss *non-linear* schemes that combine the higher order accuracy and the prevention of producing unwanted oscillations.

### 3.10 Computer implementation (adv-1): grids and arrays

So far everything in this chapter was theoretical. Now let us see how we can put things in practice. Everything here will be explained in terms of the computer programming language IDL *Interactive Data Language* (or its public domain clone “GDL”, *Gnu Data Language*). This language starts counting array elements always from 0 (like C, but unlike Fortran, although in Fortran one can set the range by hand to start from 0).

Let us write a program for testing the upwind algorithm and let us call it ‘advection.pro’. Both the  $x$ -grid and the quantity  $q$  will now be arrays in the computer:

```

nx    = 100
x     = dblarr(nx)
q     = dblarr(nx)
qnew  = dblarr(nx)

```

We can produce a regular grid in  $x$  in the following way

```
dx = 1.d0          ; Set grid spacing
for i=0,nx-1 do x[i] = i*dx
```

In IDL this can be done even easier with the command `dindgen`, but let us ignore this for the moment. Now let us put some function on the grid:

```
for i=0,nx-1 do if x[i] lt 30. then q[i]=1.d0 else q[i]=0.d0
```

Now let us define a velocity, a time step and a final time:

```
u      = 1.d0
dt     = 2d-1
tend   = 30.d0
```

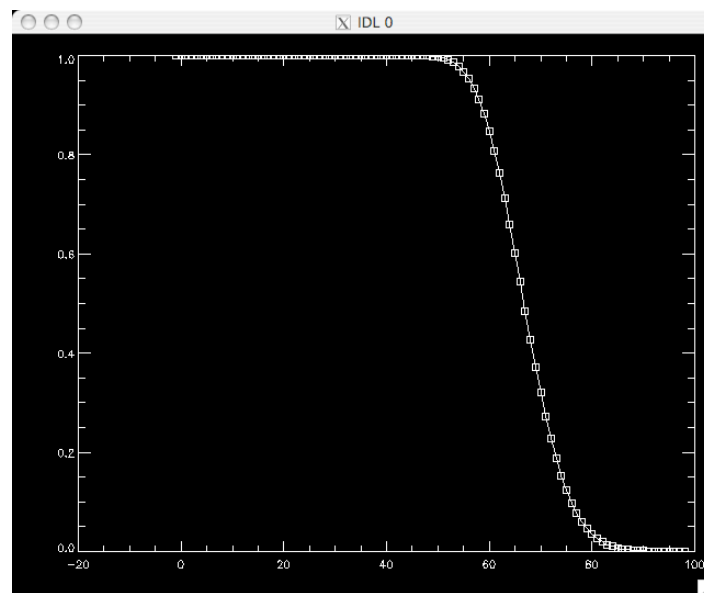
As a left boundary condition (since  $u > 0$ ) we can take

```
qlleft = q[0]
```

Now the simple upwind algorithm can be done for grid point  $i=1, nx-1$ :

```
time = 0.d0
while time lt tend do begin
    ;;
    ;; Check if end time will not be exceeded
    ;;
    if time + dt lt tend then begin
        dtused = dt
    endif else begin
        dtused = tend-time
    endelse
    ;;
    ;; Do the advection
    ;;
    for i=1,nx-1 do begin
        qnew[i] = q[i] - u * ( q[i] - q[i-1] ) * dtused / dx
    endfor
    ;;
    ;; Copy qnew back to q
    ;;
    for i=1,nx-1 do begin
        q[i] = qnew[i]
    endfor
    ;;
    ;; Set the boundary condition at left side (because u>0)
    ;; (Note: this is not explicitly necessary since we
    ;; didn't touch q[0])
    ;;
    q[0] = qlleft
```





**Figure 3.6.** The plot resulting from the `advect.pro` program of Section 3.10.

```
;;
;; Update time
;;
time = time + dtused
endwhile
```

Now we can plot the result

```
plot,x,q,psym=-6
```

At the end of the program we must put an

```
end
```

Now we can go into IDL and type `.r advection.pro` and we should get a plot on the screen (Fig. 3.6).



# Chapter 4

## Advection algorithms II. Flux conservation, subgrid models and flux limiters

In this chapter we will focus on the flux-conserving formalism of advection algorithms, and we shall discuss techniques to have higher order accuracy without the risk of getting spurious oscillations in the solutions. This chapter follows in many respects the book of LeVeque, but at times with slightly modified notation, to remain consistent with the rest of the lecture notes.

### 4.1 Flux conserving formulation: the principles

So far we have focused mostly on how to move a function over a grid. If we want to apply this to the equations of hydrodynamics, then we need to keep in mind that these equations are in fact *conservation equations*. This property is of fundamental importance. If we make systematic errors in the conservation of conserved quantities, then we acquire behavior of our solutions that is clearly unphysical. Moreover, the fact that we have conserved quantities in our system is also a big help: if we can formulate our algorithm to identically conserve these quantities numerically (down to machine precision of 16 digits in doubleprecision), we have eliminated at least a few of possible things that can go wrong.

To numerically conserve conserved quantities is not just a matter of precision. Even with a reasonably precise algorithm, one could make small but *systematic* errors each time step. Suppose the relative error in the total energy is  $10^{-4}$  per time step, but we need to perform  $10^5$  time steps to reach the end-time of the simulation, then if the error is always in one direction, the error has accumulated to a factor of 10 by the end of the simulation. Such results are clearly useless as we have created energy from nothing. Therefore it is a highly desirable property of a hydrodynamical code (or any solver of hyperbolic equations) if it is formulated in a *numerically flux conserving form*.

Of course, even the best flux-conserving formulation is not perfectly flux conserving, as numerical errors are always made when performing floating-point operations. However, the IEEE math routines used by any programming language (or built into any chip) are such that such errors are always only at the last digit (8th digit for floating points, 16th digit for doubleprecision floats), and generally such errors are not systematic, so that they do not propagate to lower digits quickly.

### 4.1.1 Cells and fluxes

The idea of flux conserving schemes is to create *cells* out of the grid, instead of just sampling grid points. The grid points now become *cell centers* located at  $x_i$ , and we now have *cell interfaces* or *cell walls* located at:

$$x_{i+1/2} = \frac{1}{2}(x_i + x_{i+1}) \quad (4.1)$$

where the  $+1/2$  is just a notational trick to denote “in between  $i$  and  $i + 1$ ”. Note that if we have a grid of  $N$  cell centers we have  $N - 1$  cell interfaces. But it is also important to note that the cells  $i = 1$  (left boundary) and  $i = N$  (right boundary) also have to have a cell wall at resp.  $i = 1/2$  and  $i = N + 1/2$ . Eq. (4.1) does not clearly define the location of these boundary cell walls. We will take them to be at:

$$x_{1/2} = x_1 - \frac{1}{2}(x_2 - x_1) \quad (4.2)$$

$$x_{N+1/2} = x_N + \frac{1}{2}(x_N - x_{N-1}) \quad (4.3)$$

This then makes a total of  $N + 1$  cell interfaces. Let us define, as before, the cell spacing to be  $\Delta x_{i+1/2} = x_{i+1} - x_i$  and  $\Delta x_i = x_{i+1/2} - x_{i-1/2}$ . For constant spacing we can write  $\Delta x$  for both quantities.

If we now assume this 1-D problem to be in fact a 3-D flow through a pipe with cross-sectional surface  $S$ , then the volume of each cell is:  $V = \Delta x S$ . In real 3-D hydrodynamic problems the cell volumes will be something like  $V = \Delta x \Delta y \Delta z$ , and the surface is  $S = \Delta y \Delta z$ .

We can now regard the cells as volumes containing our conserved quantity  $q_i^n$ . The total content of each cell is  $Q_i^n = q_i^n V$ . The fact that the equation to solve is a conservation equation means that  $Q_i^n$  can only be reduced by moving some of it to neighboring cells. Conversely,  $Q_i^n$  can only be increased by moving something from neighboring cells into cell  $i$ . Therefore the change in  $Q_i$  can be seen as in- and out-flow through the cell interfaces. We can formulate at each cell interface a *flux*  $f_{i+1/2}$  and say that

$$\frac{dQ_i}{dt} = (f_{i-1/2} - f_{i+1/2}) \cdot S \quad (4.4)$$

In discrete form:

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} = (f_{i-1/2} - f_{i+1/2}) \cdot S \quad (4.5)$$

Stricly speak we must do this at half-time:

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} = (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \cdot S \quad (4.6)$$

Overall the sum of  $Q_i^n$  is now naturally constant, except for in- or outflow at  $x = x_{1/2}$  or  $x = x_{N+1/2}$ :

$$\frac{d}{dt} \left( \sum_{i=1}^N Q_i \right) = (f_{1/2} - f_{N+1/2}) S \quad (4.7)$$

If we make use of the expressions  $Q_i^n = q_i^n V$  and  $V = S \Delta x$  (assume constant grid spacing for now) then Eq. (4.6) becomes:

$$\frac{q_i^{n+1} - q_i^n}{\Delta t} = \frac{f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}}{\Delta x} \quad (4.8)$$

or in explicit form:

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{\Delta x} (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \quad (4.9)$$

We have now formulated our advection problem in flux-conserving form.

#### 4.1.2 Non-constant grid spacing

In flux conserving schemes it is very simple to handle non-regular grids. Suppose that we have a grid spacing in  $x$  as follows:  $x_i = \{0., 0.1, 0.3, 0.6, 1.0, 1.5, \dots\}$ , then the  $\Delta x$  is not anymore a globally constant value. Moreover, it now becomes an interesting question where we define the cell interfaces  $x_{i+1/2}$  to be. In general there is no unique recipe to define the  $x_{i+1/2}$  corresponding to the  $x_i$  for some non-regular grid spacing. As long as  $x_{i+1/2}$  somewhere in between  $x_i$  and  $x_{i+1}$  in a sensible way, then it should be fine. The algorithm then becomes:

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{x_{i+1/2} - x_{i-1/2}} (f_{i-1/2}^{n+1/2} - f_{i+1/2}^{n+1/2}) \quad (4.10)$$

with the fluxes  $f_{i\pm 1/2}^{n+1/2}$  given by the particular algorithm used, for instance the donor-cell algorithm or one of the algorithms described in the sections to come.

*All flux conserving algorithms that are first-order in time are based on Eq.(4.10). All the cleverness of the advection algorithm is hidden in how  $f_{i\pm 1/2}^{n+1/2}$  is formulated in terms of the quantities  $q_{i\pm k}^n$ . Method which use this type of flux-conserving schemes are called Finite Volume Methods.*

#### 4.1.3 Non-constant advection velocity

In most practical purposes (in particular in the case of hydrodynamics) the advection velocity is not a global constant. If it were, then we would not have had to go through all the trouble in the last chapter and in this chapter to invent good advection algorithms, because the solution would have been analytically known beforehand. So the equation we wish to solve is now:

$$\partial_t q(x, t) + \partial_x (u(x)q(x, t)) = 0 \quad (4.11)$$

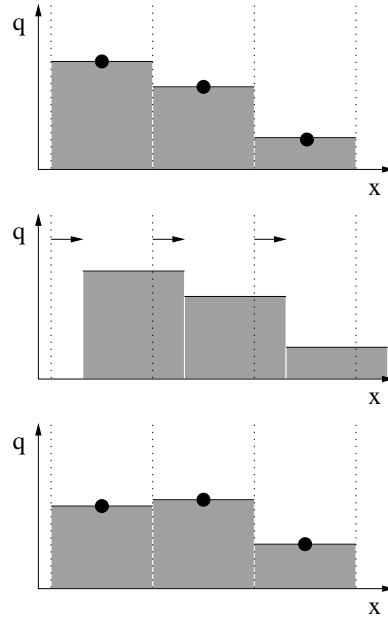
Note that now the distinction between a conservation equation and a non-conserving advection equation becomes apparent, and therefore it now becomes even more important to use flux conserving schemes if we wish to solve the conservation equation.

So how do we handle non-constant  $u(x)$ ? First of all, we would still use the fundamental form of the update described in the previous subsection: Eq. (4.10). Exactly how we construct the fluxes  $f_{i\pm 1/2}^{n+1/2}$  from  $q_i^n$  and the velocity field  $u(x)$  depends very much on the algorithm that we use for the advection. We will discuss this in each of the examples below. In general, though, it requires the definition of the velocity  $u$  at the interfaces:  $u_{i+1/2}$  because the fluxes  $f_{i+1/2}$  are defined at the interfaces. Depending on the definition of the problem these velocities are already defined on these interfaces (i.e. the problem definition specifies  $u_{i+1/2}$ ), or they have to be computed by linear interpolation from cell-centered values  $u_i$  (i.e. if the problem definition specifies  $u_i$ ). Now the discrete algorithm is then Eq. (4.10) with

$$f_{i+1/2}^{n+1/2} = \tilde{q}_{i+1/2}^{n+1/2} u_{i+1/2} \quad (4.12)$$

where  $\tilde{q}_{i+1/2}^{n+1/2}$  is some estimate of what the average state is of  $q$  at the interface:

$$\tilde{q}_{i+1/2}^{n+1/2} = \text{estimate of } \frac{1}{t_{n+1} - t_n} \int_{t_n}^{t_{n+1}} q(x_{i+1/2}, t) dt \quad (4.13)$$



**Figure 4.1.** Illustration of the piecewise constant (donor-cell) advection algorithm.

Since we (by definition of the fact that we solve a numerical problem) do not know exactly what this average state is, it is the task of the algorithm to provide a recipe that estimates this as well as possible. In the two examples below we shall describe two such algorithms.

## 4.2 Donor-cell advection

The simplest flux conserving scheme is the donor-cell scheme. In this scheme the “average interface state” is simply:

$$\tilde{q}_{i+1/2}^{n+1/2} = \begin{cases} q_i^n & \text{for } u_{i+1/2} > 0 \\ q_{i+1}^n & \text{for } u_{i+1/2} < 0 \end{cases} \quad (4.14)$$

This means that the donor-cell interface flux is:

$$f_{i+1/2}^{n+1/2} = \begin{cases} u_{i+1/2} q_i^n & \text{for } u_{i+1/2} > 0 \\ u_{i+1/2} q_{i+1}^n & \text{for } u_{i+1/2} < 0 \end{cases} \quad (4.15)$$

The physical interpretation of this method is the following. One assumes that the density is constant within each cell. We then let the material flow through the cell interfaces, from left to right for  $u_{i+1/2} > 0$ . Since the density to the left of the cell interface is constant, and since the CFL condition makes sure that the flow is no further than 1 grid cell spacing at maximum, we know that for the whole time between time  $t_n$  and  $t_{n+1}$  the flux through the cell interface (which is  $\tilde{q}_{i+1/2}^{n+1/2} u_{i+1/2}$ ) is constant, and is equal to Eq. (4.15). Once the time step is finished, the state in each cell has the form of a step function (Fig. 4.1). To get back to the original sub-grid model we need to average the quantity  $q(x)$  out over each cell, to obtain the new  $q_i^{n+1}$ . This is what happens in the donor-cell algorithm.

This method is very strongly similar to the *upstream differencing* scheme of Section 3.3.2. The difference comes to light mainly when either the velocity  $u$  is space-dependent or the grid  $x_i$  is non-constantly spaced (see Section 4.1.2). The Donor-Cell algorithm is easily implemented but, as the upstream differencing method, it is very diffusive.

- **Exercise:** Quantify the difference between the donor-cell and upstream differencing schemes for non-constant grid spacing by writing the expressions for  $q^{n+1}$  in both cases and comparing them.

### 4.3 Piecewise linear schemes

The donor-cell algorithm is a simple algorithm based on the idea that at the beginning of each time step the state within each cell is constant throughout the cell. The state at the grid center is therefore identical to that in the entire cell. The idea that the state is constant in the cell is, however, just an assumption. One must make *some* assumption, because evidently we have not more information about the state in the cell than just the cell-center state. But one could also make another assumption. One could, for instance, assume that the state within each cell is a linear function of position. One says that the state is assumed to be *piecewise linear* (see Fig. 4.2). The assumption that it is a linear function within the cell is called a *subgrid model*: it is a model of what the state looks like at spatial scales smaller than the grid spacing. The procedure of creating a subgrid model based on a discrete value at the cell center is called *reconstruction*. The piecewise linear scheme we discuss here is therefore a scheme based on *piecewise linear reconstruction* of the function from the discrete values. Such a method, and higher order versions (such as the *piecewise parabolic reconstruction* discussed in a later chapter) are sometimes called *MUSCL* ("Monotonic Uwind-centered Scheme for Conservation Laws") schemes after the original method of this kind proposed by van Leer (1977).

Within each cell the state at the beginning of the time step is now given by

$$q(x, t = t_n) = q_i^n + \sigma_i^n(x - x_i) \quad \text{for } x_{i-1/2} < x < x_{i+1/2} \quad (4.16)$$

where  $\sigma_i^n$  is some slope, which we shall discuss below. We should assure that

$$x_i = \frac{1}{2}(x_{i-1/2} + x_{i+1/2}) \quad (4.17)$$

so that the state  $q_i^n$  is equal to the average of  $q(x, t = t_n)$  over the cell irrespective of the slope  $\sigma_i^n$ . This is necessary for flux conservation.

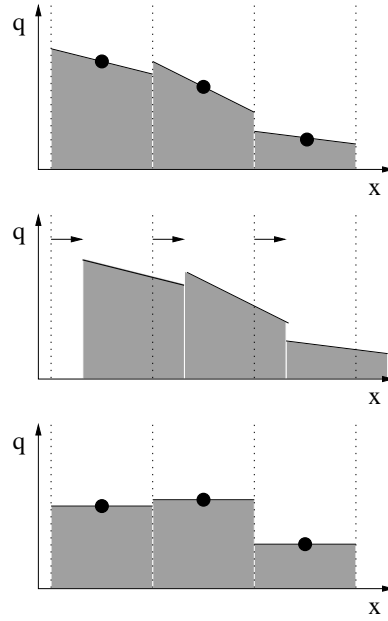
Let us, from here on, assume that the grid is equally spaced, i.e. that there is a global  $\Delta x$ . Let us also assume that  $u > 0$  is a constant.

At the interface the flux will now vary with time between  $t_n$  and  $t_{n+1}$ :

$$\begin{aligned} f_{i-1/2}(t) &= uq(x = x_{i-1/2}, t) \\ &= uq_{i-1}^n + u\sigma_{i-1}^n(x_{i-1/2} - x_{i-1} - u(t - t_n)) \\ &= uq_{i-1}^n + u\sigma_{i-1}^n \left( \frac{1}{2}\Delta x - u(t - t_n) \right) \end{aligned} \quad (4.18)$$

and similar for  $f_{i+1/2}(t)$ . The average flux over the time step  $\Delta t = (t_{n+1} - t_n)$  is then:

$$\begin{aligned} \langle f_{i-1/2}(t) \rangle_{t_n}^{t_{n+1}} &= \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} uq(x = x_{i-1/2}, t) dt \\ &= uq_{i-1}^n + \frac{1}{2}u\sigma_{i-1}^n (\Delta x - u\Delta t) \end{aligned} \quad (4.19)$$



**Figure 4.2.** Illustration of the piecewise linear advection algorithm. The slope is chosen according to Lax-Wendroff's method.

and similar for  $f_{i+1/2}(t)$ . The difference is:

$$\langle f_{i+1/2}(t) \rangle_{t_n}^{t_{n+1}} - \langle f_{i-1/2}(t) \rangle_{t_n}^{t_{n+1}} = u(q_i^n - q_{i-1}^n) + \frac{1}{2}u(\sigma_i^n - \sigma_{i-1}^n)(\Delta x - u\Delta t) \quad (4.20)$$

Using Eq. (4.10) we then obtain the update of the state after one time step:

$$q_i^{n+1} = q_i^n - \frac{u\Delta t}{\Delta x}(q_i^n - q_{i-1}^n) - \frac{u\Delta t}{\Delta x} \frac{1}{2}(\sigma_i^n - \sigma_{i-1}^n)(\Delta x - u\Delta t) \quad (4.21)$$

where we defined  $f_{i+1/2}^{n+1/2} \equiv \langle f_{i+1/2}(t) \rangle_{t_n}^{t_{n+1}}$ . Eq. (4.21) is the update of the state for a flux-conserving piecewise linear scheme (assuming that the grid spacing is constant). This is the higher-order version of the donor-cell algorithm. Note that it is identical to donor-cell if the slopes are chosen to be zero. Note also that since we chose the grid to be constantly spaced and the velocity to be globally constant, the algorithm is like an upwind scheme with a correction term.

The question is now: how shall we choose the slope  $\sigma_i^n$  of the linear function? The idea behind the piecewise linear scheme is that one uses the states at adjacent grid points in some reasonable way. There are three obvious methods:

$$\text{Centered slope: } \sigma_i^n = \frac{q_{i+1}^n - q_{i-1}^n}{2\Delta x} \quad (\text{Fromm's method}) \quad (4.22)$$

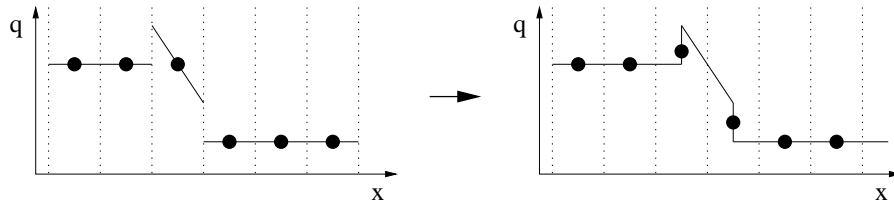
$$\text{Upwind slope: } \sigma_i^n = \frac{q_i^n - q_{i-1}^n}{\Delta x} \quad (\text{Beam-Warming method}) \quad (4.23)$$

$$\text{Downwind slope: } \sigma_i^n = \frac{q_{i+1}^n - q_i^n}{\Delta x} \quad (\text{Lax-Wendroff method}) \quad (4.24)$$

All these choices result in second-order accurate methods.

→ **Exercise:** Prove that the piecewise linear scheme with a downwind slope indeed produces the Lax-Wendroff scheme of Eq. (3.96) in Chapter 3.





**Figure 4.3.** Illustration of why higher order schemes, such as the piecewise linear Lax-Wendroff scheme shown here, produce oscillations near discontinuities.

If we now implement, for instance, Fromm's choice of slope into Eq. (4.21) then we obtain the following explicit update of the state (again, valid only for regular grid spacing and constant  $u$ ):

$$q_i^{n+1} = q_i^n - \frac{u\Delta t}{4\Delta x}(q_{i+1}^n + 3q_i^n - 5q_{i-1}^n + q_{i-2}^n) - \frac{u^2\Delta t^2}{4\Delta x^2}(q_{i+1}^n - q_i^n - q_{i-1}^n + q_{i-2}^n) \quad (4.25)$$

This is called Fromm's method of advection. One notices that the stencil of this scheme involves 4 points, and that these points are non-symmetric with respect to the updated cell. Only for the downwind slope we regain a 3-point stencil.

#### 4.4 Slope limiters: non-linear tools to prevent overshoots

As we have already seen in chapter 3 higher order schemes tend to produce oscillations near jumps. We have seen that this can mathematically be understood in terms of phase errors and dispersion. With the current piecewise linear geometric picture of the higher order methods we can now also gain a geometrical understanding of why such an overshoot occurs. This is shown in Fig. 4.3

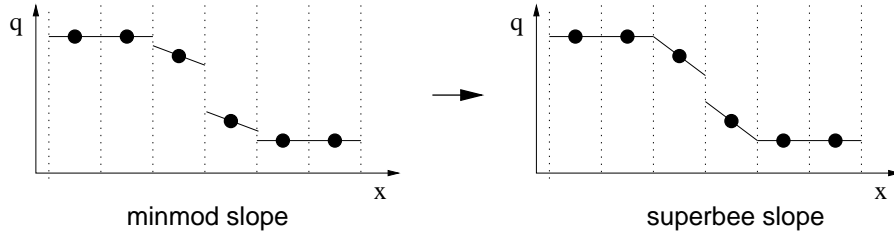
In this picture the overshoot happens because the piecewise linear elements can have overshoots. A successful method to prevent such overshoots is the use of *slope limiters*. These are non-linear conditions that modify the slope  $\sigma_i^n$  if, and only if, this is necessary to prevent overshoots. This means that, if this is applied to for instance the Lax-Wendroff method, the slope limiter keeps the second order nature of the Lax-Wendroff method in regions of smooth variation of  $q$  while it will intervene whenever an overshoot threatens to take place.

##### 4.4.1 The "Total Variation" (TV)

To measure oscillations in a solution we define the *Total Variation* (TV) of the discrete representation of  $q$  to be:

$$TV(q) = \sum_{i=1}^N |q_i - q_{i-1}| \quad (4.26)$$

where  $i = 1$  and  $i = N$  are the left and right boundaries of the numerical grid respectively. For a monotonically increasing function  $q$  the  $TV(q) = |q_1 - q_N|$ . If  $q_1$  and  $q_N$  are taken to be constant, then, as long as the function remains monotonic, the  $TV(q)$  is constant. However, if the values of  $q_i$  develop local minima or maxima, then the  $TV$  increases, by virtue of the absolute value in Eq. (4.26). The increase of  $TV$  is therefore a measure of the development of oscillations in a solution.



**Figure 4.4.** Illustration of the slope limiters minmod and superbee. Both clearly prevent overshoots in the subgrid models and hence are TVD.

A numerical scheme is said to be *total variation diminishing (TVD)* if:

$$TV(q^{n+1}) \leq TV(q^n) \quad (4.27)$$

Clearly such a scheme will not develop oscillations near a jump, because a jump is a monotonically in/decreasing function and a total variation diminishing scheme will not increase the  $TV$ , and hence in this case keep the  $TV$  identically constant. This appears to be a desirable property of the numerical scheme.

What about if we already start with a solution that has some local minima and maxima? In principle a total variation diminishing scheme could conserve the total variation (as long as it does not increase it). In practice, however, the local minima and maxima will be gradually smoothed out by such a scheme. Good schemes will do this only very weakly, and will only really diminish oscillations near sharp jumps.

#### 4.4.2 Slope limiters

So now let us try to formulate a recipe to limit the slope  $\sigma_i^n$  of the piecewise linear scheme to make sure that the resulting scheme is TVD. One obvious choice is to take the slope always zero. We then recover the donor-cell algorithm. This algorithm is TVD because the piecewise constant function has the same  $TV$  as the continuous function. In fact, as we have discussed in Section 3.9, all first order schemes are TVD.

For the piecewise linear scheme there are several methods to guarantee TVD. One method is the *minmod slope*:

$$\sigma_i^n = \text{minmod} \left( \frac{q_i^n - q_{i-1}^n}{\delta x}, \frac{q_{i+1}^n - q_i^n}{\delta x} \right) \quad (4.28)$$

where

$$\text{minmod}(a, b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } ab > 0 \\ b & \text{if } |a| > |b| \text{ and } ab > 0 \\ 0 & \text{if } ab \leq 0 \end{cases} \quad (4.29)$$

This method chooses the smallest of the two slopes, provided they both have the same sign, and chooses 0 otherwise. The minmod slope is illustrated in Fig. 4.4-left.

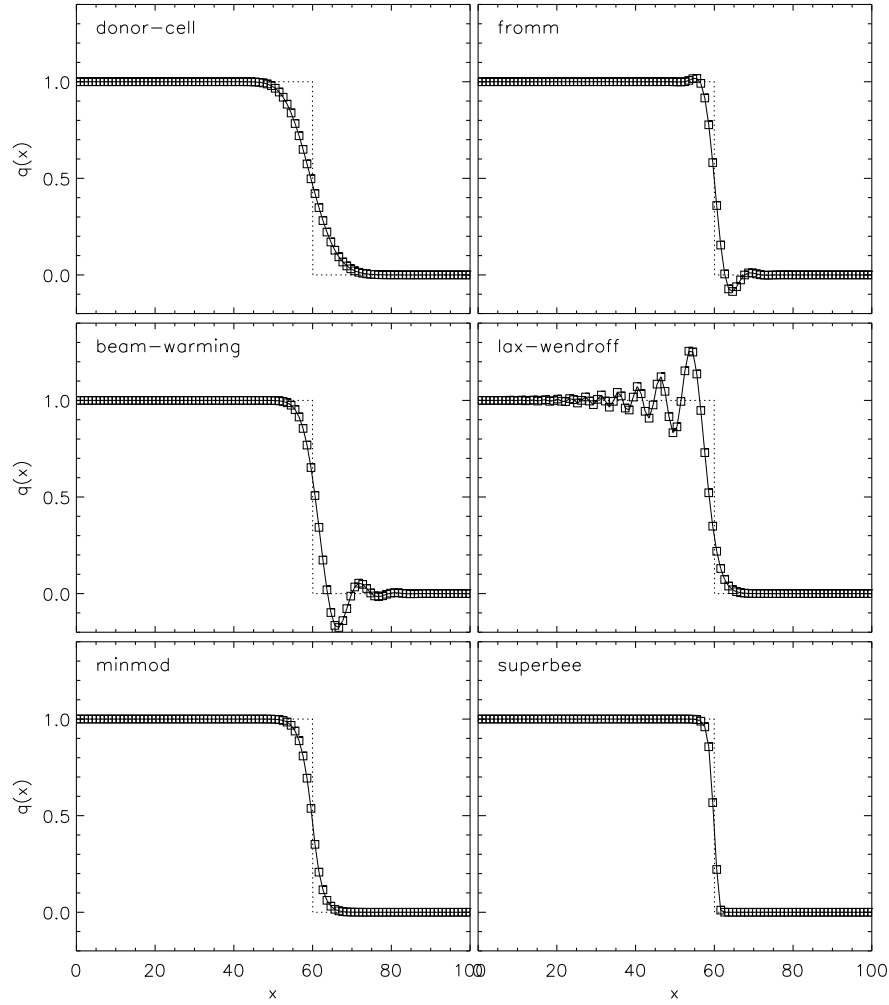
Another efficient slope limiter is the *superbee* slope limiter introduced by Phil Roe:

$$\sigma_i^n = \text{maxmod}(\sigma_i^{(1)}, \sigma_i^{(2)}) \quad (4.30)$$

with

$$\sigma_i^{(1)} = \text{minmod} \left( \frac{q_{i+1}^n - q_i^n}{\delta x}, 2 \frac{q_i^n - q_{i-1}^n}{\delta x} \right) \quad (4.31)$$

$$\sigma_i^{(2)} = \text{minmod} \left( 2 \frac{q_{i+1}^n - q_i^n}{\delta x}, \frac{q_i^n - q_{i-1}^n}{\delta x} \right) \quad (4.32)$$



**Figure 4.5.** Advection with the piecewise linear advection algorithm with 6 different choices of the slope. Results are shown of the advection of a step function over a grid of 100 points with grid spacing  $\Delta x = 1$ , after 300 time steps with  $\Delta t = 0.1$ .

The superbee slope is shown in Fig. 4.4-right.

In Fig. 4.5 the results are shown for all the piecewise linear advection algorithms discussed here.

## 4.5 Flux limiters

The concept of slope limiters is very closely related to the concept of *flux limiters*. For flux-conserving schemes the concept of flux-limiter is more appropriate and can be better implemented. Let us take another look at the time-averaged flux through a cell interface (Eq. 4.19). For arbitrary sign of  $u$  this becomes:

$$f_{i-1/2}^{n+1/2} = \begin{cases} uq_{i-1}^n + \frac{1}{2}u_{i-1/2}\sigma_{i-1}^n (\Delta x - u_{i-1/2}\Delta t) & \text{if } u_{i-1/2} \geq 0 \\ uq_i^n - \frac{1}{2}u_{i-1/2}\sigma_i^n (\Delta x + u_{i-1/2}\Delta t) & \text{if } u_{i-1/2} \leq 0 \end{cases} \quad (4.33)$$

where we dropped the  $\langle \rangle_{t_n}^{t_{n+1}}$  for notational convenience and put instead a  $^{n+1/2}$  to denote the fact that the flux average is similar to taking the flux at half-time<sup>1</sup>. We have also now allowed for a varying velocity  $u$ . The velocity  $u$  is, in this formulation, defined on the cell interfaces, while the value of  $q$  is always defined at the cell center. If we, for convenience, introduce

$$\theta_{i-1/2} \equiv \theta(u_{i-1/2}) = \begin{cases} +1 & \text{for } u_{i-1/2} \geq 0 \\ -1 & \text{for } u_{i-1/2} \leq 0 \end{cases} \quad (4.34)$$

then we can write

$$\begin{aligned} f_{i-1/2}^{n+1/2} = & \frac{1}{2} u_{i-1/2} \left[ (1 + \theta_{i-1/2}) q_{i-1}^n + (1 - \theta_{i-1/2}) q_i^n \right] + \\ & \frac{1}{4} |u_{i-1/2}| \left( 1 - \left| \frac{u_{i-1/2} \Delta t}{\Delta x} \right| \right) \Delta x \left[ (1 + \theta_{i-1/2}) \sigma_{i-1}^n + (1 - \theta_{i-1/2}) \sigma_i^n \right] \end{aligned} \quad (4.35)$$

This expression is identical to Eq. (4.33). It is just written in a single formula using the flip-flop function  $\theta_{i-1/2}$  which is  $+1$  for positive advection velocity and  $-1$  for negative advection velocity. This expression is valid for any regularly or non-regularly spaced grid, and for any constant or varying advection velocity. It is therefore the kind of expression we need for numerical hydrodynamics in which non-regular grids and, more importantly, varying velocity fields are common.

In Eq. (4.35) it is now possible to replace

$$\frac{1}{2} \Delta x \left[ (1 + \theta_{i-1/2}) \sigma_{i-1}^n + (1 - \theta_{i-1/2}) \sigma_i^n \right] \rightarrow \phi(r_{i-1/2}^n) (q_i^n - q_{i-1}^n) \quad (4.36)$$

where we introduced a *flux limiter*  $\phi(r_{i-1/2}^n)$  where  $r_{i-1/2}^n$  is defined as:

$$r_{i-1/2}^n = \begin{cases} \frac{q_{i-1}^n - q_{i-2}^n}{q_i^n - q_{i-1}^n} & \text{for } u_{i-1/2} \geq 0 \\ \frac{q_{i+1}^n - q_i^n}{q_i^n - q_{i-1}^n} & \text{for } u_{i-1/2} \leq 0 \end{cases} \quad (4.37)$$

We shall discuss expressions for  $\phi(r_{i-1/2}^n)$  in a minute, but first let us see how Eq. (4.35) is modified by this replacement:

$$\begin{aligned} f_{i-1/2}^{n+1/2} = & \frac{1}{2} u_{i-1/2} \left[ (1 + \theta_{i-1/2}) q_{i-1}^n + (1 - \theta_{i-1/2}) q_i^n \right] + \\ & \frac{1}{2} |u_{i-1/2}| \left( 1 - \left| \frac{u_{i-1/2} \Delta t}{\Delta x} \right| \right) \phi(r_{i-1/2}^n) (q_i^n - q_{i-1}^n) \end{aligned} \quad (4.38)$$

We see that this expression of the flux is still the donor-cell flux (first line) plus some correction term (second line).

We can now verify that with particular choices of the flux limiter function  $\phi(r)$  we regain the various recipes of Section 4.4:

$$\begin{aligned} \text{donor-cell :} & \quad \phi(r) = 0 \\ \text{Lax-Wendroff :} & \quad \phi(r) = 1 \\ \text{Beam-Warming :} & \quad \phi(r) = r \\ \text{Fromm :} & \quad \phi(r) = \frac{1}{2}(1 + r) \end{aligned} \quad (4.39)$$

---

<sup>1</sup>Note, however, that this does not mean that we have a Crank-Nicholson scheme here! This is not an implicit scheme!

which are the linear schemes and

$$\begin{aligned} \text{minmod} : \quad & \phi(r) = \minmod(1, r) \\ \text{superbee} : \quad & \phi(r) = \max(0, \min(1, 2r), \min(2, r)) \end{aligned} \quad (4.40)$$

which are the high-resolution non-linear schemes. Now that we are at it, there are also many other non-linear flux limiter schemes. Two examples are:

$$\begin{aligned} \text{MC} : \quad & \phi(r) = \max(0, \min((1 + r)/2, 2, 2r)) \\ \text{van Leer} : \quad & \phi(r) = (r + |r|)/(1 + |r|) \end{aligned} \quad (4.41)$$

where MC stands for *monotonized central-difference limiter*.

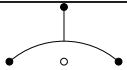
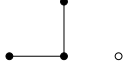
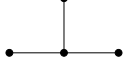
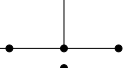
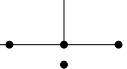
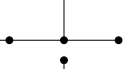
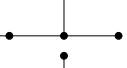
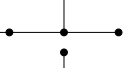

A flux limiter has the following effect:

- For smooth parts of a solution it will do second order accurate flux-conserved advection.
- For regions near a jump or a very sharp and sudden gradient it will switch to first order donor-cell (i.e. upwind) flux-conserved advection.

Flux limiters therefore make sure that one gets the best of both second order and first order methods. Flux limiters will turn out to play a major role for *shock capturing hydrodynamics codes*, in which shocks need to be kept tight and not smear out over too many grid cells. For the modeling of supersonic flows with shock waves (such as those often found in astrophysics) they are a highly useful tool. However, one should keep in mind that they may artificially steepen gradients that may not actually be meant to be steepened. They therefore have to be used with care.

## 4.6 Overview of algorithms

As a final remark we give here a table of the properties of the various algorithms:

Name	Order	Lin?	Stable?	TVD?	Stencil
Two-point symmetric	1	lin	-	-	
Upwind / Donor-cell	1	lin	+	+	
Lax-Wendroff	2	lin	+	-	
Beam-warming	2	lin	+	-	
Fromm	2	lin	+	-	
Minmod	2/1	non-lin	+	+	
Superbee	2/1	non-lin	+	+	
MC	2/1	non-lin	+	+	
van Leer	2/1	non-lin	+	+	

## 4.7 Computer implementation (adv-2): cell interfaces and ghost cells

The computer implementation is now a bit more complicated than in chapter 3 because we now have cells, and cell-interfaces. In this section we will see how to deal with this in a proper way. Also we will introduce the concept of *ghost cells* which is a simple trick to implement boundary conditions in a very easy way, even periodic boundary conditions. It will help us greatly when we will do the hydrodynamics in the later chapters.

Let us, as in the previous section, set up the problem in IDL/GDL. First the grid:

```
nx      = 100
x       = dblarr(nx+2)
xi      = dblarr(nx+3)
```

Here we have 100 regular grid points with a ghost cell on each side. We have therefore 102 cells with 103 cell walls. We now also define the array for  $q$ , for the temporary array  $q_{new}$  and for the flux  $f$ :

```
q       = dblarr(nx+2)
qnew    = dblarr(nx+2)
f       = dblarr(nx+3)
```

Now we set up the grid

```
dx      = 1.d0           ; Set grid spacing
for i=0,nx+1 do x[i] = (i-1.d0) * dx
for i=1,nx+1 do xi[i] = 0.5 * ( x[i] + x[i-1] )
xi[0]   = x[0] - 0.5 * ( x[1] - x[0] )
xi[nx+2] = x[nx+1] + 0.5 * ( x[nx+1] - x[nx] )
```

where we made the grid regular with grid spacing  $dx$ . Note that the cell walls are indexed such that wall  $i$  is left of cell  $i$  (see Fig. 4.6). Now we set up the initial condition and the advection velocity array

```
for i=0,nx-1 do if x[i] lt 30. then q[i]=1.d0 else q[i]=0.d0
u          = 1.d0 + dblarr(nx+3)
```

Note that the advection velocity array is also defined on the cell walls. Now the main part of the code is:

```
dt      = 2d-1
tend    = 70.d0
time    = 0.d0
while time lt tend do begin
    ;;
    ;; Check if end time will not be exceeded
    ;;
    if time + dt lt tend then begin
        dtused = dt
    endif else begin
        dtused = tend-time
    endelse
```

---

```

;;
;; Impose cyclic (=periodic) boundary conditions
;; using the ghost cells
;;
q[0]      = q[nx]
q[nx+1]   = q[1]
;;
;; Make the flux
;;
for i=1,nx+1 do begin
  if u[i] gt 0 then begin
    f[i] = q[i-1] * u[i]
  endif else begin
    f[i] = q[i] * u[i]
  endelse
endfor
;;
;; Do the advection
;;
for i=1,nx do begin
  qnew[i] = q[i] - dtused * ( f[i+1] - f[i] ) / $
                        ( xi[i+1]- xi[i] )

endfor
;;
;; Copy back
;;
for i=1,nx do begin
  q[i] = qnew[i]
endfor
;;
;; Update time
;;
time = time + dtused
endwhile

```

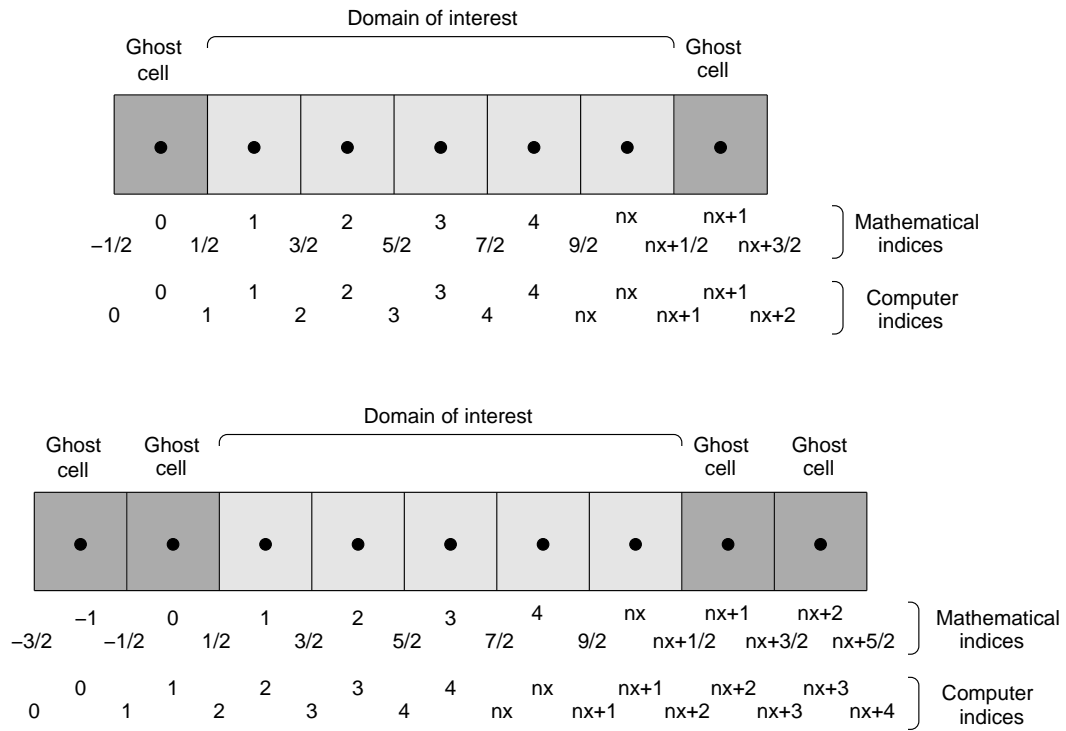
Here we have imposed cyclic (periodic) boundary conditions, just as an example how useful ghost cells can be. We have also made sure that the algorithm works for any  $u(x)$ , also if it is a function of  $x$  and if it changes sign. If we now end the program with

```

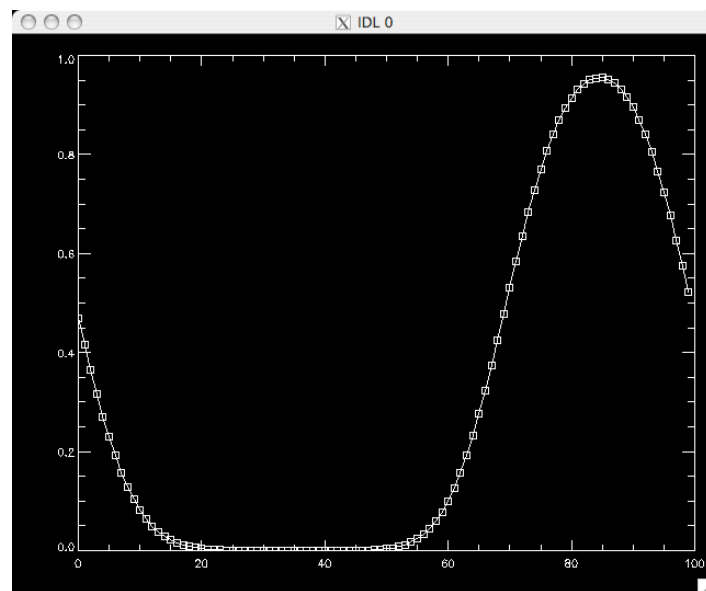
plot,x[1:nx],q[1:nx],psym=-6
end

```

then we will see a figure on the screen like Fig. 4.7. Note that the periodic boundary conditions have the effect that the diffusive smearing now affects the front- *and* the back-side of the block function, in contrast to the case of Fig. 3.6. Also note that in the present program we went all the way to  $t = 70$  instead of  $t = 30$  of Fig. 3.6. The smearing has now almost entirely destroyed the block function.



**Figure 4.6.** The grid of a flux-conserving advection program including ghost cells, for algorithms with a three-point stencil (upper panel) and for algorithms with a four- or five-point stencil (lower panel).



**Figure 4.7.** The plot resulting from the donorcell.pro program of Section 4.7. Note that in contrast to Fig. 3.6 we now have continued the simulation until time  $t = 70$  and we have imposed periodic boundary conditions. Hence the difference to Fig. 3.6.



# Chapter 5

## Classic hydrodynamics solvers

### 5.1 Basic approach

Armed with the knowledge on how to numerically advect functions over a grid, we can now start to build our first numerical hydrodynamics solvers. The knowledge of the structure of hyperbolic equations described in chapter 2 would lead us to want to diagonalize the system of equations locally and then apply the numerical advection schemes. This is in fact precisely what *approximate Riemann solvers* do (see Chapter ??). But historically the equations were approached in a somewhat different manner, and those methods also have some advantages over the Riemann solvers of later chapters and for that reason are still heavily in use today. The main term in the Euler equations that tends to mix the eigenvectors of the Jacobian (and thus makes the problem not globally diagonalizable) is the  $\nabla P$  term. If we extract this term from the left-hand-side of the momentum equation and put it to the right-hand-side, then the equation becomes an advection equation of momentum with advection velocity  $u$ . A similar trick can be done with the energy equation. The result is:

$$\partial_t \rho + \nabla \cdot (\rho \vec{u}) = 0 \quad (5.1)$$

$$\partial_t (\rho \vec{u}) + \nabla \cdot (\rho \vec{u} \vec{u}) = -\nabla P \quad (5.2)$$

$$\partial_t (\rho e_{\text{tot}}) + \nabla \cdot (\rho e_{\text{tot}} \vec{u}) = -\nabla \cdot (P \vec{u}) \quad (5.3)$$

The right-hand-side can be regarded as a source term. If we now regard  $\vec{u}$  at any time step as a given quantity of the previous time step, then the left-hand-side of the above set of equations is already in *globally diagonal form* with three equal eigenvalues:  $\lambda_{1,2,3} = u$  for the  $x$ -direction and  $\lambda_{1,2,3} = v$  for the  $y$ -direction and  $\lambda_{1,2,3} = w$  for the  $z$ -direction. The advection now has to be done in  $\rho$ ,  $\rho u$ ,  $\rho v$ ,  $\rho w$ , and  $\rho e$ , all with the same advection velocity: these are five fully independent advection problems. This is what is sometimes called *consistent transport*.

There are a number of advantages of this approach over the full characteristic approach we'll cover in Chapter ??. One of the main advantages of this approach is that the advection problem is much simpler than in the case when we diagonalize the full Jacobian with gas pressure. In fact, the above approach is very similar to Burger's equation, but with source terms. The source terms take care of the pressure forces and adiabatic compression/expansion of the gas. One can now apply a standard advection routine for the advection of conserved scalar functions (see Chapters 3 and 4) to all hydrodynamic conserved quantities  $\rho$ ,  $\rho u$  and  $\rho e_{\text{tot}}$ , and at the end of each time step add the source terms to the momentum- and energy equation.

Apart from the fact that this is easy to do, another advantage of this is that one can apply algorithms of arbitrary high order. Of course, this high-order advection only applies to fluid mo-

tion and not to the sound waves, so the high-precision advection does not make the propagation of sound waves more accurate in this approach. But for very sub-sonic flows the propagation of sound waves is anyway of lesser interest, and the fluid motions are then anyway the most important flow patterns, which can thus be advected with any precision one likes by choosing the right advection algorithm. If the gradient of the pressure (i.e. the source term on the rhs of the momentum and energy equations) is also evaluated with a higher-order accurate derivative, then the entire algorithm can be made higher order.

A final advantage of this approach is that hydrostatic solutions of fluid flows with external body forces can be easily ‘recognized’ by the algorithm. This is because all forces are handled as source terms on the right-hand-side of the equations. In equilibrium all these forces cancel exactly, and will not produce numerical disturbances to the advection part of the equation. This approach is very well suited for problems that are near some hydrostatic equilibrium, such as planetary atmospheres or fluid motions in a rotating system.

## 5.2 A simple 1-D hydrodynamics algorithm

Let us jump right into making an algorithm, but let us do this for the moment for a simplified case: the problem of 1-D isothermal hydrodynamics. The two conserved quantities to advect are  $q_1 = \rho$  and  $q_2 = \rho u$ . The pressure is  $P = \rho c_s^2$  where  $c_s^2$  is a global constant. The equations to solve are:

$$\partial_t q_1 + \partial_x(q_1 u) = 0 \quad (5.4)$$

$$\partial_t q_2 + \partial_x(q_2 u) = -\partial_x P \quad (5.5)$$

### 5.2.1 The algorithm

We will approach this using operator splitting:

1. First we do the advection without source term,

$$q_1^{n+1/2} = q_1^n - \Delta t \partial_x(q_1^n u) \quad (5.6)$$

$$q_2^{n+1/2} = q_2^n - \Delta t \partial_x(q_2^n u) \quad (5.7)$$

2. Then we will add the source term:

$$q_1^{n+1} = q_1^{n+1/2} \quad (5.8)$$

$$q_2^{n+1} = q_2^{n+1/2} - \partial_x P^{n+1/2} \quad (5.9)$$

where  $P^{n+1/2} = q_1^{n+1/2} c_s^2$ .

For the advection, let us for the moment simply choose the donor-cell algorithm. The  $q_1$  and  $q_2$  are located at the grid cell centers, so that we have  $q_{1,i}$  and  $q_{2,i}$  with  $0 \leq i \leq N - 1$ . To produce an advective flux at the cell interfaces, we need to calculate first the cell interface value of the velocity. We do this simply by averaging the velocity over the two adjacent cells:

$$u_{i+1/2} = \frac{1}{2} \left( \frac{q_{2,i}}{q_{1,i}} + \frac{q_{2,i+1}}{q_{1,i+1}} \right) \quad (5.10)$$

Then we define the flux:

$$f_{1,i+1/2} = \begin{cases} q_{1,i}^n u_{i+1/2} & \text{for } u_{i+1/2} > 0 \\ q_{1,i+1}^n u_{i+1/2} & \text{for } u_{i+1/2} < 0 \end{cases} \quad (5.11)$$

and the same for  $f_{2,i+1/2}$ . We update  $q$  as

$$q_{1,i}^{n+1/2} = q_{1,i}^n - \Delta t \frac{f_{1,i+1/2} - f_{1,i-1/2}}{x_{i+1/2} - x_{i-1/2}} \quad (5.12)$$

and the same for  $q_{2,i}^{n+1/2}$ . This is the donor-cell advection. The  $q^{n+1/2}$  does not mean that we do the advection for only half a time step. It only means that, by operator splitting, we still have another operator to go.

For this second operator, the addition of the source, we simply take the approximation:

$$\left. \frac{\partial P}{\partial x} \right|_i \rightarrow \frac{P_{i+1}^{n+1/2} - P_{i-1}^{n+1/2}}{x_{i+1} - x_{i-1}} = c_s^2 \frac{\rho_{i+1}^{n+1/2} - \rho_{i-1}^{n+1/2}}{x_{i+1} - x_{i-1}} \quad (5.13)$$

and we perform the following update, this time only for  $q_{2,i}$

$$q_{2,i}^{n+1} = q_{2,i}^{n+1/2} - c_s^2 \frac{\rho_{i+1}^{n+1/2} - \rho_{i-1}^{n+1/2}}{x_{i+1} - x_{i-1}} \quad (5.14)$$

In principle this is it, but we must take special care at the boundary. We must decide which kind of boundary condition to take. In Section 5.3 we shall go into more detail, but for now let us assume a reflective boundary condition in which the advective velocities at the left interface of the left boundary cell and the right interface of the right boundary cell are assumed to be zero, and in which the pressure outside of the domain is assumed to be equal to the pressure in the boundary cell. If  $i = 0$  is the first cell, and  $i = N - 1$  is the last cell, then we write  $u_{-1/2} = u_{N-1/2} = 0$  for the advection. For the pressure source term in the momentum equation we take (only the left boundary condition as an example; right goes similar):

$$q_{2,0}^{n+1} = q_{2,0}^{n+1/2} - \frac{1}{2} c_s^2 \frac{\rho_1^{n+1/2} - \rho_0^{n+1/2}}{x_1 - x_0} \quad (5.15)$$

The factor 1/2 comes in because actually the  $\Delta x = x_1 - x_{-1}$ , but since without ghost cells  $x_{-1}$  does not exist, we do it with a factor 1/2. Note, then, that ghost cells may indeed be handy. We will use them later.

### 5.2.2 The computer implementation

For the computer implementation we must first identify the indices of the interfaces, because a computer cannot address  $i + 1/2$  in an array. As usual we *always* take the interface to be *left* of the cell with the same number:  $u_{i-1/2} \equiv u[i]$ . We make an array of  $N$  values for  $q_1 \equiv \rho$  and  $q_2 \equiv \rho u$ , and any interface quantities will be arrays of  $N + 1$  values, starting with the left interface of the leftmost cell and ending with the right interface of the rightmost cell.

Now let us create a subroutine for doing one single time step of the hydrodynamics:

```
pro hydroiso_cen,x,xi,rho,rhou,e,gamma,dt
nx = n_elements(x)
```

```
;
; Compute the velocity at the cell interfaces
;
ui = dblarr(nx+1)
for ix=1,nx-1 do begin
    ui[ix] = 0.5 * ( rhou[ix]/rho[ix] + rhou[ix-1]/rho[ix-1] )
endfor
;
; Compute the flux for rho
;
fluxrho = dblarr(nx+1)
for ix=1,nx-1 do begin
    if ui[ix] gt 0. then begin
        fluxrho[ix] = rho[ix-1] * ui[ix]
    endif else begin
        fluxrho[ix] = rho[ix] * ui[ix]
    endelse
end
end
;
; Update the density
;
for ix=0,nx-1 do begin
    rho[ix] = rho[ix] - (dt/(xi[ix+1]-xi[ix]))*$
                    (fluxrho[ix+1]-fluxrho[ix])
endfor
;
; Compute the flux for rho u
;
fluxrhoul = dblarr(nx+1)
for ix=1,nx-1 do begin
    if ui[ix] gt 0. then begin
        fluxrhoul[ix] = rhoul[ix-1]^2 / rho[ix-1]
    endif else begin
        fluxrhoul[ix] = rhoul[ix]^2 / rho[ix]
    endelse
end
end
;
; Update the momentum
;
for ix=0,nx-1 do begin
    rhoul[ix] = rhoul[ix] - (dt/(xi[ix+1]-xi[ix]))*$
                    (fluxrhoul[ix+1]-fluxrhoul[ix])
endfor
;
; Compute the pressure
;
p      = (gamma-1.d0)*rho*e
```

---

```

;
; Now add the pressure force, for all cells
; except the ones near the boundary
;
for ix=1,nx-2 do begin
    rhou[ix] = rhou[ix] - dt*(p[ix+1]-p[ix-1])/(x[ix+1]-x[ix-1])
endfor
;
; Now do the boundary cells, assuming mirror
; symmetry in the boundaries
;
rhoul[0]      = rhoul[0]      - 0.5*dt*(p[1]-p[0])/(x[1]-x[0])
rhoul[nx-1] = rhoul[nx-1] - 0.5*dt*(p[nx-1]-p[nx-2])/(x[nx-1]-x[nx-2])
;
; Done
;
end

```

This contains everything we discussed above: the advection, the computation of the new pressure, the implementation of the boundary conditions.

Now let us define a test problem.

```

nx          = 100
nt          = 1000
x0          = 0.d0
x1          = 100.d0
xmid        = 0.5 * (x0+x1)
dt          = 0.25
cfl         = 0.5
x           = x0 + (x1-x0)*(dindgen(nx)/(nx-1.d0))
gamma       = 7./5.
rho         = dblarr(nx,nt+1)
rhoul       = dblarr(nx,nt+1)
e           = dblarr(nx)+1.d0
time        = dblarr(nt+1)
dg          = 0.1*(x1-x0)
rho[:,0]    = 1.d0+0.3*exp(-(x-xmid)^2/dg^2)

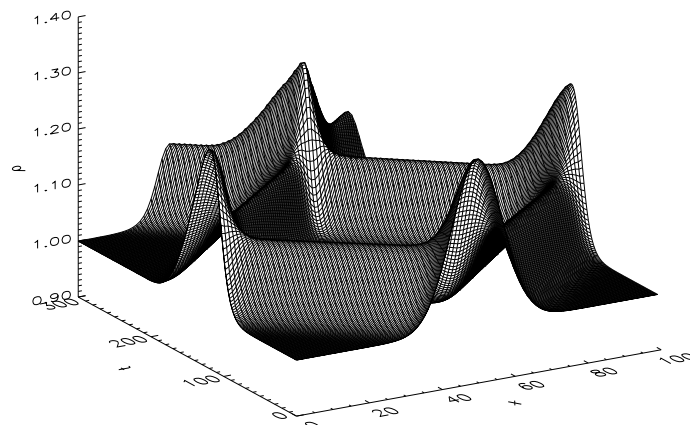
```

And produce the xi array which is needed, as well as a dx array.

```

;
; Now some additional arrays are set up
;
xi          = dblarr(nx+1)
xi[1:nx-1] = 0.5 * ( x[1:nx-1] + x[0:nx-2] )
xi[0]       = 2*xi[1] - xi[2]
xi[nx]      = 2*xi[nx-1] - xi[nx-2]
dx          = ( xi[1:nx] - xi[0:nx-1] )

```



**Figure 5.1.** The density as a function of space  $x$  and time  $t$  with initial condition of a gaussian perturbation, solved with the simple first-order hydrodynamics algorithm of Section 5.2.

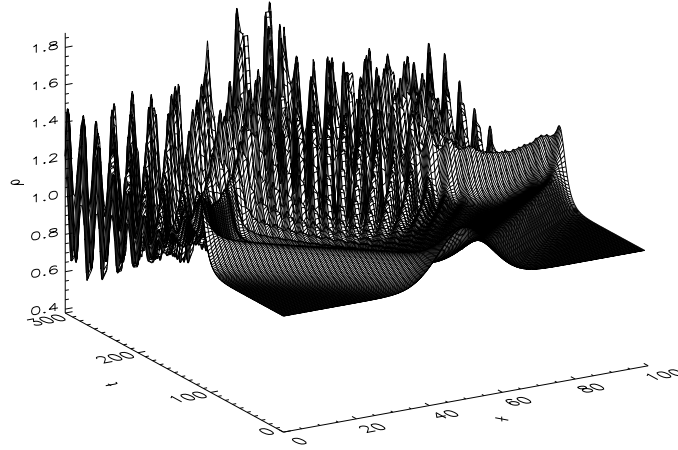
We now want to compute 1000 time steps of the hydrodynamics. We must recompute the smallest allowed  $\Delta t$  each time step. We define a dimensionless *Courant number* which tells how much smaller than the formal maximum we wish to take the time step. And here we go:

```

;
; Now the hydro is done
;
for it=1,nt do begin
    qrho      = rho[*,it-1]
    qrhou     = rhou[*,it-1]
    cs        = sqrt(gamma*(gamma-1)*e)
    dum       = dx/(cs+abs(qrhou/qrho))
    dt        = cfl*min(dum)
    time[it]  = time[it-1]+dt
    ;;
    print,'Time step ',it,', Time = ',time[it],', Dt = ',dt
    ;;
    hydroiso_cen,x,xi,qrho,qrhou,e,gamma,dt
    ;;
    rho[*,it] = qrho
    rhou[*,it] = qrhou
endfor

```

The results are shown in Fig. 5.1. It is seen that the Gaussian blob in the density splits up in a left- and a right-moving blob, which both reflect off the walls of the domain and return toward the center of the domain. By that time, due to the non-linearity of the equations and the strength of the perturbation (30% of the background value), the wave has steepened into something that looks like a shock, but is still spread over multiple cells. This wave pattern repeats many times and the shock front steepens, but still remains smeared over a number of cells. Nevertheless, it seems that we have produced our first succesful hydrodynamics code.



**Figure 5.2.** Same as Fig. 5.1, but now using always the start-of-the-time-step variables for the update of the state. A numerical instability appears.

### 5.2.3 The order of evaluation matters...

The above exercise may have not seemed too difficult. But still we only have a first order algorithm and of course we have been lucky that we have not fallen into the many pitfalls that can exist. One possible problem appears if the order of the evaluations is done in a different way. Suppose that the evaluation of the pressure is done before the advection of the density. In effect we then get

$$q_{2,i}^{n+1} = q_{2,i}^{n+1/2} - c_s^2 \frac{\rho_{i+1}^n - \rho_{i-1}^n}{x_{i+1} - x_{i-1}} \quad (5.16)$$

instead of Eq. 5.14. This is only a very minor change in the algorithm. It simply means that we use the values at the start of the time step for any of the operations we do (advection, source addition). It may not seem as a mistake, but Fig. 5.2 shows the result. In the beginning the wave propagates in the way that was expected, but soon a serious instability appears and wrecks the entire solution. This shows that when sources like the pressure source are added, a true operator splitting is required: first the advection, then recompute the pressure from the new variables, and then add the pressure source.

## 5.3 Boundary conditions, ghost cells

We have seen already in chapter 4 that the use of ghost cells at the boundaries can be very useful to easily implement boundary conditions. There are a number of different kinds of boundary conditions one can install in this way. Here is a list of common ones (where we mainly focus on the left boundary).

1. Periodic boundary conditions:  $\rho[0] = \rho[N_x]$ ,  $u[0] = u[N_x]$  (and similarly  $\rho[N_x + 1] = \rho[1]$ ,  $u[N_x + 1] = u[1]$ ).
2. Reflective boundary conditions:  $\rho[0] = \rho[1]$ ,  $u[0] = -u[1]$ .
3. Free outflow/inflow:  $\rho[0] = \rho[1]$ ,  $u[0] = u[1]$ .
4. Free outflow, no inflow:  $\rho[0] = \rho[1]$ ,  $u[0] = -\text{abs}(u[1])$ .



5. Given-state boundary condition:  $\rho[0] = \rho_l(t)$ ,  $u[0] = u_l(t)$ .

and similar for the right boundary. In 2-D and 3-D hydrodynamics these types of boundary conditions are the same, but then including also the other two velocity components.

Periodic boundary conditions have many uses. First of all, sometimes the system we wish to model is clearly periodic (such as if we wish to model the atmosphere of the Earth where going East eventually brings you back where you came from). But also for other applications can periodic boundary conditions be useful: if we wish, for instance, to model the nature of turbulence, then we are anyway interested in flow patterns with scales that are much smaller than the computational domain. But we do wish to ensure that there are no artificial damping or forcing effects caused by imposing some arbitrary boundary condition. A periodic boundary condition basically removes beforehand any artificial boundary effects because the turbulence can now not feel the boundary at all. The only way that the turbulence can 'feel' the fact that the computational domain is limited is that it cannot grow modes larger than the size of the computational domain.

Reflective boundary conditions are also very often used. The advantage is that no mass nor energy can flow off the grid. Energy and mass are therefore globally conserved, or in other words: the system is closed. There are no external influences that may affect the solution because the problem is perfectly self-contained. A disadvantage of reflective boundary conditions is that any waves that are spawned by the interesting part of the flow pattern that we are modeling are also perfectly reflected, and typically return to the area of interest and start interfering with that area. For instance, if we want to model how a rotating irregular body in a volume of gas stirs waves and how these waves transport away energy and damp the rotation of the body, then the reflective boundary conditions bring back the waves to the body and start affecting its motion. This is then clearly undesired.

A free outflow/inflow boundary condition has the advantage that any waves that are produced in the model are simply advected off the grid. The waves do not return anymore. This is clearly desirable in many cases. But the problem is that if for some reason the velocity  $u[1]$  becomes inflowing ( $u[1] > 0$ ), then the state at cell  $i$  will determine the influx of matter. This can typically cause completely arbitrary influx of matter. One can see why this is the case: one of the two characteristics is pointing inward into the grid ( $\lambda_+ = u + c_s$ ). So there is always an inflow of information into the grid. But if the state in the ghost cell is purely given as a function of the state at  $i = 1$ , then the state at  $i = 1$  determines itself what kind of information it wishes to receive from grid point  $i = 0$  (ghost cell) through that characteristic. Clearly this is unphysical because the inward-pointing characteristic ( $\lambda_+ = u + c_s$ ) should transport information only toward positive  $x$  and *never* receive any information from larger  $x$ . This boundary condition is therefore quite dangerous. Only if we have a *supersonic* outflowing motion toward this boundary can we be sure that this boundary condition will never cause problems. The reason is that in the case of supersonic outflow both characteristics point out of the grid:  $\lambda_- = u - c_s < 0$  and  $\lambda_+ = u + c_s < 0$ . Then by copying the state to the ghost cell no information is propagated in the upstream direction.

The outflow-but-no-inflow condition is an attempt to solve the problem of arbitrary inflow in subsonic cases. It clearly does not allow unphysical inflow of matter, but it allows matter to flow off the grid. It is, so to speak, a sink. It is not ideal, because there is still the problem of propagating information upstream, but if the goal of imposing this boundary condition is to get rid of any matter that comes near the boundary, then this is sometimes used.



With the given-state boundary condition one can do many things. For instance, one can force waves, or impose some influx of matter at the boundary. However, with this method this does not always work. Suppose we impose an influx of  $f = \rho[0] * u[0] = 1$  at the boundary by putting  $\rho[0] = 1$ ,  $u[0] = 1$ , but the state in the modeling domain is  $\rho[1] = 1000$ ,  $u[1] = 1$ . We may think that we imposed  $f = 1$  at the boundary, but instead the pressure in cell 1 is completely overwhelming the pressure in ghost cell 0, and a negative flux of matter will result. So the imposed-state boundary condition works only if the influx has a momentum flux that clearly overcomes the pressure in the modeling domain. We could, instead, decide to impose a flux at boundary  $i = 1/2$ , i.e. the left wall of the boundary cell, and completely skip the ghost cell approach. That would allow a perfect and strict imposition of a flux. A disadvantage here is that one does not know if this flux is physical in this situation.

### 5.3.1 Ghost cells for schemes with a 5-point stencil

A single ghost cell on each boundary is sufficient if the stencil of the advection scheme is symmetric and 3-point. For linear advection schemes such as Donor-cell and Lax-Wendroff this is the case. However, some other schemes such as Beam-warming or Fromm require, for the update at point  $i$  also information of the cells at  $i \pm 2$  in addition to the  $i \pm 1$  points. Also, when the flux limiter method is used with a non-linear flux limiter recipe, then often the  $i \pm 2$  are used. Such schemes are therefore in principle 5-point schemes, even if, for a given velocity direction, only 3 points are used ( $i - 2, i - 1, i$  or  $i, i + 1, i + 2$ ) because we do not know a-priori which direction the velocity points. For such schemes a single ghost cell on each boundary is not sufficient. We must implement a double layer of ghost cells. For the direct flux from these cells into the physical domain the first ghost cell (the one next to the physical domain) is of most importance, because that is the donor of the flux. But the second ghost cell will, in the case of flux-limiters, determine what the value of the flux limiter is, and can thereby significantly affect the solution nonetheless. The implementation of the boundary condition goes, for the rest, similar to the single-ghost cell case.

## 5.4 Hydrodynamics with ghost cells

So now that we see that ghost cells are extremely useful to implement boundary conditions, let us start all over again and produce a new algorithm in which two ghost cells are used on each side. Also, let us implement a standardized advection subroutine which we shall call `advect()`, which advects any conserved quantity with a given velocity given at the cell interfaces, and a standardized boundary condition routine `boundary()` that sets the ghost cells to the proper values consistent with the kind of boundary condition used. Let us first show the routine `advect()`, but note that a more sophisticated version of `advect()` (with more choices of flux limiters) will be made available for the exercises:

```

pro advect,x,xi,q,ui,dt,fluxlim,nghost
nx      = n_elements(xi)-1
if n_elements(x) ne nx then stop
if n_elements(xi) ne nx+1 then stop
if n_elements(q) ne nx then stop
if n_elements(ui) ne nx+1 then stop
if nghost lt 1 then stop
;
```

```
; Determine the  $r_{\{i-1/2\}}$  for the flux limiter
;
r = dblarr(nx+1)
for i=2,nx-2 do begin
    dq = (q[i]-q[i-1])
    if abs(dq) gt 0.d0 then begin
        if(ui[i] ge 0.d0) then begin
            r[i] = (q[i-1]-q[i-2])/dq
        endif else begin
            r[i] = (q[i+1]-q[i])/dq
        endelse
    endif
endfor
;
; Determine the flux limiter
; (many other flux limiters can be implemented here!)
;
case fluxlim of
    'donor-cell': begin
        phi = dblarr(nx+1)
    end
    'superbee': begin
        phi = dblarr(nx+1)
        for i=1,nx-1 do begin
            a = min([1.d0,2.d0*r[i]])
            b = min([2.d0,r[i]])
            phi[i] = max([0.d0,a,b])
        endfor
    end
    else: stop
endcase
;
; Now construct the flux
;
flux = dblarr(nx+1)
for i=1,nx-1 do begin
    if ui[i] ge 0.d0 then begin
        flux[i] = ui[i] * q[i-1]
    endif else begin
        flux[i] = ui[i] * q[i]
    endelse
    flux[i] = flux[i] + 0.5 * abs(ui[i]) * $
        (1-abs(ui[i]*dt/(x[i]-x[i-1]))) * $
        phi[i] * (q[i]-q[i-1])
endfor
;
; Update the cells, except the ghost cells
```

---

```

;
for i=nghost,nx-1-nghost do begin
    q[i] = q[i] - dt * ( flux[i+1]-flux[i] ) / ( xi[i+1] - xi[i] )
endfor
;
end

```

Now the boundary condition implementation routine, which is only necessary for imposing periodic boundary conditions or mirror boundary conditions, is:

```

pro boundary,rho,rhou,periodic=periodic,mirror=mirror
;
; Get the number of grid points including the ghost cells
;
nx = n_elements(rho)
;
; If periodic, then install periodic BC, using two ghost cells
; on each side (two are required for the non-lin flux limiter)
;
if keyword_set(periodic) then begin
    rho[0]      = rho[nx-4]
    rho[1]      = rho[nx-3]
    rho[nx-2]   = rho[2]
    rho[nx-1]   = rho[3]
    rhou[0]     = rhou[nx-4]
    rhou[1]     = rhou[nx-3]
    rhou[nx-2]  = rhou[2]
    rhou[nx-1]  = rhou[3]
endif
;
; If mirror symmetry, then install mirror BC, using two ghost cells
; on each side (two are required for the non-lin flux limiter)
;
if keyword_set(mirror) then begin
    rho[0]      = rho[3]
    rho[1]      = rho[2]
    rho[nx-2]   = rho[nx-3]
    rho[nx-1]   = rho[nx-4]
    rhou[0]     = -rhou[3]
    rhou[1]     = -rhou[2]
    rhou[nx-2]  = -rhou[nx-3]
    rhou[nx-1]  = -rhou[nx-4]
endif
end

```

The actual hydrodynamics subroutine makes use of the above routines. It reads:

```

pro hydrostep,x,xi,rho,rhou,e,gamma,dt,periodic=periodic,$

```

```
        mirror=mirror,fluxlim=fluxlim,nrvisc=nrvisc
;
; Check for conflicting settings
;
if keyword_set(mirror) and keyword_set(periodic) then stop
;
; Use 2 ghost cells on each side
;
nghost = 2
;
; If not defined, install default flux limiter
;
if not keyword_set(fluxlim) then fluxlim='donor-cell'
;
; Get the number of grid points including the ghost cells
;
nx = n_elements(x)
;
; Impose boundary conditions
;
boundary,rho,rhou,periodic=periodic,mirror=mirror
;
; Compute the velocity at the cell interfaces
;
ui      = dblarr(nx+1)
for ix=1,nx-1 do begin
    ui[ix] = 0.5 * ( rhou[ix]/rho[ix] + rhou[ix-1]/rho[ix-1] )
endfor
;
; Advect rho
;
advect,x,xi,rho,ui,dt,fluxlim,nghost
;
; Advect rho u
;
advect,x,xi,rhou,ui,dt,fluxlim,nghost
;
; Re-impose boundary conditions
;
boundary,rho,rhou,periodic=periodic,mirror=mirror
;
; Compute the pressure
;
p      = (gamma-1.d0)*rho*e
;
; Now add the pressure force, for all cells except the ghost cells
;
```

```

for ix=2,nx-3 do begin
    rhou[ix] = rhou[ix] - dt*(p[ix+1]-p[ix-1])/(x[ix+1]-x[ix-1])
endfor
;
; Re-impose boundary conditions a last time (not
; strictly necessary)
;
boundary,rho,rhou,periodic=periodic,mirror=mirror
;
; Done
;
end

```

This routine makes sure to always re-implement the boundary conditions for the ghost cells. This is done without checking which variables have to be updated, so it is a bit inefficient, but it is safe. Note, also, that in the above routines `nx` denotes the number of cells including the ghost cells. With the above routines one can again do experiments and see how the solutions behave for different flux limiters and boundary conditions. One true advantage of the above implementation is that it is now very easy to impose periodic boundary conditions, because we only copy the state variables from the left to the right boundary and from the right to the left boundary.

- **Exercise:** Implement the above routines and create a setup with a sound wave moving from left to right. Impose periodic boundary conditions. Tip: Make sure that the sine-wave is such that `rho[0]=rho[nx-4]` and `rho[1]=rho[nx-3]`, so that the periodicity of the wave is perfect, in the presence of the two ghost cells on each side. Describe how the sound wave steepens, sheds smaller waves and forms a shock.

## 5.5 Now including the energy equation

So far we have neglected the energy equation because we assumed that the temperature was constant at all times. However, most interesting applications do not have this property. We therefore must include the energy equation, including the work source term due to the pressure force:

$$\partial_t(\rho e_{\text{tot}}) + \partial_x(\rho e_{\text{tot}} u) = -\partial_x(Pu) \quad (5.17)$$

We define a third conserved quantity  $q_3 \equiv \rho e_{\text{tot}}$ . As in the case of the momentum equation we apply the method of operator splitting here: we first solve the equation  $\partial_t q_3 + \partial_x(q_3 u) = 0$  for one time step, and then solve (with the new variables) the equation  $\partial_t q_3 = -\partial_x(Pu)$  for the same time step. The advection of the total energy is done in exactly the same way as for the density and the momentum. In fact, we can likewise use the subroutine `advect()` for it.

The difficulty lies in the work term  $-\partial_x(Pu)$ . First of all we need to calculate the pressure  $P$  from the three conserved quantities  $q_1 \equiv \rho$ ,  $q_2 \equiv \rho u$  and  $q_3 \equiv \rho e_{\text{tot}}$ . In fact, we also need this  $P$  for the source term in the momentum equation. The way this can be done is to first compute the thermal energy  $e_{\text{th}}$  from the total energy:

$$u = q_2/q_1 \quad (5.18)$$

$$e_{\text{tot}} = q_3/q_1 \quad (5.19)$$

$$e_{\text{kin}} = u^2/2 \quad (5.20)$$

$$e_{\text{th}} = e_{\text{tot}} - e_{\text{kin}} \quad (5.21)$$

Once we know this, we compute the pressure according to  $P = (\gamma - 1)\rho e_{\text{th}}$ .

Then we use  $P$  for both the source term in the momentum equation (the force) and for the source term in the energy equation (the work). For the former we write

$$\frac{q_{2,i}^{n+1} - q_{2,i}^{n+1/2}}{\Delta t} = -\frac{P_{i+1} - P_{i-1}}{2\Delta x} \quad (5.22)$$

where  $q_{2,i}^{n+1/2}$  denotes the update of  $q_{2,i}^n$  due to the advection. For the latter we can write:

$$\frac{q_{3,i}^{n+1} - q_{3,i}^{n+1/2}}{\Delta t} = -\frac{P_{i+1}u_{i+1} - P_{i-1}u_{i-1}}{2\Delta x} \quad (5.23)$$

These new elements can be readily built in into the algorithm of Section 5.4, and this should produce a working algorithm.

### 5.5.1 Are these equations conservative?

As was mentioned in Chapter 4 it can be very important to make sure that conserved quantities actually remain conserved in the numerical simulation. In the current approach, however, we have put the pressure force and work to the right-hand-side of the conservation equation, as a source term. This raises the question: are the equations still conservative? The answer is: in principle yes, if the force and work terms are written in a proper way. Define the momentum flux through interface  $i - 1/2$  to be  $f_{p,i-1/2} = \frac{1}{2}(P_{i-1} + P_i)$ , and similar for  $i + 1/2$ . Then we obtain:

$$q_{2,i}^{n+1} = q_{2,i}^{n+1/2} - \Delta t \frac{f_{p,i+1/2} - f_{p,i-1/2}}{x_{i+1/2} - x_{i-1/2}} = q_{2,i}^{n+1/2} - \frac{\Delta t}{2} \frac{P_{i+1} - P_{i-1}}{x_{i+1/2} - x_{i-1/2}} \quad (5.24)$$

and similar for the energy equation. For a regular grid this becomes equal to the expressions of Eqs.(5.22,5.23). This shows that implicitly the equations are numerically conservative, even though we have not explicitly put them in a flux-conserved form. For non-regular gridding one should stick to the presently derived expression, so that flux conservation is guaranteed.

## 5.6 Shock waves and the Von Neumann - Richtmyer artificial viscosity

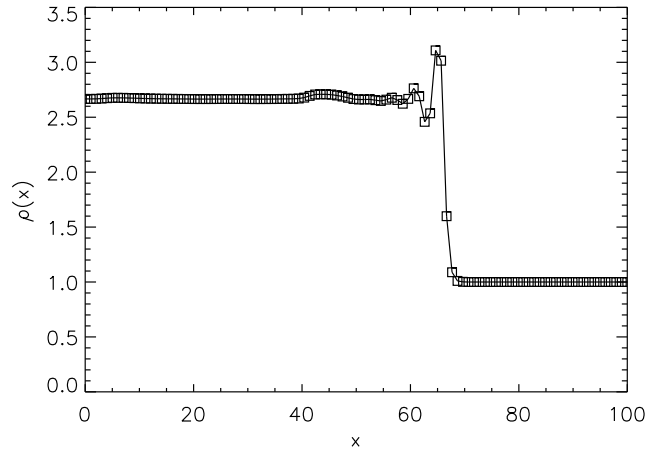
We have seen in the examples above that waves tend to steepen and form shocks. Let us take a closer look, by simulating a true right-moving shock wave and analyzing its behavior. Let us assume that a shock wave is moving from the left boundary into a non-moving medium. We can define the *Mach number*  $\mathcal{M}$  to be the ratio of the shock speed  $u_s$  to the sound speed  $C_s$  on the pre-shock medium. Let us call the pre-shock medium the 'right domain' and the post-shock region the 'left domain', because the shock moves from left to right. We therefore define  $\mathcal{M} \equiv u_s/C_{s,r}$ . According to the conservation of density, momentum and energy over the shock (Rankine-Hugoniot) we can write:

$$\rho_l = \rho_r \frac{(\gamma + 1)\mathcal{M}^2}{(\gamma - 1)\mathcal{M}^2 + 2} \quad (5.25)$$

$$P_l = P_r \frac{2\gamma\mathcal{M}^2 - (\gamma - 1)}{\gamma + 1} \quad (5.26)$$

$$u_s = \mathcal{M} \sqrt{\gamma \frac{P_r}{\rho_r}} \quad (5.27)$$

$$u_l = u_s \frac{\rho_l - \rho_r}{\rho_l} \quad (5.28)$$



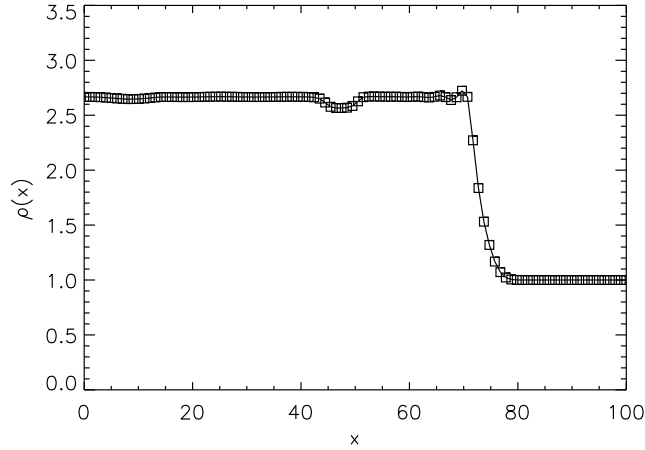
**Figure 5.3.** The result of a test problem with a  $\mathcal{M} = 2$  shock wave moving into a non-moving medium of  $\rho = 1$ ,  $P = 0.1$ , modeled using advection with a superbee flux limiter.

This is then the state that we plug into the ghost cells on the left boundary. It should, in principle, produce a clean shock front moving from the left boundary to the right. Now let us see what we get if we use advection according to the superbee flux limiter for a shock of Mach 2. This is depicted in Fig. 5.3 What one sees is that in principle the shock wave is well reproduced, but there are some oscillations at the shock front. This has a natural explanation. The equations for the hydrodynamics solver were derived from the continuous Euler equations. These equations are valid for smooth flow, but break down for shock fronts, so we should not expect the hydrodynamics solver to be able to handle shocks. In fact, a shock is the only location where non-viscous hydrodynamics can increase the entropy of the gas flow. The way that Nature does this is that at very small spatial scales the molecular viscosity becomes important, and this viscosity produces entropy in the shock front. Basically one can say that in Nature the shock front has a small but non-negligible width  $L$  such that the Reynolds number  $\text{Re} = \frac{uL}{\nu}$  is of order unity in this shock front. Kinetic energy can then be dissipated to heat in this shock, which increases the entropy of the gas.

In numerical hydrodynamics the grid cell size is usually orders of magnitude larger than the true width of the shock. And the equations that we used for the hydrodynamics solver did not explicitly include any entropy-generating viscosity terms. Therefore we *expect* that our algorithm should produce errors in the flow near the shock front, in particular downstream of the shock front. The interesting thing is, in fact, that the method already does surprisingly well, given these thoughts (see, however, Section 5.10.2 for what happens if a different energy equation is used). Far downstream of the shock front the state is reasonably correct and the entropy increase, expected from the shock, is indeed there. So one should ask oneself two questions:

1. Why does our solver handle the shock reasonably well, even though we never inserted any entropy-generating viscous terms by hand?
2. How can we do better, so that we do not get the wiggles behind the shock?

To answer question 1, the answer lies in the fact that due to the intrinsic diffusivity of the advection algorithm, any oscillations are smeared out quickly. The question remains then, how does the code know how much entropy to generate? This can be traced back to the fact that the algorithm is in conservative form. If the post-shock state variables are smoothed out and all



**Figure 5.4.** As Fig. 5.3, but now with von Neumann-Richtmyer artificial viscosity with  $\xi = 3$ .

oscillations have gone, then there *is only one set of state variables* consistent with the post-shock density, momentum and energy flux. So diffusing oscillations out, but keeping the algorithm perfectly conservative, must automatically produce the right state and therefore the right increase in entropy compared to the pre-shock state. Diffusion is therefore a good thing in this respect. In fact, if we would use the donor-cell algorithm for the advection, which has intrinsically more numerical diffusion, then the post-shock oscillations are already quite low. But the drawback of donor-cell algorithm is that *any* flow feature is smeared out. This can therefore not be the final solution.

To answer question 2: we need to find a way to increase the diffusivity near the shock front, but not elsewhere. In fact, we can follow a physically motivated path by introducing an *artificial viscosity* (as opposed to artificial diffusivity) which sort of mimics the physical shock viscosity, but then smeared out over a region the size of a few grid cells instead of the unresolvable true shock width. The most popular artificial viscosity recipe for handling shocks is the *von Neumann-Richtmyer artificial viscosity*. This is a bulk viscosity which acts as an additional pressure:

$$\Pi_i = \begin{cases} \frac{1}{4}\xi^2(u_{i+1} - u_{i-1})^2\rho_i & \text{if } u_{i+1} \leq u_{i-1} \\ 0 & \text{if } u_{i+1} > u_{i-1} \end{cases} \quad (5.29)$$

where  $\xi$  is a tuning parameter which specifies over how many grid cells a shock should be spread out. It is typically chosen to be of the order of 2 or 3. To implement it in the algorithm one replaces any instance of the pressure  $P$  with  $P + \Pi$ . This takes care of the force as well as the work done by this bulk viscosity. For the above test problem, the result for  $\xi = 3.0$  is shown in Fig. 5.4. One sees that the post-shock oscillations have gone. The penalty is that the viscosity affects the pre-shock region and the shock is therefore not as sharp anymore as we had before. Also there is a little dip in the density profile where no dip should be. This has formed in the very initial phase as the shock started to propagate into the domain. Since the shock as modeled by the numerical algorithm is not infinitely sharp, the initial conditions were not in agreement with the “numerical form of the shock”, and startup oscillations were present. They produced a bit too much entropy in this region, causing the density (for the same pressure) to be a bit lower than should be.

Von Neumann-Richtmyer artificial viscosity has the good property that it only becomes strong when its presence is required, and becomes negligible when the flow is smooth. It is



therefore much better than the uncontrolled numerical diffusivity of for instance the donor-cell algorithm. The von Neumann-Richtmyer artificial viscosity, and varieties thereof, is used in a great many numerical hydrodynamics codes.

## 5.7 Odd-even decoupling

The numerical algorithms we have constructed so far have an interesting problematic property that is only seldomly serious, but might be useful to keep in mind.

Suppose we start with a situation in which the velocity is everywhere zero:  $u(x) = 0$ . We assume that the specific thermal energy ( $\propto$  temperature) is also constant, for simplicity. Let us take it:  $e_{\text{th}}(x) = 1$ . But we let the density vary as a ‘sawtooth’ profile:

$$\rho_i = \begin{cases} 1 & \text{if } i = 1, 3, 5, 7 \dots \\ 2 & \text{if } i = 2, 4, 6, 8 \dots \end{cases} \quad (5.30)$$

The pressure in these cells is then  $P_i = 1$  for  $i = 1, 3, 5, \dots$  and  $P_i = 2$  for  $i = 2, 4, 6, \dots$ . So in principle the even cells are over-pressurized compared to their neighbors. Physically these cells should quickly depressurize by moving part of their content to their under-pressurized neighbors. This should give a high-frequency oscillation, and with the usual numerical viscosity this should then automatically damp out. However, for this case the momentum flux through cell interfaces  $i - 1/2$  and  $i + 1/2$  is

$$F_{i-1/2}^{(1)} = \frac{1}{2}(P_{i-1} + P_i) \quad F_{i+1/2}^{(1)} = \frac{1}{2}(P_i + P_{i+1}) \quad (5.31)$$

The update of the momentum at cell center  $i$  is:

$$\frac{\rho_i^{n+1} u_i^{n+1}}{\Delta t} = -\frac{F_{i+1/2}^{(1)} - F_{i-1/2}^{(1)}}{\Delta x} = -\frac{1}{2\Delta x}(P_{i+1} - P_{i-1}) = 0 \quad (5.32)$$

So because the  $P_i$  cancels out, the update of the momentum is zero. This is also logical, since the fluxes of momentum in both interfaces are the same, and therefore their differences cancel. This is a potential problem because the code leaves these oscillations there without damping them out, i.e. it leaves an unphysical solution. In fact, it is fundamentally impossible to solve this problem if momentum, density and pressure are located on the same gridpoints, unless artificial diffusion is invoked. This phenomenon is called *odd-even decoupling*.

One sees that the odd-even decoupling does not involve an exponentially growing unstable mode. But there is also no damping. So any numerical noise or other causes of errors may simply get ‘stuck’ in an odd-even mode where they do not get amplified, but they do not get damped either. Such ‘sawtooth’ modes are clearly an unwanted effect. One simple way to get rid of them would be diffusion, but that would also diffuse out flow patterns that we wish to resolve. A better way, which also automatically increases the order of the algorithm, is to use *staggered grids*.

## 5.8 Staggered grids

A numerical hydrodynamics scheme with a *staggered grid* places any vectorial components, such as the momentum density  $q_2 \equiv \rho u$ , not on the cell centers but on the cell interfaces. In such a scheme one would have  $q_1 \equiv \rho$  and  $q_3 \equiv \rho e_{\text{tot}}$  located at the cell centers and  $\rho u$  located on the

cell interfaces. But since  $q_2$  is also a conserved quantity we must also define a cell around  $q_2$ . The location of the cell boundaries for  $q_1$  and  $q_3$  act as cell centers for  $q_2$ , whereas the locations of the cell centers for  $q_1$  and  $q_3$  act as cell boundaries for  $q_2$ .

Using staggered grids improves the accuracy of the algorithm and it also solves the odd-even decoupling problem. It is also more stable numerically than cell-centered algorithms. Where cell-centered algorithms sometimes crash when the algorithm is stretched too far, a staggered grid algorithm usually survives those conditions.

→ **Exercise:** Show, using the example above, that a staggered grid does not have the odd-even decoupling problem.

A drawback of staggered grids is that it requires a much more complex book-keeping of the cells and interfaces, especially when one goes to multiple dimensions. This also could make it slightly more difficult to implement adaptive mesh refinement methods and such.

A number of hydrodynamics codes use staggered grids. For instance, the famous ZEUS code, often used in astrophysics, is based on staggered grids. But also many code are based on cell-centered schemes. Both approaches have their special advantages and disadvantages.

## 5.9 External gravity force

It possible, without much effort, to add an external force to the hydrodynamics scheme we have produced so far. For instance, if we wish to model the flow of gas in a gravity field, we need to solve the following equations (1-D for simplicity):

$$\partial_t \rho + \partial_x(\rho u) = 0 \quad (5.33)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + P) = -\rho \partial_x \Phi \quad (5.34)$$

$$\partial_t(\rho e_{\text{tot}}) + \partial_x[(\rho e_{\text{tot}} + P)u] = -\rho u \partial_x \Phi \quad (5.35)$$

where  $\Phi(x)$  is the gravitational potential. The  $-\rho \partial_x \Phi$  term on the rhs of the momentum equation is the force, whereas the  $-\rho u \partial_x \Phi$  term on the rhs of the energy equation is the work done by the gravity on the *total* energy. The idea behind the latter term is that even though the gravity force does not directly affect the thermal energy of the gas, it *does* affect the *total* energy of the gas: the kinetic energy  $e_{\text{kin}} = u^2/2$  changes due to a change in  $u$ . The source term in the energy equation takes care of this.

An example of an application of this set of equations is that of a perturbation on an otherwise hydrostatic atmosphere on the surface of a planet (such as Earth). Let us define  $x$  to be the height above the surface of the planet and the gravity potential  $\Phi(x) = g x$ . A static solution must be such that the  $\partial_t$  terms of the above equations will be identically zero. Also we must have, by definition,  $u = 0$  everywhere. The continuity equation is then automatically solved. The momentum equation, on the other hand, becomes:

$$\frac{\partial P}{\partial x} = -\rho \frac{\partial \Phi}{\partial x} \quad (5.36)$$

In discrete form this becomes:

$$\frac{P_{i+1} - P_{i-1}}{x_{i+1} - x_{i-1}} = -\rho_i \frac{\Phi_{i+1} - \Phi_{i-1}}{x_{i+1} - x_{i-1}} \quad (5.37)$$

So if we can construct a static atmosphere which obeys this equation, then the numerical hydrodynamics algorithm will produce perfectly zero time derivatives. In other words: it has ‘recognized a static solution’. We can then add a small perturbation on this solution and see how it moves.

## 5.10 Alternative methods for the energy equation

### 5.10.1 Including the gravitational potential into the total energy

If gravity forces are included, the global energy conservation is not guaranteed anymore. Some codes therefore include the potential  $\Phi$  into the total energy:  $e_{\text{tot}} = e_{\text{th}} + u^2/2 + \Phi$ . In that case one can derive that the energy equation becomes

$$\partial_t[\rho(e_{\text{th}} + u^2/2 + \Phi)] + \partial_x[\rho(e_{\text{th}} + u^2/2 + \Phi + P/\rho)u] = \rho\partial_t\Phi \quad (5.38)$$

where the only remaining source term on the rhs is the time derivative of the potential. In a static potential this would therefore be zero, and perfect conservation of total energy is again guaranteed, even in the presence of gravity.

### 5.10.2 Using the thermal energy or entropy as the advected variable

A drawback of using the total energy  $\rho e_{\text{tot}} = \rho(e_{\text{th}} + u^2/2)$  (or  $\rho e_{\text{tot}} = \rho(e_{\text{th}} + u^2/2 + \Phi)$  in case of the inclusion of gravity) as the to-be-advected quantity is that this could lead to negative pressures in the simulation, which could crash the simulation. To find the pressure at any time in the algorithm we must first compute the  $u^2/2$  from the velocity, and then subtract this from the total energy. In case of gravity, one should also subtract  $\Phi$ . So we have:

$$e_{\text{th}} = e_{\text{tot}} - u^2/2 - \Phi \quad (5.39)$$

Now suppose we have a situation in which  $e_{\text{th}} \ll u^2/2$  and/or  $e_{\text{th}} \ll \Phi$ , then the value of  $e_{\text{th}}$  is the result of the difference between two large, and nearly equal numbers. Now, the momentum equation (which determines  $u$ ) and the total energy equation (which determines  $e_{\text{tot}}$ ) may always have some small numerical errors. So these errors could then easily lead to an unwanted flip of sign of  $e_{\text{tot}} - u^2/2 - \Phi$ , leading to a negative thermal energy, and hence a negative pressure. This would mean that the program will have to do an emergency stop. Problems of this kind can occur when the flow is extremely supersonic. The total energy is then nearly completely dominated by the kinetic energy. A tiny error in either  $u$  or  $e_{\text{tot}}$  could then lead to negative energies.

For this reason some codes prefer to let go of strict energy conservation and use another form of the energy equation:

$$D_t e_{\text{th}} \equiv \partial_t e_{\text{th}} + u \partial_x e_{\text{th}} = -\frac{P}{\rho} \partial_x u \quad (5.40)$$

which is the Lagrange form of the energy conservation equation (see Chapter 1). The ZEUS code uses this kind of energy equation. The numerical integration of this equation then proceeds with the algorithms of Chapter 3, and the rhs of the equation is added in the usual way. This form of the energy equation has the clear advantage that it is easier to control, and negative energies do not occur unless the time step was taken too big. The drawback is, of course, that energy is now not anymore strictly conserved, and one must always check a-posteriori if the energy error has not increased to unacceptable levels.

An alternative, but rather similar approach is to advect the *entropy* instead of the internal energy:

$$TD_t s \equiv T(\partial_t s + u \partial_x s) = Q \quad (5.41)$$

where  $Q$  is the entropy generation source term. The main advantage is that entropy generation can be strictly controlled. This can be of great advantage, for instance in simulating atmospheres where the entropy gradients decide about convective stability or instability of an atmosphere.

This method of using the entropy equation is used in, for instance, the PENCIL code, which is a code for modeling accretion disks in astrophysics.

If the thermal energy equation or the entropy equation is used instead of the total energy, then the importance of the use of the Von Neumann-Richtmyer artificial viscosity is amplified enormously. This can be easily understood by looking at Eq. (5.41). Suppose we start with an isentropic state everywhere, but we have conditions such that a shock wave forms. A shock wave generates entropy. But if we have  $Q = 0$ , then the entropy equation will not generate entropy. The solution of the state behind the shock will therefore clearly be wrong, and in practice it will generate strong oscillations or other unwanted behavior. Now we *must* include an entropy-generating source term in order to produce the right amount of entropy.

## 5.11 2-D/3-D Hydrodynamics

So far we have focused primarily on 1-D hydrodynamics. In many cases our final interest lies in multi-dimensional gas flow, as this is a more realistic and interesting projection of reality. Fortunately the methods we have covered so far can be relatively easily used for 2-D and 3-D gas flow. This is done with the method of *operator splitting*, in which we apply our algorithms of hydrodynamic integration alternatively in  $x$ ,  $y$  and  $z$  direction. We have applied operator splitting already before in other contexts. Here we use this technique to split the 2-D or 3-D problem into its separate directions (Strang, 1968, SIAM, J. Num. Anal, 5, 506).

There are also computer codes that do not use directional operator splitting. They solve the multi-dimensional problem in one go. One of the advantages of such non-splitting algorithms is that they tend to be less prone to numerical artifacts. But the disadvantage of non-splitting algorithms is that they have to be developed from scratch, i.e. they cannot build on algorithms developed for 1-D. They are therefore usually harder to develop. In this lecture we will therefore exclusively focus on the use of Strang's directional operator splitting.

Let us write the equations of hydrodynamics in 2-D:

$$\partial_t \rho + \partial_x(\rho u) + \partial_y(\rho v) = 0 \quad (5.42)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + P) + \partial_y(\rho uv) = 0 \quad (5.43)$$

$$\partial_t(\rho v) + \partial_x(\rho uv) + \partial_y(\rho v^2 + P) = 0 \quad (5.44)$$

$$\partial_t(\rho e_{\text{tot}}) + \partial_x[(\rho e_{\text{tot}} + P)u] + \partial_y[(\rho e_{\text{tot}} + P)v] = 0 \quad (5.45)$$

The splitting is now that we first solve

$$\partial_t \rho + \partial_x(\rho u) = 0 \quad (5.46)$$

$$\partial_t(\rho u) + \partial_x(\rho u^2 + P) = 0 \quad (5.47)$$

$$\partial_t(\rho v) + \partial_x(\rho uv) = 0 \quad (5.48)$$

$$\partial_t(\rho e_{\text{tot}}) + \partial_x[(\rho e_{\text{tot}} + P)u] = 0 \quad (5.49)$$

for one time step, and then solve

$$\partial_t \rho + \partial_y(\rho v) = 0 \quad (5.50)$$

$$\partial_t(\rho u) + \partial_y(\rho uv) = 0 \quad (5.51)$$

$$\partial_t(\rho v) + \partial_y(\rho v^2 + P) = 0 \quad (5.52)$$

$$\partial_t(\rho e_{\text{tot}}) + \partial_y[(\rho e_{\text{tot}} + P)v] = 0 \quad (5.53)$$

for the same time step. Some codes do first 1/2 a time step in x-direction, then 1 time step in y-direction and finally another 1/2 time step in x-direction. This has a slightly higher accuracy.

If all the quantities live at the grid cell centers (like the algorithms shown in this chapter, i.e. not the staggered grid ones) then one can simplify the system even more, because then the full 2-D problem of  $N_x \times N_y$  grid cells can be split into a set of  $N_y$  1-D problems in x-direction plus a set of  $N_x$  1-D problems in y-direction. We can then simply use the 1-D hydro solver  $N_x$  times in x-direction and  $N_y$  times in y-direction, provided that we include the perpendicular momentum components also into the equation. So for the  $N_y$  1-D problems in x-direction we must include  $\rho v$  as another conserved quantity and likewise for the  $N_x$  1-D problems in y-direction we must include  $\rho u$  as a conserved quantity. Since these perpendicular momentum components act as passive tracers (i.e. have no influence on the flow pattern), the inclusion of these components is nearly trivial. Note that this method of splitting the 2-D problem into sets of 1-D problems is only possible for non-staggered grids. The reason is that if, like the ZEUS code, one uses staggered grids, then the x-momentum lives on  $(i + 1/2, j)$  interfaces while the y-momentum lives on  $(i, j + 1/2)$  interfaces. The two momentum components therefore do not live on gridpoints that are located on the same 1-D line. Therefore, the schemes used in ZEUS, while they use operator splitting, they cannot go this extra step of reducing the problem to separate 1-D problems. For codes where the momenta are at the cell centers this problem is not there and the reduction to 1-D problems is possible.

With cell-centered algorithms the 2-D or 3-D problem is therefore only marginally more complicated (technically) than the 1-D problem. All our effort in creating a 1-D hydro solver therefore pays off: with only a little effort a fully functional multi-dimensional program can be created, with the 1-D hydro subroutine at its basis.

## 5.12 Practical matters: input and output of data

Once we wish to do serious modeling we cannot afford ad-hoc management of time steps and output of data anymore. We must think carefully about how to input and output our data, in particular if we wish to write a work horse application in a fast programming language such as Fortran (either F77, F90 or F95), or C/C++. There are a couple of thoughts which might be of use, but some thoughts apply only to codes that go beyond the simple IDL programming done in this lecture:

1. In case of workhorse codes, it often is useful if the initial density, energy and velocity distribution is read into the code in a file format that is similar to the output file format. In this way an output of the hydro code can later be used as the starting point of a continued simulation.
2. In true applications one rarely wants to store the results of each time step. That is too storage-intensive. One typically sets pre-defined model times at which the code should output a frame. The time stepping is then done using the usual CFL time step limit, but if the time is about to jump over one of the pre-defined save-time, then one shortens the time step such that the precise save time is reached. Then a dump of the current state is made, and the simulation is continued.
3. Especially for 2-D and 3-D simulations a compact file format is useful. A simple and reasonably compact way is to use fortran-style unformatted output. However, one should keep in mind that on some platforms the byte-order is swapped. If a frame written on one

computer looks unintelligible on another computer, then one may want to look into “endian swapping”. In IDL there is a keyword in the file management routines to automatically swap endian. Another, quite popular method of storage is the HDF format. It is very portable to other platforms and is particularly suited for large data volumes.

4. For safety, in particular for large multi-dimensional applications, one might want to set a maximum number of time steps, so that if a simulation gets stuck, it won't continue to eat up all CPU time. Also, for similar safety concerns one might wish to set a maximum nr of steps the code can do without writing a safety dump of its variables. And finally, it is useful to make the code dump the entire state as it was before the time step if an unrecoverable error occurs.

# Chapter 6

## Riemann solvers I

The numerical hydrodynamics algorithms we have devised in Chapter 5 were based on the idea of operator splitting between the advection and pressure force terms. The advection was done, for all conserved quantities, using the gas velocity, while the pressure force and work terms were treated as source terms. From Chapter 2 we know, however, that the characteristics of the Euler equations are not necessarily all equal to the gas velocity. We have seen that there exist an eigenvector which indeed has the gas velocity as eigenvectors,  $\lambda_0 = u$ , but there are two eigenvectors which have eigenvalues  $\lambda_{\pm} = u \pm C_s$  which belong to the forward and backward sound propagation. Mathematically speaking one should do the advection in these three eigenvectors, using their eigenvalues as advection velocity. The methods in Chapter 5 do not do this. By extracting the pressure terms out of the advection part and adding them as a source term, the advection part has been reduced essentially to *Burger's equation*, and the propagation of sound waves is entirely driven by the addition of the source terms. Such methods therefore do not formally propagate the sound waves using advection, even though mathematically they should be. All the effort we have done in Chapters 3 and 4 to create the best advection schemes possible will therefore have no effect on the propagation of sound waves. One could say that for two out of three characteristics our ingenious advection scheme is useless.

Riemann solvers on the other hand keep the pressure terms within the to-be-advection system. There is no pressure source term in these equations. The mathematical character of the equations remains intact. Such solvers therefore propagate all the characteristics on equal footing. We shall see that Riemann solvers are based on the concept of the *Riemann problem*, so we will first dig into this concept. We will then cover the purest version of a Riemann solver: the Godunov solver, but we will then quickly turn our attention to linearized Riemann solvers, which are simpler to program and are conceptually more closely linked to the concept of characteristic transport. Perhaps the most powerful linear Riemann solver is the *Roe solver* which has the particular advantage that it recognizes shock waves and transports all characteristics nicely.

As we shall see, Riemann solvers tend to have advantages, but also some disadvantages. One can therefore not say that they are always the method of choice. However, for problems involving shock waves, contact discontinuities and other high-resolution flow features, Riemann solvers remain unparalleled in keeping these flow features sharp. For that reason they are becoming ever more popular.

Many of the things covered in this chapter and in the next were inspired by the book of Randall LeVeque, “Finite Volume Methods for Hyperbolic Problems”.



## 6.1 Simple waves, integral curves and Riemann invariants

Before we can delve into the concepts of Riemann problems and, lateron, Riemann solvers, we must first solidify our understanding of characteristic families, and the associated concepts of simple waves, integral curves and Riemann invariants. Since these concepts are important conceptually, but not of too great importance quantitatively, we shall remain brief here. Let us recall the following form of the Euler equations (cf. Eq. 6.1):

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2}\rho u^2 & (3-\gamma)u & (\gamma-1) \\ -\{\gamma e_{\text{tot}}u + (\gamma-1)u^3\} & \{\gamma e_{\text{tot}} + \frac{3}{2}(1-\gamma)u^2\} & \gamma u \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = 0 \quad (6.1)$$

where  $q_1 = \rho$ ,  $q_2 = \rho u$  and  $q_3 = \rho e_{\text{tot}}$ . The eigenvalues are

$$\lambda_1 = u - C_s \quad (6.2)$$

$$\lambda_2 = u \quad (6.3)$$

$$\lambda_3 = u + C_s \quad (6.4)$$

with eigenvectors:

$$e_1 = \begin{pmatrix} 1 \\ u - C_s \\ h_{\text{tot}} - C_s u \end{pmatrix} \quad e_2 = \begin{pmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{pmatrix} \quad e_3 = \begin{pmatrix} 1 \\ u + C_s \\ h_{\text{tot}} + C_s u \end{pmatrix} \quad (6.5)$$

where  $h_{\text{tot}} = e_{\text{tot}} + P/\rho$  is the total specific enthalpy and  $C_s = \sqrt{\gamma P/\rho}$  is the adiabatic sound speed.

The definition of the eigenvectors depend entirely and only on the state  $q = (q_1, q_2, q_3)$ , so in the 3-D state-space these eigenvectors set up three vector fields. We can now look for set of states  $q(\xi) = (q_1(\xi), q_2(\xi), q_3(\xi))$  that connect to some starting state  $q_s = (q_{s,1}, q_{s,2}, q_{s,3})$  through integration along one of these vector fields. These constitute *integral curves of the characteristic family*. Two states  $q_a$  and  $q_b$  belong to the same 1-characteristic integral curve, if they are connected via the integral:

$$q_b = q_a + \int_a^b de_1 \quad (6.6)$$

The concept of integral curves can be understood the easiest if we return to linear hyperbolic equations with a constant advection matrix: in that case we could decompose  $q$  entirely in eigen-components. A 1-characteristic integral curve in state-space is a set of states for which only the component along the  $e_1$  eigenvector varies, while the components along the other eigenvectors may be non-zero but should be non-varying. For non-linear equations the decomposition of the full state vector is no longer a useful concept, but the integral curves are the non-linear equivalent of this idea.

Typically one can express integral curves not only as integrals along the eigenvectors of the Jacobian, but also curves for which some special scalars are constant. In the 3-D parameter space of our  $q = (q_1, q_2, q_3)$  state vector each curve is defined by two of such scalars. Such scalar fields are called *Riemann invariants* of the characteristic family. One can regard these integral curves now as the crossing lines between the two contour curves of the two Riemann invariants. The value of each of the two Riemann invariants now identifies each of the characteristic integral curves.



For the eigenvectors of the Euler equations above the Riemann invariants are:

$$\begin{aligned}
 \text{1-Riemann invariants:} & \quad s, \quad u + \frac{2C_s}{\gamma-1} \\
 \text{2-Riemann invariants:} & \quad u, \quad P \\
 \text{3-Riemann invariants:} & \quad s, \quad u - \frac{2C_s}{\gamma-1}
 \end{aligned} \tag{6.7}$$

The 1- and 3- characteristics represent sound waves. Indeed, sound waves (if they do not topple over to become shocks) preserve entropy, and hence the entropy  $s$  is a Riemann invariant of these two families. The 2- characteristic represents an *entropy wave* which means that the entropy can vary along this wave. This is actually not a wave in the way we know it. It is simply the comoving fluid, and adjacent fluid packages may have different entropy. The fact that  $u$  and  $P$  are Riemann invariants of this wave can be seen by integrating the vector  $(1, u, u^2/2)$  in parameter space. One sees that  $\rho$  varies, but  $u$  does not. Also one sees that  $q_3 = \rho(e_{th} + u^2/2)$  varies only in the kinetic energy component. The value  $\rho e_{th}$  stays constant, meaning that the pressure  $P = (\gamma - 1)\rho e_{th}$  remains constant. So while the density may increase along this integral curve, the  $e_{th}$  will then decrease enough to keep the pressure constant. This means that the entropy goes down, hence the term “entropy wave”.

In time-dependent fluid motion a wave is called a *simple wave* if the states along the wave all lie on the same integral curve of one of the characteristic families. One can say that this is then a pure wave in only one of the eigenvectors. A simple wave in the 2-characteristic family is a wave in which  $u = \text{const}$  and  $P = \text{const}$ , but in which the entropy may vary. A simple wave in the 3-characteristic family is for instance an infinitesimally weak sound wave in one direction. In Section 6.3 we shall encounter also another simple wave of the 1- or 3- characteristic family: a *rarefaction wave*.

As we shall see in the section on Riemann problems below, there can exist situations in which two fluids of different entropy lie directly next to each other, causing an entropy jump, but zero pressure or velocity jump. This is also a simple wave in the 2-family, but a special one: a discrete jump wave. This kind of wave is called a *contact discontinuity*.

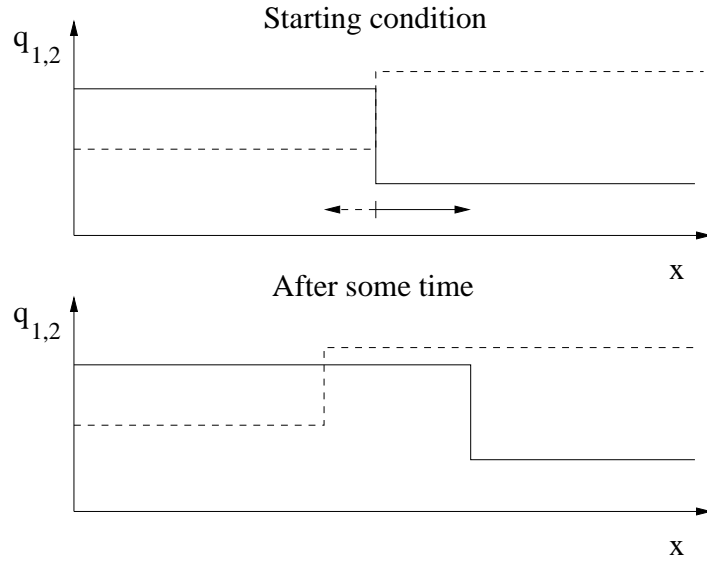
Another jump-like wave is a *shock wave* which can be either from the 1-characteristic family or from the 3-characteristic family. However, shock waves are waves for which the Riemann invariants are no longer perfectly invariant. In particular the entropy will no longer be constant over a shock front. Nevertheless, shock fronts can still be associated to either the 1-characteristic or 3-characteristic family. The states on both sides of the shock front however, do not lie on the same integral curve. They lie instead on a *Hugoniot locus*.

## 6.2 Riemann problems

A Riemann problem in the theory of hyperbolic equations is a problem in which the initial state of the system is defined as:

$$q(x, t = 0) = \begin{cases} q_L & \text{for } x \leq 0 \\ q_R & \text{for } x > 0 \end{cases} \tag{6.8}$$

In other words: the initial state is constant for all negative  $x$ , and constant for all positive  $x$ , but differs between left and right. For hydrodynamic problems one can consider this to be a 1-D hydrodynamics problem in which gas with one temperature and density is located to the left of a removable wall and gas with another temperature and density to the right of that wall. At time  $t = 0$  the wall is instantly removed, and it is watched what happens.



**Figure 6.1.** Example of the solution of a linear Riemann problem with constant and diagonal advection matrix. Top: initial condition (solid line is  $q_1$ , dashed line is  $q_2$ ). Bottom: after some time, the  $q_1$  component has moved to the right ( $\lambda_1 > 0$ ) while the  $q_2$  component has moved to the left ( $\lambda_2 < 0$ ).

For hydrodynamic problems such *shock tube tests* are used to test the performance of numerical hydrodynamics algorithms. This was first done by (Sod 1978, J. Comp. Phys 27, 1), hence the name *Sod shock tube tests*. But such tests were also carried out in the laboratory (see e.g. the book by Liepmann & Roshko).

### 6.2.1 Riemann problems for linear advection problems

The simplest Riemann problems are those of linear advection problems with constant advection velocity, or constant Jacobian matrix. Consider the following equation:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (6.9)$$

Consider the following Riemann problem for this set of equations:

$$q_1(x, 0) = \begin{cases} q_{1,l} & \text{for } x < 0 \\ q_{1,r} & \text{for } x > 0 \end{cases} \quad (6.10)$$

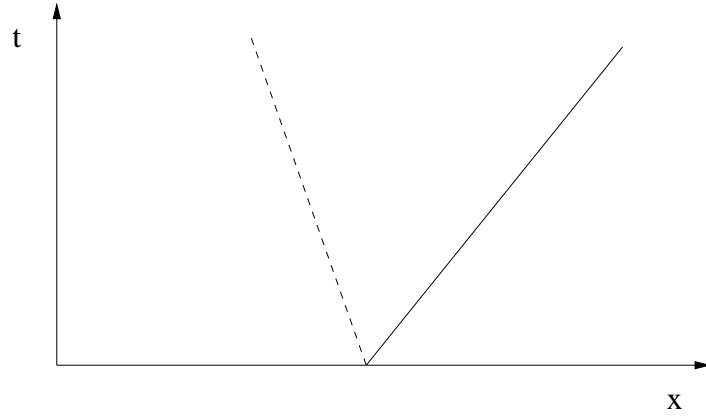
$$q_2(x, 0) = \begin{cases} q_{2,l} & \text{for } x < 0 \\ q_{2,r} & \text{for } x > 0 \end{cases} \quad (6.11)$$

Clearly the solution is:

$$q_1(x, t) = q_1(x - \lambda_1 t, 0) \quad (6.12)$$

$$q_2(x, t) = q_2(x - \lambda_2 t, 0) \quad (6.13)$$

which is simply that the function  $q_1(x)$  is shifted with velocity  $\lambda_1$  and the function  $q_2(x)$  is shifted with velocity  $\lambda_2$ . An example is shown in Fig. 6.1 A very similar solution is found if the matrix is not diagonal, but has real eigenvalues: we then simply decompose  $(q_1, q_2)$  into eigenvectors, obtaining  $(\tilde{q}_1, \tilde{q}_2)$ , shift  $\tilde{q}_1$  and  $\tilde{q}_2$  according to their own advection velocity (eigenvalue of the matrix), and then reconstruct the  $q_1$  and  $q_2$  from  $\tilde{q}_1$  and  $\tilde{q}_2$ .



**Figure 6.2.** The characteristics of the problem solved in Fig. 6.1.

→ **Exercise:** Solve in this way the general Riemann problem for the equation

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (6.14)$$

These examples are for hyperbolic equations with two characteristics, but this procedure can be done for any number of characteristics.

Note that if we look at this problem in an  $(x, t)$  diagram, then we see two waves propagating, one moving with velocity  $\lambda_1$  and one with velocity  $\lambda_2$ . We also see that the solution is self-similar:

$$q_{1,2}(x, t_b) = q_{1,2}(xt_a/t_b, t_a) \quad (6.15)$$

### 6.3 Riemann problems for the equations of hydrodynamics

Riemann problems for the Euler equations are much more complex than those for the simple linear hyperbolic equations shown above. This is because of the strong non-linearity of the equations. A Riemann problem for the equations of hydrodynamics is defined as:

$$\rho, u, P = \begin{cases} \rho_l, u_l, P_l & \text{for } x < 0 \\ \rho_r, u_r, P_r & \text{for } x > 0 \end{cases} \quad (6.16)$$

The general solution is quite complex and even the qualitative shape of the solution depends strongly on the Riemann problem at hand. In this section we will discuss two special cases.

#### 6.3.1 Special case: The converging flow test

The simplest Riemann problem for the hydrodynamic equation is that in which  $P_l = P_r$ ,  $\rho_l = \rho_r$  and  $u_l = -u_r$  with  $u_l > 0$ . This is a symmetric case in which the gas on both sides of the dividing line are moving toward each other: a converging flow. From intuition and/or from numerical experience it can be said that the resulting solution is a compressed region that is expanding in the form of two shock waves moving away from each other. Without a-priori proof (we shall check a-posteriori) let us assume that the compressed region in between the two shock waves has a constant density and pressure, and by symmetry has a zero velocity. We also assume that the converging gas that has not yet gone through the shock front is undisturbed.

The problem we now have to solve is to find the shock velocity  $v_s$  (which is the same but opposite in each direction) and the density and pressure in the compressed region:  $\rho_c, P_c$ . For a

given  $v_s$  the Mach number  $\mathcal{M}$  of the shock is:

$$\mathcal{M} = \frac{u_l + v_s}{C_{s,l}} = (u_l + v_s) \sqrt{\frac{P_l}{\gamma \rho_l}} \quad (6.17)$$

(we take by definition  $v_s > 0$ ). We now need the Rankine-Hugoniot adiabat in the form of Eq. (1.97),

$$\frac{\rho_l}{\rho_c} = \frac{(\gamma - 1)\mathcal{M}^2 + 2}{(\gamma + 1)\mathcal{M}^2} = \frac{\gamma - 1}{\gamma + 1} + \frac{2}{(\gamma + 1)\mathcal{M}^2} \quad (6.18)$$

as well as the condition for mass conservation

$$\rho_l(u_l + v_s) = \rho_c v_s \quad (6.19)$$

By writing  $v_s = v_s + u_l - u_l = (\mathcal{M} - u_l/C_{s,l})C_{s,l}$  in the latter equation we can eliminate  $\rho_l/\rho_c$  in both equations to obtain

$$\frac{\gamma - 1}{\gamma + 1} \mathcal{M}^2 + \frac{2}{\gamma + 1} = \mathcal{M}^2 - \frac{u_l}{C_{s,l}} \mathcal{M} \quad (6.20)$$

which reduces to

$$\mathcal{M}^2 - \frac{\gamma + 1}{2} \frac{u_l}{C_{s,l}} \mathcal{M} - 1 = 0 \quad (6.21)$$

The solution is:

$$\mathcal{M} = \frac{1}{2} \left\{ \left( \frac{\gamma + 1}{2} \right) \frac{u_l}{C_{s,l}} \pm \sqrt{\frac{(\gamma + 1)^2}{4} \frac{u_l^2}{C_{s,l}^2} + 4} \right\} \quad (6.22)$$

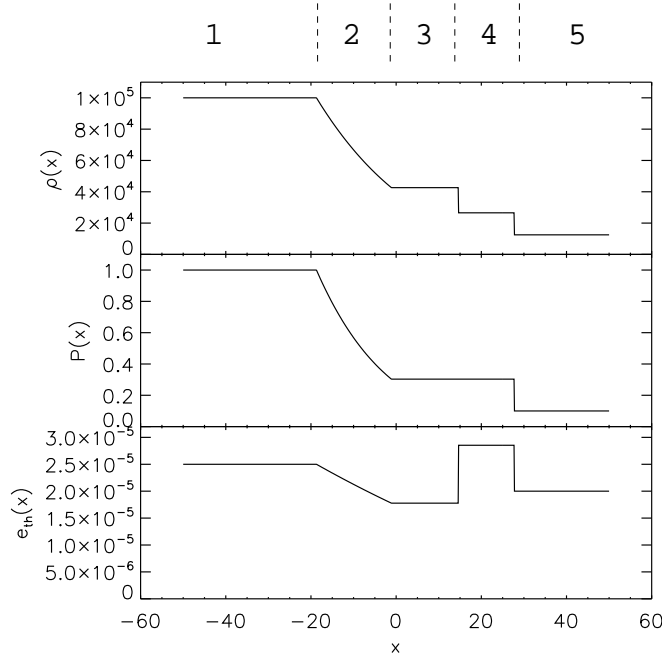
For our purpose we need to choose the positive root. One sees that there is always a solution, and that one can find two limits:

- Limit 1,  $u_l \ll C_{s,l}$ : The solution is  $\mathcal{M} = 1$ . This means that in this limit the shock wave reduces to a sound wave.
- Limit 2,  $u_l \gg C_{s,l}$ : The solution is  $\mathcal{M} = (u_l/C_{s,l})(\gamma + 1)/2$ . This is the strong shock limit: the compression reaches its maximum of  $\rho_c \rightarrow \rho_l(\gamma + 1)/(\gamma - 1)$ . The strong shock limit is the limit in which the pre-shock thermal energy is so small compared to the post-shock value that it can be regarded as being zero.

### 6.3.2 Special case: Sod's shock tube tests

A special case of a Riemann problem in Eulerian hydrodynamics is when the initial state has zero velocity on both sides of the dividing point, but the pressure has a jump. We shall discuss these solutions here following the book by Bodenheimer et al. (2007). The complete solution of the Sod shock tube test is rather difficult to derive, so here we shall derive only the most obvious relations, and write down other relations without derivation.

The most important first step is to find out what the qualitative form of the solution is. Here we rely on experience: If one solves such problems using numerical hydrodynamics or laboratory experiments one finds that the self-similar solution that follows from such a problem typically has 5 regions which we shall call region 1,2,3,4,5 as depicted in Fig. 6.3. Region 1 and 5 have states which correspond to the left and the right initial state respectively. Regions 3 and 4 have steady states (independent of  $x$  within the region) and region 2 has an  $x$ -dependent state. This



**Figure 6.3.** The solution to the shock tube problem of Sod for  $\gamma = 7/5$ ,  $\rho_l = 10^5$ ,  $P_l = 1$ ,  $\rho_r = 1.25 \times 10^4$  and  $P_r = 0.1$ , shown at time  $t = 5000$ . The regions 1 to 5, as mentioned in the text, are annotated at the top.

region 2 represents an *expansion wave*, also called *rarefaction wave*. This is a simple wave of the left-going characteristic family (the 1-characteristic family in the terminology of Section 6.1). It is the only non-constant region in the solution. The dividing line between region 3 and 4 is a *contact discontinuity*, i.e. a line separating two fluids of different entropy but the same pressure and the same velocity. This is a “wave” of the middle characteristic family (the 2-characteristic family in the terminology of Section 6.1). Therefore  $u_3 = u_4$  and  $P_3 = P_4$ . The propagation speed of the contact discontinuity is therefore also  $u_c = u_4$  and the location of this discontinuity at some time  $t$  is  $x_{\text{contact}} = u_c t$ . Regions 4 and 5 are separated by a forward moving shock wave. This is a jump in the forward moving characteristic family (the 3-characteristic family in the terminology of Section 6.1). Since  $u_5 = 0$  one can invoke mass conservation to write the shock propagation speed  $u_s$  in terms of the velocity  $u_4$  and the densities in both regions:

$$u_s = u_4 \frac{\rho_4}{\rho_4 - \rho_5} \quad (6.23)$$

The location of the shock wave at time  $t$  is therefore  $x_{\text{shock}} = u_s t$ . According to the Rankine-Hugoniot conditions derived in Section 1.9 we can also relate the density ratio and the pressure ratio over the shock:

$$\frac{\rho_4}{\rho_5} = \frac{P_4 + m^2 P_5}{P_5 + m^2 P_4} \quad (6.24)$$

where  $m^2 = (\gamma - 1)/(\gamma + 1)$ . From these relations we can derive the velocity in region 4, because we know that  $u_5 = 0$ . We obtain

$$u_4 = (P_4 - P_5) \sqrt{\frac{1 - m^2}{\rho_5 (P_4 + m^2 P_5)}} \quad (6.25)$$

This is about as far as we get on the shock front. Let us now focus on the expansion wave (region 2). The leftmost onset of the expansion wave propagates to the left with the local sound speed. So we have, at some time  $t$ , this point located at  $x_{\text{wave}} = -C_{s,1}t$ , where  $C_{s,1} = \sqrt{\gamma P_1/\rho_1}$  is the sound speed in region 1. Without further derivation (see Hawley et al. 1984) we write that the gas velocity in region 2 can be expressed as

$$u_2 = \sqrt{\frac{(1 - m^4)P_1^{1/\gamma}}{m^4\rho_1}} \left( P_1^{\frac{\gamma-1}{2\gamma}} - P_2^{\frac{\gamma-1}{2\gamma}} \right) \quad (6.26)$$

To find the dividing line between regions 2 and 3 we now solve the equation  $u_2 = u_4$ , i.e. Eq.(6.26) – Eq.(6.25) = 0, for the only remaining unknown  $P_3$ . We do this using a numerical root-finding method, for instance the `zbrent` method of the book *Numerical Recipes* by Price et al.. This will yield us a numerical value for  $P_3 = P_4$ . Then Eq.(6.25) directly leads to  $u_c = u_3 = u_4$ . The Hugoniot adiabat of the shock (Eq. 6.24) now gives us  $\rho_4$ . Now with Eq. (6.23) we can compute the shock velocity  $u_s$ , and thereby the location of the shock front  $x_{\text{shock}} = u_s t$ . The density in region 3 can be found by realizing that none of the gas to the left of the contact discontinuity has ever gone through a shock front. It must therefore still have the same entropy as the gas in region 1. Using the law for polytropic gases  $P = K\rho^\gamma$  we can say that  $K$  is the same everywhere left of the contact discontinuity (i.e. in regions 1,2 and 3). Therefore we can write that  $\rho_3 = \rho_1(P_3/P_1)^{1/\gamma}$ . At this point we know the density, the pressure and the gas velocity in regions 1,3,4,5. We can therefore easily calculate any of the other quantities in these regions, such as the sound speed  $C_s$  or the internal thermal energy  $e_{\text{th}}$ . The remaining unknown region is region 2, and we also do not yet know the location of the separation between regions 2 and 3. Without derivation (see Hawley et al. 1984) we state that in region 2:

$$u(x, t) = (1 - m^2) \left( \frac{x}{t} + C_{s,1} \right) \quad (6.27)$$

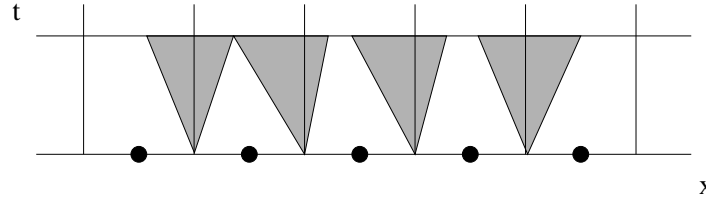
which indeed has the property that  $u = 0$  at  $x = -C_{s,1}t$ . We can find the location of the separation between regions 2 and 3 by numerically solving  $u(x, t) = u_3$  for  $x$ . The expression for the sound speed  $C_s(x, t)$  in region 2 is now derived by noting that region 2 is a classical *expansion fan*, in which the left-moving characteristic  $\lambda_-$  must, by nature of self-similar solutions, have the form  $\lambda_- \equiv u(x, t) - C_s(x, t) = x/t$ . This yields for region 2:

$$C_s^2(x, t) \equiv \gamma \frac{P(x, t)}{\rho(x, t)} = \left( u(x, t) - \frac{x}{t} \right)^2 \quad (6.28)$$

Also here we know that  $P(x, t) = K\rho(x, t)^\gamma$  with the same  $K$  as in region 1. Therefore we obtain:

$$\rho(x, t) = \left[ \frac{\rho_1^\gamma}{\gamma P_1} \left( u(x, t) - \frac{x}{t} \right)^2 \right]^{1/(\gamma-1)} \quad (6.29)$$

from which  $P(x, t)$  can be directly derived using  $P(x, t) = K\rho(x, t)^\gamma$ , and the sound speed and internal thermal energy follow then immediately. We now have the total solution complete, and for a particular example this solution is shown in Fig. 6.3. Later in this chapter we shall use these solutions as simple test cases to verify the accuracy and performance of hydrodynamic algorithms.



**Figure 6.4.** Godunov's method: solving a self-similar Riemann problem at each interface (grey), and making sure that the time step is small enough that they do not overlap. The two leftmost self-similar Riemann solutions just manage to touch by the end of the time step, which means that the time step can not be made larger before they will interfere.

## 6.4 Godunov's method

We can now apply what we learned about the solution of Riemann problems to devise a new numerical method for numerical hydrodynamics. Consider our numerical solution at some time  $t_n$  to be given by  $q_i^n$ . These are values of  $q$  given at the cell centers located at  $x = x_i$ . We define cell interfaces  $x_{i+1/2}$  in the usual way (see Chapter 4) to be located in between the cell centers  $x_i$  and  $x_{i+1}$ . As our subgrid model we assume that at the start of the time step the state within each cell is strictly constant (piecewise constant method, see Chapter 4). At each interface the state variables now describe a jump. If we zoom in to the region around this interface we see that this is precisely the definition of a Riemann problem, but this time locally within the two adjacent cells. We can now calculate what the self-similar solution of the Riemann problem at each cell interface  $i + 1/2$  would be. This is a subgrid analytic evolution of the hydrodynamic system *within each pair of cells*. This self-similar solution is calculated at each interface, so in order to preserve the self-similar character of these solutions we must prevent the solutions from two adjacent interfaces to overlap. This is depicted in Fig. 6.4. The time step is therefore restricted to

$$\Delta t \leq \min(\Delta t_i) \quad (6.30)$$

where

$$\Delta t_i = \frac{x_{i+1/2} - x_{i-1/2}}{\max(\lambda_{i-1/2,k+}) - \min(\lambda_{i+1/2,k-})} \quad (6.31)$$

where  $\lambda_{i-1/2,k+}$  denotes the maximum positive eigenvalue at interface  $i - 1/2$ , and will be 0 in case no positive eigenvalues exist at that interface. Likewise  $\lambda_{i+1/2,k-}$  denotes the smallest (i.e. most negative) negative eigenvalue at interface  $i + 1/2$ , or 0 if no negative eigenvalues exist.

How to proceed from here, i.e. how to create a numerical algorithm from this concept, can be seen in two different way, which we will highlight in the two next subsections.

### 6.4.1 One way to look at Godunov's method

At the end of the time step each cell  $i$  consists of three regions: a left region which is affected by the Riemann solution at interface  $i - 1/2$ , a middle region which is not yet affected, and a right region which is affected by the Riemann solution at interface  $i + 1/2$ . Since we know the (semi-)analytic solutions of the Riemann problems and we of course know the unaffected state in the middle region, we can (semi-)analytically average all state variables over the cell. This averaging then results in the cell-center value of  $q_i^{n+1}$ . This averaging procedure is very similar to what was done in the donor-cell algorithm, but this time the state in the cell at the end of the time step is far more complex than in the simple donor-cell algorithm. Because of this complexity we shall not work this out in this chapter.



### 6.4.2 Another way to look at Godunov's method

Another way to look at Godunov's method is by looking at the flux at the interface. We know that the Riemann solutions around the cell interfaces are self-similar in the dimensionless space variable  $\xi = (x - x_{i+1/2})/(t - t_n)$ . This means that the state at the interface in this solution is constant in time (at least between  $t = t_n$  and  $t = t_{n+1}$ ). This then implies that the flux  $f_{i+1/2}$  is also constant in this time interval. We can therefore then write:

$$q_i^{n+1} = q_i^n - \Delta t \frac{f_{i+1/2}^n - f_{i-1/2}^n}{x_{i+1/2} - x_{i-1/2}} \quad (6.32)$$

where  $f_{i+1/2}^n$  and  $f_{i-1/2}^n$  are the fluxes calculated from the Riemann problems at the cell interfaces. Note that this is true as much for linear sets of hyperbolic equations as well as for non-linear ones. The complexity still remains in determining the state in the Riemann problem at the cell interfaces, but that is already much less difficult than determining the entire Riemann solution and averaging over it. Nevertheless for hydrodynamics the method remains complex and we will therefore not go into the Godunov method for these equations.

## 6.5 Godunov for linear hyperbolic problems: a characteristic solver

### 6.5.1 Example for two coupled equations

Instead of demonstrating how a Godunov solver works for the full non-linear set of equations of hydrodynamics, we show here how it works for linear hyperbolic sets of equations. The nice thing is that in this case the Riemann problem at each cell interface can be solved analytically. Moreover, we will then naturally be led to a new concept: that of a *characteristic solver*. For linear problems Riemann solvers and characteristic solvers are identical. Later, when dealing with the full set of non-linear hydrodynamics equations, we shall be using both concepts.

Let us consider the following equation:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \begin{pmatrix} a & b \\ c & d \end{pmatrix} \partial_x \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = 0 \quad (6.33)$$

where the matrix is diagonalizable and has two real eigenvalues. We wish to solve this numerically. Since the advection matrix, in this example, is constant, we were able to bring it out of the  $\partial_x$  operator without flux conservation violation. The way we proceed is first to define the state vector on the left- and right- side of the interface  $i + 1/2$ :

$$q_{i+1/2,L} \equiv q_i \quad (6.34)$$

$$q_{i+1/2,R} \equiv q_{i+1} \quad (6.35)$$

Now define the eigenvalues and eigenvectors of the problem:

$$\lambda_{(\pm)} = \frac{1}{2} \left\{ (a + d) \pm \sqrt{(a - d)^2 + 4bc} \right\} \quad (6.36)$$

and

$$e_{(\pm)} = \begin{pmatrix} \lambda_{(\pm)} + 2d \\ 2c \end{pmatrix} \quad (6.37)$$

Note that we use indices  $(-)$  and  $(+)$  because in this special case the eigenvalues are clearly identifiable with left- and right-moving characteristics unless the problem is "supersonic" in that



both eigenvalues are negative or both are positive. In general we would simply use  $\lambda_1, \lambda_2$  etc. This is just a notation issue. Now, any state

$$q = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \quad (6.38)$$

can be decomposed into these eigenvectors:

$$\tilde{q}_{(-)} = \frac{1}{\lambda_{(+)} - \lambda_{(-)}} \left\{ \frac{\lambda_{(+)} + 2d}{2c} q_2 - q_1 \right\} \quad (6.39)$$

$$\tilde{q}_{(+)} = \frac{1}{\lambda_{(-)} - \lambda_{(+)}} \left\{ \frac{\lambda_{(-)} + 2d}{2c} q_2 - q_1 \right\} \quad (6.40)$$

So we can define the decomposed state on each side of the interface:

$$\tilde{q}_{(-),i+1/2,L} = \tilde{q}_{(-),i} \quad (6.41)$$

$$\tilde{q}_{(+),i+1/2,L} = \tilde{q}_{(+),i} \quad (6.42)$$

$$\tilde{q}_{(-),i+1/2,R} = \tilde{q}_{(-),i+1} \quad (6.43)$$

$$\tilde{q}_{(+),i+1/2,R} = \tilde{q}_{(+),i+1} \quad (6.44)$$

Now we can construct the flux at the interface. Suppose that  $\lambda_1 > 0$ , then clearly the flux for  $\tilde{q}_{(+),i+1/2}$  is determined solely by  $\tilde{q}_{(+),i+1/2,L}$  and not by  $\tilde{q}_{(+),i+1/2,R}$  (the upwind principle):

$$\tilde{f}_{(-),i+1/2} = \begin{cases} \lambda_{(-)} \tilde{q}_{(-),i+1/2,L} & \text{for } \lambda_{(-)} > 0 \\ \lambda_{(-)} \tilde{q}_{(-),i+1/2,R} & \text{for } \lambda_{(-)} < 0 \end{cases} \quad (6.45)$$

and similar for  $\tilde{f}_{(+),i+1/2}$ . The total flux for  $q$  is then:

$$f_{i+1/2} = \tilde{f}_{(-),i+1/2} e_{(-)} + \tilde{f}_{(+),i+1/2} e_{(+)} \quad (6.46)$$

We see that Godunov's method for linear advection equations is nothing else than the donor-cell algorithm applied to each characteristic separately.

We can generalize this method to non-constant advection matrix. Consider the following problem:

$$\partial_t \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} + \partial_x \left[ \begin{pmatrix} a(x) & b(x) \\ c(x) & d(x) \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \right] = 0 \quad (6.47)$$

The procedure is now the same, except that we must do the eigenvector decomposition with the *local* matrix at the interface  $i + 1/2$ . Both the eigenvectors and the eigenvalues are now local to the interface, and so will the decomposition be. In this case  $\tilde{q}_{(-),i+1/2,L} \neq \tilde{q}_{(-),i-1/2,R}$ , while in the case of constant matrix we had  $\tilde{q}_{(-),i+1/2,L} = \tilde{q}_{(-),i-1/2,R}$ . For the rest we construct the fluxes in the same way as above for the constant matrix.

We see that in the simple case of linear advection problems, the Godunov method (based on the Riemann problem) is actually nothing else than a *characteristic solver*: the problem is decomposed into characteristics, which are advected each in their own directions. Indeed, for linear problems the principle of using Riemann problems at each interface to perform the numerical integration of the equations is identical to the principle of decomposing into the eigenvectors of the Jacobian and advecting each component with its own eigenvalue as characteristic speed. In other words: *For linear problems a Riemann solver is identical to a characteristic solver*. This is not true anymore for non-linear problems: as we shall see later on, a characteristic solver for hydrodynamics is not a true Riemann solver but an *approximate Riemann solver* or equivalently a *linearized Riemann solver*. However, let us, for now, stick to linear problems a bit longer.

### 6.5.2 General expressions for Godunov solvers for linear problems

We can make a general expression for the flux, for any number of characteristics:

$$f_{i+1/2} = \sum_{k=1 \dots K} \tilde{f}_{k,i+1/2} e_k \quad (6.48)$$

where  $K$  is the number of characteristics (i.e. number coupled equations, or number of eigenvectors and eigenvalues), and where

$$\tilde{f}_{k,i+1/2} = \frac{1}{2} \lambda_k \left[ (1 + \theta_k) \tilde{q}_i^n + (1 - \theta_k) \tilde{q}_{i+1}^n \right] \quad (6.49)$$

where  $\theta_k = 1$  if  $\lambda_k > 0$  and  $\theta_k = -1$  if  $\lambda_k < 0$ . This is identical to the expressions we derived in Subsection 6.5, but now more general.

### 6.5.3 Higher order Godunov scheme

As we know from Chapter 4, the donor-cell algorithm is not the most sophisticated advection algorithm. We therefore expect the solutions based on Eq. (6.49) to be smeared out quite a bit. In Chapter 4 we found solutions to this problem by dropping the condition that the states are piecewise constant (as we have done in the Godunov scheme so far) and introduce a piecewise linear subgrid model, possibly with a flux limiter. In principle, for linear problems the Godunov scheme is identical to the advection problem for each individual characteristic, and therefore we can apply such linear subgrid models here too. In this way we generalize the Godunov scheme in such a way that we have a slightly more complex subgrid model in each cell (i.e. non-constant), but the principle remains the same. At each interface we define  $\tilde{r}_{k,i-1/2}$ :

$$\tilde{r}_{k,i-1/2}^n = \begin{cases} \frac{\tilde{q}_{k,i-1}^n - \tilde{q}_{k,i-2}^n}{\tilde{q}_{k,i}^n - \tilde{q}_{k,i-1}^n} & \text{for } \lambda_{k,i-1/2} \geq 0 \\ \frac{\tilde{q}_{k,i+1}^n - \tilde{q}_{k,i}^n}{\tilde{q}_{k,i}^n - \tilde{q}_{k,i-1}^n} & \text{for } \lambda_{k,i-1/2} \leq 0 \end{cases} \quad (6.50)$$

where again  $k$  denotes the eigenvector/-value, i.e. the characteristic. We can now define the flux limiter  $\phi(\tilde{r}_{k,i-1/2})$  for each of these characteristics according to the formulae in Section 4.5 (i.e. Eqs. 4.39, 4.40, 4.41). Then the flux is given by (cf. Eq. 4.38)

$$\begin{aligned} \tilde{f}_{k,i-1/2}^{n+1/2} = & \frac{1}{2} \lambda_{k,i-1/2} \left[ (1 + \theta_{k,i-1/2}) \tilde{q}_{k,i-1/2,L}^n + (1 - \theta_{k,i-1/2}) \tilde{q}_{k,i-1/2,R}^n \right] + \\ & \frac{1}{2} |\lambda_{k,i-1/2}| \left( 1 - \left| \frac{\lambda_{k,i-1/2} \Delta t}{\Delta x} \right| \right) \phi(\tilde{r}_{k,i-1/2}^n) (\tilde{q}_{k,i-1/2,R}^n - \tilde{q}_{k,i-1/2,L}^n) \end{aligned} \quad (6.51)$$

It is useful, for later, to derive an alternative form of this same equation, which can be obtained with a bit of algebraic manipulation starting from Eq. (6.51). We use the identity  $|\lambda_{k,i-1/2}| = \theta_{k,i-1/2} \lambda_{k,i-1/2}$  and the definition  $\epsilon_{k,i-1/2} \equiv \lambda_{k,i-1/2} \Delta t / (x_i - x_{i-1})$  and obtain:

$$\begin{aligned} \tilde{f}_{k,i-1/2}^{n+1/2} = & \frac{1}{2} \lambda_{k,i-1/2} (\tilde{q}_{k,i-1/2,R}^n + \tilde{q}_{k,i-1/2,L}^n) \\ & - \frac{1}{2} \lambda_{k,i-1/2} (\tilde{q}_{k,i-1/2,R}^n - \tilde{q}_{k,i-1/2,L}^n) [\theta_{k,i-1/2} + \tilde{\phi}_{k,i-1/2} (\epsilon_{k,i-1/2} - \theta_{k,i-1/2})] \end{aligned} \quad (6.52)$$

where  $\tilde{\phi}_{k,i-1/2} \equiv \phi(\tilde{r}_{k,i-1/2})$ . If we define the decomposed fluxes at the left and right side of the interface as

$$\tilde{f}_{k,i-1/2,L} = \lambda_{k,i-1/2} \tilde{q}_{k,i-1/2,L} \quad (6.53)$$

$$\tilde{f}_{k,i-1/2,R} = \lambda_{k,i-1/2} \tilde{q}_{k,i-1/2,R} \quad (6.54)$$

then we obtain:

$$\begin{aligned} \tilde{f}_{k,i-1/2}^{n+1/2} = & \frac{1}{2}(\tilde{f}_{k,i-1/2,R}^n + \tilde{f}_{k,i-1/2,L}^n) \\ & - \frac{1}{2}(\tilde{f}_{k,i-1/2,R}^n - \tilde{f}_{k,i-1/2,L}^n)[\theta_{k,i-1/2} + \tilde{\phi}_{k,i-1/2}(\epsilon_{k,i-1/2} - \theta_{k,i-1/2})] \end{aligned} \quad (6.55)$$

Now we can arrive at our final expression by adding up all the partial fluxes (i.e. the fluxes of all eigen-components):

$$\begin{aligned} f_{i-1/2}^{n+1/2} = & \frac{1}{2}(f_{i-1/2,R}^n + f_{i-1/2,L}^n) \\ & - \frac{1}{2} \sum_{k=1 \dots K} (\tilde{f}_{k,i-1/2,R}^n - \tilde{f}_{k,i-1/2,L}^n)[\theta_{k,i-1/2} + \tilde{\phi}_{k,i-1/2}(\epsilon_{k,i-1/2} - \theta_{k,i-1/2})] \end{aligned} \quad (6.56)$$

This is our final expression for the (time-step-averaged) interface flux.

There are a number of things we can learn from this expression:

1. The interface flux is the simple average flux plus a diffusive correction term. All the ingenuity of the characteristic solver lies in the diffusive correction term.
2. The flux limiter can be seen as a switch between donor-cell ( $\tilde{\phi} = 0$ ) and Lax-Wendroff ( $\tilde{\phi} = 1$ ), where we here see yet again another interpretation of Lax-Wendroff: the method in which the interface flux is found using a linear upwind interpolation (in contrast to upwinding, where the new *state* is found using linear upwind interpolation). Of course, if  $\tilde{\phi}$  is one of the other expressions from Section 4.5, we get the various other schemes.

In all these derivations we must keep in mind the following caveats:

- Now that the states in the adjacent cells is no longer constant, the Riemann problem is no longer self-similar: The flux at the interface changes with time.
- In case the advection matrix is non-constant in space, the determination of the slopes becomes a bit less mathematically clean: Since the eigenvectors now change from one cell interface to the next, the meaning of  $\tilde{q}_{k,i+1/2,R} - \tilde{q}_{k,i+1/2,L}$  is no longer identical to the meaning of  $\tilde{q}_{k,i-1/2,R} - \tilde{q}_{k,i-1/2,L}$ . Although the method works well, the mathematical foundation for this method is now slightly less strong.

# Chapter 7

## Riemann solvers II

### 7.1 Roe's linearized Riemann solver

#### 7.1.1 The equations of hydrodynamics revisited

Before we construct our linearized Riemann solver, let us make a slightly modified definition of the state vector  $q$ , which allows us an easy generalization of our algorithms to 3-D. We define  $q$  to be

$$q = \begin{pmatrix} \rho e_{\text{tot}} \\ \rho u \\ \rho v \\ \rho w \\ \rho \end{pmatrix} \quad (7.1)$$

For convenience we shall index it from 0 to 4:

$$q_0 = \rho e_{\text{tot}} \quad q_1 = \rho u \quad q_2 = \rho v \quad q_3 = \rho w \quad q_4 = \rho \quad (7.2)$$

The full set of equations for 3-D hydrodynamics is then:

$$\partial_t q + \partial_x f_x(q) + \partial_y f_y(q) + \partial_z f_z(q) = 0 \quad (7.3)$$

where

$$f_x = \begin{pmatrix} \rho h_{\text{tot}} u \\ \rho u^2 + P \\ \rho v u \\ \rho w u \\ \rho u \end{pmatrix} \quad f_y = \begin{pmatrix} \rho h_{\text{tot}} v \\ \rho u v \\ \rho v^2 + P \\ \rho w v \\ \rho v \end{pmatrix} \quad f_z = \begin{pmatrix} \rho h_{\text{tot}} w \\ \rho u w \\ \rho v w \\ \rho w^2 + P \\ \rho w \end{pmatrix} \quad (7.4)$$

#### 7.1.2 Linearized Riemann solvers

We have seen that for linear problems the Riemann solver reduces to a characteristic solver. For the full non-linear set of equations of hydrodynamics this is no longer the case. A Riemann solver, such as Godunov's method, is then a rather complex solver because it involves complex and non-linear solutions to the Riemann problem at each cell interface. It is also rather costly to solve numerically. Cheaper and elegant simplifications are *linearized Riemann solvers*. The way this is done is by expressing our interface fluxes as much as possible only using the differences in the state variables:

$$\Delta q_{k,i-1/2} \equiv q_{k,i-1/2,R} - q_{k,i-1/2,L} \quad (7.5)$$

If these differences are small, then much of the algebra can be linearized to first order in  $\Delta q_{k,i-1/2}$ .

If we now set the state at the beginning of each time step constant within each cell, then the cell interfaces have jumps of the state, i.e. they define a Riemann problem. The way to linearize this is to define an average state at the interface  $\hat{q}_{k,i-1/2}$  (note: here we retain the index  $k$  of the index notation) in some way:

$$\hat{q}_{k,i-1/2} = \text{Average}[q_{k,i}, q_{k,i-1}] \quad (7.6)$$

where the precise definition of the average will be defined later. For now we can simply set  $\hat{q}_{k,i-1/2} = (q_{k,i} + q_{k,i-1})/2$  for instance. Now we can express the Riemann problem in the deviation from this average:

$$\delta q_{k,i-1/2,L} \equiv q_{k,i-1} - \hat{q}_{k,i-1/2} \quad (7.7)$$

$$\delta q_{k,i-1/2,R} \equiv q_{k,i} - \hat{q}_{k,i-1/2} \quad (7.8)$$

If  $|\delta q_{k,i-1/2,L/R}| \ll |\hat{q}_{k,i-1/2}|$  then, locally, the Riemann problem can be regarded as a linear Riemann problem, which we have extensively discussed in Section 6.5. The advection matrix in  $x$  direction is now simply the Jacobian  $\partial f_x(q)/\partial q$ , so the equation, locally between  $x_{i-1} < x < x_i$  becomes:

$$\partial_t \delta q + \left( \frac{\partial f_x}{\partial q} \right) \partial_x \delta q = 0 \quad (7.9)$$

The eigenvalues of the Jacobian  $\partial f_x/\partial q$  at the interface  $i - 1/2$  are (for convenience we leave out the  $i - 1/2$  index):

$$\lambda_1 = \hat{u} - \hat{C}_s \quad (7.10)$$

$$\lambda_2 = \hat{u} + \hat{C}_s \quad (7.11)$$

$$\lambda_3 = \hat{u} \quad (7.12)$$

$$\lambda_4 = \hat{u} \quad (7.13)$$

$$\lambda_5 = \hat{u} \quad (7.14)$$

with eigenvectors:

$$e_1 = \begin{pmatrix} \hat{h}_{\text{tot}} - \hat{C}_s \hat{u} \\ \hat{u} - \hat{C}_s \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad e_2 = \begin{pmatrix} \hat{h}_{\text{tot}} + \hat{C}_s \hat{u} \\ \hat{u} + \hat{C}_s \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad (7.15)$$

$$e_3 = \begin{pmatrix} \frac{1}{2} \hat{u}^2 \\ \hat{u} \\ \hat{v} \\ \hat{w} \\ 1 \end{pmatrix} \quad e_4 = \begin{pmatrix} \hat{v}^2 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad e_5 = \begin{pmatrix} \hat{w}^2 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad (7.16)$$

where  $\hat{h}_{\text{tot}} = \hat{e}_{\text{tot}} + \hat{P}/\hat{\rho}$  is the total specific enthalpy and  $\hat{C}_s = \sqrt{\gamma \hat{P}/\hat{\rho}}$  is the adiabatic sound speed. In all symbols the caret  $\hat{\phantom{x}}$  indicates that these are the primitive variables as derived from the “average state”  $\hat{q}_{i-1/2}$  at the location of interface  $i - 1/2$ . Note that we can derive similar expressions for the advection in  $y$  and  $z$  direction.

We can now directly insert those formulae to Eq. (6.56) and apply this to the values of  $\delta q_{k,i-1/2,L/R}$ . Now the special form of Eq.(6.56) comes to our advantage, because since this expression (and the expressions for the flux limiters) only depend on the difference  $\delta q_{k,i-1/2,R} - \delta q_{k,i-1/2,L} \equiv q_{k,i-1/2,R} - q_{k,i-1/2,L} \equiv \Delta q_{k,i-1/2}$ , we can now forget about  $\delta q_{k,i-1/2,L/R}$  and focus entirely on  $\Delta q_{k,i-1/2}$ , which is the jump of the state over the interface. We can now decompose  $\Delta q_{k,i-1/2}$  into the eigenvectors Eq. (7.15...7.16):

$$\Delta q_{k,i-1/2} = \sum_{m=1 \dots 5} \tilde{\Delta} q_{m,i-1/2} e_{m,k,i-1/2} \quad (7.17)$$

where (again for clarity we omit the index  $i - 1/2$ ):

$$\tilde{\Delta} q_1 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ \hat{e}_{\text{kin}} \Delta q_4 - \xi \} - \frac{\Delta q_1 - \hat{u} \Delta q_4}{2\hat{C}_s} \quad (7.18)$$

$$\tilde{\Delta} q_2 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ \hat{e}_{\text{kin}} \Delta q_4 - \xi \} + \frac{\Delta q_1 - \hat{u} \Delta q_4}{2\hat{C}_s} \quad (7.19)$$

$$\tilde{\Delta} q_3 = \frac{\gamma - 1}{2\hat{C}_s^2} \{ (\hat{h}_{\text{tot}} - 2\hat{e}_{\text{kin}}) \Delta q_4 + \xi \} \quad (7.20)$$

$$\tilde{\Delta} q_4 = \Delta q_2 - v \Delta q_4 \quad (7.21)$$

$$\tilde{\Delta} q_5 = \Delta q_3 - w \Delta q_4 \quad (7.22)$$

where  $\hat{e}_{\text{kin}} = (\hat{u}^2 + \hat{v}^2 + \hat{w}^2)/2$  and  $\xi \equiv u \Delta q_1 + v \Delta q_2 + w \Delta q_3 - \Delta q_0$ . With these expressions for  $\tilde{\Delta} q_{i-1/2}$  the flux at the interface becomes (cf. Eq. 6.56) becomes

$$f_{k,i-1/2}^{n+1/2} = \frac{1}{2} (f_{k,i-1/2,R}^n + f_{k,i-1/2,L}^n) - \frac{1}{2} \sum_{m=1 \dots 5} \lambda_{m,i-1/2} \tilde{\Delta} q_{m,i-1/2} [\theta_{m,i-1/2} + \tilde{\phi}_{m,i-1/2} (\epsilon_{m,i-1/2} - \theta_{m,i-1/2})] \quad (7.23)$$

where we retained the index  $k$  in the expression for the flux:  $f_{k,i-1/2}^{n+1/2}$  according to index notation. We now see that the interface flux for this linearized Riemann solver consists of the average of the non-linear fluxes plus a correction term in which the difference of the flux over the interface is decomposed into eigenvectors and each component advected in its own upwind fashion.

A linearized Riemann solver is evidently not an exact Riemann solver, since the Riemann problem is solved in an approximate way only. This is why linearized Riemann solvers are part of the (larger) family of *approximate Riemann solvers*.

### 7.1.3 Roe's average interface state

The final missing piece of the algorithm is a suitable expression for the “average interface state”, or better, the “average interface primitive variables”  $\hat{u}_{i-1/2}$ ,  $\hat{v}_{i-1/2}$ ,  $\hat{w}_{i-1/2}$ ,  $\hat{\rho}_{i-1/2}$ ,  $\hat{h}_{\text{tot},i-1/2}$ . As long as the numerical solution is very smooth, i.e. that  $|\Delta q_{k,i-1/2}| \ll |\hat{q}_{k,i-1/2}|$ , then any reasonable average would do and would probably give the right results. However, when contact discontinuities and/or shock waves are present in the solution, then it becomes extremely important to define the proper average such that the “linearization” (which is then strictly speaking no longer valid) still produces the right propagation of these discontinuities.

A *Roe solver* is a linearized Riemann solver with a special kind of averaged state at the

interface. These state variables are defined as:

$$\hat{u} = \frac{\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.24)$$

$$\hat{v} = \frac{\sqrt{\rho_L}v_L + \sqrt{\rho_R}v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.25)$$

$$\hat{w} = \frac{\sqrt{\rho_L}w_L + \sqrt{\rho_R}w_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.26)$$

$$\hat{h}_{\text{tot}} = \frac{\sqrt{\rho_L}h_{\text{tot},L} + \sqrt{\rho_R}h_{\text{tot},R}}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (7.27)$$

With these expressions for the average interface state and the above defined eigenvector decomposition and interface flux expressions, as well as the flux limiter, the Roe solver is complete.

#### 7.1.4 Background of Roe's average interface state

So where do these expressions for the interface state come from? Here we follow the book by LeVeque. The basic idea behind the Roe solver is that it should recognize when a jump of the state  $q_R - q_L$  is a pure jump in one characteristic family only, and in that case produce the exact propagation velocity. So for a shock or for a contact discontinuity belonging to just one characteristic family the interface-average propagation matrix  $\hat{A}$  (which is the interface average of the Jacobian  $\partial f_x / \partial q$ ) should have the property:

$$\hat{A}(q_R - q_L) = f_{x,R} - f_{x,L} = s(q_R - q_L) \quad (7.28)$$

where  $s$  is the propagation speed of this wave. Note that this should not only hold for small  $q_R - q_L$ , but also when the jump is macroscopic. In case of a contact discontinuity it should get  $s = u$  and for a shock wave it should obtain  $s = v_s$  where  $v_s$  is the shock velocity.

There is an elegant mathematical derivation of how we obtain Roe's parameterization from the above condition, which is based on the definition of a state vector  $W$  defined as:

$$W = \sqrt{\rho} \begin{pmatrix} h_{\text{tot}} \\ u \\ v \\ w \\ 1 \end{pmatrix} \quad (7.29)$$

also counted from 0 to 4. The nice property of this parameter vector is that the state vector is perfectly quadratic in the components of  $W$ :

$$q = \begin{pmatrix} \frac{1}{\gamma}W_0W_4 + \frac{\gamma-1}{2\gamma}(W_1^2 + W_2^2 + W_3^2) \\ W_1W_4 \\ W_2W_4 \\ W_3W_4 \\ W_4^2 \end{pmatrix} \quad (7.30)$$

and so is the flux  $f_x$ :

$$f_x = \begin{pmatrix} W_0W_1 \\ \frac{\gamma-1}{\gamma}W_0W_4 + \frac{\gamma+1}{2\gamma}W_1^2 - \frac{\gamma-1}{2\gamma}(W_2^2 + W_3^2) \\ W_1W_2 \\ W_1W_3 \\ W_1W_4 \end{pmatrix} \quad (7.31)$$

and similar for  $f_y$  and  $f_z$ . This purely quadratic relation is very useful for the following argumentation.

If two states  $q_R$  and  $q_L$  are connected by a single wave (shock or contact discontinuity), then they obey

$$f_x(q_R) - f_x(q_L) = s(q_R - q_L) \quad (7.32)$$

where  $s$  is the propagation speed of this wave. We wish that our matrix  $\hat{A}$  (our “Jacobian”) recognizes this. So it should have the property that

$$\hat{A}(q_R - q_L) = s(q_R - q_L) \quad (7.33)$$

or in other words:

$$\hat{A}(q_R - q_L) = f_x(q_R) - f_x(q_L) \quad (7.34)$$

One way to obtain such a matrix is to integrate the Jacobian matrix  $A(q)$  over a suitable path in parameter space from  $q_L$  to  $q_R$ . We can do this for instance by defining a straight path:

$$q(\xi) = q_L + (q_R - q_L)\xi \quad (7.35)$$

so that we get

$$\begin{aligned} f_x(q_R) - f_x(q_L) &= \int_0^1 \frac{\partial f_x(q(\xi))}{\partial \xi} d\xi \\ &= \int_0^1 \frac{\partial f_x(q(\xi))}{\partial q} \frac{\partial q(\xi)}{\partial \xi} d\xi \\ &= \left[ \int_0^1 \frac{\partial f_x(q)}{\partial q} d\xi \right] (q_R - q_L) \end{aligned} \quad (7.36)$$

where we used  $\partial q(\xi)/\partial \xi = (q_R - q_L)$ . This then gives us the matrix  $\hat{A}$ :

$$\hat{A} = \int_0^1 \frac{\partial f_x(q)}{\partial q} d\xi \quad (7.37)$$

The problem with this integral is that it is not guaranteed that this produces a matrix which has real eigenvalues. Moreover, it is hard to evaluate, and therefore computationally costly.

Now here the parameter vector  $W$  comes in. Let us do the same trick of integration, but this time not in  $q$  space but in  $W$  space:

$$W(\xi) = W_L + (W_R - W_L)\xi \quad (7.38)$$

so that we get

$$\begin{aligned} f_x(q_R) - f_x(q_L) &= \left[ \int_0^1 \frac{\partial f_x(W)}{\partial W} d\xi \right] (W_R - W_L) \\ &\equiv \hat{C}(W_R - W_L) \end{aligned} \quad (7.39)$$

We can do the same for  $q_R - q_L$ :

$$\begin{aligned} q_R - q_L &= \left[ \int_0^1 \frac{\partial q(W)}{\partial W} d\xi \right] (W_R - W_L) \\ &\equiv \hat{B}(W_R - W_L) \end{aligned} \quad (7.40)$$



So we have now two matrices  $B$  and  $C$ . We can now construct the matrix  $\hat{A}$ :

$$\hat{A} = \hat{C}\hat{B}^{-1} \quad (7.41)$$

Now the nice thing of the parameter vector  $W$  is that both  $q$  and  $f_x$  are purely quadratic in  $W$ , and therefore the  $\partial q/\partial W$  and  $\partial f_x/\partial W$  are *linear* in  $W$ . This makes it much easier to evaluate the integral through  $W$  space! We shall not derive the final expressions. It suffices to say that if we would derive  $\hat{A}$  in the above way we obtain a matrix with eigenvalues and eigenvectors as we have given above. Roe's choice of average interface state variables can now be understood as originating from the fact that this particular choice of parameter vector  $W$  makes  $q$  and  $f_x$  quadratic in  $W$ .

### 7.1.5 The complete recipe of a Roe solver

Although we have discussed the complete algorithm already, let us finish this section with a point-by-point recipe for a Roe solver:

1. Determine first the  $\Delta t$  using a CFL number of 0.5
2. Construct the state vector  $q = (\rho e_{\text{tot}}, \rho u, \rho v, \rho w, \rho)$  at each cell center. This  $q_i$  also automatically defines the states at each side of the interfaces:  $q_{i-1/2,R} = q_i$ ,  $q_{i-1/2,L} = q_{i-1}$ .
3. Construct Roe's averages at the cell interfaces.
4. Using Roe's averages, create the eigenvectors and eigenvalues
5. Compute the flux jump over the interface and decompose this jump into the eigen-components to obtain the values of  $\tilde{\Delta}q_{k,i-1/2}$ .
6. Compute the flux limiter for each of the eigen-components.
7. Compute the symmetric flux average  $(f_{i-1/2,L} + f_{i-1/2,R})/2$ , and add the diffusive correction term using the flux limiter and the  $\tilde{\Delta}q$ . This creates the interface flux.
8. Now update the state vector using the interface fluxes and the  $\Delta t$  computed using the CFL condition.
9. End of time step; off to the next time step

## 7.2 Properties of the Roe solver

### 7.2.1 Strengths of the Roe solver

The Roe solver is a very powerful solver:

- It resolves shocks and contact discontinuities very tightly (in roughly 3 grid points). It is therefore significantly higher resolution than classical hydrodynamics solvers.
- It does not require any artificial viscosity for shocks because it treats shocks directly. It is, so to speak, a *shock-capturing scheme*.
- It has a very low numerical viscosity/diffusivity.
- It is strictly conserving in mass, momentum and energy.

- Because it treats pressure gradients as characteristics, sound waves are propagated with the same precision as moving matter.

### 7.2.2 Problems of the Roe solver

The Roe solver has excellent performance for many problems, but sometimes it can produce problems. Here is a list of known problems of the solver:

- Under some (rare) conditions it can try to create *expansion shocks* where it should create smooth expansion waves. This is because the Roe solver is built in such a way as to recognize Rankine-Hugoniot jump conditions even if they are the reverse. A Roe solver decomposes any smooth wave into a series of small contact discontinuities or shocks. Even for expansion waves it can happen that one of these shocks becomes strong and produces an expansion shock. This is clearly unphysical and violates the entropy condition. If this happens an *entropy fix* is necessary: a trick that disallows an expansion shock to form. We refer to LeVeque's book for details.
- Just behind strong shocks in 2-D or 3-D flows, when the shock is parallel to the grid, sometimes waves can appear. This is an odd-even decoupling problem, and fixes have been proposed which involve a minute amount of diffusion parallel to the shock, but only in the neighborhood of the shock.
- Also for grid-parallel shocks one can sometimes observe severe protrusions appearing which have a pyramid shape and a checkerboard pattern. This is known as the *carbuncle phenomenon*. This problem is rare, but if it happens it is hard to solve.
- Due to the fact that the total energy equations is used in Roe solvers, in case of extremely supersonic flows it can happen that small errors in the momentum equation can yield negative thermal energies (because  $e_{\text{th}} = e_{\text{tot}} - u^2/2$ ). This can be a potentially serious problem of all schemes that use the total energy equation. Various fixes can be considered, which can be useful in various regimes, but a generally valid solution is difficult.
- A rare, but possible problem of the Roe solver is that it does not guarantee that the interface flux that it produces is physical. For instance in the Riemann problem with  $\gamma = 1.5$ ,  $\rho_L = 1$ ,  $u_L = -2$ ,  $P_L = 4/3$  and  $\rho_R = 4$ ,  $u_R = 1$  and  $P_R = 13/3$  the first order upwind flux has an energy flux but no mass flux (Eulerink & Mellema 1995). This is clearly unphysical, and can lead to numerical problems.

In general, though, the Roe solver is quite stable and very non-viscous. It is a truly high-resolution method.

## 7.3 The HLL family of solvers

We have seen that the Roe solver splits the  $\Delta q$  (where  $q$  is the state vector) into their projections onto the basis of eigenvectors of the Jacobian. Let us write each of these components of  $\Delta q$  as  $\Delta_k q$  where  $k$  is the index of which eigenvector is meant. Each of the  $\Delta_k q$  jumps is advected with its own speed: the characteristic velocity  $\lambda_k$ , which is the eigenvalue of the Jacobian matrix. This is precisely the scenario depicted in Fig. 6.1. Each of these moving jumps is in fact a *wave*, and  $\lambda_k$  is the wave speed. We know from the true solutions of the Riemann problem for the Euler equations (Fig. 6.3) that in reality not every wave is a jump: in particular the expansion wave

is not a jump. In the Roe solver we just approximate each of them to be a jump, based on the semi-linearized set of equations.

Another Riemann solver, which also approximates all waves as jumps, is the HLL Riemann solver and its various derivatives. HLL stands for Harten, Lax and van Leer, who first proposed a method of this kind. The difference between the HLL family of solvers and the Roe solver is the wave propagation speeds  $\lambda_k$  and the wave decompositions of  $\Delta q$  into  $\Delta_k q$  are not rigorously derived from the eigenvectors and eigenvalues of some approximation of the Jacobian matrix. Instead some simpler physical arguments are used to “guess” these values. Strictly speaking the HLL family of solvers allow an infinite number of variants because it is partly left to the developer of the HLL method which recipe to take to construct the wave propagation speeds  $\lambda_k$  and the decomposition of  $\Delta q$  into waves  $\Delta_k q$ .

Let us first look at HLL type solvers in a general sense, and later worry about how exactly to compute the  $\lambda_k$  and the waves  $\Delta_k q$ .

Suppose we have  $K$  waves (i.e.  $k = 1 \dots K$ ) ordered such that  $\lambda_k \leq \lambda_{k+1}$ . Once we have decomposed  $\Delta q$  into  $\Delta_k q$  with wave speeds  $\lambda_k$  we can write the state  $q(x, t)$  near the interface  $i + 1/2$  as:

$$q(x, t + \Delta t) = q_i + \sum_{k: x > x_{i+1/2} + \lambda_k \Delta t} \Delta_k q_{i+1/2} \equiv q_{i+1} - \sum_{k: x \leq x_{i+1/2} + \lambda_k \Delta t} \Delta_k q_{i+1/2} \quad (7.42)$$

That is: the state  $q(x, t)$  near the cell interface is split into  $K + 1$  regions. The first and the last have  $q = q_i$  and  $q = q_{i+1}$  respectively. And in the wave fan region we have  $K - 1$  sub-regions of constant state  $q$  given by the above equation. For the flux that use for the cell updates we are only interested in  $\bar{q}_{i+1/2} \equiv q(x = x_{i+1/2}, t + \Delta t)$ , meaning we get:

$$\bar{q}_{i+1/2} \equiv q_i + \sum_{k: \lambda_k \Delta t < 0} \Delta_k q_{i+1/2} \equiv q_{i+1} - \sum_{k: \lambda_k \Delta t \geq 0} \Delta_k q_{i+1/2} \quad (7.43)$$

Strictly speaking, we can now compute a flux  $\bar{f}_{i+1/2} = f(\bar{q}_{i+1/2})$  and we are done: we now have a way to express the interface flux of this approximation of the Riemann solution. However, it turns out that a better (more stable and reliable) way to compute the interface flux is to note that one can construct also wave jumps in the flux  $\Delta_k f$  and use the property that:

$$\Delta_k f = \lambda_k \Delta_k q \quad (7.44)$$

This property is the equivalent of the Rankine-Hugoniot condition for shocks, but now applied to any wave. It basically guarantees that wave  $k$  indeed propagates with speed  $\lambda_k$ . In this way we can now construct the flux at the interface by adding up the jumps, like in the case of the construction of  $\bar{q}_{i+1/2}$ :

$$\bar{f}_{i+1/2} \equiv f_i + \sum_{k: \lambda_k \Delta t < 0} \lambda_k \Delta_k q_{i+1/2} \equiv f_{i+1} - \sum_{k: \lambda_k \Delta t \geq 0} \lambda_k \Delta_k q_{i+1/2} \quad (7.45)$$

Constructing the interface flux in this way is the basis of the HLL family of approximate Riemann solvers.

The above expression is still first order. To make the scheme second order one can use Eq. (6.56), which we repeat here using the  $\Delta_k$  notation:

$$\begin{aligned} f_{i+1/2}^{n+1/2} = & \frac{1}{2} (f_{i+1/2,R}^n + f_{i+1/2,L}^n) \\ & - \frac{1}{2} \sum_{k=1 \dots K} [\theta_{k,i+1/2} + \tilde{\phi}_{k,i+1/2} (\epsilon_{k,i+1/2} - \theta_{k,i+1/2})] \Delta_k f_{i+1/2}^n \end{aligned} \quad (7.46)$$

where  $\phi_{k,i+1/2}$  is the flux limiter of wave  $k$ . This is still precisely the same method of making the scheme higher order as we used for the Roe solver.

To complete our scheme we must now choose  $\lambda_k$  and  $\Delta q_k$  in a clever way. This is the topic of the rest of this section.

### 7.3.1 The HLL solver

The simplest solver of this kind is the original one. It ignores the middle wave (the mass-advection wave) and decomposes  $\Delta q$  into *two* waves  $\Delta_{(-)}q$  and  $\Delta_{(+)}q$  which are the forward and backward moving sound waves. This gives us a left state  $q_{L,i+1/2} = q_i$ , a right state  $q_{R,i+1/2} = q_{i+1}$  and a middle state  $q_{M,i+1/2}$  which is assumed to be:

$$q_M = \frac{\lambda_{(+)}q_R - \lambda_{(-)}q_L + f_L - f_R}{\lambda_{(+)} - \lambda_{(-)}} \quad (7.47)$$

(where we dropped the  $i + 1/2$  for notation convenience). Using the above expressions one can derive that the flux at the interface is then:

$$\bar{f}_{i+1/2} = \begin{cases} f_i & \text{if } \lambda_{(-)} \geq 0 \\ \frac{\lambda_{(+)}f_i - \lambda_{(-)}f_{i+1} + \lambda_{(+)}\lambda_{(-)}(q_{i+1} - q_i)}{\lambda_{(+)} - \lambda_{(-)}} & \text{if } \lambda_{(-)} < 0 < \lambda_{(+)} \\ f_{i+1} & \text{if } \lambda_{(+)} \leq 0 \end{cases} \quad (7.48)$$

So what about the expressions for  $\lambda_{(-)}$  and  $\lambda_{(+)}$ ? There is a whole variety of proposed expressions for these values. The simplest is due to Davis (1988):

$$\lambda_{(-),i+1/2} = u_i - a_i \quad \lambda_{(+),i+1/2} = u_{i+1} + a_{i+1} \quad (7.49)$$

where  $a$  is the sound speed. But there are many other versions too. See the book by Toro for an in-depth discussion.

One of the main disadvantages of this HLL solver is that it cannot keep contact discontinuities sharp. This is not surprising since we have no middle wave in this scheme.

### 7.3.2 The HLLC solver

A newer version of the HLL scheme is the HLLC scheme, where the C stands for central wave: this is a method which *does* include the middle wave that is missing from the standard HLL scheme. The general way to construct this scheme is the same as shown above. Instead of 2 waves and 3 regions of constant  $q$  we now have 3 waves and 4 regions of constant  $q$ . While the general method is the same, the details of the construction of the HLLC method (and its various estimates of the wave speeds) requires some more in-depth discussion. We refer to the book of Toro for these details.

## 7.4 Source extrapolation methods

One major disadvantage of Riemann solvers in general is that they are, by their structure, less capable of “recognizing” (semi-)static solutions in which pressure gradients are compensated by an external force (typically gravity). To demonstrate what is meant let us take the example of a hydrostatic atmosphere (e.g. Earth’s atmosphere) with small perturbations on it (e.g. the formation of clouds). We want that the unperturbed atmosphere is recognized by the method

in the sense that the method perfectly keeps the static atmosphere intact and does not produce wiggles, or worse: entropy. A hydrostatic atmosphere obeys:

$$\frac{dP}{dz} = -\rho g \quad (7.50)$$

where  $z$  is the vertical coordinate and  $g$  is the gravitational constant of the atmosphere ( $g \simeq 1000\text{cm/s}^2$ ). Let us discretize this as:

$$\frac{P_{i+1} - P_i}{z_{i+1} - z_i} = -\frac{1}{2}(\rho_{i+1} + \rho_i)g \quad (7.51)$$

and let us construct the solution by choosing  $P = K\rho^\gamma$  with  $K$  constant (an adiabatic atmosphere) and choosing  $\rho(z=0)$  as the density at the base. We can then integrate (for a choice of  $K$ ) from bottom to some height above the surface using Eq.(7.51). Since this equation is implicit in  $\rho_{i+1}$  one must solve for each new  $\rho_{i+1}$  using for instance an iteration at each new grid point until Eq.(7.51) is satisfied. We then get a hydrostatic atmosphere that is consistent.

If we insert this into a time-dependent hydro code we want that this hydrocode leaves this solution exactly intact (i.e. that it does not introduce perturbations). For classical numerical hydro schemes of Chapter 5 this is in fact not difficult, especially not if a staggered grid is used. This is because the  $\partial P/\partial z$  term is treated as a source term in such methods. We then have two source terms: the  $\partial P/\partial z$  and the  $-\rho g$  term. If we discretize these two terms in exactly the same way as in Eq.7.51, then both terms cancel out exactly (for this hydrostatic solution) and the hydrostatic solution is kept exactly intact (to machine precision). We can then safely study tiny perturbations of this atmosphere without the worry that we may in fact be involuntarily studying the intrinsic noise of the method instead.

For Riemann solvers, however, this is not so easy. It is a fundamental property of these methods to include the  $\partial P/\partial z$  term in the advection part, while the gravity force remains a source term (and can not be treated in the advection part). The pressure gradient force and the gravity force are therefore treated in fundamentally different ways and one cannot guarantee that they will *exactly* cancel for hydrostatic solutions.

To solve this problem Eulerink & Mellema (1995, A&ASup 110, 587) introduced the concept of *spatial flux extrapolation*. A very similar, but easier method was proposed by LeVeque (1998, J.Comp.Phys. 146, 346). We will use a combination of both formalisms here.

Consider the generalized hyperbolic equation with a source term:

$$\partial_t q(x, t) + \partial_x f(q(x, t)) = s(x) \quad (7.52)$$

The traditional way to do operator splitting is to first solve  $\partial_t q(x, t) + \partial_x f(q(x, t)) = 0$  for one time step using for instance a Riemann solver and then solve  $\partial_t q(x, t) = s(x)$  for the same time step (by simply adding the source). This gives us the problem of not recognizing steady states. A steady state is a solution of the equation

$$\partial_x f(q(x, t)) = s(x) \quad (7.53)$$

The *source extrapolation* method for time-dependent hydrodynamics is inspired on this stationary equation. At the start of each time step we construct a *subgrid model*  $q(x, t = t_n)$  (for  $x_{i-1/2} < x < x_{i+1/2}$ ) where  $q(x = x_i, t = t_n) = q_i$  such that within the cell  $q(x, t = t_n)$  is a solution of 7.53. To make things simpler we assume that  $s(x)$  is constant within a cell, so we get:

$$\partial_x f(q(x, t)) = s_i \quad \text{for } x_{i-1/2} < x < x_{i+1/2} \quad (7.54)$$

From this equation we could in principle solve for  $q(x, t)$  within cell  $i$  and thereby obtain values of  $q$  at the left and right sides of each interface:  $q_{L,i+1/2}$  and  $q_{R,i+1/2}$  (we come back in an minute to this). These are then the values we put into a Riemann solver to produce our interface flux  $f_{i+1/2}$  which we use for the state update. It should be noted, however, that since now the states on both sides of the interface are no longer spatially constant (they follow the subgrid model). So in principle we have a *generalized Riemann problem* on the cell interface, in which the states are not constant. Such generalized Riemann problems are very difficult to solve. So instead, we simply ignore the fact that the states are strictly speaking not constant on both sides, and simply use  $q_{L,i+1/2}$  and  $q_{R,i+1/2}$  as the two states of a classical Riemann problem which we then solve with our favorite method.

Now let's come back to the solution of Eq. 7.54. If  $f(q)$  is a linear function of  $q$ , say  $f(q) = uq$  where  $u$  is a velocity, then the solution of Eq. 7.54 is simple:

$$q_{R,i-1/2} = q_i - \frac{1}{2}\Delta x_i s_i / u_i \quad (7.55)$$

$$q_{L,i+1/2} = q_i + \frac{1}{2}\Delta x_i s_i / u_i \quad (7.56)$$

This works fine as long as  $u \neq 0$ . Once  $u$  approaches zero we are in trouble. Moreover, if  $f(q)$  is a *non-linear* function of  $q$  then we can write:

$$f_{R,i-1/2} = f(q_i) - \frac{1}{2}\Delta x_i s_i \quad (7.57)$$

$$f_{L,i+1/2} = f(q_i) + \frac{1}{2}\Delta x_i s_i \quad (7.58)$$

which is fine, but solving  $q_{R,i+1/2}$  from a given flux  $f_{R,i+1/2}$  is non-trivial, and in the case of the Euler equations typically has two different solutions, or one single solution or no solutions. It is therefore often unpractical to try to determine the  $q_{L/R,i+1/2}$  from  $f_{L/R,i+1/2}$ . It turns out, however, that for Riemann solvers that are based entirely on propagating waves  $\Delta_k f = \lambda_k \Delta_k q$  where  $\Delta_k$  is the  $k$ -th wave and  $\lambda_k$  is its propagation speed, one can use  $\Delta f_{i+1/2} = f_{R,i+1/2} - f_{L,i+1/2}$  as input to the wave-decomposition routine while keeping  $q_{L,i+1/2} = q_i$  and  $q_{R,i+1/2} = q_{i+1}$  as the primitive variables used to compute the eigenvectors and eigenvalues. In particular for the Roe solver this hybrid approach works well.

Now how does this solve the problem of recognizing a steady-state solution? The trick is that if the linear subgrid model described in Eq. (??) is sufficiently accurate, then one will obtain for the steady state solution:

$$f_{L,i+1/2} = f_{R,i+1/2} \quad (7.59)$$

meaning that

$$\Delta f_{i+1/2} = 0 \quad (7.60)$$

If we now use Eq.(7.46) then we get:

$$f_{i+1/2}^{n+1/2} = \frac{1}{2}(f_{i+1/2,R}^n + f_{i+1/2,L}^n) \quad (7.61)$$

because by virtue of  $\Delta f_{i+1/2} = 0$  the entire Riemann solver part drops out of the equation now. What is left is:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x} [f(q_{i+1}^n) - f(q_{i-1}^n)] + \Delta t s_i^n \quad (7.62)$$

So if we construct our steady state initial condition such that

$$\frac{f(q_{i+1}) - f(q_{i-1})}{2\Delta x} + s_i = 0 \quad (7.63)$$

then the method finally recognizes the static solution down to machine precision.



## 7.5 Employing slope limiters *before* the Riemann solver step

So far we have studied solvers in which we could clearly distinguish two or three wave families. We were then able to apply flux limiters (or equivalently, slope limiters) to each wave (cf. Eq.7.46). This method ensures that the slope in each wave mode is taken into account and therefore gives second order accuracy in space.

Another method of making Riemann solvers higher-order is to apply slope limiters (not flux limiters) *before* the Riemann problem is solved. The trick is to extrapolate the cell-centered primitive variables  $\rho$ ,  $P$ ,  $\vec{u}$  to the cell-walls using a linear subgrid model. We can also extrapolate the conserved quantities  $\rho$ ,  $\rho\vec{u}$ ,  $\rho e_{\text{tot}}$ , whichever produces the best results. This then gives us the left and right states at the cell walls ( $q_{L,i+1/2}$  and  $q_{R,i+1/2}$ ), which defines a Riemann problem which we can then solve using any solver we want. Contrary to the flux limiter recipe of Eq.7.46 this slope limiter method can also be applied to Riemann solvers that do not approximate the problem into jump-like waves. This is therefore a way to make the original Godunov solver higher order.

This method is in a way similar to Section 7.4, but there is an essential difference: here we use neighboring points to compute the slope of the linear subgrid model (instead of the source function). We therefore get non-zero slopes also for cases without source. We can employ all the machinery of Chapter 4 to produce the best possible linear subgrid model using e.g. MINMOD or SUPERBEE slope limiters.

One major problem that we encounter if we simply apply this method as-is, is that it quickly becomes unstable. It turns out (see Toro's book) that this instability problem can be elegantly solved by producing adapted left- and right interface states  $\tilde{q}_{L,i+1/2}$  and  $\tilde{q}_{R,i+1/2}$  from the interface states  $q_{L,i+1/2}$  and  $q_{R,i+1/2}$  that come out of the slope limiter method by advancing these half a time step in time in the following way:

$$\tilde{q}_{L,i+1/2} = q_{L,i+1/2} + \frac{1}{2} \frac{\Delta t}{\Delta x} [f(q_{R,i-1/2}) - f(q_{L,i+1/2})] \quad (7.64)$$

$$\tilde{q}_{R,i+1/2} = q_{R,i+1/2} + \frac{1}{2} \frac{\Delta t}{\Delta x} [f(q_{R,i+1/2}) - f(q_{L,i+3/2})] \quad (7.65)$$

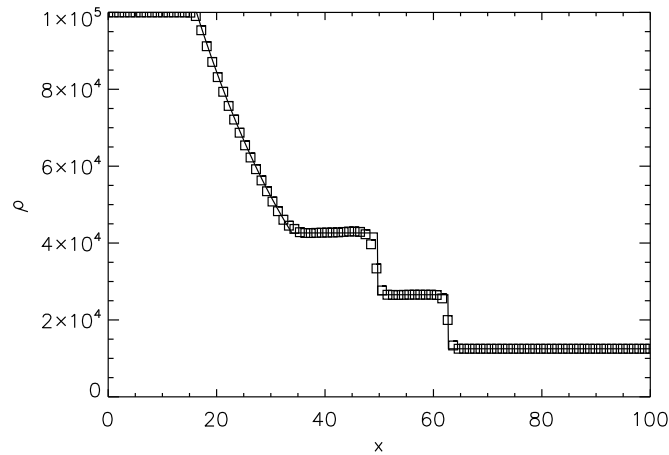
(see book by Toro, but beware of different notation). Note that the left one is updated using  $q_{R,i-1/2}$  and  $q_{L,i+1/2}$ , while the right one is updated using  $q_{R,i+1/2}$  and  $q_{L,i+3/2}$ . These equations are saying that the subgrid model *within each cell individually* is advanced half a time step. Using this method we obtain a stable higher-order scheme.

The method is now completed by inserting  $\tilde{q}_{L,i+1/2}$  and  $\tilde{q}_{R,i+1/2}$  into our favorite Riemann solver and obtaining the interface flux with which we update the cell centered state variables.

**NOTE:** The use of the slope limiters before the Riemann step excludes the use of the source extrapolation method described in Section 7.4, and vice versa. If one wishes to use the source extrapolation method then only the flux limiter method after the Riemann step (Eq. 7.46) can be used.

## 7.6 The PPM Method

A very well-known Riemann solver method is the “Piecewise Parabolic Method” (Colella & Woodward 1984, J.Comp.Phys. 54, 174). It was designed before many of the above described methods were developed, and therefore it lies a bit off from the main track as described above, but in large part it follows (and indeed pioneered) the same ideas. It has proved to be a pretty



**Figure 7.1.** The result of the Roe solver with superbee flux limiter on a Sod shocktube test with  $\rho_L = 10^5$ ,  $P_L = 1$ ,  $u_L = 0$ ,  $\rho_R = 1.25 \times 10^4$ ,  $P_R = 0.1$ ,  $u_R = 0$  and  $\gamma = 7/5$ . Solid line: analytic solution; symbols: Roe solver.

powerful method and is still very often used. We will describe it very briefly here, but we will not go into details because the method is rather complex and has many fine-tuning aspects.

The main idea of this Riemann solver is to use a reconstruction step *before* the Riemann step (see Section 7.5). But while we used linear reconstruction in Section 7.5, the PPM method uses quadratic reconstruction. In other words: starting from the cell-center values of the primitive variables  $\rho$ ,  $p$  and  $\vec{u}$  each cell subgrid model is a parabola. In the PPM method care is taken that no overshoots happen (TVD scheme) and that in general the reconstruction is well-behaved. Using these parabolic subgrid models, the primitive variables on both sides of each of the interfaces can be calculated. We know from Section 7.5 that if we directly insert these values into our Riemann problem solver, we get an unstable scheme. The PPM method solves this using an approximation. It finds the fastest moving left- and right- characteristics and makes an average of the primitive variables over these domains. This is the PPM version of the half-step-interface-update described in Section 7.5. It is less mathematically rigorous, but in practice it works.

Now these left- and right- interface values can be inserted in a Riemann solver code. Again, as in Section 7.4, the solution to the true (generalized) Riemann problem in this case is very complex and no analytic solutions exist of generalized Riemann problems with parabolic spatial dependency of the variables on both sides. Therefore, as before, the approximation is made to solve the classical Riemann problem, with constant states on both sides, even though we know that the states on both sides are not constant.

We refer to the original paper by Colella & Woodward 1984 for details of the method and how the method is fine-tuned.

## 7.7 Code testing: the Sod shock tube tests

The Sod shock tube test of Section 6.3.2 can be used to test the performance of our computer program. We leave it to the reader to test the algorithms of the previous chapter on this kind of test. Here we merely shock the performance of the Roe solver on a test with  $\rho_L = 10^5$ ,  $P_L = 1$ ,  $u_L = 0$ ,  $\rho_R = 1.25 \times 10^4$ ,  $P_R = 0.1$ ,  $u_R = 0$  and  $\gamma = 7/5$ . The result is shown in Fig. 7.1.



# Chapter 8

## Coordinate systems and gridding techniques

So far we have assumed that we are doing modeling in a cartesian coordinate system with a rectangular mesh. For many applications this is, however, too restrictive. For instance, in astrophysics, if we are interested in modeling the hydrodynamics of a rotating star or a rotating collapsing molecular cloud core, then we know (or assume) beforehand that our solution will be nearly spherical, and that we are modeling deviations from such a spherical distribution. It is then clear that a polar coordinate system  $(r, \theta, \phi)$  (often also called a “spherical coordinate system”) is better and easier to use. Another example is if we wish to study axisymmetric systems. In this case we can restrict ourselves to two dimensions:  $(r, \theta)$  in polar coordinates or  $(r, z)$  in cylindrical coordinates. Since 2-D calculations are very much faster than 3-D ones, this choice of coordinates saves a lot of computing time for such problems. Therefore non-cartesian coordinates are very often used, in particular in astrophysics.

But sometimes the usage of a different coordinate system does not help. For instance, in astrophysics, if we want to model the collapse of a complex and clumpy molecular cloud, in which a number of clumps contract enormously and eventually become stars. No choice of coordinate system helps here. One must refine the grid around each clump in an adaptive (time-dependent) way. For this type of gridding problem astrophysicists typically use *Adaptive Mesh Refinement (AMR)*.

A more down-to-earth example is the flow of gas or fluid around obstacles, such as the flow of air around a wing or around a racing car. The shape of the obstacle is typically too complex to warp a particular coordinate system around it. On the other hand, if we try to solve this problem on a cartesian grid, then the obstacle surfaces are jagged surfaces (like a slope of a hill created with LEGO blocks). These jagged surfaces tend to produce disturbances that, in high Reynolds number flows, can really cause havoc in the solution. In these cases special purpose *unstructured grids* are used.

In this chapter we will touch upon all three issues (the polar coordinate system, AMR techniques and unstructured grids), but we will be rather brief. We will concern ourselves only with Riemann solvers, but much of the philosophy can be directly transferred to classical hydrodynamics schemes as well.

**NOTE:** This chapter is written very recently, so there may still be errors in the equations. Beware.

## 8.1 Hydrodynamics in the polar coordinate system

### 8.1.1 A scalar conservation law in polar coordinates

There are two popular non-cartesian coordinate systems that are often used: cylindrical coordinates  $(r, \phi, z)$  and polar/spherical coordinates  $(r, \theta, \phi)$ . In this section we will focus on the polar coordinate system. The equations for cylindrical coordinates can be derived from these by taking  $z \simeq (\pi/2 - \theta)r$  and taking the limit  $z \ll r$  to remove all sine and cosine factors (and once the equations are derived,  $z$  can again be large).

The coordinates are:  $r > 0$ , the radial coordinate,  $0 \leq \theta < \pi$  the polar coordinate ( $\theta = 0$  is the north pole,  $\theta = \pi$  is the south pole and  $\theta = \pi/2$  is the equatorial plane), and  $0 \leq \phi < 2\pi$  the azimuthal coordinate. Let us define  $\vec{u}$  the velocity vector and  $(u, v, w)$  its components in  $r, \theta$  and  $\phi$  directions.

Each grid point  $(r_i, \theta_j, \phi_k)$  is at the center of a cell. A cell is given by:  $r_{i-1/2} \leq r < r_{i+1/2}$ ,  $\theta_{j-1/2} \leq \theta < \theta_{j+1/2}$ ,  $\phi_{k-1/2} \leq \phi < \phi_{k+1/2}$ . The cell walls in  $r$  and  $\theta$  direction are curved while the cell walls in  $\phi$  direction are straight surfaces. This curvature will turn out to produce new terms in the momentum equations, but let us first look at how we can formulate a scalar conservation problem in polar coordinates. The partial differential equation for a scalar conservation law in polar coordinates is:

$$\frac{\partial q}{\partial t} + \nabla \cdot (q\vec{u}) = 0 = \frac{\partial q}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 qu)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta qv)}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(qw)}{\partial \phi} = 0 \quad (8.1)$$

As one can see, there are various geometric factors involved.

One can easily see how these factors arise if one derives a discrete version of this conservation equation. Consider a cell  $(i, j, k)$  given by  $r_{i-1/2} \leq r < r_{i+1/2}$ ,  $\theta_{j-1/2} \leq \theta < \theta_{j+1/2}$  and  $\phi_{k-1/2} \leq \phi < \phi_{k+1/2}$ . The volume of this cell is:

$$V_{i,j,k} = \frac{1}{3}(r_{i+1/2}^3 - r_{i-1/2}^3)(\cos \theta_{j-1/2} - \cos \theta_{j+1/2})(\phi_{k+1/2} - \phi_{k-1/2}) \quad (8.2)$$

The  $i + 1/2$  surface of the cell in  $r$ -direction (i.e. an  $r = \text{constant}$  surface) is:

$$S_{(r),i+1/2,j,k} = r_{i+1/2}^2(\cos \theta_{j-1/2} - \cos \theta_{j+1/2})(\phi_{k+1/2} - \phi_{k-1/2}) \quad (8.3)$$

and similar for  $i - 1/2$ . The  $j + 1/2$  surface of the cell in  $\theta$ -direction (i.e. a  $\theta = \text{constant}$  surface) is:

$$S_{(\theta),i,j+1/2,k} = \frac{1}{2}(r_{i+1/2}^2 - r_{i-1/2}^2) \sin \theta_{j+1/2}(\phi_{k+1/2} - \phi_{k-1/2}) \quad (8.4)$$

and similar for  $j - 1/2$ . The  $k + 1/2$  surface of the cell in  $\phi$ -direction (i.e. a  $\phi = \text{constant}$  surface) is:

$$S_{(\phi),i,j,k+1/2} = \frac{1}{2}(r_{i+1/2}^2 - r_{i-1/2}^2)(\theta_{j+1/2} - \theta_{j-1/2}) \quad (8.5)$$

and identical for  $k - 1/2$ .

If we now difference the equations in a conservative form:

$$\begin{aligned} \frac{\partial(q_{i,j,k} V_{i,j,k})}{\partial t} = & F_{i-1/2,j,k} S_{(r),i-1/2,j,k} - F_{i+1/2,j,k} S_{(r),i+1/2,j,k} \\ & + F_{i,j-1/2,k} S_{(\theta),i,j-1/2,k} - F_{i,j+1/2,k} S_{(\theta),i,j+1/2,k} \\ & + F_{i,j,k-1/2} S_{(\phi),i,j,k-1/2} - F_{i,j,k+1/2} S_{(\phi),i,j,k+1/2} \end{aligned} \quad (8.6)$$

where  $F$  is the flux for  $q$ . Eq.(8.6) is the conservatively discretized form of the scalar conservation equation in a general coordinate system. With Eqs.(??) inserted into Eq.(8.6), we obtain the conservatively discretized form of the scalar conservation equation in polar coordinates.

For use later in this chapter, let us introduce  $\Delta_r$ ,  $\Delta_\theta$ , and  $\Delta_\phi$ , as well as  $S_r$ ,  $S_\theta$  and  $S_\phi$ , and finally  $F_r$ ,  $F_\theta$  and  $F_\phi$  such that Eq. (8.6) for polar coordinates can be written as:

$$\frac{\partial(q_{i,j,k}V_{i,j,k})}{\partial t} + (\Delta_r F_r S_r)_{i,j,k} + (\Delta_\theta F_\theta S_\theta)_{i,j,k} + (\Delta_\phi F_\phi S_\phi)_{i,j,k} = 0 \quad (8.7)$$

Here the  $\Delta_r$  symbol denotes that a difference is computed of the argument (here:  $F_r S_r$ ) between its value at the  $i + 1/2$  interface and the  $i - 1/2$  interface. Same for  $\theta$  and  $\phi$ .

→ **Exercise:** Derive, from this discrete form of the conservation equation, the differential form of the equation, Eq. (8.1).

### 8.1.2 The Euler equations in polar coordinates

The discretized conservation law in the form Eq.(8.6) (or equivalently Eq.(8.7)) is the form of the equation which we use for our Riemann solvers for hydrodynamics. Because Riemann solvers by definition produce an interface flux ( $F_{i+1/2,j,k}$ ,  $F_{i,j+1/2,k}$  or  $F_{i,j,k+1/2}$ ), this form of the conservation equation lends itself perfectly for Riemann solvers. One simply substitutes the scalar  $q_{i,j,k}$  with  $(\rho e_{\text{tot}}, \rho u, \rho v, \rho w, \rho)$  and the scalar fluxes  $F_{i+1/2,j,k}$ ,  $F_{i,j+1/2,k}$  and  $F_{i,j,k+1/2}$  with the fluxes from the Riemann problems in each of these directions. This is indeed the way we can implement Riemann solvers in non-cartesian coordinates. However, this is unfortunately not yet the full story.

The above derivation only holds for scalar quantities. Indeed,  $\rho$  and  $\rho e_{\text{tot}}$  are scalars, and both obey Eq. (8.1):

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) &= 0 \\ &= \frac{\partial \rho}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho u)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho v)}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(\rho w)}{\partial \phi} = 0 \end{aligned} \quad (8.8)$$

and

$$\begin{aligned} \frac{\partial \rho e_{\text{tot}}}{\partial t} + \nabla \cdot (\rho h_{\text{tot}} \vec{u}) &= 0 \\ &= \frac{\partial \rho e_{\text{tot}}}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho h_{\text{tot}} u)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho h_{\text{tot}} v)}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(\rho h_{\text{tot}} w)}{\partial \phi} = 0 \end{aligned} \quad (8.9)$$

Both these equations can indeed be solved using the discrete form we derived above (see Eq. 8.6). These give for the density equation:

$$\frac{\partial(\rho V)_{i,j,k}}{\partial t} + (\Delta_r F_{\rho r} S_r)_{i,j,k} + (\Delta_\theta F_{\rho \theta} S_\theta)_{i,j,k} + (\Delta_\phi F_{\rho \phi} S_\phi)_{i,j,k} = 0 \quad (8.10)$$

where  $F_{\rho r}$ ,  $F_{\rho \theta}$  and  $F_{\rho \phi}$  are the Riemann fluxes for the density. For the energy equation:

$$\frac{\partial(\rho e_{\text{tot}} V)_{i,j,k}}{\partial t} + (\Delta_r F_{\rho e_{\text{tot}} r} S_r)_{i,j,k} + (\Delta_\theta F_{\rho e_{\text{tot}} \theta} S_\theta)_{i,j,k} + (\Delta_\phi F_{\rho e_{\text{tot}} \phi} S_\phi)_{i,j,k} = 0 \quad (8.11)$$

where  $F_{\rho e_{\text{tot}} r}$ ,  $F_{\rho e_{\text{tot}} \theta}$  and  $F_{\rho e_{\text{tot}} \phi}$  are the Riemann fluxes for the energy.

However, for the momentum components  $\rho u$ ,  $\rho v$  and  $\rho w$  this is not the case. These three form the  $r$ ,  $\theta$  and  $\phi$  components of the momentum *vector*  $\vec{p}$ , which is sensitive to spatial direction. Now here comes the difficulty: Suppose one has a momentum vector  $\vec{p}_1 = (1, 0, 0)$  at location  $(r = 1, \theta = \pi/2, \phi = 0)$ . If we compare this to the momentum vector  $\vec{p}_2 = (1, 0, 0)$  at location  $(r = 1, \theta = \pi/2, \phi = \pi/2)$ , then we see that these two vectors, although they have precisely the same components in polar coordinates, they point in different directions. This is because the vector basis in which the vector components are formulated in the polar coordinates is a *local* vector basis: the first component always points away from the center, the second away from the pole and the third in the azimuthal direction. These directions are different at different positions. Since physical objects, like fluid/gas packages, do not have knowledge of the polar coordinate system and its peculiarities, their momentum vectors behave “naturally” when they are expressed in a *global* vector basis. If they are cast into a local basis that has a different orientation at different positions, then they behave strangely in such a basis. Therefore, in polar coordinates the conservation laws of vector quantities such as the momentum of a fluid parcel yields extra terms in the equations. They look as if an additional force is present, and therefore these terms are also called “pseudo forces”. They have the property that they *only* change the direction of the momentum vector as expressed in the local basis of the non-cartesian coordinate system. The absolute value of the momentum vector (and thereby of the velocity vector) is not changed. This is logical, because pseudo forces are not real forces and can therefore not truly accelerate or decelerate a fluid parcel. However, in Section 8.1.5 we will see that if one uses a *rotating* polar coordinate system, then it may seem as if a true acceleration/deceleration is taking place, but again, this turn out to be not a real effect.

It is not trivial to derive the extra terms that arise for the momentum equations. The most robust and reliable way of doing this is to use Riemannian Geometry, and this method also allows one to derive the equations of viscous hydrodynamics (the Navier-Stokes equation), which spawns even more complex additional terms when cast in polar coordinates. We will not go into this mess here, and instead we will simply quote the resulting Euler equations in polar coordinates.

The radial momentum equation becomes:

$$\begin{aligned} \frac{\partial \rho u}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho u^2)}{\partial r} + \frac{\partial P}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho u v)}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(\rho u w)}{\partial \phi} \\ - \frac{\rho(v^2 + w^2)}{r} = 0 \end{aligned} \quad (8.12)$$

In discrete form for the Riemann solver:

$$\begin{aligned} \frac{\partial(\rho u V)}{\partial t} + \Delta_r(F_{rr} S_r) + \Delta_\theta(F_{r\theta} S_\theta) + \Delta_\phi(F_{r\phi} S_\phi) \\ = \left[ \frac{2P}{r} + \frac{\rho(v^2 + w^2)}{r} \right] V \end{aligned} \quad (8.13)$$

where  $V$  is the volume of the cell,  $S_r$ ,  $S_\theta$  and  $S_\phi$  the surfaces of the cell, and  $F_{rr} = \rho u^2 + P$ ,  $F_{r\theta} = \rho u v$  and  $F_{r\phi} = \rho u w$  the momentum fluxes for this equation. These momentum fluxes (together with the density and energy fluxes) are the ones that we obtain from the Riemann solver of our choice at the interfaces.

The  $\theta$  momentum equation becomes:

$$\begin{aligned} \frac{\partial \rho v}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho u v)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho v^2)}{\partial \theta} + \frac{1}{r} \frac{\partial P}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(\rho w v)}{\partial \phi} \\ + \rho \frac{v u}{r} - \frac{\cos \theta}{\sin \theta} \rho \frac{w^2}{r} = 0 \end{aligned} \quad (8.14)$$

In discrete form for the Riemann solver:

$$\begin{aligned} \frac{\partial(\rho v V)}{\partial t} + \Delta_r(F_{\theta r} S_r) + \Delta_\theta(F_{\theta \theta} S_\theta) + \Delta_\phi(F_{\theta \phi} S_\phi) \\ = - \left[ \rho \frac{v u}{r} - \frac{\cos \theta}{\sin \theta} \frac{\rho w^2 + P}{r} \right] V \end{aligned} \quad (8.15)$$

where  $F_{\theta r} = \rho v u$ ,  $F_{\theta \theta} = \rho v^2 + P$  and  $F_{\theta \phi} = \rho v w$  are the momentum fluxes for this equation.

The  $\phi$  momentum equation becomes:

$$\begin{aligned} \frac{\partial \rho w}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho u w)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho v w)}{\partial \theta} + \frac{1}{r \sin \theta} \frac{\partial(\rho w^2 + P)}{\partial \phi} \\ + \frac{\rho u w}{r} + \frac{\cos \theta}{\sin \theta} \frac{\rho v w}{r} = 0 \end{aligned} \quad (8.16)$$

In discrete form for the Riemann solver:

$$\begin{aligned} \frac{\partial(\rho w V)}{\partial t} + \Delta_r(F_{\phi r} S_r) + \Delta_\theta(F_{\phi \theta} S_\theta) + \Delta_\phi(F_{\phi \phi} S_\phi) \\ = - \left[ \rho \frac{u w}{r} + \frac{\cos \theta}{\sin \theta} \frac{\rho v w}{r} \right] V \end{aligned} \quad (8.17)$$

where  $F_{\phi r} = \rho w u$ ,  $F_{\phi \theta} = \rho w v$  and  $F_{\phi \phi} = \rho w^2 + P$  are the momentum fluxes for this equation.

The recipe for applying a Riemann solver to polar coordinates is then:

- At the start of the run we pre-compute the cell volumes and cell surfaces, as well as the necessary cosine and sine values. This saves much CPU time, in particular since cosine and sine functions are heavy in terms of CPU time required.
- We perform directional operator splitting (Strang splitting). For each direction we do 1-D independent problems and copy their results back into the 3-D grid.
- For each 1-D problem we compute the left- and right- interface states in precisely the same way as the chapter on Riemann solvers. These left and right states are now used to compute the Riemann interface fluxes. Note that the Roe or HLLC subroutine used here does not know about any geometric issues. It simply produces a flux as if it were a Cartesian Riemann solver.
- These Riemann fluxes are now inserted in the  $r$ ,  $\theta$  or  $\phi$  splitted part of Eqs.(??). By virtue of operator splitting the other terms in these equations are put to zero.
- The source terms in these equations can be added at the very end of the time step, as another operator splitting step. But sometimes it can be useful instead to add one of them at the end of one directional splitting step, while the other at the end of another directional splitting step. This is something the designer of the algorithm can choose freely. Various choices have various special advantages and disadvantages. It depends on the problem at hand which strategy is the best.

### 8.1.3 A note of caution for computing the geometric source terms

Let us briefly return to the discrete radial momentum equation Eq.(8.13). The term  $2P/r$  appeared because we had to convert the  $\partial P/\partial r$  into a  $(1/r^2)\partial(P r^2)/\partial r$  in order to incorporate it into the momentum flux difference. It matters, however, how one computes this term. If done without care, one could easily get a situation in which the case  $P(r, \theta, \phi) = \text{constant}$  and  $u, v, w = 0$  is not recognized by the solver as a static solution. One can, however, replace:

$$\frac{2P}{r}V \rightarrow P\Delta_r S_r \equiv (S_{(r),i+1/2,j,k} - S_{(r),i-1/2,j,k})P_{i,j,k} \quad (8.18)$$

If one now computes this geometric pressure source term in this way, then for the case of  $P(r, \theta, \phi) = \text{constant}$  and  $u, v, w = 0$  the solution is kept unaltered, as it should be, down to machine precision. In other words, the code now “recognizes” the  $P(r, \theta, \phi) = \text{constant}$  solution. The reason why this is the case is because the term  $2P/r$  arises because of the differences in the surface areas of the cell interfaces at  $i - 1/2$  and  $i + 1/2$  which are used in the  $\Delta_r(F_{rr}S_r)$  term. So by compensating this with exactly the same but opposite we get an exact cancellation, and thereby a constant pressure will be recognized as such.

The same trick can be applied to the geometric pressure source term of the  $\theta$ -momentum equation:

$$\frac{\cos \theta}{\sin \theta} \frac{P}{r}V \rightarrow P\Delta_\theta S_\theta \equiv (S_{(\theta),i,j+1/2,k} - S_{(\theta),i,j-1/2,k})P_{i,j,k} \quad (8.19)$$

For the  $\phi$ -momentum equation no such geometric pressure term arises.

### 8.1.4 Replacing $\phi$ -momentum with angular momentum

If one models rotating systems such as accretion disks or rotating stars it can prove to be advantageous in terms of accuracy and reliability to not advect the  $\phi$ -momentum but the  $\phi$ -angular momentum instead (see e.g. Kley 1998, A&A 338, L37).

Let us define  $l = w \sin \theta r$ . We only replace  $w$  by  $l$  in the advection in  $r$  and in  $\theta$  direction.

The physical form of the  $\phi$ -momentum equation is then:

$$\begin{aligned} \frac{\partial \rho l}{\partial t} + \frac{\partial(\rho u l)}{\partial r} + \frac{1}{r} \frac{\partial(\rho v l)}{\partial \theta} + \frac{\partial(\rho w^2 + P)}{\partial \phi} \\ + 2 \sin \theta \rho u w + \cos \theta \rho v w = 0 \end{aligned} \quad (8.20)$$

which can be written as:

$$\frac{\partial \rho l}{\partial t} + \frac{1}{r^2} \frac{\partial(r^2 \rho u l)}{\partial r} + \frac{1}{r \sin \theta} \frac{\partial(\sin \theta \rho v l)}{\partial \theta} + \frac{\partial(\rho w^2 + P)}{\partial \phi} = 0 \quad (8.21)$$

The discrete form for the Riemann solver is:

$$\frac{\partial(\rho l V)}{\partial t} + \Delta_r(r \sin \theta F_{\phi r} S_r) + \Delta_\theta(r \sin \theta F_{\phi \theta} S_\theta) + \Delta_\phi(r \sin \theta F_{\phi \phi} S_\phi) = 0 \quad (8.22)$$

One sees that the source terms on the right-hand side are completely gone. This is good, because source terms are always a menace and if they can be avoided, then that is a positive thing.

For convenience we can now define the fluxes for the  $\phi$  angular momentum:

$$\tilde{F}_{\phi r} = r \sin \theta F_{\phi r} = \rho u l \quad (8.23)$$

$$\tilde{F}_{\phi \theta} = r \sin \theta F_{\phi \theta} = \rho v l \quad (8.24)$$

$$\tilde{F}_{\phi \phi} = r \sin \theta F_{\phi \phi} = \rho w l + P r \sin \theta \quad (8.25)$$

so then we obtain:

$$\frac{\partial(\rho l V)}{\partial t} + \Delta_r(\tilde{F}_{\phi r} S_r) + \Delta_\theta(\tilde{F}_{\phi \theta} S_\theta) + \Delta_\phi(\tilde{F}_{\phi \phi} S_\phi) = 0 \quad (8.26)$$

This means that all we have to do is to construct the normal fluxes  $F$  from the Riemann solver, and convert them into  $\tilde{F}$  for the angular momentum equation (the other equations remain untouched), and difference them.

### 8.1.5 Rotating polar coordinate system

For many purposes it may be desirable to let the coordinate system rotate with a certain rate. For instance, if we model the flow of air in the Earth's atmosphere, we want to use a polar coordinate system that rotates with the Earth's rotation rate, so that we can anchor the coordinate system to the surface of the Earth. Another example is the study of accretion disks around young stars and black holes. This will, however, lead to yet again more "pseudo forces": the Coriolis force and the centrifugal force. In fact, they are precisely the same as the pseudo forces we have found in the static spherical coordinates for moving gas. This time, however, they also appear if the gas is not moving with respect to the rotating coordinate system, because having zero velocity in the rotating frame means having non-zero velocity in the lab frame.

We can derive the equations from the above equations. We replace:

$$\partial_t \rightarrow \bar{\partial}_t - \Omega_0 \partial_\phi \quad (8.27)$$

$$w \rightarrow \bar{w} + \Omega_0 r \sin \theta \quad (8.28)$$

where  $\Omega_0$  is the rotation rate of the polar coordinate system around the polar axis and  $\bar{w}$  is the  $\phi$ -velocity in the rotating frame. We keep, however, the angular momentum variable  $l$  still in the lab frame, because only in the lab frame this quantity makes physical sense.

One can show that this coordinate transformation does not affect the continuity equation. The energy conservation equation is also not affected, as long as one still expresses the kinetic energy in the non-rotating lab frame:  $(u^2 + v^2 + w^2)/2$  and not  $(u^2 + v^2 + \bar{w}^2)/2$ . The reason is that energy only has physical meaning in a non-rotating frame. Only the momentum equations are affected.

So let us look at the  $r$ -momentum equation. Because one replaces the time derivative with  $\bar{\partial}_t - \Omega_0 \partial_\phi$ , one gains an extra term  $-\Omega_0 \partial_\phi(\rho u V)$ . But one also replaces  $w$  with  $\bar{w} + \Omega_0 r \sin \theta$  in the term with the  $\Delta_\phi$ , which gives another extra term  $\Omega_0 r \sin \theta \Delta_\phi(\rho u S_\phi)$ . By writing the first extra term into a discrete form, one sees that the two extra terms cancel. One can also replace the  $V w^2/r$  geometric source term with  $V(\bar{w} + \Omega_0 r \sin \theta)^2/r$ . This gives two extra terms:

$$V w^2/r = V(\bar{w} + \Omega_0 r \sin \theta)^2/r = V(\bar{w}^2 + 2\bar{w}\Omega_0 r \sin \theta + (\Omega_0 r \sin \theta)^2) \quad (8.29)$$

These are the centrifugal force terms. However, there is no reason why one could not keep the equation hybrid in  $w$  and  $\bar{w}$ , as long as the original  $w$  only appears outside of the derivatives or differences. If we use that strategy, then our radial momentum equation in the rotating frame becomes:

$$\bar{\partial}_t(\rho u V) + \Delta_r(F_{rr} S_r) + \Delta_\theta(F_{r\theta} S_\theta) + \Delta_\phi(\bar{F}_{r\phi} S_\phi) = \left[ \frac{2P}{r} + \frac{\rho(v^2 + w^2)}{r} \right] V \quad (8.30)$$

where we have left the original  $w$  in the source term (right hand side) and replaced  $F_{r\phi}$  with  $\bar{F}_{r\phi} = F_{r\phi} - \Omega_0 r \sin \theta \rho u = \rho u \bar{w} - \Omega_0 r \sin \theta \rho u = \rho u \bar{w}$ , or in other words,  $\bar{F}_{r\phi}$  is the  $F_{r\phi}$



Riemann flux as measured in the local moving reference frame (i.e. as seen by an observer moving with velocity  $\Omega_0 r \sin \theta$ ). So it follows that the structure of the equation does not change. Just the left-hand-side now refers to fluxes in the moving frame, while the source terms on the right-hand-side still refer to the lab-frame  $w$ . One could also write out  $w^2$  in terms of  $\bar{w}$  as in Eq. (8.29).

Using the same philosophy we can derive the  $\theta$ -momentum equation for the case of a rotating coordinate system:

$$\bar{\partial}_t(\rho v V) + \Delta_r(F_{\theta r} S_r) + \Delta_\theta(F_{\theta \theta} S_\theta) + \Delta_\phi(\bar{F}_{\theta \phi} S_\phi) = - \left[ \rho \frac{vu}{r} - \frac{\cos \theta}{\sin \theta} \frac{\rho w^2 + P}{r} \right] V \quad (8.31)$$

where  $\bar{F}_{\theta \phi} = F_{\theta \phi} - \Omega_0 r \sin \theta \rho v = \rho v w - \Omega_0 r \sin \theta \rho v = \rho v \bar{w}$  is the Riemann flux in the comoving frame.

Finally, for the angular momentum equation we get:

$$\bar{\partial}_t(\rho l V) + \Delta_r(\tilde{F}_{\phi r} S_r) + \Delta_\theta(\tilde{F}_{\phi \theta} S_\theta) + \Delta_\phi(\tilde{\tilde{F}}_{\phi \phi} S_\phi) = 0 \quad (8.32)$$

where  $\tilde{\tilde{F}}_{\phi \phi} = \tilde{F}_{\phi \phi} - \Omega_0 r \sin \theta \rho l = \rho l w + P r \sin \theta - \Omega_0 r \sin \theta \rho l = \rho l \bar{w} + P r \sin \theta$ . In contrast to  $\bar{F}_{r \phi}$  and  $\bar{F}_{\theta \phi}$ , the  $\tilde{\tilde{F}}_{\phi \phi}$  is not just a multiplication of the comoving flux. Once we obtain the comoving Riemann flux  $\bar{F}_{\phi \phi} = \rho \bar{w}^2 + P$  from the Riemann solver, here is how to obtain  $\tilde{\tilde{F}}_{\phi \phi}$ :

$$\tilde{\tilde{F}}_{\phi \phi} = r \sin \theta (\tilde{F}_{\phi \phi} + \rho \bar{w} r \sin \theta) \quad (8.33)$$

So we obtain the comoving Riemann flux  $\tilde{F}_{\phi \phi}$ , modify it with the above equation to obtain  $\tilde{\tilde{F}}_{\phi \phi}$  and insert this into the difference equation.

### 8.1.6 The FARGO-trick for modeling nearly Keplerian disks

Using a rotating polar coordinate system is very useful for instance for the study of an accretion disk around a star. There is, however, a slight problem. An accretion disk has a rotation velocity according to Kepler's law  $\Omega_K(r) = \sqrt{GM/r^3}$ , meaning it is *differentially rotating*. There is no single global  $\Omega_0$  that describes the main rotation of the fluid so that one can concentrate on perturbations on this disk. Instead, if one fixes the rotation rate of the grid  $\Omega_0$  to the Kepler frequency  $\Omega_K(r_0)$  at radius  $r_0$ , then the accretion disk will have a comoving frame  $\phi$ -velocity  $\bar{w}$  that is about zero at  $r_0$ , but it will be strongly negative for  $r \gg r_0$  and strongly positive for  $r \ll r_0$ . Since the highest characteristic velocities are the ones that limit the global time step, one may be forced to take very small time steps if the outer radius of the grid is much larger than  $r_0$  or the inner radius much smaller than  $r_0$ . This may put a severe limitation on the spatial dynamic range.

The FARGO code of Frederic Masset solves this problem in an elegant way. The trick is to make a scheme that, for this special case, can advect over more than 1 cell per time step. This may sound peculiar, because if it were so easy, why did we do all the effort to design the advection algorithms in the previous chapters? The reason we had to do it there was that for normal hydrodynamics various characteristics move with different speeds and the same characteristics move with different speeds at different locations. Therefore it was not possible to find a single time step by which one could simply remap the quantity  $q_i$  to  $q_{i+3}$  for example.

Here we can partly use this remapping technique, while doing the rest using the normal advection schemes. Suppose that within a single time step the gas should move over a distance



of  $3.423 \phi$ -grid cells in  $\phi$ -direction. We can now perform the remapping trick: we first remap all quantities over precisely 3 grid points. What is left is an advection over 0.423 grid points, which is then done using our favorite advection scheme or a Riemann solver. The distance in number of grid cells to move in the remapping step will depend on radius  $r$ . The larger  $\text{abs}(r - r_0)$ , the larger the remapping step. The number of steps to take equals the distance of advection of the gas in the present time step, rounded to the nearest integer value. The CFL time step is now determined by requiring that, after the remapping step, none of the characteristics moves farther than 1 cell size in either direction.

## 8.2 Adaptive Mesh Refinement

Using different coordinate systems to fit best to the problem at hand is a useful technique for problems with a reasonably simple geometry. However, once the geometry becomes more complex, such as the geometry of a Giant Molecular Cloud with multiple dense cores, then polar coordinates do not help. In fact, they only make it more complex. One is then best advised to use 3-D Cartesian coordinates again. The problem is that these coordinates are not adapted to the problem at hand. The dense clumps may require a much finer mesh than the rest of the problem. Taking a fine mesh throughout the volume may be too computationally demanding.

One way to solve this problem is to use the technique of *mesh refinement*. This works by replacing a part of the course grid by a finer grid, possibly even recursively. One thus gets a patch-work hierarchical grid: Coarser grids higher up the hierarchy and finer grids down in the hierarchy. If these patches obey certain rules, for instance that each patch of refinement must either be fully inside a patch of lower resolution or fully outside, then the data management of these patches can be kept reasonably in check. A hierarchical grid is then built up as a tree. The base grid stands at the stem of the tree, and each patch of finer resolution mesh is a branch, which can then branch off itself into even finer patches.

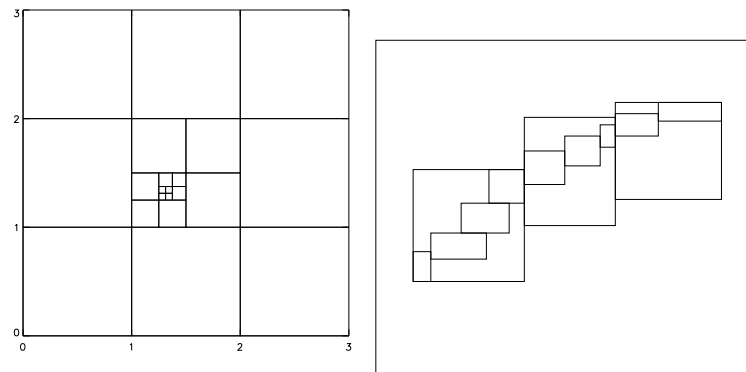
There are several libraries that take care of such hierarchical grid refinement. For instance the CHOMBO library (written mostly in C++ but with some F77 parts in it) allows a very large freedom to choose the size and position of the patches. Another library, PARAMESH, is written in F90/F95 and has a somewhat more rigid structuring of the patches, with the advantage of efficiency. Here the highest level of refinement is a patch of  $N_x \times N_y \times N_z$  grid cells. One can combine these into a larger block of  $2 \times 2 \times 2$  of these grid patches, and this continues all the way up to the main (lowest resolution) grid.

We will not go into detail on the precise structure of such AMR libraries in this text. We will just briefly discuss how these things work.

In Fig.8.1 one can see how the two above described AMR techniques work geometrically.

To construct such grids requires rather complex data management. In a hierarchical structure, where levels of gridding are strictly based on sub-division of higher-level grids the data structure is typically a tree-like structure. And typically each level of refinement is an integer multiple of lower resolution grids, such that a cell at higher resolution will at most have one neighbor of equal or lower refinement level at each of its cell walls.

Communication between cells in an AMR tree is complex. The easiest way to do this is to pad each grid block with ghost cells (1 row for first order schemes, 2 rows for second order schemes etc). At the beginning of each time step one can then fill the ghost cells and all the complexity of the AMR data management is done in this ghost-cell filling. Once this is done, each block of cells can then be handled independently for one time step. This works very well



**Figure 8.1.** Illustration of AMR gridding. Left: A hierarchical cell-division type of AMR (e.g. Paramesh library). Right: AMR based on patches (e.g. Chombo library).

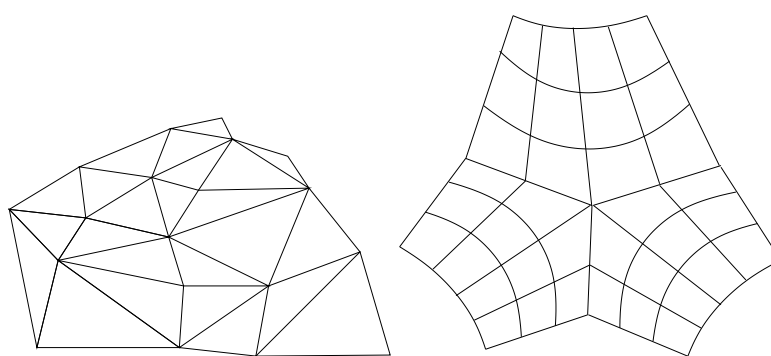
(and is implemented in PARAMESH), but if the highest-refinement level consists of blocks of  $1 \times 1 \times 1$ , then there is a huge overhead of ghost cells: for a second order scheme in 2-D each cell is then padded with 24 ghost cells (and in 3-D it's even worse: 124). The easiest way out is so make the highest-refinement levels to consist of  $N \times N$  cells, so that the ghost-cell overhead compared to the true cells is not so large anymore. If one wants to save memory and overhead, then a direct management of communication between true cells is desirable. This is, however, rather challenging from a programming perspective.

In any case, it is strongly advised to use existing libraries for the data-management of AMR, instead of writing such libraries oneself, because of the complexity of the matter.

### 8.3 Unstructured grids, or partly structured grids

For many purposes the boundary conditions of the problem at hand are very complex. For instance: flow in a complex pipe structure, or in a reactor of a factory, or the flow of gas over a car or airplane etc. For such problems it is important to allow very flexible gridding. There are many ways of doing this. The most flexible is the gridding using triangles in 2-D or tetraeders in 3-D (Fig. 8.2-left). The creation of good grids using this triangulation is non-trivial, and this is an entire field of engineering by itself. One of the main disadvantages of such completely flexible gridding is that it has difficulty making use of the highly parallel computer architectures now available, in particular architectures with extremely parallel graphics card accelerators. Such accelerators need simply structured grids to acquire substantial speeds-ups. For this reason the complex geometries of the grid are sometimes generated out of patches of regular but curved) coordinate systems patched together at their interfaces (see Fig. 8.2-right).

We will not go into details here.



**Figure 8.2.** Illustration of unstructured gridding. Left: with triangles (or in 3-D: tetraeders). Right: with piecewise structured curved patches.

## Chapter 9

# Implicit integration, incompressible flows

The methods we discussed so far work well for problems of hydrodynamics in which the flow speeds of interest are not orders of magnitude smaller than the sound speed. There are, however, many situations in which we are interested in flows that are very subsonic. For instance the flow of air in the Earth's lower atmosphere, the flow of water in oceans, etc. In such cases it is reasonable to assume that there is pressure equilibrium everywhere to the extent that the gas/fluid will not be accelerated to velocities anywhere near the sound speed. In other words, the flow is always very subsonic:  $|\vec{u}| \ll C_s$ . In such cases we are not interested in the detailed propagation of sound waves, but in the much slower movement of the fluid itself. Can we use the methods of the previous chapter for such problems? Yes, we can, but it will become extremely computationally expensive to model any appreciable flow of the fluid. Here is why: if we determine the time step from the CFL condition (see Chapter 3), then the time step is limited by the fastest characteristics. In this case these are clearly the sound waves. If, for example, our typical flow moves at speeds of less than 0.01 times  $C_s$ , then the time step is 100 times smaller than required for the advection of the fluid itself. So the fast characteristics may not be interesting for us, but they do exist, and not obeying the CFL condition for them (i.e. taking the CFL condition for the fluid movement only, which gives a 100 times larger time step in our example) would simply blow up our code. The problem here is that the sound waves exist, if we like them or not. And they limit the time step. Suppose we want to model the flow of fluid from the left boundary of our domain to the right, over 100 grid points. For pure advection (disregarding the sound waves) the CFL condition with CFL number 0.5 requires of the order of 200 time steps for the fluid to move into the domain on the left and out of the domain on the right. But because the sound waves are much faster, say  $100\times$  faster, they limit the time step to a  $100\times$  smaller value, meaning that our model now requires 20000 time steps. It is clear that this is not very practical, and alternative methods must be found to allow us to keep our time step such that it obeys the CFL condition for the fluid motion only, not for the sound waves.

Problems of this kind, where the time scales of interest are much larger than the formal shortest time scale of the problem, are called “stiff”. This is a very common complication in numerical integration of ordinary and partial differential equations (PDEs). If the rapid modes of the problem quickly relax to a secular equilibrium (i.e. a quasi-equilibrium), then one may be more interested in the much longer time scale evolution of the system, without wanting to waste CPU time on the short-time-scale modes that are anyway related to their quasi equilibrium, and hence do not change on these short time scales. An example of a typical “stiff” partial differential equation is that of the diffusion equation. The shortest time scales are represented by the smearing-out of the narrowest peaks. In more mathematical terms: the highest  $\vec{k}$  modes

damp out the quickest. In typical applications these highest  $\vec{k}$  modes damp out so quickly that they have reached their quasi-equilibrium already in the first few time steps. The more interesting longer timescale evolution (“secular evolution”) concerns much smaller  $\vec{k}$  modes, i.e. much longer wavelength modes. These could in principle be followed sufficiently accurately with time steps much larger than the formal CFL condition. The CFL condition limits the time step to the damping time of the modes with a wavelength as large as two grid points. However, since these modes are expected to be always near their equilibrium, we are not interested in modeling this system at these very small time steps. Numerical stability, however, forces us to do this, thereby forcing us to waste a lot of CPU time.

To solve problems of “stiff” nature, such as the diffusion problem or the problem of very subsonic flow, a fundamentally different way of integration of the partial differential equations has to be introduced. For very subsonic flow we also make an approximation to the flow equations. All these techniques require the solution of large sets of coupled linear equations, and the solution of these equations requires special techniques, which we will briefly outline here (though this is a topic that deserves an entire one-semester lecture!).

We will start out with a simple example of an ordinary differential equation to demonstrate the principle of stiffness. Then we will concern ourselves with the diffusion equation and how to solve this efficiently using “implicit integration”. We will then generalize this to 2-D and 3-D, and turn then to general methods for the solution of multi-dimensional parabolic partial differential equations.

We will then turn to the equations very subsonic (i.e. incompressible) flows. We shall see that the solution requires similar techniques as for parabolic PDEs to solve for the pressure. Once this is done, we can use our classic explicit advection schemes for the propagation of the fluid.

## 9.1 Simple example: an ordinary differential equation

### 9.1.1 Stiffness of an ordinary differential equation

Let us start with a simple example of a stiff ordinary differential equation:

$$\frac{d}{dt} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & -100 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \quad (9.1)$$

Suppose at  $t = 0$  we start with  $q_1 = 1$  and  $q_2 = 1$ . Now let us integrate this using forward Euler integration, i.e. simple first order explicit integration:

$$\begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \end{pmatrix} = \begin{pmatrix} q_1^n \\ q_2^n \end{pmatrix} + \Delta t \begin{pmatrix} -1 & 0 \\ 0 & -100 \end{pmatrix} \begin{pmatrix} q_1^n \\ q_2^n \end{pmatrix} \quad (9.2)$$

The CLF condition for this equation, with CFL number 0.5, is  $\Delta t = 0.005$ . If we are interested mainly in the behavior of  $q_1$ , then this time step is very small. For  $q_1$  alone a time step of  $\Delta t = 0.5$  would be fine. Of course, in this simple example we could indeed integrate  $q_1$  with a different time step than  $q_2$ , but in more realistic cases one cannot always separate the variable so easily. In such cases one must integrate the entire set of equations together, and a common time step is required, which must then be the smallest of the two, i.e.  $\Delta t = 0.005$ . For following our variable of interest,  $q_1$ , we now have to perform many hundreds of time steps, much more than we would expect judging solely from the slow variation of  $q_1$ . In fact, after the initial fast decay of  $q_2$ , even  $q_2$  does not change very much anymore, as it reached its equilibrium state already long before  $q_1$  has changed appreciably. So in this example we are taking very small time steps,

much smaller than strictly required for following the variations of  $q_1$  and  $q_2$ . It is only required for numerical stability.

### 9.1.2 Implicit integration: the backward Euler method

An efficient and very stable way to solve this stiffness problem is to use *implicit integration*, which is also often called *backward Euler integration*. We have seen already an example of this for an ordinary differential equation (ODE) in chapter 3. The trick is to use not the values of  $q$  at time  $t = t_n$  but their future values at time  $t = t_{n+1}$  for the evaluation of the right-hand-side of the equation. Since these future values are not yet known at  $t = t_n$ , this poses a chicken-or-egg problem. This problem can be solved if one writes the set of equations for  $q^{n+1}$  such that all the future variables  $q^{n+1}$  that were initially on the right hand side are now on the left hand side. For linear equations this can then be written as a matrix equation which can be solved using standard matrix equation solvers.

Let us put our above example in implicit form:

$$\begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \end{pmatrix} = \begin{pmatrix} q_1^n \\ q_2^n \end{pmatrix} + \Delta t \begin{pmatrix} -1 & 0 \\ 0 & -100 \end{pmatrix} \begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \end{pmatrix} \quad (9.3)$$

where the difference with Eq.(9.2) is only in the last vector where  $n$  was replaced by  $n + 1$ . Now let us reorder the  $n + 1$  variables to the left and all  $n$  variables to the right:

$$\left[ 1 - \Delta t \begin{pmatrix} -1 & 0 \\ 0 & -100 \end{pmatrix} \right] \begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \end{pmatrix} = \begin{pmatrix} q_1^n \\ q_2^n \end{pmatrix} \quad (9.4)$$

If we define the matrix  $\mathbf{M}$  to be:

$$\mathbf{M} = 1 - \Delta t \begin{pmatrix} -1 & 0 \\ 0 & -100 \end{pmatrix} \quad (9.5)$$

and vector  $\mathbf{q}$  to be

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} \quad (9.6)$$

then the above matrix equation is:

$$\mathbf{M}\mathbf{q}^{n+1} = \mathbf{q}^n \quad (9.7)$$

This is the matrix equation we now have to solve at each time step. For our simple example this becomes:

$$\begin{pmatrix} 1 + \Delta t & 0 \\ 0 & 1 + 100\Delta t \end{pmatrix} \begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \end{pmatrix} = \begin{pmatrix} q_1^n \\ q_2^n \end{pmatrix} \quad (9.8)$$

which can be easily solved:

$$q_1^{n+1} = q_1^n / (1 + \Delta t) \quad (9.9)$$

$$q_2^{n+1} = q_2^n / (1 + 100\Delta t) \quad (9.10)$$

The nice property of this solution is that there is no positive  $\Delta t$  for which  $q_1^{n+1}$  or  $q_2^{n+1}$  becomes negative. When using the explicit Euler method taking  $\Delta t$  larger than the CFL limit will lead to negative values. Implicit integration, as demonstrated here, is therefore *unconditionally stable*, even though for large  $\Delta t$  one should not expect the solution to be accurate. However, suppose we take  $\Delta t$  to be too large for an accurate integration of  $q_2$ , this is not too problematic, because we

are anyway interested mainly in  $q_1$  in our example. Could an inaccurate integration of  $q_2$ , in more complex examples, compromise the accuracy of  $q_1$ ? Yes, it could. But typically the faster modes (in this case  $q_2$ ) quickly reach a semi-steady-state (a quasi-equilibrium), and this state is reached even when taking very large time steps for  $q_2$ . The way in which  $q_2$  reaches its quasi-steady state may be inaccurate if one takes too large time steps, but once it reaches it semi-steady state, it stays there and the implicit method gives the right answer for  $q_2$  from that point onward. So as long as we are interested solely in  $q_1$  and we take a time step that is small enough for an accurate integration of  $q_1$ , then using an implicit method guarantees stability and accuracy for  $q_1$ , even if the problem is more complex than shown above.

So does this mean that the above simple implicit integration method is a simple method that can easily be applied to any problem? The answer is, unfortunately, no. In the above example it was easy to solve the matrix equation because the matrix was diagonal. In more general cases the matrix will not be strictly diagonal, and may be pretty large. For instance, if one has a  $\mathbf{q}$  vector of 100 components, and the  $100 \times 100$  matrix  $\mathbf{M}$  has all its elements non-zero, then an analytic solution for  $\mathbf{q}^{n+1}$  from  $\mathbf{q}^n$  is not practical. One then needs to resort to standard matrix equation solvers.

### 9.1.3 A standard matrix equation solver: LU decomposition

There are many different methods for solving matrix equations. I can warmly recommend the book by Press et al. *Numerical Recipes in C* or *Numerical Recipes in Fortran*. Here we shall focus on the most popular method, the LU decomposition.

Suppose we wish to solve the following equation:

$$\begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \quad (9.11)$$

for  $(x_1, x_2, x_3, x_4)$ . The matrix here is *lower triangular*, meaning that the elements above the diagonal are all zero. For this special case the solution is simple. We start by solving the first equation:  $x_1 = y_1/\alpha_{11}$ . Once we know this, the second equation becomes  $x_2 = (y_2 - \alpha_{21}x_1)/\alpha_{22}$ . Since  $x_1$  is known,  $x_2$  is directly obtained by this expression. Next, we find  $x_3$  by  $x_3 = (y_3 - \alpha_{31}x_1 - \alpha_{32}x_2)/\alpha_{33}$ , etc until all values of  $x$  are obtained. Equations of the type 9.11 are therefore trivially solved. Also *upper-triangular* matrix equations are trivially solved in a similar manner:

$$\begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \quad (9.12)$$

Now, suppose we wish to solve the following non-trivial matrix equation:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \quad (9.13)$$

Then unfortunately we do not have such a simple method. We can try to create an upper-triangular or lower-triangular matrix equation out of this by Gaussian elimination. But a more



useful method is to try to write the matrix as a product of a lower-triangular matrix and an upper-triangular matrix:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \quad (9.14)$$

or in matrix notation:

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} \quad (9.15)$$

Finding the components of  $\mathbf{L}$  and  $\mathbf{U}$  for any given matrix  $\mathbf{A}$  is not trivial, but numerous off-the-shelf routines are available to make this LU-decomposition of the matrix  $\mathbf{A}$ , for instance, in the book of Press et al, cited above. But once they are found, the solution to the equation

$$\mathbf{L} \cdot \mathbf{U} \cdot \mathbf{x} = \mathbf{y} \quad (9.16)$$

is easily found using the above sequential procedure.

The nice property of this LU-decomposition method is that once these two triangular matrices are found, the solution  $\mathbf{x}$  can be found for a great number of vectors  $\mathbf{y}$  without the need for recomputing the matrices  $\mathbf{L}$  and  $\mathbf{U}$  all the time.

## 9.2 A 1-D diffusion equation

Suppose we wish to solve the following diffusion equation

$$\frac{\partial q}{\partial t} - \frac{\partial}{\partial x} \left( D \frac{\partial q}{\partial x} \right) = b \quad (9.17)$$

where  $D$  is a diffusion constant. Let us assume  $D = \text{constant}$  and the grid spacing  $\Delta x$  is also constant. The source term  $b$  does not necessarily have to be constant. The forward Euler (i.e. explicit) integration of the equation can be written as:

$$q_i^{n+1} = q_i^n + \frac{\Delta t D}{\Delta x^2} (q_{i+1}^n - 2q_i^n + q_{i-1}^n) + \Delta t b \quad (9.18)$$

The CFL condition for this equation is

$$\Delta t \lesssim 0.5 \frac{\Delta x^2}{D} \quad (9.19)$$

This condition is very restrictive for fine grid resolution (small  $\Delta x$ ). The short time scale comes about because the highest  $k$  modes (with a wavelength twice the grid spacing) have the shortest time scale. It turns out, however, that these high- $k$  modes also damp out quickly, so after a short time they have become irrelevant; we do not need to model them any longer. But these modes are still there and if  $\Delta t$  is larger than the CFL condition these modes explode.

So let us write the equation in implicit form:

$$q_i^{n+1} = q_i^n + \frac{\Delta t D}{\Delta x^2} (q_{i+1}^{n+1} - 2q_i^{n+1} + q_{i-1}^{n+1}) + \Delta t b \quad (9.20)$$



Now sort things:

$$q_i^{n+1} - \frac{\Delta t D}{\Delta x^2} (q_{i+1}^{n+1} - 2q_i^{n+1} + q_{i-1}^{n+1}) = q_i^n + \Delta t b \quad (9.21)$$

We can write this as a matrix equation:

$$\mathbf{A} \mathbf{q}^{n+1} = \mathbf{q}^n + \Delta t \mathbf{b} \quad (9.22)$$

The solution would then formally be:

$$\mathbf{q}^{n+1} = \mathbf{A}^{-1} (\mathbf{q}^n + \Delta t \mathbf{b}) \quad (9.23)$$

but in practice one never calculates the matrix  $\mathbf{A}^{-1}$ , nor its LU components. The reason is that the matrix  $\mathbf{A}$  is a *sparse matrix*, meaning that only a very limited subset of its elements are non-zero. It requires therefore very little storage space to store the matrix  $\mathbf{A}$  in the computer. However, the inverse matrix  $\mathbf{A}^{-1}$  is non-sparse. Typically all its elements are non-zero. It would therefore require quite a bit of storage space, and moreover it would require an enormous amount of computational effort to create the matrix  $\mathbf{A}^{-1}$ .

Instead, there are various methods for solving matrix equations where the matrix is sparse. The above matrix has a special structure: it is *tridiagonal*. A tridiagonal matrix of, say  $7 \times 7$  has the following form:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 & 0 \\ 0 & 0 & 0 & 0 & a_6 & b_6 & c_6 \\ 0 & 0 & 0 & 0 & 0 & a_7 & b_7 \end{pmatrix} \quad (9.24)$$

→ **Exercise:** Prove that, for Eq.(9.21), the matrix  $\mathbf{A}$  has a tridiagonal form. Give an expression for  $a_i$ ,  $b_i$  and  $c_i$  for  $2 \leq i \leq N - 1$  (where  $N$  is the grid size). At the left boundary require that  $q = K$  (for some arbitrary number  $K$ ) and at the right boundary require that  $\partial q / \partial x = 0$ .

Once Eq.(9.21) is written in matrix form Eq.(9.22) with  $\mathbf{A}$  a tridiagonal matrix, then the solution is relatively simple. Let's assume we have  $N = 7$ , i.e. 7 grid points, so that we can write this equation in the following way:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 & 0 \\ 0 & 0 & 0 & 0 & a_6 & b_6 & c_6 \\ 0 & 0 & 0 & 0 & 0 & a_7 & b_7 \end{pmatrix} \begin{pmatrix} q_1^{n+1} \\ q_2^{n+1} \\ q_3^{n+1} \\ q_4^{n+1} \\ q_5^{n+1} \\ q_6^{n+1} \\ q_7^{n+1} \end{pmatrix} = \begin{pmatrix} r_1^{n+1} \\ r_2^{n+1} \\ r_3^{n+1} \\ r_4^{n+1} \\ r_5^{n+1} \\ r_6^{n+1} \\ r_7^{n+1} \end{pmatrix} \quad (9.25)$$

where  $\mathbf{r}$  contains the entire right-hand-side of the equation. One can simply start an elimination procedure:

$$q_1^{n+1} = (r_1 - c_1 q_2^{n+1}) / b_1 \quad (9.26)$$

$$q_2^{n+1} = (r_2 - a_2 r_1 / b_1 - c_2 q_3^{n+1}) / (b_2 - a_2 c_1 / b_1) \quad (9.27)$$

$$q_3^{n+1} = \dots \quad (9.28)$$

One sees that every new equation solves on variable but introduces a new one. However, once one arrives at  $i = N$  then the equation does not introduce any new variables and the equation gives the first actual number. One can then back-substitute everything from  $i = N$  to  $i = 1$  and the solution is done. This procedure is called the *forward-elimination backward-substitution method*. It works very well and robustly for problems involving tridiagonal matrices.

Once this equation is solved, the  $q_i^{n+1}$  values are found and the next time step can be made.

### 9.3 Diffusion equation in 2-D and 3-D: a prelude

Now let us do the same thing for a 2-D diffusion equation:

$$\frac{\partial q}{\partial t} - \frac{\partial}{\partial x} \left( D \frac{\partial q}{\partial x} \right) - \frac{\partial}{\partial y} \left( D \frac{\partial q}{\partial y} \right) = b \quad (9.29)$$

In discrete implicit form:

$$q_{i,j}^{n+1} = q_{i,j}^n + \frac{\Delta t D}{\Delta x^2} (q_{i+1,j}^{n+1} - 2q_{i,j}^{n+1} + q_{i-1,j}^{n+1}) + \frac{\Delta t D}{\Delta y^2} (q_{i,j+1}^{n+1} - 2q_{i,j}^{n+1} + q_{i,j-1}^{n+1}) + \Delta t b \quad (9.30)$$

and sort this:

$$q_{i,j}^{n+1} - \frac{\Delta t D}{\Delta x^2} (q_{i+1,j}^{n+1} - 2q_{i,j}^{n+1} + q_{i-1,j}^{n+1}) - \frac{\Delta t D}{\Delta y^2} (q_{i,j+1}^{n+1} - 2q_{i,j}^{n+1} + q_{i,j-1}^{n+1}) = q_{i,j}^n + \Delta t b \quad (9.31)$$

How to write this in a matrix form? First we have to make a 1-D vector  $\mathbf{q}$  out of the 2-D arrangement of variables  $q_{i,j}$ . This can be done in the following way (here: a  $4 \times 4$  grid):

$$\mathbf{q} = (q_{1,1}, q_{2,1}, q_{3,1}, q_{4,1}, q_{1,2}, q_{2,2}, q_{3,2}, q_{4,2}, q_{1,3}, q_{2,3}, q_{3,3}, q_{4,3}, q_{1,4}, q_{2,4}, q_{3,4}, q_{4,4})^T \quad (9.32)$$

So for an  $N_x \times N_y$  grid we get a vector  $\mathbf{q}$  of length  $N = N_x N_y$ . The corresponding matrix  $\mathbf{A}$  will then have  $N \times N$  components, i.e.  $(N_x N_y)^2$  components. Again, most of these components will be 0, so we will want to store these matrix elements in a clever way, like the  $a, b, c$  case for the tridiagonal case. In this way we can keep the matrix storage manageable. For a 3-D grid all of this goes similarly, and for an  $N_x \times N_y \times N_z$  grid we get a vector  $\mathbf{q}$  of length  $N = N_x N_y N_z$  and a matrix of  $(N_x N_y N_z)^2$  elements, again most of which are 0. But let us stick to the 2-D case for now.

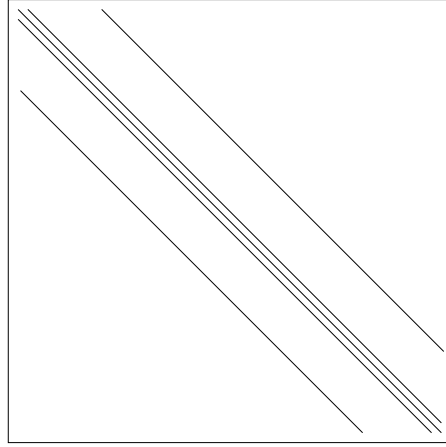
So what is the structure of this matrix? The matrix will be partly tridiagonal, but it will have side-bands too. See Fig.9.1.

→ **Exercise:** What is the location of the side bands in the matrix (i.e. how many rows or columns offset from the true diagonal)?

For this kind of matrix the forward elimination backward substitution method does not work. For this we need iterative solvers for sparse matrices.

### 9.4 Iterative solvers for sparse matrix equations

There is a large variety of iterative methods for solving sparse matrix equations. Some basic methods are *Jacobi iteration*, *Successive Overrelaxation (SOR)*, *Incomplete LU factorization*, *Gauss-Seidel iteration* etc. These methods are not very efficient, but it turns out that in combination with other methods they still have their use.



**Figure 9.1.** Graphical representation of the  $\mathbf{A}$  matrix resulting from the 2-D diffusion equation.

#### 9.4.1 Conjugate gradient methods

Here we will focus on a particular class of methods called *conjugate gradient methods* (again we refer here to the book by Press et al. “Numerical Recipes”, but also to the book by Ferziger & Peric “Computational Methods for Fluid Dynamics”). Consider the general matrix equation

$$\mathbf{A}\mathbf{q} = \mathbf{b} \quad (9.33)$$

If *and only if* the matrix  $\mathbf{A}$  is symmetric and positive definite one can devise the following method. Define a function

$$f(\mathbf{q}) = \frac{1}{2} \mathbf{q} \cdot \mathbf{A} \cdot \mathbf{q} - \mathbf{b} \cdot \mathbf{q} \quad (9.34)$$

At the point where  $f(\mathbf{q})$  has a minimum,  $\mathbf{q}$  is a solution of the matrix equation. This minimum is located where

$$\nabla f = \mathbf{A}\mathbf{q} - \mathbf{b} = 0 \quad (9.35)$$

proving that indeed this minimum corresponds to the solution of the matrix equation. The trick is now to start with an initial guess and walk toward the point where the minimum is. At each iteration  $k$  we make a search direction  $\mathbf{p}_k$  and we find the value of a quantity  $\alpha_k$  such that  $f(\mathbf{q}_{k+1})$  is minimized, where  $\mathbf{q}_{k+1}$  is given by:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha_k \mathbf{p}_k \quad (9.36)$$

So how do we determine  $\mathbf{p}_k$ ? This could be done using the “steepest descent” method. This method is guaranteed to converge, but it can converge very slowly. In particular if the minimum lies in a very long and narrow valley. The *conjugate gradient* method is a version of this method in which the new search direction  $\mathbf{p}_{k+1}$  is chosen to be as different from all the previous ones as possible. Let us define  $\mathbf{p}_k$  and  $\mathbf{p}_l$  too be *conjugate vectors* if they obey:

$$\mathbf{p}_k \cdot \mathbf{A} \cdot \mathbf{p}_l = 0 \quad (9.37)$$

i.e. they are, so to speak, orthogonal with respect to the “metric”  $\mathbf{A}$ . In the conjugate gradient method each new search direction is required to be conjugate to all previous ones. This choice of  $\mathbf{p}_{k+1}$  ensures that if one finds the  $\alpha_{k+1}$  such that  $f(\mathbf{q}_{k+1} + \alpha_{k+1} \mathbf{p}_{k+1})$  is minimized with respect to  $\alpha_{k+1}$  it is also minimized with respect to all previous  $\alpha_{l < k+1}$  and  $\mathbf{p}_{l < k+1}$ . This means that if

one has  $N = N_x N_y N_z$  grid points, then this method is guaranteed to find the exact solution in at most  $N$  iterations. In practice, however, one does not want to store all previous search directions, nor do the large amount of work to find a conjugate to a large number of vectors. So in practice the number of successively conjugate search directions is taken to be a limited value, and after that the algorithm is restarted. This does not guarantee convergence in a finite number of steps, but still it converges relatively rapidly.

Another version of this algorithm uses another scalar function to minimize: the length of the residual vector  $\mathbf{r}$ . This residual is defined as:

$$\mathbf{r}_k = \mathbf{A}\mathbf{q}_k - \mathbf{b} \quad (9.38)$$

The function to minimize is then:

$$f(\mathbf{q}) = \frac{1}{2} \mathbf{r} \cdot \mathbf{r} = \frac{1}{2} |\mathbf{A}\mathbf{q} - \mathbf{b}|^2 \quad (9.39)$$

This is the *minimum residual method*. It works for symmetric matrices, also those which are non-positive-definite.

The above methods work fine, but they are restricted to symmetric matrices. There are several generalizations of this to generalize it to more general matrices. For the above mentioned basic conjugate gradient method there is a *biconjugate gradient* method (BCG method) which does not have this restriction. But it is also not clearly linked to minimizing a function.

For the minimum residual method there is a generalized version called the *generalized minimum residual* method (GMRES). This is a very stable method and does not have any restriction on the matrix, except that it is not degenerate.

#### 9.4.2 Implementation and preconditioning

The nice property of this class of methods, also called *Krylov subspace methods*, is that it involves only multiplications of the matrix  $\mathbf{A}$  or  $\mathbf{A}^T$  with vectors supplied by the algorithm and the computation of inner products between two vectors supplied by the algorithm. In fact, you can, as a user, supply your own subroutine for multiplying either  $\mathbf{A}$  or  $\mathbf{A}^T$  with a vector that the conjugate gradient supplies, as well as a subroutine for computing the inner product between two vectors. This leaves all the implementation and all the time-consuming computations in your own hands and you can figure out how to most efficiently compute these matrix-vector and vector-vector products.

This nice property also reveals a weakness of these methods. By definition a matrix product with a vector only propagates information from one grid point to its neighbors. So if one has an  $N_x \times N_y$  grid then one cannot hope to get a solution with fewer than  $\max(N_x, N_y)$  iterations, and generally it may take up to  $\max(N_x^2, N_y^2)$  iterations. The reason is simply that one cell on the left of the grid does not “feel” anything from another cell on the right side of the grid until this information is propagated. Each iteration of the Krylov method propagates information only a single cell, and since information may have to travel multiple times back-and-forth before a solution is found, it can take many iterations.

More mathematically one can say that this matrix is *ill conditioned*. If  $\lambda_1 \cdots \lambda_N$  are the eigenvalues of the matrix, then the condition number  $\kappa$  is defined as

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \quad (9.40)$$

If this number is large then the matrix is ill conditioned

One way of improving the situation is to use *preconditioning* of the matrix equation by introducing a preconditioning matrix  $\Lambda$  such that one solves the following equation:

$$\Lambda A \Lambda^{-1} \Lambda \mathbf{q} = \Lambda \mathbf{b} \quad (9.41)$$

This equation is mathematically identical to the original one, but it is numerically different. If one can find a suitable preconditioning matrix  $\Lambda$ , then the convergence can be sped up much. Ideally  $\Lambda$  is as closely similar as  $A$  but still easy to invert. In the Krylov methods what happens in practice is that the algorithm will ask you to solve the equation  $\Lambda \mathbf{y} = \mathbf{w}$  for  $\mathbf{y}$  for some given vector  $\mathbf{w}$ . If you take  $\Lambda = 1$ , then this is trivial:  $\mathbf{y} = \mathbf{w}$  and the method reduces to the non-preconditioned method. If, however,  $\Lambda$  is cleverly chosen, then you can solve the equation each time the algorithm asks you to do so, and the method can speed up a lot. Basically what you need to do is to find a  $\Lambda$  that propagates information quickly across the grid without making it difficult to invert.

### 9.4.3 Libraries

It is in general not a good idea to try to program one's own matrix equation solver. Libraries of subroutines for doing this have been developed over decades, and it is generally a good idea to look for an appropriate library and use that library as a "black box". Here is a very incomplete list of some known libraries, some of which are freeware, some of which are proprietary software:

- The subroutines associated with the book by Press et al. "Numerical Recipes in C" and/or "Numerical Recipes in Fortran-77".  
<http://www.nr.com/com/storefront.html>
- SixPack: Linear solvers for Finite Volume / Finite Difference equations.  
<http://www.engineers.auckland.ac.nz/~snor007/software.html>
- Hypre: Scalable Linear Solvers.  
<https://computation.llnl.gov/casc/hypre/software.html>
- PETSc: Portable, Extensible Toolkit for Scientific Computation.  
<http://www-unix.mcs.anl.gov/petsc/petsc-as/>

But there are many more libraries available.

## 9.5 Incompressible fluid equations

Now let us return to the problem of very subsonic fluids. Very subsonic means in fact that one can approximate the system as being *incompressible*. This means that

$$\nabla \cdot \vec{u} = 0 \quad (9.42)$$

and the density  $\rho = \text{constant}$ . Since the density is assumed to be constant, we do not need to solve the continuity equation anymore. But instead we get a condition on the pressure gradients. Let us look at the momentum equation:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla P \quad (9.43)$$

If we take the divergence of this equation, then we obtain:

$$\frac{\partial \nabla \cdot \vec{u}}{\partial t} = -\nabla \cdot (\vec{u} \cdot \nabla \vec{u}) - \frac{1}{\rho} \nabla^2 P \quad (9.44)$$

where we have used  $\nabla \rho = 0$ . If we say  $\nabla \cdot \vec{u} = 0$  then we obtain

$$\nabla^2 P = -\rho \nabla \cdot (\vec{u} \cdot \nabla \vec{u}) \quad (9.45)$$

In index notation, and with use again of  $\nabla \cdot \vec{u} = 0$  we obtain

$$\nabla_k \nabla_k P = -\rho \nabla_k u_l \nabla_l u_k \quad (9.46)$$

This means that the condition of incompressibility creates a *Poisson equation for the pressure*. In numerical form this becomes:

$$\frac{\partial}{\partial x_i} \left( \frac{\partial P}{\partial x_i} \right) = -\frac{\partial}{\partial x_i} \left[ \frac{\partial (\rho u_i u_j)}{\partial x_j} \right] \quad (9.47)$$

How to solve this? One way to do so is to see the similarity between the Poisson equation in 2-D and stationary solutions of the 2-D time-dependent diffusion equation. So one can use the same methods as above, but then taking  $\Delta t \rightarrow \infty$ . Or one can keep  $\Delta t$  finite or even small enough to do explicit time integration until a steady state is reached. This is, however, very CPU time consuming.

### 9.5.1 Boundary conditions

Suppose we have flow in a restricted volume (a pipe or a river), then the boundary conditions are such that  $\vec{u} \cdot \vec{n} = 0$ , where  $\vec{n}$  is the normal to the surface. The pressure gradient must be such that this remains true after one time step. Multiplying the momentum equation with  $\vec{n}$  gives:

$$\frac{\partial \vec{n} \cdot \vec{u}}{\partial t} = -\vec{n} \cdot (\vec{u} \cdot \nabla \vec{u}) - \frac{1}{\rho} \vec{n} \cdot \nabla P \quad (9.48)$$

which reduces to

$$\frac{1}{\rho} n_k \nabla_k P = -\vec{n}_k u_l \nabla_l u_k \quad (9.49)$$

which can be implemented as boundary condition.

### 9.5.2 Numerical issues

In solving the Poisson equation for the pressure one must make sure that one uses the same numerical methods for computing the source term of this equation as one uses for the update of the velocities. If one does not do this, the zero divergence cannot be guaranteed. Also one must start from a velocity field that is already divergence free, otherwise the solution of the Poisson equation will not help.

# Chapter 10

## Smooth Particle Hydrodynamics Solvers

So far we have only looked at method for solving the equations of hydrodynamics based on a fixed pre-defined grid. These methods are very suitable for many circumstances, but there are cases where these methods become quite cumbersome. In particular when very strong and localized density enhancements (clumps) appear, for instance due to gravity in astrophysical problems, the fixed-grid methods fail to resolve all relevant scales in those clumps. Using *Adaptive Mesh Refinement* (AMR) one can then refine the grid around these clumps, but such methods are technically very complex. A method that is particularly useful for modeling problems with strong clumping, and yet is fairly simple to implement, is the method of *Smooth Particle Hydrodynamics* (SPH). In this method the to-be-modeled gas cloud is represented by a set of discrete blobs of gas. These SPH particles interact with neighboring particles through a repelling force that represents the gas pressure, but otherwise act like normal particles. The nice thing of this method is that the computational power is automatically focused there where the mass is. So if a clump forms, for instance due to self-gravity, the computational power is focused in this very small region where all the mass is concentrated. It is so-to-speak an automatic AMR scheme without technical complexities. For this reason this method is often used in astrophysics, where extreme density gradients are very common.

This chapter is based on the works by Monaghan (1992, ARA&A 30, 543) and Springel (2005, MNRAS 364, 1105).

### 10.1 Lagrange equations of hydrodynamics for SPH

In Smooth Particle Hydrodynamics a gas cloud of mass  $M_{\text{cloud}}$  is respresented by  $N$  ‘particles’, i.e.  $N$  gas subclouds of mass  $m = M_{\text{cloud}}/N$ . Each of these blobs moves according to the equations of Newtonian mechanics, including a pressure force. For each of these particles the equations are based on Lagrange form of the Euler equations (cf. Chapter 1) including the equation of propagation:

$$D_t \vec{x} = \vec{u} \quad (10.1)$$

$$D_t \rho = -\rho \vec{\nabla} \cdot \vec{u} \quad (10.2)$$

$$D_t \vec{u} = -\frac{\vec{\nabla} P}{\rho} \quad (10.3)$$

$$TD_t s = 0 \quad (10.4)$$



To derive expressions for  $\nabla P$  and  $\nabla \cdot \vec{u}$  we need to let go of the ‘particle’ picture and view the ‘particles’ instead as *comoving coordinates* rather than true particles. In this picture the flow is again described on a set of grid points, but this time the grid points are irregularly spaced (not in a mesh) and they move strictly along with the flow. If we now wish to compute numerical derivatives, we have no longer the luxury of a grid-like coordinate system in which it is obvious which gridpoints are neighbors of grid point  $i, j$  (namely  $i, j - 1; i, j + 1; i - 1, j; i + 1, j$ ), but an irregularly spaced set of neighbors.

## 10.2 The SPH Kernel

The way this practical problem is solved in the SPH method is by defining a *kernel*  $W(\vec{x} - \vec{x}', h)$  around every  $\vec{x}'$  location of an SPH particle. This is a function that is maximum at  $\vec{x} = \vec{x}'$  and falls off smoothly to zero with distance from  $\vec{x}'$ . Its value only depends on  $r \equiv |\vec{x} - \vec{x}'|$ . This function defines the shape of the SPH ‘blob’. The value  $h$  is the parameter which defines the ‘size’ of this blob and is called the *smoothing length*. The kernel is normalized to unity in the following way:

$$\int_V W(\vec{x} - \vec{x}', h) d\vec{x}' = 1 \quad (10.5)$$

and we also have

$$\lim_{h \rightarrow 0} W(\vec{x} - \vec{x}', h) = \delta(\vec{x} - \vec{x}') \quad (10.6)$$

Again, we should not regard this ‘blob’ as a true blob of gas. In fact, SPH blobs will overlap quite a bit, which would not be true for true balls of gas. We should regard the SPH kernel as defining a *region of influence* of the SPH particle. It will turn out that only SPH particles that lie within each others regions of influence will be able to interact with each other.

Now let us, for a brief moment, do some math with the function  $W(\vec{x} - \vec{x}', h)$ , forgetting any reference to discrete SPH particles. Suppose that we have some function  $A(\vec{x}')$  defined at every position in space. We can now define the convolution of  $A(\vec{x}')$  with the kernel  $W(\vec{x} - \vec{x}', h)$ :

$$\tilde{A}(\vec{x}) = \int_V A(\vec{x}') W(\vec{x} - \vec{x}', h) d\vec{x}' \quad (10.7)$$

In this way the function  $A(\vec{x}')$  is blurred using the kernel  $W(\vec{x} - \vec{x}', h)$  as some kind of point spread function. If we now return to the discrete SPH picture, the values of  $\vec{x}'$  are now the set of discrete positions of the SPH particles, and the integral must be replaced by a sum in some way. To do this we must find a discrete expression for the integration volume  $d\vec{x}'$ . It turns out that the way to express  $\tilde{A}(\vec{x})$  is

$$\tilde{A}(\vec{x}) = \sum_{i=1, N} \frac{m}{\rho_i} A_i W(\vec{x} - \vec{x}_i, h) \quad (10.8)$$

where  $\rho_i$  is the density of gas corresponding to SPH particle  $i$ , for which we will derive an expression later, and  $A_i$  is the value of the function  $A$  corresponding to SPH particle  $i$ . Let us look carefully at this expression. The ratio  $m/\rho_i$  gives the volume that SPH particle  $i$  takes up. So if you have a situation where 100 SPH particles are spread over a large volume but another 100 SPH particles are jammed within a clump with volume much less than  $h^3$ , then the ratio  $m/\rho_i$  makes sure that the 100 particles in the clump together count for only the volume they occupy, which is much less than the volume occupied by the other 100 SPH particles. The ratio



$m/\rho_i$  therefore makes sure that the sum over SPH particles in fact represents an integral over *volume* and not over mass. Therefore, Eq. (10.8) is the discrete representation of Eq. (10.8).

The usefulness of Eq. (10.8) now lies in the fact that we have produced a smooth function  $\tilde{A}(\vec{x})$  from a function only known at discrete points  $A_i$ . It is not guaranteed that  $\tilde{A}(\vec{x}_i) = A_i$ , but it will also not be too far from this value. Later we will replace  $A$  by the pressure  $P$  or the density  $\rho$  so that we can use the above expressions to create a smooth pressure and density function from the discrete one. But let us for now stick to the abstract function  $A$ . Now that we have this smooth function we can define the derivative:

$$\nabla \tilde{A} = \sum_{i=1,N} \frac{m}{\rho_i} A_i \nabla W(\vec{x} - \vec{x}_i, h) \quad (10.9)$$

where we see that the derivative operator eventually works only on the kernel. Since we know the shape of the kernel analytically, we know the derivatives analytically.

The two most common kernels are a Gaussian kernel

$$W(r, h) = \frac{1}{h\sqrt{\pi}} e^{-r^2/h^2} \quad (10.10)$$

and the spline kernel <sup>1</sup>

$$W(r, h) = \frac{8}{\pi h^3} \begin{cases} 1 - 6 \left(\frac{r}{h}\right)^2 + 6 \left(\frac{r}{h}\right)^3, & 0 \leq \frac{r}{h} \leq \frac{1}{2} \\ 2 \left(1 - \frac{r}{h}\right)^3, & \frac{1}{2} < \frac{r}{h} \leq 1 \\ 0, & \frac{r}{h} > 1 \end{cases} \quad (10.11)$$

The Gaussian Kernel may be more natural but has the disadvantage that strictly speaking the region of influence of each SPH particle is infinite, even though the kernel may have very low values at large distances. The spline kernel has a very strictly defined outer edge, so that we can a-priori ignore anything that is further away from an SPH particle than a distance of  $h$ . For a computer implementation of the SPH method this property will save enormous amounts of computing time, because the sum over all particles then reduces to the sum over all particles that have a distance less than  $h$  to the point  $\vec{x}_i$ .

→ **Exercise:** The expression for the spline kernel, Eq. (10.11), has a normalization constant  $8/\phi h^3$  that was chosen such that the kernel integrates to unity in 3-D. Derive the normalization for the case of SPH in 1-D.

### 10.3 Some expressions in SPH-form

Let us write down some expressions that will be useful for later. First of all the smoothed form of the density  $\tilde{\rho}$  at a given position  $\vec{x}$ :

$$\tilde{\rho}(\vec{x}) = \sum_{i=1,N} m W(\vec{x} - \vec{x}_i, h) \quad (10.12)$$

This has the desired effect that the more SPH particles are located in the same small region with size  $L < h$ , the higher is the density. However, if the SPH particles are further away from each

<sup>1</sup>Note: This is the definition used by Springel (2005, MNRAS 364, 1105). Monaghan (1992, ARA&A 30, 543) uses a formula that is equal to this one if his  $h$  is replaced by  $h/2$ .

other than  $h$ , then the SPH kernels do not overlap and the gas density around each SPH particle simply reflects the kernel of this particle. We see here that SPH (at least with a globally fixed value of  $h$ ) has the unfortunate property that it cannot model regions of very low density. It will instead model such regions as being filled with localized blobs of gas separated by vacuum. A variable smoothing length  $h \rightarrow h_i$  can solve this problem, but let us save this for later.

Now one may raise the question, what is the density  $\rho_i$  associated with each SPH particle, which need not necessarily be equal to  $\tilde{\rho}(\vec{x}_i)$ ? One way to define this is using the following expression:

$$\frac{4\pi}{3}h^3\rho_i = N_{h,i}m \quad (10.13)$$

where  $N_{h,i}$  is the number of nearest neighboring particles that happen to lie within a radius  $h$  of particle  $i$ .

→ **Exercise:** Argue that both Eq. (10.12) and Eq. (10.13) produce the same result for the case of regularly spaced SPH particles with inter-spacing distance  $\Delta x \ll h$ .

Now what about  $\nabla \cdot \tilde{\mathbf{u}}$ ? We have

$$\tilde{\mathbf{u}}(\vec{x}) = \sum_{i=1,N} \frac{m_i}{\rho_i} W(\vec{x} - \vec{x}_i, h) \mathbf{u}_i \quad (10.14)$$

So we get

$$\nabla \cdot \tilde{\mathbf{u}}(\vec{x}) = \sum_{i=1,N} \frac{m_i}{\rho_i} \mathbf{u}_i \cdot \nabla W(\vec{x} - \vec{x}_i, h) \quad (10.15)$$

But we can also derive an alternative expression. Let us first write

$$\nabla \cdot \tilde{\mathbf{u}} = \frac{1}{\tilde{\rho}} \left[ \nabla \cdot (\tilde{\rho} \tilde{\mathbf{u}}) - \tilde{\mathbf{u}} \cdot \nabla \tilde{\rho} \right] \quad (10.16)$$

We have

$$\nabla \cdot (\tilde{\rho} \tilde{\mathbf{u}}) = \sum_{i=1,N} m_i \mathbf{u}_i \cdot \nabla W(\vec{x} - \vec{x}_i, h) \quad (10.17)$$

$$\nabla \tilde{\rho} = \sum_{i=1,N} m_i \nabla W(\vec{x} - \vec{x}_i, h) \quad (10.18)$$

So we obtain:

$$\rho_k(\nabla \cdot \tilde{\mathbf{u}})_k = \sum_{i=1,N} m_i (\vec{u}_i - \vec{u}_k) \cdot \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.19)$$

where  $\nabla_k$  is the derivative operator with respect to  $x_k$ . Now a little check: If we insert the Gaussian kernel we obtain

$$\rho_k(\nabla \cdot \tilde{\mathbf{u}})_k = \sum_{i=1,N} \frac{2m_i}{h^2} (\vec{u}_k - \vec{u}_i) \cdot (\vec{x}_k - \vec{x}_i) W(\vec{x}_k - \vec{x}_i, h) \quad (10.20)$$

For each pair of SPH particles this expression indeed gives a positive contribution when these particles move away from each other, as it should be.

## 10.4 The SPH equations of motion

With the expressions we derived above, and with the exercise we have obtained in manipulating derivatives with SPH, we can now attempt to write down the final set of SPH equations.

### 10.4.1 Momentum equation

The SPH version of the momentum equation starts from

$$D_t \vec{u}_k = - \left( \frac{\vec{\nabla} P}{\tilde{\rho}} \right)_k \quad (10.21)$$

Following the above examples, we can write  $\nabla \tilde{P}$  in various forms. For instance

$$(\nabla \tilde{P})_k = \sum_{i=1,N} \frac{m_i}{\rho_i} P_i \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.22)$$

or

$$(\nabla \tilde{P})_k = \frac{1}{\rho_k} \sum_{i=1,N} m_i (P_i - P_k) \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.23)$$

Neither of these expressions conserves momentum. But if we write

$$\frac{\nabla P}{\rho} = \nabla \left( \frac{P}{\rho} \right) + \frac{P}{\rho^2} \nabla \rho \quad (10.24)$$

then we obtain

$$(\nabla \tilde{P})_k = \rho_k \sum_{i=1,N} m_i \left( \frac{P_i}{\rho_i^2} + \frac{P_k}{\rho_k^2} \right) \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.25)$$

The momentum equation now becomes

$$D_t \vec{u}_k = - \sum_{i=1,N} m_i \left( \frac{P_i}{\rho_i^2} + \frac{P_k}{\rho_k^2} \right) \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.26)$$

This has the desirable property that it obeys the ‘action is minus reaction’ of Newtonian mechanics: the fact that momentum is conserved. To see this we write the momentum equation as a force equation:

$$D_t (m_k \vec{u}_k) = - \sum_{i=1,N} m_k m_i \left( \frac{P_i}{\rho_i^2} + \frac{P_k}{\rho_k^2} \right) \nabla_k W(\vec{x}_k - \vec{x}_i, h) \quad (10.27)$$

We see for each pair of SPH particles that the gain of momentum of one is the gain of opposite momentum of the other, exactly conserving momentum as desired.

### 10.4.2 Energy equation

The energy equation can be written in the form

$$T D_t s = 0 \quad (10.28)$$

where  $s$  is the entropy. Basically this means that each SPH particle has a specific entropy, and it keeps this entropy as the particle moves along. Another way of saying this is that if the density  $\rho_i$  is known, the pressure follows from

$$P_i = K_i \rho_i^\gamma \quad (10.29)$$

where  $K_i$  is another way of writing the entropy (see Eq. 1.22 of Chapter 1). So, for each SPH particle  $K_i$  remains constant in time. Of course, if a shock appears, then the entropy must increase, which is discussed in Section 10.6.

### 10.4.3 Propagation equation

The next thing to do is to integrate the motion of the SPH particles, i.e. to compute the  $\vec{x}_i(t)$ . In doing so we must keep in mind that we would like elementary conserved quantities to remain conserved as much as possible. Mass is obviously conserved, so that is no problem. Total energy is complicated to keep globally conserved, but specific entropy is conserved for each SPH particle. Momentum conservation requires that if two SPH particles exchange momentum through their mutual pressure force, the one must receive exactly what the other loses. A simple first order integration scheme can be built using operator splitting. The spatial advection operator is:

$$x_i \rightarrow x_i + \Delta t u_i \quad (10.30)$$

This is a *drift operator*. Then we do the pressure forces:

$$u_i \rightarrow u_i + \Delta t f_i \quad (10.31)$$

where  $f_i$  is the pressure force computed using the methods described above, expressed at the new location  $x'_i$ . To get slightly better accuracy one often does:

$$x_i \rightarrow x_i + \frac{1}{2} \Delta t u_i \quad (10.32)$$

$$u_i \rightarrow u_i + \Delta t f_i \quad (10.33)$$

$$x_i \rightarrow x_i + \frac{1}{2} \Delta t u_i \quad (10.34)$$

This is called the *leapfrog* method. It has the advantage that it is very simple and it has the benign property that it is a so-called *symplectic integrator*. Such an integrator is particularly well suited for equations based on a Hamiltonian, and conserves the Poincare integral invariants. In particular for systems with gravity this is useful, for instance when integrating the orbit of a particle around a star. This method conserves the orbital elements much better than ordinary integrators such as Runge-Kutta. We refer to Springel (2005, MNRAS 364, 1105) and references therein for an in-depth discussion of the leapfrog method, symplectic integrators in general and their properties.

## 10.5 Variable smoothing length

As mentioned above, one of the problems of taking a fixed smoothing length for all SPH particles is that at low densities the gas is not well represented: it is represented by isolated blobs. There is also an opposite problem with a constant smoothing length: in regions of very dense clustering of SPH particles, where hundreds or thousands of SPH particles are all within a region the size of  $h$ , the spatial resolution can not become any better than  $h$ , and much computing power is wasted on these thousands of particles. A variable smoothing length is therefore of absolute necessity if problems of high density contrast are to be modeled.

One simple way to define the variable smoothing length of particle  $i$  is to take it such that there are always  $N_{h,i} = N_{nb}$  nearest neighbors within a radius  $h_i$  from that position, and where  $N_{nb}$  is a constant for the simulation. This constant can be typically taken to be about 60, so that in each direction there are roughly two ‘layers’ of particles within the radius  $h_i$ .

With variable smoothing length one must reconsider some of the equations in which the interaction between two SPH particles with potentially different smoothing lengths is computed.

For the momentum equation Springel & Hernquist (2002, MNRAS 333, 649) derive the following expression:

$$D_t \vec{u}_k = - \sum_{i=1,N} m_i \left( f_i \frac{P_i}{\rho_i^2} \nabla_k W(\vec{x}_k - \vec{x}_i, h_i) + f_k \frac{P_k}{\rho_k^2} \nabla_k W(\vec{x}_k - \vec{x}_i, h_k) \right) \quad (10.35)$$

with  $f$  defined as

$$f_i = \left( 1 + \frac{h_i}{3\rho_i \frac{\partial \rho_i}{\partial h_i}} \right)^{-1} \quad (10.36)$$

## 10.6 Shocks: artificial viscosity

As mentioned above, using the energy equation in the form of  $Tds = 0$  guarantees perfect conservation of specific entropy for each SPH blob. However, if the gas moves through a shock, then physically it is clear that entropy *must* be generated. It is therefore clear that if we do not include some form of artificial viscosity, such as the von Neumann-Richtmyer viscosity, then our solutions are bound to fail miserably. The way this artificial viscosity is implemented in Gadget-2 (see Section 10.8) is

$$(D_t \vec{u}_k)_{\text{visc}} = - \sum_{i=1,N} m_i \Pi_{ki} \nabla_k \bar{W}(\vec{x}_k - \vec{x}_i) \quad (10.37)$$

(see Springel 2005, MNRAS 364, 1105), where  $\Pi_{ki} \geq 0$  is only non-zero when particles approach each other. The associated increase in ‘entropy’  $K_i$  is

$$(D_t \vec{u}_k)_{\text{visc}} = \frac{1}{2} \frac{\gamma - 1}{\rho_i^{\gamma-1}} \sum_{i=1,N} m_i \Pi_{ki} (\vec{u}_k - \vec{u}_i) \nabla_k \bar{W}(\vec{x}_k - \vec{x}_i) \quad (10.38)$$

where  $\bar{W}$  is the arithmetic average of the two kernels. An example of a recipe for the artificial viscosity is that of Monaghan & Gingold (1983) and Balsara (1995):

$$\Pi_{ki} = \begin{cases} (-\alpha c_{ki} \mu_{ki} + \beta \mu_{ki}^2) / \rho_{ki} & \text{if } \vec{u}_{ki} \cdot \vec{x}_{ki} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (10.39)$$

with

$$\mu_{ki} = \frac{h_{ki} \vec{u}_{ki} \cdot \vec{x}_{ki}}{|\vec{x}_{ki}|^2} \quad (10.40)$$

where  $\vec{u}_{ki} = \vec{u}_k - \vec{u}_i$ ,  $\vec{x}_{ki} = \vec{x}_k - \vec{x}_i$ , and  $h_{ki}$ ,  $\rho_{ki}$  and  $c_{ki}$  are the arithmetic mean of  $h$ ,  $\rho$  resp. the sound speed of both particles. Typically the values of  $\alpha$  and  $\beta$  are taken to be  $\alpha \simeq 0.5 - 1.0$  and  $\beta = 2\alpha$ .

## 10.7 Some thoughts about SPH

SPH is a flexible method for hydrodynamics. One can relatively easily build in various source terms and forces such as for instance gravity. For very large simulations it becomes a challenge to manage the particles and to make sure that the nearest neighbors are quickly found. Moreover, to manage gravitational forces over both short and long ranges requires rather sophisticated

schemes such as *tree codes* which organize the particles in a way that near particles are quickly found. A nice aspect of SPH is that it is Lagrangian. One can easily include for instance chemistry of the gas, as long as no mixing between gas in neighboring SPH particles is required. If that is necessary, a special recipe for mixing must be used, which is a bit more complicated. Another nice aspect of the method is that it has no numerical bulk viscosity. But one has to pay a price: the numerical shear viscosity can be enormous. If too few SPH particles are used, the intrinsic clumpiness of SPH particles causes a strong shear viscosity which is usually bigger than intrinsic physical shear viscosity of the system to be modeled. The shear viscosity can be suppressed only by going to very large numbers of SPH particles, but that is numerically costly. It is sometimes reported that SPH is intrinsically too diffusive for certain problems. Also it is very difficult to build in physics like radiative transfer into the SPH method.

All in all the method is very useful for many problems of astrophysics, as long as the many inaccuracies and potential problems are kept in mind.

## **10.8 The code GADGET-2**

There are many SPH and N-body codes in the literature, and many can be downloaded from the web. A code that has gained particular popularity in recent years in the cosmological community and elsewhere is the code GADGET-2 by Volker Springel (Max-Planck-Institut für Astrophysik in Garching, Germany). The code is well documented, well tested, is easy to install and use and it is accompanied by a scientific paper describing the methodology of the code in detail (Springel 2005, MNRAS 364, 1105). It allows, among many things, for SPH gas dynamics and at the same time for N-body dynamics, including gravitational interactions between all bodies. The code can be downloaded from: <http://www.mpa-garching.mpg.de/gadget/>.

# Chapter 11

## Afterword

Now that this lecture is over, here are a few words of thought. In this lecture we have discussed various numerical techniques for solving hydrodynamics problems. For simplicity we have focused mainly on 1-D, but we have seen that such 1-D techniques (for cell-centered algorithms) are easily generalized to 2-D and 3-D. In principle this knowledge allows you to write your own 3-D hydrodynamics solver for Cartesian coordinate systems. The question is, of course, should you? There are numerous ready-to-use codes on the web for doing numerical hydrodynamic simulations which include much physics such as (self-)gravity, magneto-hydrodynamics, radiative transfer etc, and numerical sophistication such as Adaptive Mesh Refinement. Each of these codes has its own advantages and disadvantages, and it requires some “shopping around” to find the right code for the purpose you have. The main advantages of using ready-to-use codes are that this a) saves enormous amounts of development time, b) gives you a code that is likely to be well-tested, c) optimized in speed, d) provides many options that may be too complex to develop yourself (such as AMR), e) does not require you to know all the intricate details of such codes and finally f) usually provides you with a user group whom you can ask question to. Usually these arguments are enough to make the decision clear to use ready-to-use codes. However the black-box use of such codes can be highly dangerous. A reasonable knowledge of what goes on inside is important in order to know what you are doing. This lecture was meant to convey this knowledge.

However, I would also like to say a word of caution. The fact that codes have been around for many years does not mean at all that they are bug-free. There will always be corners of parameter space in which these codes were not well enough tested and in which bugs are still present. If you have a bit more than just a standard purpose for a code you are likely to treat into new territory where the code could produce unpredictable and untested behavior. One main advantage of writing one’s own code is that one knows exactly where the code has been tested and where not. One also knows very well what the behavior of the code is because one has necessarily spent months if not years to develop the code. Developing your own code costs enormous amounts of time and in general a bit more knowledge than that which has been shown in this lecture.

All in all it appears that for most purposes it would be best to use an off-the-shelf code, but use it with great caution and spend weeks if not months just to test the behavior of the code against various test problems, some of which have been published in scientific papers, others of which are analytic solutions (e.g. the Sod shock tube problem) and again others are described on web sites of famous codes.

Here is a list of some well-known codes used in astrophysics (no claim to be complete!!):



**FLASH code:** <http://flash.uchicago.edu/website/home/>

Major project for 3-D Cartesian hydro, many modules, AMR, parallelization, large user base etc. Specialized for conflagration problems (supernova explosions etc). This project is financed by the Department of Energy (DOE), meaning that it originates from nuclear weapons research.

**PENCIL code:** <http://www.nordita.org/software/pencil-code/>

Widely used MHD code for Cartesian coordinates featuring a higher order advection method. Mainly used for shearing box calculations of turbulence in protoplanetary disks, but also used for global simulations of disks or other objects. Parallelized.

**ZEUS code:** <http://www.astro.princeton.edu/~jstone/zeus.html>

One of the most famous (M)HD codes in astronomy, written by Jim Stone. Very widely used classical solver. It uses a staggered grid, can be used in various coordinate systems (cartesian, cylindrical, polar), features gravity, self-gravity and radiation transfer. Perhaps one of the most flexible allround astrophysics codes for (magneto)hydrodynamics.

**PLUTO code:** <http://plutocode.to.astro.it/>

A versatile Riemann solver code for HD and MHD with AMR capabilities. Parallelized with MPIA. Curvilinear coordinates supported. Also relativistic flows supported.

**ATHENA code:** <http://www.astro.princeton.edu/~jstone/athena.html>

Stone's new code based on various types of Riemann solvers (including also Roe's algorithm). For now this code can only do Cartesian coordinates (though apparently a mode for cylindrical and polar coordinates is being developed). It features MHD, AMR, parallelization (MPI) etc.

**GADGET code:** <http://www.mpa-garching.mpg.de/gadget/> A well documented SPH and N-body code for astrophysical applications. Has a large user base.

Keep also in mind that there are a great number of well-tested codes that are not publically available, but are available upon request (or in collaboration with) the author of that code.

So all that remains now is to say: have fun and success with numerical hydrodynamics!



# Appendix A

## The index notation of tensor calculus

The index notation is a very powerful way of dealing with vectors, matrices and tensors. The topic of tensor mathematics and the index notation is far too far-reaching to cover in depth in this short appendix. Here we only cover the absolute basics.

A vector (i.e. an object with spatial direction) is denoted in normal vector notation as  $\vec{u}$ , or sometimes also as  $\mathbf{u}$  (in this lecture we write  $\vec{u}$ ). A vector is a tensor of rank 1. In index notation we refer to the individual components of the vector by writing it as  $u_i$ , where  $i$  is an index that can be chosen to be either 1, 2 or 3.

A matrix is often written as  $\mathbf{M}$ . It has two indices, i.e. 9 components (if related to true space, i.e. if it is a transformation in space like a rotation). A matrix is a tensor of rank 2. In index notation it is written as  $M_{ij}$ , where the indices  $i$  and  $j$  are 1, 2 or 3. The unit matrix is  $\delta_{ij}$ , which is defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (\text{A.1})$$

which is called the *Kronecker delta function* in tensor mathematics.

A matrix is not the only kind of second rank tensor. The pressure tensor or stress tensor in hydrodynamics is also one, and in many respects that kind of object behaves very similar to a matrix. But formally it is not entirely the same. This can not be seen as long as one works in cartesian coordinates: then the pressure/stress tensor behaves exactly like a matrix. But in general coordinate transformations, in which the local basis is no longer cartesian, the two kinds of objects behave differently. For details see the tensor syllabus mentioned above, or other literature on tensor calculus. For completeness: a matrix is *contravariant* in the first index, and *covariant* in the second (or vice-versa) while the stress tensor is contravariant in both indices. Again, as long as one remains in cartesian coordinates (i.e.  $x$ ,  $y$  and  $z$  or rotations thereof) this distinction is irrelevant.

A very important thing of the index notation is the *summation convention*. This convention says that *if an index appears twice in the same product, then it is automatically assumed that this is being summed over, from 1 to 3*. So the following expression is a matrix multiplication with a vector:

$$M_{ik}u_k \equiv \sum_{k=1}^3 M_{ik}u_k \equiv \sum_{m=1}^3 M_{im}u_m \equiv M_{im}u_m \quad (\text{A.2})$$

and all four expressions are (by the summation convention) the same.

The inner-product of a vector  $\vec{u}$  with  $\vec{w}$  is then written as:

$$\vec{u} \cdot \vec{w} = u_i w_i \quad (\text{A.3})$$

(Note: here  $u_i$  and  $w_k$  are not the  $(u, v, w)$  components of  $\vec{u}$ , as used often in this lecture, but they are the  $i$ -th component of  $\vec{u}$  and the  $k$ -th component of vector  $\vec{w}$  respectively here.)

The divergence of a vector field is written as:

$$\nabla \cdot \vec{u} = \nabla_i u_i \equiv \partial_i u_i \quad (\text{A.4})$$

The two ways of writing (with  $\nabla_i$  or  $\partial_i$ ) are both often used, and I will use both ways of writing in this lecture without notice. The divergence of a tensor field  $\rho \vec{u} \vec{u}$  (in index notation  $\rho u_i u_j$ ) is:

$$\nabla \cdot (\rho \vec{u} \vec{u})|_{\text{component } k} = \nabla_i (\rho u_i u_k) \quad (\text{A.5})$$

# Appendix B

## A brief introduction to IDL/GDL

Although the exercises may be done in any true programming language that the student is familiar with (e.g. C, C++, F77, F90, F95, PASCAL, ...), we will give support mainly in a simple programming environment called IDL (Interactive Data Language). Since IDL is proprietary software, and licenses are extremely expensive, we will make use of an open-source clone of IDL, called GDL (Gnu Data Language). This project is still under development, so it will not have the complete functionality of IDL. However, it will be sufficient for making small programs and making plots.

The main advantages of using GDL for the exercises compared to many other programming languages are:

1. The programming language is *interpreter-based*, meaning that you do not compile a program first and then run it from the command-line, but you go into GDL, get a GDL-prompt and you can do command-line calculations. The prompt is so-to-speak the main routine, and any command you type is immediately executed. It is therefore an *interactive* programming language. At the same time you can also run a program (which is almost - but not 100% - the same as a series of command-line statements bound together in a file or in a subroutine).
2. You can plot the results of your programs immediately, without having to write results, read them into GNUPLOT or something similar.
3. If a program stops due to a STOP statement or an error, then the prompts is stuck right there in the subroutine where the stop or error took place. So you can type “print, a” to see what variable a contains, even if this is a local variable in a subroutine. By deliberately putting in STOP statements into a program that contains an unfixed bug you can debug the code. A simple “.c” is sufficient to continue the program after that stop statement. This is the ultimate debugging tool!
4. The GDL programming language is very easy (although it has its nasty unlogical things, too): it is very similar to the BASIC programming language.
5. It has the possibility to handle large arrays at once, with a single command. This requires a bit of exercise, but can be very handy.

The disadvantages are:

1. GDL is slow as a programming language. For multi-D hydrodynamics modeling it is utterly inadequate. But we shall use it as an *interface* for external Fortran programs. In GDL we will set up the problem, write a set of input files for the Fortran program, then we run the Fortran program externally from GDL, and at the end we read the output files of the Fortran program into GDL for plotting. This will be done for all more complex problems involving multi-D hydrodynamics.
2. GDL is still an experimental package, so sometimes things that work brilliantly in IDL do not work in GDL. We will have to see when this takes place, although this presumably does not take place for the basic things we want to do.

## B.1 Getting started

Let us make a few command-line exercises. First we go into GDL by typing:

```
> gdl
```

in the prompt (the `>` is the shell prompt). We then get a GDL prompt:

```
gdl>
```

From now on whenever there is a “gdl>” we mean that this is the prompt, and the text behind it is what we type in. Any next lines are usually the result of what we type. So now type

```
gdl> print, 'Hello world'
Hello world
```

to get the famous sentence. We can define a variable:

```
gdl> a=7
gdl> print, a
7
```

By typing `a=7` instead of `a=7.` we tell GDL that `a` is an integer and not a floating-point variable. So of we do

```
gdl> a=7
gdl> print, a/2
3
```

But now this:

```
gdl> a=7.
gdl> print, a/2
3.5
```

Note that there is a difference between real (\*8) floating points and doubleprecision (\*16) floating points:

```

gdl> a=7.e0
gdl> print,sqrt(a),format='(E29.22)'
  2.6457512378692626953125E+00
gdl> a=7.d0
gdl> print,sqrt(a),format='(E29.22)'
  2.6457513110645907161711E+00

```

and notice the way that formatting works. To find out what type a variable has we can type

```

gdl> help,a
<Expression>      DOUBLE      =      7.0000000

```

Now type

```

gdl> a=fltarr(10)
gdl> print,a
  0.00000    0.00000    0.00000    0.00000
  0.00000    0.00000    0.00000    0.00000
  0.00000    0.00000

```

This is an array of numbers. We can do

```

gdl> a=dblarr(10)
gdl> for i=0,9 do a[i]=i
gdl> print,a
  0.00000    1.00000    2.00000    3.00000
  4.00000    5.00000    6.00000    7.00000
  8.00000    9.00000

```

There is a powerful shortcut for this:

```

gdl> a=dindgen(10)
gdl> print,a
  0.00000    1.00000    2.00000    3.00000
  4.00000    5.00000    6.00000    7.00000
  8.00000    9.00000
gdl> help,a
A              DOUBLE      = Array[10]

```

Note that dblarr has a (\*8) float variant called fltarr, and dindgen has findgen. Now type

```

gdl> a = a+6
gdl> print,a
  6.0000000    7.0000000    8.0000000    9.0000000
 10.0000000    11.0000000    12.0000000    13.0000000
 14.0000000    15.0000000
gdl> b=dindgen(10)
gdl> print,a-b
  6.0000000    6.0000000    6.0000000    6.0000000
  6.0000000    6.0000000    6.0000000    6.0000000
  6.0000000    6.0000000

```

This shows the ease with which one can manipulate data in GDL.

Now for some plotting

```
gdl> nx=100
gdl> xmin=-2*!pi
gdl> xmax=2*!pi
gdl> x=xmin+(xmax-xmin)*dindgen(nx)/(nx-1.d0)
gdl> y=sin(x)
gdl> plot,x,y
```

This should produce a nice sinus curve. Now try

```
gdl> plot,x,y,psym=6
gdl> plot,x,y,psym=-6
gdl> plot,x,y,psym=3
gdl> plot,x,y,line=2
gdl> window,2
gdl> plot,y,x
gdl> wset,0
gdl> plot,y,x
```

Now with titles and stuff

```
gdl> plot,x,y,xtitle='x',ytitle='y',title='A nice plot'
```

We can also constrain the plotting domain

```
gdl> plot,x,y,xrange=[-4,4]
```

Sometimes GDL thinks a bit too much, and does things not the way we want. We can force it to:

```
gdl> plot,x,y,yrange=[-1.2,1.2]
gdl> plot,x,y,yrange=[-1.2,1.2],ystyle=1
```

Color is a bit more complex in IDL. First we have to choose if we want to have full control over the RGB colors, or if we want to have a color-map (in which 256 colors are pre-chosen and you only have to give a number 0 to 255 to choose which color).

```
gdl> rc=1L
gdl> gc=256L
gdl> bc=256L*256L
gdl> device,decomposed=1
gdl> plot,x,y,color=120*rc,80*gc,250*bc
gdl> device,decomposed=0
gdl> loadct,3
% LOADCT: Loading table RED TEMPERATURE
gdl> plot,x,y,color=160
```

The 256L (the L) stands for long-integer. We can also do this

---

```

gdl> device,decomposed=0
gdl> loadct,12
% LOADCT: Loading table 16 LEVEL
gdl> plot,x,y,color=160
gdl> plot,x,y,/nodata,color=255
gdl> oplot,x,y,color=140
gdl> oplot,x,y*0.4,color=20

```

We can also plot to postscript files.

```

gdl> set_plot,'ps'
gdl> plot,x,y
gdl> device,/close
gdl> set_plot,'x'

```

This plots to the file `idl.ps`. Note that this time the plot is on a white background...

*CAREFUL:* The device,/close is crucial!

*CAREFUL:* It is useful to set the plot back to 'x' after any 'ps' plotting, because otherwise one may wonder why one does not see anything on the screen!

We can also directly specify the file name:

```

gdl> set_plot,'ps'
gdl> device,file='myplot.ps'
gdl> plot,x,y
gdl> device,/close
gdl> set_plot,'x'

```

In postscript, color works a bit different from x. We have, for lines, only color tables, and we must explicitly tell the device to use color and how 'true' the color should be:

```

gdl> set_plot,'ps'
gdl> device,file='myplot.ps',/color,bits_per_pixel=24
gdl> loadct,12
gdl> plot,x,y,color=160
gdl> device,/close
gdl> set_plot,'x'

```

Opening and writing files

```

gdl> a=dindgen(10)
gdl> openw,1,'mydata.out'
gdl> printf,1,a
gdl> close,1

```

Reading this file back in:

```

gdl> a=dindgen(10)
gdl> openr,1,'mydata.out'
gdl> readf,1,a
gdl> close,1
gdl> print,a

```

Notice the openr instead of openw here.

Finally let us create a *structure*, which is nothing else than a package of variables put together into a single entity. It is often the most useful way to transfer heterogeneous sets of data in one go.

```
gdl> a=1.5
gdl> b='Hello'
gdl> c=1
gdl> d=[4.,5.,7.]
gdl> s={a:a,b:b,c:c,d:d}
gdl> print, s.a
      1.5000000
gdl> print, s.b+'. This is a full sentence.'
      Hello. This is a full sentence.
```

## B.2 Programming in GDL

The above commands can be grouped into a subroutine or a program. In such modules we can even use more kinds of constructions. Let us make a little program (to show that it is a file, it is marked here with <filename>, but of course this is NOT the first line of that file.)

```
<myprog.pro>
a=1.d0
for i=1,10 do begin
  a=sqrt(a+1.)
endfor
print,a
end
```

Now to run this program we type

```
gdl> .r myprog.pro
      1.6180286
```

We can also have if-statements:

```
<myprog.pro>
a=1.d0
for i=1,10 do begin
  a=sqrt(a+1.)
  if a lt 1.6 then print,'yes!'
endfor
print,a
end
```

or

```
<myprog.pro>
a=1.d0
```



```
for i=1,10 do begin
  a=sqrt(a+1.)
  if a lt 1.6 then begin
    print,'yes!'
    print,'  once more... Yes!'
  endif else begin
    print,'No...'
  endelse
endfor
print,a
end
```