Timm Krüger, Halim Kusumaatmaja,
Alexander Kuzmin, Orest Shardt,
Gonçalo Silva, Erlend Magnus Viggen

The lattice Boltzmann method

Principles and practice

July 27, 2015

Springer

Contents

Acronyms

CFD	Computational Fluid Dynamics
LB	Lattice Boltzmann
LBM	Lattice Boltzmann Method

Chapter 1
Basics of hydrodynamics and kinetic theory

THINK (OS): Introduction to hydrodynamics and kinetic theory

1.1 Navier-Stokes and continuum theory

While the lattice Boltzmann method (LBM) has found applications in fields as diverse as quantum mechanics and image processing, it has historically been and predominantly remains a computational fluid dynamics method. This is also the spirit of this book, in which we largely develop and apply the LBM for solving fluid mechanics phenomena.[1] To facilitate discussions in subsequent chapters, we therefore summarise here the most basic theory of fluid dynamics. In particular, we will review the continuity, Navier-Stokes and energy equations, which are direct consequences of conservation of mass, momentum and energy.

While this section aims to give an overview of fluid dynamics, it is necessarily somewhat brief and cannot replace a proper fluid dynamics textbook, such as [1, 2, 3]. NOTE (EMV): Please add more good fluid dynamics textbooks to this list as you see fit!

Throughout this chapter and the rest of the book we will utilise index notation, using Greek indices to denote an arbitrary component of a vector or a tensor, *e.g.* $F_\alpha \in \{F_x, F_y, F_z\}$. Repeated indices imply a summation, *e.g.* $a_\beta b_\beta = \boldsymbol{a} \cdot \boldsymbol{b}$. This style of notation is explained in more depth in appendix A.1.

[1] More advanced and non-standard applications of LBM will be covered in chapter **??**. TODO (all): add reference

1.1.1 Continuity equation

The field of fluid dynamics concerns itself with *macroscopic* phenomena of fluid motion. This implies that the fluid concept is a continuum one. Even when we speak about a fluid element, such a volume contains many molecules. This fluid element is small with respect to the system size, but is large in comparison to the size of each individual molecule and the typical distance between them. We will discuss the breakdown of this assumption in section 1.2, but for most applications in fluid dynamics this is a very robust approximation.

Let us now consider a small fluid element with density ρ which occupies some volume V_0. The mass of this fluid element is simply $\int_{V_0} \rho \, \mathrm{d}V$. If we consider the change of this mass per unit time, it must be due to fluid flow into or out of the volume element because fluid mass cannot be created or destroyed. Mathematically, this may be written as

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{V_0} \rho \, \mathrm{d}V = - \oint_{\partial V_0} \rho \boldsymbol{u} \cdot \mathrm{d}\boldsymbol{A}, \tag{1.1}$$

where the closed area integral is taken over the boundary ∂V_0 of the volume element V_0, $\boldsymbol{u}$ is the fluid velocity, and we take the outward normal as the direction of $\mathrm{d}\boldsymbol{A}$. The surface integral on the right-hand side of eq. (1.1) can be transformed into a volume integral using the divergence theorem to give

$$\int_{V_0} \frac{\partial \rho}{\partial t} \, \mathrm{d}V = - \int_{V_0} \boldsymbol{\nabla} \cdot (\rho \boldsymbol{u}) \, \mathrm{d}V \tag{1.2}$$

NOTE (OS): explain change on L.H.S.? and thus

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{1.3}$$

since the volume V_0 is stationary and arbitrary. Eq. (1.3) is the *continuity equation* in fluid dynamics. It is a partial differential equation (PDE) reflecting the conservation of mass. The vector

$$\boldsymbol{j} = \rho \boldsymbol{u} \tag{1.4}$$

is called the *mass flux density* or *momentum density*.

In the literature, the continuity equation is also often written in the forms

$$\frac{\partial \rho}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \rho + \rho \boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \quad \text{or} \quad \frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho \boldsymbol{\nabla} \cdot \boldsymbol{u} = 0. \tag{1.5}$$

Here we have introduced the *material derivative*

$$\frac{\mathrm{D}}{\mathrm{D}t} = \frac{\partial}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla}, \tag{1.6}$$

NOTE (OS): notation issue. There's no material derivative in previous equation which denotes the rate of change as the fluid element moves about in space, rather than the rate of change $\partial/\partial t$ at a fixed point in space.

Exercise 1.1. Fluid conservation equations can be given in two main forms: *conservation form*, as in eq. (1.3), or *material derivative form*, as in eq. (1.5). Using the continuity equation, eq. (1.3), show for an equation for a general quantity λ that the two forms can be related asNOTE (OS): simplify to "show that for a general quantity λ", i.e. omit "for an equation"?

$$\frac{\partial(\rho\lambda)}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\boldsymbol{u}\lambda) = \rho\frac{\mathrm{d}\lambda}{\mathrm{d}t}. \tag{1.7}$$

1.1.2 Navier-Stokes equation

Similar to our analysis above, we can consider the change of momentum of a fluid element with density ρ and velocity $\boldsymbol{u}$, occupying a small volume V_0. For a simple ideal fluid, the change of net momentum can be due to (i) flow of momentum into or out of the fluid element, (ii) pressure differences and (iii) external body forces. TODO (all): We have to define how to denote forces and force densities (force per volume). I think it is currently not consistenly handled.THINK (OS): Use f for density; $\boldsymbol{F}$ for force (vector), and F for force (scalar)? Each of these contributions is written respectively on the right-hand side of the following momentum balance equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{V_0}\rho\boldsymbol{u}\,\mathrm{d}V = -\oint_{\partial V_0}\rho\boldsymbol{u}\boldsymbol{u}\cdot\mathrm{d}\boldsymbol{A} - \oint_{\partial V_0} p\,\mathrm{d}\boldsymbol{A} + \int_{V_0}\boldsymbol{F}\,\mathrm{d}V. \tag{1.8}$$

NOTE (OS): We need to adhere to index notation: $\boldsymbol{uu}$ Transforming the surface integrals into volume integrals using the divergence theorem, the above equation can be rewritten as

$$\int_{V_0}\frac{\partial(\rho\boldsymbol{u})}{\partial t}\mathrm{d}V = -\int_{V_0}\boldsymbol{\nabla}\cdot(\rho\boldsymbol{u}\boldsymbol{u})\,\mathrm{d}V - \int_{V_0}\boldsymbol{\nabla}p\,\mathrm{d}V + \int_{V_0}\boldsymbol{F}\,\mathrm{d}V \tag{1.9}$$

and therefore

$$\frac{\partial(\rho\boldsymbol{u})}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\boldsymbol{u}\boldsymbol{u}) = -\boldsymbol{\nabla}p + \boldsymbol{F}. \tag{1.10}$$

This is the *Euler equation* for an ideal fluid.

More generallyTHINK (OS): This deserves more of an explanation. In what way are nonideal fluids different from ideal fluids? This is handled through additional stresses and generalized momentum flux..., this momentum equation can be written in the form

$$\frac{\partial(\rho\boldsymbol{u})}{\partial t} + \boldsymbol{\nabla}\cdot\boldsymbol{\Pi} = \boldsymbol{F}, \tag{1.11}$$

where we have used the *momentum flux density tensor*, defined as

$$\Pi_{\alpha\beta} = \rho u_\alpha u_\beta - \sigma_{\alpha\beta}. \tag{1.12}$$

This general momentum equation is called the *Cauchy momentum equation*. The term $\sigma_{\alpha\beta}$ is called the *stress tensor* and corresponds to the non-direct momentum transfer of the moving fluid. For simple fluids described by the Euler equation we find an isotropic stress $\sigma_{\alpha\beta} = -p\delta_{\alpha\beta}$; the stress tensor contains diagonal elements which are the same in all directions.

The momentum flux transfer in the Euler equation only includes momentum transfer which is reversible, either through the flow of mass or due to pressure forces which are conservative. For real fluids, we need to include a viscosity or internal friction term which causes dissipative, irreversible transfer of momentum from one fluid element to another, neighbouring element.

To establish the form of this *viscous stress tensor* $\boldsymbol{\sigma'}$, we first argue that such contribution must be zero when the flow is uniform, including rigid body translation and rotation. We may further argue that if the velocity gradients are small, then momentum transfer due to viscosity is well captured by terms which are proportional to the first derivatives of the velocity only. TODO (HK): Refer to Landau-Lifshitz here? At least the book should be mentioned in the introduction. The most general tensor of rank two satisfying these two arguments is

$$\sigma'_{\alpha\beta} = \eta\left(\frac{\partial u_\alpha}{\partial x_\beta} + \frac{\partial u_\beta}{\partial x_\alpha}\right) + \zeta\delta_{\alpha\beta}\frac{\partial u_\gamma}{\partial x_\gamma}, \tag{1.13}$$

where η and ζ are coefficients of viscosity. They are usually assumed to be isotropic and uniform, though this assumption will break down for more complex fluids. NOTE (TK): This is nicely explained, but not accurate. Eq. (1.13) is not the most general form as you say directly afterwards. So there is a small contradiction in what you write, but for our needs that is probably fine.

The viscous stress tensor is often separated into a traceless *shear stress* and a *normal stress*:

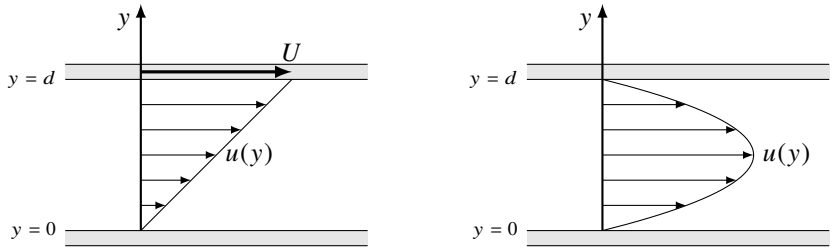
$$\sigma'_{\alpha\beta} = \eta\left(\frac{\partial u_\alpha}{\partial x_\beta} + \frac{\partial u_\beta}{\partial x_\alpha} - \frac{2}{3}\delta_{\alpha\beta}\frac{\partial u_\gamma}{\partial x_\gamma}\right) + \eta_\mathrm{B}\delta_{\alpha\beta}\frac{\partial u_\gamma}{\partial x_\gamma}. \tag{1.14}$$

The first viscosity coefficient η is usually called the *shear viscosity*, while the combination $\eta_\mathrm{B} = 2\eta/3 + \zeta$ is usually called the *bulk viscosity*.THINK (OS): Unusual notation for viscosities?

With the total stress tensor given as the sum of the pressure and viscosity contributions,

$$\sigma_{\alpha\beta} = \sigma'_{\alpha\beta} - p\delta_{\alpha\beta}, \tag{1.15}$$

we are now ready to write down the Navier-Stokes equation. Inserting eq. (1.13) and eq. (1.15) into eq. (1.11), we obtain



(a) Couette flow (b) Poiseuille flow

Fig. 1.1: Two fundamental steady-state flows between two plates.

$$\frac{\partial(\rho u_\alpha)}{\partial t} + \frac{\partial(\rho u_\alpha u_\beta)}{\partial x_\beta} = -\frac{\partial p}{\partial x_\alpha} + \frac{\partial}{\partial x_\beta}\left[\eta\left(\frac{\partial u_\alpha}{\partial x_\beta} + \frac{\partial u_\beta}{\partial x_\alpha}\right) + \left(\eta_\mathrm{B} - \frac{2\eta}{3}\right)\frac{\partial u_\gamma}{\partial x_\gamma}\delta_{\alpha\beta}\right] + F_\alpha. \tag{1.16}$$

Assuming the viscosities are nearly constant, this can be simplified to give

$$\rho\frac{\mathrm{D}u_\alpha}{\mathrm{D}t} = -\frac{\partial p}{\partial x_\alpha} + \eta\frac{\partial^2 u_\alpha}{\partial x_\beta \partial x_\beta} + \left(\eta_\mathrm{B} + \frac{\eta}{3}\right)\frac{\partial^2 u_\beta}{\partial x_\alpha \partial x_\beta} + F_\alpha. \tag{1.17}$$

The Navier-Stokes equation can be simplified considerably if the fluid may be regarded as incompressible, $\rho = \text{const}$, so that the continuity equation, eq. (1.3), reduces to $\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0$. In this case, we can write the Navier-Stokes equation in its most common form, the *incompressible Navier-Stokes equation*

$$\rho\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} = -\boldsymbol{\nabla}p + \eta\Delta\boldsymbol{u} + \boldsymbol{F}. \tag{1.18}$$

Here, $\Delta = \boldsymbol{\nabla}\cdot\boldsymbol{\nabla} = \partial^2/(\partial x_\beta \partial x_\beta)$ is the *Laplace operator*.

Exercise 1.2. Let us consider the steady shear flow known as *Couette flow*, where an incompressible fluid is sandwiched between two parallel plates as shown in fig. 1.1a. The separation between the two plates is d in the y-direction. The bottom plate is held fixed such that $\boldsymbol{u} = (0,0,0)^\top$, while the top plate moves with velocity $\boldsymbol{u} = (U,0,0)^\top$. There is no external force applied to the system. Starting from the Navier-Stokes equation (eq. (1.18)), show that the velocity profile of the fluid is given by

$$u_x(y) = \frac{U}{d}y. \tag{1.19}$$

You should assume that there is no slip between the fluid and the parallel plates. In other words, the velocity of the fluid near the wall is equal to that of the wall.

Exercise 1.3. Let us now consider a steady, incompressible fluid flow commonly known as *Poiseuille flow*. The fluid is enclosed between two parallel plates and is moving in the x-direction, as shown in fig. 1.1b. The separation between the two

plates is d along the y-axis. Poiseuille flow can in fact be driven by either (i) a constant pressure gradient or (ii) an external body force (such as gravity) in the x-direction. Assume no-slip boundary condition between the fluid and the plates.

1. For a pressure gradient driven flow, show that the fluid velocity profile is given by
$$u_x(y) = -\frac{1}{2\eta}\frac{\mathrm{d}p}{\mathrm{d}x}y(y-d). \tag{1.20}$$
2. Derive the corresponding velocity profile when the flow is driven by an external body force.

1.1.3 Equations of state

At this point we have four equations that describe the behaviour of a fluid: the continuity equation, eq. (1.3), describes the conservation of mass and the conservation of momentum is described by the three Euler equations or the Navier-Stokes equations (one for each spatial component) in eq. (1.10) and eq. (1.16), respectively.[2]

However, this system of equations is not closed. While we have five unknowns (density ρ, pressure p, and the three velocity components u_x, u_y, u_z), we have only four equations to describe their evolution. Consequently, the system of equations is *unsolvable*, unless we can fix variables, *e.g.* by assuming the density to be constant.

Another equation can be added to the system thanks to the *state principle* of equilibrium thermodynamicsTHINK (OS): reference would be nice, which relates the *state variables* that describe the local thermodynamic state of the fluid, such as the density ρ, the pressure p, the temperature T, the internal energy e, and the entropy s. We will defer a more detailed description of the temperature, the internal energy, and the entropy to section 1.3. The state principle declares that any of these state variables can be related to any other two state variables through an *equation of state* [2].

The most famous such equation of state is the *ideal gas law*,

$$p = \rho RT, \tag{1.21}$$

which relates the pressure to the density and the temperature through the *specific gas constant* R, with units $[R] = \mathrm{J/(kg\,K)}$.[3]

Another equation of state for ideal gases expresses the pressure as a function of the density and the entropy, [2]

[2] A fifth conservation equation for energy can also be derived, though we will only briefly address it later in section 1.3.4 since it is less important both in fluid mechanics and in the lattice Boltzmann method.

[3] The ideal gas law is expressed in many forms throughout science, often with quantities given in moles. Eq. (1.21) is expressed using the state variables employed in fluid mechanics, the cost being that the specific gas constant R *varies between different gases*. Here, $R = k_\mathrm{B}/m$, where k_B is Boltzmann's constant and m is the mass of the a gas molecule.

$$\frac{p}{p_0} = \left(\frac{\rho}{\rho_0}\right)^{\gamma} \mathrm{e}^{(s-s_0)/c_V}. \tag{1.22}$$

The constants p_0, ρ_0, and s_0 refer to values at some constant reference state. This equation makes use of the heat capacities at *constant volume* c_V and *constant pressure* c_p and their ratio γ, also known as the *adiabatic index*. These are defined generally as [2]

$$c_V = \left(\frac{\partial e}{\partial T}\right)_V, \quad c_p = \left(\frac{\partial (e + p/\rho)}{\partial T}\right)_p, \quad \gamma = \frac{c_p}{c_V}. \tag{1.23}$$

In an ideal gas, the two heat capacities are related as $c_p = c_V + R$.

The attentive reader may have realised that any equation of state must introduce a *third* variable into the system of equations: for instance, eq. (1.21) introduces the temperature T and eq. (1.22) introduces the entropy s. Consequently, introducing the equation of state does not itself directly close the system of equations. The system can only be fully closed if an equation that describes the evolution of the third state variable is also derived from the aforementioned energy equation. However, the resulting system of equations is very cumbersome.

Instead, introducing the equation of state gives us more options to close the system of equations through suitable approximations. For example, most of acoustics is based on the assumption that s is approximately constant [4, 2], which simplifies eq. (1.22) to the *isentropic equation of state*

$$p = p_0 \left(\frac{\rho}{\rho_0}\right)^{\gamma} \tag{1.24}$$

and closes the system of equations. Similarly, we can approximate the fluid as having a constant temperature $T \approx T_0$, which simplifies eq. (1.21) to the *isothermal equation of state*

$$p = \rho R T_0, \tag{1.25}$$

which has a linear relationship between the pressure p and the density ρ.

Exercise 1.4. The isothermal equation of state will be very central to the lattice Boltzmann method further on in the book. Show that it is merely a special case of the isentropic equation of state Eq. (1.24) with $\gamma = 1$.

Another approximation mentioned previously in section 1.1.2 is the incompressible fluid of constant density $\rho = \rho_0$. In this case, an equation of state is not used, as the incompressible continuity equation $\nabla \cdot \boldsymbol{u}$ and the incompressible Navier-Stokes equation in eq. (1.18) are by themselves sufficient. Together, these form a closed system of four equations for the four remaining variables p, u_x, u_y, u_z.

For small deviations from a reference state, nonlinear equations of state such as Eq. (1.22) may also be approximated by linearisation. For instance, using the total differential we may linearise any equation of state $p(\rho, s)$ as

$$p = p_0 + p' \approx p_0 + \left(\frac{\partial p}{\partial \rho}\right)_s \rho' + \left(\frac{\partial p}{\partial s}\right)_\rho s'. \tag{1.26}$$

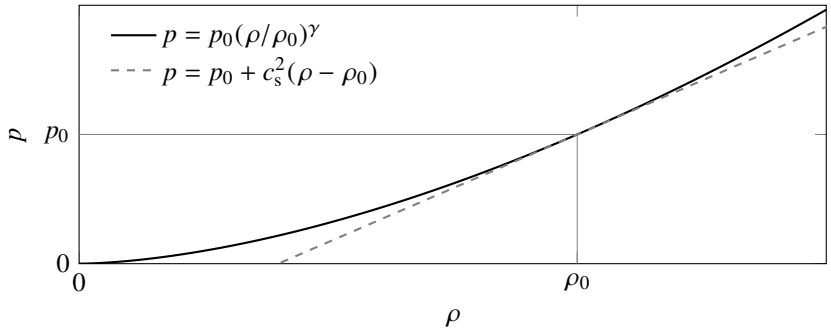


Fig. 1.2: Comparison of the isentropic equation of state in Eq. (1.24) and its linearised version in Eq. (1.27) for $\gamma = 5/3$.

Here, the derivatives are evaluated around $p = p_0$, and the primed variables represent deviations from the reference state defined by p_0, ρ_0, and s_0.

For the isentropic equation of state in Eq. (1.24), Eq. (1.26) is further simplified to

$$p \approx p_0 + c_s^2 \rho', \tag{1.27}$$

since the speed of sound c_s is in general given by the relation [4, 2]

$$c_s^2 = \left(\frac{\partial p}{\partial \rho}\right)_s. \tag{1.28}$$

In the case of the isentropic and isothermal equations of state in Eq. (1.24) and Eq. (1.25) we find that $c_s = \sqrt{\gamma R T_0}$ and $c_s = \sqrt{R T_0}$, respectively.

Note that the constant reference pressure p_0 is completely insignificant to the Navier-Stokes equation, as only the pressure gradient $\nabla p = \nabla(p_0 + p') = \nabla p'$ is present in the equation.[4] Therefore, the isothermal equation of state, which can be expressed as

$$p = c_s^2 \rho \quad \Rightarrow \quad p = c_s^2 \rho_0 + c_s^2 \rho' \tag{1.29}$$

can be used to model other equations of state in the linear regime if the entropy is near-constant: As long as the speed of sound is matched, it does not matter if the reference pressure p_0 is different.

1.2 Relevant scales

[4] While p_0 is relevant to the energy equation that we will see later in Section 1.3.4, this equation is usually not taken into account in lattice Boltzmann simulations.

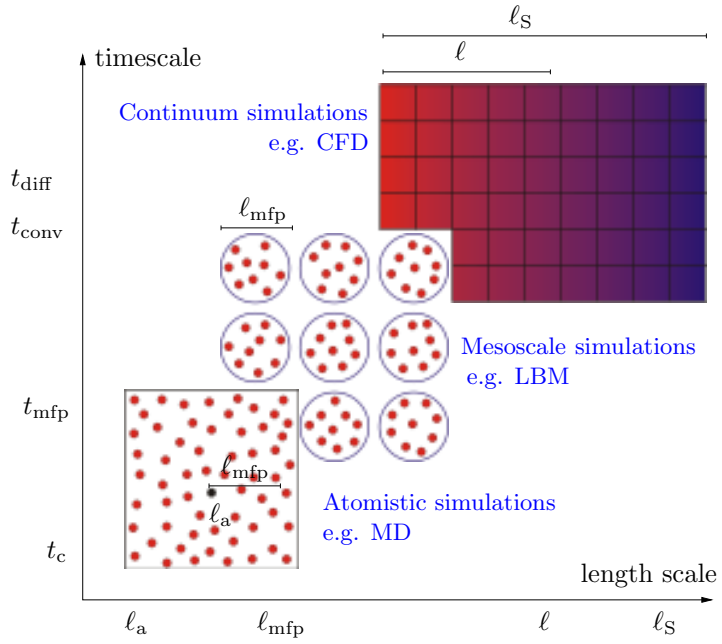


Fig. 1.3: The hierarchy of length and time scales in a typical fluid dynamics problem. Depending on the level of details required, different simulation techniques are suitable. TODO (HK): This is an excellent figure. We should adapt it at the end (grey-scale, italics 'e.g.' and so on.

NOTE (EMV): Please make sure to define and describe the concepts of the microscopic scale, the mesoscopic scale, and the macroscopic scale in this section. I think you can tie it nicely to the figure!

As discussed in the previous section, the mathematical descriptions of fluid dynamics rely on the continuum assumption, where we operate at length and time scales sufficiently large that the atomistic picture can be averaged out. To formalise this discussion, let us start by considering the hierarchy of length scales associated with a typical fluid flow problem. If we stay within the classical mechanics picture, from small to large, we have (i) the size of the fluid atom or molecule ℓ_a, (ii) the mean free path (distance travelled between two successive collisions) ℓ_{mfp}, (iii) the typical scale for gradients in some macroscopic properties ℓ and (iv) the system size ℓ_S. The typical ordering of these length scales is $\ell_a \ll \ell_{mfp} \ll \ell \leq \ell_S$, as illustrated in fig. 1.3.

Coupled to this hierarchy of length scales is the hierarchy of time scales. At very short time scale, we can define the collision time $t_c \sim \ell_a/v_T$, *i.e.* the duration of a collision event, where $v_T = (k_B T/m)^{1/2}$ is the average thermal velocity of the

molecules. Within Boltzmann's standard kinetic theory, we usually assume $t_c \to 0$, *i.e.* collisions happen instantaneously. Note that the thermal velocity v_T is different from the macroscopic fluid velocity, $u \ll v_T$.[5] Next we can define the mean flight time between two successive collitions, $t_{mfp} = \ell_{mfp}/v_T$. This is the time scale at which kinetic theory operates and where the system relaxes to local equilibrium through collision events. Local equilibrium, however, does not mean that the system is in global equilibrium. In fact, the opposite is often the case, and we are interested in studying these situations.

At longer time and larger length scales, there can exist hydrodynamic flow from one region of the fluid to another. Depending on whether we have convective (inertial regime) or diffusive (viscous regime) dynamics, the shortest (most relevant) time scale is either $t_{conv} \sim \ell/u$ or $t_{diff} \sim \ell^2/\nu$, where ν is the kinematic viscosity. The kinetic viscosity is related to the dynamic shear viscosity by $\eta = \rho\nu$. The ratio between these two hydrodynamic time scales is the well-known *Reynolds number*:

$$\mathrm{Re} = \frac{t_{diff}}{t_{conv}} = \frac{u\ell}{\nu}. \tag{1.30}$$

Both high and low Reynolds number flows are of interest. High Reynolds number flows, on the one hand, are usually dominated by turbulence and are relevant for vehicle aerodynamics, building designs, and many other applications. On the other hand, there is a surge of interest in low Reynolds number flows due to their importance in microfluidics and biophysics.

Another important macroscopic time scale is the acoustic time scale, $t_{sound} \sim \ell/c_s$, where c_s is the speed of sound in the fluid. This time scale determines how fast compression waves propagate in the fluid. When the acoustic time scale is fast in comparison to the convective time scale, the fluid behaves similarly to an incompressible fluid. Otherwise, the fluid compressibility is an important factor, which provides a number of additional physics such as shock waves. The *Mach number*

$$\mathrm{Ma} = \frac{t_{sound}}{t_{conv}} = \frac{u}{c_s} \tag{1.31}$$

defines the ratio between the acoustic and convective time scales. In practice, we can usually assume non-acoustic fluid flow with $\mathrm{Ma} \leq 0.1$ to be incompressible. NOTE (EMV): I've done a few corrections here; if there can still be sound waves in a fluid, then we cannot consider it incompressible, though the non-acoustic parts of the flow behave as if it were.

It must be noted that there are a number of situations where the above-mentioned ordering of length and time scales is not satisfied. Examples include flows of rarified gases and nanofluidics. For the former, the mean free path becomes large enough so that it is comparable to the macroscopic length scale: $\ell_{mfp} \sim \ell$. On the other hand, the miniaturisation of fluidic devices makes the system size in nanofluidics comparable to the mean free path: $\ell_S \sim \ell_{mfp}$. A particularly useful parameter is therefore the *Knudsen number*

[5] However, the thermal velocity v_T is on the order of the speed of sound c_s [2].

$$\mathrm{Kn} = \frac{\ell_{\mathrm{mfp}}}{\ell} \tag{1.32}$$

which defines the ratio between the mean free path and the representative physical length scale. For $\mathrm{Kn} \ll 1$, the hydrodynamic picture (Navier-Stokes) is valid, whereas for $\mathrm{Kn} \gtrsim 1$, one has to go back to the kinetic theory description. As we shall see later in section 1.3.4, the Knudsen number is in fact the (small) expansion parameter used in the Chapman-Enskog theory to derive the Navier-Stokes equation from the Boltzmann equation. The Knudsen number is also closely related to the Mach and Reynolds number. It was first shown by Von Kármán that

$$\mathrm{Kn} = \alpha \frac{\mathrm{Ma}}{\mathrm{Re}} \tag{1.33}$$

with α being a numerical constant. This relation is thus known as the *von Kármán relation*.

Dimensionless numbers such as the Reynolds, Knudsen and Mach numbers proliferate in the fluid mechanics literature. These numbers are in fact very useful. Primarily, it must be appreciated that fluid flows which share the same dimensionless numbers provide the same physics upon a simple scaling by the typical length and velocity scales in the problem. This important statement is called the *law of similarity*. To illustrate this, let us rewrite the Navier-Stokes equation in its dimensionless form. We renormalise any length scale in the system by ℓ and velocity by the mean fluid velocity V, such that

$$\boldsymbol{u}^\star = \frac{\boldsymbol{u}}{V}, \quad p^\star = \frac{p}{\rho V^2}, \quad \boldsymbol{F}^\star = \frac{\boldsymbol{F}\ell}{\rho V^2}, \quad \frac{\partial}{\partial t^\star} = \frac{\ell}{V}\frac{\partial}{\partial t}, \quad \boldsymbol{\nabla}^\star = \ell\boldsymbol{\nabla} \tag{1.34}$$

and hence

$$\frac{\mathrm{d}\boldsymbol{u}^\star}{\mathrm{d}t^\star} = -\boldsymbol{\nabla}^\star p^\star + \frac{1}{\mathrm{Re}}\Delta^\star \boldsymbol{u}^\star + \boldsymbol{F}^\star. \tag{1.35}$$

The Reynolds number therefore measures the relative importance of inertial to viscous terms in the Navier-Stokes equation.

TODO (TK): Please revise the following sentence. I do not understand what it is linked to. It should be noted that our scaling arguments here do not guarantee that all the dimensionless numbers will then be the same. However, as long as the ordering of length and time scales is unchanged, their exact values may not matter much. The key here is usually the separation of scales. For example, if the Knudsen or Mach number is sufficiently small, their actual values are irrelevant for the hydrodynamic flows of interests, where the Reynolds number is the key parameter. Therefore, usually it can be argued that all flows with the same Reynolds number are comparable to one another. In chapter 7 we will come back to the non-dimensionalisation and how to take advantage of the law of similarity to convert parameters from the physical world to a simulation and back.

THINK (OS): Might have missed it, but I think it is worthwhile to mention near here the link between Mach number and incompressible flow.

1.3 Kinetic theory

As mentioned in Section 1.2, kinetic theory is a fluid description that lies between the *microscopic* scale where we track the motion of individual molecules and the *macroscopic* scale where we describe the fluid using more tangible quantities such as density, fluid velocity, and temperature. In the *mesoscopic* kinetic theory, we describe the *distribution* of particles in a gas, a quantity which evolves on timescales around the mean collision time t_{mfp}.

While kinetic theory can in principle be used to describe any fluid, it is most commonly applied to the simplest case of a dilute gas. There we can assume that the constituent molecules spend very little of their time actually *colliding* (*i.e.* $t_{\mathrm{c}} \ll t_{\mathrm{mfp}}$ using the terminology of the previous section). This is the same as assuming that the molecules almost always collide one-on-one, with three particles almost never simultaneously being involved in a collision. This assumption does not hold as well for dense gases where molecules are closer together and therefore spend more of their time colliding, and it does not hold at all for liquids where molecules are held close to each other by intermolecular attracting forces and thus constantly interact. While it is possible to formulate a kinetic theory of liquids, this is much harder than for dilute gases [5].

For simplicity, we will constrain our discussion in this section to the kinetic theory of dilute *monatomic* gases. Single atoms collide elastically, so that all translational energy is conserved in a collision. On the other hand, molecules consisting of several atoms have inner degrees of freedom; they may contain rotational and vibrational energy. Therefore, while total energy is always conserved in collisions, a collision between two such molecules may also be *inelastic* (*i.e.* translational energy becomes rotational or vibrational energy) or *superelastic* (*i.e.* rotational or vibrational energy becomes translational energy). In addition, molecular rotation and vibration must be treated quantum mechanically. The kinetic theory of polyatomic gases can be found in the literature [6, 7, 8, 9]. However, the macroscopic behaviour of polyatomic and monatomic gases is largely similar.

Apart from the quantisation of rotational and vibrational energy in polyatomic gases, the kinetic theory of gases can be considered to be completely classical physics, as it is a statistical description of a large number of particles; as per Bohr's correspondence principle, the quantum behaviour of a system reduces to classical behaviour when the system becomes large enough.

1.3.1 The distribution function and its moments

The fundamental variable in kinetic theory is the *distribution function* $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$. It can be seen as a generalization of density ρ which also takes the microscopic *particle velocity* $\boldsymbol{\xi}$ into account. While $\rho(\boldsymbol{x}, t)$ represents the density of mass in *physical* space, $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ simultaneously represents the density of mass in both three-dimensional *physical* space *and* in three-dimensional *velocity* space. Therefore, f

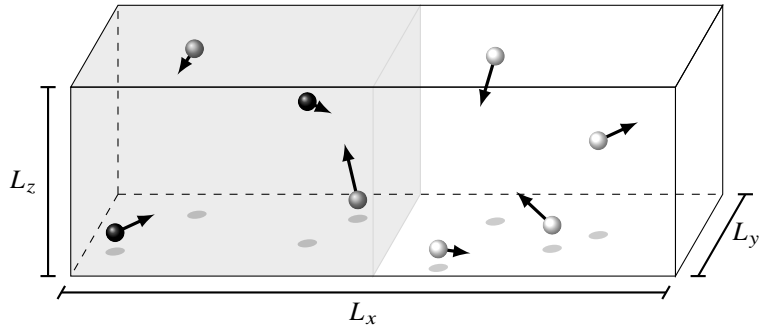


Fig. 1.4: Particles in a box. *Right-moving* particles in the *left half* of the box are marked as black. The total mass of such particles can be found from f as in Eq. 1.36.

has the units

$$[f] = \mathrm{kg} \times \frac{1}{\mathrm{m}^3} \times \frac{1}{(\mathrm{m/s})^3} = \frac{\mathrm{kg\,s}^3}{\mathrm{m}^6}.$$

In other words, the distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ represents the density of particles with velocity $\boldsymbol{\xi} = (\xi_x, \xi_y, \xi_z)$ at position $\boldsymbol{x}$ and time t.

To demonstrate how the distribution function extends the concept of density, let us consider a gas in a box of size $V = L_x \times L_y \times L_z$, as shown in Figure 1.4. The total mass of the gas inside the box is of course given by the integral of the density over the box, $\int_V \rho \,\mathrm{d}^3x$. We can also calculate more specific things using the density: For instance, the mass in the *left half* of the box is $\int_{x=0}^{x=L_x/2} \rho \,\mathrm{d}^3x$. The distribution function f would let us find even more specific things: For example, the mass of *right-moving* particles, *i.e.* particles with $\xi_x > 0$, in the *left half* of the box is

$$\int_{x=0}^{x=L_x/2} \int_{\xi_x>0} f \,\mathrm{d}^3\xi \,\mathrm{d}^3x. \tag{1.36}$$

The distribution function f is also connected to macroscopic variables like the density ρ and the fluid velocity u from its *moments*. These moments are integrals of f, weighted with some function of $\boldsymbol{\xi}$, over the entire velocity space. For instance, the macroscopic *mass density* can be found as the moment

$$\rho(\boldsymbol{x}, t) = \int f(\boldsymbol{x}, \boldsymbol{\xi}, t) \,\mathrm{d}^3\xi. \tag{1.37a}$$

By integrating over velocity space in this way, we are considering the contribution to the density of particles of *all possible velocities* at position $\boldsymbol{x}$ and time t.

We can also consider the particles' contribution $\boldsymbol{\xi} f$ to the momentum density. Again considering all possible velocities, we find the macroscopic *momentum density* as the moment

$$\rho(\boldsymbol{x},t)\boldsymbol{u}(\boldsymbol{x},t) = \int \boldsymbol{\xi} f(\boldsymbol{x},\boldsymbol{\xi},t)\,\mathrm{d}^3\xi. \tag{1.37b}$$

Similarly, we can find the macroscopic *total energy density* as the moment

$$\rho(\boldsymbol{x},t)E(\boldsymbol{x},t) = \frac{1}{2}\int |\boldsymbol{\xi}|^2 f(\boldsymbol{x},\boldsymbol{\xi},t)\,\mathrm{d}^3\xi. \tag{1.37c}$$

This contains two types of energy; the energy $\frac{1}{2}\rho|\boldsymbol{u}|^2$ due to the bulk motion of the fluid, and the internal energy due to the random thermal motion of the gas particles. It is also possible to find only the latter type, the macroscopic *internal energy density*, as a moment,

$$\rho(\boldsymbol{x},t)e(\boldsymbol{x},t) = \frac{1}{2}\int |\boldsymbol{v}|^2 f(\boldsymbol{x},\boldsymbol{\xi},t)\,\mathrm{d}^3\xi. \tag{1.37d}$$

Here we have introduced the *relative velocity* $\boldsymbol{v}$, which is the deviation of the particle velocity from the mean velocity:

$$\boldsymbol{v} = \boldsymbol{\xi} - \boldsymbol{u}. \tag{1.38}$$

These expressions only consider the *translational* energy of the molecules, *i.e.* the energy due to the movement with their velocity $\boldsymbol{\xi}$. In the more difficult kinetic theory of polyatomic gases, the internal energy must include additional degrees of freedom, such as molecular vibrational and rotational energies.

Exercise 1.5. Consider a somewhat unrealistic spatially homogeneous gas where all particles are moving with the same velocity $\boldsymbol{u}$, so that the distribution function is $f(\boldsymbol{x},\boldsymbol{\xi},t) = \rho\,\delta(\boldsymbol{\xi}-\boldsymbol{u})$. Verify from its moments that its density is ρ and its momentum density is $\rho\boldsymbol{u}$. Additionally, find its moments of total energy density and internal energy density.

Exercise 1.6. Show that the relative velocity moment of f is

$$\int \boldsymbol{v} f(\boldsymbol{x},\boldsymbol{\xi},t)\,\mathrm{d}^3\xi = 0. \tag{1.39a}$$

Using this and the identity $|\boldsymbol{v}|^2 = |\boldsymbol{\xi}|^2 - 2(\boldsymbol{\xi}\cdot\boldsymbol{u}) + |\boldsymbol{u}|^2$, show that the total and internal energy densities are related as

$$\rho e = \rho E - \frac{1}{2}\rho|\boldsymbol{u}|^2. \tag{1.39b}$$

The pressure can also be found as a moment of the distribution function. There are several ways to do this. The most direct route, presented *e.g.* in [10, 11], is to consider that particles impart momentum when bouncing off a surface. At higher particle velocities, more momentum is imparted, and more particles can bounce off in a given time. A closer analysis results in an expression for pressure as a moment of f.

A shortcut to the same expression for pressure can be made by using the equipartition theorem of classical statistical mechanics. This gives a specific internal energy of $RT/2$ for each *degree of freedom* [2]. These degrees of freedom are typically molecular translation, vibration, and rotation. For monatomic gases, there is no inner molecular structure and there can be no vibration or rotation, leaving only the translational movement in the three spatial dimensions.

Thus, with three degrees of freedom, we can use the ideal gas law Eq. (1.21) to find for an ideal monatomic gas that

$$\rho e = \frac{3}{2}\rho RT = \frac{3}{2}p. \tag{1.40}$$

Consequently, both the pressure and the temperature can be found proportional through the same moment as internal energy,

$$p = \rho RT = \frac{2}{3}\rho e = \frac{1}{3}\int |\boldsymbol{v}|^2 f(\boldsymbol{x}, \boldsymbol{\xi}, t)\, \mathrm{d}^3\xi. \tag{1.41}$$

Exercise 1.7. Show from Eq. (1.40) and the heat capacity definitions in Eq. (1.23) that the specific heat capacities and the heat capacity ratio of an ideal monatomic gas are

$$c_V = \frac{3}{2}R, \quad c_p = \frac{5}{2}R \quad \Rightarrow \quad \gamma = \frac{5}{3}. \tag{1.42}$$

1.3.2 The equilibrium distribution function

The outgoing directions of two elastically colliding hard spheres is highly sensitive to small variations in their initial relative positions, as illustrated in Fig. 1.5. This is not only true for hard spheres like pool balls that only really interact when they touch, but also for molecules that interact at a distance, *e.g.* via electromagnetic forces. Therefore, collisions tend to even out the angular distribution of particle velocities in a gas around the mean velocity $\boldsymbol{u}$.

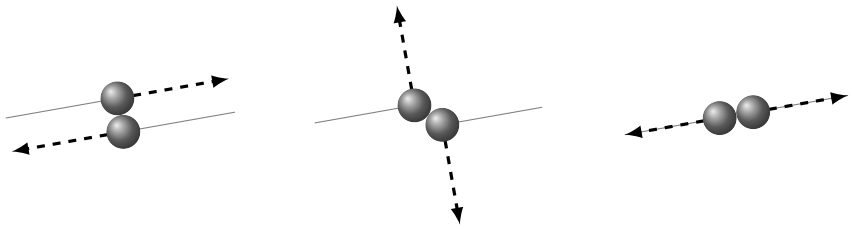
Consequently, when a gas has been left alone for sufficiently long, we may assume that the distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ will reach an *equilibrium distribution* $f^{\mathrm{eq}}(\boldsymbol{x}, \boldsymbol{\xi}, t)$ which is isotropic in velocity space around $\boldsymbol{\xi} = \boldsymbol{u}$: In a reference frame moving with speed $\boldsymbol{u}$, the equilibrium distribution can be expressed as $f^{\mathrm{eq}}(\boldsymbol{x}, |\boldsymbol{v}|, t)$.

Let us perform one additional assumption: We limit our search for an equilibrium distribution to solutions on the separable form

$$f^{\mathrm{eq}}\left(|\boldsymbol{v}|^2\right) = f^{\mathrm{eq}}\left(v_x^2 + v_y^2 + v_z^2\right) = f_{1\mathrm{D}}^{(0)}\left(v_x^2\right) f_{1\mathrm{D}}^{(0)}\left(v_y^2\right) f_{1\mathrm{D}}^{(0)}\left(v_z^2\right).$$

In other words, we assume that the three-dimensional equilibrium distribution is the product of three one-dimensional equilibrium distributions.

If we hold the magnitude of the velocity constant, *i.e.* with $|\boldsymbol{v}|^2 = v_x^2 + v_y^2 + v_z^2 = \text{const}$, we have that



(a) Grazing collision (b) Angled collision (c) Head-on collision

Fig. 1.5: Some collisions between hard spheres, with incoming paths shown in gray and outgoing paths in black

$$f^{\mathrm{eq}}\left(|\boldsymbol{v}|^2\right) = \text{const} \quad \Rightarrow \quad \ln f^{\mathrm{eq}}\left(v_x^2\right) + \ln f^{\mathrm{eq}}\left(v_y^2\right) + \ln f^{\mathrm{eq}}\left(v_z^2\right) = \text{const}.$$

This is fulfilled for 1D equilibria on the form $\ln f_{1\mathrm{D}}^{\mathrm{eq}}\left(v_x^2\right) = a + bv_x^2$, with a and b being generic constants. Consequently,

$$\ln f_{1\mathrm{D}}^{\mathrm{eq}}(v_x) + \ln f_{1\mathrm{D}}^{\mathrm{eq}}(v_y) + \ln f_{1\mathrm{D}}^{\mathrm{eq}}(v_z) = 3a + b\left(v_x^2 + v_y^2 + v_z^2\right) = \text{const},$$

and the full three-dimensional equilibrium distribution is of the form

$$f^{\mathrm{eq}}(|\boldsymbol{v}|) = \mathrm{e}^{3a}\mathrm{e}^{b|\boldsymbol{v}^2|}. \tag{1.43}$$

Since monatomic collisions conserve mass, momentum, and energy, the constants a and b can be found explicitly by demanding that f^{eq} has the same moments of density and energy as f. Additionally using Eq. (1.40), the equilibrium distribution can be expressed as any of the forms

$$\begin{aligned} f^{\mathrm{eq}}(\boldsymbol{x}, |\boldsymbol{v}|, t) &= \rho\left(\frac{3}{4\pi e}\right)^{3/2} \mathrm{e}^{-3|\boldsymbol{v}|^2/4e} = \rho\left(\frac{\rho}{2\pi p}\right)^{3/2} \mathrm{e}^{-p|\boldsymbol{v}|^2/2\rho} \\ &= \rho\left(\frac{1}{2\pi RT}\right)^{3/2} \mathrm{e}^{-|\boldsymbol{v}|^2/2RT}. \end{aligned} \tag{1.44}$$

This brief derivation has followed the same lines as Maxwell's original derivation. The equilibrium distribution fulfills all the assumptions we have placed upon it, but we have not proven that it is *unique*. However, the same distribution can be found uniquely using more substantiated statistical mechanics [10], as done later by Boltzmann. In honor of these two, this equilibrium distribution is often called the *Maxwell-Boltzmann distribution*.

Exercise 1.8. Show that the moments in Eq. (1.37), applied to the equilibrium distribution Eq. (1.44), result in a density ρ, a fluid velocity $\boldsymbol{u} = 0$, and internal energy e.

Hint: Consider the symmetries of the integrands. If an integrand is spherically symmetric around $\boldsymbol{v} = 0$*, the substitution* $\mathrm{d}^3\xi = 4\pi|\boldsymbol{v}|^2\,\mathrm{d}|\boldsymbol{v}|$ *can be performed.*

1.3.3 The Boltzmann equation and the collision operator

Now we know what the distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ represents and what we can obtain from it. But how does it evolve? We will now find the equation that describes its evolution in time.

Since f is a function of position $\boldsymbol{x}$, particle velocity $\boldsymbol{\xi}$, and time t, its total derivative with respect to time t must be

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \left(\frac{\partial f}{\partial t}\right)\frac{\mathrm{d}t}{\mathrm{d}t} + \left(\frac{\partial f}{\partial x_\beta}\right)\frac{\mathrm{d}x_\beta}{\mathrm{d}t} + \left(\frac{\partial f}{\partial \xi_\beta}\right)\frac{\mathrm{d}\xi_\beta}{\mathrm{d}t}.$$

Looking at the each term on the right-hand side in order, we have $\mathrm{d}t/\mathrm{d}t = 1$, we have the particle velocity $\mathrm{d}x_\beta/\mathrm{d}t = \xi_\beta$, and from Newton's second law we have the specific body force $\mathrm{d}\xi_\beta/\mathrm{d}t = F_\beta/\rho$, which has the units $[\boldsymbol{F}/\rho] = \mathrm{N/kg}$. Using the common notation $\Omega(f) = \mathrm{d}f/\mathrm{d}t$ for the total differential, we get the *Boltzmann equation*

$$\frac{\partial f}{\partial t} + \xi_\beta\frac{\partial f}{\partial x_\beta} + \frac{F_\beta}{\rho}\frac{\partial f}{\partial \xi_\beta} = \Omega(f). \tag{1.45}$$

This can be seen as a kind of advection equation: The two first terms represent the distribution function being advected with the velocity $\boldsymbol{\xi}$ of its particles. The third term represents forces affecting this velocity. On the right hand side, we have a source term.

What then does this source term $\Omega(f)$ represent? Without it and without forces, the Boltzmann equation is a simple advection equation, $\partial f/\partial t + \xi_\beta \partial f/\partial x_\beta = 0$, which represents the distribution function being propagated unchanged through space at a velocity $\boldsymbol{\xi}$. This would not happen, as particle collisions causes a local redistribution of the distribution function f. This redistribution is what the *collision operator* $\Omega(f)$ represents.

We know that collisions conserve the quantities of mass, momentum, and in our monatomic case[6], translational energy. These conservation constraints can be represented as moments of the collision operator, similarly to those in Eq. (1.37):

[6] Collisions in a monatomic gas are always elastic, as there is no inner degrees of freedom such as vibration or rotation for the energy to be transferred to. For polyatomic gases, collisions may be inelastic or superelastic, though the *total* energy is always conserved [8].

$$\text{mass conservation:} \qquad \int \Omega(f)\,\mathrm{d}^3\xi = 0, \tag{1.46a}$$

$$\text{momentum conservation:} \qquad \int \boldsymbol{\xi}\,\Omega(f)\,\mathrm{d}^3\xi = 0, \tag{1.46b}$$

$$\text{total energy conservation:} \qquad \int |\boldsymbol{\xi}|^2\,\Omega(f)\,\mathrm{d}^3\xi = 0, \tag{1.46c}$$

$$\text{internal energy conservation:} \qquad \int |\boldsymbol{v}|^2\,\Omega(f)\,\mathrm{d}^3\xi = 0. \tag{1.46d}$$

Exercise 1.9. Show using Eq. (1.38) that the total and internal energy conervation constraints are equivalent.

Boltzmann's original collision operator was of the form of a complicated and cumbersome double integral over velocity space. It considered all the possible outcomes of two-particle collisions for any choice of intermolecular forces. However, the collision operators used in the LBM are generally based on the same principle as the much simpler *BGK collision operator* [12],

$$\Omega(f) = -\frac{1}{\tau}\left(f - f^{\mathrm{eq}}\right). \tag{1.47}$$

This operator, named after its inventors Bhatnagar, Gross, and Krook, directly captures the relaxation of the distribution function towards the equilibrium distribution. The time constant τ, which determines the speed of this equilibration, is known as the *relaxation time*. The value of τ directly determines the transport coefficients such as viscosity and heat conduction, as will be shown later in Section 4.1.

Any useful collision operator must both respect the conserved quantities as expressed in Eq. (1.46) and ensure that the distribution function f locally evolves towards its equilibrium f^{eq}. The BGK operator is the simplest possible collision operator given these constraints. However, it is not as exact as Boltzmann's original operator: The BGK operator predicts a *Prandtl number*, which indicates the ratio of viscosity and thermal conduction, of $\mathrm{Pr} = 1$. Boltzmann's original operator correctly predicts $\mathrm{Pr} \simeq 2/3$, a value also found in lab experiments on monatomic gases [9].

Exercise 1.10. For a forceless, spatially homogeneous case $f(\boldsymbol{x}, \boldsymbol{\xi}, t) \to f(\boldsymbol{\xi}, t)$, show that the BGK operator will relax an initial distribution function $f(\boldsymbol{\xi}, t = 0)$ exponentially to the equilibrium distribution $f^{\mathrm{eq}}(\boldsymbol{\xi})$ as

$$f(\boldsymbol{\xi}, t) = f^{\mathrm{eq}}(\boldsymbol{\xi}) + \left(f(\boldsymbol{\xi}, 0) - f^{\mathrm{eq}}(\boldsymbol{\xi})\right) \mathrm{e}^{-t/\tau}.$$

1.3.4 Macroscopic conservation equations

The macroscopic equations of fluid mechanics can actually be found directly from the Boltzmann equation, Eq. (1.3.3). This is done by taking the moments of the

equation, *i.e.* by multiplying it with functions of $\boldsymbol{\xi}$ and integrating over velocity space.

For convenience, we introduce a general notation for the moments of f,

$$\begin{aligned} \Pi_0 &= \int f \,\mathrm{d}^3\xi = \rho, & \Pi_\alpha &= \int \xi_\alpha f \,\mathrm{d}^3\xi = \rho u_\alpha, \\ \Pi_{\alpha\beta} &= \int \xi_\alpha \xi_\beta f \,\mathrm{d}^3\xi, & \Pi_{\alpha\beta\gamma} &= \int \xi_\alpha \xi_\beta \xi_\gamma f \,\mathrm{d}^3\xi. \end{aligned} \tag{1.48}$$

The first two moments are already known as the moments of mass and momentum density. The second order moment $\Pi_{\alpha\beta}$ will soon be shown to be the momentum flux tensor from Eq. (1.12). As we can see from their definitions, these moments are not altered if their indices are reordered.

To deal with the force terms, we need to know the moments of the force term, which can be found directly using multidimensional integration by parts as

$$\begin{aligned} \int \frac{\partial f}{\partial \xi_\beta} \,\mathrm{d}^3\xi &= 0, \\ \int \xi_\alpha \frac{\partial f}{\partial \xi_\beta} \,\mathrm{d}^3\xi &= -\int \frac{\partial \xi_\alpha}{\partial \xi_\beta} f \,\mathrm{d}^3\xi = -\rho\delta_{\alpha\beta}, \\ \int \xi_\alpha \xi_\alpha \frac{\partial f}{\partial \xi_\beta} \,\mathrm{d}^3\xi &= -\int \frac{\partial (\xi_\alpha \xi_\alpha)}{\partial \xi_\beta} f \,\mathrm{d}^3\xi = -2\rho u_\beta. \end{aligned} \tag{1.49}$$

Mass conservation equation

The simplest equation we can find from the Boltzmann equation is the *continuity equation*, which describes the conservation of mass. Directly integrating the Boltzmann equation over velocity space, we find

$$\frac{\partial}{\partial t}\int f \,\mathrm{d}^3\xi + \frac{\partial}{\partial x_\beta}\int \xi_\beta f \,\mathrm{d}^3\xi + \frac{F_\beta}{\rho}\int \frac{\partial f}{\partial \xi_\beta}\,\mathrm{d}^3\xi = \int \Omega(f)\,\mathrm{d}^3\xi.$$

The integrals in each term can be resolved according to the moments in Eqs. eq. (1.37) and eq. (1.49) in addition to the collision operator's mass conservation property in Eq. (1.46a). Thus we find the *continuity equation* from Eq. (1.3) which describes mass conservation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_\beta)}{\partial x_\beta} = 0. \tag{1.50}$$

Note that this equation only depends on the conserved moments ρ and ρu_α. It does not depend on the particular form of f, unlike the following conservation equations.

Momentum conservation equation

If we similarly take the first moment of the Boltzmann equation, *i.e.* we multiply by ξ_α before integrating over velocity space, we find

$$\frac{\partial(\rho u_\alpha)}{\partial t} + \frac{\partial \Pi_{\alpha\beta}}{\partial x_\beta} = F_\alpha.$$

The moment $\Pi_{\alpha\beta}$ defined in Eq. (1.48) is the *momentum flux tensor*, Eq. (1.12)..

Exercise 1.11. By splitting the particle velocity as $\boldsymbol{\xi} = \boldsymbol{u} + \boldsymbol{v}$, show that the momentum flux tensor can be decomposed as

$$\Pi_{\alpha\beta} = \rho u_\alpha u_\beta + \int v_\alpha v_\beta f \, \mathrm{d}^3\xi \tag{1.51}$$

Thus, the second moment of the Boltzmann equation becomes the *Cauchy momentum equation* previously seen in Eq. (1.11),

$$\frac{\partial(\rho u_\alpha)}{\partial t} + \frac{\partial(\rho u_\alpha u_\beta)}{\partial x_\beta} = \frac{\partial \sigma_{\alpha\beta}}{\partial x_\beta} + F_\alpha. \tag{1.52}$$

However, this equation is not closed as we do not know the *stress tensor*

$$\sigma_{\alpha\beta} = -\int v_\alpha v_\beta f \, \mathrm{d}^3\xi \tag{1.53}$$

explicitly. To approximate this stress tensor, we must somehow find an explicit approximation of the distribution function f. We soon discuss how this may be done.

Energy conservation equation

Finally, the energy equation can be found from the trace of the second moment. In other words, we multiply by $\xi_\alpha \xi_\alpha$ before integrating over velocity space, resulting in

$$\frac{\partial(\rho E)}{\partial t} + \frac{1}{2}\frac{\partial \Pi_{\alpha\alpha\beta}}{\partial x_\beta} = F_\beta u_\beta.$$

This can be simplified in two steps. First we can split the moment in the same way as for the momentum equation, giving the *total energy equation*

$$\frac{\partial(\rho E)}{\partial t} + \frac{\partial(\rho u_\beta E)}{\partial x_\beta} = \frac{\partial(u_\alpha \sigma_{\alpha\beta})}{\partial x_\beta} + F_\beta u_\beta - \frac{\partial q_\beta}{\partial x_\beta} \tag{1.54}$$

with the *heat flux* q given as the moment

$$q_\beta = \frac{1}{2}\int v_\alpha v_\alpha v_\beta f \, \mathrm{d}^3\xi. \tag{1.55}$$

Secondly, we can eliminate the bulk motion energy component $\frac{1}{2}\rho|\boldsymbol{u}|^2$ from the equation by subtracting Eq. (1.52) multiplied with u_α. The end result is the *internal energy equation*

$$\frac{\partial(\rho e)}{\partial t} + \frac{\partial(\rho u_\beta e)}{\partial x_\beta} = \sigma_{\alpha\beta}\frac{\partial u_\alpha}{\partial x_\beta} - \frac{\partial q_\beta}{\partial x_\beta}. \tag{1.56}$$

Discussion

Finding the macroscopic conservation equations from basic kinetic theory has shown us that the mass equation is exact and invariable, while the momentum and energy equations depend on the stress tensor and the heat flux vector, which themselves depend on the form of f.

At this point we do not know much about f except its value at equilibrium. It can be shown that approximating Eq. (1.53) and Eq. (1.55) by assuming $f \simeq f^{\mathrm{eq}}$ results in the Euler momentum equation from Eq. (1.10) and a simplified energy equation sometimes known as the *Euler energy equation*.

Exercise 1.12. Assume that $f \simeq f^{\mathrm{eq}}$ for an ideal gas. From Eqs. eq. (1.53) and eq. (1.55), show that the general momentum and internal energy conservation equations result in the Euler momentum and energy equations

$$\frac{\partial(\rho u_\alpha)}{\partial t} + \frac{\partial(\rho u_\alpha u_\beta)}{\partial x_\beta} = -\frac{\partial p}{\partial x_\alpha} + F_\alpha, \quad \frac{\partial(\rho e)}{\partial t} + \frac{\partial(\rho u_\beta e)}{\partial x_\beta} = -p\frac{\partial u_\beta}{\partial x_\beta}. \tag{1.57}$$

Both these Euler equations lack the viscous stress tensor $\boldsymbol{\sigma}'$ and the heat flux found in the Navier-Stokes-Fourier momentum and energy equations. The viscous stress has previously been found in Eq. (1.14), while the heat flux is [2]

$$\boldsymbol{q} = -\kappa\nabla T, \tag{1.58}$$

κ being the fluid's thermal conductivity.

The fact that the Euler equations are found for a particle distribution f at *equilibrium* indicates that the phenomena of viscosity and heat conduction are connected to *nonequilibrium*, *i.e.* the deviation $f - f^{\mathrm{eq}}$. How, then, can we find a more general form of f which takes this deviation into account?

The *Chapman-Enskog analysis* is an established method of connecting the kinetic and continuum pictures by finding nonequilibrium contributions to f. Its main idea is expressing f as a perturbation expansion about f^{eq}:

$$f = f^{\mathrm{eq}} + \epsilon f^{(1)} + \epsilon^2 f^{(2)} + \dots .$$

The *smallness parameter* ϵ corresponds to each term's level in the *Knudsen number* $\mathrm{Kn} = \ell_{\mathrm{mfp}}/\ell$, as defined in Eq. (1.32). For $\mathrm{Kn} \to 0$, when the fluid is dominated by collisions, the particle distribution is approximately at equilibrium, and the fluid's behaviour is described by the Euler equation.

The perturbation of f combined with a nondimensionalisation analysis lets us explicitly find the first-order perturbation $f^{(1)}$ from the macroscopic derivatives of the equilibrium distribution f^{eq} [11]. Using this perturbation to approximate the stress tensor and heat flux moments in Eq. (1.53) and Eq. (1.55), we find the same stress tensor and heat flux as in Eq. (1.14) and Eq. (1.58), respectively. The resulting transport coefficients are all proportional with τ,

$$\eta = p\tau, \quad \eta_{\text{B}} = 0, \quad \kappa = \frac{5}{2} Rp\tau. \tag{1.59}$$

We will come back to the topic of the Chapman-Enskog analysis in section 4.1.

What have we seen here? On a macroscopic scale, the Boltzmann equation describes the macroscopic behaviour of a fluid. To a zeroth-order approximation $f \approx f^{\text{eq}}$, it describes the macroscopic equations of the *Euler model*. To a first-order approximation $f \approx f^{\text{eq}} + \epsilon f^{(1)}$, it describes the *Navier-Stokes-Fourier model* with its viscous stress and heat conduction.

It is also possible to go further; the second-order approximation $f \approx f^{\text{eq}} + \epsilon f^{(1)} + \epsilon^2 f^{(2)}$ gives the so-called *Burnett model*, which in principle gives even more detailed and accurate equations for the motion of a fluid. In practice, however, the Burnett and Navier-Stokes-Fourier models are only distinguishable at high Knudsen numbers [7], where the Burnett model predicts *e.g.* ultrasonic sound propagation with a better agreement with experiments [13]. However, even higher-order approximations paradoxically give a *poorer* prediction of ultrasonic sound propagation [13]. A proposed reason for this strange result is that the Chapman-Enskog expansion is actually *asymptotic* [7], meaning that f *diverges* as more terms are added in its expansion. This also casts some doubt on the Burnett model. In practice, we do not need to consider expansions beyond the first-order approximation $f \approx f^{\text{eq}} + \epsilon f^{(1)}$ which gives the Navier-Stokes-Fourier model.

1.3.5 Boltzmann's $\mathcal{H}$-theorem

The thermodynamic property of *entropy* can also be related to the distribution function f. The *entropy density* is denoted by ρs, with units $[\rho s] = \text{J/kg m}^3$.

Boltzmann himself showed that the quantity

$$\mathcal{H} = \int f \ln f \, \mathrm{d}^3\xi \tag{1.60}$$

can only ever decrease, and that it reaches its minimum value when the distribution function f reaches equilibrium.

This can be seen directly from the Boltzmann equation in eq. (1.3.3). By multiplying it with $(1+\ln f)$, using the chain rule in reverse, and taking the zeroth moment of the resulting equation, we can find

$$\frac{\partial}{\partial t}\int f \ln f \,\mathrm{d}^3\xi + \frac{\partial}{\partial x_\alpha}\int \xi_\alpha f \ln f \,\mathrm{d}^3\xi = \int \ln f\, \Omega(f)\,\mathrm{d}^3\xi. \tag{1.61}$$

This equation is a balance equation for the quantity $\mathcal{H}$. This is found as a moment of the Boltzmann equation, similarly to the the mass, momentum, and energy conservation equations found in section 1.3.4. As in these equations, the quantity $\int \xi_\alpha f \ln f \,\mathrm{d}^3\xi = \mathcal{H}_\alpha$ is the flow of the quantity $\mathcal{H}$, which can be split into an advective component $u_\alpha \int f \ln f \,\mathrm{d}^3\xi$ and a diffusive component $\int v_\alpha f \ln f \,\mathrm{d}^3\xi$.

For the BGK collision operator, the right-hand side of Eq. (1.61) can be found to be non-positive,

$$\begin{aligned}\int \ln f\, \Omega(f)\,\mathrm{d}^3\xi &= \int \ln\left(\frac{f}{f^{\mathrm{eq}}}\right)\Omega(f)\,\mathrm{d}^3\xi + \int \ln\left(f^{\mathrm{eq}}\right)\Omega(f)\,\mathrm{d}^3\xi \\ &= \frac{1}{\tau}\int \ln\left(\frac{f}{f^{\mathrm{eq}}}\right)\left(f^{\mathrm{eq}} - f\right)\mathrm{d}^3\xi \\ &= \frac{1}{\tau}\int f^{\mathrm{eq}} \ln\left(\frac{f}{f^{\mathrm{eq}}}\right)\left(1 - \frac{f}{f^{\mathrm{eq}}}\right)\mathrm{d}^3\xi \le 0.\end{aligned} \tag{1.62}$$

The $\ln(f^{\mathrm{eq}})\Omega(f)$ integral can be shown to disappear by inserting for f^{eq} and using the conservation constraints of the collision operator in Eq. (1.46). The last inequality follows from the general inequality $\ln x(1-x) \le 0$ for all $x > 0$. For $x = 1$, which corresponds to the equilibrium $f = f^{\mathrm{eq}}$, the inequality is exactly zero. This inequality can also be shown for Boltzmann's original collision operator, and can be considered a necessary criterion for any collision operator in kinetic theory.

Consequently, eq. (1.61) corresponds to the equation

$$\frac{\partial \mathcal{H}}{\partial t} + \frac{\partial \mathcal{H}_\alpha}{\partial x_\alpha} \le 0. \tag{1.63}$$

This shows us that $\mathcal{H}$ is not conserved in the system: It never increases, but instead it decreases unless the particle distribution has reached an equilibrium. This is called the *Boltzmann $\mathcal{H}$-theorem*. It states that molecular collisions invariably drive the distribution function towards equilibrium.[7]

At first sight, this seems analogous to how the thermodynamic quantity of entropy always increases in a system unless the system has reached an equilibrium characterised by an entropy maximum. Indeed, for ideal gases $\mathcal{H}$ is actually proportional to the entropy density ρs, [9, 14]

$$\rho s = -R\mathcal{H}. \tag{1.64}$$

The $\mathcal{H}$-theorem holds not only for the BGK collision operator, but also for Boltzmann's full collision operator for which the theorem was originally shown.

[7] A more expansive and rigorous explanation of the $\mathcal{H}$-theorem can be found elsewhere. [9, 7]

References

1. G.K. Batchelor, *An Introduction to Fluid Dynamics* (Cambridge University Press, 2000)
2. P.A. Thompson, *Compressible-Fluid Dynamics* (McGraw-Hill, 1972)
3. L.D. Landau, E.M. Lifshitz, *Fluid Mechanics* (Pergamon Press, 1987)
4. L.E. Kinsler, A.R. Frey, A.B. Coppens, J.V. Sanders, *Fundamentals of acoustics*, 4th edn. (John Wiley & Sons, 2000)
5. M. Born, H.S. Green, Proc. R. Soc. A **188**(1012), 10 (1946)
6. C.S. Wang Chang, G. Uhlenbeck, Transport phenomena in polyatomic gases. Tech. Rep. M604-6, University of Michigan (1951)
7. S. Chapman, T.G. Cowling, *The Mathematical Theory of Non-uniform Gases*, 2nd edn. (Cambridge University Press, 1952)
8. T.F. Morse, Phys. Fluids **7**(2), 159 (1964)
9. C. Cercignani, *The Boltzmann equation and its applications* (Springer, 1988)
10. T.I. Gombosi, *Gaskinetic Theory* (Cambridge University Press, 1994)
11. D. Hänel, *Molekulare Gasdynamik* (Springer, 2004)
12. P.L. Bhatnagar, E.P. Gross, M. Krook, Phys. Rev. E **94**(3), 511 (1954)
13. M. Greenspan, in *Physical Acoustics*, vol. IIA, ed. by W.P. Mason (Academic Press, 1965), pp. 1–45
14. E.T. Jaynes, American Journal of Physics **33**(5), 391 (1965)

Chapter 2
Numerical solvers

Scope of this chapter

- not more than a short overview for orientation for most methods (focus on FD and FV for the conventional CFD and only mention FEM and spectral methods in passing)
- highlighting existing approaches and their advantages and disadvantages; a detailed comparison of those methods with LBM or with each other is *not* the scope of this chapter
- at the end of this chapter, we should have a set of messages why the LBM is actually a good thing and worth investing time in
- say something about the properties of each scheme: time integration, convergence in space and time, computational cost and scalability, complexity of implementation and usage, limiting factors, applicability, numerical stability
- TODO (GS): You mentioned that you are aware of a few articles comparing LBM to other NS solvers. Could you add the references here?
- There are also direct simulation Monte-Carlo (DSMC) and Brownian Dynamics (BD) methods. We may mention them as well.

Brain storming: advantages and disadvantages of LBM

+ For interface phenomena, the kinetic nature of LBM should be more valid than top-down approaches in conventional CFD [1].
+ LBM has shown its strength in complex geometries and flows with moving boundaries [1] subject to mass conservation. Mesh generation in FEM for moving obstacles is a pain.
+ LBM has the potential to simulate flows with larger Knudsen number (but still < 1).
+ The viscosity in LBM is exact and one can, in the incompressible limit, obtain accurate solutions for any viscosity or Reynolds number [1].

+ As other pseudocompressible methods, LBM does not have a Poisson equation, which makes the algorithm simpler [1] and more scalable.
+ According to AK, LBM is of advantage for multicomponent flow in porous media (complexity of boundaries) and high Reynolds number flows (complexity of boundaries plus local momentum flux tensor).
– LBM is not well suited for the simulation of *compressible* flows due to the coupling of space and time steps [1].
– LBM is not well suited for thermal flows; reasons: Prandtl number restriction for BGK, allowable temperature variations are rather small, prone to instability [1].
– (maybe better moved to chapter 9) The Shan-Chen model has several disadvantages, *e.g.* the lack of a thermodynamically consistent temperature and the coupling of surface tension and equation of state [1].
– (maybe better moved to chapter 9) The free-energy model lacks Galilean invariance, and it does not take advantage of the kinetic nature of LBM (similar to Shan-Chen, actually) [1].
– There is no real kinetic multi-component model in LBM which is different from conventional CFD approaches. Also, the achievable ratio of viscosities and densities is a problem for multi-component/phase simulations [1].
– unstable in some parameter regions since H-theorem is lost during discretisation

Further comments regarding LBM and other solvers

- TK: [2] give reasons why lattice-based methods are more suitable for soft-matter simulations (section 2.3.2).
- AK: problem to simulate large Re is not only with LBM, but with any other numerical method. Moreover, the LBM is really effective as the explicit solver to resolve the turbulence if we don't use LES
- Wolf-Gladrow [3] also gives a short CFD overview in chapter 1. He also writes about LBM advantages and disadvantages in his summary (section 5.10).
- OS: Take a look at Timm's post in the forum about this, which refers to Succi's book regarding linearity/locality. Some advantages in LBM aren't that clear: e.g. for porous media and high Re. In my opinion, multiphase LBM wastes a lot of computations on aspects that don't contribute to the physics (with e.g. 2 fluids, you solve for their flow everywhere!)
- The article [4] contains a comparison of LBM with FV and FEM for laminar flows.
- There is a recent overview book [5].
- A brief overview of other CFD methods is provided in [2]. It is especially helpful to take a look at the Navier-Stokes bit (section 2.3.7).

TODO (EMV): Write an introduction to this chapter. Give a quick definition of order in accuracy in it (and refer to the proper one in Chapter 4), as the later sections use the term fairly often.

2.1 Conventional Navier-Stokes solvers

Conventional numerical methods work by taking the equation (or coupled system of equations) of interest and directly solving them by a particular method of approximation. In the case of computational fluid dynamics (CFD), the basic equations to be solved are the continuity equation and the Navier-Stokes equation (or their incompressible counterparts). Additional equations, such as an energy equation and an equation of state, may augment these; the choice of such additional equations depends on the physics to be simulated and the approximations used.

The derivatives in these equations are always discretised in some form so that the equations may be solved approximately on a computer. One simple example is the Euler approximation of a time derivative. By definition, a variable's derivative is its slope over an infinitesimal interval Δt, and this can be approximated using a finite interval Δt as

$$\frac{\partial y(t)}{\partial t} = \lim_{\Delta t \to 0} \frac{y(t+\Delta t) - y(t)}{\Delta t} \approx \frac{y(t+\Delta t) - y(t)}{\Delta t}. \tag{2.1}$$

Unsurprisingly, the accuracy of this approximation increases as Δt is made smaller and thus closer to its infinitesimal ideal. Additionally, the accuracy depends on the solution itself; as a rule of thumb, a rapidly varying solution requires a smaller value of Δt to reach a good level of accuracy than a slowly varying solution does.

Example 2.1. The forward Euler method can be used to find a numerical solution to simple equations. Consider the equation $\partial y(t)/\partial t = -y(t)$, with $y(0) = 1$. If we did not know already that the answer is $y(t) = \mathrm{e}^{-t}$ we might want to solve it step-by-step for discrete time steps $t_n = n\Delta t$ as

$$y_{n+1} = y_n + \Delta t \left.\frac{\partial y}{\partial t}\right|_{y=y_n} = (1 - \Delta t) y_n. \tag{2.2}$$

Here, y_n is the numerical approximation to $y(t_n)$. In this way, we would find $y_1 = (1 - \Delta t)y_0 = (1 - \Delta t)$, $y_2 = (1 - \Delta t)y_1$, and so forth. The resulting solutions for various values of Δt is shown in Fig. 2.1.

Exercise 2.1. Write a script implementing Eq. (2.2) from $t = 0$ to $t = 3$. Try out different values of Δt and show that the difference between the numerical solution $y_k|_{t_k=3}$ and the analytical solution $y(3) = \mathrm{e}^{-3}$ varies linearly with Δt.

While the forward Euler method is the simplest and fastest method to step the solution forward in time, other methods such as the implicit backward Euler method or

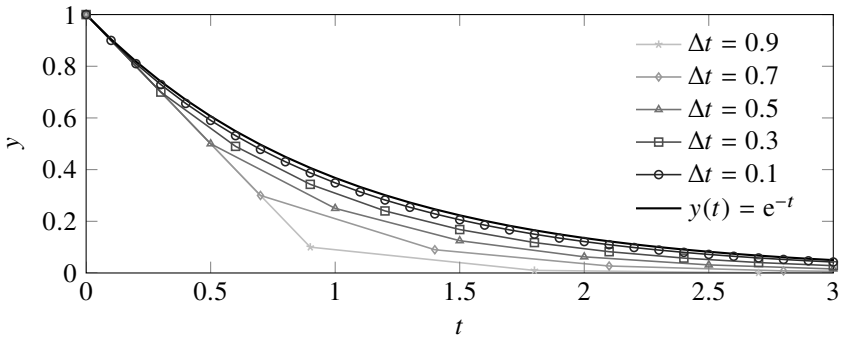


Fig. 2.1: Comparison of the analytical solution of $\partial y(t)/\partial t = -y(t)$ to forward Euler solutions with different values of Δt.

Runge-Kutta methods beat it in stability and accuracy [6].[1] Typically, conventional methods for unsteady CFD can use any of these methods in order to determine the solution at the next time step from the solution at the current time step.

However, these conventional CFD methods are distinguished the approach they use to represent the solution in continuous physical space by a finite set of numbers. All these methods must represent the fluid variables, such as fluid velocity $\boldsymbol{u}$ and pressure p, in such a way that their spatial derivatives can be found throughout the entire domain. In the following sections we will take a brief look at the basics of some of these methods.

2.1.1 Finite difference method

In the finite difference (FD) method, physical space is divided into a regular grid of nodes. In one dimension, these nodes are placed at the position $x_j = j\Delta x$. On each of these nodes, the solution variables are represented by a number; for a general quantity $\lambda(x)$, the exact solution $\lambda(x_j)$ is approximated by the discretised approximation denoted as λ_j.

Finite difference approximations of derivatives

At the base of the finite difference method, derivatives of λ are approximated by linear combinations ('finite differences') of λ_j. To find these differences, we consider the Taylor series of $\lambda(x)$ about x_j:

[1] Stability and accuracy, especially in terms of the lattice Boltzmann method, are later covered in Section 4.4 and Section 4.5, respectively.

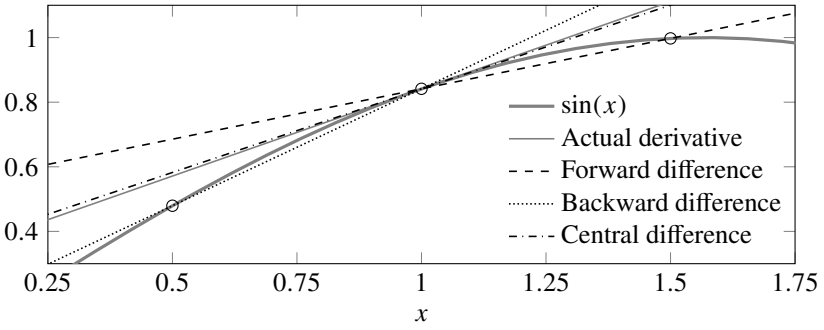


Fig. 2.2: Approximations of the derivative of $\sin(x)$ at $x = 1$, with $\Delta x = 0.5$.

$$\begin{aligned}\lambda(x_j + n\Delta x) &= \lambda(x_j) + (n\Delta x)\frac{\partial \lambda(x_j)}{\partial x} + \frac{(n\Delta x)^2}{2}\frac{\partial^2 \lambda(x_j)}{\partial x^2} + \dots \\ &= \sum_{m=0}^{\infty} \frac{(n\Delta x)^m}{m!}\frac{\partial^m \lambda(x_j)}{\partial x^m}.\end{aligned} \tag{2.3}$$

From this we can find three simple approximations for the first-order derivative,

$$\left.\frac{\partial \lambda}{\partial x}\right|_{x_j} \approx \frac{\lambda_{j+1} - \lambda_j}{\Delta x}, \quad \left.\frac{\partial \lambda}{\partial x}\right|_{x_j} \approx \frac{\lambda_{j+1} - \lambda_{j-1}}{2\Delta x}, \quad \left.\frac{\partial \lambda}{\partial x}\right|_{x_j} \approx \frac{\lambda_j - \lambda_{j-1}}{\Delta x}. \tag{2.4}$$

These three approximations are called the *forward* difference,[2] the *central* difference, and the *backward* difference approximations, respectively.

Exercise 2.2. Prove that the four approximations in Eq. (2.4) are valid by letting *e.g.* $\lambda_{j+1} \to \lambda(x_{j+1})$ and inserting Eq. (2.3), and show that the truncation error of the forward and backward difference approximations are $O(\Delta x)$ while that of the central difference approximation is $O(\Delta x^2)$.

The comparison in Fig. 2.2 indicates that central differences approximate the first derivative better, which is typically true and which can also be seen from its smaller $\mathrm{O}(\Delta x^2)$ truncation error.

We can also find an approximation for the second-order derivative with a $\mathrm{O}(\Delta x^2)$ truncation error:

$$\left.\frac{\partial^2 \lambda}{\partial x^2}\right|_{x_j} \approx \frac{\lambda_{j+1} - 2\lambda_j + \lambda_{j-1}}{\Delta x}. \tag{2.5}$$

From any such given finite difference scheme, it is possible to insert the Taylor expansion in order to determine not only what the scheme approximates, but also the truncation error of the approximation. This is detailed further in Section 4.5.1.

[2] The forward difference approximation corresponds to the forward Euler approximation for time discretisation, shown in Eq. (2.1).

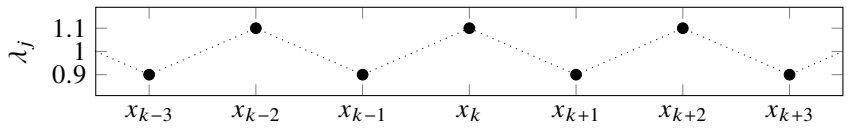


Fig. 2.3: A one-dimensional 'checkerboard' field around x_k.

Example 2.2. A finite difference approximation of the heat equation $\partial T/\partial t = \alpha \partial^2 T/\partial x^2$, where $T(x,t)$ is the temperature and α is the thermal diffusivity, is

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} = \alpha \frac{T_{j+1}^n - 2T_j^n - T_{j-1}^n}{\Delta x^2}. \tag{2.6}$$

Here the superscripts indicate the time step and the subscripts the spatial position, *e.g.* $T(x_j, t_n) \approx T_j^n$. We have used the forward Euler approximation from Eq. (2.1) to discretise the time derivative and Eq. (2.6) for the spatial second derivative. If we know the value of the solution at every point x_j at time t_n, along with the values at the edges of the system at all times, we can from these values determine the temperature at t_{n+1} for every point.

Finite difference methods for CFD

The finite difference method is simple in principle; just take a set of equations and replace the derivatives by finite difference approximations. However, this simple approach is often not sufficient in practice, and special techniques may be required for the set of equations in question. We will now touch on some problems and techniques of finding FD solutions of the Navier-Stokes equation, all of which are covered in more depth in the straightforward finite difference CFD book by Patankar [7].

We found above that the central difference scheme for first derivatives is typically more accurate than forward or backward schemes. However, in the advection term $\partial(\rho u_\alpha u_\beta)/\partial x_\beta$, information comes only from the opposite direction of the fluid flow, *i.e. upstream* or *upwind*.[3] Since the central difference scheme looks both upwind and downwind, it is possible to improve on it by using an *upwind scheme*, where either a forward or a backward scheme is used depending on the direction of fluid flow.

An issue requiring special treatment is the problem of *checkerboard instabilities*, where patterns of alternatingly high and low values emerge, reminiscent of the pattern on a checkerboard. A one-dimensional example is shown in Fig. 2.3. In short, the reason behind this pattern is that a central difference scheme would report the first derivative as being zero, so that the rapidly varying field is felt as being uniform.

[3] As a practical example, a deer can smell a hunter who is upwind of it, since the wind blows the hunter's scent towards the deer.

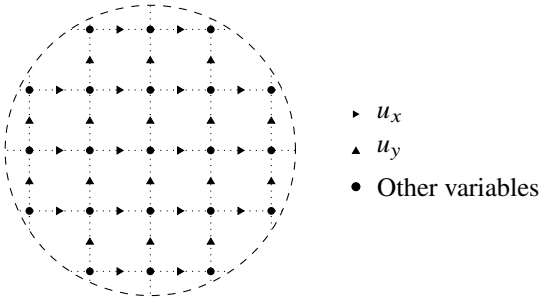


Fig. 2.4: A cutout of a staggered grid, where u_x and u_y are each stored in their own shifted grid of nodes.

A remedy to this problem is using a staggered grid as shown in Fig. 2.4, where different grids of nodes are used for different variables. The different grids are shifted relative to each other. Thus, when evaluating *e.g.* $\partial u_x/\partial x$ in one of the p nodes or $\partial p/\partial x$ in one of the u_x nodes, we can use a central difference scheme where only *adjacent* nodes are used. Thus, a field like that shown in Fig. 2.3 is no longer felt as being uniform, and checkerboard instabilities cannot emerge.

The Navier-Stokes equation is nonlinear, in particular in its advection term. Nonlinear equations are typically handled by iterating a series of 'guesses' for the nonlinear quantity. This is additionally complicated by having to couple a simultaneous set of equations. In the classic FD algorithms for incompressible flow called SIMPLE and SIMPLER, guesses for the pressure field and the velocity field are coupled and successively iterated using equations tailored for the purpose. More information on these somewhat complex algorithms can be found elsewhere [7].

Advantages and disadvantages

The crowning advantage of the finite difference method is that it is really quite simple in principle. For a number of simple equations it is not that much more difficult in practice, though some care must be taken in order to maintain stability and consistency [6].

However, fluids are governed by a complex set of coupled equations that contain several variables. Therefore, a number of special techniques need to be applied in order to use the FD method for CFD, which increases the amount of understanding and effort required to implement a FD CFD solver. However, the FD method can still be simple and effective compared to other conventional methods [5].

There are certain numerical weaknesses inherent to FD CFD. Unless special care is taken, the scheme is not conservative, meaning that the numerical

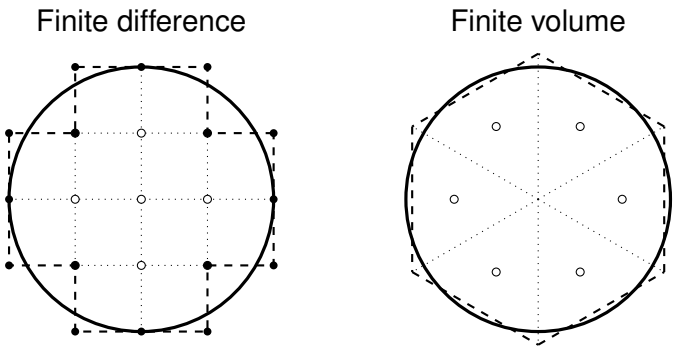


Fig. 2.5: Simple finite difference and finite volume discretisations of the volume inside a circular surface. The effective surface in each case is shown as black dashed lines, and interior nodes as white circles. To the right, the dotted lines show the finite volumes' interior edges.

errors cause the conservation of quantities like mass, momentum, and energy to not be perfectly respected [5]. Additionally, advective FD schemes are subject to false diffusion, where numerical errors cause the advected quantity to be diffused even in pure-advection cases that should have no diffusion [7]. (FD on irregular grids is in principle possible, but in practice it is hardly used [5].) Finally, since the FD method is based on a regular grid it has issues with complex geometries that do not conform to the grid itself [5]. The latter point is possibly the most important reason why other CFD methods have become more popular.

2.1.2 Finite volume method

In the finite volume (FV) method, space does not need to be divided into a *regular* grid. Instead, we subdivide the simulated volume V into many smaller volumes V_i, which may have different sizes and shapes from each other.[4] This allows for a better representation of complex geometries than *e.g.* the finite difference method, as illustrated in Fig. 2.5. In the middle of each finite volume V_i, there is a node where each solution variable $\lambda(x)$ is represented by its approximate average value $\bar{\lambda}_i$ within that volume.

[4] We here use the term 'volume' in a general sense, where a two-dimensional volume is an area and a one-dimensional volume is a line segment.

Finite volume approximation of conservation equations

The FV method is not as general as the FD method which can in principle be used for any equation. Rather, the FV method is designed to solve *conservation equations*, the type of equations which we typically find in *e.g.* fluid mechanics.[5] The FV method is *conservative* by design, which means that *e.g.* mass and momentum will always be conserved perfectly, unlike in the FD method.

To show the general principle of how FV approximates a conservation equation, we start with a steady advection-diffusion equation for a general quantity $\lambda(\boldsymbol{x}, t)$,

$$\nabla \cdot (\rho \lambda \boldsymbol{u}) = \nabla \cdot (\Gamma \nabla \lambda) + Q, \tag{2.7}$$

where the density ρ and the flow field $\boldsymbol{u}$ are assumed known, Γ is a diffusion coefficient for λ, and Q is a source term. By integrating this equation over the entire volume V and applying the divergence theorem, we get

$$\int_S (\rho \lambda \boldsymbol{u}) \cdot \mathrm{d}\boldsymbol{A} = \int_S (\Gamma \nabla \lambda) \cdot \mathrm{d}\boldsymbol{A} + \int_V Q \, \mathrm{d}V, \tag{2.8}$$

where S is the surface of the volume V and $\mathrm{d}\boldsymbol{A}$ is an infinitesimal surface normal element. The concept of the divergence theorem is as central to the FV method as it is for conservation equations in general: Sources and sinks of a quantity within a volume are balanced by that quantity's flux across the volume's boundaries.

This integral over the entire volume can be split up as a sum of integrals over the finite volumes V_i and their surfaces S_i,[6] and each such integral can be written as

$$\sum_{s_j \in S_i} \left[\int_{s_j} (\rho \lambda \boldsymbol{u}) \cdot \mathrm{d}\boldsymbol{A} \right] = \sum_{s_j \in S_i} \left[\int_{s_j} (\Gamma \nabla \lambda) \cdot \mathrm{d}\boldsymbol{A} \right] + V_i \bar{Q}_i. \tag{2.9}$$

Here, we have additionally split up the integrals over the surface S_i as a sum over its component surface segments s_j,[7] and the volume integral is replaced with the integrand's average value $\bar{Q}_i$ times the volume V_i.

Eq. (2.9) is still exact; no approximations have been made as long as the finite volumes $\{V_i\}$ together perfectly fill out the total volume V. However, as λ and $\nabla\lambda$ are not known on the surfaces S_j, the surface integrals must be related to the volume averages $\bar{\lambda}_i$. Using linear interpolation, this can be done in a simple way that leads to second-order accuracy [5]. Starting with the values of $\bar{\lambda}_i$ of the two volumes adjacent to the surface s_j, λ can be linearly interpolated between the two volumes' nodes so that each node point $\boldsymbol{x}_i$ has its corresponding volume's value of $\lambda(\boldsymbol{x}_i) = \bar{\lambda}_i$. At the point where the straight line between the two nodes crosses the surface s_j,

[5] That is not to say that the FV method is limited to conservation equations; it can also be used to solve more general hyperbolic problems [8].

[6] For the internal surfaces between adjacent finite volumes, the surface integrals from the two volumes will cancel each other.

[7] In Fig. 2.5, S_i is the triangular surface around each volume, and s_j represents the straight-line faces of these triangles.

we can find the linearly interpolated values of λ and $(\nabla\lambda) \cdot \mathrm{d}\boldsymbol{A}$. These values can then be applied to the entire surface in the surface integral.

Higher-order accuracy can be achieved by estimating the values of λ and $\nabla\lambda$ at more points on the surface, such as the surface edges which can be determined by interpolation from all the adjacent volumes [5]. Additionally, the interpolation of values on the surface may use node values from further-away volumes [5, 9].

Finite volume methods for CFD

While the basic formulations of finite volume and finite difference methods are different, CFD using FV methods bear many similarities to finite difference CFD as discussed in Section 2.1.1. For instance, for higher-order interpolation schemes, it is still generally a good idea to use more points in the upwind direction than in the downwind direction [5, 9]. Additionally, the iterative finite difference SIMPLE and SIMPLER schemes for CFD [7] and their descendants may also be adapted for finite volume simulations [9].

One difference is that the staggered grids typically used in FD CFD become too cumbersome to use for the irregular volumes typically used in FV CFD. While the issue of checkerboard instabilities is also present in the FV method for non-staggered grids, this is dealt with by the use of schemes that use more than two node values to approximate the first derivative at a point [9].

Advantages and disadvantages

While finite volume methods are formulated differently to finite difference methods, the two methods are comparable in their relative simplicity. The FV method has some additional advantages, however. The control volume formulation makes it fundamentally conservative; e.g. mass and momentum will be conserved throughout the entire domain in a closed system. Additionally, the FV method is very appropriate for use with irregular grids, which means that complex geometries can be captured well (the grid is adapted to the geometry), and it is straightforward to 'spend' more resolution on critical regions in the simulation by making the grid finer in these regions.

The downside of irregular grids is that making appropriate grids for complex geometries is itself a fairly complex problem; it is an entire field of study by itself. Additionally, higher-order FV methods are not straightforward to deal with, in particular in three dimensions and for irregular grids [5]. Additionally, FV is not as general a method as FD in terms of what equations it can solve, though this is typically not an issue for the equations encountered in CFD.

2.1.3 Other methods

Finite element methods

In finite element methods (FEM), PDEs are solved using an integral form known as the *weak form*, where the PDE itself is multiplied with a weight function $w(x)$ and integrated over the domain of interest. For example, the Helmholtz equation $\nabla^2\lambda + k^2\lambda = 0$ (a time-steady wave equation with wavenumber k, further explained in Section 12.1.4) in 1D becomes

$$\int w(x)\frac{\partial^2\lambda(x)}{\partial x^2}\,\mathrm{d}x + k^2\int w(x)\lambda(x)\,\mathrm{d}x = 0. \tag{2.10}$$

Generally, an unstructured grid can be used with FEM, with a discretised solution variable λ_i represented at each grid corner node $\boldsymbol{x}_i$. Between the grid corners, the variable $\lambda(\boldsymbol{x})$ is interpolated using *basis functions* $\phi_i(\boldsymbol{x})$ fulfilling certain conditions, *i.e.*

$$\lambda(x) \approx \sum_i \lambda_i\phi_i(x), \quad \text{for } \{\phi_i\} \text{ such that } \quad \lambda(x_i) = \lambda_i, \quad \sum_i \phi_i(x) = 1 \tag{2.11}$$

in our 1D example. The simplest 1D basis functions are linear functions such that $\phi_i(x_i) = 1$, $\phi_i(x_{j\neq i}) = 0$, and are non-zero only in the interval (x_{i-1}, x_{i+1}). However, a large variety of basis functions that are not linear (*e.g.* quadratic and cubic ones) are also available, and the order of accuracy is typically tied to the order of the basis functions.

Usually, the basis functions themselves are chosen as weighting functions, $w(x) = \phi_i(x)$. This leads to a system of equations, one for each unknown value λ_i. Through the integrals, each value of λ_i in our 1D example is related with λ_{i-1} and λ_{i+1}, assuming linear basis functions.

The main advantage of FEM is that it is mathematically well-equipped for unstructured grids and for increasing the order of accuracy through higher-order basis functions (though these also require more unknowns λ_i). These grids can be dynamically altered to compensate for moving geometry, as in the case of simulating a car crash. One disadvantage of FEM is that, like FD methods, it is not conservative by default like FV methods are. Another disadvantage is its complexity relative to FD and FV methods. For instance, the integrals become tricky to solve on general unstructured grids. And as with FD and FV methods, solving the complex Navier-Stokes system of equations is not straightforward [10]. The checkerboard instabilities described in Section 2.1.1 may appear here also unless special care is taken to deal with these [11]. While FEM has become ubiquitous in the field of solid mechanics, it has not been adopted as widely in CFD as FV and FD methods. NOTE (EMV): Does anyone have a good reference for this last statement?

2.2 Particle-based Navier-Stokes solvers

2.2.1 Multi-particle collision dynamics

Background

In 1999, Malevanets and Kapral [12, 13] introduced the multi-particle collision (MPC) dynamics, which has since become a popular method in the soft matter community. The paradigm of MPC is to coarse-grain the physical system as much as possible while still resolving the essential features of the underlying problem.

The MPC is a method of choice for complex systems where both hydrodynamic interactions and thermal fluctuations are relevant. Due to its particle-based nature, it is relatively easy to implement coupled systems of solvent and solutes. Therefore, MPC is most suitable and often employed for the modelling of colloids, polymers, vesicles and biological cells in equilibrium and external flow fields. MPC unfolds its strengths particularly for systems with Reynolds and Péclet numbers between 0.1 and 10 and for applications where consistent thermodynamics is required and where the macroscopic transport coefficients (viscosity, thermal diffusivity, self-diffusion coefficient) have to be accurately known [14].

There exist also MPC extensions for non-ideal [15], multi-component [16] and viscoelastic fluids [17]. We refer to [14, 18] for thorough reviews and to [19] for a recent overview.

Algorithm

The essential features of the MPC algorithm are: (i) alternating streaming and collision steps, (ii) local conservation of mass, momentum and, unlike standard LBM schemes, energy, (iii) isotropic discretisation. The last two properties ensure that MPC can be used as a viable Navier-Stokes solver.

The basic MPC setup comprises a large number of point-like particles with mass m. These particles can either be fluid or immersed (*e.g.* colloidal) particles. This feature allows a treatment of solvent and solutes on an equal footing. For example, immersed particles can be directly coupled by letting them participate in the collision and streaming steps [20]. This approach has been successfully employed in numerous colloid and polymer simulations (see [14] and references therein).

During propagation, space and velocity are continuous and the particles move along straight lines for one time step Δt:

$$\boldsymbol{x}_i(t + \Delta t) = \boldsymbol{x}_i(t) + \boldsymbol{c}_i(t)\Delta t, \tag{2.12}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{c}_i$ are particle position and velocity. After propagation, particles collide. How the collision step looks like in detail depends on the chosen MPC al-

gorithm. Generally, each particle-based algorithm with local mass and momentum conservation and an $\mathcal{H}$-theorem is called a multi-particle collision algorithm.

One special case is the so-called stochastic rotation dynamics (SRD) algorithm. During collision, all particles are sorted into cells of a usually regular cubic lattice with lattice constant Δx. On average, there are N_c particles in each cell. The velocity of each particle in one cell is decomposed into the average cell velocity (as given by the average velocity of all particles in that cell) and the relative velocity. The relative velocities are then rotated in space. In 2D, velocities are rotated by $\pm\alpha$ where α is a fixed angle and the sign is randomly chosen. In 3D, the rotation is defined by a fixed angle α and a random rotation axis. Rotations are the same for all particles in a given cell but statistically independent for different cells. Apart from this rotation, there is no direct interaction between particles. In particular, particles can penetrate each other, which makes a collision detection unnecessary. It can be shown that the equilibrium velocity distribution is Maxwellian.

It should be noted that the originally proposed SRD algorithm [12, 13] violated Galilean invariance. This problem, which was particularly important for small time steps (*i.e.* a small mean free path), could be corrected by shifting the lattice by a random distance $d \in [-\Delta x/2, +\Delta x/2]$ before each collision step [21].

Other collision models than SRD are available. For example, the Anderson thermostat (MPC-AT) [22, 23] is used to produce new particle velocities according to the canonical ensemble rather than merely rotating the existing velocity vectors in space.

It is also possible to implement a repulsion force between colloids and solvent particles [13] in order to keep the fluid outside the colloids. This, however, requires relatively large repulsion forces and therefore small time steps. Additional coupling approaches are reviewed in [14]. Slip [24] and no-slip boundary conditions [19] are available as well.

Advantages and disadvantages

All MPC algorithms locally conserve mass, momentum and energy and have an $\mathcal{H}$-theorem which makes them unconditionally stable [12]. Due to its locality the MPC algorithm is straightforward to implement and to use, computationally efficient and easy to parallelise. MPC has been successfully ported to GPUs with a performance gain of up to two orders of magnitude [19]. But due to its strong artificial compressibility, MPC is not suitable for the simulation of Stokes flow ($\mathrm{Re} \to 0$) or compressible hydrodynamics [14].

Both hydrodynamics and thermal fluctuations are consistently taken into account. For example, interfacial fluctuations in binary fluids are accurately captured. The hydrodynamic interactions can be switched off [25], which makes it possible to study their relevance. However, it is recommended to use other methods like Langevin or Brownian Dynamics if hydrodynamics is not desired [14]. When hydrodynamics is included, it allows for larger time steps than in MD-like methods. Therefore longer time intervals can be simulated with MPC [14].

Compared to LBM, MPC naturally provides thermal fluctuations, which can be an advantage. Yet, for systems where those fluctuations are not required or even distracting, MPC is not an ideal numerical method. Conventional Navier-Stokes solvers or the LBM are more favorable in those situations [14].

As MPC is a particle-based method, immersed objects such as colloids or polymers can be implemented in a relatively straightforward fashion. This makes MPC particularly suitable for the simulation of soft matter systems. On top, the transport coefficients (viscosity, thermal diffusivity, self-diffusion coefficient) can be accurately predicted as function of the simulation parameters [26, 21, 27, 28, 14]. On the other hand, it is not a simple task to impose hydrodynamic boundary conditions, especially for the pressure. Furthermore, the discussions in [19] show that no-slip boundary conditions and forcing are not as well under control as for LBM. Multicomponent fluids and also surfactants can be simulated within the MPC framework [16]. However, more work in this direction has been done in the LB community.

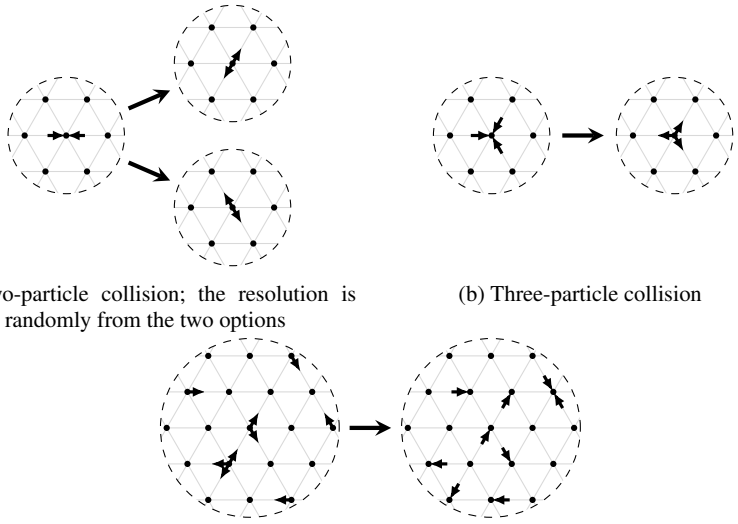
2.2.2 Lattice gas models

Background

Lattice gas models were first introduced in 1973 by Hardy, Pomeau, and de Pazzis as an extremely simple model of two-dimensional gas dynamics [29]. This particular model was subsequently named the HPP model after its inventors. In this model, ficticious particles exist on a square lattice where they stream forwards and collide in a manner that respects conservation of mass and momentum, much in the same way as molecules in a real gas. As the HPP lattice was square, each node had four neighbours and each particle had one of the four possible velocities $\boldsymbol{c}_i$ that would bring a particle to a neighbouring node in one time step.

However, it was not until 1986 that Frisch, Hasslacher, and Pomeau published a lattice gas model that could actually be used to simulate fluids [30]. Their model was also named after its inventors as the FHP model. The difference to the original HPP model is small, but significant: Instead of the square lattice and four velocities of the HPP model, the FHP model had a triangular lattice and six velocities $\boldsymbol{c}_i$. This change turned out to give the model sufficient lattice isotropy to perform fluid simulations [31, 32, 3].

Lattice gas models are especially interesting in the context of the lattice Boltzmann method, as the latter grew out of the former. Indeed, early lattice Boltzmann articles are difficult to read without knowledge of lattice gases, as these articles use much of the same formalism and methods.



(a) Two-particle collision; the resolution is chosen randomly from the two options

(b) Three-particle collision

(c) Streaming

Fig. 2.6: Rules of the FHP lattice gas model: collision and streaming.

Algorithm

Only up to one particle of a certain velocity can be present in a node at any time. Whether or not a particle of velocity $\boldsymbol{c}_i$ exists at the lattice node at $\boldsymbol{x}$ at time t is expressed by the *occupation number* $n_i(\boldsymbol{x}, t)$, where the index i refers to the velocity $\boldsymbol{c}_i$. This occupation number n_i is a Boolean variable with possible values of 0 and 1, representing the absence and presence of a particle, respectively.

This occupation number can be directly used to determine macroscopic observables: The mass density and momentum density in a node can be expressed as [32]

$$\rho(\boldsymbol{x}, t) = \frac{m}{v_0} \sum_i n_i(\boldsymbol{x}, t), \qquad \rho \boldsymbol{u}(\boldsymbol{x}, t) = \frac{m}{v_0} \sum_i \boldsymbol{c}_i n_i(\boldsymbol{x}, t), \tag{2.13}$$

respectively, where m is the mass of a particle and v_0 is the volume covered by the node.

There are two rules that determine the time evolution of a lattice gas. The first rule is *collision*, where particles that meet in a node may be redistributed in a way that conserves the mass and momentum in the node. Generally, collisions can be mathematically expressed as

$$n_i^\star = n_i + \Omega_i, \tag{2.14}$$

where $n_i^\star$ is the post-collision occupation number and $\Omega_i \in \{-1, 0, 1\}$ is a collision operator that may redistribute particles in a node, based on all occupation numbers $\{n_i\}$ in that node [31].

Which collisions may occur (*i.e.* the dependence of Ω_i on $\{n_i\}$) varies between different types of lattice gases, but in any case this is cumbersome to express mathematically [32, 3]. Rather, we will show graphically the two types of collisions in the original FHP model [30]. Fig. 2.6(a) shows the two possible resolutions between head-on collisions of two particles, which are chosen randomly with equal probability. Fig. 2.6(b) shows the resolution of a three-particle collision: When three particles meet with equal angles between each other, they are turned back to where they came from.

Exercise 2.3. Show that the macroscopic quantities of Eq. (2.13) are preserved by these collisions.

The second rule of a lattice gas is *streaming*: after collisions, particles move from their current node to a neigbouring node in their direction of velocity, as shown in Fig. 2.6(c). The particle velocities $\boldsymbol{c}_i$ are such that particles move exactly from one node to another from one time step to the next. For the FHP model, which has six velocities of equal magnitude, we have $|\boldsymbol{c}_i| = \Delta x/\Delta t$, Δx being the distance between nodes and Δt being the time step. Thus, the streaming can be expressed mathematically as

$$n_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t) = n_i^\star(\boldsymbol{x}, t). \tag{2.15}$$

Both rules can be combined into a single equation:

$$n_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = n_i(\boldsymbol{x}, t) + \Omega_i(\boldsymbol{x}, t). \tag{2.16}$$

In addition to the HPP and FHP models, there is a number of more complex lattice gas models. Their various features include rest particles with zero velocity, additional collisions, and additional higher-speed particles [32, 3]. However, all of these can be expressed mathematically through Eq. (2.16); the difference between them lies in the velocities $\boldsymbol{c}_i$ and the form of the collision operator Ω_i. All of these models which fulfil certain requirements on lattice isotropy (*e.g.*, FHP fulfils them while HPP does not) can be used for fluid simulations.

Advantages and disadvantages

One of the touted advantages of lattice gas models was the fact that, since the occupation numbers n_i are Boolean variables (particles are either *there* or *not there*), collisions are in a sense perfect so that the roundoff error inherent in the floating-point operations used in other CFD methods do not affect lattice gas models [31]. Additionally, collisions are completely local within each node, and since collisions are the most computationally demanding component of a lattice gas program, lattice gases can be massively parallelised [31].

However, a downside of these collisions is that they get out of hand for larger number of velocities. For example, for the three-dimensional lattice gas with 24 velocities, there are $2^{24} \approx 16.8 \times 10^6$ possible states in a node. The resolution of

any collision in this model was typically determined by lookup in a huge table made by a dedicated program [31].

The FHP model additionally has problems with isotropy of the Navier-Stokes equations, which can only disappear in the limit of low Mach numbers, *i.e.* for a quasi-incompressible flow [31]. Additionally, lattice gases struggled to reach as high Reynolds numbers as comparable CFD methods [31].

The major issue with lattice gases, however, was *statistical noise*. Like real gases, lattice gases are teeming with activity at the microscopic level. Even for a gas at equilibrium, when we make a control volume smaller and smaller, the density (mass per volume) inside it will fluctuate more and more strongly with time; molecules continually move in and out, and the law of large numbers applies less for smaller volumes. This is also the case with lattice gases, where the macroscopic values from Eq. (2.13) will fluctuate even for a lattice gas at equilibrium.

In one sense, it may be an advantage that lattice gases can qualitatively capture the thermal fluctuations of a real gas [32]. But if the goal is to simulate a macroscopic fluid, these fluctuations are a nuisance. For that reason, lattice gas simulations would typically report density and fluid velocity found through averaging in space and/or time (*i.e.* over several neighbouring nodes and/or several adjacent time steps), and even averaging over multiple ensembles (*i.e.* macroscopically similar but microscopically different realisations of the system) [32], though this could only reduce the problem and some noise would always remain.

The problem of statistical noise was more completely dealt with by the invention of the lattice Boltzmann method (LBM). This method was first introduced by tracking the occupation number expectation's value $f_i = \langle n_i \rangle$ rather than the occupation number itself, thus eliminating the statistical noise. This was the original method of deriving the LBM, and it was not until 1997 that it was found how to derive it from the kinetic theory of gases presented in section 1.3 [33]. This more modern approach of derivation is the one that we will follow in Chapter 3.

2.2.3 Other methods

We will now give a very brief overview of some other particle-based methods.

Smoothed-particle hydrodynamics (SPH)

SPH was invented in the 1970s to deal with the particular challenges of 3D astrophysics. Since then it has been used in a large number of applications, in recent years also computer graphics where it can simulate convincing fluid flow relatively cheaply. Several books have been written on SPH, such as a mathematically rigorous introduction by Violeau [34] and a more practical introduction by Liu and Liu [35] on which the rest of our description will be based.

At the base of SPH is an interpolation scheme which uses point particles that influence their vicinity. For instance, any quantity λ at point $\boldsymbol{x}$ can be approximated as a sum over all particles, with each particle j positioned at $\boldsymbol{x}_j$:

$$\lambda(\boldsymbol{x}) = \sum_j \frac{m_j}{\rho_j} \lambda_j W(|\boldsymbol{x} - \boldsymbol{x}_j|, h_j). \tag{2.17}$$

Here, m_j is the particle's mass, ρ_j is the density at $\boldsymbol{x}_j$, λ_j is the particle's value of λ, and $W(|\boldsymbol{x} - \boldsymbol{x}_j|, h_j)$ is a kernel function with characteristic size h_j which defines the region of influence of particle j.[8] Thus, SPH particles can be seen as overlapping blobs, and the sum these blobs at $\boldsymbol{x}$ determines $\lambda(\boldsymbol{x})$. For instance, the density ρ_i at particle i can be found by setting $\lambda(\boldsymbol{x}_i) = \rho(\boldsymbol{x}_i)$, so that

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W(|\boldsymbol{x} - \boldsymbol{x}_j|, h_j) = \sum_j m_j W(|\boldsymbol{x} - \boldsymbol{x}_j|, h_j). \tag{2.18}$$

The formulation of SPH and its adaptive resolution gives it a great advantage when dealing with large unbounded domains with huge density variations, such as in astrophysics. It can also deal with extreme problems with large deformations, such as explosions and high-velocity impacts, where more traditional methods may struggle. The particle formulation of SPH also allows for perfect conservation of mass and momentum.

On the downside, SPH has problems with accuracy and it is not quite simple to deal with boundary conditions. Additionally, the formulation of SPH makes it difficult to mathematically prove that the numerical method is consistent with the equations of the hydrodynamics that it is meant to simulate.

2.3 Summary

NOTE (EMV): I have added some notes as comments in the .tex file.

References

1. R.R. Nourgaliev, T.N. Dinh, T.G. Theofanous, D. Joseph, Int. J. Multiphas. Flow **29**(1), 117 (2003)
2. B. Dünweg, A.J.C. Ladd, in *Advances in Polymer Science* (Springer Berlin Heidelberg, 2008), pp. 1–78
3. D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models* (Springer, 2005)

[8] While the kernel function can be *e.g.* Gaussian, it is advantageous to choose kernels that are zero for $|\boldsymbol{x} - \boldsymbol{x}_j| > h$, so that only particles in the vicinity of $\boldsymbol{x}$ need be included in the sum. Additionally, the fact that h_j can be particle-specific and varying allows adaptive resolution.

4. S. Geller, M. Krafczyk, J. Tölke, S. Turek, J. Hron, Computers & Fluids **35**(8–9), 888 (2006)
5. J.H. Ferziger, M. Peric, A. Leonard, *Computational Methods for Fluid Dynamics*, vol. 50, 3rd edn. (Springer, 2002)
6. R.J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady State and Time Dependent Problems* (SIAM, 2007)
7. S.V. Patankar, *Numerical heat transfer and fluid flow* (Taylor & Francis, 1980)
8. R.J. LeVeque, *Finite-Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics (Cambridge University Press, 2004)
9. H.K. Versteeg, W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*, 2nd edn. (Pearson Education Ltd, 2007)
10. O.C. Zienkiewicz, R.L. Taylor, P. Nithiarasu, *The finite element method for fluid dynamics*, 7th edn. (Butterworth-Heinemann, 2014)
11. C.A.J. Fletcher, *Computational techniques for fluid dynamics*, vol. 2 (Springer-Verlag, 1988)
12. A. Malevanets, R. Kapral, J. Chem. Phys. **110**(17), 8605 (1999)
13. A. Malevanets, R. Kapral, J. Chem. Phys. **112**(16), 7260 (2000)
14. G. Gompper, T. Ihle, D.M. Kroll, R.G. Winkler, Advances in Polymer Science (Springer Berlin Heidelberg, 2008), pp. 1–87
15. T. Ihle, E. Tüzel, D.M. Kroll, Europhys. Lett. **73**(5), 664 (2006)
16. E. Tüzel, G. Pan, T. Ihle, D.M. Kroll, Europhys. Lett. **80**(4), 40010 (2007)
17. Y.G. Tao, I.O. Götze, G. Gompper, J. Chem. Phys. **128**(14), 144902 (2008)
18. R. Kapral, in *Advances in Chemical Physics*, ed. by S.A. Rice (John Wiley & Sons, Inc., 2008), p. 89–146
19. E. Westphal, S.P. Singh, C.C. Huang, G. Gompper, R.G. Winkler, Comput. Phys. Commun. **185**(2), 495 (2014)
20. A. Malevanets, J.M. Yeomans, Europhys. Lett. **52**(2), 231 (2000)
21. T. Ihle, D.M. Kroll, Phys. Rev. E **67**(6), 066706 (2003)
22. E. Allahyarov, G. Gompper, Phys. Rev. E **66**(3), 036702 (2002)
23. H. Noguchi, N. Kikuchi, G. Gompper, Europhys. Lett. **78**(1), 10005 (2007)
24. J.K. Whitmer, E. Luijten, J. Phys. Condens. Matter **22**(10), 104106 (2010)
25. M. Ripoll, R.G. Winkler, G. Gompper, Eur. Phys. J. E **23**(4), 349 (2007)
26. N. Kikuchi, C.M. Pooley, J.F. Ryder, J.M. Yeomans, J. Chem. Phys. **119**(12), 6388 (2003)
27. T. Ihle, E. Tüzel, D.M. Kroll, Phys. Rev. E **72**(4), 046707 (2005)
28. C.M. Pooley, J.M. Yeomans, J. Phys. Chem. B **109**(14), 6505 (2005)
29. J. Hardy, Y. Pomeau, O. de Pazzis, Journal of Mathematical Physics **14**(12), 1746 (1973)
30. U. Frisch, B. Hasslacher, Y. Pomeau, Phys. Rev. Lett. **56**(14), 1505 (1986)
31. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, 2001)
32. J.P. Rivet, J.P. Boon, *Lattice Gas Hydrodynamics* (Cambridge University Press, 2001)
33. X. He, L.S. Luo, Phys. Rev. E **56**(6), 6811 (1997)
34. D. Violeau, *Fluid Mechanics and the SPH Method: Theory and Applications*, 1st edn. (Oxford University Press, Oxford, UK, 2012)
35. G.R. Liu, M.B. Liu, *Smoothed Particle Hydrodynamics: a meshfree particle method* (World Scientific Publishing, Singapore, 2003)

Chapter 3
The lattice Boltzmann equation

The equations of fluid mechanics are notoriously difficult to solve in general. Analytical solutions can be found only for quite basic cases, such as the Couette or Poiseuille flows shown in Fig. 1.1. Cases with more complex geometries and boundary conditions must typically be solved using numerical methods.

However, as we have seen in chapter 2, the numerical methods used to solve the equations of fluid mechanics can be difficult both to implement and to parallelise. Many of these methods use so-called *implicit* numerical schemes which require performing matrix operations that are expensive and hard to parallelise. TODO (EMV): I really need to revisit this statement when that chapter has actually been written...

The Boltzmann equation in section 1.3 describes the dynamics of a gas on a mesoscopic scale. From section 1.3.4 we also know that the Boltzmann equation leads to the equations of fluid dynamics on the macroscale. Therefore, from a solution of the Boltzmann equation for a given case we can often find a solution to the Navier-Stokes equation for the same case.[1]

The problem with this idea is that the Boltzmann equation is even more difficult to solve analytically than the Navier-Stokes equation. Indeed, its fundamental variable, the distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$, is a function of seven parameters: x, y, z, ξ_x, ξ_y, ξ_z and t. However, we can try a different approach; if we can solve the Boltzmann equation *numerically*, this may also indirectly give us a solution to the Navier-Stokes equation.

In fact, the numerical scheme for the Boltzmann equation somewhat paradoxically turns out to be *quite simple*, both to implement and to parallelise. The reason is that the force-free Boltzmann equation is a simple hyperbolic equation which essentially describes the advection of the distribution function f with the particle velocity $\boldsymbol{\xi}$. In addition, the source term $\Omega(f)$ depends only on the local value of f and not on its gradients.

Not only is the discretised Boltzmann equation simple to implement, it also has certain numerical advantages over conventional methods that directly discretise the

[1] Since the Boltzmann equation is more general, it also has solutions that *do not* correspond to Navier-Stokes solutions. The connections between these two equations will be further explored in Section 4.1.

equations of fluid mechanics, such as finite difference or finite volume methods. A major difficulty with these methods is discretising the advection term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$; complicated numerical schemes with approximation errors are introduced to deal with this. On the other hand, the discretised Boltzmann equation takes a very different approach that results in ideal advection. NOTE (EMV): Anyone (especially TK), do you have a source for this?

This chapter starts with a simple overview of the lattice Boltzmann method. In Section 3.1, the equation is briefly presented without derivation in order to give an initial understanding. Section 3.2 presents general advice on implementing the algorithm on a computer. Together, these two sections should be sufficient to write simple LBM codes.

Next, we present the derivation of the lattice Boltzmann equation, which is found by discretising the Boltzmann equation. This is done in two steps. First, we discretise velocity space in section 3.3 by limiting the continuous particle velocity $\boldsymbol{\xi}$ to a discrete set of velocities $\{\boldsymbol{\xi}_i\}$. Secondly, we discretise physical space and time in section 3.4 by *integrating along characteristics*. The result of these two steps is the *lattice Boltzmann equation* (LBE).

Historically, the LBE was not found along these lines. It rather evolved out of the lattice gas automata described in section 2.2.2 and many early articles on the lattice Boltzmann method (LBM) are written from this perspective. The connection between lattice gases and the LBE is described in detail *e.g.* in [1].

Throughout this chapter we use the force-free form of the Boltzmann equation for the sake of simplicity. The inclusion of external forces in the lattice Boltzmann (LB) scheme is covered in Chapter 6 and follows the same approach as presented here.

3.1 Summary of the lattice Boltzmann method

Before diving into the derivation of the lattice Boltzmann equation, we will first give a summary of the lattice Boltzmann method. This serves two purposes: First, it is unnecessary to know all the details of the derivation in order to implement simple codes. Therefore, this section, along with Section 3.2, gives a quick introduction for readers who only wish to know the most relevant results. Secondly, for readers who *are* interested in understanding the derivations, knowing their end result will make them more simple to understand.

3.1.1 Overview

The basic quantity of the LBM is the *discrete-velocity distribution function* $f_i(\boldsymbol{x}, t)$. Similar to the distribution function introduced in Section 1.3, it represents the density of particles with velocity $\boldsymbol{c}_i = (c_{ix}, c_{iy}, c_{iz})$ at position $\boldsymbol{x}$ and time t. Likewise, the

mass density ρ and momentum density $\rho\boldsymbol{u}$ at $(\boldsymbol{x}, t)$ can be found through weighted sums known as *moments* of f_i,

$$\rho(\boldsymbol{x}, t) = \sum_i f_i(\boldsymbol{x}, t), \qquad \rho\boldsymbol{u}_i(\boldsymbol{x}, t) = \sum_i \boldsymbol{c} f_i(\boldsymbol{x}, t). \tag{3.1}$$

The major difference between f_i and the continuous distribution function f is that all of the argument variables of f_i are discrete. $\boldsymbol{c}_i$, to which the subscript i in f_i refers, is one of a small discrete set of velocities $\{\boldsymbol{c}_i\}$. The points $\boldsymbol{x}$ at which f_i is defined are positioned as a square lattice in space, with lattice spacing Δx. Additionally, f_i is defined only at certain times t, separated by a time step Δt.

The discrete velocities $\boldsymbol{c}_i$ require further explanation. Together with a corresponding set of weighting coefficients w_i which will be explained shortly, they form *velocity sets* $\{\boldsymbol{c}_i, w_i\}$. Different velocity sets are used for different purposes. These velocity sets are usually denoted by the notation DdQq, where d is the number of spatial dimensions the velocity set covers and q is the set's number of velocities.

The most commonly used velocity sets to solve the Navier-Stokes equation are D1Q3, D2Q9, D3Q15, D3Q19 and D3Q27. They are shown in fig. 3.3 and fig. 3.4 and characterised in tab. 3.1. The velocity components $\{c_{i\alpha}\}$ and weights $\{w_i\}$ are also collected in tab. 3.2, tab. 3.3, tab. 3.4, tab. 3.5, and tab. 3.6.

Typically, we want to use as few velocities as possible as this minimises memory requirements and computation speed. However, there is a tradeoff between smaller velocity sets[2] (*e.g.* D3Q15) and higher accuracy (*e.g.* D3Q27). In 3D, a suitable all-purpose velocity set is D3Q19.

NOTE (GS): Could you please add some reference to sentence "In 3D, a suitable all-purpose velocity set is D3Q19.". I find this sentence too generic, and possibly not even true (although often taken as so). With a reference we are defended. NOTE (EMV): I don't have a reference. Anyone else?

A constant c_s^2 can also be found from each velocity set as in Eq. (3.62). In the basic isothermal LB equation, it determines the relation $p = c_s^2\rho$ between pressure p and density ρ. Because of this relation, it can be shown that c_s represents the isothermal model's *speed of sound*.[3] In all the velocity sets mentioned above, this constant is $c_s^2 = 1/3$.

By discretising the Boltzmann equation in velocity space, physical space, and time, the *lattice Boltzmann equation*,

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + \Omega_i(\boldsymbol{x}, t), \tag{3.2}$$

[2] There is a limit to how small a velocity set can be, though, as it has to obey the requirements shown in Eq. (3.62) to be suitable for Navier-Stokes simulations. The smallest velocity set in 3D is D3Q13, but it has the disadvantage of being tricky to apply.

[3] This statement is not proven in this chapter, but rather in section 12.1 on sound waves.

is found. This expresses that particles $f_i(\boldsymbol{x}, t)$ move with their velocity $\boldsymbol{c}_i$ to a neighbouring point $\boldsymbol{x} + \boldsymbol{c}_i \Delta t$ at the next time step $t + \Delta t$ as shown in Fig. 3.1,[4] while being affected by a collision operator Ω_i. This operator models particle collisions by redistributing particles among the populations f_i in each node, so that populations are typically not streamed unchanged: typically, $f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) \neq f_i(\boldsymbol{x}, t)$.

While there are many different collision operators Ω_i available, the simplest one that can be used for Navier-Stokes simulations is the Bhatnagar-Gross-Krook (BGK) operator,

$$\Omega_i(f) = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau} \Delta t, \tag{3.3}$$

which relaxes the populations towards an equilibrium f_i^{eq} at a rate determined by the relaxation time τ.[5]

This equilibrium is given by

$$f_i^{\mathrm{eq}}(\boldsymbol{x}, t) = w_i \rho \left(1 + \frac{\boldsymbol{u} \cdot \boldsymbol{c}_i}{c_\mathrm{s}^2} + \frac{(\boldsymbol{u} \cdot \boldsymbol{c}_i)^2}{2 c_\mathrm{s}^4} - \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2 c_\mathrm{s}^2} \right), \tag{3.4}$$

and the velocity set-specific weights w_i are used here. The equilibrium is such that its moments are the same as those of f_i, *i.e.* $\sum_i f_i^{\mathrm{eq}} = \sum_i f_i = \rho$ and $\sum_i \boldsymbol{c}_i f_i^{\mathrm{eq}} = \sum_i \boldsymbol{c}_i f_i = \rho \boldsymbol{u}$.[6] The equilibrium f_i^{eq} depends on the local quantities of density ρ and fluid velocity $\boldsymbol{u}$. These are calculated from the local values of f_i by Eq. (3.1), with the fluid velocity found as $\boldsymbol{u}(\boldsymbol{x}, t) = \rho \boldsymbol{u}(\boldsymbol{x}, t) / \rho(\boldsymbol{x}, t)$.

The link between the lattice Boltzmann equation can be determined using the Chapman-Enskog analysis, as in Section 4.1. Through this, it can be shown that the LB equation results in macroscopic behaviour according to the Navier-Stokes equation, with the kinematic shear viscosity given by the relaxation time τ as

$$\nu = c_\mathrm{s}^2 \left(\tau - \frac{\Delta t}{2} \right). \tag{3.5}$$

Additionally, the viscous stress tensor can be calculated as

$$\sigma_{\alpha\beta} \approx -\left(1 - \frac{\Delta t}{2\tau} \right) \sum_i c_{i\alpha} c_{i\beta} f_i^{\mathrm{neq}}, \tag{3.6}$$

where $f_i^{\mathrm{neq}} = f_i - f_i^{\mathrm{eq}}$ is the deviation of f_i from equilibrium. (However, computing the stress tensor explicitly in this way is usually not a necessary step when performing simulations.)

---

[4] Usually, the velocity sets $\{\boldsymbol{c}_i\}$ are chosen such that any spatial vector $\boldsymbol{c}_i \Delta t$ points from one lattice site to a neighbouring lattice site. This guarantees that the populations f_i always reach another lattice site during a time step Δt, rather than being trapped between them.

[5] Other collision operators are available which use additional relaxation times to achieve increased accuracy and stability. These are discussed in Section **??**.

[6] As a consequence of this, mass and momentum are conserved in collisions. This conservation can be expressed mathematically as $\sum_i \Omega_i = 0$ and $\sum_i \boldsymbol{c}_i \Omega_i = \boldsymbol{0}$.

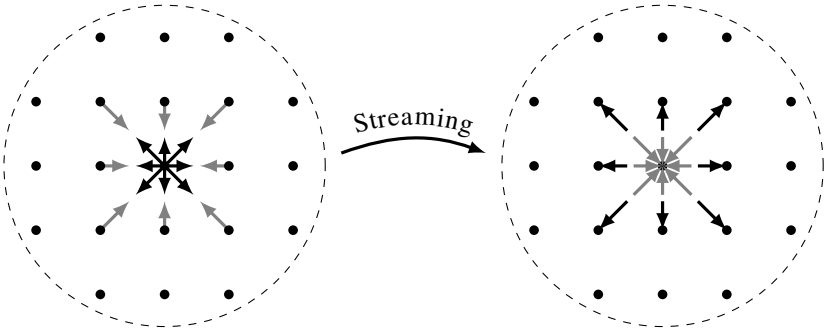


Fig. 3.1: Particles (black) streaming from the central node to its neighbours, from which particles (gray) are streamed back. To the left we see post-collision distributions $f_i^\star$ *before* streaming, and to the right we see pre-collision distributions f_i *after* streaming.

3.1.2 The time step: collision and streaming

In full, the lattice BGK (LBGK) equation (*i.e.* the fully discretised Boltzmann equation with the BGK collision operator) reads

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right). \tag{3.7}$$

This equation can be decomposed into two distinct parts that are performed in succession:

1. The first part is collision (or relaxation),

$$f_i^\star(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right), \tag{3.8}$$

   where $f_i^\star$ represents the distribution function *after* collisions and f_i^{eq} is found from f_i through Eq. (3.4).[7]
2. The second part is streaming (or propagation),

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i^\star(\boldsymbol{x}, t), \tag{3.10}$$

   as shown in Fig. 3.1.

[7] It is convenient and efficient to implement collision in the form

$$f_i^\star(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t)\left(1 - \frac{\Delta t}{\tau}\right) + f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\frac{\Delta t}{\tau}. \tag{3.9}$$

This becomes particularly simple for $\tau/\Delta t = 1$, where $f_i^\star(\boldsymbol{x}, t) = f_i^{\mathrm{eq}}(\boldsymbol{x}, t)$.

Collision is local and streaming is exact (*cf.* section 3.4).

Overall, the LBE concept is straightforward. It consists of two parts: *collision* and *streaming*. The collision is simply an algebraic *local* operation. First, one calculates the density ρ and the macroscopic velocity $\boldsymbol{u}$. This allows finding the equilibrium distributions f_i^{eq} as in eq. (3.4) and the post-collision distribution $f_i^\star$ as in eq. (3.9). Once this collision operation is performed, we transfer the resulting distribution $f_i^\star$ into neighbouring nodes as in eq. (3.10); this operation is called *streaming*. When these two operations have been performed, one time step has elapsed, and the operations are repeated.

3.2 Implementation

This section first covers initialisation and the LB algorithm. Then, some details about the underlying memory structure and coding hints are provided. The case covered here is the simplest force-free LB algorithm in the absence of boundaries. The inclusion of a force and boundary conditions are just additional steps added to the LB algorithm core. Boundary conditions are discussed in chapter 5 while forces are introduced in chapter 6. More in-depth implementation advice can be found in chapter 13.

3.2.1 Initialisation

The simplest approach to initialise the populations at the start of a simulation is to set them to $f_i^{\mathrm{eq}}(\boldsymbol{x}, t = 0) = f_i^{\mathrm{eq}}(\rho(\boldsymbol{x}, t = 0), \boldsymbol{u}(\boldsymbol{x}, t = 0))$ *via* eq. (3.4). Often, the values $\rho(\boldsymbol{x}, t = 0) = 1$ and $\boldsymbol{u}(\boldsymbol{x}, t = 0) = \boldsymbol{0}$ are used for the initialisation. More details about different initialisation schemes are given in section 5.4.

3.2.2 Timestep algorithm

Overall, the LBM core algorithm consists of a cyclic sequence of substeps, with one cycle for each timestep. These are also visualised in fig. 3.2:

1. Perform streaming (propagation) *via* eq. (3.10). Note that the streaming and increase of the time step can be moved in this cyclic algorithm right after the collision step, *i.e.* item 6.
2. Increase the time step, $t \to t + \Delta t$.
3. Compute the macroscopic moments $\rho(\boldsymbol{x}, t)$ and $\boldsymbol{u}(\boldsymbol{x}, t)$ from $f_i(\boldsymbol{x}, t)$ *via* eq. (**??**).

4. Obtain the equilibrium distribution $f_i^{\mathrm{eq}}(\boldsymbol{x}, t)$ from eq. (3.4).[8]
5. If desired, compute the viscous stress tensor from eq. (3.6).
6. Perform collision (relaxation) as shown in eq. (3.9).
7. If desired, write the macroscopic fields $\rho(\boldsymbol{x}, t)$, $\boldsymbol{u}(\boldsymbol{x}, t)$ and/or $\boldsymbol{\sigma}(\boldsymbol{x}, t)$ to the disk for visualisation or post-processing.
8. Go back to step 2 until the last time step or convergence of a simulation has been reached.

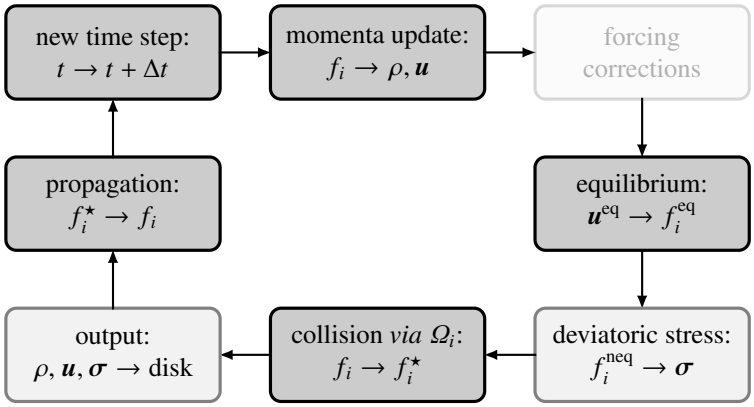


Fig. 3.2: Order of the bulk LB algorithm in the absence of forces. Lighter boxes denote optional sub-steps (deviatoric stress tensor and data output) which are not relevant for the evolution of the LB algorithm itself. The pale box in the top right corner already indicates where forcing corrections will be added in chapter 6. Note that the output may also be written before collision as collision does not overwrite the macroscopic fields.

Note that the certain order of these substeps is important. In particular the stress tensor has to be computed after f_i^{eq} is obtained and before collision is performed. The macroscopic fields may also be written to the disk before collision. Some people start a simulation with collision rather with streaming. This does not make a significant difference. Details are provided in section 5.4.

3.2.3 Notes on memory layout and coding hints

TODO (OS): Please take a close look and elaborate on this if necessary. We should provide basic features here and should get back to this point in the final chapter, where you can be much more specific. In general I do not know how many details to

[8] For those who want ready-to-program expressions, the unrolled equilibrium functions for D1Q3 and D2Q9 are shown in eq. (3.67) and eq. (3.68).

give at this point. Some of the readers may not at all have coding experience, but we cannot provide coding lectures as well. NOTE (AK): I think Orest needs to rewrite this section in the order which resembles the order above. I sketched it. As well I strongly recommend to have some codes on lbmworkshop.com

This section gives some coding hints which one may utilise while setting up LBM simulations.

We follow the order of the LB algorithm presented above:

Initialization

The arrays need to be allocated for the macroscopic fields, *i.e.* ρ and $\boldsymbol{u}$, and populations. Usually, for two-dimensional case the arrays are allocated as two-dimensional arrays $\rho[\mathrm{N_x}][\mathrm{N_y}]$ and $\boldsymbol{u}[\mathrm{N_x}][\mathrm{N_y}]$ and populations are defined as a three-dimensional array $f[\mathrm{N_x}][\mathrm{N_y}][\mathrm{q}]$, where $\mathrm{N_x}$ and $\mathrm{N_y}$ are the number of lattice nodes in x and y directions, q is the number of populations for D2Qq model. Here we follow C++ notation but users can easily adapt the scheme to any other programming language. For the three-dimensional problems (and sometimes for the two-dimensional problems too), one usually maps the three-dimensional space to the one-dimensional array as not many programming languages support multi-dimensional arrays. This might be done using the following equations, which represent one possible mapping:

$$\begin{aligned} \{\rho, \boldsymbol{u}\}[\mathrm{i}][\mathrm{j}][\mathrm{k}] &\rightarrow \{\rho, \boldsymbol{u}\}[\mathrm{kN_xN_y} + \mathrm{jN_x} + \mathrm{i}] \\ f[\mathrm{i}][\mathrm{j}][\mathrm{k}][\mathrm{l}] &\rightarrow f[(\mathrm{kN_xN_y} + \mathrm{jN_x} + \mathrm{i})\mathrm{q} + \mathrm{l}], \end{aligned} \tag{3.11}$$

where i is the x cell index, j is the y cell index, k is the z cell index, l is the population index.

Streaming

The streaming step has to be implemented such that streamed populations do not overwrite memory blocks which contain still unstreamed populations. There are two common ways to do it:

- One way is to run through memory in 'opposite streaming direction' and use a small temporary buffer for a single population. The difficulty is that memory has to be sweeped in different directions for each discrete velocity, which is an excellent breeding ground for bugs.
- Allocate memory for two sets of populations, f_i^{old} and f_i^{new}, in order to store two subsequent time steps. During streaming, copy data from $f_i^{\mathrm{old}}(\boldsymbol{x})$ and write to $f_i^{\mathrm{new}}(\boldsymbol{x} + \boldsymbol{c}_i\Delta t)$. Swap f_i^{old} and f_i^{new} after each time step so that the previously new populations become the currently old populations. This way, one does not have to care about how to stream populations without overwriting data which is

still required. The resulting code is significantly easier than for the first option above, but one requires two times more memory[9].

Updating macroscopic variables

As has been shown previously, the local density and momentum are calculated as summation of the populations and velocity set:

$$\begin{aligned} \rho[\mathrm{i}][\mathrm{j}][\mathrm{k}] &= \sum_l f[\mathrm{i}][\mathrm{j}][\mathrm{k}][\mathrm{l}] \\ \rho[\mathrm{i}][\mathrm{j}][\mathrm{k}]\boldsymbol{u}[\mathrm{i}][\mathrm{j}][\mathrm{k}] &= \sum_l f[\mathrm{i}][\mathrm{j}][\mathrm{k}][\mathrm{l}]\boldsymbol{c}_l, \end{aligned} \tag{3.12}$$

where indices i,j,k are x, y, z indices of the current grid cell (node). It is advisable to unroll the summation by index l. Most velocity components are zero, and one does not want to waste CPU time by summing zeroes. For example, for the D1Q3 velocity set we have:

$$\rho = f[0] + f[1] + f[2],\ u_x = (f[1] - f[2])/\rho, \tag{3.13}$$

where we have dropped the current cell indices. For the D2Q9 velocity set as defined in eq. (**??**), the velocity computation may be implemented like

$$\begin{aligned} \rho &= f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8, \\ u_x &= \frac{f_1 - f_3 + f_5 - f_6 - f_7 + f_8}{\rho}, \\ u_y &= \frac{f_2 - f_4 + f_5 + f_6 - f_7 - f_8}{\rho}. \end{aligned} \tag{3.14}$$

THINK (OS): I prefer e.g. $(f_1 + f_5 + f_8) - (f_3 + f_6 + f_7)$

Equilibrium

In the same way it is possible to accelerate the equilibrium computation in eq. (3.4). One needs to unroll the loop over populations.

As well, it is strongly recommended to replace the inverse powers of the speed of sound, c_s^{-2} and c_s^{-4} by new variables. For the standard lattices, these values happen to be 3 and 9. We recommend, however, to keep distinct variables since the speed of sound for higher-order lattices is different and any code should be written in a suffi-

[9] For example, for a moderate simulation domain of $100 \times 100 \times 100$ lattice sites and the D3Q19 velocity set, one copy of populations f_i requires about 300 MByte memory, where double precision (two bytes per variable) have been assumed.

ciently flexible manner to facilitate future generalisations. Anyway, the equilibrium computation can be written without a single expensive division.

Collision

From eq. (3.9) we see that the terms $(1 - \Delta t/\tau)$ and $\Delta t/\tau$ have to be computed for each time step, lattice site and velocity direction. By defining two additional constant variables, *e.g.* $\omega = \Delta t/\tau$ and $\omega' = 1 - \omega$, which are computed only once, any expensive division operations can be removed from collision:

$$f_i^\star(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t)\omega' + f_i^{\text{eq}}(\boldsymbol{x}, t)\omega. \tag{3.15}$$

This can save a significant amount of otherwise wasted CPU time.

The above considerations only show the 'common sense' implementation and common practices. For those, who wants to go through deeper issues as parallelisation of a code, chapter 13 provides detailed explanations. As well one can always check the simplest book code implementation available ....

3.3 Discretisation in velocity space

In this section we will derive the velocity discretisation of the Boltzmann equation. One problem mentioned above is that the particle distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ spans the seven-dimensional space defined by the coordinates x, y, z, ξ_x, ξ_y, ξ_z and t. Solving equations in this high-dimensional space is computationally exhausting and requires large-scale computers (*e.g.* clusters) and large-scale programming efforts to develop simulation code.

However, this effort is usually not necessary. As we found in section 1.3.4, the *moments* of the Boltzmann equation give the correct equations for mass, momentum and energy conservation. Thus, much of the underlying physics is not relevant if we are only interested in getting the correct *macroscopic* behaviour.

For example, the moments are nothing else than weighted integrals of f in velocity space. It is obvious that there is a vast number of different functions whose integrals are identical to those of f. As we will see shortly, there are approaches to simplify the continuous Boltzmann equation without sacrificing the macroscopic (*i.e.* moment-based) behaviour. The discretisation of velocity space allows us to reduce the continuous 3D velocity space to a small number of discrete velocities without compromising the validity of the macroscopic equations.

The velocity discretisation can be performed using either the Mach number expansion[2] or the Hermite series expansion [3] of the equilibrium distribution function. Both approaches give the same form of the equilibrium on the Navier-Stokes level. Although the Mach expansion approach is simpler, we will follow the Hermite polynomial approach as it provides a strong mathematical basis. As a

bonus, it also provides not only a variety of suitable discrete velocity sets but it can also correctly restore equations beyond the Navier-Stokes level, i.e. Burnett-type equations [3].

Despite the rather heavy mathematical background of this section, the main idea is not too difficult:

In contrast to the unknown distribution function f, the *equilibrium* distribution function f^{eq} is a known function on exponential form. f^{eq} can consequently be expressed through the exponential *weight function* (or *generating function*) of Hermite polynomials. Additionally, the mass and momentum moments can be represented as integrals of f^{eq} multiplied with Hermite polynomials.

These features let us apply two clever techniques in succession. First, we can express f^{eq} in a reduced form through a truncated sum of Hermite polynomials, while retaining the correct mass and momentum moment integrals. Secondly, the moment integrals are then on a form where they can be evaluated exactly as a *discrete sum* over the polynomial integrand evaluated at specific points $\boldsymbol{\xi}_i$, called *abscissae*, in velocity space. This allows us to represent f^{eq} as being discrete in velocity space instead of continuous. These techniques can also be apply the particle distribution f itself.

In other words, one does not need to retain all the underlying mesoscopic information in order to retain correct macroscopic conservation laws. A simplified equilibrium f^{eq} and a discrete velocity space is sufficient.

This section deals with discretisation in velocity space. Later in Section 3.4, the simultaneous discretisation of space and time will be performed *via* integration along characteristics, which is a powerful technique to solve first-order hyperbolic partial differential equations.

3.3.1 Non-dimensionalisation

Before we start with the Hermite series expansion, we will *non-dimensionalise* the governing equations to simplify the subsequent steps. (Non-dimensionalisation is also useful when implementing computational models and for relating physical laws in form of mathematical equations to the real world, as discussed in Chapter 7.)

Let us recall the Boltzmann equation in continuous velocity space:

$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} + \frac{F_\alpha}{\rho} \frac{\partial f}{\partial \xi_\alpha} = \Omega(f), \tag{3.16}$$

where $\Omega(f)$ is the collision operator. Its specific form is not important here.[10]

The Boltzmann equation describes the evolution of the distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$, *i.e.* the density of particles with velocity $\boldsymbol{\xi}$ at position $\boldsymbol{x}$ and time t. In a force-free, homogeneous and steady situation, the left-hand-side of eq. (3.16) vanishes and the solution of the Boltzmann equation becomes the equilibrium distribution function f^{eq}. As we found in Section 1.3.2, f^{eq} can be written in terms of the macroscopic quantities of density ρ, fluid velocity $\boldsymbol{u}$ and temperature T as

$$f^{\mathrm{eq}}(\rho, \boldsymbol{u}, T, \boldsymbol{\xi}) = \frac{\rho}{(2\pi RT)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2RT)}, \tag{3.17}$$

where d is the number of spatial dimensions and the gas constant $R = k_{\mathrm{B}}/m$ is given by the Boltzmann constant k_{B} and the particle mass m.

Physical phenomena occur on certain space and time scales. For example, the wavelengths of tsunamis and electromagnetic waves in the visible spectrum are on the order of hundreds of kilometers and nanometers, respectively, and their propagation speeds are a few hundred meters per second and the speed of light. One can therefore classify these phenomena by identifying their characteristic scales. As covered in more detail in section 1.2, we can analyse the properties of a fluid in terms of its characteristic length ℓ, velocity V and density ρ_0. The characteristic time scale is then given by $t_0 = \ell/V$.

Using stars to denote non-dimensionalised quantities, we first introduce the non-dimensional derivatives:

$$\frac{\partial}{\partial t^\star} = \frac{\ell}{V}\frac{\partial}{\partial t}, \quad \frac{\partial}{\partial x^\star} = \ell\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial \xi^\star} = V\frac{\partial}{\partial \xi}. \tag{3.18}$$

This leads to the non-dimensional continuous Boltzmann equation:

$$\frac{\partial f^\star}{\partial t^\star} + \xi_\alpha^\star \frac{\partial f^\star}{\partial x_\alpha^\star} + \frac{F_\alpha^\star}{\rho^\star}\frac{\partial f^\star}{\partial \xi_\alpha^\star} = \Omega^\star(f^\star), \tag{3.19}$$

where $f^\star = fV^d/\rho_0$, $\boldsymbol{F}^\star = \boldsymbol{F}\ell/(\rho_0 V^2)$, $\rho^\star = \rho/\rho_0$ and $\Omega^\star = \Omega\ell V^2/\rho_0$. The non-dimensional equilibrium distribution function reads

$$f^{\mathrm{eq}\star} = \frac{\rho^\star}{(2\pi\theta^\star)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}^\star-\boldsymbol{u}^\star)^2/(2\theta^\star)}, \tag{3.20}$$

where the non-dimensional temperature is $\theta^\star = RT/V^2$.

Exercise 3.1. Check that the above expressions are correct by showing that all quantities with a star are non-dimensional.

We will hereafter omit the symbol $\star$ indicating non-dimensionalisation in order to keep the notation compact. Note that when the Boltzmann equation is referred to

[10] As we will see later in chapter 10, the collision operator can have different forms all of which locally conserve the moments (mass, momentum and energy) and, thus, yield the correct macroscopic behaviour.

from now on in this chapter, the non-dimensional version is implied unless otherwise specified.

As mentioned in the introduction, the Hermite series expansion will be performed in the force-free case ($\boldsymbol{F} = 0$) to reduce the level of complexity. Forces are thoroughly discussed in chapter 6.

In summary, we obtain the non-dimensional continuous and force-free Boltzmann equation as

$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} = \Omega(f), \tag{3.21}$$

and the non-dimensional equilibrium distribution as

$$f^{\mathrm{eq}}(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}) = \frac{\rho}{(2\pi\theta)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2\theta)}. \tag{3.22}$$

3.3.2 Conservation laws

The conservation laws have already been discussed in section 1.3.3. We saw that the collision operator conserves certain moments of the distribution function. This conservation implies that the moments of the equilibrium distribution function f^{eq} and the particle distribution function f coincide:

$$\begin{aligned}
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right)\, \mathrm{d}^3\xi &&= \rho(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\boldsymbol{\xi}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \boldsymbol{\xi}\, \mathrm{d}^3\xi &&= \rho\boldsymbol{u}(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\frac{|\boldsymbol{\xi}|^2}{2}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \frac{|\boldsymbol{\xi}|^2}{2}\, \mathrm{d}^3\xi &&= \rho E(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\frac{|\boldsymbol{\xi}-\boldsymbol{u}|^2}{2}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \frac{|\boldsymbol{\xi}-\boldsymbol{u}|^2}{2}\, \mathrm{d}^3\xi &&= \rho e(\boldsymbol{x}, t).
\end{aligned} \tag{3.23}$$

The dependency on space and time in f^{eq} enters only through $\rho(\boldsymbol{x}, t)$, $\boldsymbol{u}(\boldsymbol{x}, t)$ and $\theta(\boldsymbol{x}, t)$.

As we see, all conserved quantities on the right-hand-side of eq. (3.23) can be obtained as integrals of f or f^{eq} in velocity space. The basic idea of the Hermite series expansion is to turn the continuous integrals into discrete sums evaluated at certain points in velocity space (*i.e.* for specific values of $\boldsymbol{\xi}$). These sums can be found exactly for f^{eq} as it has the same shape as the weight function of Hermite polynomials. The velocity discretisation of f^{eq} can also be extrapolated to f while still satisfying the conservation laws.

We will now discuss the properties of Hermite polynomials.

3.3.3 Hermite polynomials

Among the infinite number of different functions and polynomials, one particularly interesting set of polynomials used for the discretisation of integrals are the *Hermite polynomials* (HPs). They naturally appear in the quantum-mechanical wave functions for harmonic potentials. Before we will make use of their integral discretisation properties, we will first characterise the HPs and give some examples.

The derivation of the LBE (and also working with the Navier-Stokes equation) requires some knowledge of tensor notation which can be challenging for novices. Readers without little experience with tensors should take some time to learn the basics of tensor calculus [4]. TODO (AK): Provide full bibliographic information for this reference.

Definition and construction of Hermite polynomials

One-dimensional HPs are polynomials which can be obtained from the *weight function* (also called the *generating function*),

$$\omega(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2}. \tag{3.24}$$

This weight function $\omega(x)$ allows us to construct the 1D HP of n-th order as

$$H^{(n)}(x) = (-1)^n \frac{1}{\omega(x)} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \omega(x), \tag{3.25}$$

where $n \geq 0$ is an integer. The first six of these polynomials are

$$\begin{aligned} H^{(0)}(x) &= 1, & H^{(1)}(x) &= x, \\ H^{(2)}(x) &= x^2 - 1, & H^{(3)}(x) &= x^3 - 3x, \\ H^{(4)}(x) &= x^4 - 6x^2 + 3, & H^{(5)}(x) &= x^5 - 10x^3 + 15x. \end{aligned} \tag{3.26}$$

Exercise 3.2. Show that eq. (3.24) and eq. (3.25) lead to the Hermite polynomials in eq. (3.26).

The 1D definition of the HPs in eq. (3.25) can be extended to d spatial dimensions [5, 3]:

$$\boldsymbol{H}^{(n)}(\boldsymbol{x}) = (-1)^n \frac{1}{\omega(\boldsymbol{x})} \boldsymbol{\nabla}^{(n)} \omega(\boldsymbol{x}), \quad \omega(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}} \mathrm{e}^{-\boldsymbol{x}^2/2}. \tag{3.27}$$

This notation requires several comments. Both $\boldsymbol{H}^{(n)}$ and $\boldsymbol{\nabla}^{(n)}$ are tensors of rank n, *i.e.* we can represent $\boldsymbol{H}^{(n)}$ and $\boldsymbol{\nabla}^{(n)}$ by their d^n components $H^{(n)}_{\alpha_1 \ldots \alpha_n}$ and $\nabla^{(n)}_{\alpha_1 \ldots \alpha_n}$, respectively, where $\{\alpha_1, \ldots, \alpha_n\}$ are n indices running from 1 to d each. $\nabla^{(n)}_{\alpha_1 \ldots \alpha_n}$ is a short notation for n consecutive spatial derivatives:

$$\nabla^{(n)}_{\alpha_1\ldots\alpha_n} = \frac{\partial}{\partial x_{\alpha_1}} \cdots \frac{\partial}{\partial x_{\alpha_n}}. \tag{3.28}$$

Note that the derivatives are symmetric upon permutation of the indices if we assume that derivatives commute,[11] *e.g.* $\nabla^{(3)}_{xxy} = \nabla^{(3)}_{xyx} = \nabla^{(3)}_{yxx}$.

We are mostly interested in the cases $d = 2$ or $d = 3$ where we write $\alpha_1, \ldots, \alpha_n \in \{x, y\}$ or $\{x, y, z\}$, respectively.

Example 3.1. To make this clearer, we explicitly write down the 2D ($d = 2$) HPs up to second order ($n = 0, 1, 2$). Using

$$\nabla^{(2)}_{xx} = \frac{\partial}{\partial x}\frac{\partial}{\partial x}, \quad \nabla^{(2)}_{xy} = \frac{\partial}{\partial x}\frac{\partial}{\partial y}, \quad \nabla^{(2)}_{yx} = \frac{\partial}{\partial y}\frac{\partial}{\partial x}, \quad \nabla^{(2)}_{yy} = \frac{\partial}{\partial y}\frac{\partial}{\partial y} \tag{3.29}$$

we find

$$H^{(0)} = 1, \tag{3.30}$$

for $n = 0$,

$$\begin{aligned} H^{(1)}_x &= -\frac{1}{\mathrm{e}^{-(x^2+y^2)/2}}\partial_x \mathrm{e}^{-(x^2+y^2)/2} = x, \\ H^{(1)}_y &= -\frac{1}{\mathrm{e}^{-(x^2+y^2)/2}}\partial_y \mathrm{e}^{-(x^2+y^2)/2} = y, \end{aligned} \tag{3.31}$$

for $n = 1$ and

$$\begin{aligned} H^{(2)}_{xx} &= \frac{1}{\mathrm{e}^{-(x^2+y^2)/2}}\partial_x\partial_x \mathrm{e}^{-(x^2+y^2)/2} = x^2 - 1, \\ H^{(2)}_{xy} = H^{(2)}_{yx} &= \frac{1}{\mathrm{e}^{-(x^2+y^2)/2}}\partial_x\partial_y \mathrm{e}^{-(x^2+y^2)/2} = xy, \\ H^{(2)}_{yy} &= \frac{1}{\mathrm{e}^{-(x^2+y^2)/2}}\partial_y\partial_y \mathrm{e}^{-(x^2+y^2)/2} = y^2 - 1 \end{aligned} \tag{3.32}$$

for $n = 2$.

Exercise 3.3. Construct the eight third-order HPs $H^{(3)}_{\alpha_1\alpha_2\alpha_3}$ for $d = 2$.

Orthogonality and series expansion

So far we have only discussed the definition and construction of the HPs. Let us now turn our attention to their mathematical properties which we will soon make use of. One of the nice features of the Hermite polynomials is their orthogonality. It is possible to express the equilibrium and distribution functions f^{eq} and f as a the series of Hermite polynomials, where each series coefficient is independent of the others due to this orthogonality.

It will turn out that these Hermite polynomial series can be truncated at the second order as this is sufficient to represent the mass and momentum conservation and

[11] This is the case for sufficiently smooth functions.

to recover the Navier-Stokes equation. Retaining additional terms of the Hermite series will provide a finer description of the underlying physics without affecting the coefficients or physics obtained before. Thus we may additionally recover an energy conservation equation, and we may furthermore obtain higher-order 'correction terms' for the standard continuum equations [3].

The 1D HPs are *orthogonal* with the respect to $\omega(x)$:

$$\int_{-\infty}^{\infty} \omega(x) H^{(n)}(x) H^{(m)}(x)\,\mathrm{d}x = n!\delta^{(2)}_{nm}, \tag{3.33}$$

where $\delta^{(2)}_{nm}$ is the Kronecker symbol.

The orthogonality of the HPs can be generalised to d dimensions:

$$\int \omega(\boldsymbol{x}) H^{(n)}_{\boldsymbol{\alpha}}(\boldsymbol{x}) H^{(m)}_{\boldsymbol{\beta}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \prod_{i=1}^{d} n_i!\,\delta^{(2)}_{nm}\delta^{(n+m)}_{\boldsymbol{\alpha\beta}}, \tag{3.34}$$

where $\delta^{(n+m)}_{\boldsymbol{\alpha\beta}}$ is a generalised Kronecker symbol which is 1 only if $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is a permutation of $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$ and 0 otherwise. For example, (x, x, z, y) is a permutation of (y, x, z, x) but not of (x, y, x, y). n_x, n_y and n_z are the numbers of occurrences of x, y and z in $\boldsymbol{\alpha}$. For example, for $\boldsymbol{\alpha} = (x, x, y)$ one gets $n_x = 2$, $n_y = 1$, $n_z = 0$. For $d = 3$ in particular, Eq. (3.34) reads:

$$\int \omega(\boldsymbol{x}) H^{(n)}_{\boldsymbol{\alpha}}(\boldsymbol{x}) H^{(m)}_{\boldsymbol{\beta}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = n_x!\,n_y!\,n_z!\,\delta^{(2)}_{mn}\delta^{(n+m)}_{\boldsymbol{\alpha\beta}}, \tag{3.35}$$

Example 3.2. We consider some concrete examples of eq. (3.35). The following integrals vanish because the indices $\boldsymbol{\alpha}$ are no permutations of $\boldsymbol{\beta}$, although the order of both HPs is identical ($m = n$):

$$\begin{aligned} \int \omega(\boldsymbol{x}) H^{(1)}_{x}(\boldsymbol{x}) H^{(1)}_{y}(\boldsymbol{x})\,\mathrm{d}^3x &= 0, \\ \int \omega(\boldsymbol{x}) H^{(2)}_{xy}(\boldsymbol{x}) H^{(2)}_{xx}(\boldsymbol{x})\,\mathrm{d}^3x &= 0. \end{aligned} \tag{3.36}$$

The integral

$$\int \omega(\boldsymbol{x}) H^{(1)}_{x}(\boldsymbol{x}) H^{(2)}_{xy}(\boldsymbol{x})\,\mathrm{d}^3x = 0 \tag{3.37}$$

vanishes because the orders are not identical: $m \neq n$. The remaining integrals

$$\begin{aligned} \int \omega(\boldsymbol{x}) H^{(1)}_{x}(\boldsymbol{x}) H^{(1)}_{x}(\boldsymbol{x})\,\mathrm{d}^3x &= 1! = 1, \\ \int \omega(\boldsymbol{x}) H^{(3)}_{xxx}(\boldsymbol{x}) H^{(3)}_{xxx}(\boldsymbol{x})\,\mathrm{d}^3x &= 3! = 6, \\ \int \omega(\boldsymbol{x}) H^{(3)}_{xxy}(\boldsymbol{x}) H^{(3)}_{xyx}(\boldsymbol{x})\,\mathrm{d}^3x &= 2!\,1! = 2 \end{aligned} \tag{3.38}$$

do not vanish since i) $m = n$ and ii) $\boldsymbol{\alpha}$ is a permutation of $\boldsymbol{\beta}$.

Exercise 3.4. Show that eq. (3.34) reduces to eq. (3.33) for $d = 1$.

The HPs form a complete basis in $\mathbb{R}$, *i.e.* any good enough[12] continuous function $f(x) \in \mathbb{R}$ can be represented as a series of Hermite polynomials:

$$f(x) = \omega(x) \sum_{n=0}^{\infty} \frac{1}{n!} a^{(n)} H^{(n)}(x), \quad a^{(n)} = \int f(x) H^{(n)}(x) \, \mathrm{d}x. \tag{3.39}$$

This can be extended to d dimensions:

$$f(\boldsymbol{x}) = \omega(\boldsymbol{x}) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}^{(n)} \cdot \boldsymbol{H}^{(n)}(\boldsymbol{x}), \quad \boldsymbol{a}^{(n)} = \int f(\boldsymbol{x}) \boldsymbol{H}^{(n)}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \tag{3.40}$$

The expansion coefficients $\boldsymbol{a}^{(n)}$ are also tensors of rank n, and the dot product $\boldsymbol{a}^{(n)} \cdot \boldsymbol{H}^{(n)}$ is defined as the full contraction $a^{(n)}_{\alpha_1 \ldots \alpha_n} H^{(n)}_{\alpha_1 \ldots \alpha_n}$, *i.e.* the summation over all possible indices.

We have now the necessary apparatus available to perform the Hermite series expansion of the equilibrium distribution function. This will then lead to the desired velocity discretisation.

3.3.4 Hermite series expansion of the equilibrium distribution

Let us now apply the Hermite series expansion in eq. (3.40) to the equilibrium distribution function f^{eq} in $\boldsymbol{\xi}$-space:

$$\begin{aligned} f^{\mathrm{eq}}(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}) &= \omega(\boldsymbol{\xi}) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}^{(n),\mathrm{eq}}(\rho, \boldsymbol{u}, \theta) \cdot \boldsymbol{H}^{(n)}(\boldsymbol{\xi}), \\ \boldsymbol{a}^{(n),\mathrm{eq}}(\rho, \boldsymbol{u}, \theta) &= \int f^{\mathrm{eq}}(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}) \, \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi}. \end{aligned} \tag{3.41}$$

It is important to note that the equilibrium distribution function $f^{\mathrm{eq}}(\boldsymbol{\xi})$ has an equivalent form to the Hermite polynomials' weight function $\omega(\boldsymbol{\xi})$ in eq. (3.27):

$$f^{\mathrm{eq}}(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}) = \frac{\rho}{(2\pi\theta)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2\theta)} = \frac{\rho}{\theta^{d/2}} \omega\left(\frac{\boldsymbol{\xi} - \boldsymbol{u}}{\sqrt{\theta}}\right). \tag{3.42}$$

[12] The goal of this book is to give practical aspects of the derivation and usage of the LBM rather than a rigorous mathematical theory for some assumptions used. However, for interested readers we recommend the book of N. Wiener for the rigorous proof [6].

This crucial relation can be used to calculate the series coefficients:

$$\boldsymbol{a}^{(n),\mathrm{eq}} = \frac{\rho}{\theta^{d/2}} \int \omega\left(\frac{\boldsymbol{\xi} - \boldsymbol{u}}{\sqrt{\theta}}\right) \boldsymbol{H}^{(n)}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi}. \tag{3.43}$$

The substitution $\boldsymbol{\eta} = (\boldsymbol{\xi} - \boldsymbol{u})/\sqrt{\theta}$ yields

$$\boldsymbol{a}^{(n),\mathrm{eq}} = \rho \int \omega(\boldsymbol{\eta}) \boldsymbol{H}^{(n)}\left(\sqrt{\theta}\boldsymbol{\eta} + \boldsymbol{u}\right) \mathrm{d}\boldsymbol{\eta}. \tag{3.44}$$

These integrals can be directly computed, for example with the help of mathematical software packages:

$$\begin{aligned}
a^{(0),\mathrm{eq}} &= \rho, \\
a_\alpha^{(1),\mathrm{eq}} &= \rho u_\alpha, \\
a_{\alpha\beta}^{(2),\mathrm{eq}} &= \rho\left(u_\alpha u_\beta + (\theta - 1)\delta_{\alpha\beta}\right), \\
a_{\alpha\beta\gamma}^{(3),\mathrm{eq}} &= \rho\left(u_\alpha u_\beta u_\gamma + (\theta - 1)\left(\delta_{\alpha\beta}u_\gamma + \delta_{\beta\gamma}u_\alpha + \delta_{\gamma\alpha}u_\beta\right)\right).
\end{aligned} \tag{3.45}$$

A close look at eq. (3.45) reveals that the coefficients in the Hermite series expansion of the equilibrium distribution function f^{eq} are related to the conserved moments; the first three coefficients are connected to the density, momentum and energy. At the same time, it turns out that the conserved quantities can also be represented by the Hermite series expansion coefficients of the d-dimensional particle distribution function f:

$$\begin{aligned}
a^{(0),\mathrm{eq}} &= \int f^{\mathrm{eq}}\,\mathrm{d}^d\xi \quad &&= \rho \quad &&= \int f\,\mathrm{d}^d\xi \quad &&= a^{(0)}, \\
a_\alpha^{(1),\mathrm{eq}} &= \int f^{\mathrm{eq}}\xi_\alpha\,\mathrm{d}^d\xi \quad &&= \rho u_\alpha &&= \int f\xi_\alpha\,\mathrm{d}^d\xi \quad &&= a_\alpha^{(1)}, \\
\frac{a_{\alpha\alpha}^{(2),\mathrm{eq}} + \rho d}{2} &= \int f^{\mathrm{eq}}\frac{|\boldsymbol{\xi}|^2}{2}\,\mathrm{d}^d\xi = \rho E \ &&= &&\int f\frac{|\boldsymbol{\xi}|^2}{2}\,\mathrm{d}^d\xi &&= \frac{a_{\alpha\alpha}^{(2)} + \rho d}{2}.
\end{aligned} \tag{3.46}$$

This is one of the reasons why the Hermite series expansion is used for the Boltzmann equation; the series coefficients are directly connected to the conserved moments or even coincide with them. Only the first three expansion coefficients ($n = 0, 1, 2$) are required to fulfill the conservation laws and represent the macroscopic equations, although some authors have indicated that the inclusion of higher expansion terms can add additional numerical stability and numerical accuracy of the macroscopic equations simulation (*cf.* chapter 10) [7, 8, 9, 10].

In order to reproduce the relevant physics, *i.e.* to satisfy the conservation laws on the *macroscopic* level, one does not need to consider the full microscopic equilibrium and particle distribution functions. Instead, the first three terms of

the Hermite series expansion are sufficient to recover the macroscopic laws of interest. This allows for a significant reduction of numerical effort when the Boltzmann equation is used to solve hydrodynamical problems.

Limiting the expansion to the N-th order, the equilibrium and particle distribution functions can be represented as

$$\begin{aligned} f^{\mathrm{eq}}(\boldsymbol{\xi}) &\approx \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}^{(n),\mathrm{eq}} \cdot \boldsymbol{H}^{(n)}(\boldsymbol{\xi}), \\ f(\boldsymbol{\xi}) &\approx \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}^{(n)} \cdot \boldsymbol{H}^{(n)}(\boldsymbol{\xi}), \end{aligned} \tag{3.47}$$

where we have only denoted the $\boldsymbol{\xi}$-dependence.

We can now explicitly write down the equilibrium distribution function approximation up to the third moment, *i.e.* up to the second order in $\boldsymbol{\xi}$ ($N = 2$):

$$\begin{aligned} f^{\mathrm{eq}}(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}) &\approx \omega(\boldsymbol{\xi})\rho \left[1 + \xi_\alpha u_\alpha + \left(u_\alpha u_\beta + (\theta - 1)\delta_{\alpha\beta}\right)\left(\xi_\alpha \xi_\beta - \delta_{\alpha\beta}\right)\right] \\ &= \omega(\boldsymbol{\xi})\rho Q(\boldsymbol{u}, \theta, \boldsymbol{\xi}), \end{aligned} \tag{3.48}$$

where $Q(\boldsymbol{u}, \theta, \boldsymbol{\xi})$ is a multi-dimensional polynomial in $\boldsymbol{\xi}$. Note that this form of the equilibrium function conserves first three moments.

Exercise 3.5. Show the validity of eq. (3.48).

At this point we should consider the Mach number expansion. If one expands the equilibrium distribution function in eq. (3.22) up to second order in $\boldsymbol{u}$, then one would also obtain eq. (3.48). However, at the next order (corresponding to energy conservation) the Hermite polynomial expansion and the Mach number expansion approaches would give different results. Due to the orthogonality between Hermite polynomials, the Hermite polynomial approach does not mix the lower order moments related to the Navier-Stokes equation with the higher order moments related to the energy equation and beyond. This is not the case with the low Mach number approach [11]. Thus, the Hermite polynomial approach is generally preferable.

Moreover, we will shortly see that the Hermite polynomial approach readily provides discrete velocity sets: In the next step we can finally discretise the velocity space by replacing the continuous $\boldsymbol{\xi}$ by suitable discrete velocity sets $\{\boldsymbol{\xi}_i\}$.

3.3.5 Discretisation of the equilibrium distribution function

We have seen that the Hermite series expansion is a suitable expansion method since the equilibrium distribution function $f^{\mathrm{eq}}(\boldsymbol{\xi})$ has the same form as the Hermite

weight function $\omega(\boldsymbol{\xi})$. But there is also another compelling reason to use Hermite polynomials: As mentioned earlier, it is possible to calculate integrals of certain functions by taking integral function values at a small number of discrete points, the so-called *abscissae*.

Let us take a 1D polynomial $P^{(N)}(x)$ of order N as example. As explained in more detail in appendix A.4, the integral $\int \omega(x)P^{(N)}(x)\,\mathrm{d}x$ can be calculated exactly by considering integral function values in certain points x_i. This rule is called the *Gauss-Hermite quadrature rule* and assumes the form

$$\int \omega(x)P^{(N)}(x)\,\mathrm{d}x = \sum_{i=1}^{n} w_i P^{(N)}(x_i) \tag{3.49}$$

for polynomials if the n values x_i are the roots of the HP $H^{(n)}(x)$, *i.e.* $H^{(n)}(x_i) = 0$ and $N = 2n - 1$. In order to exactly integrate a polynomial of order N, one requires at least $n = (N + 1)/2$ abscissae x_i and associated weights w_i. An expression for the latter is given in eq. (A.30). Higher-order polynomials obviously require a larger number of abscissae and higher-order HPs.

The generalisation to multiple dimensions reads as:

$$\int \omega(\boldsymbol{x})P^{(N)}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \sum_{i=1}^{n} w_i P^{(N)}(\boldsymbol{x}_i), \tag{3.50}$$

where each of the component of the multidimensional point $\boldsymbol{x}_i$, *i.e.* $x_{i\alpha}$ with $\alpha = 1, \ldots, d$, is a root of the one-dimensional Hermite polynomial $H^{(n)}(x_{i\alpha}) = 0$ (see Appendix A.4). This comes from the factorisation of multidimensional Hermite polynomials.

The Gauss-Hermite quadrature rule can be used to calculate moments and coefficients of the Hermite series expansion. Let us take a look at the definition of the coefficient for the equilibrium function, eq. (3.44):

$$\boldsymbol{a}^{(n),\mathrm{eq}} = \rho \int \omega(\boldsymbol{\eta})\boldsymbol{H}^{(n)}\left(\sqrt{\theta}\boldsymbol{\eta} + \boldsymbol{u}\right)\mathrm{d}\boldsymbol{\eta} = \rho \int \omega(\boldsymbol{\eta})\boldsymbol{P}^{(n)}(\boldsymbol{\eta})\,\mathrm{d}\boldsymbol{\eta}, \tag{3.51}$$

where $\boldsymbol{P}^{(n)}(\boldsymbol{\eta})$ is a tensor-valued polynomial of rank n in $\boldsymbol{\eta}$. As one can see, the expression $\int \omega(\boldsymbol{\eta})\boldsymbol{P}^{(n)}(\boldsymbol{\eta})\,\mathrm{d}\boldsymbol{\eta}$ can be discretised using the Gauss-Hermite quadrature rule. However, it takes an additional effort to calculate the polynomial $\boldsymbol{P}^{(n)}(\boldsymbol{\eta})$ from the Hermite polynomial $\boldsymbol{H}^{(n)}(\sqrt{\theta}\boldsymbol{\eta} + \boldsymbol{u})$. Instead, we follow a simpler route and use the truncated series expansion of the equilibrium distribution function f^{eq}, which yields the same macroscopic properties.

From eq. (3.41) and eq. (3.48) we first obtain

$$\boldsymbol{a}^{(n),\mathrm{eq}} = \int f^{\mathrm{eq}}(\boldsymbol{\xi})\boldsymbol{H}^{(n)}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} = \rho \int \omega(\boldsymbol{\xi})Q(\boldsymbol{\xi})\boldsymbol{H}^{(n)}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi}, \tag{3.52}$$

where we have again only denoted the $\boldsymbol{\xi}$-dependence. Now, we take the composed polynomial $\boldsymbol{R}(\boldsymbol{\xi}) = Q(\boldsymbol{\xi})\boldsymbol{H}^{(n)}(\boldsymbol{\xi})$ and apply the Gauss-Hermite quadrature rule to it:

$$\boldsymbol{a}^{(n),\mathrm{eq}} = \rho \int \omega(\boldsymbol{\xi})\boldsymbol{R}(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi} = \rho \sum_{i=1}^{n} w_i \boldsymbol{R}(\boldsymbol{\xi}_i) = \rho \sum_{i=1}^{n} w_i Q(\boldsymbol{\xi}_i)\boldsymbol{H}^{(n)}(\boldsymbol{\xi}_i). \tag{3.53}$$

This is the *discretised* Hermite series expansion where n is the required number of abscissae.

Let us turn our attention to the abscissae $\boldsymbol{\xi}_i$. In order to fullfil all relevant conservation laws, one has to ensure that the polynomial with the highest-occuring degree can be correctly integrated. This highest-order polynomial, seen in Eq. (3.46), is related to the energy and is connected to the second-order HP $\boldsymbol{H}^{(2)}(\boldsymbol{\xi})$. If we further assume that the polynomial $Q(\boldsymbol{\xi})$ is of second order as in Eq. (3.48), the polynomial $\boldsymbol{R}(\boldsymbol{\xi}) = Q(\boldsymbol{\xi})\boldsymbol{H}^{(2)}(\boldsymbol{\xi})$ is of fourth order. Such a polynomial ($N = 4$) can be exactly calculated as a sum using the abscissae $\boldsymbol{\xi}_i$ found from the roots of $H^{(3)}(\xi_{i\alpha})$ (remember that $n = 3$ is sufficient to integrate exactly a polynomial of order up to $N = 2n - 1 = 5$). The actual choice of the abscissae and details of the velocity set construction are presented in appendix A.4. The most common LB velocity sets are thoroughly discussed in section 3.3.7.

We have now successfully passed the 'heavy' mathematical part of this section. Only a few more minor things have to be considered before we have completed the velocity discretisation.

Let us define the n quantities $f_i^{\mathrm{eq}}(\boldsymbol{x},t) = w_i\rho(\boldsymbol{x},t)Q(\boldsymbol{u}(\boldsymbol{x},t),\theta(\boldsymbol{x},t),\boldsymbol{\xi}_i)$ as the equilibrium distribution function related to the velocity (direction) $\boldsymbol{\xi}_i$. Instead of having a continuous function $f^{\mathrm{eq}}(\boldsymbol{\xi})$, we only consider a finite set of quantities $f_i^{\mathrm{eq}} = f^{\mathrm{eq}}(\boldsymbol{\xi}_i)$ with the following form, *cf.* eq. (3.48):

$$f_i^{\mathrm{eq}} = w_i\rho\left[1 + \xi_{i\alpha}u_\alpha + \frac{1}{2}\left(u_\alpha u_\beta + (\theta - 1)\delta_{\alpha\beta}\right)\left(\xi_{i\alpha}\xi_{i\beta} - \delta_{\alpha\beta}\right)\right]. \tag{3.54}$$

Note that the set of the discrete-$\boldsymbol{\xi}$ functions $\{f^{\mathrm{eq}}(\boldsymbol{\xi}_i)\}$ is equivalent to the original continuous-$\boldsymbol{\xi}$ equilibrium function $f^{\mathrm{eq}}(\boldsymbol{\xi})$ in terms of the conservation laws, *i.e.* the first three moments (density, momentum and energy) are conserved.

Note that space and time have not yet been discretised. This means that $f_i^{\mathrm{eq}}(\boldsymbol{x},t)$ is defined at every point in space and time, with the dependence on $\boldsymbol{x}$ and t entering *via* the *continuous* moments $\rho(\boldsymbol{x},t)$, $\boldsymbol{u}(\boldsymbol{x},t)$ and $\theta(\boldsymbol{x},t)$. Discretisation in space and time will be discussed in section 3.4.

Further simplifications of eq. (3.54) are possible:

Isothermal assumption From the Boltzmann equation one can obtain a whole hierarchy of equations including the continuity equation, the Navier-Stokes equation and the energy equation [12, 13]. However, the main focus in this book is on hydrodynamic applications. The energy equation can alternatively be simulated as a separate equation (usually as an advection-diffusion equation) which might be coupled through a force (Boussinesq approximation, *cf.* chapter 8) or through the velocity obtained from the Navier-Stokes equation. The *isothermal*

assumption implies $\theta = 1$ which removes the temperature from the equilibrium distribution in eq. (3.54).

Velocity rescaling Many abscissae $\boldsymbol{\xi}_i$ reported in appendix A.4 contain unhandy factors of $\sqrt{3}$. The natural simplification is to introduce a new particle velocity:

$$\boldsymbol{c}_i = \frac{\boldsymbol{\xi}_i}{\sqrt{3}}. \tag{3.55}$$

Note that this also leads to a redefinition of the macroscopic velocity: $\boldsymbol{u} \to \boldsymbol{u}/\sqrt{3}$. NOTE (EMV): What does this mean? Macroscopic velocity is what it is; a flow of 1 m/s must still be at 1 m/s whatever we do here.

With the above simplifications, the discrete equilibrium distribution finally becomes

$$f_i^{\mathrm{eq}} = w_i \rho \left(1 + \frac{c_{i\alpha} u_\alpha}{c_\mathrm{s}^2} + \frac{u_\alpha u_\beta \left(c_{i\alpha} c_{i\beta} - c_\mathrm{s}^2 \delta_{\alpha\beta} \right)}{2 c_\mathrm{s}^4} \right), \tag{3.56}$$

where $c_\mathrm{s} = 1/\sqrt{3}$ is the so-called *lattice speed of sound* which will appear later on in the equation of state as proportionality factor between pressure and density, *cf.* section 4.1. Why c_s is called the speed of *sound* is shown later in chapter 12.

Eq. (3.56) is one of the most important equations in this book as it represents the discretised equilibrium distribution used in most LB simulations. The specific velocity sets, characterised by the abscissae $\{\boldsymbol{c}_i\}$, weights $\{w_i\}$ and the speed of sound c_s are discussed in section 3.3.7.

Note that $c_\mathrm{s} = 1/\sqrt{3}$ does not hold for all lattices. Higher-order lattices, which can be used to solve the energy equation as well, usually lead to larger values of c_s. However, $c_\mathrm{s} = 1/\sqrt{3}$ holds for all common lattices used for LB simulations of hydrodynamics (*cf.* section 3.3.7).

Concluding, the most important idea one needs to take out of the previous 'heavy' derivations is that the *macroscopic* conservation laws can be satisfied if the equilibrium distribution function is properly discretised in velocity space.

3.3.6 Discretisation of the particle distribution function

We have already mentioned the Hermite series expansion of the population distribution function $f(\boldsymbol{\xi}_i)$ in eq. (3.47):

$$f(\boldsymbol{x}, \boldsymbol{\xi}, t) \approx \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}^{(n)}(\boldsymbol{x}, t) \cdot \boldsymbol{H}^{(n)}(\boldsymbol{\xi}). \tag{3.57}$$

Note that the first two coefficients of the expansion are the same as for the equilibrium distribution which is a consequence of the conservation laws.[13] The discrete velocities $\{\boldsymbol{c}_i\}$ are chosen such that those moments are conserved. Therefore, if one discretises the distribution function f in the same way as f^{eq}, it is guaranteed that the conservation laws of Eq. (3.46) are still satisfied.

Without repeating calculations that we have already performed for the equilibrium distribution function, one can discretise f as

$$f_i(\boldsymbol{x},t) = \frac{w_i}{\omega(\boldsymbol{c}_i)} f(\boldsymbol{x},\boldsymbol{c}_i,t) \tag{3.58}$$

where $\omega(\boldsymbol{c}_i)$ is added to satisfy the Gauss-Hermite rule:

$$\begin{aligned}\boldsymbol{a}^{(n)}(\boldsymbol{x},t) &= \int f(\boldsymbol{x},\boldsymbol{c},t)\boldsymbol{H}^{(n)}\mathrm{d}\boldsymbol{c} = \int \frac{\omega(\boldsymbol{c})}{\omega(\boldsymbol{c})} f(\boldsymbol{x},\boldsymbol{\xi},t)\boldsymbol{H}^{(n)}\mathrm{d}\boldsymbol{\xi} = \sum_i \frac{w_i}{\omega(\boldsymbol{c}_i)} f(\boldsymbol{x},\boldsymbol{c}_i,t)\boldsymbol{H}^{(n)}(\boldsymbol{c}_i) \\ &= \sum_i \boldsymbol{H}_i^{(n)} f_i(\boldsymbol{x},t)\end{aligned} \tag{3.59}$$

NOTE (EMV): Missing Hermite polynomial in the final sum?TODO (Alex): Check c vs xi notation

We have now q separate functions $f_i(\boldsymbol{x},t)$. Similarly to $f_i^{\mathrm{eq}}(\boldsymbol{x},t)$, each is related to one discrete velocity $\boldsymbol{c}_i$, but all are still continuous in space and time.

We can now write down the discrete-velocity Boltzmann equation:

$$\partial_t f_i + c_{i\alpha}\partial_\alpha f_i = \Omega\left(f_i\right), \quad i = 0,\ldots,q-1. \tag{3.60}$$

Furthermore, the macroscopic moments (density and momentum) can then be computed from the finite sums

$$\begin{aligned}\rho &= \sum_{i=0}^{q-1} f_i = \sum_{i=0}^{q-1} f_i^{\mathrm{eq}}, \\ \rho\boldsymbol{u} &= \sum_{i=0}^{q-1} f_i\boldsymbol{c}_i = \sum_{i=0}^{q-1} f_i^{\mathrm{eq}}\boldsymbol{c}_i,\end{aligned} \tag{3.61}$$

rather than from integrals of $f(\boldsymbol{\xi})$ or $f^{\mathrm{eq}}(\boldsymbol{\xi})$ in velocity space.

To finalise the velocity discretisation, we still have to discuss the available velocity sets and their properties.

[13] In the isothermal case, only density and momentum are considered.

3.3.7 Velocity sets

We have now seen how velocity space may be discretised. This naturally leads to the question which discrete velocity set $\{\boldsymbol{c}_i\}$ to choose. On the one hand, an appropriate set has to be sufficiently well-resolved to allow for consistent solutions of the Navier-Stokes or even Navier-Stokes-Fourier equations. On the other hand, the numerical cost of the algorithm scales with the number of velocities. It is therefore an important task to find a set with a minimum number of velocities and yet the ability to capture the desired physics. For a detailed discussion of the history and properties of velocity sets we refer to chapters 3 and 5 in the book by Wolf-Gladrow [1].

General comments and definitions

We name each velocity set by its number d of spatial dimensions and the number q of discrete velocities using the notation DdQq [14]. Two famous examples are D2Q9 (9 discrete velocities in 2D) and D3Q19 (19 discrete velocities in 3D) which will shortly be discussed more thoroughly. A velocity set for the LB algorithm is fully defined by two sets of quantities: the velocities $\{\boldsymbol{c}_i\}$ and the corresponding weights $\{w_i\}$. Another important quantity that can be derived from these two sets is the speed of sound c_s.

The numbering of the velocities in a set is not consistently handled throughout the literature. Some authors choose the index i to run from 0 to $q-1$, others from 1 to q. Neither are there fixed rules in which order the velocities are counted. This means that one has to be careful when working from different articles. It is always recommended to sketch the chosen velocity set and clearly indicate the applied numbering scheme in a coordinate system. For this reason, the most common LB velocity sets are illustrated and their velocity vectors numbered below.

Most velocity sets have one *rest velocity* with zero magnitude, which represents stationary particles. This velocity is often assigned the index $i = 0$: $\boldsymbol{c}_0 = \mathbf{0}$. In this book, we will count from 0 to $q - 1$ if the set has a rest velocity (with 0 indicating the rest velocity) and from 1 to q if there is no rest velocity. For example, D3Q19 has one rest velocity ($i = 0$) and 18 non-rest velocities ($i = 1, \ldots, 18$).

Construction and requirements of velocity sets

There are various approaches to construct LB velocity sets. The first is based on the the Gauss-Hermite quadrature rule described above which leads to the D1Q3, D2Q7, D2Q9, D3Q15, D3Q19, and D3Q27 velocity sets (*cf.* appendix A.4 and [15, 3]). Additionally, D2Q9 and D3Q27 can also be constructed as tensor products of D1Q3.

Another possibility is to construct a $(d - 1)$-dimensional velocity set *via lattice projection* from a known velocity set in d dimensions. We will briefly discuss this later on.

Yet another possibility is to find general conditions a velocity set has to obey. Apart from the conservation of mass and momentum, a paramount requirement is the rotational *isotropy* of the lattice [16]. It depends on the underlying physics what a 'sufficiently isotropic lattice' means. In most cases, LB is used to solve the Navier-Stokes equation for which one requires all moments of the weight w_i up to the fifth order to be isotropic (recall fifth order integration via Hermite polynomials). This leads to the following conditions [17, 18]:

$$\begin{aligned}
\sum_i w_i &= 1, \\
\sum_i w_i c_{i\alpha} &= 0, \\
\sum_i w_i c_{i\alpha} c_{i\beta} &= c_\mathrm{s}^2 \delta_{\alpha\beta}, \\
\sum_i w_i c_{i\alpha} c_{i\beta} c_{i\gamma} &= 0, \\
\sum_i w_i c_{i\alpha} c_{i\beta} c_{i\gamma} c_{i\mu} &= c_\mathrm{s}^4 (\delta_{\alpha\beta}\delta_{\gamma\mu} + \delta_{\alpha\gamma}\delta_{\beta\mu} + \delta_{\alpha\mu}\delta_{\beta\gamma}), \\
\sum_i w_i c_{i\alpha} c_{i\beta} c_{i\gamma} c_{i\mu} c_{i\nu} &= 0.
\end{aligned} \tag{3.62}$$

Additionally, all weights w_i have to be non-negative. Any velocity set which fails to satisfy these conditions is not suitable for LB as a Navier-Stokes solver.

As a general simple approach, one may specify a new velocity set $\{\boldsymbol{c}_i\}$ and use eq. (3.62) as conditional equation for the unknown weights w_i and the speed of sound c_s. On the one hand, if too few velocities are chosen, not all conditions may be satisfied at the same time. For example, the D2Q5 lattice can only satisfy the first four equations in eq. (3.62). On the other hand, the equation system may be overdetermined if more velocities are introduced. For example, for D3Q27 the weights w_i and the sound speed c_s can vary [19]. These free parameters can be used to optimise certain properties (*e.g.* stability).

LB may also be used to simulate advection-diffusion problems for which a lower level of isotropy is sufficient. In this case, only the first four equations in eq. (3.62) have to be fulfilled. If one wants to solve the Navier-Stokes-Fourier system of equations, which includes the energy equation, larger velocity sets are required since higher moments have to be resolved. This leads to so-called *extended lattices*. The discussion of the advection-diffusion equation, energy transfer and their commonly used lattices is put off until chapter 8. In the following, we focus on the lattices used to simulate the Navier-Stokes equation.

We will see in section 3.4 that the LBE is spatially and temporary discretised on a lattice with lattice constant Δx and time step Δt. It is extremely convenient to use velocity sets where all velocities (in this context also called *lattice vectors*) $\boldsymbol{c}_i$ directly connect lattice sites, rather than ending up somewhere between them. We will therefore only consider velocity sets for which all components of the vectors

$\boldsymbol{c}_i$ are integer multiples of $\Delta x/\Delta t$. Usually, Δx and Δt are chosen as being 1 in simulations as discussed in chapter 7. Thus, the velocity vector components $c_{i\alpha}$ are usually integers. Still, it is possible to construct velocity sets without this restriction [3, 17].

Common velocity sets for hydrodynamics

Fig. 3.3 shows the common D1Q3 and D2Q9 sets for 1D and 2D simulations of hydrodynamics. The most popular sets for 3D are D3Q15 and D3Q19 as shown in fig. 3.4, along with the less frequently used D3Q27 velocity set. The velocities and weights of those velocity sets are summarised in tab. 3.1, and given explicitly in Tables 3.2–3.6.

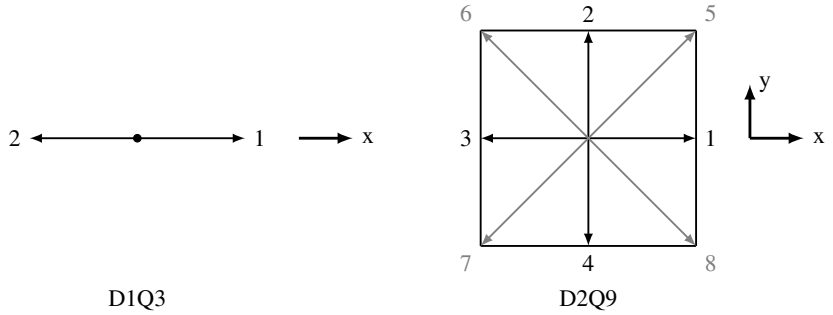


Fig. 3.3: D1Q3 and D2Q9 velocity sets. The square denoted by solid lines has an edge length $2\Delta x$. Velocities with length $|\boldsymbol{c}_i| = 1$ and $\sqrt{2}$ are shown in black and gray, respectively. Rest velocity vectors $\boldsymbol{c}_0 = \boldsymbol{0}$ are not shown. See Tab. 3.1, Tab. 3.2, and Tab. 3.3 for more details.

Exercise 3.6. Show that all velocity sets in tab. 3.1 obey the conditions in eq. (3.62).

It is important to consider which of the three 3D schemes (D3Q15, D3Q19 or D3Q27) is most suitable in a given situation. First of all, D3Q15 is more computationally efficient than D3Q19 which in turn is more efficient than D3Q27. All of these lattices allow to recover hydrodynamics to leading order. However, the functional form of the truncation errors is different. Indeed, numerical errors are typically less significant and the stability is usually better for larger velocity sets. D3Q27 was disregarded for a long time as it was not considered superior to D3Q19 and required 40% more memory and computing power. However, it has been shown that some

Table 3.1: Properties of the most popular velocity sets suitable for Navier-Stokes simulation (compiled from [14, 20]). The speed of sound for all of these velocity sets is $c_s = 1/\sqrt{3}$. These velocity sets are also given explicitly in Tables 3.2–3.6.

notation	velocities $\boldsymbol{c}_i$	number	length $\lvert\boldsymbol{c}_i\rvert$	weight w_i
D1Q3	(0)	1	0	2/3
	(±1)	2	1	1/6
D2Q9	$(0,0)$	1	0	4/9
	$(\pm1,0),(0,\pm1)$	4	1	1/9
	$(\pm1,\pm1)$	4	$\sqrt{2}$	1/36
D3Q15	$(0,0,0)$	1	0	2/9
	$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	6	1	1/9
	$(\pm1,\pm1,\pm1)$	8	$\sqrt{3}$	1/72
D3Q19	$(0,0,0)$	1	0	1/3
	$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	6	1	1/18
	$(\pm1,\pm1,0),(\pm1,0,\pm1),(0,\pm1,\pm1)$	12	$\sqrt{2}$	1/36
D3Q27	$(0,0,0)$	1	0	8/27
	$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	6	1	2/27
	$(\pm1,\pm1,0),(\pm1,0,\pm1),(0,\pm1,\pm1)$	12	$\sqrt{2}$	1/54
	$(\pm1,\pm1,\pm1)$	8	$\sqrt{3}$	1/216

Table 3.2: The D1Q3 velocity set on explicit form

i	0	1	2
w_i	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
c_{ix}	0	+1	−1

Table 3.3: The D2Q9 velocity set on explicit form

i	0	1	2	3	4	5	6	7	8
w_i	$\frac{4}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$
c_{ix}	0	+1	0	−1	0	+1	−1	−1	+1
c_{iy}	0	0	+1	0	−1	+1	+1	−1	−1

Table 3.4: The D3Q15 velocity set on explicit form

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
w_i	$\frac{2}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$	$\frac{1}{72}$
c_{ix}	0	+1	−1	0	0	0	0	+1	−1	+1	−1	+1	−1	−1	+1
c_{iy}	0	0	0	+1	−1	0	0	+1	−1	+1	−1	−1	+1	+1	−1
c_{iz}	0	0	0	0	0	+1	−1	+1	−1	−1	+1	+1	−1	+1	−1

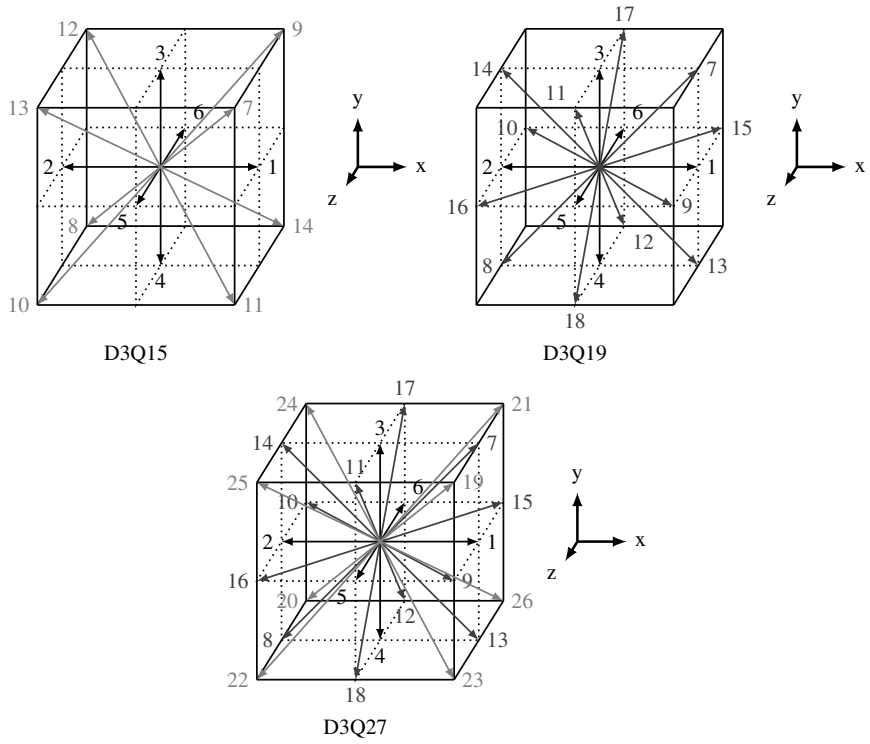


Fig. 3.4: D3Q15, D3Q19 and D3Q27 velocity sets. The cube denoted by solid lines has edge length $2\Delta x$. Velocities with length $|\boldsymbol{c}_i| = 1,\ \sqrt{2},\ \sqrt{3}$ are shown in black, darker grey and lighter grey, respectively. Rest velocity vectors $\boldsymbol{c}_0 = \mathbf{0}$ are not shown. Note that D3Q15 has no $\sqrt{2}$-velocities and D3Q19 has no $\sqrt{3}$-velocities. See Tab. 3.1, Tab. 3.4, Tab. 3.5, and Tab. 3.6 for more details.

Table 3.5: The D3Q19 velocity set on explicit form

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
w_i	$\frac{1}{3}$	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$	$\frac{1}{36}$
c_{ix}	0	+1	−1	0	0	0	0	+1	−1	+1	−1	0	0	+1	−1	+1	−1	0	0
c_{iy}	0	0	0	+1	−1	0	0	+1	−1	0	0	+1	−1	−1	+1	0	0	+1	−1
c_{iz}	0	0	0	0	0	+1	−1	0	0	+1	−1	+1	−1	0	0	−1	+1	−1	+1

Table 3.6: The D3Q27 velocity set on explicit form

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
w_i	$\frac{8}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{54}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$	$\frac{1}{216}$
c_{ix}	0	+1	−1	0	0	0	0	+1	−1	+1	−1	0	0	+1	−1	+1	−1	0	0	+1	−1	+1	−1	+1	−1	−1	+1
c_{iy}	0	0	0	+1	−1	0	0	+1	−1	0	0	+1	−1	−1	+1	0	0	+1	−1	+1	−1	+1	−1	−1	+1	+1	−1
c_{iz}	0	0	0	0	0	+1	−1	0	0	+1	−1	+1	−1	0	0	−1	+1	−1	+1	+1	−1	−1	+1	+1	−1	+1	−1

error terms of D3Q15 and D3Q19, in contrast to D3Q27, are not rotationally invariant, which can lead to problems when high Reynolds number flows are simulated [21, 22]. Therefore, D3Q27 is probably the best choice for turbulence modelling, but D3Q19 is usually a good compromise for laminar flows.

As a sidenote, there also exists a D3Q13 velocity set which only works within the framework of multi-relaxation-time LB (*cf.* chapter 10) [23]. D3Q13 is an example of a lattice exhibiting the so-called *chessboard* or *checkerboard instability* [1] previously covered in Section 2.1.1. D3Q13 is the minimal velocity set to simulate the Navier-Stokes equation in three dimensions and can be very efficiently implemented on Graphical Processing Units (GPUs) [24].

We now turn our attention to the projection of velocity sets to lower-dimensional spaces.

Velocity set relations

Historically, the 2D lattice gas models described in Section 2.2.2 used a hexagonal D2Q6 velocity set [1]. The initial paradigm was that all velocities $\boldsymbol{c}_i$ should have the same magnitude and weight (*i.e.* single-speed velocity sets with $w_i = 1/q$ for all i) and therefore point at the surface of a single sphere in velocity space. This ruled out the existence of rest velocities.

While the D1Q2 and D2Q6 single-speed sets were fairly simple to determine, it was surprisingly difficult to determine such a single-set velocity speed in 3D [1]. The solution to this problem was to construct a D4Q24 single-speed velocity set in four dimensions, with 24 velocities consisting of all the various spatial permutations of $(\pm 1, \pm 1, 0, 0)$. This 4D velocity set was then *projected* down to 3D:

$$\begin{aligned}
&\begin{pmatrix}\pm1\\0\\0\\\pm1\end{pmatrix} \to \begin{pmatrix}\pm1\\0\\0\end{pmatrix}, \qquad \begin{pmatrix}0\\\pm1\\0\\\pm1\end{pmatrix} \to \begin{pmatrix}0\\\pm1\\0\end{pmatrix}, \qquad \begin{pmatrix}0\\0\\\pm1\\\pm1\end{pmatrix} \to \begin{pmatrix}0\\0\\\pm1\end{pmatrix}. \\
&\begin{pmatrix}\pm1\\\pm1\\0\\0\end{pmatrix} \to \begin{pmatrix}\pm1\\\pm1\\0\end{pmatrix}, \qquad \begin{pmatrix}\pm1\\0\\\pm1\\0\end{pmatrix} \to \begin{pmatrix}\pm1\\0\\\pm1\end{pmatrix}, \qquad \begin{pmatrix}0\\\pm1\\\pm1\\0\end{pmatrix} \to \begin{pmatrix}0\\\pm1\\\pm1\end{pmatrix},
\end{aligned} \tag{3.63}$$

Thus, the single-speed D4Q24 velocity set leads to a *multi-speed* D3Q18 velocity set, with velocities of length 1 and $\sqrt{2}$.

Note an important difference between the upper and the lower rows of Eq. (3.63). In the lower row, the projections are one-to-one: Each resulting 3D velocity vector corresponds to only one 4D velocity vector. In the upper row, the projections are *degenerate*: Each 3D vector here corresponds to *two different* 4D velocity vectors. This implies that the resulting multi-speed 3D velocity set has nonequal weights w_i such that the shorter velocity vectors in the upper row have twice the weight of the

longer vectors in the lower row. This is in contrast to the single-speed D2Q6 and D4Q24 velocity sets, where all the velocities are weighted equally.

In fact, all the velocity sets listed in tab. 3.1 are related through such projections: D2Q9 is the two-dimensional projection of the D3Q15, D3Q19, and D3Q27 velocity sets, while D1Q3 is the one-dimensional projection of all of these.

Example 3.3. D2Q9 can be obtained from D3Q19 by projecting the latter onto the x-y plane:

$$\begin{gathered}
\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\\pm1\end{pmatrix} \to \begin{pmatrix}0\\0\end{pmatrix}, \qquad \begin{pmatrix}\pm1\\0\\0\end{pmatrix}, \begin{pmatrix}\pm1\\0\\\pm1\end{pmatrix} \to \begin{pmatrix}\pm1\\0\end{pmatrix}, \\
\begin{pmatrix}0\\\pm1\\0\end{pmatrix}, \begin{pmatrix}0\\\pm1\\\pm1\end{pmatrix} \to \begin{pmatrix}0\\\pm1\end{pmatrix}, \qquad \begin{pmatrix}\pm1\\\pm1\\0\end{pmatrix} \to \begin{pmatrix}\pm1\\\pm1\end{pmatrix}.
\end{gathered} \tag{3.64}$$

The D2Q9 weights can also be obtained from this projection. For instance, the above shows us that the D2Q9 zero-velocity vector $\boldsymbol{c}_0$ is a projection of three different D3Q19 velocity vectors, whose total weight $1/3 + 1/18 + 1/18 = 4/9$ is the D2Q9 rest velocity weight. It is similarly easy to obtain the other D2Q9 weights from the D3Q19 weights.

Example 3.4. Similarly, D1Q3 can be obtained by projecting D2Q9 onto the x axis:

$$\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}0\\\pm1\end{pmatrix} \to (0), \qquad \begin{pmatrix}\pm1\\0\end{pmatrix}, \begin{pmatrix}\pm1\\\pm1\end{pmatrix} \to (\pm1). \tag{3.65}$$

The D1Q3 rest velocity then obtains the weight $4/9 + 1/9 + 1/9 = 2/3$, while the non-rest velocities each obtain weights of $1/9 + 1/36 + 1/36 = 1/6$.

Exercise 3.7. Show that the D3Q27 velocity set becomes the D2Q9 and the D1Q3 velocity sets when projected down to two and one dimensions, respectively.

We have already mentioned that D3Q27 is a 3D generalisation of D1Q3. In addition, D3Q15 and D3Q19 can be obtained from D3Q27 *via* so-called *pruning* [25, 3], where some of the particle velocities are discarded in order to create a smaller velocity set. In turn, D3Q27 can be understood as a superposition of D3Q15 weighted by $1/3$ and D3Q19 weighted by $2/3$, as can be seen from tab. 3.1. The relations between these velocity sets are shown in fig. 3.5.

The fact that lower-dimensional velocity sets may be projections of higher-dimensional ones means that simulations may simplified in cases where the simulated problem is invariant along one or more axes. For instance, a plane sound wave propagating along the x axis is invariant in the y and z directions. The macroscopic results of such a simulation will be the same whether it is simulated with the D1Q3 velocity set or with the two- and three-dimensional velocity sets discussed above combined with periodic y and z boundary conditions [18].

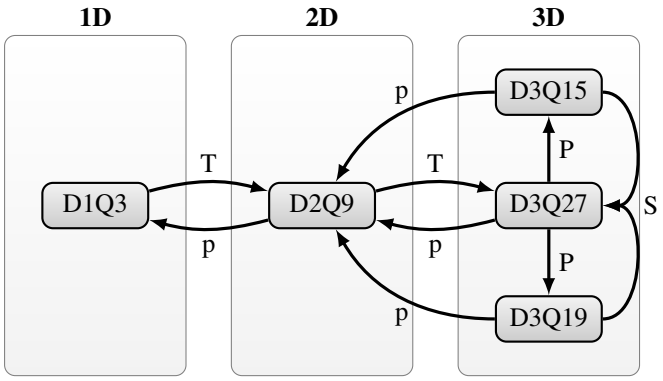


Fig. 3.5: Mutual relations of common LB velocity sets. Lower-dimensional velocity sets can be obtained from higher-dimensional ones *via* projection (p). D2Q9 and D3Q27 are tensor (T) products of D1Q3. D3Q15 and D3Q19 can be obtained from D3Q27 *via* pruning (P). D3Q27 can in turn be considered a superposition (S) of D3Q15 and D3Q19.

Equilibrium distributions

In this section we will present equilibrium distribution functions for D1Q3 and D2Q9 lattices that can be directly used in simulations. Sometimes students experience difficulties to unroll tensor notations used for the equilibrium function representation. Thus, as reference we here present the D1Q3 and D2Q9 equilibrium distribution functions.

We have already found the discretised equilibrium distribution function, eq. (3.56):

$$f_i^{\mathrm{eq}} = w_i \rho \left(1 + \frac{c_{i\alpha} u_\alpha}{c_\mathrm{s}^2} + \frac{u_\alpha u_\beta \left(c_{i\alpha} c_{i\beta} - c_\mathrm{s}^2 \delta_{\alpha\beta} \right)}{2 c_\mathrm{s}^4} \right). \tag{3.66}$$

This polynomial equilibrium of second order in $\boldsymbol{u}$ is sufficient to recover Navier-Stokes behaviour and is valid for any of the above-mentioned velocity sets (D1Q3, D2Q9, D3Q15, D3Q19 and D3Q27) for which the speed of sound is $c_\mathrm{s} = 1/\sqrt{3}$. (Different equilibria may be used for other purposes; we will come back to this in Section 4.3 and Chapter 8.)

In order to implement eq. (3.66) numerically, one can either write a for- or do-loop (which is less susceptible to bugs but usually slower) or one can manually

unroll the q discretised equilibrium distributions (which is normally more computationally efficient but also more cumbersome to implement).[14]

Example 3.5. For the D1Q3 velocity set in Tab. 3.2, the equilibrium distribution reads (with $\boldsymbol{u} = (u)$ and $c_\mathrm{s}^2 = 1/3$):

$$\begin{aligned} f_0^\mathrm{eq} &= \frac{\rho}{3}\left(2 - 3u^2\right), \\ f_1^\mathrm{eq} = \frac{\rho}{6}\left[1 + 3\left(u + u^2\right)\right], &\qquad f_2^\mathrm{eq} = \frac{\rho}{6}\left[1 - 3\left(u - u^2\right)\right]. \end{aligned} \tag{3.67}$$

Example 3.6. The D2Q9 equilibrium distribution for the chosen velocity vectors in Tab. 3.3 is given by (using $\boldsymbol{u} = (u_x, u_y)^\top$, $\boldsymbol{u}^2 = u_x^2 + u_y^2$ and $c_\mathrm{s}^2 = 1/3$):

$$\begin{aligned} f_0^\mathrm{eq} &= \frac{2\rho}{9}\left(2 - 3\boldsymbol{u}^2\right), \\ f_1^\mathrm{eq} &= \frac{\rho}{18}\left(2 + 6u_x + 18u_x^2 - 3\boldsymbol{u}^2\right), \quad f_5^\mathrm{eq} = \frac{\rho}{36}\left[1 + 3(u_x + u_y) + 9u_xu_y + 3\boldsymbol{u}^2\right], \\ f_2^\mathrm{eq} &= \frac{\rho}{18}\left(2 + 6u_y + 18u_y^2 - 3\boldsymbol{u}^2\right), \quad f_6^\mathrm{eq} = \frac{\rho}{36}\left[1 - 3(u_x - u_y) - 9u_xu_y + 3\boldsymbol{u}^2\right], \\ f_3^\mathrm{eq} &= \frac{\rho}{18}\left(2 - 6u_x + 18u_x^2 - 3\boldsymbol{u}^2\right), \quad f_7^\mathrm{eq} = \frac{\rho}{36}\left[1 + 3(u_x + u_y) + 9u_xu_y + 3\boldsymbol{u}^2\right], \\ f_4^\mathrm{eq} &= \frac{\rho}{18}\left(2 - 6u_y + 18u_y^2 - 3\boldsymbol{u}^2\right), \quad f_8^\mathrm{eq} = \frac{\rho}{36}\left[1 + 3(u_x - u_y) - 9u_xu_y + 3\boldsymbol{u}^2\right]. \end{aligned} \tag{3.68}$$

Exercise 3.8. Write down the equilibria for the 3D lattices.

Macroscopic moments

We have already stated in eq. (3.61) how to compute the moments in discretised velocity space. Using the general equilibrium distribution in eq. (3.66) together with the isotropy conditions in eq. (3.62), which hold for all the velocity sets we have discussed here (*i.e.* D1Q3, D2Q9, D3Q15, D3Q19, and D3Q27), we can find the zeroth to third order moments explicitly:

$$\begin{aligned} \Pi^\mathrm{eq} &= \sum_i f_i^\mathrm{eq} = \rho, \\ \Pi_\alpha^\mathrm{eq} &= \sum_i f_i^\mathrm{eq} c_{i\alpha} = \rho u_\alpha, \\ \Pi_{\alpha\beta}^\mathrm{eq} &= \sum_i f_i^\mathrm{eq} c_{i\alpha} c_{i\beta} = \rho c_\mathrm{s}^2 \delta_{\alpha\beta} + \rho u_\alpha u_\beta, \\ \Pi_{\alpha\beta\gamma}^\mathrm{eq} &= \sum_i f_i^\mathrm{eq} c_{i\alpha} c_{i\beta} c_{i\gamma} = \rho c_\mathrm{s}^2 \left(u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\alpha\gamma} + u_\gamma \delta_{\alpha\beta}\right). \end{aligned} \tag{3.69}$$

Exercise 3.9. Show the validity of these relations by explicitly calculating these moments of eq. (3.66) using the conditions in eq. (3.62).

[14] Some modern compilers unroll loops automatically, but we encourage checking this numerically with performance tools.

One already can see some similarities with the Navier-Stokes equation. For example, the Navier-Stokes equation contains a term $\nabla(p + \rho u_\alpha u_\beta)$. Comparing this with the second order moments of the discrete-velocity Boltzmann equation, we can *guess* (and we will *prove* it later in section 4.1) that the equation of state for the lattice Boltzmann equation is $p = c_s^2 \rho$.

3.4 Discretisation in space and time

So far we have only performed the discretisation of velocity space. This section is dedicated to the final step towards the LBE: the discretisation of space and time.

As we discussed in Section 2.1, the space discretisation of some conventional CFD methods, such as finite volume or finite element methods, is arbitrary to some extent. Each volume or element can have many possible shapes, such as triangles, quads, tetrahedra, pyramids or hexes. This is not the case for the 'classical' LBM. Though there do exist LBM discretisations on unstructured grids [26, 27], the most common form of space discretisaton is a uniform and structured grid. As we will see this also implies a strong coupling of the spatial and temporal discretisations in the LBM.

Overall, the original LB algorithm assumes that populations f_i move with velocity $\boldsymbol{c}_i$ from one lattice site to another. After one time step Δt, each population should exactly reach a neighbouring site. This is guaranteed if i) the underlying spatial lattice is uniform and regular with lattice constant Δx and ii) the velocity components are integer multiples of $\Delta x/\Delta t$, *i.e.* $c_{i\alpha} = n\Delta x/\Delta t$.

We have already mentioned in section 3.3.7 that all common velocity sets obey this condition. Thus, we can expect that the populations starting at a lattice site at $\boldsymbol{x}$ end up at another lattice site at $\boldsymbol{x} + \boldsymbol{c}_i\Delta t$, *i.e.* that the populations do not 'get stuck' between lattice sites.[15]

In the following we will present how the discrete-velocity Boltzmann equation can be further discretised in physical space and time.

3.4.1 Method of characteristics

Let us recall from Eq. (3.60) the non-dimensional force-free discrete-velocity Boltzmann equation with a general collision operator Ω_i that conserves density and momentum:

$$\partial_t f_i + c_{i\alpha}\partial_\alpha f_i = \Omega_i, \tag{3.70}$$

where $f_i(\boldsymbol{x}, t) = f(\boldsymbol{x}, \boldsymbol{c}_i, t)$ is the particle distribution function discretised in velocity space. We also call f_i the population of particles moving in direction $\boldsymbol{c}_i$. Note that

[15] This is not a hard-and-fast rule, though; it is possible to work with cases where $\boldsymbol{x} + \boldsymbol{c}_i\Delta t$ falls between lattice sites [3, 17].

the form of the collision operator is not yet specified. It is rather assumed that it somehow depends on the discretised populations $\{f_i\}$ and the equilibrium populations $\{f_i^{\mathrm{eq}}\}$. Also note that the equilibrium populations depend on the macroscopic quantities such as density and velocity, which can be explicitly found through the moments of the populations $\{f_i\}$. Thus, we can assume that the collision operator Ω can be determined fully through the discretised populations $\{f_i\}$.

Eq. (3.70) can be classified as a first-order partial differential equation (PDE). Each velocity $\boldsymbol{c}_i$ is a known constant. There exist a number of techniques to solve equations like eq. (3.70). One particularly powerful approach to tackle linear first-order PDEs is the so-called *method of characteristics* (or *method of trajectories*).

This method of characteristics exploits the existence of trajectories in the space of a PDE's independent variables, *i.e.* $\boldsymbol{x}$ and t for Eq. (3.70), which lets us simplify the PDE. What does this mean physically? Let us consider the hyperbolic equation

$$\frac{\partial g}{\partial t} + \boldsymbol{a} \cdot \boldsymbol{\nabla} g = 0, \tag{3.71}$$

where the vector $\boldsymbol{a}$ is a constant. This equation describes the advection of the quantity g at a velocity given by the vector $\boldsymbol{a}$. One can therefore simplify the solution of the PDE by defining a trajectory $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{a}t$ or $\boldsymbol{x} - \boldsymbol{a}t = \boldsymbol{x}_0$, where $\boldsymbol{x}_0$ is an arbitrary constant.

Exercise 3.10. Show using the chain rule that any function $g = g(\boldsymbol{x} - \boldsymbol{a}t)$ solves eq. (3.71).

In the method of characteristics, one can generally parametrise a PDE's independent variables in such a way that the PDE can be re-expressed as an ODE. We can write the solution of eq. (3.70) in the form $f_i = f_i(\boldsymbol{x}(\zeta), t(\zeta))$, where ζ parametrises a trajectory in space.

By converting the left-hand side of eq. (3.70) into a total derivative with respect to the parameter ζ, we find

$$\frac{\mathrm{d}f_i}{\mathrm{d}\zeta} = \left(\frac{\partial f_i}{\partial t}\right)\frac{\mathrm{d}t}{\mathrm{d}\zeta} + \left(\frac{\partial f_i}{\partial x_\alpha}\right)\frac{\mathrm{d}x_\alpha}{\mathrm{d}\zeta} = \Omega_i\left(\boldsymbol{x}(\zeta), t(\zeta)\right). \tag{3.72}$$

For Eq. (3.72) to be equal to Eq. (3.70), we must have

$$\frac{\mathrm{d}t}{\mathrm{d}\zeta} = 1, \quad \frac{\mathrm{d}x_\alpha}{\mathrm{d}\zeta} = c_{i\alpha}. \tag{3.73}$$

Exercise 3.11. Show from Eq. (3.73) that the solution follows a trajectory given by $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{c}_i t$, where $\boldsymbol{x}_0$ is an arbitrary constant.

Now we want to integrate both sides of eq. (3.72) along the trajectory, but we have to specify initial conditions first. Let us take a look at the trajectory passing the point $(\boldsymbol{x}_0, t_0)$. This implies $t(\zeta = 0) = t_0$ and $\boldsymbol{x}(\zeta = 0) = \boldsymbol{x}_0$. The integration from $\zeta = 0$ to $\zeta = \Delta t$ of Eq. (3.72) then yields:

$$f_i(\boldsymbol{x}_0 + \boldsymbol{c}_i\Delta t, t_0 + \Delta t) - f_i(\boldsymbol{x}_0, t_0) = \int_0^{\Delta t} \Omega_i(x_0 + \boldsymbol{c}_i\zeta, t_0 + \zeta)\,\mathrm{d}\zeta. \tag{3.74}$$

By the fundamental theorem of calculus, the integration of the left-hand side is exact. Note that the point $(\boldsymbol{x}_0, t_0)$ is arbitrary so that we can more generally write

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = \int_0^{\Delta t} \Omega_i(x + \boldsymbol{c}_i\zeta, t + \zeta)\,\mathrm{d}\zeta. \tag{3.75}$$

However, the right-hand side is not as simple to determine as the left-hand side; in the following section we will see how the integral may be *approximated*.

In eq. (3.75) we can already see the discretisation pattern introduced above: During the time step Δt, the population $f_i(\boldsymbol{x}, t)$ moves from $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{c}_i\Delta t$, giving $f_i(\boldsymbol{x} + \boldsymbol{c}_i, t + \Delta t, t + \Delta t)$. This supports our aforementioned 'naive' idea of space and time discretisation, where populations exactly reach neighbouring lattice sites after one time step, given that the lattice is uniform and regular and the chosen velocity set is such that $c_{i\alpha} = n\Delta x/\Delta t$.

3.4.2 First- and second-order discretisation

There are several ways to approximate the right-hand side of eq. (3.75). The most common space-time integration methods include the *Crank-Nicolson* [28] and the *Runge-Kutta* schemes [29]. Although these methods, among others, typically allow a more accurate integration, the 'classical' LBE employs the simple explicit *forward Euler* scheme previously seen in Section 2.1.

There are good reasons for this. Runge-Kutta-type schemes require tracking the populations f_i at several points in time (and for this reason they are known as *multi-step* schemes). This is memory-intensive, especially when D3Q27 is used. The Crank-Nicolson time-space discretisation of the LBE [10, 28] leads to the original LBE after introducing new variables as we will see below. Other implicit discretisations of Eq. (3.75) leads to a linear system of equations which has to be solved. It is needless to say that this is computationally demanding, especially for D3Q27 with its 27 populations for each node, and is less attractive than it is for the four variables (*i.e.* pressure and three velocity components) of the incompressible Navier-Stokes equation in a finite volume solver.

One interesting alternate space-time discretisation of the discrete-velocity Boltzmann equation is the finite volume formulations of the lattice Boltzmann equation [30, 26, 27].[16] The finite-volume formulations are more flexible in terms of generating meshes that fit complex geometries [30], especially if one wants to use local grid-refinement, which is difficult to computationally implement for uniform grids [31]. However, explicit finite volume formulations [32, 33] (using forward Euler or Runge-Kutta formulations) are inferior in terms of stability and/or computational ef-

[16] We have described the general principles of such finite volume methods in Section 2.1.2.

ficiency. Implicit finite volume formulations lead to a system of equations with high computational overhead. We will not discuss non-'classical' discretisations of the Boltzmann equation in more detail and we instead refer to the literature indicated above.

The beauty and the weakness of the LBE lies in its explicitness and uniform grid. The explicit discretisation of velocities, space and time allows for a relatively easy setup of complex boundary conditions, *e.g.* for multiphase flows in porous media. Also, the uniform grid allows for effective parallelisation [24]. However, this comes with a price: there are some stability restrictions on the lattice constant Δx and the time step Δt.

We will now take a closer look at first-order and second-order discretisations of the right-hand side of eq. (3.75).

First-order discretisation

The first-order discretisation, also denoted *rectangular discretisation*, approximates the collision operator integral by just one point:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = \Delta t\, \Omega_i(\boldsymbol{x}, t). \tag{3.76}$$

The scheme in eq. (3.76) is fully explicit and the most used for LB simulations. In its form

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + \Delta t\, \Omega_i(\boldsymbol{x}, t) \tag{3.77}$$

it is called the *lattice Boltzmann equation* (LBE).

However, eq. (3.77) alone is still not of much use since the collision operator Ω_i has not yet been explicitly specified. We will get back to it in section 3.4.3. Yet we could generally expect that the explicit scheme in eq. (3.77) is of first-order accuracy in time as the right-hand side was determined by a first-order approximation. However, as we will see below the second-order discretisation of the integral leads to the same form of the lattice Boltzmann equation. Thus, the lattice Boltzmann equation in Eq. (3.77) is actually second-order accurate in time.

Second-order discretisation

We obtain a more accurate approximation of the right-hand side of eq. (3.75) *via* the *trapezoidal rule*:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = \Delta t \frac{\Omega_i(\boldsymbol{x}, t) + \Omega_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t)}{2}. \tag{3.78}$$

This is a second-order accurate discretisation.

Eq. (3.78) is implicit since $\Omega_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t)$ depends on f_i at $t + \Delta t$. However, it is possible to transform this equation into an explicit form by introducing

$$\bar{f}_i = f_i - \frac{\Delta t \Omega_i}{2}. \tag{3.79}$$

This transformation has some important properties. First, the new populations $\bar{f}_i$ conserve the same moments as the original populations:[17]

$$\begin{aligned} \sum_i \bar{f}_i &= \sum_i f_i - \frac{\Delta t}{2} \sum_i \Omega_i = \rho, \\ \sum_i \bar{f}_i \boldsymbol{c}_i &= \sum_i f_i \boldsymbol{c}_i - \frac{\Delta t}{2} \sum_i \Omega_i \boldsymbol{c}_i = \rho \boldsymbol{u}. \end{aligned} \tag{3.80}$$

As the equilibrium f_i^{eq} is determined by these moments, it can be calculated from f_i or from $\bar{f}_i$ with the same result.

After some algebra one can rewrite eq. (3.78) in terms of the new populations:

$$\bar{f}_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = \bar{f}_i(\boldsymbol{x}, t) + \Delta t \Omega_i. \tag{3.81}$$

This equation is on the same form as Eq. (3.77), but with f_i changed to the transformed variable $\bar{f}_i$. Ideally, we would like to eliminate the untransformed variable f_i from Eq. (3.81) so that we can solve this equation for $\bar{f}_i$ without having to determine f_i. While the collision operator in Eq. (3.81) is still given by f_i, all common collision operators can be re-expressed without the use of f_i. This is covered for one particular collision operator in Exercise 3.13.

Therefore, we have found the somewhat surprising result that the second-order discretisation in eq. (3.81) looks exactly like the 'first-order' discretisation in eq. (3.77), which indicates that discretisations are actually second-order. The second-order accuracy of the LBE in both forms (*i.e.* eq. (3.77) and eq. (3.81)) can also be proven by other methods [28].

As already mentioned above, neither method is very useful without a properly defined collision operator. We will now investigate the so-called BGK collision operator, the simplest and most widely used collision operator for the LBE. Other, more elaborate forms will be introduced and thoroughly discussed in chapter 10.

[17] The situation changes in the presence of forces as will be discussed in chapter 6.

3.4.3 Collision operator

We mentioned already in section 1.3.3 that the collision operator of the original Boltzmann equation considers all possible outcomes of binary collisions and has a rather complicated and cumbersome mathematical form. This collision operator is only suitable for gas simulations, as it only accounts for binary collisions between molecules. However, due to the significantly larger densities, molecules in liquids can undergo more complicated interactions involving three and more particles. So one could naively assume that more complicated integrals accounting for all these possible interactions are required to characterise the collisions in liquids. Fortunately, this is not necessary.

As we discussed in section 3.3, one does not need to know all underlying microscopic information to recover the macroscopic equations. This important observation can be used to simplify the collision operator *significantly*. In particular, one can get rid of any complicated integrals. The first step is to *approximate* the collision operator and write it in terms of the known variables, the populations f_i and the equilibrium populations f_i^{eq}. The simplest non-trivial functional form is a linear relation, so we assume that Ω_i should contain both f_i and f_i^{eq} only linearly. Let us take a closer look at the form $\Omega_i \propto (f_i - f_i^{\mathrm{eq}})$. Notice that the linear form also conserves mass and momentum, as we require for the Navier-Stokes behaviour:

$$\begin{aligned} \sum_i \Omega_i &\propto \sum_i (f_i - f_i^{\mathrm{eq}}) = 0, \\ \sum_i \Omega_i \boldsymbol{c}_i &\propto \sum_i \left(f_i \boldsymbol{c}_i - f_i^{\mathrm{eq}} \boldsymbol{c}_i \right) = 0. \end{aligned} \tag{3.82}$$

The most important property of collision operators is mass and momentum conservation. One can easily construct simple collision operators by writing down linear functions of $\{f_i\}$ and $\{f_i^{\mathrm{eq}}\}$.

The BGK collision operator

Let us now adapt from Eq. (1.47) the *Bhatnagar-Gross-Krook* (*BGK*) collision operator [34]:

$$\Omega_i = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau}. \tag{3.83}$$

What does eq. (3.83) mean physically? It can be interpreted as the tendency of the population f_i to approach its equilibrium state f_i^{eq} after a time τ. This process is also called *relaxation towards equilibrium* and τ is therefore denoted the *relaxation time*.

Exercise 3.12. Show that the solution of

$$\frac{\mathrm{d}f_i}{\mathrm{d}t} = \frac{f_i - f_i^{\mathrm{eq}}}{\tau} \tag{3.84}$$

leads to an exponential decay, $(f_i - f_i^{\mathrm{eq}}) \propto \exp(-t/\tau)$ if we assume that f_i^{eq} is constant.

Substituting the BGK collision operator from eq. (3.83) into the 'first-order' approximation of the collision operator integral in eq. (3.77) gives the lattice Boltzmann equation with BGK collision operator, also sometimes called the *lattice BGK (LBGK)* equation:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left[f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right]. \tag{3.85}$$

As we will see in section 4.1, this very crude approximation of the original Boltzmann collision operator works astonishingly well in most cases. In particular we will show that the LBE with this simple BGK collision operator is able to reproduce the continuity and the Navier-Stokes equations. This is one of the main reasons why the LBM has become so popular.

Note that the BGK collision operator is not the only possible approximation of the original collision operator. For example, there exist two-relaxation-times (TRT) and multi-relaxation-times (MRT) collision operators which utilise more than just a single relaxation time. (The BGK operator is also often called a *single-relaxation-time (SRT)* collision operator.) These extended collision operators allow avoiding or mitigating some limitations of the BGK collision operator, such as stability and accuracy issues. We will get back to this more advanced topic in chapter 10.

Exercise 3.13. Show that substituting the BGK collision operator from Eq. (3.83) into the 'second-order' LBE in Eq. (3.78) leads to a scheme

$$\bar{f}_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = \bar{f}_i(\boldsymbol{x}, t) - \frac{\Delta t}{\bar{\tau}}\left[\bar{f}_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right], \tag{3.86}$$

with a redefined relaxation time $\bar{\tau} = \tau + \Delta t/2$.

As the form of Eq. (3.85) and Eq. (3.86) is absolutely the same, there is no practical difference[18] between using the first order or second order approximation of the collision integral. This is one of several proofs of the second-order accuracy of the LBE in time [28].[19]

[18] The difference appears in the presence of forces.

[19] Another example of the second-order accuracy proof is covered in appendix A.5 and works by directly solving the PDE

$$\frac{\partial f_i}{\partial t} + c_{i\alpha}\frac{\partial f_i}{\partial x_\alpha} = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau}. \tag{3.87}$$

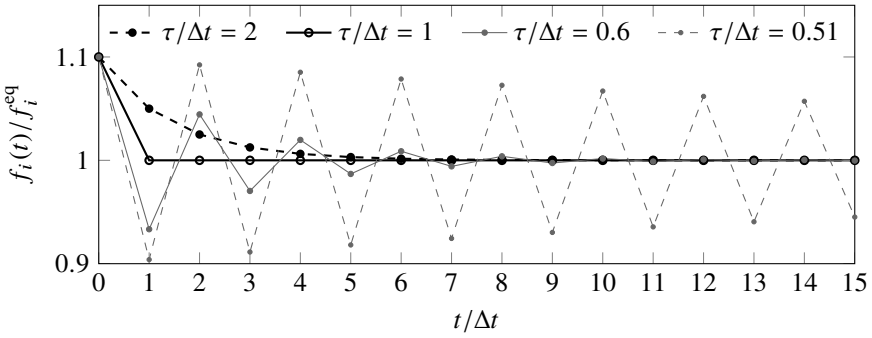


Fig. 3.6: Simple example of under-, full, and over-relaxation for the spatially homogeneous lattice BGK equation in Eq. (3.88) with an initial condition $f_i(0)/f_i^{\mathrm{eq}} = 1.1$ and constant f_i^{eq}.

Under-, full, and over-relaxation

The lattice BGK equation, being discrete in time and space, differs from the continuous BGK equation in one major respect. While the latter always evolves f_i *towards* f_i^{eq} (see Exercise 3.12), the lattice BGK equation can also evolve f_i *immediately to* f_i^{eq} or even *past* f_i^{eq}.

To see why this is so, we briefly look at the discrete analogue of the spatially homogeneous continuous BGK equation Eq. (3.84). From Eq. (3.85), this is

$$f_i(t + \Delta t) = \left(1 - \frac{\Delta t}{\tau}\right) f_i(t) + \frac{\Delta t}{\tau} f_i^{\mathrm{eq}}. \tag{3.88}$$

Depending on the choice of $\tau/\Delta t$, we find that f_i relaxes in one of three different ways:

- **Under-relaxation** for $\tau/\Delta t > 1$, where f_i decays *exponentially towards* f_i^{eq} like in the continuous-time BGK equation.
- **Full relaxation** for $\tau/\Delta t = 1$, where f_i decays *immediately to* f_i^{eq}.
- **Over-relaxation** for $1/2 < \tau/\Delta t < 1$, where f_i *oscillates around* f_i^{eq} with an exponentially decreasing amplitude.

These cases are illustrated in Fig. 3.6. A fourth, *unstable*, case is $\tau/\Delta t < 1/2$, where f_i oscillates around f_i^{eq} with an exponentially *increasing* amplitude. Consequently, $\tau/\Delta t \geq 1/2$ is a necessary condition for stability. (Other stability conditions will be covered in Section 4.4.)

Finally, we are done discretising the Boltzmann equation in velocity space, physical space, and time, and we have replaced its complicated collision operator by the

simple BGK collision operator. Before starting to write LB simulations on a computer, it is helpful to understand the concept of separating eq. (3.85) into streaming (or propagation) and collision (or relaxation) substeps.

3.4.4 Streaming and collision

By having a close look at eq. (3.85) we can identify two separate parts. One comes from the integration along characteristics, $f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t)$. The other comes from the local collision operator, $-\Delta t[f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)]/\tau$. We can therefore logically separate the LBGK equation into distinct streaming (or propagation) and collision steps.

Overall, each lattice site at point $\boldsymbol{x}$ and time t stores q populations f_i. In the *collision step* or *relaxation step*, each population $f_i(\boldsymbol{x}, t)$ receives a collisional contribution and becomes

$$f_i^{\star}(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left[f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right]. \tag{3.89}$$

Collision is a purely local and algebraic operation. $f_i^{\star}$ denotes the state of the population *after* collision.[20]

The other step is the *streaming step* or *propagation step*. Here, the post-collision populations $f_i^{\star}(\boldsymbol{x}, t)$ just stream along their associated direction $\boldsymbol{c}_i$ to reach a neighbouring lattice site where they become $f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t)$:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i^{\star}(\boldsymbol{x}, t). \tag{3.90}$$

This is a non-local operation. Practically, one has to copy the memory content of $f_i^{\star}(\boldsymbol{x}, t)$ to the lattice site located at $\boldsymbol{x} + \boldsymbol{c}_i\Delta t$ and overwrite its old information. (One has to be careful not to overwrite populations which are still required.) Usually two set of populations are used for implementation (see chapter 13).

In summary, the LBGK equation is implemented using the collision and streaming steps:

[20] Note that is is more convenient for code implementation to write eq. (3.89) in the form

$$f_i^{\star}(\boldsymbol{x}, t) = \left(1 - \frac{\Delta t}{\tau}\right) f_i(\boldsymbol{x}, t) + \frac{\Delta t}{\tau} f_i^{\mathrm{eq}}(\boldsymbol{x}, t).$$

The specific choice $\tau = \Delta t$ (which is quite common in LB simulations) leads to the extremely efficient collision rule

$$f_i^{\star}(\boldsymbol{x}, t) = f_i^{\mathrm{eq}}(\boldsymbol{x}, t),$$

i.e. the populations directly go to their equilibrium and forget about their previous state. More details about efficient implementations of eq. (3.89) are provided in chapter 13.

$$
\begin{aligned}
f_i^\star(\boldsymbol{x},t) &= f_i(\boldsymbol{x},t) - \frac{\Delta t}{\tau}\left[f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right] \quad \text{(collision)},\\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) &= f_i^\star(\boldsymbol{x},t) \quad \text{(streaming)}.
\end{aligned}
\tag{3.91}
$$

Now we have derived everything required to write a first LB simulation code, except boundary conditions. The most important results of this chapter were already collected in section 3.1, and simple implementation hints were covered in Section 3.2. In the next chapter, we will show that the LBGK equation actually simulates the Navier-Stokes equation.

References

1. D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models* (Springer, 2005)
2. X. He, L.S. Luo, Phys. Rev. E **56**(6), 6811 (1997)
3. X. Shan, X.F. Yuan, H. Chen, J. Fluid Mech. **550**, 413 (2006)
4. *Introduction to Tensor Calculus*
5. H. Grad, Commun. Pure Appl. Math. **2**(4), 325
6. N. Wiener, *The Fourier integral and certain of its applications* (Cambridge University Press, 1933)
7. D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, L.S. Luo, Phil. Trans. R. Soc. Lond. A **360**, 437 (2002)
8. A. Kuzmin, A. Mohamad, S. Succi, Int. J. Mod. Phys. C **19**(6), 875 (2008)
9. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
10. P. Dellar, Phys. Rev. E **64**(3) (2001)
11. Z. Guo, C. Zheng, B. Shi, T. Zhao, Phys. Rev. E **75**(036704), 1 (2007)
12. G. Uhlenbeck, G. Ford, *Lectures in statistical mechanics*. Lectures in applied mathematics (American Mathematical Society, 1974)
13. S. Chapman, T.G. Cowling, *The Mathematical Theory of Non-uniform Gases*, 2nd edn. (Cambridge University Press, 1952)
14. Y.H. Qian, D. D'Humières, P. Lallemand, Europhys. Lett. **17**(6), 479 (1992)
15. X. He, L.S. Luo, Phys. Rev. E **56**(6), 6811 (1997)
16. U. Frisch, B. Hasslacher, Y. Pomeau, Phys. Rev. Lett. **56**(14), 1505 (1986)
17. W.P. Yudistiawan, S.K. Kwak, D.V. Patil, S. Ansumali, Phys. Rev. E **82**(4), 046701 (2010)
18. E.M. Viggen, The lattice Boltzmann method: Fundamentals and acoustics. Ph.D. thesis, Norwegian University of Science and Technology (NTNU), Trondheim (2014)
19. I. Ginzburg, D. d'Humieres, A. Kuzmin, J. Stat. Phys. **139**, 1090 (2010)
20. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, 2001)
21. A.T. White, C.K. Chong, J. Comput. Phys. **230**(16), 6367 (2011)
22. S.K. Kang, Y.A. Hassan, J. Comput. Phys. **232**(1), 100 (2013)
23. D. d'Humières, M. Bouzidi, P. Lallemand, Phys. Rev. E **63**(6), 066702 (2001)
24. J. Tolke, M. Krafczyk, Int. J. Comp. Fluid Dynamics **22**(7), 443 (2008)
25. I. Karlin, P. Asinari, Physica A **389**(8), 1530 (2010)
26. S. Ubertini, S. Succi, Prog. Comput. Fluid Dynamics **5**(1/2), 85 (2005)
27. M.K. Misztal, A. Hernandez-Garcia, R. Matin, H.O. Sørensen, J. Mathiesen, Journal of Computational Physics **297**, 316 (2015)

28. S. Ubertini, P. Asinari, S. Succi, Phys. Rev. E **81**(1), 016311 (2010)
29. J. Hoffmann, *Numerical Methods for Engineers and Scientists* (McGraw-Hill, 1992)
30. H. Xi, G. Peng, S.H. Chou, Phys. Rev. E **59**(5), 6202 (1999)
31. O. Filippova, D. Hänel, J. Comput. Phys. **147**, 219 (1998)
32. S. Ubertini, S. Succi, Commun. Comput. Phys **3**, 342 (2008)
33. S. Ubertini, S. Succi, G. Bella, Phil. Trans. R. Soc. Lond. A **362**, 1763 (2004)
34. Phys. Rev. **94**, 511

Chapter 4
Analysis of the lattice Boltzmann equation

In Chapter 3 we derived the discrete numerical method known as the lattice Boltzmann (LB) equation from the Boltzmann equation, the latter being a fundamental equation from the field of kinetic theory. The motivation for doing so is that we wish to use the LB equation to simulate the behaviour of fluids. However, the LB method is a different approach to fluid simulation than standard Navier-Stokes solvers: While the latter start with a set of fluid conservation equations and discretise them, the LB method discretises another equation which means that it is less obvious that it works as a method for simulating fluids.

It has been known for around a century that the Boltzmann equation results in macroscopic behaviour corresponding to the well-known conservation equations of fluid mechanics [1]. Still, we also have to show that our discretised LB equation *also* macroscopically behaves according to these conservation equations.

The lattice Boltzmann method is a numerical scheme like many others, where it is important to verify both that the method tends to the correct equations (a condition called *consistency*), and that it is not susceptible to exponentially increasing errors (a condition called *stability*), at least in a certain range of parameters. (According to the Lax equivalence theorem, a method is *convergent* if these two conditions are met, which means that the numerical solution goes towards the 'true' solution as the simulation resolution is increased.)

In this chapter we will shed some light on these topics by analysing the LB equation in further depth. In Section 4.1 we apply a method known as the Chapman-Enskog analysis to determine the connection between the LB equation and the macroscopic equations of fluid mechanics, and Section 4.2 discusses additional important aspects of this analysis. Section 4.3 shows how important the choice of equilibrium f_i^{eq} is to the resulting macroscopic behaviour, and discusses a few specific alternatives including the widely-used equilibrium for incompressible flow. Section 4.4 covers the requirements for stability of the LB equation, *i.e.* the range of simulation parameters within which the LB equation will be safe from errors that grow exponentially fast. Section 4.5 examines factors, among others time and space discretisation errors, that separate LB solutions from the 'true' physical solution, and discusses how these factors may be minimised.

It should be pointed out that this chapter is oriented towards the theory of the LB equation, and that a thorough understanding of the contents of this chapter is not required for performing LB simulations.

4.1 The Chapman-Enskog analysis

Now that we have found the LB equation, we need to show that it *actually can be used* to simulate the behaviour of fluids. While we previously looked at the macroscopic behaviour of the undiscretised Boltzmann equation in section 1.3.4, and found that it behaves according to the continuity equation and a general Cauchy momentum equation with an unknown stress tensor, we have not yet seen that the latter specifically corresponds to the Navier-Stokes momentum equation. In this section we will show this correspondence for the LB equation using the most common method: the *Chapman-Enskog analysis*. Other methods are also available, and will be touched on in Section 4.2.4.

The analysis is named after Sydney Chapman (1888–1970) and David Enskog (1884–1947), two mathematical physicists from the UK and Sweden, respectively. In 1917, both independently developed similar methods of finding macroscopic equations from the Boltzmann equation with Boltzmann's original collision operator. In his book on the kinetic theory of gases, Chapman later combined the two approaches into what is now known as the Chapman-Enskog analysis [1].

In section 1.3.4 we saw that the assumption $f \simeq f^{\mathrm{eq}}$ results in the Euler momentum equation. Therefore, any macroscopic behaviour beyond the Euler equation must be connected to the *non-equilibrium* part of f, *i.e.* $f^{\mathrm{neq}} = f - f^{\mathrm{eq}}$. This is also true for a discretised velocity space: The general momentum equation in section 1.3.4 only depends on moments of f which are also equal to the corresponding moments of f_i. Throughout the rest of this section we shall assume that velocity space is discretised.

It is not obvious how to determine this nonequilibrium part, however. This is where the Chapman-Enskog analysis comes in. At its heart is a perturbation expansion of f_i around the equilibrium distribution f_i^{eq} with the Knudsen number Kn as the expansion parameter. Using the label ϵ^n to indicate terms of order Kn^n,[1] the expansion is

$$f_i = f_i^{\mathrm{eq}} + \epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)} + \dots \,. \tag{4.1}$$

(Throughout the literature, the equilibrium distribution f_i^{eq} is often written as $f_i^{(0)}$, giving a fully consistent notation in the expansion.)

[1] For instance, the relative order of ϵ shows us immediately in Eq. (4.1) that $f_i^{(2)}/f_i^{\mathrm{eq}} = O(\mathrm{Kn}^2)$. However, in the literature it is often stated that $\epsilon = \mathrm{Kn}$, unlike here where we treat it as a mere label. This is another possible approach to the expansion, where the Knudsen number is separated from the higher-order terms so that e.g. $f_i^{(2)}/f_i^{\mathrm{eq}} = O(1)$ while $\epsilon^2 f_i^{(2)}/f_i^{\mathrm{eq}} = O(\mathrm{Kn}^2)$. Which approach to use is simply a matter of taste, and the following equations in this section are the same in both cases.

Introducing the label ϵ lets us more easily group the terms according to their relative order in the Knudsen number. Central to this perturbation analysis is the concept that *in the perturbed equation, each order in* Kn *forms a semi-independent equation by itself*. As mentioned, the lowest-order terms in Kn give us the Euler momentum equation. Consequently, the higher-order terms may be seen as correction terms, analogously to how the viscous stress tensor in the Navier-Stokes equation may be seen as a correction term to the Euler equation. The perturbation must be performed in such a way that the equations at different orders in Kn still retain some tie to each other, so that the higher-order correction terms are still connected the lower-order equations.

In perturbation analyses, the perturbation terms at the two lowest orders together often result in a sufficiently accurate description of the system. We therefore make the ansatz that *only the two lowest orders in* Kn *are required to find the Navier-Stokes momentum equation*. Under this ansatz, we do not need to look closely at higher-order components of f_i than f_i^{eq} and $f_i^{(1)}$.

This derivation will be based on the lattice Boltzmann equation with the BGK collision operator,

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\tau}\left(f_i(\boldsymbol{x}, t) - f^{\text{eq}}(\boldsymbol{x}, t)\right). \tag{4.2}$$

While we could also perform the Chapman-Enskog analysis for a more general collision operator, we will use this BGK operator for its simplicity.

Like any collision operator, the BGK operator must conserve mass and momentum. As per Eq. (3.82), this conservation can be expressed as

$$\sum_i f_i^{\text{neq}} = \sum_i \boldsymbol{c}_i f_i^{\text{neq}} = 0. \tag{4.3}$$

We can express this for the expanded f_i in Eq. (4.1). In the literature, these are often called the *solvability conditions*. These can be further strengthened by the assumption that they hold individually at each order [1], *i.e.*

$$\sum_i f_i^{(n)} = \sum_i \boldsymbol{c}_i f_i^{(n)} = 0 \quad \text{for all } n \geq 1. \tag{4.4}$$

While this assumption simplifies the following derivation considerably, it can be done without and it is not always made in the literature, *e.g.* [2, 3, 4].

4.1.1 Taylor expansion, perturbation and separation

When the discrete-velocity Boltzmann equation in Eq. (3.60) was transformed into the LB equation by the discretisation of time and space, a first-order integral approximation was used.[2] By Taylor expanding the LB equation, which results in

$$\Delta t \left(\partial_t + c_{i\alpha}\partial_\alpha\right) f_i + \frac{\Delta t^2}{2}\left(\partial_t + c_{i\alpha}\partial_\alpha\right)^2 f_i + O\left(\Delta t^3\right) = -\frac{\Delta t}{\tau} f_i^{\mathrm{neq}}, \tag{4.5}$$

we obtain an equation which is continuous in time and space but which retains the discretisation error of the LB equation. Indeed, apart from the higher-order derivative terms

$$\sum_{n=2}^{\infty} \frac{\Delta t^n}{n!}\left(\partial_t + c_{i\alpha}\partial_\alpha\right)^n$$

on the left-hand side, the Taylor expanded LB equation is identical to the discrete-velocity Boltzmann equation in Eq. (3.60), from which the LB equation was discretised.

In the following, we will neglect the terms with third order derivatives or higher. The short explanation for this is that these terms tend to be very small and to not significantly affect the macroscopic behaviour. The longer justification, detailed in Appendix A.2.1, is that $\Delta t^n \left(\partial_t + c_{i\alpha}\partial_\alpha\right)^n f_i$ scales with $O(\mathrm{Kn}^n)$, and the terms at third order and higher can thus be neglected according to our ansatz that we only need the two lowest orders in the Knudsen number to find the Navier-Stokes momentum equation.

This assumes that the changes in f_i are slow, occurring only on a macroscopic scale. If *e.g.* numerical errors cause rapid changes in f_i, this assumption is no longer valid and the macroscopic equations that result from the Chapman-Enskog analysis are no longer suitable descriptions of the LBE's macroscopic behaviour.

We can get rid of the second order derivative terms in Eq. (4.5) by subtracting $(\Delta t/2)(\partial_t + c_{i\alpha}\partial_\alpha)$ applied to the equation itself. The resulting equation, which we will be basing the rest of this analysis on, is

$$\Delta t \left(\partial_t + c_{i\alpha}\partial_\alpha\right) f_i = -\frac{\Delta t}{\tau} f_i^{\mathrm{neq}} + \Delta t \left(\partial_t + c_{i\alpha}\partial_\alpha\right) \frac{\Delta t}{2\tau} f_i^{\mathrm{neq}}. \tag{4.6}$$

Here we have neglected the $O(\Delta t^3)$ terms. The f^{neq} derivative terms on the right-hand side is the only non-negligible remnant of the discretisation error.[3]

---

[2] However, we will see that the numerical scheme is still second-order accurate in the end. As we shall show, the first-order error terms only cause a change in the definition of the viscosity. The alternate second-order integral approximation causes no such change, but in many cases the two discretisations are interchangeable [5].

[3] With the second-order time and space discretisation described in Section 3.4, these terms would have been cancelled by terms from the Taylor expansion of the discretised BGK operator. At the two lowest orders in Kn, the discrete-velocity Boltzmann equation would be captured without error [6].

Before expanding f_i in Eq. (4.6) we make another ansatz: *It is also necessary to expand the time derivative into terms spanning several orders in* Kn.[4] Similarly labelling the spatial derivative without expanding it, the time and space derivatives become

$$\Delta t \partial_t f_i = \Delta t \left(\epsilon \partial_t^{(1)} f_i + \epsilon^2 \partial_t^{(2)} f_i + \ldots \right), \quad \Delta t c_{i\alpha} \partial_\alpha f_i = \Delta t \left(\epsilon c_{i\alpha} \partial_\alpha^{(1)} \right) f_i. \tag{4.7}$$

These different components of ∂_t at various orders in Kn should not themselves be considered time derivatives [1]; instead, they are the terms at different orders in Kn that when summed are equal to the time derivative.[5] This is shown by an example in section 4.2.

If we apply both the expansion of f_i from Eq. (4.1) and the derivative expansion from Eq. (4.7) to Eq. (4.6) and separate the equation into terms of different order in Kn, we find

$$O(\epsilon): \qquad \left(\partial_t^{(1)} + c_{i\alpha} \partial_\alpha^{(1)} \right) f_i^{\mathrm{eq}} = -\frac{1}{\tau} f_i^{(1)}, \tag{4.8a}$$

$$O(\epsilon^2): \qquad \partial_t^{(2)} f_i^{\mathrm{eq}} + \left(\partial_t^{(1)} + c_{i\alpha} \partial_\alpha^{(1)} \right) \left(1 - \frac{\Delta t}{2\tau} \right) f_i^{(1)} = -\frac{1}{\tau} f_i^{(2)} \tag{4.8b}$$

for the two lowest orders in Kn.[6]

4.1.2 Moments and recombination

Taking the zeroth to second moment of Eq. (4.8a) (i.e. multiplying by 1, $c_{i\alpha}$, and $c_{i\alpha} c_{i\beta}$, respectively, and then summing over i), we can find the $O(\epsilon)$ moment equations:

$$\partial_t^{(1)} \rho + \partial_\gamma^{(1)} \left(\rho u_\gamma \right) = 0, \tag{4.9a}$$

$$\partial_t^{(1)} (\rho u_\alpha) + \partial_\beta^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} = 0, \tag{4.9b}$$

$$\partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} + \partial_\gamma^{(1)} \Pi_{\alpha\beta\gamma}^{\mathrm{eq}} = -\frac{1}{\tau} \Pi_{\alpha\beta}^{(1)}, \tag{4.9c}$$

---

[4] Given the assumptions made in this derivation, this expansion should be done even in the steady-state where $\partial_t f_i = 0$. However, it is possible to make fewer assumptions so that the time derivative expansion is not necessary. This, and the steady-state case in general, is discussed in more detail in Section 4.2.3.

[5] Such derivative expansions, often called *multiple-scale expansions*, are also used in general perturbation theory to deal with expansions that otherwise result in terms which grow without bound at one order but is cancelled by similar terms at later orders [7]. We will get back to the interpretation of this expansion in Section 4.2.2.

[6] The parenthesis $(1 - \Delta t/2\tau)$ in Eq. (4.8b) is the only remnant of the space and time discretisation error terms on the right-hand side of Eq. (4.6). If we had based this analysis directly on the discrete-velocity Boltzmann equation in Eq. (3.60), where there is no such discretisation error, the $-\Delta t/2\tau$ term would not have been present here.

which contain the moments

$$\Pi^{\mathrm{eq}}_{\alpha\beta} = \sum_i c_{i\alpha} c_{i\beta} f_i^{\mathrm{eq}} = \rho u_\alpha u_\beta + \rho c_{\mathrm{s}}^2 \delta_{\alpha\beta}, \tag{4.10a}$$

$$\Pi^{\mathrm{eq}}_{\alpha\beta\gamma} = \sum_i c_{i\alpha} c_{i\beta} c_{i\gamma} f_i^{\mathrm{eq}} = \rho c_{\mathrm{s}}^2 \left(u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\alpha\gamma} + u_\gamma \delta_{\alpha\beta} \right), \tag{4.10b}$$

$$\Pi^{(1)}_{\alpha\beta} = \sum_i c_{i\alpha} c_{i\beta} f_i^{(1)}. \tag{4.10c}$$

The first two of these moments are known from Eq. (3.69), while the third is unknown as of yet. Note that the third moment $\Pi^{\mathrm{eq}}_{\alpha\beta\gamma}$ is lacking a term $\rho u_\alpha u_\beta u_\gamma$ as f_i^{eq} only contains terms of up to $O(u^2)$. If we take the moment equations in Eq. (4.9) and reverse the expansions from Eq. (4.7) under the assumption that $\partial_t f_i \approx \epsilon \partial_t^{(1)} f_i$ (*i.e.* neglecting the contributions of higher-order terms in Kn), the first equation is the continuity equation while the second equation is the Euler equation. We will return to the third equation shortly.

Similarly taking the zeroth and first moment of Eq. (4.8b), we can find the $O(\mathrm{Kn}^2)$ moment equations:

$$\partial_t^{(2)} \rho = 0, \tag{4.11a}$$

$$\partial_t^{(2)} (\rho u_\alpha) + \partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau} \right) \Pi^{(1)}_{\alpha\beta} = 0. \tag{4.11b}$$

These equations can both be interpreted as $O(\epsilon^2)$ corrections to the $O(\epsilon)$ equations above. The continuity equation is exact already at $O(\epsilon)$, so we might have expected that the $O(\epsilon^2)$ correction is zero. However, the $O(\epsilon^2)$ correction to the Euler momentum equation is non-zero, though given by the as of yet unknown moment $\Pi^{(1)}_{\alpha\beta}$.

Assembling the mass and momentum equations from their $O(\epsilon)$ and $O(\epsilon^2)$ component equations in Eq. (4.9) and Eq. (4.11) respectively, we find

$$\left(\epsilon \partial_t^{(1)} + \epsilon^2 \partial_t^{(2)} \right) \rho + \epsilon \partial_\gamma^{(1)} (\rho u_\gamma) = 0, \tag{4.12a}$$

$$\left(\epsilon \partial_t^{(1)} + \epsilon^2 \partial_t^{(2)} \right) (\rho u_\alpha) + \epsilon \partial_\beta^{(1)} \Pi^{\mathrm{eq}}_{\alpha\beta} = -\epsilon^2 \partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau} \right) \Pi^{(1)}_{\alpha\beta}. \tag{4.12b}$$

Reversing the derivative expansions of Eq. (4.7), these equations become the continuity equation and a momentum conservation equation with an as of yet unknown viscous stress tensor

$$\sigma'_{\alpha\beta} = -\left(1 - \frac{\Delta t}{2\tau} \right) \Pi^{(1)}_{\alpha\beta}. \tag{4.13}$$

Note that the recombination of the two different orders in ϵ is done through the expanded time derivative. Indeed, without expanding the time derivative across mul-

tiple orders in Kn as in Eq. (4.7), Eq. (4.11b) would be missing its $\partial_t^{(2)}(\rho u_\alpha)$ term, and would consequently be mistakenly predicting $\partial_\beta^{(1)} \Pi_{\alpha\beta}^{(1)} = 0$.[7]

The final piece of the puzzle is finding an explicit expression for the perturbation moment $\Pi_{\alpha\beta}^{(1)}$. This can be directly found from derivatives of the equilibrium moments using Eq. (4.9c). Doing so requires some amount of algebra which has been relegated to the Appendix section A.2.2. For the isothermal equation of state and the equilibrium distribution f_i^{eq} expanded only to $O(u^2)$, the end result is

$$\Pi_{\alpha\beta}^{(1)} = -\rho c_s^2 \tau \left(\partial_\beta^{(1)} u_\alpha + \partial_\alpha^{(1)} u_\beta \right) + \tau \partial_\gamma^{(1)} \left(\rho u_\alpha u_\beta u_\gamma \right). \tag{4.14}$$

As we do not at any point in the derivation assume that $\partial_t \tau = 0$ or $\partial_\alpha \tau = 0$, this holds even if τ is a function of space and time. TODO (EMV): Add cross-reference if we talk about non-Newtonian fluids. Of the two terms on the right-hand side, the first corresponds to a Navier-Stokes viscous stress tensor, while the second is an error term stemming from the lack of a correct $O(u^3)$ term in the equilibrium distribution f_i^{eq}.

The error term is negligible in most cases. From a closer look at the magnitudes of the two terms, we find that the $O(u^3)$ error term can be neglected if $u^2 \ll c_s^2$, which is equivalent to the condition $\text{Ma}^2 \ll 1$ on the Mach number $\text{Ma} = u/c_s$ [8].

For this reason, it is often stated in the literature (*e.g.* [9, 10]) that the lattice Boltzmann method is only valid for *weakly compressible* phenomena, as opposed to the *strongly compressible* phenomena which occur when Ma goes towards unity and beyond [11].[8]

4.1.3 Macroscopic equations

We now have all the pieces to determine the macroscopic equations simulated by the lattice Boltzmann equation. Inserting Eq. (4.14) (neglecting the $O(u^3)$ error term) into Eq. (4.12) and reversing the derivative expansion from Eq. (4.7), we finally find the continuity equation as in Eq. (1.3) and the Navier-Stokes momentum equation from Eq. (1.16):

$$\partial_t \rho + \partial_\gamma \left(\rho u_\gamma \right) = 0, \tag{4.15a}$$

$$\partial_t (\rho u_\alpha) + \partial_\beta \left(\rho u_\alpha u_\beta \right) = -\partial_\alpha p + \partial_\beta \left[\eta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right) \right], \tag{4.15b}$$

with

[7] As explained in section 4.2.3, it is possible to do without the time derivative expansion; however, this invalidates some simplifying assumptions taken in this derivation.

[8] Sound propagation can generally be considered a weakly compressible phenomenon; sound waves cause the pressure p, the density ρ, and the fluid velocity $\boldsymbol{u}$ to all fluctuate, though these fluctuations are weak enough that $\text{Ma} = u/c_s \ll 1$ for all sounds within the range of normal human experience [12].

$$p = \rho c_s^2, \quad \eta = \rho c_s^2 \left(\frac{\tau}{\Delta t} - \frac{1}{2} \right) \Delta t, \quad \eta_B = \frac{2}{3}\eta. \tag{4.16}$$

The bulk viscosity η_B is not itself directly visible in Eq. (4.15b), but follows from comparison with Eq. (1.16).

In monatomic kinetic theory, the bulk viscosity η_B is normally found to be zero, while here we have found it to be on the order of the shear viscosity η. This difference is caused by the use of the isothermal equation of state [13], which is fundamentally incompatible with the monatomic assumption.[9]

We could also have found the same macroscopic equations as above using more general collision operators than the BGK operator. The only time that τ comes into play in the above derivation is as the relaxation time of the moment $\Pi_{\alpha\beta}$ in Eq. (4.9c). In the more general Multiple Relaxation Time (MRT) collision operators which we will take a closer look at in chapter 10, each moment can relax to equilibrium at a different rate. Good choices of these various relaxation times will increase the stability and accuracy of LB simulations.[10]

4.2 Discussion of the Chapman-Enskog analysis

Despite its long history, the Chapman-Enskog analysis is still subject to debate and is generally difficult to digest and understand in its fundamental meaning [15]. Therefore we take a brief look at some implications of the Chapman-Enskog analysis and its alternatives.

As this section discusses the details of Section 4.1, it is necessarily somewhat more complex. However, first-time readers may safely skip this entire section as it is not necessary for a basic understanding of the Chapman-Enskog analysis.

4.2.1 Dependence of velocity moments

The Hermite polynomial approach taken in section 3.3 yields velocity sets $\boldsymbol{c}_i$ and equilibrium functions f_i^{eq} so that the zeroth to second order moments Π^{eq}, Π_α^{eq}, and $\Pi_{\alpha\beta}^{\mathrm{eq}}$ of f_i^{eq} equal those of the continuous-velocity distribution f^{eq}.

However, the third order moment $\Pi_{\alpha\beta\gamma}^{\mathrm{eq}}$ is not fully correct, leading to the $\rho u_\alpha u_\beta u_\gamma$ error term in Eq. (4.14) that further leads to a similar error term in the macroscopic momentum equation. It is not difficult to see why it *cannot be* for

[9] In fact, by imposing in the analysis the isentropic equation of state $p/p_0 = (\rho/\rho_0)^\gamma$ previously described in section 1.1.3, we would find a bulk viscosity $\eta_B = \eta(5/3 - \gamma)$ [6]. In the monatomic limit $\gamma = 5/3$ we find $\eta_B = 0$, while in the isothermal limit $\gamma = 1$ we find $\eta_B = 2\eta/3$. (How to impose other equations of state is described in Section 4.3.3.)

[10] It is also possible to alter the shear and bulk viscosity separately through a more complex relaxation of the $\Pi_{\alpha\beta}$ moment [14]. TODO (EMV): Refer to the MRT chapter instead.

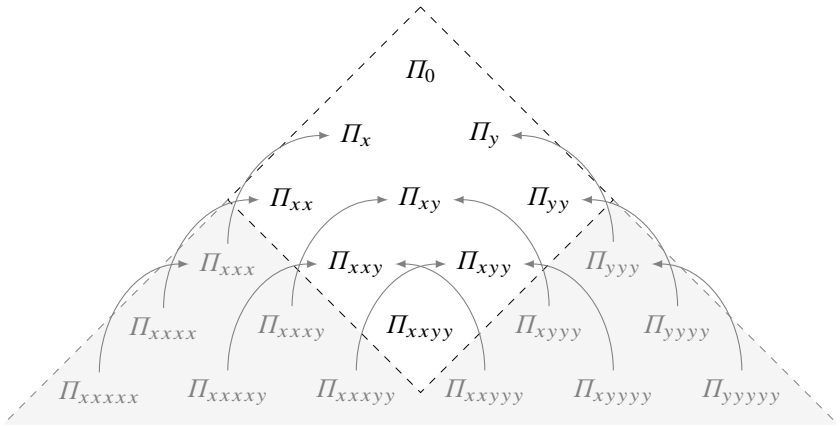


Fig. 4.1: Dependency map of D2Q9 velocity moments. Higher-order moments (dark on light grey) depend on nine lower-order independent moments (black on white).

any of the velocity sets given in Tab. 3.1: In all of these velocity sets, we find that $c_{i\alpha} \in \{-1, 0, +1\}$, so that $c_{i\alpha}^3 = c_{i\alpha}$. Consequently,

$$\Pi_{xxx}^{\mathrm{eq}} = \sum_i c_{ix}^3 f_i^{\mathrm{eq}} = \sum_i c_{ix} f_i^{\mathrm{eq}} = \Pi_x^{\mathrm{eq}}, \tag{4.17}$$

i.e. the third order xxx, yyy, and zzz moments are equal to the first order x, y, and z moments, respectively.

Exercise 4.1. From the general forms of Π_α^{eq} and $\Pi_{\alpha\beta\gamma}^{\mathrm{eq}}$ in Eq. (3.69), show that $\Pi_{xxx}^{\mathrm{eq}} = \Pi_x^{\mathrm{eq}}$.

Other higher-order moments are also equal to lower-order ones, though the specific equalities vary between velocity sets. For the D2Q9 velocity set, we can find from the fact that $c_{i\alpha}^3 = c_{i\alpha}$ that there are only nine independent moments, as shown in Fig. 4.1.[11]

Exercise 4.2. Show the dependencies given in Fig. 4.1 by equations like Eq. (4.17).

Eq. (4.17) shows us that at least some of the third order moments are equal to first order moments for all the velocity sets that we have examined previously. Consequently, it is not possible for these velocity sets to correctly recover the $\rho u_\alpha u_\beta u_\gamma$ term in $\Pi_{\alpha\beta\gamma}^{\mathrm{eq}}$ and thus avoid the $O(u^3)$ error term in Eq. (4.14) which carries over to the macroscopic stress tensor. Additionally, these velocity sets are insufficient for some of the thermal models which will be discussed in chapter 8, as these require

[11] In general, for a velocity set with q velocities there are q different independent moments, as we shall see in chapter 10.

independent third-order moments. In cases where these issues cannot be safely ignored, extended velocity sets with more velocities and higher-order equilibrium distributions must be used.[12] These can be found using the same Hermite approach as in Section 3.3.

Exercise 4.3. Consider a rotated and rescaled D2Q9 velocity set consisting of the zero-velocity vector $(0,0)$, four short velocity vectors $(\pm 1/2, \pm 1/2)$ and four long velocity vectors $(\pm 1, 0)$, $(0, \pm 1)$. Show that the nine moments Π, Π_x, Π_y, Π_{xx}, Π_{xy}, Π_{yy}, Π_{xxx}, Π_{yyy}, and Π_{xxyy} are independent of each other. Show also that $\Pi_{xxy} = \Pi_{xyy} = \frac{1}{2}\Pi_{xy}$.

4.2.2 The time scale interpretation

In the Chapman-Enskog analysis presented in Section 4.1, the time derivative ∂_t was expanded into terms $\epsilon^n \partial_t^{(n)}$ at different orders of smallness in Eq. (4.7). This decomposition is sometimes interpreted as a decomposition into different *time scales*, *i.e.* 'clocks ticking at different speeds'. This interpretation may in some cases lead to false conclusions. Whenever the link between different orders in $f_i^{(n)}$ is broken by the strengthened solvability conditions in Eq. (4.4), it is not difficult to show that the time scale interpretation leads to such a false conclusion.

From this interpretation, one would expect for steady-state that when the time derivative $\partial_t g$ of a function g vanishes, all 'time scale' derivatives $\partial_t^{(n)} g$ can be removed as well. After all, 'steady-state' means that nothing can change on *any* time scale.

Let's look at a steady Poiseuille flow. On the one hand, we know macroscopically that a Poiseuille flow obeys

$$\boldsymbol{\nabla} p = \boldsymbol{\nabla} \cdot \boldsymbol{\sigma}, \tag{4.18}$$

i.e. the flow is driven by a pressure gradient which is exactly balanced by the divergence of the viscous stress tensor $\boldsymbol{\sigma}$. On the other hand, eq. (4.9) and eq. (4.11) describe relations for the momentum on the ϵ- and ϵ^2-scales:

$$\partial_t^{(1)}(\rho u_\alpha) = -\partial_\beta^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}}, \qquad \partial_t^{(2)}(\rho u_\alpha) = -\partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau}\right) \Pi_{\alpha\beta}^{(1)}. \tag{4.19}$$

Obviously, $\partial_t(\rho u_\alpha) = 0$ holds for steady Poiseuille flow. From the time scale interpretation, $\partial_t^{(1)}(\rho u_\alpha)$ and $\partial_t^{(2)}(\rho u_\alpha)$ must also vanish independently, so that Eq. (4.19) becomes

$$\partial_\beta^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} = 0, \qquad \partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau}\right) \Pi_{\alpha\beta}^{(1)} = 0. \tag{4.20}$$

From Equations (4.10a) and (4.13) we know that

[12] Alternatively, it has also been proposed to add correction terms to the lattice Boltzmann equation to counteract these errors [16].

$$\Pi^{\text{eq}}_{\alpha\beta} = \rho u_\alpha u_\beta + p\delta_{\alpha\beta}, \qquad \left(1 - \frac{\Delta t}{2\tau}\right)\Pi^{(1)}_{\alpha\beta} = -\sigma_{\alpha\beta},$$

and thus Eq. (4.20) states

$$\partial^{(1)}_\alpha p = 0, \qquad \partial^{(1)}_\beta \sigma_{\alpha\beta} = 0,$$

i.e. no flow at all!

In other words: given Eq. (4.19), Poiseuille flow would be impossible if $\partial^{(1)}_t(\rho u_\alpha)$ and $\partial^{(2)}_t(\rho u_\alpha)$ vanished independently. It is therefore misleading to say that time itself is decomposed into different scales. It is rather the term $\partial_t g$ which is decomposed into contributions from different orders in the perturbation expansion, contributions which are themselves not time derivatives [1]. Only the *sum* of these contributions vanishes, *i.e.*

$$\partial_t(\rho u_\alpha) = 0 \quad \Longrightarrow \quad \partial^{(1)}_t(\rho u_\alpha) + \partial^{(2)}_t(\rho u_\alpha) = 0 \quad \Longrightarrow \quad \partial^{(1)}_\alpha p = \partial^{(1)}_\beta \sigma_{\alpha\beta},$$

neglecting terms at $O(\epsilon^3)$ and higher orders.

In this section we have seen that setting $\partial^{(n)}_t f_i = 0$ leads to false conclusions in the Chapman-Enskog analysis. It must be pointed out that the reason that this does not work is the simplifying (but not, strictly speaking, necessary) assumption that leads from Eq. (4.3) to Eq. (4.4). Without this assumption we would not get zeroes on the right-hand side of Eq. (4.20). We will discuss this assumption further at the end of Section 4.2.3.

4.2.3 Chapman-Enskog analysis for steady flow

The Chapman-Enskog analysis performed in Section 4.1 depends on the expansion of the time derivative into components, each component at a different order in ϵ. The expanded time derivative gives us an unique way to reconnect the equations at different orders in ϵ after having performed the perturbation expansion.

For a time-invariant case with $\partial_t f_i = 0$, this approach is still valid; there is nothing wrong with keeping the time derivative and expanding it even though it is equal to zero. (Indeed, we saw in Section 4.2.2 that setting it to zero before expanding it may lead to false conclusions.) However, if we take the final time-dependent macroscopic equations from the Chapman-Enskog analysis in Section 4.1 and set the time derivative to zero, the resulting steady-state macroscopic equations are somewhat misleading. As we will soon see, the time-invariant stress tensor can be expressed differently to the time-variant one.

In a time-invariant case, the expanded time derivative is

$$\Delta t \partial_t f_i = \Delta t \left(\epsilon \partial^{(1)}_t f_i + \epsilon^2 \partial^{(2)}_t f_i + \ldots\right) = 0, \quad \Rightarrow \quad \epsilon \partial^{(1)}_t f_i = O(\epsilon^2). \tag{4.21}$$

Thus, $\partial_t^{(1)} f_i$ increases by one order in smallness in this case. Consequently, the mass and momentum moment equations in Eq. (4.9) become

$$\partial_\gamma^{(1)} \left(\rho u_\gamma\right) = -\partial_t^{(1)} \rho = O(\epsilon), \tag{4.22a}$$

$$\partial_\beta^{(1)} \left(\rho c_s^2 \delta_{\alpha\beta} + \rho u_\alpha u_\beta\right) = -\partial_t^{(1)} \left(\rho u_\alpha\right) = O(\epsilon). \tag{4.22b}$$

Thus, the left-hand sides in these subequations are one order higher in ϵ than they would be in a time-variant case. Similarly, the second-order moment equation in Eq. (4.9c), which gives us the viscous stress tensor through Eq. (4.13), becomes

$$\Pi_{\alpha\beta}^{(1)} = -\tau \left(\partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} + \partial_\gamma^{(1)} \Pi_{\alpha\beta\gamma}^{\mathrm{eq}}\right) = -\tau \partial_\gamma^{(1)} \Pi_{\alpha\beta\gamma}^{\mathrm{eq}} + O(\epsilon). \tag{4.23}$$

By applying Eq. (A.13) and Eq. (4.22a), this becomes

$$\Pi_{\alpha\beta}^{(1)} = -\tau c_s^2 \left(\partial_\beta^{(1)} (\rho u_\alpha) + \partial_\alpha^{(1)} (\rho u_\beta)\right) + O(\epsilon) \tag{4.24}$$

Finally we can insert the above equations into Eq. (4.12) to find the steady macroscopic equations exactly predicted by the Chapman-Enskog analysis of the LBE:

$$\partial_\gamma \left(\rho u_\gamma\right) = 0, \tag{4.25a}$$

$$\partial_\beta \left(\rho u_\alpha u_\beta\right) = -\partial_\alpha p + \partial_\beta \nu \left[\partial_\beta (\rho u_\alpha) + \partial_\alpha (\rho u_\beta)\right]. \tag{4.25b}$$

with pressure $p = \rho c_s^2$, kinematic shear viscosity $\nu = c_s^2(\tau/\Delta t - 1/2)\Delta t$, and kinematic bulk viscosity $\nu_B = 2\nu/3$. This steady momentum equation lacks the $O(u^3)$ error term of the unsteady momentum equation. Instead, its stress tensor contains the gradients

$$\partial_\beta (\rho u_\alpha) + \partial_\alpha (\rho u_\beta) = \rho \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right) + \left(u_\alpha \partial_\beta + u_\beta \partial_\alpha\right) \rho$$

instead of the gradients $\rho \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right)$ found in the correct Navier-Stokes equation. However, since $\partial_\alpha \rho = O(u^2)$ for steady flow at $\mathrm{Ma} \ll 1$ [11], the error in this macroscopic momentum equation remains at $O(u^3)$. (For a particular LBE variant designed for incompressible flow, this error disappears so that its momentum equation is error-free at steady-state. This variant is described in Section 4.3.2.)

Exercise 4.4. Show that it is also possible to find the steady perturbation moment in Eq. (4.24) directly from the unsteady perturbation moment in Eq. (4.14). *Hint: You will need to use Eq. (A.12) along with Eq. (4.22).*

It must be pointed out that this approach to the steady Chapman-Enskog analysis is not the only possible one. It is also possible to perform a steady analysis if we assume time invariance initially by setting $\partial_t f_i = 0$ *a priori* without expanding it. However, without the expanded time derivative, we must link the equations at different order in ϵ in a different way. This can be done through the solvability

conditions in Eq. (4.3), which can be expanded as

$$\begin{aligned} \sum_i f_i^{\text{neq}} &= \sum_i \left(\epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)} + \ldots\right) = 0, \\ \sum_i \boldsymbol{c}_i f_i^{\text{neq}} &= \sum_i \boldsymbol{c}_i \left(\epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)} + \ldots\right) = 0. \end{aligned} \tag{4.26}$$

This approach is incompatible with the additional simplifying assumption made in Eq. (4.4), which breaks the link between orders by assuming that the solvability conditions hold individually at each order. Regardless, this approach results in the same steady macroscopic equations as the approach taken above without any loss of generality.

TODO (GS): Add a reference to some publication that takes this approach.

4.2.4 Alternate multi-scale methods

While the Chapman-Enskog analysis is the classical multi-scale tool to link the LB equation and its resulting macroscopic equations [], alternative mathematical techniques exist. Since their detailed analysis is far outside the scope of this book, this section only provides a brief list of the most relevant examples, with further references for interested readers.

Similarly to the Chapman-Enskog expansion, most of these alternate multi-scale methods had their origins in kinetic theory and were later adapted to the LB field. Below, we illustrate three approaches that have followed this route.

The *asymptotic expansion* technique was pioneered by Sone [] in the study of solutions of the Boltzmann equation at small Knudsen and finite Reynolds numbers. Such a technique diverts from the classical Chapman-Enskog analysis in two main aspects. First, the macroscopic solutions are formally expanded in a series of small Knudsen number, similarly to the mesoscopic variables. Second, it restricts the analysis of the dynamical processes to the diffusive scaling, *i.e.* $\Delta t \propto \Delta x^2 \sim \epsilon^2$ (where ϵ is some smallness parameter used in the expansion), based on the argument that the target solution belongs to the incompressible NSEs. The asymptotic analysis was introduced in the LB field by [] [Inamuro], and later it experienced further developments in [] [Junk]. After the Chapman-Enskog analysis, this technique is possibly the one with the broadest acceptance in the LB field.

The *Hermite expansion series* technique was introduced by Grad [] as a way to approximate the solution of the Boltzmann equation in terms of a finite set of Hermite polynomials. Such a representation provides a convenient closure for describing the fluxes in the macroscopic laws. The idea is that the truncation order of the Hermite expansion of the mesoscopic variable determines the range of validity of the resulting macroscopic fluid system. This technique differs from the Chapman-Enskog analysis as it recovers the governing macroscopic dynamics in a non-perturbative way. The Hermite expansion series was introduced in the

LB field by [] [Shan], and subsequently further developed by [] [Shan,ShanPRE, Malaspinas].

The *Maxwellian iteration* technique was proposed by Ikenberry and Truesdell [] as a systematic procedure to find a closure for the fluxes in the governing macroscopic flow equations. The idea is to work with an hierarchy of moments of the mesoscopic variable, which enables completing the missing information from the lower-order moments using higher-order ones. In order to assess whether the higher-order terms in this hierarchy should be neglected or retained, this process is supplemented by an order of magnitude analysis. Therefore, this technique employs elements from both the Grad expansion method and the Chapman-Enskog analysis. In the LB field its use was proposed by [] [Asinari, Bennet], exploiting the fact that it applies rather naturally to LB models working with the MRT collision operator.

The inspection of the macroscopic behaviour of LB models is also possible through more numerically oriented procedures. Even though they may introduce a higher level of abstraction, they have the advantage of providing access to the macroscopic information directly by Taylor expanding the discrete numerical system, without resorting to any results from kinetic theory. There are three main approaches. First, the *equivalent equation method* [] [Dubois], where the Taylor expansion method is applied directly to the LB equation. Second, the method proposed by [] [Holdych] that, while following the same Taylor expansion procedure, applies it to a *recursive representation* of the LB equation. Third, the approach proposed by [] [Ginzburg] where the LB equation is first written in *recurrence form*, *i.e.* in terms of finite-difference stencils, with the macroscopic information recovered order by order by Taylor expanding the finite-difference operators.

The number of existing LB multi-scale approaches has led to debate about their equivalency, *e.g.* [] [Wagner,Daniel]. The work [] [Caiazzo,Rheinländer] contributed to resolving this question. It showed that the Chapman-Enskog analysis is nothing but a particular example of a general expansion procedure, which also encompasses many other multi-scale methods such as the asymptotic expansion technique []. As general conclusion, this study showed that, providing the requirements of such a general expansion procedure are fulfilled, then distinctive multi-scale methods offer identical consistency information, possibly only differing in higher-order terms. NOTE (EMV): This last sentence is too long and convoluted.

4.3 Alternate-equilibrium models

In Chapter 3, we derived the discrete equilibrium distribution f_i^{eq} *via* an expansion of the continuous Maxwell-Boltzmann distribution f^{eq} in Hermite polynomials. We have also mentioned the simpler but less physically illuminating method of Taylor expanding f^{eq} to find f_i^{eq}.

However, a more heuristic approach can be taken [17, 18, 19]: Instead of deriving f_i^{eq} directly from f^{eq}, a more general equilibrium distribution is chosen by ansatz as *e.g.*

$$f_i^{\mathrm{eq}} = w_i \rho \left(1 + a_1 c_{i\alpha} u_\alpha + a_2 c_{i\alpha} c_{i\beta} u_\alpha u_\beta - a_3 u_\alpha u_\alpha\right). \tag{4.27}$$

Here, a_1, a_2, and a_3 are constants that may be kept throughout the Chapman-Enskog analysis. The resulting macroscopic equations will then be functions of these constants, which may subsequently be chosen in order to get the desired macroscopic equations.

These macroscopic equations are not limited to the compressible flow case we have looked at previously in this chapter; in fact, this approach can be used to derive LBEs to solve a much more general class of PDEs. Indeed, this goes to show that a great deal of the physics of the LBE is determined by the choice of the equilibrium distribution f_i^{eq}. As a simple example of this, the subtle change of imposing a given flow field $\boldsymbol{u}$ in the basic LB equilibrium of Eq. (3.56) instead of taking $\boldsymbol{u}$ from the populations f_i, results in macroscopic behaviour according to the advection-diffusion equation.[13] We will cover this more in depth in Chapter 8.

However, it is not always necessary to derive alternate equilibria through this elaborate process. In some cases, the standard equilibrium in Eq. (3.56) may be altered directly in order to derive alternate models.

In this section, we will describe in some depth a few different models where the equilibrium distribution is changed in order to change the macroscopic behaviour. We will then refer to other such models in the literature.

4.3.1 Linearised fluid flow

The research field of acoustics is based almost entirely on linearised versions of the conservation equations [12], where we assume that the fluid quantities are small variations around a constant rest state,

$$\begin{aligned} \rho(\boldsymbol{x}, t) &= \rho_0 + \rho'(\boldsymbol{x}, t), \\ p(\boldsymbol{x}, t) &= p_0 + p'(\boldsymbol{x}, t), \\ \boldsymbol{u}(\boldsymbol{x}, t) &= \boldsymbol{0} + \boldsymbol{u}'(\boldsymbol{x}, t). \end{aligned} \tag{4.28}$$

The rest state constants are labeled with subscripted zeros, while the primed variables represent deviations from this rest state. Terms which are nonlinear in these deviations are neglected.[14] This approximation corresponds *e.g.* to creeping flows where viscosity dominates over advection, or to cases of linear (*i.e.* low-Ma) sound propagation.

---

[13] Breaking the link between f_i and $\boldsymbol{u}$ in this way also breaks the conservation of momentum in collisions, as $\sum_i \boldsymbol{c}_i f_i \neq \sum_i \boldsymbol{c}_i f_i^{\mathrm{eq}}$. Thus, this advection-diffusion model is an example of an alternate LBE where the physical meaning of f_i has changed.

[14] The advection term in Navier-Stokes is one such nonlinear term. By neglecting it, we break the Galilean invariance of the fluid model. To avoid this issue, we can instead linearise the velocity around a non-zero rest state velocity as necessary.

TODO (EMV): Later, add to this discussion the fact that we can subtract the rest state distribution $w_i\rho_0$ to get the distribution fluctiation $f_i'^{\text{eq}}$. Make test code showing that it's safe to only track the f_i' fluctuation and give it to Alex to look at.

By similarly neglecting nonlinear deviations from equilibrium in Eq. (3.56), we find a *linearised* equilibrium distribution,

$$f_i^{\text{eq}} = w_i \left(\rho + \rho_0 \frac{c_{i\alpha} u_\alpha}{c_{\text{s}}^2} \right), \tag{4.29}$$

ρ_0 being the rest state density. From this equilibrium distribution and the isotropy conditions in Eq. (3.62), we can directly find the equilibrium moments

$$\begin{gathered} \Pi^{\text{eq}} = \rho, \qquad \Pi_\alpha^{\text{eq}} = \rho_0 u_\alpha, \qquad \Pi_{\alpha\beta}^{\text{eq}} = \rho c_0^2 \delta_{\alpha\beta}, \\ \Pi_{\alpha\beta\gamma}^{\text{eq}} = \rho_0 c_0^2 \left(u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\alpha\gamma} + u_\gamma \delta_{\alpha\beta} \right). \end{gathered} \tag{4.30}$$

Exercise 4.5. Confirm these moments by recalculating them.

Performing the Chapman-Enskog analysis as in Section 4.1 with this simplified equilibrium distribution results in the macroscopic mass and momentum conservation equations

$$\begin{aligned} \partial_t \rho + \rho_0 \partial_\alpha u_\alpha &= 0, \\ \rho_0 \partial_t u_\alpha &= -\partial_\alpha p + \eta \partial_\beta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right), \end{aligned} \tag{4.31}$$

with pressure $p = \rho c_{\text{s}}^2$ and dynamic shear viscosity $\eta = \rho_0 c_{\text{s}}^2 (\tau/\Delta t - 1/2)\Delta t$. By comparison with Eq. (4.15), we find that these are perfectly linearised versions of the equations that we get with the normal nonlinear equilibrium distribution f_i^{eq}. Interestingly, since this linear model drops any nonlinearity, it is *not* affected by the nonlinear $O(u^3)$ error terms in the momentum equation, or indeed any other error terms, unlike the standard model we have looked at previously.

Exercise 4.6. Find these macroscopic equations through a Chapman-Enskog analysis of the LBE with linearised equilibrium. (The linearised equilibrium makes the analysis significantly simpler!)

This linearised equilibrium has been used to ensure full linearity in LB acoustics simulations so that sound waves may be simulated on complex exponential form in which a plane wave would be represented as as varying with *e.g.* $\text{e}^{\text{i}(\omega t - kx)}$ [20, 6]. We will come back to this technique in Chapter 12.1.2.

In the case of steady flow, these linearised equations become the equations of *Stokes flow*, *i.e.* of viscosity-dominated steady flow at $\text{Re} \ll 1$:

$$\begin{aligned} \partial_\alpha u_\alpha &= 0, \\ \eta \partial_\beta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right) - \partial_\alpha p &= 0. \end{aligned} \tag{4.32}$$

4.3.2 Incompressible flow

In the linearised model described in Section 4.3.1, we assumed that the density ρ, the pressure p and the fluid velocity $\boldsymbol{u}$ deviated only very little from a constant rest state. This assumption does not hold for flows at larger Reynolds numbers.

However, the deviation of *density* from a rest state may be smaller than that of the *velocity*. Indeed, it is known for steady flow at $\mathrm{Ma} \ll 1$ that $\rho'/\rho_0 = O(\mathrm{Ma}^2)$ [11].[15] In this limit, we may derive a LBE that approximates incompressibility by neglecting terms in f_i^{eq} that are above $O(\mathrm{Ma}^2)$:

$$f_i^{\mathrm{eq}} = w_i \rho + w_i \rho_0 \left[\frac{c_{i\alpha} u_\alpha}{c_\mathrm{s}^2} + \frac{u_\alpha u_\beta \left(c_{i\alpha} c_{i\beta} - c_\mathrm{s}^2 \delta_{\alpha\beta} \right)}{2 c_\mathrm{s}^4} \right]. \tag{4.33}$$

Such equilibria were proposed in the literature early in the history of the LBM [21, 22]. The moments of this particular equilibrium can be found using Eq. (3.62) to be

$$\begin{gathered} \Pi^{\mathrm{eq}} = \rho, \qquad \Pi_\alpha^{\mathrm{eq}} = \rho_0 u_\alpha, \qquad \Pi_{\alpha\beta}^{\mathrm{eq}} = \rho c_0^2 \delta_{\alpha\beta} + \rho_0 u_\alpha u_\beta, \\ \Pi_{\alpha\beta\gamma}^{\mathrm{eq}} = \rho_0 c_0^2 \left(u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\alpha\gamma} + u_\gamma \delta_{\alpha\beta} \right). \end{gathered} \tag{4.34}$$

We may perform the Chapman-Enskog analysis as in Section 4.1 using this equilibrium in order to find the simulated macroscopic equations. The resulting stress tensor is

$$\begin{aligned} \sigma_{\alpha\beta} = \left(\frac{\tau}{\Delta t} - \frac{1}{2} \right) \Big[& \rho_0 c_\mathrm{s}^2 \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right) - \rho_0 \left(u_\alpha \partial_\gamma \left(u_\beta u_\gamma \right) + u_\beta \partial_\gamma \left(u_\alpha u_\gamma \right) \right) \\ & - c_\mathrm{s}^2 \left(u_\alpha \partial_\beta \rho + u_\beta \partial_\alpha \rho \right) \Big]. \end{aligned} \tag{4.35}$$

Of the three terms in the brackets, the latter two are error terms. The first of these error terms is of a similar nature to the $O(u^3)$ error term in Eq. (4.14). The second error term appears due to the different pressure in the second and third equilibrium moments: ρc_s^2 in $\Pi_{\alpha\beta}^{\mathrm{eq}}$ and $\rho_0 c_\mathrm{s}^2$ in $\Pi_{\alpha\beta\gamma}^{\mathrm{eq}}$.

Neglecting these error terms, the Chapman-Enskog analysis results in macroscopic mass and momentum equations

$$\frac{1}{c_\mathrm{s}^2} \partial_t p + \rho_0 \partial_\gamma u_\gamma = 0, \tag{4.36a}$$

$$\rho_0 \left(\partial_t u_\alpha + \partial_\beta u_\alpha u_\beta \right) = -\partial_\alpha p + \eta \partial_\beta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right), \tag{4.36b}$$

with pressure $p = c_\mathrm{s}^2 \rho$ and dynamic shear viscosity $\eta = \rho_0 c_\mathrm{s}^2 (\tau/\Delta t - 1/2) \Delta t$.

For a steady flow case, a similar analysis to Section 4.2.3 can be used to show that the error terms in Eq. (4.35) disappear. Consequently, the macroscopic equations in

[15] Note that this holds only for *steady flow*, and not for cases where the flow field is time dependent. For instance, we have for plane sound waves that $|\rho'/\rho_0| = O(\mathrm{Ma})$ [12].

Eq. (4.36) (without time derivatives) is exactly recovered to the Navier-Stokes level. (However, higher-order truncation error terms may still be present.)

Exercise 4.7. Apply the analysis in Section 4.2.3 to the incompressible LBE model to show that the steady macroscopic equations (*i.e.* Eq. (4.36) without time derivatives) are exactly recovered to the Navier-Stokes level.

4.3.3 Alternate equations of state

Previously in Chapter 3 it was found that the simple square velocity sets lead to an isothermal equation of state $p = \rho c_s^2$, and that a more realistic thermal model requires extended velocity sets. However, alternate equilibria may also be applied in order to perform simulations with different equations of state, though as we soon shall see this approach has some weaknesses. Different equations of state also allow for different speeds of sound c: The speed of sound is determined by the equation of state $p(\rho, s)$ as $c = \sqrt{(\partial p/\partial \rho)_s}$, where s is entropy [12, 11].

From the Chapman-Enskog analysis in section 4.1, the pressure in the macroscopic momentum equation depends on the equilibrium moments

$$\Pi^{\mathrm{eq}}_{\alpha\beta} = p\delta_{\alpha\beta} + \rho u_\alpha u_\beta, \tag{4.37a}$$

$$\Pi^{\mathrm{eq}}_{\alpha\beta\gamma} = p\left(u_\alpha\delta_{\beta\gamma} + u_\beta\delta_{\alpha\gamma} + u_\gamma\delta_{\alpha\beta}\right). \tag{4.37b}$$

The first of these moments directly determines the pressure gradient in the momentum equations, and in the Chapman-Enskog analysis, terms stemming from the third-order moment cancel against terms stemming from the pressure term in the second-order moment.

However, no equilibrium distribution for the velocity sets given in section 3.3.7 can give a freely chosen pressure p in both these equilibrium moments: From the discussion in section 4.2, these velocity sets are *too small* for the third-order equilibrium moments $\Pi^{\mathrm{eq}}_{\alpha\beta\gamma}$ to be independent of lower-order equilibrium moments. Therefore, we may only have the third-order moment with $p \to \rho c_s^2$ without an extended velocity set.

While we can find an equilibrium with a freely chosen pressure p in the second-order equilibrium moment, this leads to an inconsistent pressure between the second- and third-order moments: p in the former and ρc_s^2 in the latter. Consequently, the aforementioned cancellation does not happen for a number of terms in the Chapman-Enskog analysis of an LBE with such an equilibrium, and we recover a macroscopic momentum equation containing several additional error terms [23],

$$\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = -\partial_\alpha p + \partial_\beta \rho c_s^2(\tau - \tfrac{1}{2})\left[\partial_\beta u_\alpha + \partial_\alpha u_\beta + \left(1 - \tfrac{c^2}{c_s^2}\right)\delta_{\alpha\beta}\partial_\gamma u_\gamma\right]$$
$$- \partial_\beta(\tau - \tfrac{1}{2})\partial_\gamma(\rho u_\alpha u_\beta u_\gamma)$$
$$+ \partial_\beta(\tau - \tfrac{1}{2})\left[\left(c_s^2 - c^2\right)\delta_{\alpha\beta}u_\gamma\partial_\gamma\rho + \left(u_\alpha\partial_\beta + u_\beta\partial_\alpha\right)\left(\rho c_s^2 - p\right)\right]$$

Here, c_s is the isothermal speed of sound and $c = \sqrt{(\partial p/\partial\rho)_s}$ is the actual speed of sound for a given pressure $p(\rho, s)$.

The first line of this momentum equation is the same as the momentum equation Eq. (4.15b) found from the Chapman-Enskog analysis in Section 4.1, except with a bulk viscosity of $\eta_B = \eta(5/3 - c^2/c_s^2)$ instead of $\eta_B = 2\eta/3$. This bulk viscosity unphysically depends on the velocity set constant c_s^2, but it is possible to correct the bulk viscosity in LB simulations [13, 24, 14]. The second line of this momentum equation is the same error as exists in the usual isothermal LBE, as seen in Eq. (4.14). The third line are new error terms that appear due to the aforementioned inconsistent pressure between the second- and third-order equilibrium moments.

However, these new error terms all scale as $O(\mathrm{Ma}\rho')$. For steady flow at $\mathrm{Ma} \ll 1$ it is known that $\rho'/\rho_0 = O(\mathrm{Ma}^2)$ [11], and for plane sound waves we have that $|\rho'/\rho_0| = O(\mathrm{Ma})$ [12]. Consequently, these new error terms are, in the worst case, of $O(\mathrm{Ma}^2)$, and can therefore be safely neglected along with the previous $O(\mathrm{Ma}^3)$ error terms as long as $\mathrm{Ma} \ll 1$.

Up to this point, we have only described the constraints on the moments of the equilibrium distribution f_i^{eq}, but we have not looked at what this equilibrium distribution can look like. In fact, a number of such equilibria have been proposed in the literature, both for general and specific equations of state [25, 26, 27, 17, 28]. However, many of these are specific to a particular velocity set, and those which are not have not been validated for all the velocity sets described in Section 3.3.7. It is not given that these equilibria are stable or accurate in all of these velocity sets [28].

As an example of an LB equilibrium that allows a non-isothermal equation of state for simple velocity sets, we can look at the equilibrium for D1Q3 which is uniquely given by [23, 6]

$$f_0^{(0)} = \rho - p - \rho uu, \quad f_{i\neq 0}^{(0)} = \tfrac{1}{2}\left(p + \rho u c_i + \rho uu\right). \tag{4.38}$$

The pressure p can be freely chosen, and its relation to ρ determines the speed of sound as described above. However, this equilibrium is still subject to stability conditions as described in Section 4.4, and not all equations of state will therefore be stable.

Exercise 4.8. Show that the equilibrium distribution in Eq. (4.38) has zeroth- and first-order moments $\Pi^{\mathrm{eq}} = \rho$ and Π_x^{eq}, and that its second- and third-order moments Π_{xx}^{eq} and Π_{xxx}^{eq} are given by Eq. (4.37) with $p \to \rho c_s^2$ in the third-order moment.

4.3.4 Other models

LBM has originally been used to solve the (weekly compressible) Navier-Stokes equations. Therefore, the standard equilibrium in eq. (3.56) follows from an expansion of the Maxwell-Boltzmann distribution. However, researchers have realised that the form of the macroscopic equations that are solved by the LB scheme can be changed by choosing different and appropriate equilibrium distributions. We conclude this section by mentioning a few recent developments where LBM is employed for other purposes than standard fluid dynamics.

LBM is often applied to shallow water problems. This research goes back to the late 1990s [29, 30] and is still ongoing [31]. As we will also point out in section 5.4, LBM can be used as Poisson solver for the pressure if a velocity field is defined [32]. This method is often applied to find a consistent initial state of an LBM simulation. The Navier-Stokes equation can also be solved *via* LBM in the vorticity-streamfunction formulation [33]. A few years ago, Mendoza *et al.* suggested LBM modifications to solve mildly relativistic fluid flows [34] and Maxwell's equations in materials [35]. More recent applications are LBM for image processing [36], solving elliptic equations [37] and consistent modelling of compressible flows [38]. In chapter 9 we will cover multi-phase and multi-component flows. The so-called free energy approach can be implemented by redefining the local equilibrium distribution function. TODO (TK): More precise reference required once the multi-phase chapter is finished.

4.4 Stability

We have previously discussed how the discrete numerical lattice Boltzmann equation may be used to solve the continuous Navier-Stokes equation. As the LBM is a numerical scheme, it brings issues associated with any numerical scheme. Those include the stability and accuracy of simulations. Since the very beginning of the LBM, its numerical stability has attracted considerable interest.

To simulate physical systems, the relevant dimensionless quantities need to be matched in the simulation (*cf.* chapter 7), which involves the choice of simulation-specific parameters. This choice is not arbitrary, and it is a whole art to map the physics of a real system to a simulation. Thus, as the choice of parameters is not arbitrary it's relatively easy to fall in the trap of parameters leading to numerical instability. Instability in an LBM simulations refers to situations where the values of *errors* for density, velocity and populations are exponentially growing. One should distinguish those simulations from ones where *solutions* grow exponentially. Both eventually lead to NaN (Not a Number) values for density, velocity and populations fields. Usually instabilities are attributed to truncation errors (between numerical and analytical solutions) ill-posed time evolution.

We will not present an exhaustive mathematical description for stability analysis [39, 40, 41], but rather show some practical guidelines to improve numerical

stability. After discussing some general concepts, we will cover the BGK collision operator, followed by advanced collision operators. We will finish the discussion about stability with the simple guideline of improving simulations stability.

4.4.1 Stability analysis

A common way to analyze stability is through a linear *von Neumann analysis* [42]. This analysis allows representing a variable we are solving for in the Fourier form (wave form) ($A\exp(\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}-\mathrm{i}\Omega t)$ with $\boldsymbol{x}$ and t being discretised spatial coordinates and time ($A\exp(-\mathrm{i}\Omega t)$ if the function only depends on time). Substituting this expression into the discretised equation, one searches for the dispersion relation $\Omega(\boldsymbol{k})$. Simulations are stable when $|\Omega|\leq 1$ for all possible wavenumbers $\boldsymbol{k}$.

For CFD one typically encounters the *Courant number* $C=|\boldsymbol{u}|\Delta t/\Delta x$ which is linked to stability [43]. The Courant number is not only a mathematical feature, but it has a more concrete meaning: it contrasts the speed $\Delta x/\Delta t$ at which information propagates in the model with the physical speed $|\boldsymbol{u}|$ at which the fluid field is advected. If $\Delta x/\Delta t<|\boldsymbol{u}|$ (*i.e.* $C>1$), the simulation cannot propagate the physical solution quickly enough, which tends to make the simulation unstable. Therefore, $C\leq 1$ is often necessary for stability.

Unfortunately, the stability analysis is more complicated for the LBM and the Courant number is not the determining factor. In addition to physical parameters Δx, Δt and $\boldsymbol{u}$, LB has additional degrees of freedom: one or more relaxation times. Moreover, it involves multiple equations $\{f_i\}$, one for each direction $\boldsymbol{c}_i$. Thus, finding the dispersion relation $\Omega(\boldsymbol{k})$ requires inversion of a $q\times q$ matrix, with q being the number of populations in a DdQq model. Despite those difficulties, there has been some progress in applying the linear von Neumann analysis to LBM [44, 15], especially in the field of applying LBM to simulate the advection-diffusion equation [45].

First, let us say a few words about how stability can be represented. For the LBM with the BGK collision operator, the *stability map* is represented by the maximum achievable velocity magnitude for a given relaxation time τ, *i.e.* $|\boldsymbol{u}_{\max}|(\tau)$. Let us give an example: if one wants to achieve a high Reynolds number $\mathrm{Re}=|\boldsymbol{u}|N/\nu$, then it is possible to either increase the macroscopic velocity magnitude $|\boldsymbol{u}|$, or grid number N, or decrease the viscosity ν. The increase of N is limited by the required memory and runtime constraints. Thus, the usual approach is to decrease the viscosity and/or increase the macroscopic velocity. Obviously, we cannot decrease viscosity to zero (*i.e.* $\tau\to 1/2$) as it would be possible to resolve any high Reynolds number leading to a solution of complex and unresolved turbulent flows. As well, we cannot increase velocity to infinity as the LBE recovers the Navier-Stokes equation only in the low-Mach limit (*cf.* chapter 3). Thus, for high Reynolds number flows simulations there is always a compromise between maximum velocity attained and minimum viscosity for which simulations are stable.

The stability map $|\boldsymbol{u}_{\max}|(\tau)$ tells us the maximum achievable velocity magnitude for a given value of τ before instability sets in.

One distinguishes between *sufficient* and *necessary stability conditions*. If a sufficient condition is met, the simulation is always stable. A necessary condition means that it has to be fulfilled in order to achieve stability. Sufficient conditions can never be *less* restrictive than necessary conditions. For example, if a single necessary condition for stability is met, the simulation is not automatically stable because another necessary condition may not apply. One can think of sufficiency as the combination of all possible necessary conditions. The least restrictive combination of necessary conditions across a range of parameters is called *optimal*.

Let us tell some words about how this is applicable to the BGK collision operator LBM. When sufficient and/or necessary conditions are mentioned, they are specified for specific τ. The optimal stability condition provides the largest value of the velocity magnitude $|\boldsymbol{u}|$ across all possible values of the relaxation time. This is also achieved for specific value of τ. For example, one will see that the optimal stability condition for the BGK LBM is achieved for $\tau \geq 1$.

The definitions for necessary/sufficient conditions are rather general. However, for the LBM those are obtained analytically for the bulk equation and for the infinite domain. In real simulations, there are always boundaries present and round-off errors. Thus, those conditions obtained analytically are deteriorated. Often those conditions have a recommendation character. It is always recommended to start with those and further refine to improve stability.

4.4.2 BGK stability

We will now present sufficient and necessary stability conditions for the BGK collision operator. Again we briefly mention that those conditions were obtained analytically for the bulk LB equation without boundaries present, and they have a recommendation character.

For the BGK collision operator, the sufficient stability condition for all $\tau > 1/2$ ($\tau > 1/2$ ensures the positivity of a viscosity) is the non-negativity of all equilibrium populations [46, 45, 41]. As long as $f_i^{\mathrm{eq}} \geq 0$ for all i, the simulation is stable. This condition says that any velocity $\boldsymbol{u}$ that insure non-negativity of all equilibrium functions also provides stable simulations independently of the choice of τ.

The non-negativity condition for the basic equilibrium populations of Eq. (3.56) is a complicated function of components of velocity $\boldsymbol{u}$. This is illustrated for D2Q9 in fig. 4.2.

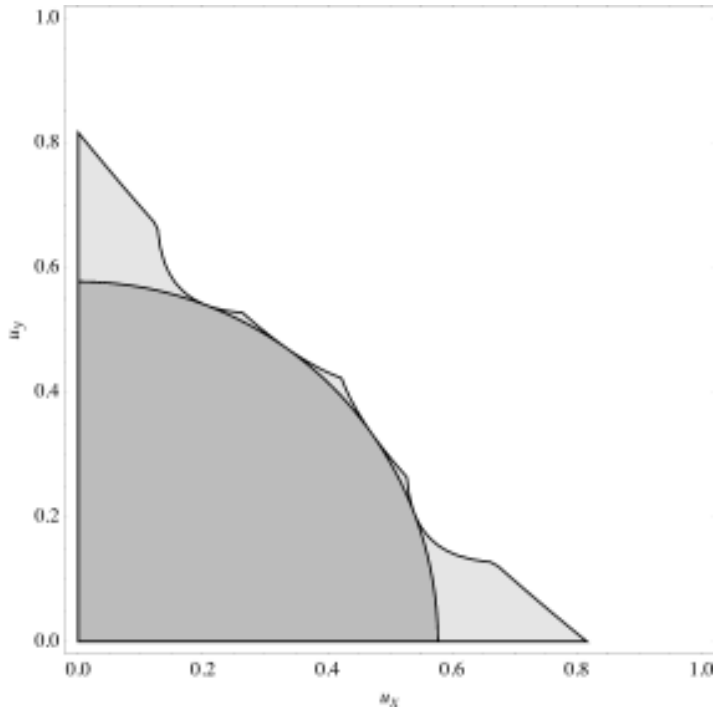


Fig. 4.2: Complex interplay of velocity components u_x and u_y that provide non-negative equilibrium populations given by Eq. (3.56) for the D2Q9 lattice. The light grey area denotes the sufficient stability condition independent of τ. The darker area represents a simplified condition of the form $|\boldsymbol{u}| < |\boldsymbol{u}_{\max}|$, *i.e.* containing only the magnitude of the velocity.

However, for practical purposes we will use only the representation with maximum achievable velocity *magnitude* $|\boldsymbol{u}_{\max}|$ that still provides stable simulations without taking into account individual velocity components. Following this idea, the complex stability condition in fig. 4.2 can be significantly simplified. For the BGK collision operator with the usual equilibrium function from eq. (3.56) it can then be written as:

$$|\boldsymbol{u}_{\max}| < \begin{cases} \sqrt{\frac{2}{3}} \approx 0.816 & \text{for D1Q3} \\ \sqrt{\frac{1}{3}} \approx 0.577 & \text{for D2Q9, D3Q15, D3Q19, D3Q27} \end{cases} . \tag{4.39}$$

In addition to the non-negativity of the equilibrium f_i^{eq}, some authors claim that the non-negativity of the populations f_i is required to ensure stability [47]. Special techniques to ensure that the populations remain positive have also been proposed [48, 49].

So far we have only discussed the sufficient stability condition which is independent on the relaxation time $\tau > 1/2$. This condition can be obtained from the union of all necessary conditions across all relaxation times. Among those, one necessary condition is of particular interest. It is the non-negativity of the immobile equilibrium population f_0^{eq}. It turns out that for all $\tau \geq 1$ this condition becomes sufficient and optimal: it provides the largest velocity magnitude across all τ.

For the BGK collision operator, the optimal stability condition is represented by non-negativity condition of the rest equilibrium population f_0^{eq} [45]:

$$|\boldsymbol{u}| < \sqrt{\frac{2}{3}}. \tag{4.40}$$

This condition is applicable when $\tau \geq 1$. It is also sufficient, thus it is always fulfilled for $\tau \geq 1$.

For all other relaxation time parameterers, i.e. $1/2 < \tau < 1$ the attained maximum velocity magnitude $|\boldsymbol{u}|$ is a complicated function of τ [41]. We indicate that maximum velocity magnitude $|\boldsymbol{u}|$ is bounded between two stability conditions: the sufficient stability condition for all $\tau > 1/2$ and the optimal stability condition. We sketch this stability region in fig. 4.3.

Unfortunately, as we indicated earlier those results above were obtained analytically for the bulk LB equation. They are too optimistic. In real simulations, it is not possible to have stable simulations for all values of $\tau > 1/2$ for all velocity magnitudes $|\boldsymbol{u}| < |\boldsymbol{u}_{\max}|$ from eq. (4.39). The stability condition deteriorates (*i.e.* $|\boldsymbol{u}_{\max}| \to 0$) with the decrease of the viscosity (*i.e.* $\tau \to 1/2$). A number of different works have showed the deterioration of stability when $\tau \to 1/2$. Some of them show it through the non-negativity of populations (not equilibrium populations as above) [47], some show it through the von Neumann analysis [44, 40]. However, we should emphasize that the sufficient stability boundaries for $\tau > 1/2$ between those works are different. For example, in comparison with the analytical results presented above, the von Neumann stability is done numerically and/or only for certain chosen wavenumbers $\boldsymbol{k}$. Thus, we will just underline that for general case flows (when boundary conditions implementation and numerical round-off computer errors affect stability), it is a safe assumption that the maximum achievable velocity magnitude $|\boldsymbol{u}_{\max}| \to 0$ with the relaxation time $\tau \to 1/2$. For example, Niu *et al.* [40] investigated the dependence of $|\boldsymbol{u}_{\max}|$ on τ for D2Q9. They found a linear relation for small τ:

$$|\boldsymbol{u}_{\max}|(\tau) = 8\left(\tau - \frac{1}{2}\right), \quad \tau < 0.55. \tag{4.41}$$

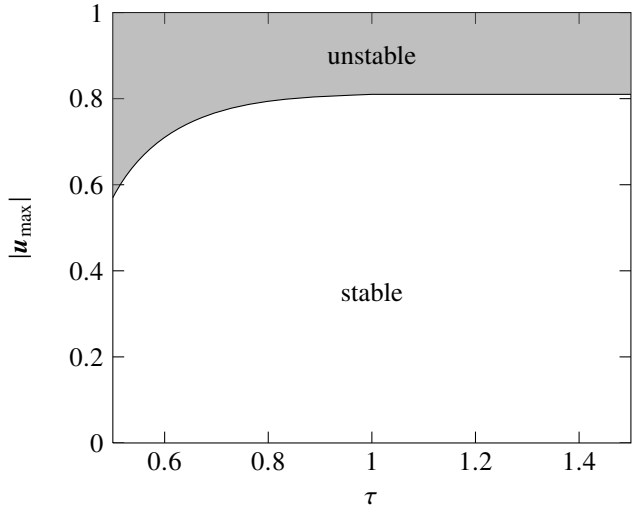


Fig. 4.3: Sketch of analytical stability region for two-dimensional and three-dimensional models. One can see the sufficient stability condition $|\boldsymbol{u}| < 1/\sqrt{3}$ when $\tau \to 1/2$ and the optimal stability condition $|\boldsymbol{u}| < \sqrt{2/3}$ when $\tau \geq 1$.

We show eq. (4.41) as a rough sketch of $|\boldsymbol{u}_{\max}|(\tau)$ in fig. 4.4 (refer to Fig. 2 in [40] for a more accurate picture). Note, however, that eq. (4.41) is only valid given certain assumptions and may not hold for more general flow problems.

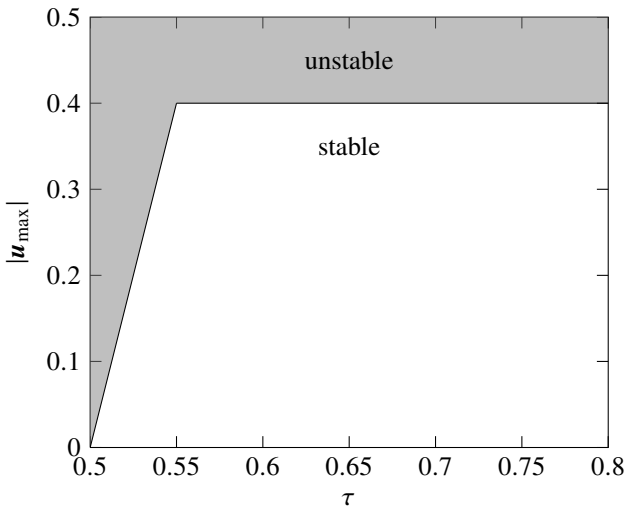


Fig. 4.4: Stability regions of $|\boldsymbol{u}_{\max}|$ *vs.* τ (simplified version of Fig. 2 in [40]).

If one needs to have τ close to $1/2$ for simulations, one should start as a first approximation with the sufficient stability condition for all $\tau > 1/2$ obtained analytically, *i.e.* bounding the velocity magnitude as $|\boldsymbol{u}| < u_{\max}$ from eq. (4.39). However, if simulations are unstable, then one needs to establish and utilise the dependence of the maximum achievable macroscopic velocity magnitude $\boldsymbol{u}_{\max}$ on the relaxation time τ by performing a few simulations with different values of $\boldsymbol{u}$ and τ close to $1/2$ and interpolating those results with a fitting curve. The curve is somewhat similar to fig. 4.4.

Certain things can be made for $\tau \approx 1/2$ to achieve larger velocity magnitudes. For example, choosing another collision operator (as discussed in more detail in chapter 10) might improve stability. We will briefly touch upon this in the next section.

4.4.3 Stability for advanced collision operators

As explained in more detail in Chapter 10, there is a hierarchy of collision operators. The BGK collision operator has only one relaxation time τ, the two-relaxation-time (TRT) collision operator has two relaxation times, τ^+ and τ^-, and for the multi-relaxation-time (MRT) collision operator there are q relaxation times τ_i related to a DdQq model. Thus, the BGK model is a special case of the TRT model, and the TRT model is a special case of the MRT model.

In this section we only provide results for the TRT model because there are analytical results available for it [41]; the MRT model is far more complex to analyze. Moreover, the results of the TRT model can be directly applied to the MRT model (*cf.* chapter 10).

On the macroscopic level of the Navier-Stokes equation, the TRT model and the BGK model are indistinguishable from each other if $\tau^+ = \tau$, where τ^+ is the relaxation time of the TRT model and τ is the relaxation time of the BGK model. Thus, for the TRT model there exists an additional parameter to tune, *i.e.* the relaxation time τ^-. Thus, the stability analysis for the TRT model is far more complex as the additional parameter is introduced. For example, the stability map changes from $|\boldsymbol{u}_{\max}|(\tau)$ to $|\boldsymbol{u}_{\max}|(\tau^+, \tau^-)$.

Luckily, one doesn't need to analyze stability for all possible values of τ^+ and τ^-. For the TRT model, there is a certain combination of τ^+ and τ^- (historically called the 'magic' parameter) that governs the stability and accuracy of simulations [50]:

$$\Lambda = \left(\tau^+ - \frac{1}{2}\right)\left(\tau^- - \frac{1}{2}\right). \tag{4.42}$$

The BGK collision operator reaches optimal stability for $\tau \geq 1$, *i.e.* it is possible for any $\tau \geq 1$ to reach the largest velocity magnitude where simulations are stable.

In the TRT framework the optimal condition reads $\Lambda = (1 - 1/2)(1 - 1/2) = 1/4$ [45], which for the BGK model reduces to $\tau^+ = \tau^- = 1$ (the BGK optimal stability condition). This means that the TRT model is able to satisfy the optimal stability condition *independently* of the individual relaxation rates τ^+ and τ^-.

The parameter value $\Lambda = 1/4$ provides the optimal stability condition, *i.e.* the largest stable velocity magnitude. Thus, one can fix τ^+ by matching the macroscopic viscosity, and then it is possible to calculate τ^- to maintain $\Lambda = 1/4$ *via* eq. (4.42):

$$\tau^- = \frac{\tau^+}{2\tau^+ - 1} \tag{4.43}$$

Thus, when for the BGK collision operator we see the stability deterioration with $|\boldsymbol{u}_{\max}| \approx 0$ when $\tau \approx 1/2$, it is possible to increase stable maximum velocity magnitude with the TRT collision operation by fixing τ^+ and having τ^- from eq. (4.43). The TRT collision operator with $\Lambda = 1/4$ is not a remedy that will definitely make simulations stable and will increase the maximum stable velocity magnitude. However, it is often worth to improve available stable maximum velocity magnitude with the TRT collision operator using $\Lambda = 1/4$, especially when one uses the BGK collision operator with $\tau \approx 1/2$.

4.4.4 Stability guideline

Finally, a simple guideline to improve stability for simulations is presented. Here we assume that the Reynolds number is matched between a simulation and the physical world:

$$\mathrm{Re} = \frac{|\boldsymbol{u}|N}{1/3(\tau - 1/2)}. \tag{4.44}$$

As a rule of thumb, we always start with the BGK collision operator. It is recommended to hold constant the grid number N that is comfortable for an user in terms of memory footprint and performance. Then take $\tau = 1$ and adjust $|\boldsymbol{u}|$ to match the Reynolds number.

If the obtained velocity magnitude $|\boldsymbol{u}_{\max}| < 1/\sqrt{3}$ (to insure sufficiency and low-Mach number expansion of the equilibrium function *cf.* chapter 3), and simulations are stable, then there is no need to proceed. Otherwise, then it is recommended to apply BGK/TRT/MRT collision operators with $\Lambda = 1/4$ (*cf.* chapter 10).

If the obtained velocity magnitude $|\boldsymbol{u}_{\max}| > 1/\sqrt{3}$, then one needs to reduce the relaxation parameter τ and reduce the velocity magnitude to $|\boldsymbol{u}| < 1/\sqrt{3}$ in order to keep the same Reynolds number. If the magnitude of the new scaled velocity $|\boldsymbol{u}|$ is larger than the obtained BGK collision op-

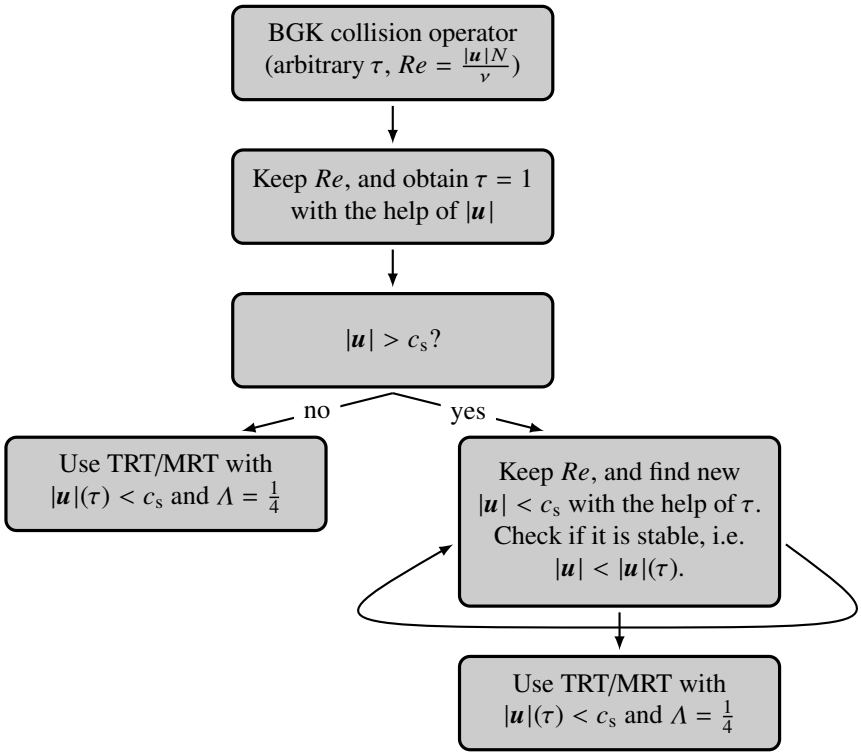


Fig. 4.5: The guideline to improve simulations stability with the fixed Reynolds number.

erator stability curve, *i.e.* $|\boldsymbol{u}_{\max}| > |\boldsymbol{u}_{\max}|(\tau)$ then one needs further reducing $\boldsymbol{u}$ (and decrease τ according to eq. (4.44)) until its magnitude is within the stability region. If the simulations are still unstable, then one needs to use TRT/MRT collision operators with the optimal stability parameter $\Lambda = 1/4$ that provides the largest stable velocity magnitude $|\boldsymbol{u}|$. In this case stability can be drastically improved.

The guideline for the stability improvement is presented graphically in fig. 4.5.

4.5 Accuracy

Now that we have learnt the basic principles determining the consistency and stability properties of LBM, it is time to take a look at its accuracy. In the present section

we will discuss how accurate LBM is as a Navier-Stokes solver, other references in this topic are [51, 52, 53, 54, 55, 4]. But before that, a few more concepts will be introduced such as what shall be understand by *order of accuracy*, section 4.5.1, and how to *measure the accuracy*, section 4.5.2. Then, in section 4.5.3 and in section 4.5.4, will review which kind of *numerical and modelling errors* come into play when doing LBM simulations. Finally, in section 4.5.5, we will show an overview of the *numerical accuracy of LBM* as a solver for the NSEs. This presentation ends in section **??** where we will discuss some guidelines on how to set-up an LBM simulation when focus is accuracy.

4.5.1 Formal order of accuracy

Let us start by explaining what accuracy order means and how can it be formally determined. While this subject could be directly discussed in the frame of LBM, the theoretical analysis of the LBM accuracy is a cumbersome task. As we have seen in section 4.1, accessing the continuum description of the LBM requires carrying out a lengthy Chapman-Enskog analysis. On the other hand, this study is much more direct for finite-difference schemes. Hence, in order to introduce concepts such as the formal order of accuracy and to show the relation between the accuracy order and the leading-order truncation errors, we will opt for the finite-difference method. The main idea will also hold for the LBM.

Consider the Couette flow depicted in fig. 1.1a and assume it is impulsively started from rest. We are interested in solving it for the velocity field $u(y,t)$, given proper initial and boundary conditions (which are not relevant for this case). In the bulk, the continuum formulation of this problem reads

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial y^2} = 0, \tag{4.45}$$

where ν is the kinematic viscosity.

Now, let us assume that we have discretised eq. (4.45) using a forward difference in time and a centered second-order finite difference scheme in space:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \nu \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta y^2} = 0. \tag{4.46}$$

Superscripts denote the time step and subscripts the spatial location.

The formal way to determine the accuracy of eq. (4.46) as an approximation to eq. (4.45) is by analysing the structure of the truncation errors in the discretised equation. These errors can be obtained in two steps. First, by performing a Taylor expansion of the discrete solution around time step n and location j.[16]

[16] This step assumes the smoothness of the solution.

$$u_j^{n+1} = u_j^n + \frac{\Delta t}{1!}\frac{\partial u}{\partial t}\bigg|_j^n + \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t^2}\bigg|_j^n + \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial t^3}\bigg|_j^n + O(\Delta t^4),$$
$$u_{j+1}^n = u_j^n + \frac{\Delta y}{1!}\frac{\partial u}{\partial y}\bigg|_j^n + \frac{\Delta y^2}{2!}\frac{\partial^2 u}{\partial y^2}\bigg|_j^n + \frac{\Delta y^3}{3!}\frac{\partial^3 u}{\partial y^3}\bigg|_j^n + O(\Delta y^4),$$
$$u_{j-1}^n = u_j^n - \frac{\Delta y}{1!}\frac{\partial u}{\partial y}\bigg|_j^n + \frac{\Delta y^2}{2!}\frac{\partial^2 u}{\partial y^2}\bigg|_j^n - \frac{\Delta y^3}{3!}\frac{\partial^3 u}{\partial y^3}\bigg|_j^n + O(\Delta y^4).$$

Then, by substituting the above expressions into eq. (4.46) which, after simplifying, yields

$$\frac{\partial u}{\partial t} - \nu\frac{\partial^2 u}{\partial y^2} = -\frac{\Delta t}{2!}\frac{\partial^2 u}{\partial t^2} + \nu\frac{\Delta y^2}{4!}\frac{\partial^4 u}{\partial y^4} + O(\Delta t^2) + O(\Delta y^4). \tag{4.47}$$

Exercise 4.9. Show that the approximation of $\frac{\partial u}{\partial t}|_j^n$ using the centered difference formula[17] $(u_j^{n+1} - u_j^{n-1})/(2\Delta t)$ leads to $O(\Delta t^2)$ leading truncation errors.

The utility of eq. (4.47) is that it displays the form of the actual PDE solved by the discretisation scheme, eq. (4.45), plus truncation errors. Those errors are the difference between the discretised equation and the targeted continuum equation. The truncation error analysis allows us to provide the following conclusions:

The discretisation scheme is said to be *consistent* on the condition that the truncation errors of a numerical scheme approximating a PDE tend to zero when Δt and Δy go to zero. The rate at which this happens establishes the *formal order of accuracy* of the discretisation scheme. (In the above example, it is first-order in time and second-order in space because the leading errors in eq. (4.47) scale with Δt and Δy^2, respectively.) If the scheme is *stable*, then the order of the discretisation errors will also dictate the *rate of convergence* of the numerical solution towards the target PDE solution. This is the main difference between the two concepts. While consistency is concerned with the governing equations when discretisation parameters go to zero, convergence focuses on the solution itself.

4.5.2 Accuracy measure

While in simple situations it is possible to evaluate the accuracy of a numerical scheme theoretically, *cf.* section 4.5.1, this undertaking is often hardly feasible as

[17] Note that this kind of approximation for the temporal term at time step n makes the algorithm effectively a three time level scheme, the so-called leapfrog scheme. Later in this section, we will note that LBM employs a similar discrete form in its time evolution approximation.

simulations involve non-linear terms and/or are affected by other error sources (as discussed later on in this section). Consequently, a theoretical measure for the accuracy of any quantity of interest in a simulation should be abandoned in favour of a more pragmatic approach.

The interest in knowing the accuracy of a numerical simulation is very practical. It allows us, for example, to know the sensitivity of our solution on the model parameters or to assess the quality of our code implementation, *i.e.* it helps identifying code mistakes (bugs) or other inconsistencies in the numerical algorithm or even in the model itself. This makes the subject of code validation and code verification a very important topic in CFD [56, 57]. A simple procedure to quantify the error of a numerical simulation consists in comparing it against a known analytical solution.[18] There are different approaches to do this, *cf.* [56, 57, 58]. For most cases in this book, we will use the so-called L_2 *error norm*.

Given an analytically known quantity $q_\mathrm{a}(\boldsymbol{x}, t)$, which is generally a function of space and time, and its numerical equivalent $q_\mathrm{n}(\boldsymbol{x}, t)$, we define the L_2 error norm as

$$\epsilon_q(t) := \sqrt{\frac{\sum_{\boldsymbol{x}} \left(q_\mathrm{n}(\boldsymbol{x}, t) - q_\mathrm{a}(\boldsymbol{x}, t)\right)^2}{\sum_{\boldsymbol{x}} q_\mathrm{a}^2(\boldsymbol{x}, t)}}. \tag{4.48}$$

The sum runs over the entire spatial domain where q is defined. The advantage of this definition is that local errors cannot cancel each other; the L_2 error is sensitive to any deviation from q_a.

The L_2 error can also be used as a criterion for convergence to steady flows. To do so, q_a and q_n are replaced by the numerical values of quantity q at a previous and the current time (or iteration) step, respectively. For example, one may define convergence by claiming that the L_2 deviation between the velocity field at subsequent time steps is below a threshold ϵ. For double precision arithmetic, typical values are around $\epsilon = 10^{-7}$. However, the suitability of this criterion should be judged with care. This is noted in section 4.5.3 when discussing the iterative error.

Some remarks are necessary:

1. If $\boldsymbol{q}$ is a vector-valued quantity, *e.g.* the velocity, one just replaces $q_\mathrm{a} \to \boldsymbol{q}_\mathrm{a}$ and $q_\mathrm{n} \to \boldsymbol{q}_\mathrm{n}$, with *e.g.* $\boldsymbol{q}_\mathrm{a}^2 = \boldsymbol{q}_\mathrm{a} \cdot \boldsymbol{q}_\mathrm{a}$.
2. If q_a is zero everywhere, the denominator in eq. (4.48) vanishes and the error is not defined. In this case another suitable normalisation has to be chosen.

Exercise 4.10. Show that the L_2 error ϵ_q is always non-negative, $\epsilon_q \geq 0$, and that equality only holds if $q_\mathrm{n}(\boldsymbol{x}) = q_\mathrm{a}(\boldsymbol{x})$ everywhere.

[18] In case an analytical solution is not available, we can use the outcome of a finer mesh simulation and take it as reference solution.

4.5.3 Numerical errors

During the process of approximating any *continuous* governing equations (with suitable boundary and initial conditions) into a *discrete* system of algebraic equations, we will inevitably introduce errors. In the CFD community it is commonly accepted that there are three error sources: the round-off error, the iterative error and the discretisation error [56, 57]. We will overview all of them in the LBM context.

Round-off error

The round-off error results from computers having finite precision. While inherent to any digital computation, this error source can be mitigated by using more significant digits, *e.g.* by switching from single precision (with 6 to 9 decimal digits, about 7 on average) to double precision (with 15 to 17 decimal digits, about 16 on average). The round-off error importance tends to increase with grid refinement and/or time step decrease.

In LBM the round-off error grows as the ratio $u/c_s \propto \mathrm{Ma}$ becomes small, as explained in [51]. It turns out we would like to keep this ratio as low as possible to reduce compressibility errors, *cf.* section 4.5.4. Hence, these two requirements lead to conflicting needs between numerical loss of precision (round-off error) and solution accuracy (compressibility errors). Fortunately, in most practical applications with double precision this problem ends up being insignificant. However, the situation is typically different for single precision computations.

Strategies to alleviate the round-off error in LBM have been considered. An early idea [51] consists of reformulating the LB equation using alternative working variables which, although mathematically equivalent to the original problem, turns LBM less susceptible to loss of numerical precision. An alternative procedure to reduce rounding errors consists of replacing the standard equilibrium by the incompressible one as point out by [59]. This last strategy however changes the mathematical form of the problem to solve [60], replacing the weakly-compressible NSE by the incompressible NSE (represented in artificial compressibility form), *cf.* section 4.3.2.

Iterative (steady-state) error

The iterative error results from the incomplete iterative convergence of the discrete equation solver. In explicit time-marching solvers this error determines the accuracy up to which *steady-state* solutions are reproduced.

As is known, the LBM achieves the steady-state through an explicit time-marching procedure. Hence, it is conceivable that such solutions, taken as a converged, may actually differ from the true steady-state by a certain amount which we call the *iterative error*. Even in cases the LBM routine adopts for solver other iterative procedures, such as time-implicit matrix formulations [61, 62, 63], a similar

kind of iterative error will inevitably affect the final steady-state solution. In fact, the main benefit of these time-implicit techniques is in accelerating the convergence towards steady-state. However, they have never received broad acceptance, in large part due to their cumbersome algorithmic complexity. Possibly the best compromise to accelerate LBM steady-state convergence while maintaining its simplicity is the preconditioned formulation of the LBM for steady-state flows developed in [64, 65, 66].

A way to infer the impact of the iterative error is to quantify how much our solution changes with time steps (iterations) based on a suitable convergence criteria. As a measure it is common to evaluate the difference (either absolute or relative) between successive iterations. Yet, this approach may lead to misleading conclusions, particularly when solutions display a slow convergence rate, *e.g.* [67]. A more reliable alternative is based on evaluating the solution residual at each time step. The idea of the residual convergence consists of measuring the remainder (residual) after plugging the current numerical solution into the discretised equation we are solving. The residual will tell us how far we are from the steady-state convergence. If the problem is well-posed, we should expect the residual to go to zero (up to the round-off error) as iterations converge.

In the LBM, the first strategy, based on the difference between successive iterations, remains the most popular apprach. But the residual evaluation, with the mass flux imbalance as convergence criterion, has been also used [68, 67]. Still, further studies are required to understand the performance of these strategies to measure the steady-state convergence of LBM solutions.

Discretisation error

Recalling section 4.5.1, the discretisation error results from approximating the continuous PDEs into a system of algebraic equations. This is the major error source separating our numerical solution from the exact solution of the PDE problem, decreasing with the refinement of the grid resolution and the time step.

For linear problems the discretisation error can be directly related to the (spatial and temporal) truncation errors, as we did in section 4.5.1. However, in more complex non-linear problems, or indirect procedures to solve the hydrodynamic problem, such as the LBM, the relationship between truncation and discretisation errors is not so evident. For that reason, the discretisation error is frequently estimated, *cf.* section 4.5.2, rather than analytically tracked. Providing certain conditions are respected (which are explained below), a direct estimation of the discretisation error is the rate of convergence of the numerical scheme. This quantity measures the sensitivity of our discretised solution at different levels of resolution as an approximation of the targeted PDE solution.

The role of numerical resolution on the discretisation error is identified in both spatial and temporal truncation terms. Let us focus on the spatial discretisation error and assume it to be the dominating numerical error in ϵ_ϕ, where ϕ is the solution of our problem (which for this purpose can be assumed an arbitrary scalar). Also for

simplicity, instead of using the L_2 error norm, let us quantify ϵ_ϕ by simply taking the difference between the numerical estimate ϕ_i (where subscript i refers to the grid refinement level) and the exact analytical solution ϕ_0.

$$\epsilon_\phi = \phi_i - \phi_0 = \alpha \, \Delta x_i^p + O\left(\Delta x_i^{p+1}\right). \tag{4.49}$$

The key point in our estimation process lies in assuming that, for sufficiently fine meshes, we can expect ϵ_ϕ to follow a power-law relation with the mesh spacing given by eq. (4.49) where Δx_i is the cell size for refinement i, p is the observed order of grid convergence, and α is a constant to be determined. In addition, $O\left(\Delta x_i^{p+1}\right)$ stands for the higher-order error terms in the truncation error (which are considered negligible in this analysis). By determining the numerical solution ϕ_i for three or more mesh sizes, say $\Delta x_1 = \Delta x$, $\Delta x_2 = r\Delta x$ and $\Delta x_3 = r^2 \Delta x$, we can estimate the convergence order p as follows:[19]

$$p = \frac{\log\left(\frac{\phi_3 - \phi_2}{\phi_2 - \phi_1}\right)}{\log(r)}. \tag{4.50}$$

For solutions lying in the asymptotic range of convergence (*i.e.* for sufficiently low Δx or Δt so that the lowest-order terms in the truncation error dominate), the order of accuracy of the numerical method can be measured by its order of convergence according to eq. (4.50).

The LBM is an $O(\epsilon^2)$ approximation of the NSEs, *cf.* section 4.1, where the second order terms in the truncation error are part of the fluid viscosity. This leaves only third-order terms as the truncation error so that the method is effectively *second order* accurate with respect to the NSEs. Since the time step is proportional to the lattice spacing, the scheme is second order in both time and space. [20] A way to identify this accuracy level is to measure the LBM numerical convergence: (1) second-order convergence in space means the error decreases quadratically with Δx, when fixing the dimensionless ratio $\Delta t \nu / \Delta x^2$; and (2) second-order convergence in time means the error decreases quadratically with Δt, when fixing the space discretisation Δx [51].

While these convergence measures are valid estimators for the accuracy of LBM solutions, it should be noted that the LBM discretisation error displays a more complex structure. The reason is that, unlike in standard CFD procedures, the continuous NSEs are not directly discretised in LBM. They are rather reproduced by a relaxation approach based on the Boltzmann equation. This means that, in addition to the

[19] Alternatively, in case we know the exact solution ϕ_0, we need ϕ_i for only two mesh sizes. Other convergence estimators are reviewed in [56, 57, 58].

[20] This accuracy assumes other error sources negligibly small. As will be discussed in section 4.5.4, if compressibility errors are on the same order of discretisation errors the LBM may no longer support second-order time accuracy for incompressible equations.

conventional dependency of truncation errors on the mesh size Δx (and also on the time step Δt if the problem is time-dependent), truncation errors will also depend on the relaxation parameter(s) [52, 53]. The specific relationship between the LBM truncation errors and the relaxation parameters has been deduced for BGK in [54] and for TRT and MRT in [69, 70, 55, 67] (see also chapter 10).

An interesting property of the LBM discretisation is that, for *steady-state* solutions, the *spatial* truncation errors are solely determined by terms of the following form:

- $\left(\tau - \frac{1}{2}\right)^2$ for BGK,
- $\Lambda = \left(\tau^+ - \frac{1}{2}\right)\left(\tau^- - \frac{1}{2}\right)$ for TRT (*cf.* chapter 10).
- $\Lambda_{j,k} = \Lambda_j^+ \Lambda_k^- = \left(\tau_j^+ - \frac{1}{2}\right)\left(\tau_k^- - \frac{1}{2}\right)$ for MRT (*cf.* chapter 10).

We can take advantage of these known functional forms. In particular, we can select certain relaxation parameter values to tune the accuracy or stability. This is a distinctive feature of LBM and not available for standard CFD procedures. Here we list possible improvements:[21]

- The third-order spatial truncation coefficient is proportional to $[(\tau - \frac{1}{2})^2 - \frac{1}{12}]$. According to the Chapman-Enskog analysis, this error appears at $O(\epsilon^3)$. It is the leading-order truncation of the Euler system of equations, as shown in [54, 70, 55, 67]. The relaxation choice that cancels the third-order spatial truncation error is $\tau = 1/\sqrt{12} + 1/2 \approx 0.789$ for BGK ($\Lambda = 1/12$) and is called the *optimal advection condition* [54, 70, 55, 67].
- The fourth-order spatial truncation coefficient is proportional to $[(\tau - \frac{1}{2})^2 - \frac{1}{6}]$. This truncation error appears at $O(\epsilon^4)$. It is the leading-order truncation of the viscous diffusion terms appearing in the NSE [54, 55, 67]. The relaxation choice that cancels the fourth-order spatial truncation error is $\tau = 1/\sqrt{6} + 1/2 \approx 0.908$ in BGK ($\Lambda = 1/6$) and is called the *optimal diffusion condition* [54, 55, 67].
- The effect of the non-equilibrium part f_i^{neq} of populations f_i on the evolution of f_i itself can be demonstrated [50, 55] to be proportional to $[(\tau - \frac{1}{2})^2 - \frac{1}{4}]$. Thus, in BGK $\tau = 1$ removes this term (and $\Lambda = 1/4$ in TRT). This choice makes LBM equivalent to a central finite difference scheme [71]. According to the linear von Neumann analysis, this choice corresponds to the *optimal stability condition* [45, 41] (*cf.* section 4.4).

It should be emphasised that the above list of discretisation coefficients is far more exhaustive; a more complete list can be found in [67, 72]. Still, for most practical purposes they should be taken as guidelines. The true impact of truncation corrections usually comes from several sources, possibly coupled in a complex way. As

[21] Although this presentation applies for the BGK model, the results can be easily extended to either TRT or MRT collision models by interpreting $(\tau - \frac{1}{2})^2$ as Λ according to eq. (4.42).

such, in general, the choice of ideal relaxation value(s) does not have an immediate answer, requiring an educated guess from the user.

Finally, we note that the dependency of discretisation errors on the relaxation parameter(s) may also have drawbacks. An important physical violation, which has no parallel in standard CFD procedures, may emerge when using the BGK model. The problem results from τ being the single relaxation parameter, controlling simultaneously the fluid viscosity and the coefficients of discretisation terms.

In contradiction to the fundamental physical requirement stating that *hydrodynamic solutions are uniquely determined by their non-dimensional physical parameters*, solutions produced by the BGK model exhibit τ-dependent and therefore viscosity-dependent characteristics.

To correct for this unphysical behaviour one must use collision models that offer the control of more than one relaxation rate, such as TRT or MRT (*cf.* chapter 10). Then, at least for steady-state solutions, we can keep the coefficients of the truncation terms independent of the viscosity value.

4.5.4 Modelling errors

Sometimes the numerical scheme adopted may not exactly reproduce the physics we are interested in, but rather approximate them in some sense. Still this choice can be totally justifiable, for example, in order to keep our numerical procedure simple. In the end, we may have to deal with an approximation problem, which we call the *modelling error*. The physical content of LBM is essentially determined by the chosen lattice (resulting from velocity space discretisation) and equilibrium. These two features are the main sources of the LBM modelling errors.

Effect of lattice and equilibrium

The velocity space discretisation is directly related to the chosen lattice (recall section 3.3). It defines which conservation laws (*via* the velocity moments) can be captured up to which accuracy level. It turns out that, for standard lattices (*i.e.* lattices whose isotropy is respected only up to the fifth order, *cf.* section 3.3.7), LBM cannot capture physical processes describing energy transport, *e.g.* [73, 74]. That is, the LBM with standard lattices is an isothermal model (or, more rigourously, an athermal model [15, 75]).

Also the mass and momentum balance equations described by LBM display limitations when simulated with standard lattices. As described in section 4.2.1, these lattices are not sufficiently isotropic to accurately describe the third-order velocity moment, which leads to a non-linear error in the viscous stress tensor, eq. (4.14).

Such a deficiency makes the LBM macroscopic equations differ from the true compressible NSE by an $O(u^3)$ error term [8]. While this cubic defect is negligible for slow flows, it may become dominant at high Ma numbers. The $O(u^3)$ error also impacts the LBM by corrupting the Galilean invariance of the macroscopic equations [8].

Standard lattices introduce an $O(u^3)$ error term, which limits the utilisation of LBM to simulations of isothermal NSEs in the weakly compressible regime.

Another modelling error comes from the LB equilibrium, which is what dictates the physics simulated by the LBM, *cf.* section 4.3. In fluid flow simulations this equilibrium is usually given by eq. (3.4), which reproduces the compressible NSEs. But due to the $O(u^3)$ error, only the weakly compressible regime (small Ma) is accessible. Therefore, LBM is typically adopted as an explicit compressible scheme for the incompressible NSEs [76, 60, 59].[22] In this case the equations reproduced by the LBM differ from the true incompressible NSEs by the so-called 'compressibility' errors, which are associated with gradients of density and velocity field divergence, and they typically scale with $O(\mathrm{Ma}^2)$ [52, 51].

A distinctive property of the compressibility error is that it is a grid-independent feature. That is, if we keep the Ma number constant while decreasing the grid resolution δx, there will be a point, at sufficiently fine meshes, where compressible effects will show up and saturate the error, blocking the convergence for smaller meshes. In case compressibility and discretisation errors are of the same order, converge towards the *incompressible* NSEs requires us to simultaneously decrease the $\mathrm{Ma} \sim \frac{u}{c}$ number and grid resolution $\delta x \sim \frac{\Delta x}{L}$, where u and L are macroscopic velocity and length scales, respectively. Hence, halving the grid spacing also halves the time step, since the two parameters are proportional in the solution update $c = \frac{\Delta x}{\Delta t} = 1$, and will also halve the Ma number owing to compressibility and discretisation errors being the same order, *i.e.* $O(\frac{u}{c}) \sim O(\frac{\Delta x}{L})$. It follows that by lowering the Ma number, the description of the same flow structure L will require us to track a longer time interval $T \sim \frac{L}{u}$. This means that, in order to decrease our numerical error by four, we will have to increase the number of time steps $\delta t \sim \frac{\Delta t}{T}$ by four as well. This observation follows from the requirement that $O(\mathrm{Ma}) \sim O(\delta x) \Rightarrow \frac{u\Delta t}{\Delta x} \sim \frac{\Delta x}{L}$, which after some manipulations leads to $\frac{uT}{L}\delta t \sim \delta x^2$ yielding $\delta t \sim \delta x^2$. Concluding, the time convergence of LBM gets reduced to the first order when compressibility and discretisation errors are of the same order [52, 51, 77].

In an attempt to get rid of the compressibility errors it was suggested, *e.g.* [21, 60], to replace the standard equilibrium in eq. (3.4) by an incompressible variant, eq. (4.33). This way, the incompressible NSEs are approximated by an artificial compressibility system [78, 76, 60]. While for steady flows this system fully removes the compressibility error, the 'trick' is not completely effective in time-dependent flows [76, 79, 80, 81], *cf.* section 4.3.2.

[22] There are exceptions, though, for example the simulation of sound waves in chapter 12.

In general, LBM simulations for the incompressible NSEs suffer from a modelling compressibility error of $O(\mathrm{Ma}^2)$. When keeping this compressibility error equal to the discretization error this makes the LB scheme effectively first-order accurate in time.

4.5.5 LBM accuracy

It is possible to formulate the LBM in terms of finite difference stencils, called *recurrence equations*. This has been derived in [50, 55]. Without going into details, it can be shown [55] that, in recurrence form, the momentum law reproduced by LBM satisfies:

$$\underbrace{\mathcal{T}(\boldsymbol{x},t)}_{\text{unsteady}}+\underbrace{\mathcal{A}(\boldsymbol{x},t)}_{\text{advection}}=-\underbrace{\mathcal{P}(\boldsymbol{x},t)}_{\text{pressure}}+\underbrace{\mathcal{D}(\boldsymbol{x},t)}_{\text{diffusion}}+\underbrace{O(\Delta x^2,\ p_s[\tau])}_{\text{spatial truncation}}+\underbrace{O(\Delta t^2,\ p_t[\tau])}_{\text{temporal truncation}}. \quad (4.51)$$

This generic form is nothing but a compact representation of eq. (4.15b) at discrete level, where each term in eq. (4.51) uses finite difference stencils rather than differential operators [55]. Let us analyse them individually.

1. The $\mathcal{T}$ operator is a three time level difference taken at time steps $(t+\Delta t, t, t-\Delta t)$. Its exact form depends on τ with the BGK.[23]
2. The $\mathcal{A}$ and $\mathcal{P}$ operators are centred difference schemes for the advection and pressure terms, respectively.
3. The $\mathcal{D}$ operator denotes the Du Fort-Frankel approximation for the diffusion term [82, 55]. As known [58], the consistency of this scheme requires $\Delta t \approx \Delta x^2$, a relation usually called *diffusive scaling*, *e.g.* [83, 5].
4. The remaining two terms evidence the truncation contributions coming from spatial Δx^2 and temporal Δt^2 terms, with an additional τ-dependent component, which takes a τ polynomial form. We denote them as $p_s[\tau]$ and $p_t[\tau]$ to specify for spatial and temporal contributions, respectively. These polynomials can be determined by terms of $(\tau-\frac{1}{2})^2$ [54, 69, 55]. [24]

LBM is a numerical scheme approximating the continuum macroscopic equations with second-order accuracy in space and time (with τ fixed). Yet, the continuum equations described by the LBE may not necessarily lead to the

[23] For those familiar with finite difference techniques, such an operator can be interpreted as a leapfrog scheme for the temporal derivative, yet not exactly identical due to the τ degree of freedom *cf.* [55].

[24] This analysis is valid for the BGK model. For the TRT/MRT collision models, the term $(\tau-\frac{1}{2})^2$ should be interpreted as Λ for spatial truncations, *cf.* chapter 10, while it is a combination of Λ and $(\Lambda^{\pm})^2$ terms for temporal truncations [55].

NSEs. The difference is due to the modelling errors and the approximation is only successful at low velocities, where $O(u^3)$ becomes negligible. Another modelling error is the compressibility error source $O(\mathrm{Ma}^2)$, which further affects the LBE in case one is interested in the incompressible NSEs. If this compressibility error is simultaneously reduced with the grid resolution, then the time accuracy of the LB scheme reduces to the first order [52, 54, 83].

Above we have only addressed the behaviour of LBM in bulk. It should be borne in mind that its actual accuracy may vary due to other sources, such as the specification of low-order boundaries and/or inaccurate procedures to initialise the solution. Both these topics will be discussed in chapter 5. Providing attention is devoted to them, LBM is a competitive NSE solver. Actually, LBM may even show better accuracy and stability characteristics in comparison to standard second-order NSE solvers at identical grids. Such a supremacy mainly comes from the LBM's control over its truncation terms in a grid-independent fashion, *e.g.* [84, 85, 4, 72].

4.5.6 Accuracy guideline

In a typical simulation scenario we want to optimise the accuracy of solutions, while respecting the non-dimensional groups of the problem. The matching between the non-dimensional numbers of the physical problem, say the Reynolds number in hydrodynamics or the Peclet number in transport problems, and those in the lattice Boltzmann scheme is the focus of chapter 7. Here, our motivation is to discuss how to select values for the velocity u, the relaxation time τ through the viscosity, and the grid number N, *cf.* eq. (4.44), in order to reach the highest accuracy possible.

Increasing the grid number N obviously enhances accuracy. However, it also significantly impacts the performance and memory requirements of simulations, and it should therefore be kept as low as possible, *e.g.* in 3D simulations an increase in N means a memory overhead of N^3. Hence, a judicious choice for the accuracy improvement must focus on the other two parameters u and τ, while keeping N fixed. To cope with the Reynolds number requirement, eq. (4.44), u and τ are not independent, but must be adjusted accordingly.

The velocity u does not strongly impact the accuracy, proving it is kept at sufficiently low values, recall discussion in section 4.5.4. Hence, the most important parameter we have for controlling the solution accuracy is the relaxation time.

As we have seen in section 4.5.3, in LBM the truncation errors depend on the relaxation parameter(s). If we use the BGK collision operator the only relaxation parameter available is τ. It can be proven that, for steady-state simulations [50], we can cancel the $O(\epsilon^3)$ or the $O(\epsilon^4)$ truncation errors with the following τ choice:

$$\begin{aligned} O(\epsilon^3): &\quad \tau = \frac{1}{2} + \frac{1}{2\sqrt{3}} \approx 0.789, \\ O(\epsilon^4): &\quad \tau = \frac{1}{2} + \frac{1}{\sqrt{6}} \approx 0.908, \end{aligned} \tag{4.52}$$

The case $\tau \approx 0.789$ is suitable for advection-dominated problems while $\tau \approx 0.908$ is advantageous in diffusion-dominated problems. Other classes of problems require the cancelling of other truncation contributions. A more complete list of τ values can be found in [67, 72].

It turns out that for accurate and consistent LB solutions it is not recommendable to use the BGK collision operator. This results from BGK solutions being viscosity-dependent, which means they are *not* uniquely determined by the hydrodynamic non-dimensional groups representing the physical problem. While for high grid refinement levels this defect may be kept within acceptable bounds (based on some subjective criteria), at coarser grids the effect of truncation terms is generally significant, making LB solutions sensitive on τ (*i.e.* viscosity). A quick fix consists of making an educated guess for the τ value (at all points of the domain) and stick with that choice. However, with this procedure we are not taking full advantage of the LBM distinctive feature: the possibility to improve the accuracy of numerical solutions in a grid-independent fashion with the tuning of relaxation parameters. Furthermore, in this way we are also compromising the computational efficiency of our simulation as to optimise accuracy a small τ value is generally required while for a rapid steady-state convergence a large τ value is necessary [50].

The TRT/MRT operators, discussed in chapter 10, solve the aforementioned problems. They enable the local prescription of the relaxation parameters without any change in viscosity. Taking as an example the TRT collision operator, its steady-state (or spatial truncation) errors depend on the combination of two relaxation times through the parameter $\Lambda = (\tau^+ - 1/2)(\tau^- - 1/2)$. The relaxation parameter τ^+ is related to the viscosity in the Navier-Stokes equation. The parameter τ^- is a free parameter. If one changes the viscosity (*i.e.* τ^+), then one should change τ^- accordingly to keep constant Λ (*i.e.* accuracy). This is a very powerful tool as it allows performing accurate simulations even with $\tau^+ \gg 1$, *e.g.* where convergence towards steady-state is faster. In the TRT framework the equivalent to eq. (4.52) is written in terms of Λ as:

$$\begin{aligned} O(\epsilon^3): &\quad \Lambda = \frac{1}{12} \approx 0.083, \\ O(\epsilon^4): &\quad \Lambda = \frac{1}{6} \approx 0.166. \end{aligned} \tag{4.53}$$

As a rule of thumb, the first action we should take is determining its non-dimensional numbers. Then, based on some educated guess, we should select the grid number N. Its value should represent a compromise between the best spatial resolution possible and the available computational resources. Also, the grid number N choice should give us the possibility to vary τ while keep-

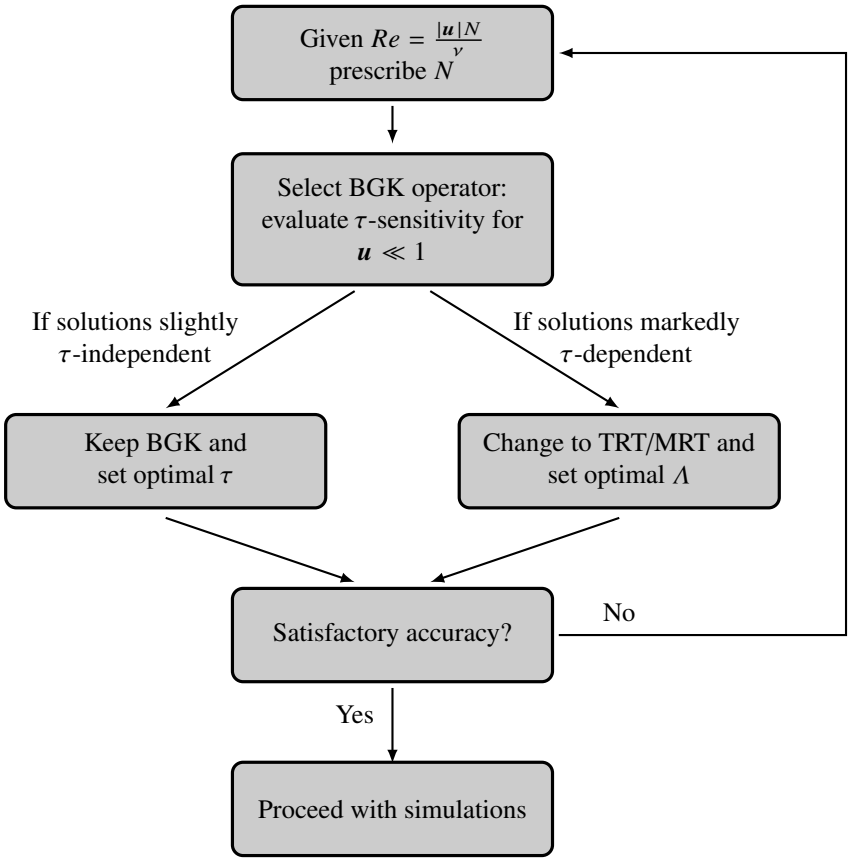


Fig. 4.6: The workflow to improve simulation accuracy for simulations that require keeping constant Re.

ing u low to satisfy the low Ma number requirement. The final parameter we should concern is τ. We can start with the BGK collision operator and check whether solutions are too τ-sensitive. In case they display a significant dependency on τ the BGK collision operator should be replaced by the TRT or MRT collision models. The next step is finding a suitable relaxation parameter, either τ or Λ, to optimise accuracy. For this task we can use our knowledge from the problem physics, *e.g.* if the problem is dominated by bulk phenomena or by boundaries of if it is dominated by convection or by diffusion etc. Based on it we can choose a proper τ or Λ value from eq. (4.52) or eq. (4.53), respectively, or from literature, *e.g.* [50, 55, 67, 72]. A more meticulous adjustment of the relaxation parameters should then follow a trial and error basis. In the

end, if the attained accuracy is still unsatisfactory we should take a larger grid number N and the restart the process.

The workflow shown in Fig. 4.6 summarises the guidelines here provided to optimise the accuracy of LB simulations.

4.6 Summary

In this chapter we have found the macroscopic conservation equations that the lattice Boltzmann equation reproduces as the grid size Δx and time step Δt tend to zero. The mass conservation equation is the continuity equation, while the momentum conservation equation becomes the Navier-Stokes equation in the limit of weak compressibility, *i.e.* as Ma^2 goes to zero. (Otherwise, it provides an approximation of the compressible Navier-Stokes equation with $O(u^3)$ error terms.)

The Chapman-Enskog analysis in Section 4.1 is used to establish the connection between the "mesoscopic" LBE and the macroscopic mass and momentum equations. For other variants of the LBE, such as the alternate-equilibrium models covered in Section 4.3, this analysis may also be used to determine the macroscopic equations that these variants imply. The Chapman-Enskog analysis is not the only possible approach here, though; some alternatives are briefly touched on in Section 4.2.4.

The picture of the LBE as a Navier-Stokes solver is complicated by the fact that the lattice Boltzmann equation is not a direct discretisation of the Navier-Stokes equation, unlike many other fluid solvers. Instead, it is a discretisation of the continuous Boltzmann equation, from which the Navier-Stokes equation can be restored.

The origin of the LBE affects its *consistency* as a Navier-Stokes solver. We saw in Section 4.1 that the Navier-Stokes equation follows from the Boltzmann equation for *low* Knudsen numbers, which we can see as a condition on the changes in f_i being sufficiently slow in time and space. Additionally, when velocity space is discretised, velocity related errors of order $O(u^3)$ are introduced in the Navier-Stokes equation due to the truncation of the equilibrium distribution. The necessity of this truncation is discussed in Section 4.2.1.

The *stability* of the LBE is not entirely simple, either. Stability analysis is often done with the help of a linear von Neumann analysis, but applying this to the LBE is complicated by the fact that the LBE is a coupled *system* of equations; one equation for each velocity $\boldsymbol{c}_i$. Moreover, the equilibrium function depends non-linearly on the populations f_i, which is an additional source of complication for stability analysis.

The *accuracy* study of the LBE is a difficult task to perform theoretically. Its analysis is only feasible in simple academical cases, *cf.* section 4.5.1. In most practical situations it is more convenient to estimate the accuracy through measurements, *cf.* section 4.5.2. Still, some theoretical concepts such as the range of errors which

affect the LBM and their impact on the accuracy should be understood in order to make good use of the LBE. These topics were treated in remaining of section 4.5.

We saw in section 4.5.3 that the *numerical errors* affecting the LBM come from three sources: the round-off error, the iterative error and the discretisation error. Typically, the first two have very limited impact compared to the discretisation error. The form of this error is controlled by the truncation terms, stemming from the discretisation of the continuous Boltzmann equation. In the LBM, the leading order spatial and temporal truncation terms decrease with Δx^2 and Δt^2. Additionally, these terms also depend on the relaxation parameter, *e.g.* τ in BGK. This extra degree-of-freedom allows controlling the LBM discretisation error in a grid-independent fashion. If used judiciously, this can make LBM superior to standard second-order NSE solvers.

When it is used as a Navier-Stokes solver the LBM contains some *modelling error* terms, *cf.* section 4.5.4. The most common examples are the compressibility error and the cubic error. The compressibility error is what differentiates the LBM solution from the true *incompressible* Navier-Stokes solution. However, as the compressibility error scales with $O(\mathrm{Ma}^2)$, it is negligible for sufficiently small Ma. The slow flow assumption in turn minimises the $O(u^3)$ error.

Considering its advantages and disadvantages, it can be said that the LBM is a competitive second-order accurate Navier-Stokes solver due to its distinctive characteristics.

References

1. S. Chapman, T.G. Cowling, *The Mathematical Theory of Non-uniform Gases*, 2nd edn. (Cambridge University Press, 1952)
2. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
3. Y.Q. Zu, S. He, Phys. Rev. E **87**, 043301 (2013)
4. G. Silva, V. Semiao, J. Comput. Phys. **269**, 259 (2014)
5. S. Ubertini, P. Asinari, S. Succi, Phys. Rev. E **81**(1), 016311 (2010)
6. E.M. Viggen, The lattice Boltzmann method: Fundamentals and acoustics. Ph.D. thesis, Norwegian University of Science and Technology (NTNU), Trondheim (2014)
7. C.M. Bender, S.A. Orszag, *Advanced mathematical methods for scientists and engineers* (McGraw-Hill, 1978)
8. Y.H. Qian, S.A. Orszag, Europhysics Letters (EPL) **21**(3), 255 (1993)
9. S. Geller, M. Krafczyk, J. Tölke, S. Turek, J. Hron, Computers & Fluids **35**(8–9), 888 (2006)
10. J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Phys. Rev. E **77**(5), 056703 (2008)
11. P.A. Thompson, *Compressible-Fluid Dynamics* (McGraw-Hill, 1972)
12. L.E. Kinsler, A.R. Frey, A.B. Coppens, J.V. Sanders, *Fundamentals of acoustics*, 4th edn. (John Wiley & Sons, 2000)
13. P. Dellar, Phys. Rev. E **64**(3) (2001)
14. S. Bennett, A lattice Boltzmann model for diffusion of binary gas mixtures. Ph.D. thesis, University of Cambridge (2010)
15. P. Lallemand, L.S. Luo, Phys. Rev. E **61**(6), 6546 (2000)
16. N. Prasianakis, I. Karlin, Physical Review E **76**(1) (2007)
17. B. Chopard, A. Dupuis, A. Masselot, P. Luthi, Adv. Complex Syst. **05**(02n03), 103 (2002)
18. D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models* (Springer, 2005)

19. B. Dünweg, A.J.C. Ladd, in *Advances in Polymer Science* (Springer Berlin Heidelberg, 2008), pp. 1–78
20. E.M. Viggen, Physical Review E **87**(2) (2013)
21. Q. Zou, S. Hou, S. Chen, G.D. Doolen, Journal of Statistical Physics **81**(1-2), 35 (1995)
22. X. He, L.S. Luo, J. Stat. Phys. **88**(3-4), 927 (1997)
23. E.M. Viggen, Phys. Rev. E **90**, 013310 (2014)
24. J. Latt, Hydrodynamic limit of lattice Boltzmann equations. Ph.D. thesis, University of Geneva (2007)
25. F. Alexander, H. Chen, S. Chen, G. Doolen, Physical Review A **46**(4), 1967 (1992)
26. X. Shan, H. Chen, Phys. Rev. E **49**(4), 2941 (1994)
27. B.J. Palmer, D.R. Rector, Journal of Computational Physics **161**(1), 1 (2000)
28. P.J. Dellar, Physical Review E **65**(3) (2002)
29. R. Salmon, J. Mar. Res. **57**(3), 503 (1999)
30. J.G. Zhou, Comput. Method. Appl. M. **191**(32), 3527 (2002)
31. S. Li, P. Huang, J. Li, Int. J. Numer. Meth. Fl. **77**(8), 441 (2015)
32. R. Mei, L.S. Luo, P. Lallemand, D. d'Humières, Comput. Fluids **35**(8–9), 855 (2006)
33. S. Chen, J. Tölke, M. Krafczyk, Comput. Method. Appl. M. **198**(3–4), 367 (2008)
34. M. Mendoza, B.M. Boghosian, H.J. Herrmann, S. Succi, Phys. Rev. Lett. **105**(1), 014502 (2010)
35. M. Mendoza, J.D. Muñoz, Phys. Rev. E **82**(5), 056708 (2010)
36. J. Chen, Z. Chai, B. Shi, W. Zhang, Comput. Math. Appl. **68**(3), 257 (2014)
37. D.V. Patil, K.N. Premnath, S. Banerjee, J. Comput. Phys. **265**, 172 (2014)
38. K. Li, C. Zhong, Int. J. Numer. Meth. Fl. **77**(6), 334 (2015)
39. D.N. Siebert, L.A. Hegele, P.C. Philippi, Phys. Rev. E **77**(2), 026707 (2008)
40. X.D. Niu, C. Shu, Y.T. Chew, T.G. Wang, J. Stat. Phys. **117**(3-4), 665 (2004)
41. A. Kuzmin, I. Ginzburg, A. Mohamad, Comp. Math. Appl. **61**, 1090 (2011)
42. J. Hoffmann, *Numerical Methods for Engineers and Scientists* (McGraw-Hill, 1992)
43. R.J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady State and Time Dependent Problems* (SIAM, 2007)
44. J.D. Sterling, S. Chen, J. Comput. Phys. **123**(1), 196 (1996)
45. I. Ginzburg, D. d'Humieres, A. Kuzmin, J. Stat. Phys. **139**, 1090 (2010)
46. S. Suga, Int. J. Mod. Phys. C **20**, 633 (2009)
47. X. Aokui, Acta Mech. Sinica **18**(6), 603 (2002)
48. R. Brownlee, A. Gorban, J. Levesley, Physica A **387**, 385 (2008)
49. F. Tosi, S. Ubertini, S. Succi, H. Chen, I. Karlin, Math. Comp. Simul. **72**(2-6), 227
50. D. d'Humières, I. Ginzburg, Comput. Math. Appl. **58**, 823 (2009)
51. P.A. Skordos, Phys. Rev. E **48**(6), 4823 (1993)
52. M. Reider, J. Sterling, Comput. Fluids **118**, 459 (1995)
53. R.S. Mayer, Int. J. Mod. Phys. C **8**, 747 (1997)
54. D.J. Holdych, D.R. Noble, J.G. Georgiadis, R.O. Buckius, J. Comput. Phys. **193**(2), 595 (2004)
55. I. Ginzburg, Commun. Comput. Phys. **11**, 1439 (2012)
56. P.J. Roache, *Verification and Validation in Computational Science and Engineering, , , 1998.*, 1st edn. (Hermosa Publishers, New Mexico, 1998)
57. C.J. Roy, J. Comp. Phys. **205**, 131 (2005)
58. J.H. Ferziger, M. Peric, A. Leonard, *Computational Methods for Fluid Dynamics*, vol. 50, 3rd edn. (Springer, 2002)
59. L.S. Luo, W. Lia, X. Chen, Y. Peng, W. Zhang, Phys. Rev. E **83**, 056710 (2011)
60. X. He, L.S. Luo, J. Stat. Phys. **88**, 927 (1997)
61. R. Verberg, A.J.C. Ladd, Phys. Rev. E **60**, 3366 (1999)
62. M. Bernaschi, S. Succi, H. Chen, J. Stat. Phys. **16**, 135 (2001)
63. J. Tolke, M. Krafczyk, E. Rank, J. Stat. Phys. **107**, 573 (2002)
64. O. Filippova, D. Hänel, J. Comp. Phys. **165**, 407 (2000)
65. Z. Guo, T.S. Zhao, Y. Shi, Phys. Rev. E **70**(6), 066706 (2004)
66. S. Izquierdo, N. Fueyo, J. Comput. Phys. **228**(17), 6479 (2009)

67. S. Khirevich, I. Ginzburg, U. Tallarek, J. Comp. Phys. **281**, 708 (2015)
68. L. Talon, D. Bauer, D. Gland, H. Auradou, I. Ginzburg, Water Resour. Res. , **48**, W04526 (2012)
69. I. Ginzburg, Phys. Rev. E **77**(066704), 1
70. B. Servan-Camas, F. Tsai, Adv. Water Res. **31**, 1113 (2008)
71. R.G.M. Van der Sman, Comput. Fluids **35**, 849 (2006)
72. I. Ginzburg, G. Silva, L. Talon, Phys. Rev. E **91**, 023307 (2015)
73. X. Shan, X.F. Yuan, H. Chen, J. Fluid Mech. **550**, 413 (2006)
74. P.C. Philipi, L.A. Hegele, L.O.E. Santos, R. Surmas, Phys. Rev. E **73**, 056702 (2006)
75. P. Lallemand, L.S. Luo, Phys. Rev. E **68**, 1 (2003)
76. X. He, G.D. Doolen, T. Clark, J. Comp. Phys. **179**, 439 (2002)
77. M. Junk, Z. Yang, J. Stat. Phys. **121**, 3 (2005)
78. A.J. Chorin, J. Comput. Phys **2**, 12 (1967)
79. P.J. Dellar, J. Comp. Phys. **190**, 351 (2003)
80. G. Hazi, C. Jimenez, Comput. Fluids **35**, 280–303 (2006)
81. G. Silva, V. Semiao, J. Fluid Mech. **698**, 282 (2012)
82. M.G. Ancona, J. Comp. Phys. **115**, 107 (1994)
83. M. Junk, A. Klar, L.S. Luo, J. Comput. Phys. **210**, 676 (2005)
84. S. Marié, D. Ricot, P. Sagaut, J. Comput. Phys. **228**(4), 1056 (2009)
85. Y. Peng, W. Liao, L.S. Luo, L.P. Wang, Comput. Fluids **39**, 568 (2010)

Chapter 5
Boundary and initial conditions

Traditional hydrodynamics theory is described by a set of partial differential equations (PDEs) [1], consisting of the mass, momentum and, in some cases, energy equations. For brevity, we shall refer to all these equations collectively as the Navier–Stokes equations (NSEs) in this chapter. Recalling calculus, solutions of PDEs cannot be uniquely determined unless proper boundary and initial conditions are specified. Hence, the role of boundary and initial conditions within continuum fluid dynamics can be comprehended as, first of all, a mathematical necessity. Their purpose is to single out, from all admissible solutions of the NSEs, the specific solution of the fluid flow problem being considered.

From a physical standpoint, the importance of boundary and initial conditions can be understood even more intuitively. Given that the general theoretical framework to model fluid flows (from the motion of air in the atmosphere to blood circulation in the human body) relies on the same set of equations, *i.e.* the NSEs, other information must be considered before a solution is possible. Otherwise, the formulation of the physical problem remains incomplete as the NSEs by themselves have no information in them about the particular flow one is interested in. The required information is contained in the boundary and initial conditions. Only after these conditions have been specified together with the NSEs one can consider the problem to be complete or, as stated in mathematics, well-posed.[1]

The preceding introduction is intended to motivate the reader about the importance of correctly formulating both boundary and initial conditions. Indeed, it is not an exaggeration to say that the specification of boundary conditions is one of the most important tasks in the setup of fluid flow problems. For that reason, this chapter covers this subject in great detail along three parts. In the first part, we overview the role of boundary and initial conditions in the frame of continuum theories and standard numerical methods, and then introduce the concept of boundary condition in LB methods. In the second part, we explain how several LB boundary conditions

[1] Although no mathematical proof exists yet, this book takes for granted that NSEs solutions exist, are unique and vary smoothly with changes to initial conditions.

work and how they can be implemented.[2] In the third part, we provide supplementary material, containing more advanced elements on LB boundary conditions.

5.1 Fundamentals

5.1.1 Concepts in continuum fluid dynamics

Whenever the flow problem is time-dependent the specification of initial conditions is required.[3] They must be specified for the full domain. That is, if $\boldsymbol{u}_0$ specifies the fluid velocity at some given instant t_0 (often conveniently chosen as $t_0 = 0$) then the time-dependent solution of the NSE must respect the initial condition:

$$\boldsymbol{u}(\boldsymbol{x}, t_0) = \boldsymbol{u}_0(\boldsymbol{x}). \tag{5.1}$$

When a physical process describes a variation of the solution along its spatial coordinates, then the prescription of boundary conditions is imperative [2]. By definition, boundary conditions specify the behaviour of the PDE solution at the boundaries of the problem domain. If the solution is time-dependent, then the prescription of boundary conditions also needs to be set for all times. Depending on which constraint they impose on the boundary, boundary conditions can be classified into three categories, all contained in the equation below, where φ denotes the solution of a generic PDE, $\boldsymbol{x}_B$ the boundary location and $\boldsymbol{n}$ the boundary normal.

$$b_1 \left.\frac{\partial \varphi}{\partial \boldsymbol{n}}\right|_{(\boldsymbol{x}_B, t)} + b_2 \varphi(\boldsymbol{x}_B, t) = b_3. \tag{5.2}$$

- The *Dirichlet* condition is set by $b_1 = 0$ and $b_2 \neq 0$ in eq. (5.2). This condition fixes the amount of φ on the boundary $\boldsymbol{x}_B$ to the value $\frac{b_3}{b_2}$.
- The *Neumann* condition is set by $b_1 \neq 0$ and $b_2 = 0$ in eq. (5.2). This condition fixes the flux of φ on the boundary $\boldsymbol{x}_B$ to the value $\frac{b_3}{b_1}$.
- The *Robin* condition is set by $b_1 \neq 0$ and $b_2 \neq 0$ in eq. (5.2). This condition entails a relation between the amount of the property φ on the boundary $\boldsymbol{x}_B$ and its flux.

In hydrodynamics, the solution of the NSE requires, in general, the prescription of boundary conditions for the fluid velocity and/or the stresses. They classify into Dirichlet and Neumann conditions for the fluid velocity solution.

---

[2] NOTE (EMV): I suggest adding this in the footnote: 'Special boundary conditions designed to let sound waves exit the system smoothly are covered in the chapter on sound waves, specifically in Section 12.4.'

[3] Exceptional cases of time-dependent problems where initial conditions have an immaterial role are discussed in section section 5.4.

NOTE (EMV): Paragraphs don't work properly inside list environments, so I took the liberty of un-itemizing the following. Hope you don't mind!

The *Dirichlet condition* for the fluid velocity is:

$$\boldsymbol{u}(\boldsymbol{x}_B, t) = \boldsymbol{U}_B(t), \tag{5.3}$$

where $\boldsymbol{U}_B$ stands for the boundary velocity.

By denoting ($\boldsymbol{n}$, $\boldsymbol{t}$) as normal and tangential boundary vectors, then the zero relative normal velocity, $[\boldsymbol{u}(\boldsymbol{x}_B, t) - \boldsymbol{U}_B(t)] \cdot \boldsymbol{n} = 0$, describes the impermeability of the material surface, while the zero relative tangential velocity, $[\boldsymbol{u}(\boldsymbol{x}_B, t) - \boldsymbol{U}_B(t)] \cdot \boldsymbol{t} = 0$, is known as the no-slip velocity condition []. NOTE (EMV): Empty cite?

The *Neumann condition* for the fluid velocity is:

$$\mathbf{n} \cdot \boldsymbol{\sigma}(\boldsymbol{x}_B, t) = \boldsymbol{T}_B(t), \tag{5.4}$$

where $\boldsymbol{\sigma}$ denotes the sum of pressure and viscous contributions as defined in eq. (1.15) and $\boldsymbol{T}_B$ is a prescribed boundary stress vector.

The continuity of the normal and tangential stresses is established, respectively, by $\boldsymbol{n} \cdot \boldsymbol{\sigma}(\boldsymbol{x}_B, t) \cdot \boldsymbol{n} = \mathbf{T}_B \cdot \mathbf{n}$ and $\mathbf{n} \cdot \boldsymbol{\sigma}(\boldsymbol{x}_B, t) \cdot \boldsymbol{t} = \boldsymbol{T}_B \cdot \boldsymbol{t}$. With their prescription we specify the status of the mechanical balance of the fluid on the boundary.

It is outlined that several combinations of boundary conditions may be taken. For example, the entire boundary can be set as Dirichlet or a part of it can be Dirichlet and the rest Neumann. Yet, one must be cautious when specifying the Neumann condition on the entire boundary. In this case the problem does not have a unique solution [2] as, for instance, an arbitrary constant may be added to the solution without modifying it. Also, we stress that the same boundary may simultaneously be subject to different kinds of boundary conditions. For example, in coupling the flow dynamics between two interacting (non-immiscible) fluids, the interfacial boundary condition between them prescribes conditions for both velocity and stresses; see Chapter 9.

5.1.2 Initial conditions in discrete numerical methods

In case the NSEs are solved numerically the specification of *initial condition* remains given by eq. (5.1). However, when set in computational codes, they are often elaborated within a more general step called *initialisation*. We alert the reader that an indiscriminate usage of both terms may lead to confusion! To differentiate them, we should keep in mind that the initial conditions are set by the problem physics, *i.e.* they only apply to time-dependent problems. The initialisation procedure, on the other hand, is required even when the problem is steady, because computations should not start with the memory filled with random values. Consequently, the initialisation is a computational step that always figures as part of the numerical procedure, regardless of the time-dependency of the modelled problem.

Given that this task involves no special efforts other than initialising appropriate arrays of data in the algorithm, we skip details on this topic here. For the interested reader the following references are suggested [3, 4]. However, given that the initialisation procedure does not extend in such a trivial way to LB implementations, we will return to this topic in section 5.4. There, adequate strategies to *initialise* LB codes are discussed focusing on both time-independent and time-dependent problems.

5.1.3 Boundary conditions in discrete numerical methods

Although boundary conditions in numerical methods share the same fundamental goals of the continuous analytical case, they develop along conceptually different lines. In fact, while analytical boundary conditions apply as additional equations that select the solution of interest from an infinite family of admissible solutions, boundary conditions in numerical methods operate as part of the solution procedure. Their working principle relies on the extrapolation of data from the boundary into the inner domain. Thus, numerical boundary conditions act in a dynamical manner, being part of the process responsible for the change of the state of the system towards the intended solution. For that reason, the task of specifying boundary conditions assumes an even greater importance in numerical methods. In case they are not properly introduced in the numerical scheme severe problems may arise during the evolution process. As a matter of fact, one of the most common causes for the divergence of numerical solutions comes from the incorrect assignment of boundary conditions.

Recalling Chapter 2, traditional numerical methods for solving the NSEs are usually designed to work directly over the equations governing the fluid flow problem. Taking the finite difference scheme (*cf.* section 2.1.1) as an example, its discretisation procedure consists of replacing the differential operators in the governing equations by appropriate difference operators. As a result, the original differential equations are converted into a system of algebraic equations from which fluid flow solutions may be obtained directly.

In the same spirit, the formulation of numerical boundary conditions follows directly from the discretisation of the boundary conditions of the continuous case. However, care must taken in turning the bulk and boundary descriptions of the problem into their respective numerical approximations. This task should satisfy the following guideline:

> The accuracy taken in the discrete formulation of boundary conditions shall, in principle, never be inferior to the level of approximation of the bulk solution. Otherwise, the accuracy of the solution may be degraded *everywhere*.

The above statement aims to dismiss a naive argument which is often mistakenly employed by beginners in numerical methods. Such an argument advocates that the numerical accuracy of boundary conditions should be an issue of minor concern as, whatever their approximation level is, it will only affect a small fraction of all the grid points in the simulated domain. Unfortunately, the preceding argument is incorrect! The accuracy of the boundary condition necessarily interferes with the level of approximation reached in the bulk as updates of the bulk points uses data from the boundary. NOTE (EMV): Maybe you want to formulate it even more strongly, such as "the values of the bulk point are dependent on values from the boundary"? In order to leave no doubts on the reader about this issue, let us illustrate it through a practical example.

Suppose we are interested in the finite difference solution of an 1D problem. For concreteness, the problem can be assumed to model a thin film of an incompressible Newtonian fluid subject to an external acceleration a_x as depicted in fig. 5.1.

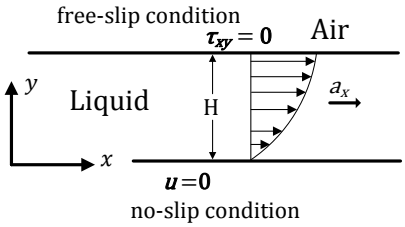


Fig. 5.1: Thin film of liquid fluid flowing due to an external acceleration a_x . It is assumed that the ambient air is quiescent and the surface tension of the liquid is negligible.

Mathematically, the above problem is described by: NOTE (EMV): Just to be super clear, might we want to very quickly point out that this comes from the time-independent Navier-Stokes equation?

$$\nu \frac{\partial^2 u_x}{\partial y^2} = -a_x, \tag{5.5}$$

subject to the following boundary conditions:

$$u_x(y = 0) = 0, \tag{5.6a}$$

$$\left.\frac{\partial u_x}{\partial y}\right|_{(y=H)} = 0. \tag{5.6b}$$

Notice that different kinds of conditions apply at each end of the fluid domain: a Dirichlet (no-slip velocity) condition at $y = 0$ and a Neumann (free-slip velocity) condition at $y = H$. The problem solution yields the well-known Poiseuille parabolic velocity profile:

$$u_x(y) = -\frac{a_x}{2\nu} y\,(y - 2H)\,. \tag{5.7}$$

The finite difference formulation of this problem replaces the derivatives in eq. (5.5) by appropriate finite difference operators. In this way, the target solution, given by the continuous velocity field $u_x(y)$, gets replaced by the vector u_j, which approximates u_x at the discrete grid points y_j. For simplicity, the spatial discretisation here adopted makes use of an uniform mesh with spacing Δy, given by $\Delta y = \frac{H}{N-1}$. Thereby, the grid points are placed at $y_j = (j-1)\Delta y$, with $j = 1, \ldots, N$. Approximating the differential operator in eq. (5.5) by a second-order central scheme, its finite difference representation reads

$$\nu\,\frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta y^2} = -a_x \tag{5.8}$$

The discrete approximations of boundary conditions are:

$$u_1 = 0, \tag{5.9a}$$

$$(1) \quad \left.\frac{\partial u_x}{\partial y}\right|_{(y=H)} \simeq \frac{u_N - u_{N-1}}{\Delta y} = 0, \tag{5.9b}$$

$$(2) \quad \left.\frac{\partial u_x}{\partial y}\right|_{(y=H)} \simeq \frac{u_{N+1} - u_{N-1}}{2\Delta y} = 0, \tag{5.9c}$$

where, for comparison purposes, the spatial derivative in the Neumann condition eq. (5.6) is discretised either with formulation (1) or (2). Although both discretisation schemes are consistent with the continuous problem, meaning that, in either case, eq. (5.6) is recovered in the continuum limit (*i.e.* when $\Delta y \to 0$), they provide different approximations at the discrete level. The first approach (1) uses a *first-order* one-sided difference scheme, while the second approach (2) employs a *second-order* central difference approximation. Similarly to the bulk case, the order of accuracy of these difference operators at the boundary is determined by the truncation error in the approximation of the differential operator (*cf.* section 2.1.1).

Before proceeding further, it is instructive to explain case (2) in more details. The approximation of the Neumann condition through a centered (second-order) scheme uses a "numerical trick" often employed in finite difference schemes.[4] It consists in assuming the presence of an additional node, placed beyond the boundary, where the Neumann condition is assigned. In the above example, this boils down to the consideration of a virtual site at $j = N+1$ as shown in fig. 5.2. Owing to the inclusion

[4] The understanding of approach (2) also proves useful in the LB context. In fact, by exploiting the similarity between LB and finite difference schemes, a procedure similar to approach (2) was proposed as a way to prescribe boundary conditions in the LB method. The original suggestion from Chen and co-workers [5] consisted of a plain transposition of the additional virtual node "trick" into the LB framework, thus meaning the consideration of additional (virtual) lattice site. Afterwards, and thanks to a more attentive consideration of the LB theory, more refined versions of this method were developed, *e.g.* [6, 7, 8, 9].

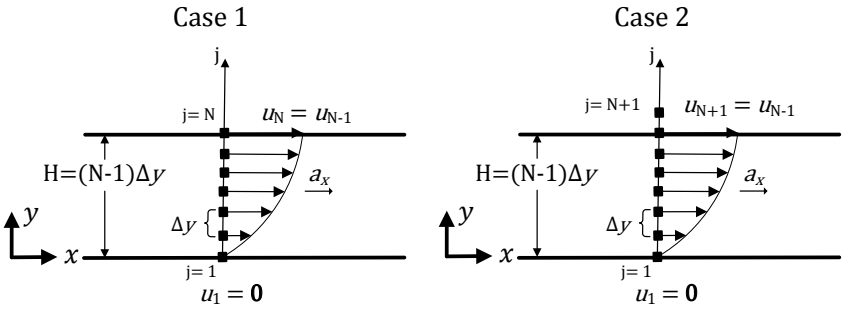


Fig. 5.2: Sketch of the finite difference formulation of the problem depicted in fig. 5.1. Comparison of two approaches to specify the Neumann boundary condition at $y = H$ using the finite difference method.

of such an additional node, one is provided with an extra equation describing the dynamics of the fluid at the boundary $j = N$. This equation, which is similar in form to that in the bulk, $\nu \frac{u_{N+1}-2u_N+u_{N-1}}{\Delta y^2} = -a_x$, must, however, be corrected by the boundary condition, $u_{N+1} = u_{N-1}$. As a result, the numerical approximation of the continuous problem at the boundary attains the desired second-order accuracy: $\nu \frac{-u_N+u_{N-1}}{\Delta y^2} = -\frac{1}{2}a_x$.

Depending on whether we take case (1) or case (2) to approximate the Neumann condition, the finite difference solution of eq. (5.8) reads:

$$(1) \qquad u_j = -\frac{a_x}{2\nu} y_j \left(y_j - 2H + \Delta y\right), \tag{5.10a}$$

$$(2) \qquad u_j = -\frac{a_x}{2\nu} y_j \left(y_j - 2H\right). \tag{5.10b}$$

These solutions are illustrated in fig. 5.3.

From the above solutions it is immediate that eq. (5.10b) reproduces *exactly* the analytical parabolic solution, described by eq. (5.7). On the other hand, although eq. (5.10a) also predicts a parabolic solution, its profile differs from the correct one due to the Δy term. This term originates from the lower accuracy of the boundary condition in case (1), and it gives rise to an incorrect velocity slope at the boundary, which, as eq. (5.10a) shows, will affect the u_j solution everywhere.

This example demonstrates that, even if the discretisation is second-order accurate at all bulk points, it suffices one point (in this case the Neumann boundary point) featuring a lower-order accuracy to degrade the overall solution accuracy.

Exercise 5.1. Using as starting point the previous example, compute the accuracy of both above numerical schemes as function of the grid resolution. For this task, take several grid refinement values at your choice. Then, answer: 1. How does the accuracy of each solution vary with the mesh size? 2. What is the relation between this result and the accuracy of the numerical approximation taken?

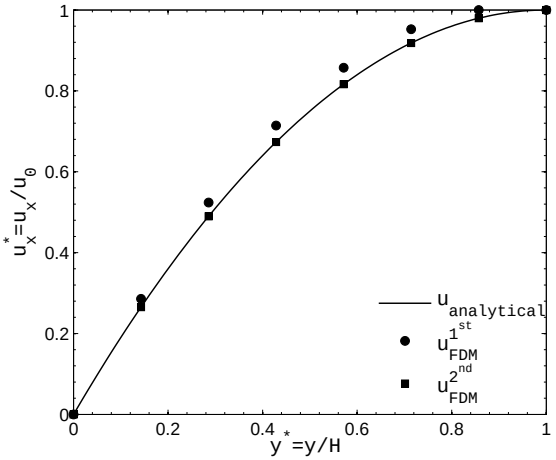


Fig. 5.3: Velocity solutions of eq. (5.7), eq. (5.10a) and eq. (5.10b). Velocity plot is made non-dimensional with the centerline velocity $u_0 = \frac{a_x H^2}{2\nu}$. The numerical solutions are obtained for a resolution of $N = 8$.

Suggestion: As a measure for computing the accuracy of the numerical solutions use the L2 error norm (*cf.* section **??**).
Answer:

1. In case (1), the numerical approximation becomes more accurate with mesh refining, displaying a first-order improvement. In case (2), the numerical solutions remain constant, *i.e.* they are mesh-independent.
2. This example takes numerical approximations on both bulk and boundary descriptions of the continuous problem.
   In the bulk, the viscous term is discretised adopting a second-order finite difference scheme. This approximation reads $\frac{\partial^2 u_x}{\partial y^2} = \frac{u_{j+1}-2u_j+u_{j-1}}{\Delta y^2} + E_T$, where the truncation error is $E_T = -\frac{\Delta y^2}{12}\frac{\partial^4 u_x}{\partial y^4} - \frac{\Delta y^4}{360}\frac{\partial^6 u_x}{\partial y^6} - O\left(\Delta y^8\right)$. Since we are searching for a parabolic u_x solution, then it follows that all derivatives higher than second order are identically zero. This implies $E_T = 0$, rendering the numerical approximation in the bulk exact.
   In the description of the Neumann boundary condition we have considered two cases. Case (1): $\frac{\partial u_x}{\partial y} = \frac{u_j-u_{j-1}}{\Delta y} + E_T$ with $E_T = -\frac{\Delta y}{2}\frac{\partial^2 u_x}{\partial y^2} + \frac{\Delta y^2}{6}\frac{\partial^3 u_x}{\partial y^3} - O\left(\Delta y^3\right)$, and case (2): $\frac{\partial u_x}{\partial y} = \frac{u_{j+1}-u_{j-1}}{2\Delta y} + E_T$ with $E_T = -\frac{\Delta y^2}{6}\frac{\partial^3 u_x}{\partial y^3} - \frac{\Delta y^4}{120}\frac{\partial^5 u_x}{\partial y^5} - O\left(\Delta y^5\right)$. It follows that case (1) yields $E_T \neq 0$ because the $O(\Delta y)$ term is non-vanishing for a parabolic solution. This error is the cause for the inaccuracies identified in the numerical results of case (1). On the contrary, case (2) keeps $E_T = 0$, which

proves the exactness of this numerical approximation to describe the parabolic solution.

NOTE (EMV): The unindented paragraphs in point 2 make me sad. :(

The message to retain from the above exercise can be comprehended as follows. Taking any numerical scheme that approximates the continuous equation in bulk with *n-order accuracy*, then a simple test to evaluate whether the boundary condition satisfies the same n-order order accuracy is to assess its ability to exactly solve a *n-order polynomial flow*. We will use this idea extensively later when discussing the accuracy of LB boundary conditions.

5.1.4 Boundary conditions in lattice Boltzmann methods

So far, the subject of boundary conditions was focused on analytical and standard numerical procedures for fluid flow problems. The rest of the chapter addresses this subject in the frame of the LB method. The present introductory section aims at explaining the following three points:

- How are boundary conditions characterised in the LB method?
- What differentiates them from other more traditional numerical methods in fluid dynamics?
- How can they be implemented?

Throughout this chapter, we will concentrate on local techniques to implement boundary conditions. Such techniques are particularly suited to describe straight boundaries, aligned with the lattice directions. Although more complex boundary shapes and/or orientations can also be handled within this framework, the resulting geometry will display a staircase representation of the true boundary. The implications of this geometrical approximation are discussed in section 11.1. A solution to treat complex boundary configurations as smooth surfaces usually comes at the price of increased complexity and/or need of data from neighboring nodes, making them non-local schemes.[5] A discussion of such "curved boundary techniques" is given in chapter 11.

At last, we underline that the content of this chapter views essentially 2D applications. Still, the majority of LB boundary conditions here covered can be naturally transposed from 2D to 3D domains. The last section of this chapter aims at discussing possible additional complexities arising in 3D problems.

[5] Noticeable exceptions are the local schemes developed in **??** and [10].

Introductory concepts and background

Let us start by introducing some important definitions on the subject of LB boundary conditions. To this end, let us address the following two questions: (i) What are boundary nodes in the LB method? and (ii) How can we identify them?

The concept of node, or lattice site, arises in the mapping of a continuous domain into a discrete configuration space. In this context, a lattice site has to fit into one of the following three node categories: *fluid node*, *solid node* or *boundary node*.

- Fluid nodes refer to sites where the LB equation applies.
- Solids nodes refer to sites completely covered by the solid object, where there is no need to solve the LB equation.
- Boundary nodes refer to sites linking fluid and solid nodes, where special dynamical rules need to be considered.

According to the above definitions [11, 12, 13], fluid nodes can be identified as nodes linking exclusively to other fluid or boundary nodes; solid nodes as nodes linking exclusively to other solid or boundary nodes; and boundary nodes as nodes having, at least, one link to a solid node. These nodes definitions are illustrated in fig. 5.4.

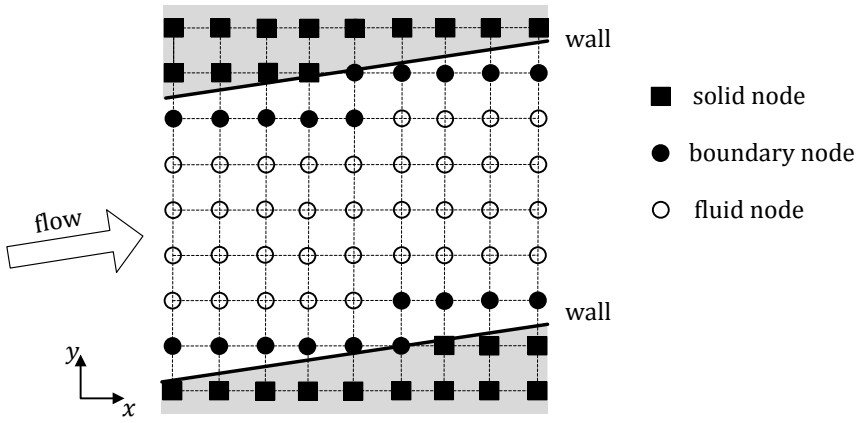


Fig. 5.4: Illustration of fluid nodes, solid nodes and boundary nodes in an inclined channel flow geometry (where grey shaded domain denotes the solid region). NOTE (EMV): The legend says “bounday”; whoopsie? :)

The *problem with boundary nodes* is that, contrary to fluid nodes, the LB equation cannot be directly applied to them. In alternative, boundary nodes must be subject to specific rules in order to complete the missing information caused by their connectivity to the solid object. Given the structure of an LB algorithm, which consists of a succession of collision and streaming steps, this missing information can

be comprehended as affecting the completion of the streaming step, like depicted in fig. 5.5. NOTE (EMV): This paragraph can be formulated more simply; it's very important and it is a bit heavy to read right now. In particular, it is not clear for a new reader why information is missing at the boundary nodes, and I fear this might be confusing even though it's explained right below.

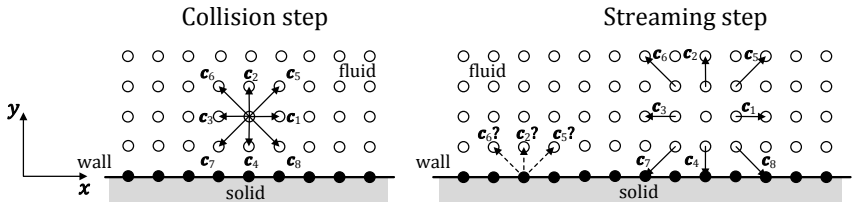


Fig. 5.5: Sketch of the boundary problem experience by the LB algorithm. During the streaming step populations belonging to fluid nodes (denoted by continuous arrows) will go to neighbouring nodes. On the contrary, populations on boundary nodes pointing to the inner domain (denoted by dashed arrows) cannot be streamed inside due to the lack of neighbouring populations from where information could come from. In order to find their post-streamed values they must be subject to specific rules, not contemplated by the LB equation. NOTE (EMV): I'm not a fan of huge figure captions; is it possible to put most of this information in the main text? Also, "contemplated"?

The *role of LB boundary conditions* is to prescribe adequate values for the incoming populations, *i.e.* those leaving the solid object into the fluid region, whose values are unknown due to the lack of neighbouring nodes where information could come from during the streaming step.

It turns out that, compared to conventional numerical solvers for the NSEs, setting boundary conditions in LB schemes is more complicated [14]. Rather than specifying the macroscopic variables of interest at the boundary, in the LB method we prescribe conditions for the *mesoscopic populations*.

The *fundamental difficulty* of this task is that the system of macroscopic variables has fewer degrees of freedom than the corresponding mesoscopic system. NOTE (EMV): I reformulated the previous sentence a bit. In other words, although it is relatively straightforward to obtain macroscopic quantities from a known mesoscopic solution (recall that the former are given by the velocity moments of the latter), the one-to-one mapping from a lower to a higher degree of freedom system, besides being a non-trivial task, leads, in general, to non-unique solutions.

Types of boundary conditions

Because of the aforementioned non-uniqueness feature, it is possible to find distinct formulations of LB boundary conditions that attain "equally consistent" hydrodynamic behaviour [15].[6] This explains the zoo of approaches existing in this field, which is evidenced in the over XX works published until 2014, based on the Web of Knowledge webpage. NOTE (EMV): "Patent"?

Despite the number of strategies presently available in the literature, all LB boundary conditions, developed for straight boundaries, reduce to one of the following two groups:

- The *link-wise* family where the boundary is considered to lie on the *lattice links*.
- The *wet-node* family which locates the boundary on the *lattice nodes*.

As it will be evidenced later on in this chapter, each of these groups relies on a quite different interpretation of the boundary concept. While the correct implementation of link-wise schemes realises the "physical" boundary as located on the lattice links, approximately[7] half-way between the solid and boundary nodes [16, 18, 17, 19, 11], wet-node boundary conditions take the boundary location as coinciding with the boundary nodes, which are assumed to lie infinitesimally within the fluid domain [20, 21, 22, 23, 24, 25, 14, 26]. This difference is illustrated in fig. 5.6.

5.2 Boundary condition methods

5.2.1 Periodic boundary conditions

Periodic boundary conditions arise from specific flow symmetry considerations and are intended to isolate a repeating flow pattern within a cyclic flow system. A common mistake performed by beginners is to confuse the periodicity of the flow geometry with the periodicity of the flow itself! As illustrated in fig. A.1 these two cases are not necessarily synonymous.

[6] Notice that the same problem is found in the specification of hydrodynamically consistent initial conditions in the LBM (*cf.* section 5.4).

[7] The word "approximately" is purposely adopted herein because in link-wise schemes the exact boundary location is not fixed, but depends on the details of the problem, such as the orientation of the boundary with respect to the lattice and, if using the SRT collision model, the fluid viscosity [16, 17, 11]. We will discuss this last feature in section 5.2.3 via numerics, and in section 5.3.1 through theoretical arguments.

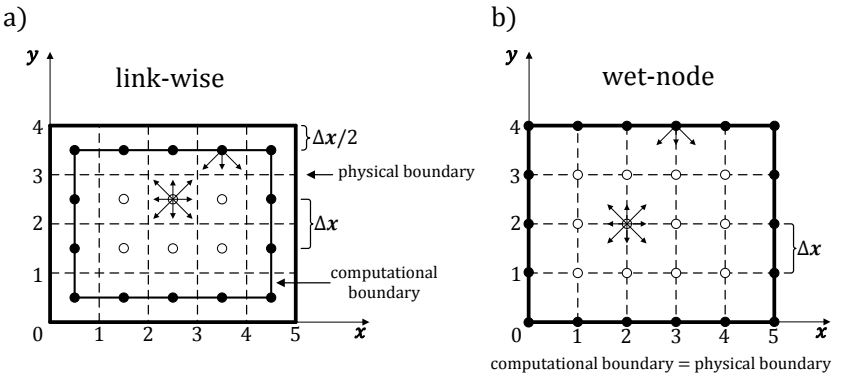


Fig. 5.6: Discretisation of the same flow configuration considering the following approaches as boundary conditions: a) link-wise or b) wet-node. Fluid nodes ∘, boundary nodes •. In the link-wise approach the grid nodes locate in the center of the computational cells, while in the wet node approach they locate at the cell vertexes. Consequently, in the link-wise approach the computational boundary is shifted half grid spacing away from the physical boundary, while in the wet node approach both computational and physical boundaries match. Evidently, the number of lattice nodes required for resolving the same domain will differ depending on the approach chosen. NOTE (EMV): It took a little while for me to understand the left subfigure. I think it might be easier if you keep the dashed grid on the nodes in both figures, making for a more consistent look. Again, putting this explanation in the main text would be nice.

We underline the concept of *periodic boundary condition applies only to the situation where the flow solution is periodic*. The periodic flow condition states that the fluid leaving the domain on one side will, instantaneously, re-enter the domain at the opposite side.

Keeping this in mind, we recognise that periodic boundary conditions *conserve mass and momentum* at all times. Nonetheless, if the flow is periodic all over, it can only survive over time providing an external source of momentum supports it. Otherwise, and regardless its initial state, the flow will decay towards a state of homogeneous velocity (which can be non-zero), due to the action of viscosity.

For the NSEs, the application of the periodic flow condition reads as follows:

$$\rho(\boldsymbol{x}, t) = \rho(\boldsymbol{x} + \boldsymbol{L}, t), \tag{5.11a}$$

$$\rho\boldsymbol{u}(\boldsymbol{x}, t) = \rho\boldsymbol{u}(\boldsymbol{x} + \boldsymbol{L}, t), \tag{5.11b}$$

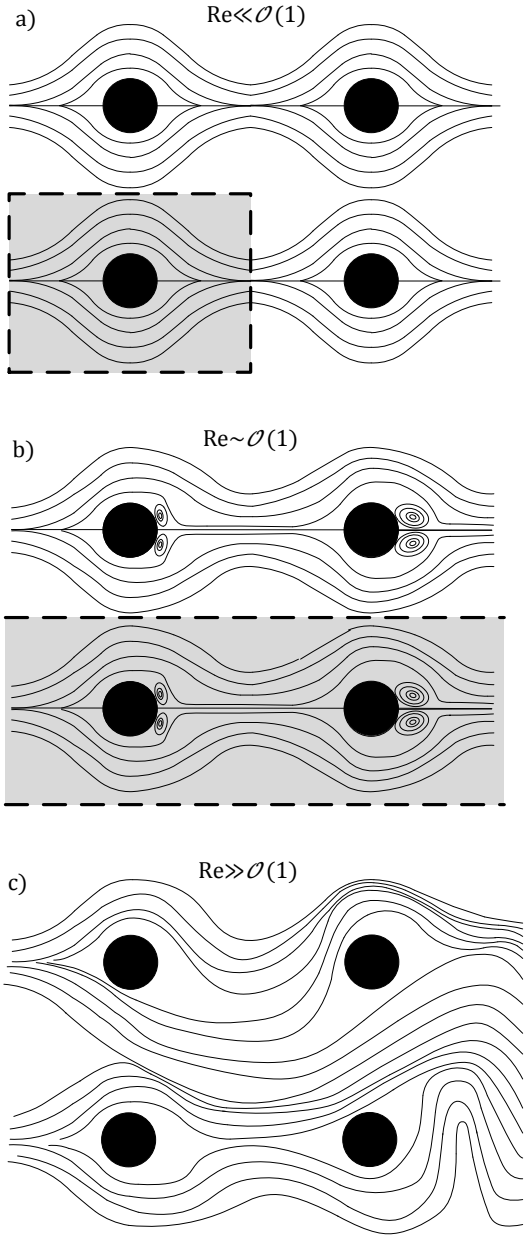


Fig. 5.7: Flow around periodic array of cylinders. The picture aims to show that the flow within a periodic geometry may not necessarily display a periodic structure. In this case, depending on the Reynolds number of the flow, the periodicity of the flow solution will vary as follows: a) In the low Re (Stokes flow) regime the symmetry of the flow solution coincides with symmetry of the geometrical domain. Hence, its study can be reduced to the grey shaded domain in plot a). b) In the intermediate Re regime, wakes start to form at the rear of cylinders. With these structures the symmetry along the horizontal direction breaks down. Yet, since wakes at different rows do not interact vertical symmetry holds. Altogether, the study can be reduced to the grey shaded domain in plot b). c) In the large Re regime wakes become unstable. This leads to interferences in the flow from surrounding cylinders. In this case the flow solution features no symmetry. NOTE (EMV): Very cool figure, but it's slightly hard telling the subfigures apart. Also, we can save space by making each subfigure slightly smaller and putting a) and b) right next to each other, and c) centred below both. As it is, the big figure and the big caption is breaking the page layout by simply being too big (not counting this overly-long note, of course).

where the vector $\boldsymbol{L}$ denotes the length of the cyclic flow pattern along the periodicity axes.

The periodic condition is quite intuitively transposed to the LB method [27, 28, 29]. Its implementation boils down to replacing the unknown incoming populations by those leaving the domain at the opposite side. This procedure applies for the post-collision populations $f_i^\star$ (*i.e.* the state before the streaming step) and reads as follows:

$$f_i^\star(\boldsymbol{x}, t) = f_i^\star(\boldsymbol{x} + \boldsymbol{L}, t). \tag{5.12}$$

For the 2D flow problem illustrated in fig. 5.8, the application of the periodic flow condition along the x direction is described as follows:

$$f_i^\star(\boldsymbol{x}, t) = f_i^\star(\boldsymbol{x} + \boldsymbol{L}, t) \Rightarrow \begin{cases} f_1^\star(x_0, y_2, t) = f_1^\star(x_N, y_2, t) \\ f_5^\star(x_0, y_2, t) = f_5^\star(x_N, y_2, t) \\ f_8^\star(x_0, y_2, t) = f_8^\star(x_N, y_2, t) \end{cases} \tag{5.13a}$$

$$f_i^\star(\boldsymbol{x} + \boldsymbol{L}, t) = f_i^\star(\boldsymbol{x}, t) \Rightarrow \begin{cases} f_3^\star(x_{N+1}, y_2, t) = f_3^\star(x_1, y_2, t) \\ f_6^\star(x_{N+1}, y_2, t) = f_6^\star(x_1, y_2, t) \\ f_7^\star(x_{N+1}, y_2, t) = f_7^\star(x_1, y_2, t) \end{cases} \tag{5.13b}$$

where, for computational convenience, additional layers of (virtual) nodes[8] are considered before and after the periodic boundaries, i.e. $x_0 = x_1 - \Delta x$ and $x_{N+1} = x_N + \Delta x$, respectively. This algorithm is equally simple to implement in 2D and 3D problems.

An equivalent way of implementing the periodic condition is also possible *without* taking virtual nodes. For that, we have to explicitly consider the edges from opposite sides of the flow domain as if they were attached together. In other words, as if the system would close over itself. As a result, the periodic boundary condition operates over the LB algorithm as a completion step to the streaming process, *i.e.* it assigns to the post-streaming populations, which enter the domain on one side, the values of the post-collision populations, which leave the domain on the opposite side, as in eq. (5.14).

$$f_i(\boldsymbol{x}, t + \Delta t) = f_i^\star(\boldsymbol{x} + \boldsymbol{L} - \boldsymbol{c}_i \Delta t, t). \tag{5.14}$$

Applying eq. (5.14) to the 2D flow problem illustrated in fig. 5.8 it reads:

[8] TODO (OS): A technical justification for the computational convenience of considering these extra layers of nodes is given in Chapter **??** (implementation chapter).

$$f_i(\boldsymbol{x}, t+\Delta t) = f_i^{\star}(\boldsymbol{x}+\boldsymbol{L}-\boldsymbol{c}_i\Delta t, t) \Rightarrow \begin{cases} f_1(x_1, y_2, t+\Delta t) = f_1^{\star}(x_N, y_2, t) \\ f_5(x_1, y_2, t+\Delta t) = f_5^{\star}(x_N, y_1, t) \\ f_8(x_1, y_2, t+\Delta t) = f_8^{\star}(x_N, y_3, t) \end{cases} \tag{5.15a}$$

$$f_i(\boldsymbol{x}+\boldsymbol{L}, t+\Delta t) = f_i^{\star}(\boldsymbol{x}-\boldsymbol{c}_i\Delta t, t) \Rightarrow \begin{cases} f_3(x_N, y_2, t+\Delta t) = f_3^{\star}(x_1, y_2, t) \\ f_6(x_N, y_2, t+\Delta t) = f_6^{\star}(x_1, y_1, t) \\ f_7(x_N, y_2, t+\Delta t) = f_7^{\star}(x_1, y_3, t) \end{cases} \tag{5.15b}$$

Although eq. (5.12) and eq. (5.14) are absolutely identical; in general, the former is easier to handle. This justifies the preference for adding virtual nodes in the implementation procedure. In any case, the periodic boundary remains at the same place, *cf.* fig. 5.8. NOTE (EMV): I simplified this sentence slightly.

The LB periodic boundary condition puts the physical periodic boundary at the lattice links approximately halfway between the boundary nodes. NOTE (EMV): I also tweaked this one.

In this context, the periodic boundary condition belongs to the link-wise category. This means that it shares similar advantages/disavantages of other link-wise methods, such as the bounce-back rule discussed ahead. We will illustrate this point in Example 5.2, where we will show that the periodic boundary location is also affected by viscosity when adopting the SRT collision model.

In a fluid flow simulation, the consideration of periodic boundary conditions is primarily justified by physical arguments: to isolate an solitary cell in a repeating flow pattern. Furthermore, it may be used to enforce the fully-developed condition in a channel flow or to establish the appropriate conditions for describing homogeneous isotropic turbulence.

However, periodic boundaries may find other, less physically oriented, purposes. For example, in solving 2D flow problems with a pre-existing 3D numerical code, the most effortless way to carry out this task consists in adopting the periodic flow condition along one of the cartesian directions. Obviously, in this situation the system size, along the periodicity direction, should be chosen as small as possible (one, ideally; but sometimes more lattice sites are required, depending on the code). NOTE (EMV): I think this is a really important and too-often-overlooked point, so I grayboxed it and added the following sentence: (Ideally though, such lower-dimensional problems should be simulated using an appropriate lower-dimensional velocity set in order to minimise memory requirements and computational time.)

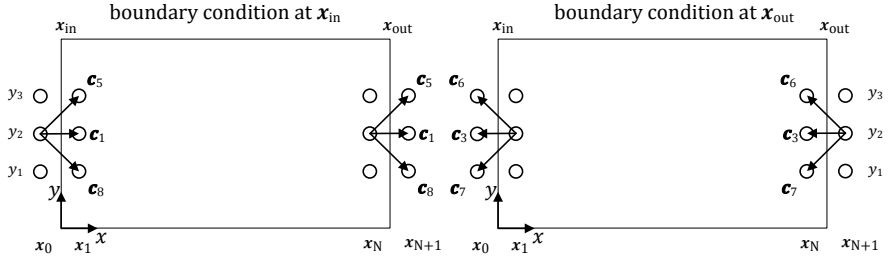


Fig. 5.8: Schematic representation of the periodic flow boundary condition in the LBM, inspired by [30]. Additional layers of (virtual) nodes are considered before and after the periodic boundaries, *i.e.* at $x_0 = x_1 - \Delta x$ and $x_{N+1} = x_N + \Delta x$, respectively. We call these nodes virtual because their values have no meaning in the physical solution. They are considered just for computational convenience. Inlet and outlet periodic boundaries are, respectively, at $x_{\text{in}} = (x_1 + x_0)/2$ and $x_{\text{out}} = (x_{N+1} + x_N)/2$, with the periodicity length defined as $L = x_{\text{out}} - x_{\text{in}} = N\Delta x$. Note, the location of periodic boundaries is same also when virtual nodes are not considered.

5.2.2 Periodic boundary conditions with pressure variations

Notwithstanding the usefulness of the standard periodic boundary conditions, there are flow problems where one may be interested in enforcing the periodicity of the velocity field, but not the pressure (or the density). Since in an isothermal fluid flow the fluid density ρ and pressure p are related to each other by a multiplicative constant, it follows immediately that eq. (5.11a) is no longer valid. Strategies to cope with this particular case of periodicity exist and are called generalised periodic boundary conditions. They modify eq. (5.11) as follows:

$$\rho(\boldsymbol{x}, t) = \rho(\boldsymbol{x} + \boldsymbol{L}, t) + \Delta\rho, \tag{5.16a}$$

$$\rho\boldsymbol{u}(\boldsymbol{x}, t) = \rho\boldsymbol{u}(\boldsymbol{x} + \boldsymbol{L}, t), \tag{5.16b}$$

where $\Delta\rho$ is a prescribed variation of the density (pressure) along $\boldsymbol{L}$. The momentum condition is kept unchanged.

However, this condition is affected by a drawback: periodicity applies over fluid momentum, eq. (5.16b). Hence, mass conservation and periodicity of velocity cannot be enforced at the same time. To overcome this limitation it may be beneficial to use the LB incompressible variant [31, 32] (*cf.* section 5.3.1). For incompressible flows the generalised periodic boundary condition reads:

$$p(\boldsymbol{x}, t) = p(\boldsymbol{x} + \boldsymbol{L}, t) + \Delta p, \tag{5.17a}$$

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}(\boldsymbol{x} + \boldsymbol{L}, t), \tag{5.17b}$$

where Δp is a prescribed variation of the pressure.

Generalised periodic boundary conditions, like those presented herein, have been used for quite a long time in standard numerical solvers for NSEs [33]. Yet, only recently they have been developed for the LB equation [34, 30, 35]. A simple and robust procedure to implement this condition in the LBM has been proposed by Kim and Pitsch [30], and it works as follows.

We start by considering the presence of an extra-layer of virtual nodes at both ends of periodic boundaries. Following fig. 5.8 we assume these extra nodes to locate at x_0 and x_{N+1}. Then, we decomposed the populations on these virtual nodes in equilibrium f_i^{eq} and non-equilibrium f_i^{neq} parts.

The equilibrium part is recomputed as follows:

$$f_i^{\mathrm{eq}}(x_0, y, t) = f_i^{\mathrm{eq}}(p_{\mathrm{in}}, \boldsymbol{u}_N), \tag{5.18a}$$

$$f_i^{\mathrm{eq}}(x_{N+1}, y, t) = f_i^{\mathrm{eq}}(p_{\mathrm{out}}, \boldsymbol{u}_1), \tag{5.18b}$$

where $\boldsymbol{u}_N$ and $\boldsymbol{u}_1$ denote the velocity at nodes x_N and x_1, respectively. The subscripts "in" and "out" in p denote the pressure values at the inlet and outlet physical boundaries, respectively. Hence, the actual pressure drop along the periodicity length $L = N\Delta x$ is $\Delta p = p_{\mathrm{out}} - p_{\mathrm{in}}$. In case of adopting the (compressible) standard LB equilibrium then eq. (5.18) becomes $f_i^{\mathrm{eq}}(\rho_{\mathrm{in}}, \rho\boldsymbol{u}_N)$ and $f_i^{\mathrm{eq}}(\rho_{\mathrm{out}}, \rho\boldsymbol{u}_1)$.

The non-equilibrium part is copied from the corresponding image point inside the real domain:

$$f_i^{\mathrm{neq}}(x_0, y, t) = f_i^{\mathrm{neq}}(x_N, y, t), \tag{5.19a}$$

$$f_i^{\mathrm{neq}}(x_{N+1}, y, t) = f_i^{\mathrm{neq}}(x_1, y, t), \tag{5.19b}$$

where non-equilibrium populations are determined from the post-collision state, *e.g.* $f_i^{\mathrm{neq}}(x_N, y, t) = f_i^{\star}(x_N, y, t) - f_i^{\mathrm{eq}}(x_N, y, t)$.

Finally, we merge the equilibrium part, eq. (5.18), with the non-equilibrium part, eq. (5.19), for instance, $f_i^{\star}(x_0, y, t) = f_i^{\mathrm{eq}}(x_0, y, t) + f_i^{\mathrm{neq}}(x_0, y, t)$, and proceed with the LB algorithm towards the streaming step.

Exercise 5.2. Implement the inlet/outlet boundary conditions for a streamwise invariant flow driven by a pressure difference Δp. Take the geometry depicted in fig. 5.8 for the periodic flow configuration.

Answer:

Incompressible flows define the pressure up to an arbitrary constant. Thereby, we can arbitrarily set $p_{\mathrm{out}} = 1$ (simulation unit) and compute the outlet pressure as $p_{\mathrm{in}} = p_{\mathrm{out}} + \Delta p$. The generalised periodic boundary condition then prescribes for the post-collision populations as follows:

- Inlet boundary condition:

$$f_i^{\star}(x_0, y, t) = f_i^{\mathrm{eq}}(p_{\mathrm{in}}, \boldsymbol{u}_N) + \left(f_i^{\star}(x_N, y, t) - f_i^{\mathrm{eq}}(x_N, y, t)\right), \tag{5.20}$$

  with $i = \{1, 5, 8\}$.

- Outlet boundary condition:

$$f_i^{\star}(x_{N+1}, y, t) = f_i^{\mathrm{eq}}(p_{\mathrm{out}}, \boldsymbol{u}_1) + \left(f_i^{\mathrm{eq}}(x_{N+1}, y, t) - f_i^{\mathrm{neq}}(x_{N+1}, y, t)\right), \quad (5.21)$$

  with $i = \{3, 6, 7\}$.

Note that, in alternative, eq. (5.20) and eq. (5.21) could have been applied to all populations $i = 0, 1, 2, \ldots, 8$. This is irrelevant as populations locate outside the physical domain. In this case, only populations pointing into the fluid region will be of interest for the solution.

5.2.3 Solid boundaries – the bounce-back approach

In hydrodynamics, the most common fluid-solid interface condition is the no-slip velocity boundary condition. As such, its correct implementation is crucial in the modelling of confined fluid flow and other problems involving solid boundaries.

In the LB field, the oldest boundary condition to model walls is the bounce-back scheme, where most noticeable contributions are, *e.g.*, [**?**, 36, 37, 16, 18, 17, 38, 11]. Its concept was imported from earlier lattice gas automata (LGA), *e.g.* [**?**, **?**, **?**] and section **??**. Notwithstanding its age, it still stands as the most popular wall boundary scheme in the LB community, mostly owing to its simplicity of implementation.

Given the importance of the bounce-back scheme as LB boundary condition, this section is entirely devoted to an overview of this technique. First, we introduce the bounce-back principle based on a particle-based picture. Then, we discuss two ways of undertaking the bounce-back rule, and point out which one should be preferred. Afterwards, we concentrate on the application of the bounce-back method for the modelling of (i) stationary and (ii) moving walls. Finally, we consolidate all previous theoretical elements with the analysis of two practical problems.

Principle of the bounce-back method

The working principle behind the bounce-back rule lies in the following idea:

> Particles hitting a rigid wall will be reflected back to where they originally came from.

Although the hydrodynamic significance of this statement may not be immediate, one may acknowledge it if imagining, for a while, particles as embodying fluid matter.[9] Then, it follows that *particles bouncing after hitting the wall* will imply *no flux crossing the boundary*, *i.e.* the wall is impermeable to the fluid. Similarly, the

[9] In an attempt to make this explanation on the bounce-back method more intuitive we will often refer to the parameter f_i as "particles", instead of "LB populations", as it should be called in rigour.

fact that *particles are bounced back*, rather than bounced forward, will imply *no relative transverse motion between fluid and boundary*, *i.e.* the fluid does not slip on the wall. These points illustrate how the bounce-back particle dynamics mirrors the Dirichlet boundary condition for the macroscopic fluid velocity solution at the wall, see fig. 5.9.

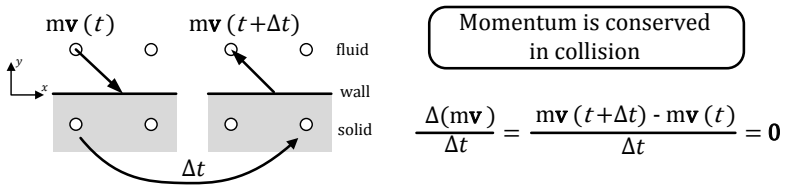


Fig. 5.9: Sketch of moving particle hitting a rigid wall, with mass m and velocity **v**. During the collision process both normal and tangential momentum components are conserved, *i.e.* $-\mathrm{mv}_y(t) + \mathrm{mv}_y(t+\Delta t) = 0$ and $\mathrm{mv}_x(t) - \mathrm{mv}_x(t+\Delta t) = 0$. NOTE (EMV): Nice figure, but you might want to rethink the Δt arrow a bit. Now it looks like it points from one wall node to another.

However, fig. 5.9, should not be taken too literally. A rigorous analysis of the macroscopic behaviour of the bounce-back rule (as any other mesoscopic model) requires the use of a multi-scale expansion technique, such as the Chapman-Enskog method (*cf.* section 5.3.1).

The bounce-back process can be realised in two different ways:

1. The first strategy, called *fullway bounce-back* [29], considers that particles travel the complete link path, from the boundary to the solid node, where the inversion of the particle velocity is expected to occur during the next collision step, see fig. 5.10(a).
2. The second strategy, called *halfway bounce-back* [18], considers that particles traverse only half of the link distance, so that the inversion of the particle velocity takes place during the streaming step, see fig. 5.10(b).

Each strategy introduces specific modifications in the LB algorithm. The local realisation of the fullway bounce-back has to explicitly consider solid nodes, where populations will be inverted during the collision step. On the other hand, in the halfway bounce-back solid nodes are not necessary as the inversion of populations occurs during the streaming of particles. In summary, while the fullway bounce-back replaces the collision step but leaves the usual streaming step unchanged, the halfway bounce-back does not modify the collision step, but reshapes the streaming step. Despite what their names might suggest, both fullway and halfway viewpoints are understood to place the boundary *approximately midway* between solid and boundary nodes.

We recall that LBM is *not* a true particle method, such as those presented in chapter 2. Rather, LBM deals with discretised forms of continuous fields (*cf.* chapter 3).

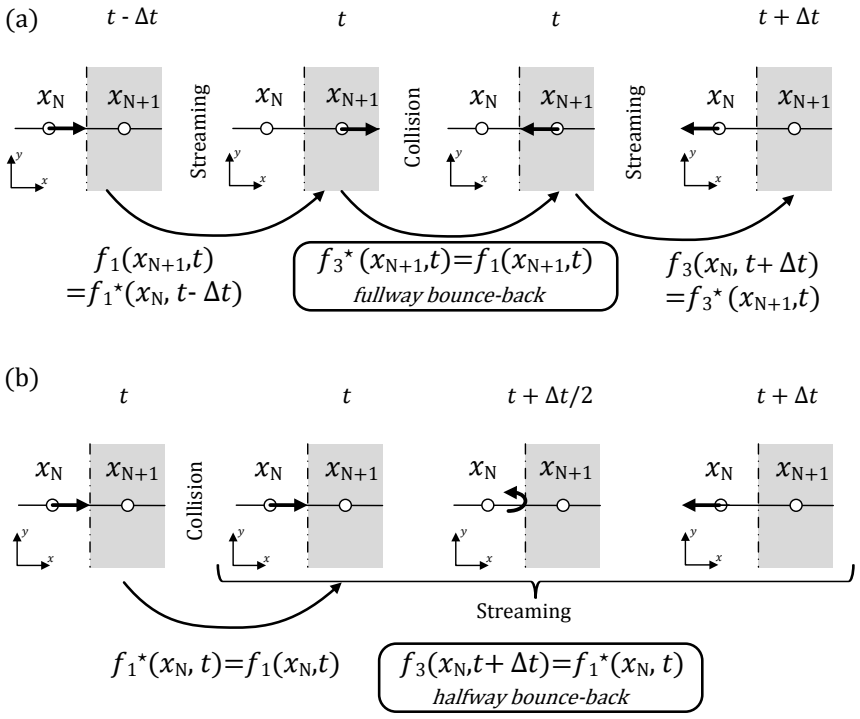


Fig. 5.10: Illustration of the time evolution for (a) fullway bounce-back and (b) halfway bounce-back in 1D. The time step is shown at the top of each action taken. In all pictures the arrow represents the particle's motion, the rightmost grey shaded domain is the solid region, the leftmost is the fluid domain, and the dashed line corresponds to the boundary. NOTE (EMV): Do you need the first "collision" subimage in (b)? Maybe I'm being stupid, but I can't see how it explains anything. I suggest keeping the four-step fullway figure and removing the first step in the midway figure to make a three-step figure.

We deliberately emphasise that the location of the bounce-back wall is *approximately*, rather than exactly, midway boundary and solid nodes. This is because its placement is quite arbitrary. [10] Still, we know that placing the bounce-back boundary on the lattice nodes produces a first-order error; while for a wall location on the middle of the lattice links the method can be formally *second-order accurate*. [11] That explains why this second option is generally preferred, even acknowledging

[10] Only in simple stationary flows invariant by translation along the wall, such as Couette and Poiseuille flows, we can determine exactly the bounce-back wall location as long as the wall is aligned with the lattice directions. We will explore these examples later in this section.

[11] Also, more details on the accuracy of the bounce-back scheme will be provided later in this section through numerics and in section 5.3.1 through a theoretical analysis.

that the wall may not be located exactly there. The bounce-back rule is the classical example of a link-wise scheme in LB boundary conditions. NOTE (EMV): Even the fullway one?

The bounce-back rule locates the boundary on the lattice links, *approximately midway* between the boundary and the solid nodes, and *not on* the grid nodes.

The question that arises in the implementation of the bounce-back rule is which strategy to follow: fullway or halfway? As usual there is no definitive answer here. If simplicity is our main criterion, then fullway bounce-back wins. Here, the boundary treatment is independent of the direction of f_i, and also it is faster in execution, *cf.* Chapter **??** NOTE (GS): Timm I don't know why fullway is faster. Anyway, either you or Orest please explain this in Chapter **??**. However, if we prioritise accuracy then halfway bounce-back has the advantage.

At steady-state both schemes provide quasi-similar outcomes.[12] However, they differ significantly in time-dependent problems. This is illustrated in fig. 5.10. The halfway bounce-back scheme takes Δt to return the particle's information back into the bulk solution, whereas the fullway bounce-back scheme requires $2\Delta t$. This delay occurs because particles are kept stored inside the solid region during an extra time step Δt. As a result, the fullway bounce-back rule degrades the *time accuracy* of the LB solution in transient problems. Another advantage of the halfway bounce-back is that it can be implemented without solid nodes. This enables the modelling of solid surfaces as narrow as one lattice width. NOTE (EMV): As narrow as zero lattice widths, surely, if two fluid nodes have a halfway bounce-back link between them? The above two points justify the preference for the halfway implementation in general applications. As this strategy will be the only one considered in the remaining of this book, we will omit the halfway denomination from the bounce-back terminology hereafter.

[12] Possible differences in the steady-state performance of fullway and halfway bounce-back schemes may be identified in the creation of grid-scaled artifacts called staggered invariants [18, 10, 39], which are generated by the halfway bounce-back. Staggered invariants manifest as a velocity oscillation between two constant values over two successive time steps [18]. Their magnitude is usually small, yet the precise value depends on several factors, such as the initial conditions, the mesh size and its parity. For example, a typical channel flow $\boldsymbol{u} = u_x(y)$ simulated with halfway bounce-back is known to conserve a constant normal velocity u_y^0 as staggered invariant, if initialised with $u_y(t = 0) = u_y^0$ and employing an odd number of nodes along y direction [39]. With an even number of lattice nodes this artifact is absent. Due to the nature of the error, the averaging of the halfway bounce-back solution between two successive time steps corrects it [18]. The fullway bounceback, as delaying the exchange of information between successive time steps, does not produce this defect. However, the conservation and stability properties of the scheme may deteriorate in this case [39].

Resting walls

Let us now discuss the case of a *resting wall*. The bounce-back approach may be realised in two different ways, depending on whether virtual nodes are considered or not.[13]

1. If we use virtual nodes, the non-local bounce-back is implemented:

> Populations reaching virtual nodes $\boldsymbol{x}_B + \boldsymbol{c}_i\Delta t$ at time $t + \Delta t$ are *instantaneously* reflected back to the original boundary node $\boldsymbol{x}_B$, from where they came from, but with shifted velocity $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$, see fig. 5.11(a). This means that, after propagation, the following algorithm has to be applied:
>
> $$f_{\bar{i}}(\boldsymbol{x}_B, t + \Delta t) = f_i(\boldsymbol{x}_B + \boldsymbol{c}_i\Delta t, t + \Delta t). \tag{5.22}$$

2. If we choose *not* to use virtual nodes, the local bounce-back is implemented:

> Populations leaving the boundary node $\boldsymbol{x}_B$ at time t meet the wall surface at time $t + \frac{\Delta t}{2}$, being afterwards reflected to the original boundary node $\boldsymbol{x}_B$ where they will arrive at time $t + \Delta t$, but with shifted velocity $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$, see fig. 5.11(b). In this case, standard propagation is replaced by
>
> $$f_{\bar{i}}(\boldsymbol{x}_B, t + \Delta t) = f_i^{\star}(\boldsymbol{x}_B, t). \tag{5.23}$$

Once again, there is no best choice between eq. (5.22) and eq. (5.23). Generally, eq. (5.22) is preferred due to computational efficiency arguments. NOTE (GS): At the moment, I cannot think of a better reason to prefer eq. (5.22) over eq. (5.23) other than computational aspects. Possibly there are many good reasons, but I cannot find one right now. Help, your input is very important here! Among them is the fact that is less memory demanding (it avoids storing both pre-streaming $f_i^{\star}$ and post-streaming f_i populations).TODO (OS): Orest and/or Timm can you complete or correct what I wrote. Thanks! On the other hand, eq. (5.23) has the advantage that is implemented locally. This makes it not only a lot easier to handle, but also the only viable option in some problems. For instance, in porous media flows or solid particle suspensions the distance between walls is often resolved by only few lattice spaces. This makes eq. (5.23) preferable because it requires only one grid spacing between surfaces. Both strategies are illustrated in fig. 5.11 for a bottom wall. Explicitly, they are coded as follows:

[13] The possibility of realising the bounce-back condition in such two ways is similar to that pointed out in section 5.2.1, regarding periodic boundary conditions. Indeed, this kind of flexibility is shared by all LB boundary schemes belonging to the link-wise family.

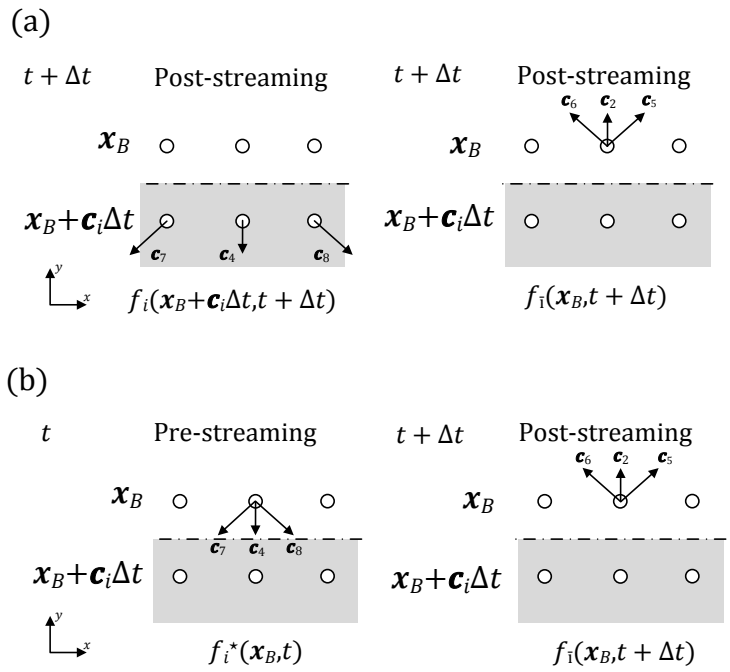


Fig. 5.11: Illustration of the bounce-back rule with the D2Q9 model for a bottom wall. Bounce-back realisation with (a) non-local scheme given by eq. (5.22) and (b) local scheme given by eq. (5.23). The arrows represent the particle's motion, the bottom grey shaded domain is the solid region, the top is the fluid domain, and the dashed line corresponds to the location of the no-slip boundary.

$$\begin{aligned} f_2(\boldsymbol{x}_B, t + \Delta t) &= f_4(\boldsymbol{x}_B + \boldsymbol{c}_4\Delta t, t + \Delta t), \\ f_5(\boldsymbol{x}_B, t + \Delta t) &= f_7(\boldsymbol{x}_B + \boldsymbol{c}_7\Delta t, t + \Delta t), \\ f_6(\boldsymbol{x}_B, t + \Delta t) &= f_8(\boldsymbol{x}_B + \boldsymbol{c}_8\Delta t, t + \Delta t), \end{aligned} \tag{5.24}$$

for non-local bounce-back, and

$$\begin{aligned} f_2(\boldsymbol{x}_B, t + \Delta t) &= f_4^\star(\boldsymbol{x}_B, t), \\ f_5(\boldsymbol{x}_B, t + \Delta t) &= f_7^\star(\boldsymbol{x}_B, t), \\ f_6(\boldsymbol{x}_B, t + \Delta t) &= f_8^\star(\boldsymbol{x}_B, t). \end{aligned} \tag{5.25}$$

for local bounce-back.

Exercise 5.3. Rewrite eq. (5.24) and eq. (5.25) for a top wall.
Answer: The top wall equivalent of eq. (5.24) is $f_4(\boldsymbol{x}_B, t+\Delta t) = f_2(\boldsymbol{x}_B+\boldsymbol{c}_2\Delta t, t+\Delta t)$, $f_7(\boldsymbol{x}_B, t+\Delta t) = f_5(\boldsymbol{x}_B+\boldsymbol{c}_5\Delta t, t+\Delta t)$ and $f_8(\boldsymbol{x}_B, t+\Delta t) = f_6(\boldsymbol{x}_B+\boldsymbol{c}_6\Delta t, t+\Delta t)$ while

the equivalent of eq. (5.25) is $f_4(\boldsymbol{x}_B, t+\Delta t) = f_2^\star(\boldsymbol{x}_B, t)$, $f_7(\boldsymbol{x}_B, t+\Delta t) = f_5^\star(\boldsymbol{x}_B, t)$ and $f_8(\boldsymbol{x}_B, t+\Delta t) = f_6^\star(\boldsymbol{x}_B, t)$.

Moving walls

The extension to *moving walls* is quite simple. It requires only a small correction to the standard bounce-back formula [18, 38]. The role of this correction can be explained resorting again to our particle-based picture of the bounce-back dynamics. Since now the wall is not at rest, after hitting the wall, the bounced-back particles have to gain or lose a given amount of momentum so that momentum is conserved during the collision process (similarly to what happens in the LB scheme in the bulk). The rigorous form of this correction term can be deduced by a Galilean transform operation of the bounced-back populations into a reference frame where the wall is at rest [**?**].[14]

The bounce-back formula applied to describe the Dirichlet boundary condition, for a prescribed boundary velocity $\boldsymbol{u}_B$, reads as follows:

$$f_{\bar{i}}(\boldsymbol{x}_B, t+\Delta t) = f_i(\boldsymbol{x}_B + \boldsymbol{c}_i\Delta t, t+\Delta t) - 2w_i\rho_B\frac{\boldsymbol{c}_i\cdot\boldsymbol{u}_B}{c_s^2}, \tag{5.26}$$

or, equivalently,

$$f_{\bar{i}}(\boldsymbol{x}_B, t+\Delta t) = f_i^\star(\boldsymbol{x}_B, t) - 2w_i\rho_B\frac{\boldsymbol{c}_i\cdot\boldsymbol{u}_B}{c_s^2}. \tag{5.27}$$

For a stationary boundary with $\boldsymbol{u}_B = \boldsymbol{0}$ the correction obviously vanishes and the above equations revert back to eq. (5.22) and eq. (5.23), respectively. It turns out that when $\boldsymbol{u}_B \neq \boldsymbol{0}$ we also need to know the local value of the density on the wall surface, *i.e.* ρ_B. If the incompressible LB model is used, this is not problematical as ρ_B amounts to an immaterial constant, uniform all over the flow domain. However, if the standard LB model is used, ρ varies locally with the pressure, and, in general, these properties are not known at the wall. A solution may be worked out by estimating ρ_B from either the local fluid density at the boundary node, *i.e.* $\rho(\boldsymbol{x}_B)$, or the averaged density value, say $\langle\rho\rangle$. The difference is $O(\mathrm{Ma}^2)$ (where Ma is the Mach number) and therefore usually small.

In flow configurations where no mass flux crosses the boundaries, such as in parallel Couette and Poiseuille flows, the total mass in the system is conserved [10]. However, for arbitrary (planar or non-planar) inclined boundaries, none of the aforementioned procedures are capable of exactly satisfying mass conservation, a feature much appreciated in the LBM. To overcome this issue, several strategies have been

[14] Such a way of determining the correction term is not unique. For instance, the original derivation by Ladd [18] employed a different method.

developed (*e.g.* [**?**, 40, **?**]) which explicitly consider that, besides momentum, also mass is exchanged between fluid and moving solid regions. It appears, however, that these strategies, while correcting for the local mass leakage across solid-fluid boundaries, can potentially lead to collateral effects as decreasing the accuracy of the LB solution [10, 38, 41, **?**]. At the moment, the effective merits of these mass-conserving strategies remain an open question. The topic of mass conservation in solid boundaries is addressed in section 5.3.2.

Advantages and disadvantages

Referring to the main characteristics of the bounce-back method, besides simplicity of implementation, this link-wise type boundary condition displays other *advantages* worth outlining:

1. The bounce-back rule is a *stable* numerical scheme [19, 11], even when the bulk LB solution is brought close to the instability limit, $\tau \to \Delta t/2$. This has to be contrasted with other LB boundary condition methods, particularly wet-node based ones, which are often the main source of numerical instabilities.[15]
2. Another point in favour of the bounce-back method is that, due to its working principle based on reflections, *mass conservation* is strictly ensured on resting boundaries. This is an important feature, particularly useful in problems where the absolute mass value plays an important role. Exact mass conservation is a property often violated by wet-node techniques, where specific corrections need to be introduced, *e.g.*, [24, 25, 26].
3. Bounce-back can be straightforwardly implemented for any number of spatial dimensions.

In terms of *disadvantages*, the bounce-back scheme is mostly criticised for the following reasons:

1. The bounce-back rule lacks accuracy in describing arbitrary boundary configurations as the wall placement cannot be determined with exactness. Even for straight boundaries, if the flow profile is not linear or parabolic, the wall exact placement will not lie necessarily midway between nodes. Indirect ways to infer the wall location can be adopted, nevertheless. For instance, the calibration methods suggested in [42, 38, 12, **?**] help solving this shortcoming.
2. The bounce-back wall location, in the frame of the SRT collision model, is a *viscosity-dependent* feature. It implies that, for the same grid but different τ values, the hydrodynamic solution will differ, even though we fix the governing physical parameters of the problem (*e.g.* Reynolds number). This is an important violation of the problem physics, and is not present in standard discretisation methods of the NSEs. For LBM to be competitive a consistent behaviour is required, and for that the SRT must be replaced by a collision operator offering the

[15] When the LBM is running close to $\tau = 1/2$, the element triggering instability often comes from the boundary rather than the bulk [26].

control of more than one relaxation rate. Viable options are the TRT [43, 39] and the MRT [44, 45, 46] collision models, explained later in chapter 10. When using those collision operators we guarantee, at least, that the bounce-back wall location is *viscosity-independent*, although we may not know with exactness where the boundary lies.

NOTE (EMV): Is the "blocky" resolution of curved walls another minus worth mentioning for bounce-back?

Accuracy of the bounce-back method

To conclude this section we will clarify some aspects of the accuracy of the bounce-back method. For concreteness, our discussion will employ two case studies: a Couette and a Poiseuille flow. They apply to the first- and second-order analysis, respectively. NOTE (EMV): You should quickly define what you mean with these terms. I'm guessing the order of spatial variation of the solution, i.e. linear and quadratic solutions? Based on their numerical solution, we intend to answer the following questions:

1. What is the effect on accuracy if we consider the bounce-back boundary to be located on the lattice nodes, rather than on the lattice links?
2. Why do we say that the location of the bounce-back boundary is approximately midway on a link, rather than exactly there?

First-order analysis:

Let us start with a first-order analysis. For that, we will study the accuracy of the bounce-back scheme in solving a Couette flow.

Without loss of generality, let us assume the fluid density is $\rho = 1$ (simulation unit)[16] and the velocity is $u_0 = 0.1$ (simulation unit) at the top wall and zero at the bottom. The computational domain is resolved by 3×9 grid nodes. NOTE (EMV): Do you really need three nodes in the x direction? Wouldn't one be enough with periodic BCs? The simulation is initialised by setting $f_i(\boldsymbol{x}, t = 0) = f_i^{\text{eq}}(\rho = 1, \boldsymbol{u} = \boldsymbol{0})$, *cf.* section 5.4, and is stopped when it reaches the steady-state criterion $L_2 \leq 10^{-10}$ (for fluid velocity), *cf.* section **??**. We use the SRT collision model with a relaxation time $\tau = 0.9\Delta t$, but other τ values are also tested.

The closure problem for the *boundary populations* is illustrated in fig. 5.12.

Inlet and outlet regions are subject to periodic boundary conditions, *cf.* section 5.2.1. At the inlet boundary they specify f_1, f_5 and f_8 according to eq. (5.13a) or eq. (5.15a); while at the outlet boundary f_3, f_6 and f_7 are prescribed by eq. (5.13b) or eq. (5.15b).

[16] Conversion between simulation and physical units will be discussed in Chapter 7.

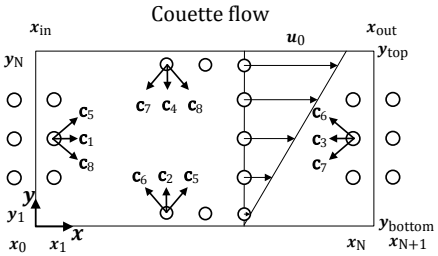


Fig. 5.12: Couette flow solved with the LB D2Q9 model. The unknown populations requiring specification by the boundary condition are explicitly represented in the 4 edges of the computational domain. The layer of wall nodes goes from x_0 to x_{N+1}, while the layer of inflow/outflow conditions goes from y_1 to y_N. Thereby, corner nodes, *e.g.* (x_0, y_0), belong to wall layers.

At walls, we use the bounce-back method to model both the stationary bottom wall and the moving top wall. Choosing the bounce-back formula which implemented locality, *i.e.* without virtual nodes, we have:[17]

- Bottom wall:

$$\begin{aligned} f_2(x, y_1, t + \Delta t) &= f_4^\star(x, y_1, t), \\ f_5(x, y_1, t + \Delta t) &= f_7^\star(x, y_1, t), \\ f_6(x, y_1, t + \Delta t) &= f_8^\star(x, y_1, t). \end{aligned} \tag{5.28}$$

- Top wall:

$$\begin{aligned} f_4(x, y_N, t + \Delta t) &= f_2^\star(x, y_N, t), \\ f_7(x, y_N, t + \Delta t) &= f_5^\star(x, y_N, t) - \frac{1}{6}u_0, \\ f_8(x, y_N, t + \Delta t) &= f_6^\star(x, y_N, t) + \frac{1}{6}u_0. \end{aligned} \tag{5.29}$$

Let us now answer the first question and evaluate the effect of the bounce-back wall location on the accuracy of the LB solution. For that, let us assume two different cases for the physical position y_j of a grid node j ($j = 1, \ldots, N$): NOTE (EMV): I find this talk of physical nodes kinda confusing. Isn't the point whether the no-slip condition occurs at the wall or the link?

a) The physical nodes coincide with the grid nodes, *i.e.* $y_j = j\Delta x$.
b) The physical nodes are centered midway between grid nodes, *i.e.* $y_j = (j - 0.5)\Delta x$.

In both cases we assume the origin of the physical coordinate system to coincide with the bottom wall. As such, the location of bounce-back walls can be interpreted as follows:

[17] Note, the numerical values in the top wall case come from $2w_i/c_s^2 = 1/6$, with $c_s = 1/\sqrt{3}$ and D2Q9 lattice weights in tab. 3.1.

a) The first case makes the bounce-back walls coincide with the boundary nodes $y_0 = y_{\text{bottom}} = 0$ and $y_{N+1} = y_{\text{top}} = (N+1)\Delta x$. This means the width of the channel is $W = y_{\text{top}} - y_{\text{bottom}} = (N+1)\Delta x$, see fig. 5.13a).
b) The second case places the bounce-back walls midway between boundary and solid nodes. The boundary nodes are placed at $y_1 = 0.5\Delta x$ and $y_N = (N-0.5)\Delta x$. The walls are located halfway outside these nodes at $y_{\text{bottom}} = 0$ and $y_{\text{top}} = N\Delta x$, respectively. Consequently, the width of the channel is $W = y_{\text{top}} - y_{\text{bottom}} = N\Delta x$, see fig. 5.13b).

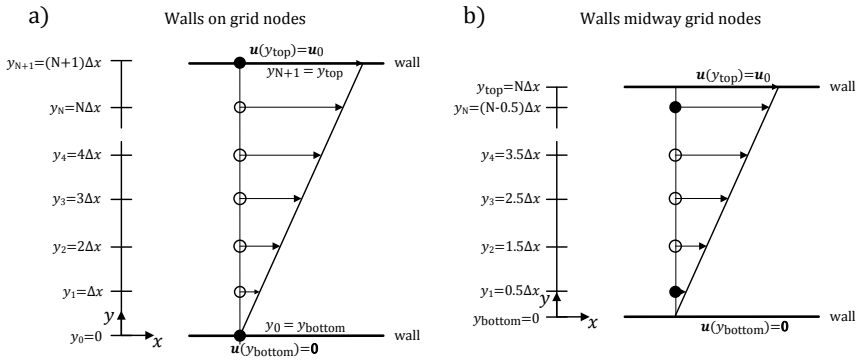


Fig. 5.13: Sketch of the discretisation of the Couette flow problem: a) assuming physical nodes coincide with grid nodes, b) considering physical nodes locate midway between grid nodes. Open circles denote fluid nodes and filled circles boundary nodes. NOTE (EMV): On the left, the black circles represent *virtual* nodes, right? That you call them boundary nodes confused me at first. Shouldn't the ones at Δx and $N\Delta x$ be the boundary nodes, as they are the ones at which the BCs are enforced?

To evaluate the accuracy of numerical solutions, each case is compared with the analytical solution of the Couette flow:

$$u_x(y_j) = \frac{u_0}{y_{\text{top}} - y_{\text{bottom}}}(y_j - y_{\text{bottom}}), \quad y_j \in [y_{\text{bottom}}, y_{\text{top}}], \tag{5.30}$$

where $y_{\text{bottom}} = 0$ and $y_{\text{top}} = (N+1)\Delta x$ for case a) and $y_{\text{bottom}} = 0$ and $y_{\text{top}} = N\Delta x$ for case b).

The LB velocity solutions for both cases are depicted in fig. 5.14. We see that case a) does not reproduce the analytical velocity profile, while case b) leads to machine-accurate results. This implies that when the boundary is located *midway* between nodes [36] the bounce-back method is, at least, first-order accurate (for lattice aligned walls). NOTE (EMV): This "first-order accurate" statement needs to be reevaluated in light of our recent email discussion. You've shown that it resolves a linear flow perfectly, not that it is first-order accurate in Δx. In addition, this numerical test also evidences that Couette flow solutions remain unaffected by the choice of

τ. Owing to the linear nature of the solution, this indicates the bounce-back scheme is viscosity-independent, at least, at first-order.

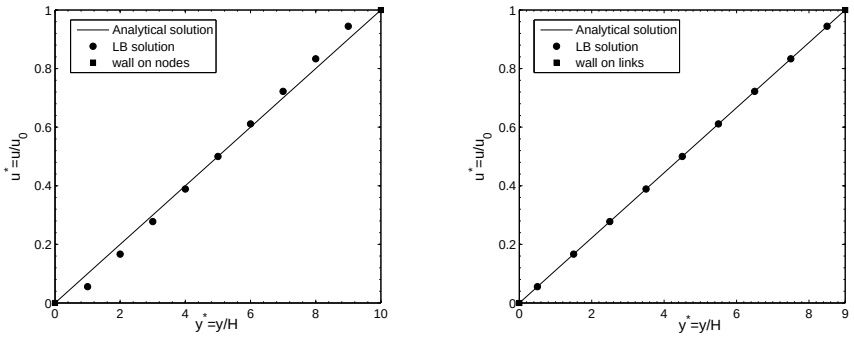


Fig. 5.14: LBM solution of Couette flow with the bounce-back rule. Comparison between analytical solution (continuous line) and the LB solution (circle markers): $N = 9$ and $\tau = 0.9\Delta t$ (different τ values provide identical solutions up to the machine accuracy). The assignment of two possible locations for the bounce-back wall is plotted in each figure for comparison purposes. Left plot: wall is placed at the lattice nodes. Right plot: wall is placed midway at lattice links. Visual inspection is sufficient to conclude that the exact velocity solution is obtained only if locating the bounce-back wall is midway at lattice links.

The numerical solution of the Couette flow reveals that: (1) the bounce-back wall is located *exactly* midway between nodes, and (2) solutions are *viscosity-independent*. As conclusions are based on a linear analysis, they only apply to *first-order* terms. NOTE (EMV): You might want to make more clear that these results hold only for similar flows.

Second-order analysis

After the first-order analysis, the next logical question is whether the bounce-back method preserves those encouraging results at the second order. To answer this question we require a second-order analysis, which can be done by studying a quadratic flow solution, *i.e.*, a Poiseuille flow.

The numerical solution of the Poiseuille flow problem runs with the same simulation parameters of the Couette flow problem, studied in the first-order analysis. The exception happens in boundary conditions. Here, both top and bottom walls are subject to a zero velocity condition, modelled with the local bounce-back rule;

while the inlet and outlet boundaries establish a fully-developed pressure-driven flow condition, operated with the generalised periodic boundary condition method (*cf.* section 5.2.2). The closure problem for the *boundary populations* is illustrated in fig. 5.15. The bounce-back scheme at walls adopts eq. (5.28) and eq. (5.29) with

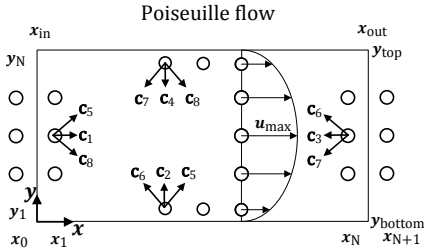


Fig. 5.15: Poiseuille flow within the D2Q9 model. The unknown boundary populations requiring specification are explicitly represented in the 4 edges of the computational domain. The layer of wall nodes goes from x_0 to x_{N+1}, while the layer of inflow/outflow conditions goes from y_1 to y_N. Thereby, corner nodes, *e.g.* (x_0, y_0), belong to wall layers.

$u_0 = 0$. The inlet and outlet boundaries, modelled with the generalised periodic boundary condition method, use the same formulas elaborated in Example 5.2. Recall, we arbitrarily set $p_{out} = 1$ (in simulation units) and compute the inlet pressure as $p_{in} = p_{out} + \Delta p$.[18] In order to keep with the small Ma requisite, we set $u_{max} = 0.1$ (simulation unit) and relate it with the pressure difference Δp as follows:

$$\Delta p = p_{in} - p_{out} = \frac{8 u_{max} \nu}{\rho} \frac{(x_{out} - x_{in})}{(y_{top} - y_{bottom})^2}.$$

In the domain discretisation let us, once again, assume the location of the grid nodes, with respect to the physical coordinate system, can be either a) coincident, $y_j = j\Delta x$, or b) displaced by one half lattice spacing, $y_j = (j - \frac{1}{2})\Delta x$ $(j = 1, \ldots, N)$. These two choices are shown in fig. 5.16.

In both cases, the numerical solutions are compared against the Poiseuille analytical solution:

$$u_x(y_j) = -\frac{1}{2\rho\nu} \frac{\Delta p}{(x_{out} - x_{in})} (y_j - y_{bottom})(y_j - y_{top}), \quad y_j \in [y_{bottom}, y_{top}], \quad (5.31)$$

[18] Since the Poiseuille flow is generated by a pressure drop Δp, it is recommendable to use the incompressible LB model (*cf.* section 4.3.2). Recall the standard LB equilibrium operates with an isothermal equation of state where pressure and density relate linearly. Therefore, a pressure gradient inevitably leads to a gradient of density, a feature not supported by incompressible hydrodynamics. With the incompressible LB model this compressibility error can be avoided (at least, in steady flows).

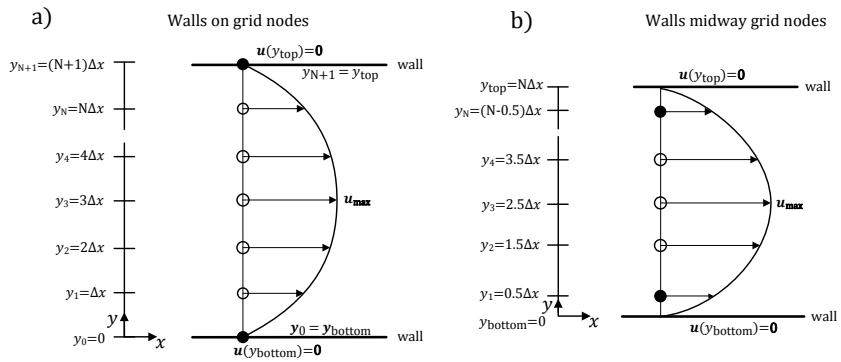


Fig. 5.16: Sketch of the Poiseuille flow problem considering the physical nodes a) coincide with the grid nodes and b) locate at the lattice links midway between grid nodes. Open circles denote fluid nodes and filled circles boundary nodes. NOTE (EMV): My comments on Fig. 5.13 apply to this one too.

where $x_{\text{in}} = (x_0 + x_1)/2$ and $x_{\text{out}} = (x_N + x_{N+1})/2$. The boundary locations are $y_{\text{bottom}} = 0$ and $y_{\text{top}} = (N+1)\Delta x$ in case a) and $y_{\text{bottom}} = 0$ and $y_{\text{top}} = N\Delta x$ in case b).

The numerical and analytical solutions are plotted in fig. 5.17. Placing the wall at grid nodes – Case a) – leads to strongly deviating results, much less accurate than for the linear Couette flow. Placing the wall midway grid nodes – Case b) – yields a nearly perfect velocity profile.

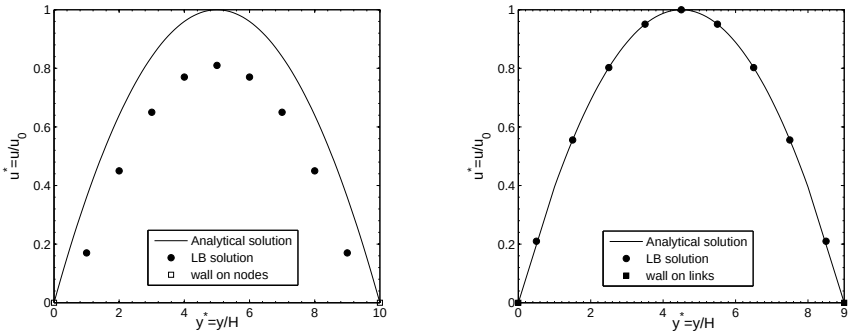


Fig. 5.17: LBM solution of Poiseuille flow with bounce-back rule. Comparison between analytical solution (continuous line) and LB solution (circles) for $N = 9$ and $\tau = 0.9\Delta t$. Left panel: wall is placed at the lattice nodes; right panel: wall is placed midway at lattice links. A second-order accurate velocity solution is obtained only in the second case.

However, contrary to the linear analysis, quadratic solutions are affected by the τ choice, see fig. 5.18. In fact, while velocity profiles remain perfect parabolas in bulk, their boundary values display a clear τ-dependent velocity slip when the (hypothetical) wall is placed midway nodes. The *exact* zero velocity condition is established if and only if $\tau = \left(\sqrt{3/16} + 1/2\right)\Delta t \approx 0.93\Delta t$. [19] The bounce-back velocity boundary condition placed at the midway location remains reasonably accurate for smaller τ values, *e.g.* velocity error of $L_2 \simeq 1.6\%$ for $\tau = 0.6\Delta t$. Yet, for larger relaxation times, such at $\tau = 3\Delta t$ or $5\Delta t$, the errors may increase up to $L_2 \simeq 100\%$ or more. The reason is that, using the SRT collision operator, the bounce-back wall becomes displaced from the desired midway position by a distance proportional to $(\tau/\Delta t - 1/2)^2$ [16, 17, 11, 47, **?**]. This helps answering the second question of this exercise: why cannot we guarantee that, in general, the boundary locates exactly midway between grid nodes. The answer is summarised in fig. 5.19.

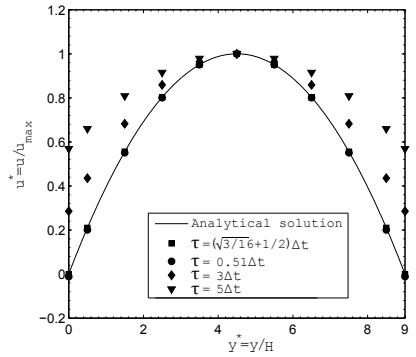


Fig. 5.18: LBM solution of the Poiseuille flow with the bounce-back rule. Comparison between analytical solution (continuous line) and the LB solutions for different τ values (computed with $N = 9$ grid nodes in vertical direction), assuming an midway wall location.

For fixed τ, the bounce-back defect in the wall location reduces with the number of grid nodes like Δx^2. This second-order spatial accuracy is shown in fig. 5.20a). The convergence of the bounce-back rule with the τ value, for a fixed grid resolution N, is quantified in fig. 5.20b), indicating the existence of a specific value of τ where the wall is located exactly midway, thus leading to an *exact* solution. Given that the error scales as $(\tau/\Delta t - 1/2)^2$, for large τ values, it grows as τ^2. The theoretical explanation of this viscosity-dependent behaviour is given in section 5.3.1.

[19] The theoretical explanation on why the exact midway location of the no-slip wall condition happens for $\tau = \left(\sqrt{3/16} + 1/2\right)\Delta t$ and not other values is given in section 5.3.1.

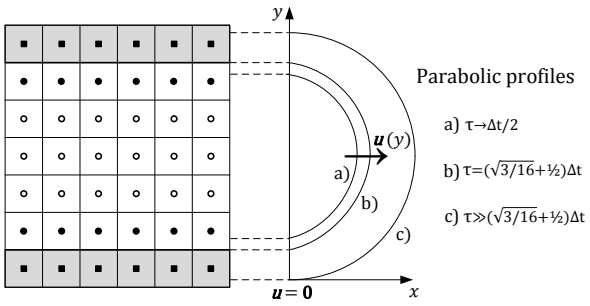
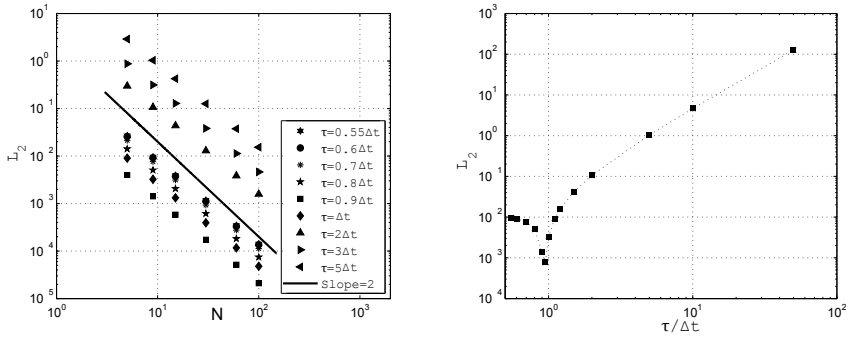


Fig. 5.19: Sketch of τ-dependency of the bounce-back wall placement in the Poiseuille flow simulated with SRT collision model. Open circles: fluid nodes; Filled circles: boundary nodes; Filled squares: solid nodes. Parabolas on the center illustrate how velocity profiles enlarge/contract with large/small τ values — image inspired by [**?**]. NOTE (EMV): From the previous figur it looks a lot like $\tau \to \Delta t/2$ gives a more decent results than this figure indicates. Also, I guess c) should be $\tau \gg (\ldots)$?



(a) Effect of N on L_2, for several τ values.

(b) Effect of τ on L_2, for $N = 9$ fixed.

Fig. 5.20: L_2 error for the LBM with SRT operator and the bounce-back model (with midway walls) applied to Poiseuille flow. Panel a) shows that numerical solutions are second-order accurate with respect to the spatial resolution, regardless the τ value. Panel b) indicates that the accuracy is optimum at $\tau = \left(\sqrt{3/16} + 1/2\right)\Delta t$. Although not explicit in the figure, for this τ value L_2 is of the order of the round-off error. At large τ values, the L_2 error grows with τ^2. NOTE (EMV): Your y axis exponents lack minus signs, so that e.g. 10^{-2} becomes 10 ^{2}.

As we have seen for Couette flow, the midway placement of the bounce-back boundary condition is exact and viscosity-independent at the first order. NOTE

(EMV): Again, I think you should call this a linear flow.

NOTE (EMV): Again, I am confused. You've shown second-order accuracy (*i.e.* convergence) but you've only showed perfect-ish accuracy for quadratic flows for a single value of τ.

5.2.4 Solid boundaries – the wet-node approach

Owing to the defects of the bounce-back method, namely its lack of explicitness in establishing the wall location up to the second order, further aggravated by its viscosity-dependent behaviour if using the SRT collision operator, it motivated the development of a different perspective to establish straight walls in LB boundary conditions. This alternative viewpoint corresponds to the *wet-node* category. Here, the boundary location coincides with the boundary node, which is assumed to lie infinitesimally close to the actual boundary, yet still inside the fluid domain. As a consequence, the standard LB steps in the bulk (*i.e.* collision and streaming) apply in the same way to the boundary nodes. The above aspects form the major differences between the wet-node and the link-wise approaches.

The purpose of the wet-node approach is to assign suitable values for the unknown LB boundary populations so that, collectively, the known and unknown mesoscopic populations f_i reproduce the intended hydrodynamic physics (and associated symmetries) at the boundary. The main challenge of this task is the discrepancy between the number of boundary populations requiring specification and the number of macroscopic conditions which need to be satisfied. This explains the many techniques developed to deal with this under-specified problem, *e.g.* [20, 21, 22, 23, 6, 24, 25, 14, 26]. Since covering all of them would be impossible in the context of this book, here our focus will be put on three of the most popular approaches available in the literature. They are the *equilibrium scheme* [17, 48], the *non-equilibrium extrapolation method* [6], and the *non-equilibrium bounce-back method* [23]. Although these three approaches do not represent the totality of wet-node techniques, they provide an accurate idea of the philosophy underlying them.

Finding the density on boundaries

Before proceeding to the analysis of the wet-node approach, it is convenient to first introduce a general procedure to find the density (pressure) on a straight boundary subject to the Dirichlet velocity condition [23, 14]. Such a procedure makes use of the flow continuity condition as a constraint to find the fluid density ρ from known fluid velocity $\boldsymbol{u}$ or mass flux $\rho\boldsymbol{u}$. [20]

Since the entire flow solution, boundaries included, must satisfy the continuity condition, it follows that, while prescribing the wall velocity, one is also implicitly enforcing the wall density (pressure) to a unique value. NOTE (EMV): That sentence was a bit too convoluted. However, the LBM does not work directly with the macroscopic fields. Consequently, the continuity condition does not arise naturally; rather it has to be enforced explicitly.

Fortunately, this is a rather simple task. The continuity of the fluid at the boundary is described by the well-known impermeable wall condition, *i.e.* zero relative normal velocity of the fluid. Hence, we just have to translate this condition to the LB populations. For illustration purposes, let us see how to apply this procedure for a top wall (*cf.* fig. 5.21). The extension to other wall orientations is straightforward, see *e.g.* [23, 14].

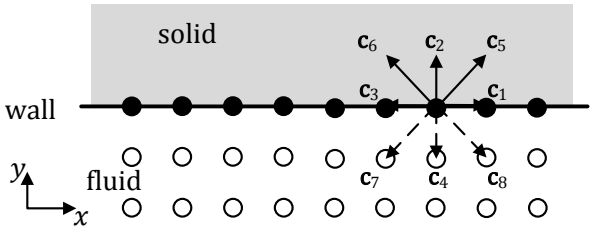


Fig. 5.21: Top wall coinciding with the horizontal lattice link. The shaded region denotes the solid and the region below the fluid domains. The line in between represents the wall. The known boundary populations are represented by continuous vectors, the unknown by dashed vectors.

The impermeable wall condition is determined from the density and vertical velocity component, which in terms of LB populations is expressed as follows:

[20] Although the LBM may reproduce the continuity condition in two different forms, depending on the equilibrium model adopted, *i.e.* $\partial_t\rho+\nabla\cdot(\rho\boldsymbol{u})=0$ for the standard compressible equilibrium or $c_s^{-2}\partial_t p+\nabla\cdot\boldsymbol{u}=0$ for the incompressible equilibrium, the procedure described here remains applicable in both cases.

$$\begin{aligned}\rho &= \sum_i f_i = \underbrace{f_0 + f_1 + f_2 + f_3 + f_5 + f_6}_{\text{known}} + \underbrace{f_4 + f_7 + f_8}_{\text{unknown}},\\ \rho u_y &= \sum_i c_{iy} f_i = \underbrace{f_2 + f_5 + f_6}_{\text{known}} - \underbrace{(f_4 + f_7 + f_8)}_{\text{unknown}},\end{aligned} \tag{5.32}$$

where we have grouped the known and unknown populations (*cf.* fig. 5.21). Combining the two equations above, we can determine ρ independently of the unknowns populations:

$$\rho = \frac{1}{1 + u_y}\left(f_0 + f_1 + f_3 + 2\left(f_2 + f_5 + f_6\right)\right). \tag{5.33}$$

Exercise 5.4. Repeat the above procedure to find ρ for a bottom wall.
Answer: $\rho = \frac{1}{1-u_y}\left(f_0 + f_1 + f_3 + 2\left(f_4 + f_7 + f_8\right)\right)$.

The major limitation of this method is that it applies only to straight walls, aligned with the lattice nodes. Still, the method is readily applicable to planar boundaries in either 2D or 3D problems.

Equilibrium scheme

The equilibrium scheme is possibly the simplest way to specifying boundary conditions in the LBM. It assumes that populations at the boundary are in a state of equilibrium [17, 14, 48]. Thereby, it assigns to the boundary populations the *equilibrium distribution value*, a quantity which is readily available from the known macroscopic properties at the boundary. The equilibrium scheme applies to the post-streamed boundary populations and reads as follows:

$$f_i(\boldsymbol{x}_B, t) = f_i^{\mathrm{eq}}\left(\rho_B, \boldsymbol{u}_B\right), \tag{5.34}$$

where the subscript B refers to properties evaluated at the boundary. We underline eq. (5.34) applies to *all* populations, rather than only to the unknown ones.

Despite its attractiveness, the equilibrium scheme is plagued by one critical deficiency. The basic principle of wet boundary nodes is that they have to be treated on an equal footing as bulk nodes, *i.e.* they have to satisfy the same streaming and collision dynamics. Wet-node techniques apply over post-streaming populations f_i, whose post-collision state is $f_i^\star = f_i^{\mathrm{eq}} + (1 - \Delta t/\tau) f_i^{\mathrm{neq}}$. But by definition, the boundary nodes are lacking any non-equilibrium contribution: $f_i = f_i^{\mathrm{eq}}$. Hence, compatibility between boundary and bulk dynamics only can be met when $\tau = \Delta t$ or $f_i^{\mathrm{neq}} = 0$. The latter condition is, however, of little interest since it leads to a trivial solution.[21] By setting $\tau = \Delta t$ the equilibrium scheme becomes consistent with the bulk dynamics and therefore a second-order accurate method.

[21] Recall that f_i^{neq} is related to the spatial derivatives of the flow field. Consequently, $f_i^{\mathrm{neq}} = 0$ corresponds to a spatially uniform solution.

The applicability of the equilibrium scheme has been subject to several studies [17, 23, 14, 48]. One of the most detailed works was provided by Mohamad and Succi [48]. Based on an order of magnitude analysis, the authors advocated that the equilibrium scheme is a suitable boundary condition method, independently of the value of τ, provided the numerical simulations are well-resolved, *i.e.* the lattice spacing Δx is sufficiently fine to resolve the smallest hydrodynamic scale $\min(|\boldsymbol{u}/\nabla\boldsymbol{u}|)$. Although the conclusion in [48] is encouraging, we suggest the adoption of a more careful judgement. The reason is that the accuracy of this boundary scheme remains, in general, one order lower than the bulk. As shown before, this feature can lead to rather inaccurate solutions. The example below aims at demonstrating the impact of this negative feature.

For the cases where the equilibrium scheme is considered to provide a satisfactory accuracy, it is worth pointing out some of its advantages. One of them is possessing advanced stability properties. NOTE (EMV): Is "advanced" the right word to use? Perhaps "excellent"? This is a remarkable feature considering the less stable behaviour usually exhibited by wet-node schemes. Another welcome advantage is its applicability to 2D and 3D problems.

Example 5.1. Using the equilibrium scheme, we repeat the studies of the Couette and pressure-driven Poiseuille flows studied in section 5.2.3. The LB description of the two flow problems is depicted in fig. 5.22.

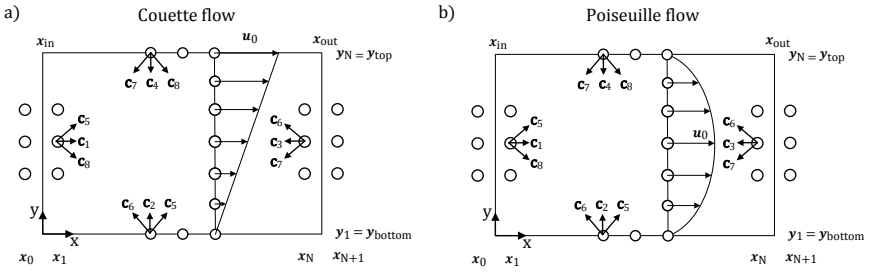


Fig. 5.22: Couette and Poiseuille flows with unknown boundary populations in the wet-node framework.

The solution of this problem makes use of the same simulation parameters employed before in section 5.2.3. The main difference is the implementation of the wall boundary conditions. Note that, for convenience, we use the incompressible linear (Stokes) equilibrium, instead of considering the standard compressible and non-linear LB equilibrium. The advantage of the incompressible LB equilibrium in modelling incompressible hydrodynamics has been presented in section 4.3.2 and further discussed in section 5.2.2 and section 5.2.3 – Accuracy of the bounce-back method. NOTE (EMV): Do you to mention the section title? The choice for a linear (Stokes) equilibrium, instead of non-linear (Navier-Stokes) one, is justified for

simplicity, as conclusions are unaffected. NOTE (EMV): Unclearly formulated sentence.

The equilibrium populations within the incompressible linear regime read

$$f_i^{\mathrm{eq}}(\boldsymbol{x}_B, t) = w_i \left(\frac{p_B}{c_\mathrm{s}^2} + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}_B}{c_\mathrm{s}^2} \right), \tag{5.35}$$

where B denotes properties evaluated at the boundary. We employ the following boundary conditions:

- Couette flow: $u_x(y_{\mathrm{bottom}}) = 0$ and $u_x(y_{\mathrm{top}}) = u_0$
- Poiseuille flow: $u_x(y_{\mathrm{bottom}}) = u_x(y_{\mathrm{top}}) = 0$

The transverse velocity u_y is always zero in both cases. The boundary pressure, included in the boundary equilibrium, eq. (5.35), is found through an identical procedure for both flows. However, since the velocity moments using the incompressible model are sightly different from the standard equilibrium case, it is worth re-writing them here, in order to show how the wall pressure is computed with the LB incompressible model. For example, for a boundary node located at the top wall, *cf.* fig. 5.21, we have

$$\begin{aligned} \frac{p}{c_\mathrm{s}^2} &= \sum_i f_i = \underbrace{f_0 + f_1 + f_2 + f_3 + f_5 + f_6}_{\text{known}} + \underbrace{f_4 + f_7 + f_8}_{\text{unknown}}, \\ u_y &= \sum_i c_{iy} f_i = \underbrace{f_2 + f_5 + f_6}_{\text{known}} - \underbrace{(f_4 + f_7 + f_8)}_{\text{unknown}}. \end{aligned}$$

Combining them, we obtain

$$\frac{p}{c_\mathrm{s}^2} = -u_y + \Big(f_0 + f_1 + f_3 + 2\,(f_2 + f_5 + f_6) \Big).$$

Repeating the idea for the bottom wall it follows that

$$\frac{p}{c_\mathrm{s}^2} = u_y + \Big(f_0 + f_1 + f_3 + 2\,(f_4 + f_7 + f_8) \Big).$$

We now have all data required to implement the equilibrium scheme. Despite its simplicity, the lack of accuracy may compromise its utility. In fact, we can confirm that neither Couette nor Poiseuille flow solutions are accurately reproduced when $\tau \neq \Delta t$ (*cf.* fig. 5.23). Obviously, the use of other equilibrium distributions (*e.g.*, the full standard equilibrium case) will not lead to improvements as the problem identified here comes from neglecting the non-equilibrium part of the boundary populations.

As fig. 5.23 a) illustrates, the equilibrium scheme is unable to catch the simple linear Couette profile. Due to the lack of non-equilibrium terms, the boundary scheme is unaware of the presence of velocity gradients near the wall. Obviously, this situation becomes worse for even steeper velocity gradients. Therefore, it is no surprise that in the Poiseuille flow even larger discrepancies are observed, as shown

in fig. 5.23 c). However, fig. 5.23 b) and d) demonstrate that the equilibrium scheme is indeed second-order accurate for $\tau = \Delta t$. Here, both flows are reproduced up to machine precision.

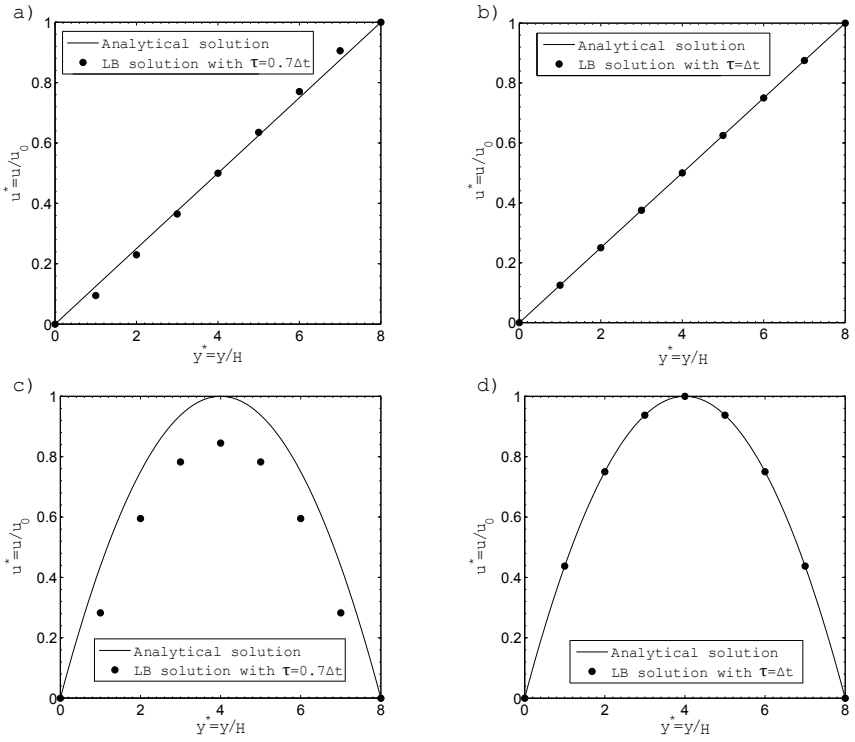


Fig. 5.23: LB solution of Couette and Poiseuille flows with the equilibrium scheme. Comparison between analytical solution (continuous line) and the LB solution (circles) with $N = 9$. Plots a) and c) for $\tau = 0.7\Delta t \neq \Delta t$ show inaccurate solutions. Plots b) and d) for $\tau = \Delta t$ show machine-accurate solutions. NOTE (EMV): Could you merge the two Couette flow images and the two Poiseuille flow images so that you show both the $\tau = 0.7$ and $\tau = 1$ solutions in each using different symbols? That would make for a better comparison and a more careful use of space.

Non-equilibrium extrapolation method

The previous example showed that the equilibrium scheme fails to be an accurate boundary method owing to the negligence of the non-equilibrium part of the boundary populations. As such, the next logical step to improve it is the inclusion of the non-equilibrium part. The difficulty arises in finding the non-equilibrium term for

the boundary populations. As there is no unique answer for this task, several procedures/approximations have been put forward, resulting in a variety of wet-node techniques.

Perhaps the simplest way to determine the non-equilibrium part of the boundary populations is to extrapolate its value from the fluid region, where f_i^{neq} is known. Such a rationale is known as the *non-equilibrium extrapolation method*, and was originally proposed by Guo *et al.* [6]. Its formulation applies to the post-streamed boundary populations as follows:

$$f_i(\boldsymbol{x}_B, t) = f_i^{\mathrm{eq}}(\rho_B, \boldsymbol{u}_B) + \left(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})\right), \tag{5.36}$$

where the non-equilibrium contribution comes from the fluid node $\boldsymbol{x} = \boldsymbol{x}_B \pm \Delta x$, with the sign depending on whether it is considered a bottom or top boundary, respectively. NOTE (EMV): This notation breaks down for y boundaries, doesn't it? I'm also not too fond of using $\boldsymbol{x}$ as anything but the node under consideration. Maybe it's best to just introduce a node $\boldsymbol{x}_N$, saying that it's the neighbouring fluid node of the boundary node? I guess it should be ρ_N and $\boldsymbol{u}_N$ in that case, too. Similar to the equilibrium scheme, the non-equilibrium extrapolation method applies to *all* boundary populations, rather than only to the unknown ones.

Originally [6, 8] suggested that eq. (5.36) is consistent with the second-order accuracy of the LBM. However, when analysed as a finite-difference approximation of the aimed hydrodynamic solutions, the non-equilibrium extrapolation method appears only first-order accurate. This result can be rigourously explained by using the Chapman-Enskog analysis to unfold the content of f_i^{neq}. According to the Chapman-Enskog analysis, we can assume that f_i^{neq} is essentially governed by its first-order perturbation term, $f_i^{(1)}$ – see section 5.3.1. Therefore we can write

$$f_i^{\mathrm{neq}}(\boldsymbol{x}, t) \simeq -w_i \frac{\tau \Delta t \rho}{c_{\mathrm{s}}^2} Q_{i,\alpha\beta} \partial_\alpha u_\beta, \tag{5.37}$$

where $Q_{i,\alpha\beta} = c_{i\alpha} c_{i\beta} - c_{\mathrm{s}}^2 \delta_{\alpha\beta}$. NOTE (HK): need a reference here to chapter 3. NOTE (EMV): Actually, Chapter 3 uses a cleaner (in my opinion) CEA which avoids explicitly deriving $f_i^{(1)}$, going instead directly for the moments. Adding a derivation of $f_i^{(1)}$ would break the flow without really being worth the space and effort in my opinion. For this equation, you're better off referring to external literature.

Based on eq. (5.37), the zeroth-order extrapolation of f_i^{neq} is equivalent to prescribing a constant $\nabla \boldsymbol{u}$. It is however known that a second-order evaluation of $\boldsymbol{u}$ requires, at least, a first-order approximation of $\nabla \boldsymbol{u}$ (or, in the LB framework, of f_i^{neq}). For that reason, although offering an improvement over the equilibrium scheme, the non-equilibrium extrapolation method, in the form of eq. (5.36), still does not attain the desired second-order accuracy. [22] We can clearly see this again by resorting to the simple Couette and Poiseuille flow examples.

[22] In light of this discussion a possible formally second-order accurate statement of the NEEM would be $f_i(\boldsymbol{x}_B, t) = f_i^{\mathrm{eq}}(\boldsymbol{x}_B, t) + 2 f_i^{\mathrm{neq}}(\boldsymbol{x}_B + 1, t) - f_i^{\mathrm{neq}}(\boldsymbol{x}_B + 2, t)$ [] [Heubes]

Exercise 5.5. Repeat the Couette and pressure-driven Poiseuille flow examples, carried out before for the equilibrium scheme, now using the non-equilibrium extrapolation method (*cf.* fig. 5.22).
Answer: We decompose the boundary populations into f_i^{eq} and f_i^{neq}. The f_i^{eq} part is computed as in the equilibrium scheme. The f_i^{neq} is simply copied from the nearest fluid node (found along the wall normal direction). For example, for a bottom wall $j = 1$, the result is $f_i(x, y_1, t) = f_i^{\mathrm{eq}}(p_b, \boldsymbol{u}_B)|_{(x,y_1,t)} + (f_i - f_i^{\mathrm{eq}})|_{(x,y_2,t)}$.

The non-equilibrium extrapolation method exactly solves the Couette flow, regardless of the τ value. This result is expected since the zeroth-order extrapolation of f_i^{neq} is an accurate procedure to capture the constant velocity gradient solution of the Couette flow. However, when applied to the Poiseuille flow, the accuracy is lost. By setting $f_i^{\mathrm{neq}}(x, y_1, t) = f_i^{\mathrm{neq}}(x, y_2, t)$ we are implicitly enforcing $\nabla \boldsymbol{u}|_{(x,y_1,t)} = \nabla \boldsymbol{u}|_{(x,y_2,t)}$, a condition that cannot be possibly satisfied in a parabolic profile. The second-order accuracy can be achieved in two ways. Either by fixing the relaxation time $\tau = \Delta t$ (where the non-equilibrium contribution is placed out of the LB dynamics); or by performing the extrapolation process in two layers of fluid nodes, *e.g.* $f_i(x, y_1, t) = f_i^{\mathrm{eq}}(x, y_1, t) + 2 f_i^{\mathrm{neq}}(x, y_2, t) - f_i^{\mathrm{neq}}(x, y_3, t)$ [] [Heubes].

Non-equilibrium bounce-back method

From the previous two examples we have learned the importance of correctly describing the non-equilibrium populations on wet boundary nodes. Moreover, the Chapman-Enskog analysis in section 4.1 has guaranteed us that is valid to assume $f_i^{\mathrm{neq}} \sim \nabla \boldsymbol{u}$. NOTE (EMV): Not quite; see my previous note. Also, I changed the $\simeq$ to a $\sim$ if that's OK; I believe the latter is usually used to indicate scaling. Hence, the main task of wet-node algorithms is how to make the non-equilibrium populations to correctly express this velocity gradient on the boundary, *e.g.* [20, 23, 24, 25, 14, 26]. Perhaps, the simplest strategy to constructing f_i^{neq} in a consistent manner is by directly evaluating $\nabla \boldsymbol{u}$ with finite differences [20, 14]. The downside of this approach is that it requires information from neighbouring nodes, therefore breaking one of the main advantages of the LB algorithm: its locality. An alternative to find f_i^{neq} in a local way was proposed by Zou and He [23]. This strategy has become quite popular and is often referred to either by the name of the authors (*Zou-He method*) or by its underlying working principle (*non-equilibrium bounce-back method*, NEBB). Here we will adopt the latter designation.

The idea behind the NEBB method is supported by a sound physical basis. In the (isothermal) NSEs the only relevant contribution coming from non-equilibrium is the stress tensor, as shown in Eq. (4.13). NOTE (EMV): I reformulated to say the stress tensor instead of the strain-rate tensor. As this is a symmetric tensor, *i.e.* unchanged under mirroring the coordinate system, we conclude that mirroring f_i^{neq} has to leave the stress tensor unchanged. NOTE (EMV): I reformulated slightly. Also, it would have been nice to support the statement of mirroring better; isn't a symmetric tensor only defined by being unchanged when swapping the indices? The simplest way to guarantee this is by enforcing

$$f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_B, t) = f_i^{\mathrm{neq}}(\boldsymbol{x}_B, t), \tag{5.38}$$

where $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$.

However, the NEBB formulation cannot be considered closed yet. Remember that for the assignment of the missing macroscopic property p_B (or ρ_B), which is necessary to close the equilibrium boundary populations $f_i^{\mathrm{eq}}(p_B, \boldsymbol{u}_B)$, we have already used two velocity moments: one for the pressure (or density) and another for the normal velocity at the boundary. There is still one velocity moment left, the tangential velocity at the boundary, which must correspond to the intended macroscopic value.[23] It turns out that the intended tangential velocity will not have the correct value if eq. (5.38) is used. This incorrect result comes from the problem being over-constrained.

A solution is the introduction of an extra parameter in eq. (5.38), subject to the constraint that it modifies only the tangential component of boundary populations. This parameter is called the *transverse momentum correction* (*e.g.* [49]) and is represented by N_α where $\alpha = x$ or y depends on the boundary tangential direction t_α. By introducing N_α in eq. (5.38), the consistent formulation of the NEBB rule reads

$$f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_B, t) = f_i^{\mathrm{neq}}(\boldsymbol{x}_B, t) - t_\alpha N_\alpha. \tag{5.39}$$

Example 5.2 illustrates how to determine N_α from the first-order velocity moment of tangential velocity. NOTE (EMV): So, is the "boundary tangential direction" $\boldsymbol{t}$ really a tangent vector, analogous to the normal vector? Isn't the tangent vector non-unique? I mean, even in 2D, $\boldsymbol{t}$ and $-\boldsymbol{t}$ are both tangent vectors.

Compared to the other two wet-node strategies previously described, where the boundary scheme replaces all populations at the boundary, the NEBB method modifies *only the missing populations*. This brings along both positive and negative consequences [14]. On the positive side, the NEBB does not touch the known populations, *i.e.* it makes full use of all information already available. This leads in general to improved accuracy. However, the preservation of the known populations carries two main disadvantages. First, the generalisation of the scheme to different lattices becomes difficult, particularly for those featuring a large number of velocities. In fact, the NEBB is cumbersome to employ in 3D domains [23, 49, 50] (see section **??**). Second, the boundary scheme itself may act as a source of instability. It has been shown in the literature [51, 52, 26] that the NEBB scheme introduces undesirable high wave number perturbations in the bulk solution, ruling out its applicability to high Reynolds number flows [14, 26].

Let us look at the NEBB algorithm in more detail with a practical example.

Example 5.2. Let us use the NEBB method to prescribe the Dirichlet velocity condition for a top wall, depicted in fig. 5.21. For generality, assume a tangentially moving wall with fluid injection, *i.e.* $\boldsymbol{u} = (u_x, u_y)^\top$. We solve the problem as follows:

[23] Note that the analysis presented here is restricted to 2D problems. The 3D case is discussed in section **??**.

1. The first step is to identify the missing populations. For a top wall they are f_4, f_7 and f_8.
2. We now have to compute the wall density in such a way that the unknown populations do not interfere with the computation process. For a top wall, the unknown ρ at the wall is given by eq. (5.33). As the wall velocity is also known, the equilibrium part f_i^{eq} of wet-node populations is determined.
3. The next step is to express the unknown populations in terms of the known parameters. This is done with the help of the non-equilibrium bounce-back formula:

$$\left.\begin{aligned} f_4^{\text{neq}} &= f_2^{\text{neq}}, \\ f_7^{\text{neq}} &= f_5^{\text{neq}} - N_x, \\ f_8^{\text{neq}} &= f_6^{\text{neq}} + N_x. \end{aligned}\right\} \Longrightarrow \left\{\begin{aligned} f_4 &= f_2 + \left(f_4^{\text{eq}} - f_2^{\text{eq}}\right), \\ f_7 &= f_5 + \left(f_7^{\text{eq}} - f_5^{\text{eq}}\right) - N_x, \\ f_8 &= f_6 + \left(f_8^{\text{eq}} - f_6^{\text{eq}}\right) + N_x. \end{aligned}\right. \tag{5.40}$$

NOTE (EMV): I added some brackets to make things tidier. Using the known equilibrium distributions, we get

$$\begin{aligned} f_4 &= f_2 - \tfrac{2}{3}\rho u_y, \\ f_7 &= f_5 - \tfrac{1}{6}\rho\left(u_x + u_y\right) - N_x, \\ f_8 &= f_6 - \tfrac{1}{6}\rho\left(-u_x + u_y\right) + N_x. \end{aligned} \tag{5.41}$$

4. Now we compute N_x by resorting to the first-order velocity moment along the boundary tangential direction:

$$\begin{aligned} \rho u_x &= \sum_i c_{ix} f_i \\ &= (f_1 + f_5 + f_8) - (f_3 + f_6 + f_7) \\ &= (f_1 - f_3) - (f_7 - f_5) + (f_8 - f_6) \\ &= (f_1 - f_3) - \tfrac{1}{3}\rho u_x + 2N_x, \end{aligned} \tag{5.42}$$

which gives

$$N_x = -\tfrac{1}{2}\left(f_1 - f_3\right) + \tfrac{1}{3}\rho u_x. \tag{5.43}$$

5. Finally, we get closed-form solutions for all unknown populations:

$$\begin{aligned} f_4 &= f_2 - \tfrac{2}{3}\rho u_y, \\ f_7 &= f_5 + \tfrac{1}{2}\left(f_1 - f_3\right) - \tfrac{1}{2}\rho u_x - \tfrac{1}{6}\rho u_y, \\ f_8 &= f_6 - \tfrac{1}{2}\left(f_1 - f_3\right) + \tfrac{1}{2}\rho u_x - \tfrac{1}{6}\rho u_y. \end{aligned} \tag{5.44}$$

NOTE (EMV): Is this exactly the Zou-He scheme? If so, I think you should say so explicitly.

Exercise 5.6. Repeat the above procedure to prescribe a Dirichlet velocity condition with the NEBB method, but now for a left wall.
Answer:

$$\rho = \frac{1}{1-u_x}\left(f_0 + f_2 + f_4 + 2\left(f_3 + f_6 + f_7\right)\right),$$
$$f_1 = f_3 + \tfrac{2}{3}\rho u_x,$$
$$f_5 = f_7 - \tfrac{1}{2}\left(f_2 - f_4\right) + \tfrac{1}{2}\rho u_y + \tfrac{1}{6}\rho u_x,$$
$$f_6 = f_8 + \tfrac{1}{2}\left(f_2 - f_4\right) - \tfrac{1}{2}\rho u_y + \tfrac{1}{6}\rho u_x.$$

By applying the NEBB method to the Couette and the Poiseuille flows, illustrated in fig. 5.22, the exact solutions are obtained in both cases, independently of τ. This result confirms that, for straight walls aligned with the lattice links, the *NEBB method is a second-order accurate boundary scheme*. This accuracy is consistent with that offered by the LB equation in the bulk. Hence, the NEBB is more accurate than the other two wet-node schemes discussed in this section. This makes it a suitable candidate for wet-node boundary condition method.

NOTE (EMV): Just a general comment on this section; I really like how you have a clear red thread throughout. You identify the problems with the most obvious option of equilibrium boundaries, show that the nonequilibrium extrapolation is an insufficient fix, and then describe NEBB as a sufficient fix. Very nice!

5.2.5 Open boundaries

Due to computational constraints it is often necessary to limit the size of the simulation domain to some truncated portion of it. Unfortunately, this simplification comes with a cost. By cropping the physical domain, new boundary conditions have to be prescribed at places where the physical problem had originally no boundaries at all. We call these *open boundaries*.

Open boundaries consist of *inlets* or *outlets* where the flow either enters or leaves the computational domain, and where we should impose, for example, velocity or density profiles. This is often non-trivial, and it can cause both physical and numerical difficulties.

Physically, the difficulty comes from the role of open boundaries: they are supposed to guarantee that what enters/leaves the computational domain is compatible with the bulk physics of the problem. However, this information is, in general, not accessible beforehand. This obliges us to rely on some level of *physical approximation*, *e.g.* assuming a fully developed flow at the inlet.

Numerically, open boundaries may also be complicated by a feature inherent to LBM: the presence of compressional waves, *cf.* chapter 12. The "simple" enforcement of either velocity or density at boundaries will generally reflect these waves back into the computational domain. Since this is undesirable, special non-reflecting open boundary conditions may be required in some cases. These are covered in section 12.4.

Open boundary conditions in the LBM are still under development, *e.g.* []. TODO (GS): Add reference. While more advanced strategies already exist to some degree

(see *e.g.* section 12.4), this section is on an introductory level. Here, we will revisit the application of basic link-wise (section 5.2.3) and wet-node boundary methods (section 5.2.4) to establish open boundaries. We will discuss how to implement a Dirichlet boundary condition for an imposed velocity or pressure profile using (i) the link-wise bounce-back method and (ii) the wet-node non-equilibrium bounce-back method. In terms of the algorithm, both schemes are similar for inlets and outlets.

NOTE (TK): Goncalo, you are covering bounce-back and the wet-node approach. Could you refer to more publications? For example Jonas Latt's paper about straight boundaries?

Velocity boundary conditions – the bounce-back approach

The Dirichlet condition for fluid velocity at open boundaries is easily realised by the bounce-back technique [18, 38]. The formulation is the same as in eq. (5.27) (or eq. (5.26)). The only conceptual difference is that the Dirichlet velocity correction $\boldsymbol{u}_B$ for a wall is generally non-zero for the tangential velocity, while in an inlet/outlet region the purpose is specifying a non-zero normal velocity. NOTE (TK): This is only true if we really limit the previous discussion to situations with tangential wall motion. This is not necessarily always the case. Also, inlets or outlets may involve non-zero tangential components. In both cases the bounce-back algorithm locates the boundary midway between the lattice nodes. This means the imposed Dirichlet boundary velocity $\boldsymbol{u}_B$ refers to the local velocity at $\boldsymbol{x}_B \pm \frac{1}{2}\boldsymbol{c}_i\Delta t$.

Pressure boundary conditions – the anti-bounce-back approach

The prescription of pressure at open boundaries can be performed with a technique similar to the bounce-back method, called the *anti-bounce-back* method [39, 53, 51]. Compared to the standard bounce-back rule for velocity, its formulation works with an inverted parity. This means that the sign of the post-collision populations participating in the anti-bounce-back dynamics is negative:

$$f_{\bar{i}}(\boldsymbol{x}_B, t+\Delta t) = -f_i^{\star}(\boldsymbol{x}_B, t) + 2w_i\rho_B\left[1 + \frac{(\boldsymbol{c}_i\cdot\boldsymbol{u}_B)^2}{2c_s^4} - \frac{\boldsymbol{u}_B\cdot\boldsymbol{u}_B}{2c_s^2}\right]. \tag{5.45}$$

Once again, the correction that specifies the Dirichlet macroscopic property at the boundary refers to the local density (pressure) at $\boldsymbol{x}_B \pm \frac{1}{2}\boldsymbol{c}_i\Delta t$. The notation in eq. (5.45) is the same as in eq. (5.27)

Eq. (5.45) requires $\boldsymbol{u}_B$ which is not generally known. The problem is similar to finding ρ_B for the velocity Dirichlet correction in the standard bounce-back rule, *cf.* eq. (5.27). A possible way to estimate $\boldsymbol{u}_B$ is interpolation. For example, following [51], we may estimate $\boldsymbol{u}_B$ as $\boldsymbol{u}_B = \boldsymbol{u}(\boldsymbol{x}_B \pm \Delta x) + \frac{1}{2}\left[\boldsymbol{u}(\boldsymbol{x}_B \pm \Delta x) - \boldsymbol{u}(\boldsymbol{x}_B \pm 2\Delta x)\right]$. NOTE (TK): This equation is more confusing than helpful. When do plus, when do

minus signs apply? It would be better to show the equation only for one case and say that the other is equally easy to write down. Also, you cannot add a scalar Δx to a vector $\boldsymbol{x}$. I have written it as $\boldsymbol{x}_B \pm \frac{1}{2}\boldsymbol{c}_i \Delta t$ above.

Since only the square of the boundary velocity appears in eq. (5.27), inaccuracies in the approximation of $\boldsymbol{u}_B$ are $O(\mathrm{Ma}^2)$ and therefore usually small. We stress that eq. (5.45) can be further augmented with a correction to eliminate second-order error terms. The explicit form of this correction can be found in [39, 53, 51] TODO (GS): (also Tallon paper).

Velocity boundary conditions – the wet node approach

The wet node formulation of velocity boundary conditions applies in the same manner for open boundaries as for walls. The only difference refers again to the velocity components specified. While for a moving solid boundaries it is the tangential velocity which describes the plane wall motion, in open boundaries mass flux is specified through the normal velocity component. Nevertheless, the ideas discussed in section 5.2.4 remain applicable here.

Pressure boundary conditions – the wet node approach

In the wet node approach, setting pressure and velocity boundary conditions is similar. In fact, the underlying algorithm is the same. The only difference happens in the strategy to find the unknown macroscopic properties at the boundary. For instance, for a prescribed inlet velocity, the two equations in eq. (5.32) are combined to find the unknown ρ which yields eq. (5.33). Contrarily, for a prescribed inlet pressure, these two equations are combined to solve for the unknown wall normal velocity. This means that eq. (5.32) is used to solve for u_y: $u_y = -1 + \frac{1}{\rho}[f_0 + f_1 + f_3 + 2(f_2 + f_5 + f_6)]$.

5.2.6 Corners

So far the discussion on boundary conditions has been limited to straight surfaces. It follows that in both 2D and 3D domains there are other geometrical features requiring specification.

In 2D domains we have to deal with the geometrical places where two straight surfaces intersect, called *corners*. In 3D cartesian domains, we need to consider both *corners* and *edges*, *i.e.* the places where two and three surfaces intersect. NOTE (TK): This is slightly misleading: a corner in 3D is a point where 3 surfaces intersect; an edge where 2 surfaces intersect.

The focus of this section is on the treatment of 2D corners. The 3D case can be deduced as an extension of the elements provided herein. That exercise will be pursued in section 5.3.4 and section A.7.

NOTE (TK): I have rewritten this paragraph (I found it too long). Instead some details are provided in the figure caption. Usually the solution of the flow field should be smooth at corners.[24] While it is clear that corners are inherent to rectangular domains, they can sometimes be avoided by switching from velocity/pressure to periodic boundary conditions as shown in fig. 5.24.

It is evident that, in most practical flow geometries, the treatment of corners cannot be avoided. The flow over a step is one such example; it features both concave and convex corners, *cf.* fig. 5.25(a). NOTE (TK): I have simplified the following sentence. Another example, as shown in fig. 5.25(b), is a staircase approximation of an inclined wall.

Even in problems where corners account for only a few points in the solution domain, like in fig. 5.25(a), they should not be underestimated. Recall that, according to the discussion in section 5.1.3, one point suffices to contaminate the numerical solution everywhere. This has been shown by Junk and Yang [54] in their study for developing one-point boundary conditions for the LBM. NOTE (TK): The meaning of the following sentence is not very clear. It turns out that, similarly to the case of plane boundaries, the state of affairs in the modelling of LB corners remains also somehow unsettled.

One of the earliest systematic approaches to treat corners in LBM was proposed by Maier et al. [] TODO (GS): Reference. Later, Zou and He [23] suggested another

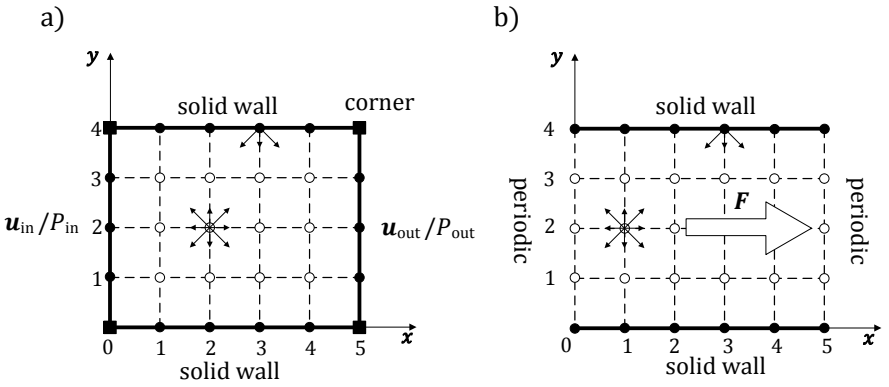


Fig. 5.24: Different implementations of a Poiseuille flow (○: fluid nodes, ●: boundary nodes, ■: corner nodes). (a) Velocity/pressure boundary conditions for the inlet and oulet and (b) periodic boundary conditions with a force density instead of a pressure gradient. While corners have to be treated in (a), this is not necessary in (b).

[24] Here we will exclude cases where this requirement is violated. An example is the lid-driven cavity flow [], an often used benchmark problem featuring a velocity discontinuity at corners.

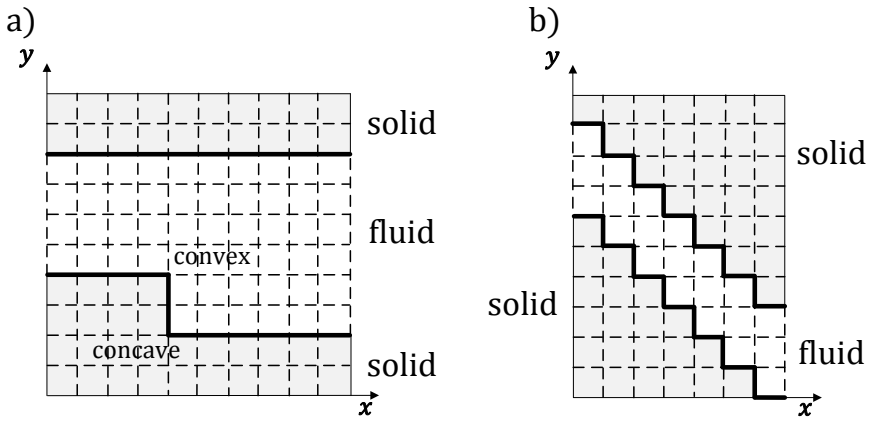


Fig. 5.25: Examples of flow geometries with corners: (a) channel with a sudden expansion, (b) discretisation of a diagonal channel.

approach to specify corners, based on their NEBB method [23]. Yet, as their approach lacked generality in 3D implementations, Hecht and Harting [49] extended it to 3D domains (using the D3Q19 as an example). Still, the number of studies concerning the treatment of corners in LBM is limited. NOTE (TK): "Limited" does not rule out "large". To me it appears that there is a plethora of boundary treatments for corners. There are also some works from Ian Halliday and others! The few works contributing to this topic are [26] TODO (GS): (Verschaeve, Tim Reis, Raul Machado), which are simpler to apply but limited to 2D problems, and [10, 54, 39, 13] which are applicable to both 2D and 3D geometries but more complex in implementation.

In the explanation that follows we will limit ourselves to the implementation of corners for the bounce-back rule as a link-wise approach and the NEBB method as a wet-node procedure. Recall that both methods have fundamental differences as shown in fig. 5.26.

Corners in link-wise approach: bounce-back rule

The bounce-back rule applied to corners follows the principles of straight boundaries: the unknown populations leaving corners are determined from the known incoming ones. This rule applies to both concave and convex corners as discussed below.

First, let us focus on a *concave corner*. Observing fig. 5.26(a), we identify three unknown populations: f_1, f_2 and f_5, which emanate from the region inside the corner into the computational domain (two boundary nodes and one fluid node, respectively). According to the bounce-back rule, the unknown populations at the corner node $\boldsymbol{x}_B$ are determined as follows:

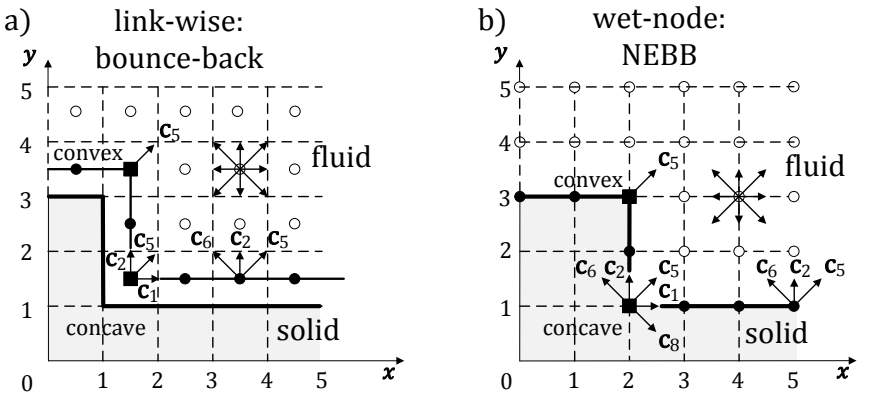


Fig. 5.26: Corner condition for (a) link-wise (bounce-back) and (b) wet-node (NEBB) methods (○: fluid nodes, ●: boundary nodes, ■: corner nodes). Each case illustrates which unknown populations require specification at boundary and corner nodes.

$$\begin{aligned} f_1(\boldsymbol{x}_B, t+\Delta t) &= f_3^\star(\boldsymbol{x}_B, t), \\ f_2(\boldsymbol{x}_B, t+\Delta t) &= f_4^\star(\boldsymbol{x}_B, t), \\ f_5(\boldsymbol{x}_B, t+\Delta t) &= f_7^\star(\boldsymbol{x}_B, t). \end{aligned} \tag{5.46}$$

Note that if the two meeting walls are modelled with the bounce-back rule then it is sufficient to set $f_5(\boldsymbol{x}_B, t+\Delta t) = f_7^\star(\boldsymbol{x}_B, t)$ in the concave corner node. This is possible because the bounce-back algorithm does not deal with tangential populations. NOTE (TK): Is it really necessary to say that explicitly? In principle you are just explaining the bounce-back rule again.

Second, let us address the case of a *convex corner*, *cf.* fig. 5.26(a). Here, there is only one unknown population emanating from the corner: f_5. The bounce-back rule then reads:

$$f_5(\boldsymbol{x}_B, t+\Delta t) = f_7^\star(\boldsymbol{x}_B, t). \tag{5.47}$$

All remaining populations pass the corner without bounce-back since they come from neighbouring inside computational nodes (either fluid or boundary nodes).

From the application point of view, the *advantages and disadvantages of the bounce-back rule in corners* are similar to those at planar surfaces. The positive aspects remain its ease of implementation, strict conservation of mass and good stability characteristics (even close to $\tau = 0.5$). On the other hand, the (possible) lower accuracy of the method, also featuring viscosity-dependent errors (if using the SRT model), remains the main disadvantage of the method.

Corners in wet node approach: NEBB method

NOTE (TK): Changed sentence to If the wet node approach is used, corners have to be treated in a special way. NOTE (TK): Changed sentence to Overall, we have to ensure that known and unknown populations yield the desired macroscopic properties at corners, just as they do at straight boundaries. Yet, before going into details about how to prescribe corner populations, we need to know which macroscopic values they are assigned.

In case the corner connects a density (pressure) boundary with a velocity boundary, the macroscopic properties at the corner can be established by the values from each side of the boundary. However, if both meeting boundaries apply velocity boundary conditions, the density (pressure) value at the corner will be missing (and the other way around). Given that eq. (5.32) only works at straight surfaces, finding the corner density (pressure) generally requires extrapolating its value from neighbouring nodes [14]. This way, all macroscopic quantities prescribed at corners (ρ, u_x, u_y) are known, and we can proceed with the determination of the missing mesoscopic populations. NOTE (TK): Do you think it is necessary to say more about the extrapolation procedure?

First, let us address the *concave corner* depicted in fig. 5.26(b). We find six unknowns: f_0, f_1, f_2, f_5, f_6 and f_8. Three of them can be determined immediately by applying the non-equilibrium bounce-back rule:

$$\begin{aligned} f_1^{\mathrm{neq}} &= f_3^{\mathrm{neq}}, \\ f_2^{\mathrm{neq}} &= f_4^{\mathrm{neq}}, \\ f_5^{\mathrm{neq}} &= f_7^{\mathrm{neq}}. \end{aligned} \tag{5.48}$$

By substituting $f_i^{\mathrm{neq}} = f_i - f_i^{\mathrm{eq}}$ and using the standard LB equilibrium for f_i^{eq}, eq. (5.48) simplifies to

$$\begin{aligned} f_1 &= f_3 + \tfrac{2}{3}\rho u_x, \\ f_2 &= f_4 + \tfrac{2}{3}\rho u_y, \\ f_5 &= f_7 + \tfrac{1}{6}\rho(u_x + u_y). \end{aligned} \tag{5.49}$$

However, the problem is not closed yet. There are still three unknown populations: f_0, f_6 and f_8. We can work out a solution by exploring the fact that the number of unknown populations to determine matches that of the conservation laws to satisfy:

$$\begin{aligned} \rho &= \underline{f_0} + f_1 + f_2 + f_3 + f_4 + f_5 + \underline{f_6} + f_7 + \underline{f_8}, \\ \rho u_x &= (f_1 + f_5 + \underline{f_8}) - (f_3 + \underline{f_6} + f_7), \\ \rho u_y &= (f_2 + f_5 + \underline{f_6}) - (f_4 + f_7 + \underline{f_8}), \end{aligned} \tag{5.50}$$

where the underlined terms are unknown. In principle, the two non-rest populations, f_6 and f_8, can be determined from solving the second and third equations. Unfortunately, the system is non-invertible. Therefore, in order to make it solvable, we must introduce an additional constraint. A possible (but non-unique) constraint may demand that f_6 and f_8 contribute equally to the momentum. NOTE (TK): What does this mean? That both of them have the same magnitude? This choice yields

$$\begin{aligned} f_6 &= \tfrac{1}{12}\rho(u_y - u_x), \\ f_8 &= \tfrac{1}{12}\rho(u_x - u_y). \end{aligned} \tag{5.51}$$

NOTE (TK): This means that one of them will be negative unless both velocity components are identical. I am not sure that this equation is actually correct. For $u = 0$ we get $f_6 = f_8 = 0$!

At this point it is worth explaining why we need to compute these two populations, contrary to the bounce-back case. As fig. 5.26(b) shows, f_6 and f_8 belong to a link that never points into the flow domain, called *buried link* [] (Mayer). Since these populations never stream into the fluid region, they could be perceived as irrelevant. However, they are necessary for the macroscopic behaviour of the corner, *i.e.* to satisfy the conservation laws at the node, eq. (5.50). Since bounce-back satisfies mass conservation out of the box, f_6 and f_8 do not have to be treated in a special way in the bounce-back approach. NOTE (TK): I am missing a conclusion here. Is the above approach so set f_6 and f_8 acceptable, or is there a better option?

At last, we need to determine the rest population f_0. We can enforce the mass conservation at the corner site by using the first equation in eq. (5.50):

$$f_0 = \rho - \sum_{i=1}^{8} f_i. \tag{5.52}$$

Altogether, the unknown populations in the concave corner are determined by eq. (5.49), eq. (5.51) and eq. (5.52). It is interesting to note that, if the corner is stationary, *i.e.* $u_x = u_y = 0$, the present approach reduces to the (node) bounce-back rule, supplemented by eq. (5.52). NOTE (TK): I have changed the equation reference. Please check that it is the correct one.

Concerning the case of a convex corner in fig. 5.26(b), we identity only one missing population streaming out of the solid region: f_5. Consequently, if using the same procedure as for concave corners, the problem becomes over-specified. The consequence is that we cannot possibly satisfy the no-slip condition at convex corner nodes [] (Mayer). An alternative to overcome this problem is to employ the simple (node) bounce-back rule:

$$f_5 = f_7. \tag{5.53}$$

NOTE (TK): I have commented out the footnote. The reader has just read about your statement; there is no reason to repeat it right away.

From the discussion above we can conclude that the complexity of the wet node approach for corners is its main drawback. This comes along with the lack of flexibility in handling certain geometrical features, such as convex corners. These problems are still worse in 3D implementations where we have to deal with an even larger number of unknown populations [49], see also section 5.3.4 and appendix A.7. In addition to these problems, the NEBB method also suffers from other disadvantages, *e.g.* the violation of the exact mass conservation and weak stability for $\tau \to \frac{1}{2}$, *cf.* section 5.2.4. Yet, this approach generally offers a superior accuracy

compared to the bounce-back rule. Furthermore, the wet node approach provides second-order viscosity-independent solutions with the BGK collision operator.

5.3 Further topics on boundary conditions

NOTE (TK): In my opinion, there should always be an introductory sentence between a section and a subsection headline, ideally referring to all following subsections (mini summary or TOC).

5.3.1 The Chapman-Enskog analysis for boundary conditions

LB boundary conditions rely on specific closure rules. Those are different from the LB bulk rules (collision and propagation). Consequently, the result from section 4.1 for the bulk behaviour cannot be expected to hold at boundaries. Still, the techniques introduced there are useful, and the present section shows how to use the Chapman-Enskog analysis to determine the relation between macroscopic and LB boundary conditions. Owing to their conceptual differences, we will treat the bounce-back and the non-equilibrium bounce-back techniques individually.

Link-wise approach: bounce-back

Recalling section 5.2.3, the macroscopic Dirichlet velocity condition described by the bounce-back rule in eq. (5.27) applies at lattice links rather than at grid nodes. However, the LBM, as a typical grid-based method, only provides solutions at the nodes. NOTE (EMV): Replaced "It follows that" with "However, the". Hence, the understanding of what happens at links must be examined in terms of *continuation* behaviour. Such a study can be carried out based on a Taylor series expansion. If we crave second-order accuracy (*i.e.* the same accuracy as for the bulk LB solution), then we can use the Taylor expansion to approximate the velocity u_x at a point x_{BC} located $\Delta x/2$ beside a grid node x:

$$u_x|_{x_{\mathrm{BC}}} = u_x|_x + \frac{\Delta x}{2}\,\partial_y u_x\Big|_x + \frac{1}{2}\left(\frac{\Delta x}{2}\right)^2 \partial_y\partial_y u_x\Big|_x + O\left(\Delta x^3\right), \quad x_{\mathrm{BC}} = x + \frac{\Delta x}{2}. \tag{5.54}$$

The necessary condition for the bounce-back rule to describe the wall velocity condition with formal second-order accuracy at $x_{\mathrm{BC}} = x + \frac{\Delta x}{2}$ is that its macroscopic structure satisfies eq. (5.54).

The evaluation of the relation between LB and macroscopic behaviour is therefore required. This can be done through the Chapman-Enskog analysis. Hereby, the LB solution is expanded to second-order $f_i \approx f_i^{\mathrm{eq}} + \epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)}$, and the procedure as explained in section 4.1 is employed. NOTE (EMV): For the following, you should point out that you're taken the approach described in section 4.2.3 of neglecting the time derivative *a priori*, instead connecting the different orders of smallness through the solvability condition. If we consider a time-independent solution where $\partial_t f_i = 0$ holds, we obtain:

$$f_i^{(1)} = -\tau c_{i\alpha} \partial_\alpha^{(1)} f_i^{\mathrm{eq}}, \tag{5.55a}$$

$$f_i^{(2)} = -\left(\tau - \frac{\Delta t}{2}\right) c_{i\alpha} \partial_\alpha^{(1)} f_i^{(1)}. \tag{5.55b}$$

For simplicity, let us focus here on a linearised flow which is described by the equilibrium $f_i^{\mathrm{eq}} = w_i(\rho + \rho_0 c_{i\gamma} u_\gamma / c_\mathrm{s}^2)$ introduced in section 4.3.1. In this case, eq. (5.55) becomes

$$f_i^{\mathrm{eq}} = w_i \left(\rho + \rho_0 \frac{c_{i\gamma} u_\gamma}{c_\mathrm{s}^2}\right), \tag{5.56a}$$

$$f_i^{(1)} = -\tau w_i c_{i\alpha} \partial_\alpha^{(1)} \left(\rho + \rho_0 \frac{c_{i\gamma} u_\gamma}{c_\mathrm{s}^2}\right), \tag{5.56b}$$

$$f_i^{(2)} = \tau \left(\tau - \frac{\Delta t}{2}\right) w_i c_{i\alpha} c_{i\beta} \partial_\alpha^{(1)} \partial_\beta^{(1)} \left(\rho + \rho_0 \frac{c_{i\gamma} u_\gamma}{c_\mathrm{s}^2}\right). \tag{5.56c}$$

TODO (GS): Throughout this chapter, it has to be checked whether τ is a relaxation time (in seconds) or a relaxation parameter (a number). This is not consistently done in the book. NOTE (TK): Is it really necessary to keep the superscript (1) when you write the spatial gradient?

Inserting the Chapman-Enskog decomposition of the populations into the bounce-back formula, eq. (5.27), we obtain

$$f_{\bar{i}}^{\mathrm{eq}} + \epsilon f_{\bar{i}}^{(1)} + \epsilon^2 f_{\bar{i}}^{(2)} = f_i^{\mathrm{eq}} + \left(1 - \frac{\Delta t}{\tau}\right)\left(\epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)}\right) - 2 w_i \frac{c_{i\gamma}\, u_\gamma\big|_{x_\mathrm{BC}}}{c_\mathrm{s}^2}. \tag{5.57}$$

Then, by substituting eq. (5.56) into eq. (5.57) and after performing some algebraic manipulations (using $p = c_\mathrm{s}^2 \rho$), we obtain

$$\begin{aligned} c_{i\gamma}\, u_\gamma\big|_{x_\mathrm{BC}} = c_{i\gamma} u_\gamma + \frac{1}{2} c_{i\alpha} \partial_\alpha (c_{i\gamma} u_\gamma) + \left(\tau - \frac{\Delta t}{2}\right)^2 c_{i\alpha} c_{i\beta} \partial_\alpha \partial_\beta (c_{i\gamma} u_\gamma) \\ - \left(\tau - \frac{\Delta t}{2}\right)\left[c_{i\alpha} \partial_\alpha \left(\frac{p}{\rho_0}\right) + \frac{1}{2} c_{i\alpha} c_{i\beta} \partial_\alpha \partial_\beta \left(\frac{p}{\rho_0}\right)\right]. \end{aligned} \tag{5.58}$$

The above equation describes the projection of the macroscopic solution onto the lattice. NOTE (EMV): Unclear what you mean by this. NOTE (TK): Yes, I agree

with EMV. For concreteness, let us assume that this problem applies for a lattice-aligned unidirectional flow where $c_{i\gamma}u_\gamma = c_{ix}u_x$ and $c_{i\alpha}\partial_\alpha = c_{iy}\partial_y$. In order to compare the resulting equation with eq. (5.54), the pressure terms have to be re-expressed in the form of velocity corrections. This is possible by assuming that we are solving for a unidirectional pressure-driven Stokes problem (*i.e.* a Poiseuille flow) which is governed by $\partial_x(p/\rho_0) = c_s^2(\tau - \Delta t/2)\partial_y\partial_y u_x$ and $\partial_x\partial_x(p/\rho_0) = 0$. Based on these assumptions, eq. (5.58) reduces to

$$u_x|_{x_{\mathrm{BC}}} = u_x|_x + \frac{\Delta x}{2}\,\partial_y u_x\Big|_x + \underbrace{(c^2 - c_s^2)\left(\tau - \frac{\Delta t}{2}\right)^2}_{=\Delta x^2/8}\partial_y\partial_y u_x\Big|_x . \tag{5.59}$$

NOTE (GS): Give a better explanation in what follows! The equivalence of eq. (5.59) and eq. (5.54) is established if the Laplacian term is weighted by $c^2/8$ where $c = \Delta x/\Delta t$. Given that $c_s^2 = c^2/3$ is fixed, the boundary location at $\Delta x/2$ becomes only formally second-order accurate if $(\tau - \Delta t/2)^2 = (3/16)\Delta t^2$ [] [Irina]. Other τ values lead to a shift of the location of the effective boundary condition. This artifact can equivalently be interpreted as a velocity slip if we assume that the boundary remains halfway between lattice nodes. In any case, this *unphysical* dependence of the boundary condition on τ (and, therefore, on fluid viscosity) is a serious limitation of the bounce-back method combined with the SRT collision operator. A solution for this problem is the adoption of a collision model with additional degrees of freedom. Adequate solutions are the TRT and MRT collision models as discussed in chapter 10.

NOTE (EMV): Nice explanation in this entire derivation; I was surprised at how easy it was to understand. One thing: At the end, maybe you want to refer to the earlier section where you showed numerically that the solution is exact for this value of τ?

Wet-node approach: non-equilibrium bounce-back

The Navier-Stokes equations only contain ϵ-order non-equilibrium corrections. NOTE (EMV): For the previous sentence, may I instead suggest something like "As shown in section **??**, only the ϵ-order non-equilibrium corrections are required to recover the Navier-Stokes equations." Hence, *at lattice nodes*, the macroscopic content of LB dynamics is completely determined by $f_i \approx f_i^{\mathrm{eq}} + \epsilon f_i^{(1)}$. NOTE (EMV): I don't agree; higher-order terms can mess with the macroscopic content to get non-NS behaviour. Recall that such a truncation order was shown in section 4.1 to be sufficient to describe the LBM macroscopic fluid dynamics in bulk nodes. NOTE (EMV): I'd rather front-load the paragraph with this statement; see my above comment. Given that wet boundary nodes operate in the same way as bulk nodes, then $f_i^{\mathrm{neq}} \approx \epsilon f_i^{(1)}$ is also sufficient for wet-node boundary conditions.

The necessary condition for the *wet-node* rule NOTE (EMV): Point out that you're talking about NEBB. to describe the wall velocity with second-order accuracy at the wall is that its macroscopic structure features correct $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ and $f_i^{\mathrm{neq}}(\nabla \boldsymbol{u})$ terms. NOTE (EMV): This is a bit of a weird statement; you can't see these mesoscopic quantities in the macroscopic structure. NOTE (TK): I agree; this statement is not precise and unclear.

From the Chapman-Enskog analysis, the macroscopic content of wet boundary nodes (at $\boldsymbol{x}_{\mathrm{B}}$) must be

$$f_i(\boldsymbol{x}_{\mathrm{B}}) = f_i^{\mathrm{eq}}(\boldsymbol{x}_{\mathrm{B}}) + f_i^{\mathrm{neq}}(\boldsymbol{x}_{\mathrm{B}}) \simeq \left(1 - \epsilon \frac{\tau}{\Delta t} c_{i\alpha} \partial_\alpha^{(1)}\right) f_i^{\mathrm{eq}}(\boldsymbol{x}_{\mathrm{B}}). \tag{5.60}$$

The assignment of correct $f_i^{\mathrm{eq}}(\boldsymbol{x}_{\mathrm{B}}) = f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ is straightforward. The tricky part lies in $c_{i\alpha}\partial_\alpha f_i^{\mathrm{eq}}(\boldsymbol{x}_{\mathrm{B}})$. This contribution can be approximated as $c_{i\alpha}c_{i\beta}\partial_\alpha u_\beta$, by neglecting higher-order terms, such as second-order derivatives of pressure and non-linear velocity terms. NOTE (EMV): Can't you clarify this by explicitly using the linear equilibrium as above? And if you do, how does the density derivative disappear?

By taking a closer look, a crucial simplification can be made for $f_i^{\mathrm{neq}}(\boldsymbol{x}_{\mathrm{B}})$ by recognising that individual values of $f_i^{\mathrm{neq}}(\boldsymbol{x}_{\mathrm{B}})$ are actually immaterial for recovering correct hydrodynamics. From the Chapman-Enskog analysis, only the second-order moment $\Pi_{\alpha\beta}^{\mathrm{neq}} = \sum_i c_{i\alpha}c_{i\beta}f_i^{\mathrm{neq}}$ plays a role. Hence, the content of $f_i^{\mathrm{neq}}(\boldsymbol{x}_{\mathrm{B}})$ can be re-expressed as a symmetric tensor: NOTE (EMV): This is not a tensor. And I'm not sure "symmetric" is the right term? You might want to point out explicitly that you mean that $f_i^{\mathrm{neq}} = f_{\bar{i}}^{\mathrm{neq}}$ since $c_{\bar{i}\alpha} = -c_{i\alpha}$ so that $c_{\bar{i}\alpha}c_{\bar{i}\beta} = c_{i\alpha}c_{i\alpha}$; that should make it clear.

$$f_i^{\mathrm{neq}} \simeq -\frac{\tau}{\Delta t} \frac{c_{i\alpha}c_{i\beta}}{c_{\mathrm{s}}^2} \rho_0 S_{\alpha\beta} \tag{5.61}$$

where $S_{\alpha\beta} = \frac{1}{2}(\partial_\alpha u_\beta + \partial_\beta u_\alpha)$ is the strain-rate tensor. NOTE (EMV): It's not immediately obvious where this equality comes from.

The NEBB method proposed by Zou-He [23] is one way to satisfy eq. (5.61). In fact, the requirement of equality between symmetrical non-equilibrium components, *e.g.* $f_2^{\mathrm{neq}} = f_4^{\mathrm{neq}}$, is synonymous to enforcing correct $S_{\alpha\beta}$.[25] NOTE (EMV): I don't understand this statement. NOTE (TK): I do, but it is not very clear. Other wet-node techniques, *e.g.* [20, 21, 22, 23, 24, 25, 14] can be understood as different strategies to reach the same consistency with eq. (5.61).[26]

[25] The symmetry of $S_{\alpha\beta}$ is not affected by the addition of transversal momentum corrections in the balance of non-equilibrium populations. The reason is that they do not interfere at the level of gradients of hydrodynamic fields.

[26] In contrast to these works, [26] showed that it is actually possible to relax on the symmetry condition of the non-diagonal components of f_i^{neq} under some specific conditions. However, this case is not the most general, and we refer to [26] for more details.

5.3.2 Mass conservation at solid boundaries

An important feature of LBM boundary conditions is whether they conserve the total mass inside the system. Mass conservation is a crucial requirement for the modelling of many physical processes such as compressible, multi-phase or multi-component flows.

The LB algorithm ensures that mass is conserved locally since collision leaves $\sum_i f_i = \rho$ invariant.[27] By inference, mass is also conserved globally in a periodic domain. However, there is no *a priori* guarantee that this property holds at boundary nodes which necessarily must behave differently than bulk fluid nodes. NOTE (EMV): Added the last clause.

For the total mass inside the system to remain constant, it is necessary that what leaves the domain (given by the post-collision values) is exactly balanced by what enters the domain during streaming. Let us take a top wall as shown in fig. 5.21 as an example to illustrate the problem. Denoting the incoming mass as ρ^{in}, the outgoing mass as ρ^{out}, NOTE (EMV): Remember that text superscripts should always be in mathrm mode. and the mass variation inside the system as $\Delta M(t)$ we have: NOTE (TK): Please do not mix mass and density. It should be either that or the other, unless you have a volume integral.

$$\Delta M(t) = \rho^{\text{in}} - \rho^{\text{out}} = (f_4 + f_7 + f_8) - (f_2^\star + f_5^\star + f_6^\star). \tag{5.62}$$

NOTE (EMV): Is this equation split really necessary or useful? Also, shouldn't the incoming populations also be starred as post-collision? Also, is ΔM supposed to be the mass variation inside the *entire* system? In that case, this is confusing as it looks like the RHS belongs to a single node.

One of the main differences between wet node techniques, where ρ^{in} is reconstructed explicitly, and link-wise methods, such as bounce-back, where ρ^{in} is found through simple reflections, is the possibility of the former to not conserve mass.

By construction, the bounce-back rule satisfies $\Delta M(t) = 0$ at every time step because $f_{\bar{i}} = f_i^\star \Rightarrow \rho^{\text{in}} = \rho^{\text{out}}$. NOTE (EMV): Made the $\Leftrightarrow$ into a $\Rightarrow$ as the LHS does not follow uniquely from the RHS. However, according to eq. (5.62), bounce-back is only mass-conserving when dealing with resting or tangentially moving walls. Normal wall motion also violates mass conservation.

Wet node methods operate with specific rules to find the incoming populations, NOTE (EMV): (Changed ρ^{out} to "the incoming populations".) which may not necessarily use ρ^{out} to enforce $\Delta M(t) = 0$.[28] The NEBB method is an example where

[27] Here, and throughout this discussion, we will assume no mass source is present.

[28] In fact, the prescription of rules to explicitly enforce local conservation of mass may lead to incorrect solutions [10, 55]. As demonstrated theoretically in [10], when explicitly enforcing $\Delta M(t) = 0$, we are indirectly introducing non-hydrodynamic boundary layers which can be quan-

$\Delta M(t) \neq 0$ is permitted. We can quantify the mass defect in NEBB by writing $f_i = f_i^{\mathrm{eq}} + f_i^{\mathrm{neq}}$ in eq. (5.62) and then applying the bounce-back rule to f_i^{neq}. NOTE (EMV): Couldn't you just say $f_i = f_i^{\mathrm{eq}} + f_{\bar{i}}^{\mathrm{neq}}$ earlier? After some algebra the result is

$$\Delta M(t) = -\rho u_y + \frac{\Delta t}{\tau}\left(f_4^{\mathrm{neq}} + f_7^{\mathrm{neq}} + f_8^{\mathrm{neq}}\right). \tag{5.63}$$

As expected, for a plane horizontal wall, a mass flux imbalance has to exist for a normal boundary movement or, equivalently, for a boundary with vertical fluid injection, *i.e.* $\rho u_y \neq 0$. The problem is that even when this term is zero we still can have $\Delta M(t) \neq 0$ from the remaining contribution due to the non-equilibrium populations. This is a purely numerical artifact.

We can analyse this additional term by resorting to a Chapman-Enskog analysis, *cf.* section 4.1:

$$f_i^{\mathrm{neq}} \approx \epsilon f_i^{(1)} + O(\epsilon^2) \approx -\frac{\tau}{\Delta t} w_i \frac{c_{i\alpha}c_{i\beta} - c_{\mathrm{s}}^2\delta_{\alpha\beta}}{c_{\mathrm{s}}^2}\partial_\beta u_\alpha + O(\epsilon^2). \tag{5.64}$$

Substituting this approximation of f_i^{neq} into eq. (5.63) we get

$$\Delta M(t) \approx -\rho u_y + c_{\mathrm{s}}^2 \partial_y u_y + O(\epsilon^2) \tag{5.65}$$

which has been simplified using the no-slip wall condition $\partial_x u_x = \partial_x u_y = 0$. It follows the continuity equation $\partial_t \rho = -\partial_y u_y$. NOTE (EMV): Missing a ρ on the RHS there and in the display equation above. Hence, in time-dependent flows we necessarily have $\Delta M(t) \neq 0$. Yet, even in steady flows, the NEBB method can only guarantee mass conservation up to the numerical approximation of the method, *i.e.* $\Delta M(t) \approx O(\epsilon^2)$.[29] Fortunately, in some particular wall configurations, these numerical errors may either be absent or cancel due to symmetry, *e.g.* plane channel flows aligned with one of the lattice axes. Under these circumstances, the steady-state regime leads to global mass conservation.

When the amount of mass varies within the system, there is no longer any strictly stationary regime [11]. The steady-state condition in this case should be understood in the sense of some suitable stopping criterion for the LB simulation [11, 12, 57, 55]. A possible solution may be re-scaling the quantities by the local density at the specific measurement time.

NOTE (GS): We emphasize that the pressure solution is defined up to a constant. When the boundary schemes allow a mass flux across the wall, the obtained density (pressure) distribution can happen to be non-stationary although the velocity field results in a steady solution. (Irina, Study of simple solutions)

---

tified for Couette and Poiseuille flows in arbitrary inclined channels. These artifacts prevail in more complex flow configurations, as was numerically confirmed in [41, **?**]. NOTE (EMV): Maybe "remain" instead of "prevail"?

[29] There are some wet-node strategies where mass conservation is enforced exactly, *e.g.* [25, 56, 26]. However, their actual merits remain to be demonstrated in face of the theoretical criticisms put forward in [10].

5.3.3 Momentum exchange method

NOTE (TK): We should briefly mention MEA for wet node boundaries.

In many situations one is interested in the force and torque acting on structures immersed in a fluid, for example due to wind or water flow around a ship or a bridge. Typical quantities of interest are the drag and lift coefficients.

Generally, the total force $\boldsymbol{F}$ acting on a boundary area A can be written as the surface integral of the *traction* vector $\boldsymbol{T} = \boldsymbol{\sigma}^{\mathrm{w}} \cdot \hat{\boldsymbol{n}}$,

$$F_\alpha = \int \mathrm{d}A\, T_\alpha = \int \mathrm{d}A\, \sigma^{\mathrm{w}}_{\alpha\beta} \hat{n}_\beta, \tag{5.66}$$

TODO (TK): Check the sign. where $\boldsymbol{\sigma}^{\mathrm{w}}$ is the stress tensor at the surface and $\hat{\boldsymbol{n}}$ the unit normal vector pointing from the surface into the fluid. The key question here is how to evaluate the integral in a computer simulation. The conventional approach is to compute the stress tensor $\boldsymbol{\sigma}$ at the surface, *e.g.* using finite differences, to approximate the integral in eq. (5.66). This can introduce additional errors and require tedious computations, in particular in 3D and for arbitrarily shaped surfaces [58].

There exists an alternative option to evaluate eq. (5.66) in the LBM, thanks to its kinetic origin. Ladd [18, 42] recognised that one can take advantage of the populations f_i which can be interpreted as the local density of particles with momentum $f_i \boldsymbol{c}_i$. Instead of finding $\boldsymbol{\sigma}$ everywhere on the surface, one just has to identify those populations f_i which cross the boundary (both from the fluid into the solid and the other way around) and sum up all corresponding momentum contributions. This leads to the *momentum exchange algorithm* (MEA).

Momentum exchange in the bounce-back method

Consider a planar bottom wall along the x-axis in 2D as shown in fig. 5.11. During propagation, the populations f_4, f_7 and f_8 stream from fluid to solid nodes. They hit the boundary half-way on their journey, are bounced back and continue their propagation as f_2, f_5 and f_6 towards their original nodes. Effectively it seems as if the populations f_4, f_7 and f_8 vanished into the boundary and f_2, f_5 and f_6 emerged from the wall.

What does this mean for the momentum exchange between fluid and wall? On the one hand, momentum is carried by f_4, f_7 and f_8 to the wall, but at the same time momentum is transported by f_2, f_5 and f_6 from the wall to the fluid. The difference is the net momentum transfer. Note that those populations which move parallel to the boundary are not relevant as they do not contribute to the momentum transfer between fluid and solid.

A population f_i moving from a fluid node $\boldsymbol{x}^{\mathrm{f}}$ to a solid node $\boldsymbol{x}^{\mathrm{s}} = \boldsymbol{x}^{\mathrm{f}} + \boldsymbol{c}_i \Delta t$ is bounced back mid-way,[30] at a wall location $\boldsymbol{x}^{\mathrm{w}} = \frac{1}{2}(\boldsymbol{x}^{\mathrm{f}} + \boldsymbol{x}^{\mathrm{s}}) = \boldsymbol{x}^{\mathrm{f}} + \frac{1}{2}\boldsymbol{c}_i \Delta t$.

[30] The MEA also works for curved boundaries as we will discuss in section 11.2.

Those lattice links $\boldsymbol{c}_i$ connecting a fluid and a solid node are called *boundary links*. For each boundary link there are exactly two populations crossing the wall, one incoming population f_i^{in} (streaming from the fluid into the wall) and one outgoing population $f_{\bar{i}}^{\text{out}}$ (moving from the wall into the fluid). All populations moving along boundary links contribute to the momentum exchange. In order to keep track of the boundary links we denote them $\boldsymbol{x}_i^{\text{w}}$; this indicates that population f_i crosses the wall at location $\boldsymbol{x}^{\text{w}}$ in direction $\boldsymbol{c}_i$.

Let us pick out a single population, say f_4, and investigate its fate more closely. Just before bounce-back, the incoming population f_4^{in} carries the momentum $\boldsymbol{p}_4^{\text{in}} = f_4^{\text{in}}\boldsymbol{c}_4$, but right after, it becomes the outgoing population f_2^{out} with momentum $\boldsymbol{p}_2^{\text{out}} = f_2^{\text{out}}\boldsymbol{c}_2$. During bounce-back, the wall has to absorb the recoil such that the total momentum of the fluid and the wall is conserved. Bounce-back happens at location $\boldsymbol{x}_4^{\text{w}}$, so we can write the corresponding momentum exchange as $\Delta\boldsymbol{p}(\boldsymbol{x}_4^{\text{w}}) = \boldsymbol{p}_4^{\text{in}} - \boldsymbol{p}_2^{\text{out}}$.

Example 5.3. For a stationary wall, bounce-back dictates $f_2^{\text{out}} = f_4^{\text{in}}$. Hence, the momentum transfer due to the bounce-back of f_4 at $\boldsymbol{x}_4^{\text{w}}$ is $\Delta\boldsymbol{p}(\boldsymbol{x}_4^{\text{w}}) = \boldsymbol{p}_4^{\text{in}} - \boldsymbol{p}_2^{\text{out}} = f_4^{\text{in}}\boldsymbol{c}_4 - f_2^{\text{out}}\boldsymbol{c}_2 = 2f_4^{\text{in}}\boldsymbol{c}_4$ since $\boldsymbol{c}_2 = -\boldsymbol{c}_4$. This is a contribution to the fluid pressure at the wall since f_4 is moving in normal direction.

The same idea holds independently for all other populations which are bounced back anywhere at the boundary. The total momentum exchange between the fluid and the wall is the sum of all those contributions by all bounced back populations.[31] It is therefore straightforward to compute the momentum exchange for any surface which is described *via* simple bounce-back boundary conditions:

1. Identify all links between boundary and solid nodes with locations $\boldsymbol{x}_i^{\text{w}}$. If the boundary is stationary, the list can be constructed once and stored in memory for the whole simulation.[32]
2. At each time step (or each other interval when the momentum exchange is required), evaluate the incoming and bounced back populations f_i^{in} and $f_{\bar{i}}^{\text{out}}$ at each of the identified links.
3. The total momentum exchange during one streaming step is the sum over all identified links and, due to $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$, can be written as

$$\Delta\boldsymbol{P} = \Delta x^3 \sum_{\boldsymbol{x}_i^{\text{w}}} \Delta\boldsymbol{p}(\boldsymbol{x}_i^{\text{w}}) = \Delta x^3 \sum_{\boldsymbol{x}_i^{\text{w}}} \left(f_i^{\text{in}} + f_{\bar{i}}^{\text{out}}\right)\boldsymbol{c}_i. \tag{5.67}$$

   The prefactor Δx^3 in 3D (or equivalently Δx^2 in 2D) ensures that the result is a momentum rather than a momentum density.

If we now assume that the momentum is exchanged smoothly during one time step Δt, we can easily find the force acting on the boundary:

[31] Note that the number of identified links varies with the chosen lattice. For example, there will be more links for the same geometry when D3Q27 rather than D3Q15 is used. This does not affect the validity of the MEA though.

[32] One possible way to implement this is to run over all boundary nodes and identify all neighbouring fluid nodes [58, 59].

$$\boldsymbol{F} = \frac{\Delta \boldsymbol{P}}{\Delta t}. \tag{5.68}$$

Similarly, the angular momentum exchange $\Delta \boldsymbol{L}$ and therefore the total torque $\boldsymbol{T} = \Delta \boldsymbol{L}/\Delta t$ acting on the wall can be computed. To do this, we have to replace $\Delta \boldsymbol{p}(\boldsymbol{x}_i^{\mathrm{w}})$ within the sum in eq. (5.67) by $(\boldsymbol{x}_i^{\mathrm{w}} - \boldsymbol{x}^{\mathrm{ref}}) \times \Delta \boldsymbol{p}(\boldsymbol{x}_i^{\mathrm{w}})$ where $\boldsymbol{x}^{\mathrm{ref}}$ is a fixed reference point, *e.g.* the origin of the coordinate system.

Obviously the MEA is computationally much simpler than solving the integral in eq. (5.66). In fact, neither the surface stress $\boldsymbol{\sigma}^{\mathrm{w}}$ nor the normal vector $\hat{\boldsymbol{n}}$ are required in the MEA. All necessary information is already contained in the populations participating in the bounce-back process.

Momentum exchange algorithm from Chapman-Enskog expansion

NOTE (TK): I have not finished the review of this subsection yet. The reason is that there are too many inconsistencies and other problems. For example, the analysis is only valid for a stationary situation and a specific (simple) choice of equilibrium. In its present form, it causes more confusion than clarification.

While the above particle-based picture provides an intuitive understanding of the MEA working principle, a Chapman-Enskog analysis sheds additional light on it. In the remainder of this section, we focus on this rather technical point and show that the MEA effectively determines the wall shear stress with second-order accuracy.

Consider again a planar bottom wall modelled with the bounce-back method, *cf.* fig. 5.11. The net tangential momentum transferred from the fluid to the wall across the middle surface between the solid layer and the first fluid layer is, according to the momentum exchange principle, described as:

$$\Delta p_x = \frac{\Delta x}{\Delta t}\left[\left(f_8^{\mathrm{s}} - f_7^{\mathrm{s}}\right) - \left(f_5^{\mathrm{f}} - f_6^{\mathrm{f}}\right)\right] = \frac{\Delta x}{\Delta t}\left[(f_5^{\mathrm{f}} - f_6^{\mathrm{f}}) - (f_8^{\star\,\mathrm{f}} - f_7^{\star\,\mathrm{f}})\right]. \tag{5.69}$$

TODO (TK): Fix this! Notice, Δp_x is computed from post-streaming populations. Hence, LBM streaming-collide dynamics can be invoked to re-write eq. (5.69) with respect to the lattice site, *e.g.* $f_7^{\star\,\mathrm{f}} = f_7^0$.

Recalling section 5.3.1, we know that the hydrodynamic content of f_i, up to second-order in steady state, is

$$\begin{aligned} f_i &= f_i^{\mathrm{eq}} + \epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)} + O\left(\epsilon^3\right) \\ &= \left[1 - \epsilon \frac{\tau}{\Delta t} c_{i\alpha} \partial_\alpha^{(1)} + \epsilon^2 \frac{\tau}{\Delta t}\left(\frac{\tau}{\Delta t} - \frac{1}{2}\right) c_{i\alpha} c_{i\beta} \partial_\alpha^{(1)} \partial_\beta^{(1)}\right] f_i^{\mathrm{eq}} + O\left(\epsilon^3\right). \end{aligned} \tag{5.70}$$

Finally, by substituting the Chapman-Enskog solution eq. (5.70) into eq. (5.69), adopting the linear equilibrium (*cf.* section 4.3.1) for f_i^{eq} and dropping superscripts for convenience, we get

$$\Delta p_x = -c_s^2 \left(\frac{\tau}{\Delta t} - \frac{1}{2}\right) \rho_0 \partial_y u_x - \frac{c_s^2}{2} \left(\frac{\tau}{\Delta t} - \frac{1}{2}\right) \rho_0 \partial_y \partial_y u_x. \tag{5.71}$$

NOTE (TK): This equation cannot be correct since the different terms have different units.

At this point, it is convenient to remember that $F_x = \Delta P_x / \Delta t$ and $\int \mathrm{d}A\, \sigma^{\mathrm{w}}_{\alpha\beta} \hat{n}_\beta = \Delta x \sigma^{\mathrm{w}}_{xy}$, where the last identity considers the momentum exchange happening across an horizontal cell surface of area $A = \Delta x$. Further recalling that $\Delta P_x = V \Delta p_x$, NOTE (EMV): Wait, is that right? Isn't ΔP_x the momentum exchange with the entire surface while Δp_x is the momentum exchange for one node? with V denoting the volume of the unit cell, which for the $D2Q9$ is $V = \Delta x^2$, we have:

$$\sigma^{\mathrm{w}}_{xy} = \tfrac{\Delta x}{\Delta t} \Delta p_x = c \Delta p_x. \tag{5.72}$$

Hence, eq. (5.71) is nothing but the continuation, in first-order Taylor series expansion, of the wall shear stress $\sigma^{\mathrm{w}}_{xy} = \nu \partial_y {u_x}^{\mathrm{w}}$ from the boundary node $\boldsymbol{x}^{\mathrm{f}}$ (where we compute it) to the halfway wall location $\boldsymbol{x}^{\mathrm{w}}$ (where we evaluate it):

$$\sigma^{\mathrm{w}}_{xy} = \nu \partial_y {u_x}^{\mathrm{w}} = \nu \partial_y {u_x}^{\mathrm{f}} + \tfrac{c}{2} \nu \partial_y \partial_y {u_x}^{\mathrm{f}}$$

TODO (GS): There is a minus signal that need to be justified. Maybe wall normal. Bearing in mind that the first-order calculation of the velocity gradient is sufficient for a second-order velocity field [12], the first-order approximation of the shear stress is consistent with the second-order accuracy of the LB scheme. Remarkably, although with the bounce-back the velocity solution is viscosity-dependent at the second order (with the SRT collision operator), the force on the walls is not affected by this artifact [18]. NOTE (EMV): Again, perhaps it's because it's a mesoscopically correct approach?

Here we have presented an example for a planar wall only, but the MEA also works for non-planar geometries as discussed in section 11.2.

5.3.4 Boundary conditions in 3D

So far we have considered the application of boundary conditions in 2D domains. The natural question is: what changes in three dimensions? The purpose of this section is to summarise the major differences when moving from 2D to 3D.

The first obvious difference is the geometry. While in 2D domains the prescription of boundary conditions is limited to edges and corners, in 3D we have to handle planes, edges and corners as illustrated in fig. 5.27. Such a diversity of geometrical features evidently adds complexity to the treatment of boundary conditions in 3D.

The additional challenges in 3D are mostly related to the numerical implementation rather than the mathematical concept. Since 3D lattices typically contain more discrete velocity vectors than 2D lattices, we have to deal with more unknowns. This is summarised in tab. 5.1.

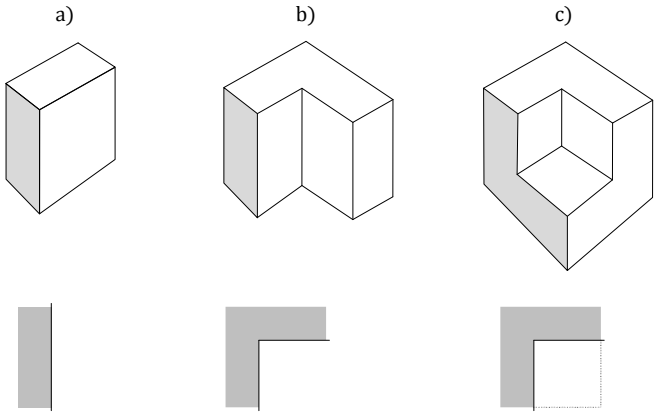


Fig. 5.27: Schematic representation of different geometrical features existing in 2D and 3D problems (inspired by [27]). Perspective views (upper) and top down views (lower) of (a) planar surface, (b) concave edge, (c) concave corner. The top down view illustrates the kind of geometries available in 2D: the plane degenerates to an edge and the other two configurations reduce to corners. NOTE (EMV): I must admit I found this picture a bit hard to parse. Maybe I was thrown off by the shading in the above images, thinking they were related to the shaded solid parts in the lower images. NOTE (TK): I agree. The figure could be clearer. It may be helpful to add axes.

Table 5.1: Number of unknown populations for different lattices and boundary configurations.

Configuration	D2Q9	D3Q15	D3Q19	D3Q27
Plane	-	5	5	9
Edge (concave)	3	8	9	15
Corner (concave)	5	10	12	17

But how does this added complexity affect the LB implementation of boundary conditions in practice? As discussed in section 5.1.4, there are link-wise and wet-node solution to treat boundary conditions. It turns out that both algorithms experience different complications when extended to 3D.

- Link-wise boundary methods specify the missing populations based on simple reflection rules, *e.g.* the bounce-back rule. Therefore, they naturally extend to 3D.
- Wet node techniques, on the other hand, are based on specific rules incorporating the consistency with bulk dynamics. Moreover, they often modify just the un-

known boundary populations, *e.g.* the NEBB method. Consequently, the added number of unknown populations makes their extension to 3D non-trivial.[33]

A difficulty particular to wet-node boundaries is the handling of those populations that never stream into the fluid domain (buried links, section 5.2.6). In 3D they appear in concave edges and corners. The number of buried populations for different lattice types is summarised in tab. 5.2. We describe in appendix A.7 a way to deal with buried links in the D3Q19 model, based on earlier work [49]. Another strategy to approach buried links can be found in [27]. Still, this problem is seldom discussed in the literature owing to the non-existence of a definitive procedure. In fact, any buried link treatment is possible providing it ensures local mass and momentum conservation, and complies with the symmetry requirements of the model (a condition necessary for consistency with equilibrium [49]).

Convex boundary nodes, on the other hand, do *not* apply the same rules as the concave case because there is only one link emanating from inside the wall. NOTE (TK): This is not correct in 3D: an edge has more than one emanating velocity vector. For corners this is more complicated as D3Q15 has no (1,1,0) and D3Q19 has no (1,1,1) velocities. As the problem has too few unknowns, one cannot enforce the no-slip condition at convex boundary nodes with general wet-node rules, and bounce-back typically has to be used [27]. As an exception, Hecht and Harting [49] developed a wet-node procedure which enables minimising slip along convex edges.

Considering the ensemble of 3D wet-node methods currently on the market, the most general and accurate one is the *local second-order boundary method* from Ginzburg and d'Humières [10]. However, its implementation is rather cumbersome, which is typical for wet-node strategies in 3D. In fact, the authors [10] later recognised this disadvantage, subsequently turning their attention to link-wise boundary methods (although non-local ones) as an alternative (*e.g.* [11, 39, 53]).

Wet-node boundary conditions may yield superior accuracy over bounce-back at flat surfaces. However, they are difficult to implement in general 3D geometries, owing to the need to distinguish between populations according to their orientation to the wall and also the cumbersome treatment of edge and corner nodes. On the

Table 5.2: Number of unknown populations belonging to *buried links* for different lattices and boundary configurations. NOTE (TK): Is this table really that important? Do we have to show it? It does not help the reader, in my opinion.

Configuration	D2Q9	D3Q15	D3Q19	D3Q27
Edge (concave)	-	4	2	6
Corner (concave)	2	6	6	10

[33] Exceptions exist though, for instance wet-node formulations that replace all populations, *e.g.* the equilibrium scheme [17], the non-equilibrium extrapolation method [6], the finite-difference velocity gradient method [14], or the regularised method [14]. As their working principle reconstructs all populations, they are algorithmically designed to be insensitive to the number of populations. The downside of these approaches is that they either decrease the accuracy or increase the complexity of implementation (*e.g.* making the scheme non-local [14]).

other hand, the application of a link-wise boundary scheme, in particular (full-way) bounce-back, does not distinguish between populations and is very easy to implement in a computer code, a feature particularly attractive to LB users.

We can conclude this section by noting that, for 3D problems, the use of link-wise boundary methods (such as bounce-back) generally offers a better compromise between accuracy and ease of implementation compared to wet-node techniques.

Finally, we emphasise that the difficulties in going from 2D to 3D become even more severe when extending the standard velocity sets to higher-order lattices. The use of lattices with a larger number of velocities is common in thermal models [], microfluidic applications with larger Knudsen number [] or multiphase problems []. So far, only few procedures have been devised to specify boundary conditions for higher-order lattices such as [] for a link-wise and [] for a wet-node implementation.

5.4 Initial conditions

In section 5.1.2 we have already explained the relevance of proper initial conditions for the Navier-Stokes equation. Here, we address how an LB simulation can be initialised. The basic question is: given a known initial divergence-free velocity field $\boldsymbol{u}_0(\boldsymbol{x})$ for the incompressible Navier-Stokes equation, how have the populations f_i to be chosen to recover a consistent initial macroscopic state? Skordos was the first who thoroughly discussed this issue in his seminal paper [20]. In the following, some general thoughts about simulation initialisation and the available LB initialisation schemes are presented, followed by the decaying Taylor-Green vortex flow as benchmark test.

5.4.1 Steady and unsteady situations

Before we discuss the available methods to initialise an LB simulation, it is worth to take a look at typical simulation scenarios. Every LB simulation falls into exactly one of the following categories:

1. Since LBM is an inherently time-dependent method, it is generally not well suited for steady problems. Although one can of course use it to obtain steady solutions, it usually takes a larger number of iteration (*i.e.* time) steps compared to methods tailored for steady problems. In steady situations, *e.g.* flow through a porous medium with a constant pressure gradient, the initial state is normally not relevant for the final outcome. The reason is that all unphysical transients caused by the initial state decay after some time, and only the desired steady

solution survives. However, there exist preconditioning techniques to accelerate convergence to steady state [60, 61].
2. The LB algorithm is particularly powerful when it is applied to unsteady problems like suspension flows, flow instabilities or fluid mixing. In many cases, one is interested in the long-time behaviour of the system or the associated statistical properties such as suspension viscosity, particle diffusivity or the mass transfer coefficients in multi-component systems. It turns out that the exact choice of the initial conditions is often not relevant since unphysical transients decay and the system "finds" its proper state after some time. In other words: the statistical long-time behaviour of such systems is usually independent of the initial state.
3. Some flows are time-periodic, for instance Womersley flow [62]. Fully converged time-periodic flows do not depend on the details of the initialisation, but in general it can take a long time until undesired transients have decayed. For Womersley flow, these transients can last for more than tens of oscillation periods.
4. There are situations where the entire fate of a simulation depends on the initial state, for example turbulence or some benchmark tests [20] such as the decaying Taylor-Green vortex flow covered in section 5.4.3. For such systems, any error in the initial state propagates in time and can detrimentally affect the accuracy of the entire simulation.

Fortunately, situations of the last kind are relatively rare. Most of the problems tackled via LB simulations belong either to the first or the second category.

5.4.2 Initial conditions in LB simulations

Role of populations and macroscopic fields

Solving the unsteady Navier-Stokes equation implies finding the velocity and the pressure fields $\boldsymbol{u}(\boldsymbol{x}, t)$ and $p(\boldsymbol{x}, t)$, respectively. This generally requires knowledge of the initial velocity and pressure fields $\boldsymbol{u}_0(\boldsymbol{x})$ and $p_0(\boldsymbol{x})$.

As it was shown in section 4.1, the populations f_i can be decomposed into the equilibrium and non-equilibrium parts. The equilibrium part, eq. (3.56), can be easily constructed from the velocity and density (pressure) fields. However, in order to initialise an LB simulation properly, one also has to specify the velocity gradient field $\boldsymbol{S}_0(\boldsymbol{x}) = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^\top)$ [63]. As we have learned before, the non-equilibrium populations are essentially proportional to the velocity gradients. Initialising populations by the equilibrium populations means that the inital state would not be second-order accurate. This is a direct consequence of the kinetic nature of the LB algorithm.

The initialisation of LB algorithms is thoroughly described in [63, 64] and, on a more mathematical basis, in [65]. In the following, we will only report the most important points and refer to the literature where necessary.

The most common situation is that the initial solenoidal (divergence-free) velocity field $\boldsymbol{u}_0(\boldsymbol{x})$ is given, but neither the pressure $p_0(\boldsymbol{x})$ nor the velocity gradients

$\boldsymbol{S}_0(\boldsymbol{x})$ are known. One could therefore be tempted to initialise the populations like

$$f_i(\boldsymbol{x}, t = 0) = f_i^{\mathrm{eq}}\left(\rho, \boldsymbol{u}_0(\boldsymbol{x})\right) \tag{5.73}$$

where ρ is some initial constant density. This will lead to an inconsistent state because the Poisson equation for the initial pressure[34],

$$\Delta p_0 = -\rho\, \partial_\beta u_{0,\alpha} \partial_\alpha u_{0,\beta}, \tag{5.74}$$

is not satisfied. In fact, Skordos [20] emphasised that an initialisation with a constant pressure is generally insufficient and Caiazzo [63] pointed out that a wrong pressure initialisation leads to undesired *initial layers* and numerical oscillations which potentially spoil the entire simulation [64]. In order to obtain a consistent initial pressure profile from the velocity, one has to solve the Poisson equation, eq. (5.74). This can be done either directly, or one may follow the approach by Mei *et al.* [64] as summarised further below.

It is worth mentioning that, for weakly compressible schemes (as most lattice Boltzmann solvers), the pressure is not entirely determined by the velocity; it is rather an independent field which requires its own initialisation [65]. However, since LB simulations are usually run in the limit of small Knudsen and Mach numbers, we assume that initialisations in accordance with the incompressible Navier-Stokes equation are also good for the slightly compressible LBM.

If we assume for a moment that the initial pressure $p_0(\boldsymbol{x})$ is known, one refines eq. (5.73) as

$$f_i(\boldsymbol{x}, t = 0) = f_i^{\mathrm{eq}}\left(\rho_0(\boldsymbol{x}), \boldsymbol{u}_0(\boldsymbol{x})\right), \tag{5.75}$$

where

$$\rho_0(\boldsymbol{x}) = \bar{\rho} + \frac{p_0(\boldsymbol{x}) - \bar{p}}{c_\mathrm{s}^2} \tag{5.76}$$

is the initial density profile with $\bar{\rho}$ being the average density and $\bar{p}$ an appropriate reference pressure. Eq. (5.76) is nothing more than the equation of state of the standard LBE. The choice of $\bar{\rho}$ is arbitrary as it is a mere scaling factor in all equations. As will be discussed more thoroughly in section 7.2.1, one usually chooses $\bar{\rho} = 1$ in lattice units.

But even if the pressure and therefore the density is set correctly, one still has to consider the non-equilibrium populations. For given velocity gradients, the BGK non-equilibrium can be approximated by [14] (see also section 4.1) TODO (TK): I have to find this equation elsewhere to relate to it properly.

$$f_i^{\mathrm{neq}}\left(\rho, \boldsymbol{S}\right) \approx -\frac{w_i \tau}{c_\mathrm{s}^2} \rho S_{\alpha\beta} Q_{i,\alpha\beta} \tag{5.77}$$

where $\boldsymbol{Q}_i = \boldsymbol{c}_i \boldsymbol{c}_i - c_\mathrm{s}^2 \boldsymbol{I}$, *i.e.* $Q_{i,\alpha\beta} = c_{i,\alpha} c_{i,\beta} - c_\mathrm{s}^2 \delta_{\alpha\beta}$.

[34] One obtains this equation by computing the divergence of the incompressible Navier-Stokes equation.

The velocity gradients $\boldsymbol{S}_0(\boldsymbol{x})$ can for example be computed from the velocity field $\boldsymbol{u}_0(\boldsymbol{x})$ analytically or through a finite difference scheme. The second approach is often employed since analytical expressions for the velocity are not always available.

Skordos [20] proposed an extended collision operator with finite-difference-approximated velocity gradients for initialisation. Holdych *et al.* [66] and van Leemput [65] reported consistent high-order initialisation schemes as generalisation of Skordos's approach. In comparsion with finite difference approximation methods, the method by Mei *et al.* [64] does not only produce a consistent initial pressure, but also a consistent initial non-equilibrium field (see details below).

Overall, if possible, the populations should be initialised according to

$$f_i(\boldsymbol{x}, t = 0) \approx f_i^{\mathrm{eq}}\left(\rho_0(\boldsymbol{x}), \boldsymbol{u}_0(\boldsymbol{x})\right) + f_i^{\mathrm{neq}}\left(\rho_0(\boldsymbol{x}), \boldsymbol{S}_0(\boldsymbol{x})\right) . \tag{5.78}$$

Before continuing, a closely related problem needs to be addressed: what comes first, propagation or collision?

Hen or egg? Order of collision and propagation

As mentioned before, in most situations a simulation is initially dominated by undesired transients until the physical solution dominates. In these cases it is not relevant whether to start a simulation with collision or propagation.

If, on the other hand, the transients are of interest, one has to start a simulation with the proper initialisation, followed by collision. The reason lies in the way the Boltzmann equation is discretised in chapter 3.

The LBE is the *explicit* discretisation of the continuous Boltzmann equation. This means that the equilibrium distribution, which is used for collision, is calculated using the known velocity and pressure fields. Only after this, propagation is performed. Initialising a simulation is basically the inverse of the momentum computation: instead of performing $f_i \rightarrow (\rho, \boldsymbol{u})$, one goes the other way around and executes $(\rho, \boldsymbol{u}) \rightarrow f_i$. Therefore, the equilibrium and non-equilibrium parts of the populations after initialisation assume a state compatible with the state after propagation, but before collision. Therefore, initialisation has to be followed by collision rather than propagation.[35]

There is a simple example showing the validity of the above argumentation. Imagine a simulation with $\tau = 1$. This means that, during collision, the non-equilibrium is set to zero everywhere and all populations relax to their equilibrium. The subsequent propagation step leads to the appearance of a new non-equilibrium if the flow field is not spatially homogeneous. In contrast, if the simulation is initialised with the correct non-equilibrium populations, a subsequent propagation will produce a new non-equilibrium which is inconsistent with a previous non-relaxed flow field. Therefore, one first has to perform collision. This example also shows that the deviatoric stress tensor (and all other moments) has to be computed after

[35] For a more accurate discussion of this issue, we refer to recent works by Dellar [67] and Schiller [68].

propagation rather than after collision. No matter how large the velocity gradients are, for $\tau = 1$ they would always be zero after collision, which is obviously not correct.

Consistent initialisation *via* a modified LB scheme

Mei *et al.* [64] proposed a modification of the incompressible LB algorithm (*cf.* section 4.3) to find a consistent initial state given a solenoidal velocity field $\boldsymbol{u}_0(\boldsymbol{x})$. The essential idea is to run the following algorithm until convergence has been obtained:

1. Initialise the populations f_i somehow, *e.g.* according to eq. (5.73).
2. Compute the local density from $\rho(\boldsymbol{x}) = \sum_i f_i(\boldsymbol{x})$.
3. Perform collision by using the modified incompressible equilibrium distributions $f_i^{\mathrm{eq}}\left(\rho(\boldsymbol{x}), \boldsymbol{u}_0(\boldsymbol{x})\right)$, *i.e.* take the updated density field from step 2 but keep the initial velocity $\boldsymbol{u}_0(\boldsymbol{x})$ rather than recomputing it.
4. Propagate.
5. Go back to step 2 and iterate until the populations f_i (and the hydrodynamical fields $\rho(\boldsymbol{x})$ and $\boldsymbol{S}(\boldsymbol{x})$) have converged to a user-defined degree. It is important to end this algorithm with propagation rather than collision; otherwise the non-equilibrium would be incorrect.

It is important to employ the incompressible LB algorithm for the consistent initialisation. The standard equilibrium leads to large initial pressure errors.

This algorithm does not obey momentum conservation (only density is conserved), it rather relaxes the velocity to its desired value at each point in space. The outcome is nearly independent of the choice of the relaxation parameter τ, although its choice affects the required number of iteration steps. It can be shown [64] that this procedure results in consistent initial populations, including the non-equilibrium part and therefore velocity gradients. For this reason, the resulting populations can be used as initial state for the actual LB simulation.

In fact, by modifying the LB algorithm as detailed above, one effectively solves the advection-diffusion equation for the density ρ as a passive scalar field and the Poisson equation for the pressure p in eq. (5.74) [64]. We will come back to this point in chapter 8. TODO (TK): Check this statement after writing the advection-diffusion chapter.

A disadvantage of this initialisation approach is that an initial force density field $\boldsymbol{f}(\boldsymbol{x})$ is not taken into account, but Caiazzo [63] presented an accelerated initialisation routine which also works in the presence of an initial force density. The major advantage is that this initialisation approach is much easier to implement than an additional Poisson solver, though depending on the parameters the solution of the advection-diffusion equation can be computationally demanding. Another advantage is that two outcomes are simultaneously available: the pressure field and consistent initial populations, including their non-equilibrium part.

We should also mention a subtle detail of the above initialisation scheme. In the original paper [64], the authors used a multi-relaxation-time LB model with $s_j = 1$, where s_j is the relaxation rate for the momentum density. Without going too much into details here, the choice $s_j = 1$ guarantees local momentum conservation so that the actual velocity field $\boldsymbol{u}(\boldsymbol{x})$ always equals its input value $\boldsymbol{u}_0(\boldsymbol{x})$ during the initialisation process. The same effect can be achieved with $\tau = 1$ in the BGK model. For s_j or τ different from unity, momentum is not conserved and the velocity as computed from the populations does *not* match the specified velocity $\boldsymbol{u}_0$. This will be seen in the example below. However, the difference between the obtained velocity field and the input velocity field $\boldsymbol{u}_0(\boldsymbol{x})$ is of second order and therefore still compatible with a consistent initialisation of the LBM.

5.4.3 Example: decaying Taylor-Green vortex flow

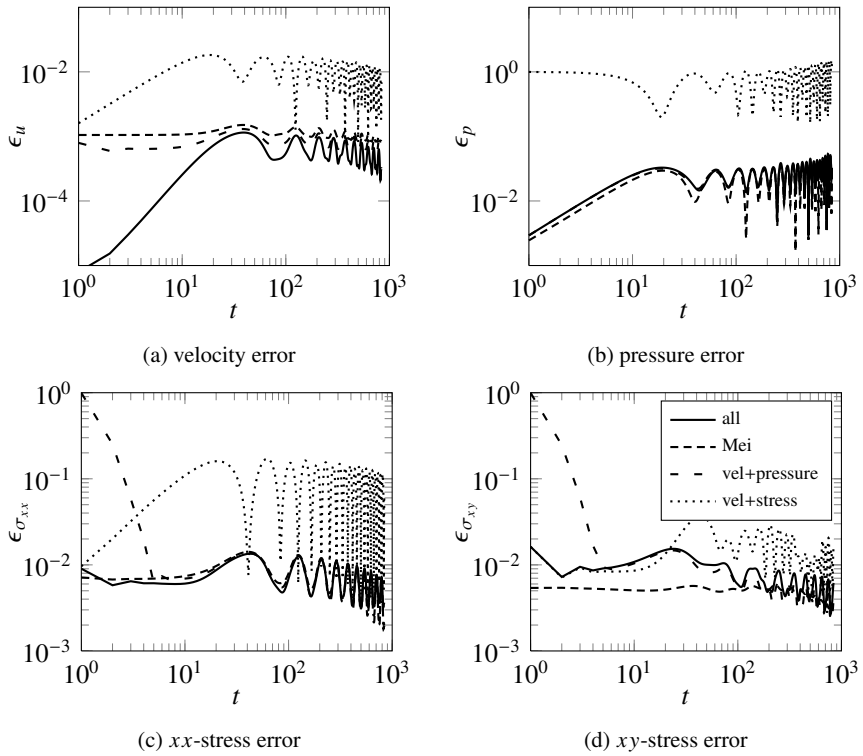
In order to show how relevant the correct initialisation of velocity, pressure and stress in a transient situation is, we simulate the decaying Taylor-Green vortex flow (*cf.* section A.3 for its definition) with several different initialisation strategies following eq. (5.78):

1. set velocity, pressure and stress analytically,
2. set velocity and pressure only (initialise with the equilibrium),
3. set velocity and stress only ($\rho_0 = 1$),
4. use the initialisation scheme by Mei *et al.* [64].

The simulation parameters are $\ell_x = 96$, $\ell_y = 72$, $\tau = 0.8$ ($\nu = 0.1$), $\hat{u}_0 = 0.03$, $\bar{\rho} = 1$ and $p_0 = 0$. The standard equilibrium is used for the actual simulations while the incompressible equilibrium is employed for Mei's initialisation scheme.[36] The vortex decay time is $t_\mathrm{d} \approx 840$. We run the simulation for one decay time and show the L_2 error (as defined in eq. (4.48)) as function of time for velocity $\boldsymbol{u}$, pressure p and the xx- and xy-components of the deviatoric stress tensor $\boldsymbol{\sigma}$. Furthermore, for Mei's initialisation scheme we use the same relaxation time τ as for the actual simulation (although in principle different relaxation times could be used), and the advection-diffusion scheme is terminated once the L_2 difference of the pressure profile at subsequent iteration steps falls below 10^{-10}. This convergence criterion is sufficient since an even lower threshold (here 10^{-12} was also tested) does not result in more accurate simulations.

The results are shown in fig. 5.28. The major observation is that an incorrect pressure is much more problematic than an incorrect stress. Furthermore, Mei's initialisation scheme produces velocity and σ_{xx} errors nearly as small as the analytical initialisation while the pressure and σ_{xy} error is slightly smaller with Mei's

[36] Using the incompressible equilibrium for the actual simulations does not result in significantly different results. This is no surprise since the incompressible model is only formally more accurate for *steady* flows.



(a) velocity error

(b) pressure error

(c) xx-stress error

(d) xy-stress error

Fig. 5.28: L_2 errors of (a) velocity, (b) pressure, (c) xx-stress and (d) xy-stress as function of time t for different initialisation schemes of the decaying Taylor-Green flow. The solid, densely dashed, loosely dashed and dotted curves denote, respectively, full initialisation, Mei's initialisation, velocity and pressure initialisation, velocity and stress initialisation. The legend in (d) applies to all subfigures.

approach. We see that the initial velocity error in the case of Mei's approach is initially not zero. The reason is that $\tau = 0.8 \neq 1$ has been used. For $\tau = 1$, the initial velocity error would be zero. Yet, the initial velocity error is of the same order as the typical velocity error and does not significantly affect the subsequent simulation results.

Disregarding the stress in the initialisation process increases the velocity error only during the first time steps. The pressure errors for initialised or non-initialised stress are virtually indistinguishable; even the stress errors behave similarly, except for very short times where they are 100%. In total, neglecting the initial stress does not lead to significant long-time deviations. The reason why the stress is much less important is that it comprises a higher-order effect in the Chapman-Enskog expansion. An initially inconsistent stress is corrected after a few time steps without significant consequences for the flow. Note that, for $\tau = 1$, a wrong (or missing) stress initialisation does not have any effect on density and momentum because the

simulation starts with collision and $\tau = 1$ leads to the total extinction of any non-equilibrium distributions. This has been confirmed by simulations (data not shown here).

The situation is different when the pressure is initially ignored. The pressure error itself does never recover from the inconsistency and oscillates in the range 10–100%. The pressure error is nearly two orders of magnitude smaller when the pressure is properly initialised. The velocity error is less susceptible: without pressure initialisation it is around 1% whereas the full initialisation leads to velocity errors below 0.1%. Both stress errors behave qualitatively differently. The error of σ_{xx} strongly depends on the pressure initialisation, while the error of σ_{xy} does not. The explanation is that σ_{xx} errors, like pressure errors, are tightly related to compressibility artifacts while σ_{xy} errors are not.

We conclude that an incorrect pressure initialisation can spoil the entire simulation while a missing stress initialisation has a nearly vanishing effect. However, if available, the stress should be included in the initialisation scheme in order to increase the accuracy and consistency of the simulation. Mei's scheme is an excellent approach to find an accurate initial state (in particular the pressure) although the velocity field is slightly incorrect for $\tau \neq 1$.

5.5 Chapter summary

TODO (GS): Has to be written.

Importance of BC in numerical methods: accuracy preservation!
It is sometimes perceived that boundaries can be readily mapped to the lattice and hence boundary conditions are simple to implement, e.g., by the bounce-back rule [5]. On closer inspection, however, it turns out that numerous difficulties arise and an efficient and at the same time accurate treatment of the boundary can be an intricate affair [101].

Question we need to address to prescribe a BC in LBM: Knowing the hydrodynamic boundary condition, can appropriate populations be found such that their specification on the boundary provides the prescribed hydrodynamic boundary condition?
Studies on this inverse problem of reproducing the LB population from a desired hydrodynamic property remain an open problem.
Rather than developing a technique that maintains a discrete particle momentum balance, the hydrodynamic approach seeks to maintain a specified velocity profile on the boundaries. During each time in he LBM procedure, the populations at each node are modified by collision, forcing, and streaming. The goal of the wet-node approach is to prescribe this process in such a fashion that the desired velocity conditions are satisfied at the end of the time step. [Noble]

We would like to point out that a second-order accuracy of the a given BC scheme in the simple flows considered does not imply their second-order for any flows.

One is encouraged to do some tests on a simplified flow of the type of flows to be simulated. [ZouHe]

Include in the end a reference to problem of setting BC in higher-order lattice!

References

1. A.J. Chorin, J.E. Marsden, *A Mathematical Introduction to Fluid Mechanics*, 3rd edn. (Springer, 2000)
2. R. Haberman, *Applied Partial Differential Equations: with Fourier Series and Boundary Value Problems* (Pearson Prentice Hall, 2004)
3. J. Anderson, *Computational Fluid Dynamics* (McGraw-Hill, 1995)
4. H.K. Versteed, M. Malalasekera, *An introduction to computational fluid dynamics, the finite volume method* (Prentice-Hall, 1996)
5. S. Chen, D. Martinez, R. Mei, Phys. Fluids **8**, 2527 (1996)
6. Z.L. Guo, C.G. Zheng, B.C. Shi, Chinese Phys. **11**, 366 (2002)
7. Z.L. Guo, C.G. Zheng, B.C. Shi, Phys. Fluids **14**, 2007 (2002)
8. M. Shankar, S. Sundar, Comput. Math. Appl. **57**, 1312 (2009)
9. X. Kang, Q. Liao, X. Zhu, Y. Yang, Appl. Thermal Eng. **30**, 1790 (2010)
10. I. Ginzbourg, D. d'Humières, J. Stat. Phys. **84**, 927 (1996)
11. I. Ginzburg, D. d'Humières, Phys. Rev. E **68**, 066614 (2003)
12. B. Chun, A.J.C. Ladd, Phys. Rev. E **75**, 066705 (2007)
13. J.C.G. Verschaeve, B. Müller, J. Comput. Phys. **229**, 6781 (2010)
14. J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Phys. Rev. E **77**(5), 056703 (2008)
15. M. Junk, Z. Yang, J. Stat. Phys. **121**, 3 (2005)
16. I. Ginzbourg, P.M. Adler, J. Phys. II France **4**(2), 191 (1994)
17. X. He, Q. Zou, L.S. Luo, M. Dembo, J. Stat. Phys. **87**(1-2), 115 (1997)
18. A.J.C. Ladd, J. Fluid Mech. **271**, 285 (1994)
19. M. Bouzidi, M. Firdaouss, P. Lallemand, Phys. Fluids **13**, 3452 (2001)
20. P.A. Skordos, Phys. Rev. E **48**(6), 4823 (1993)
21. D.R. Noble, Chen, J.G. Georgiadis, R.O. Buckius, Phys. Fluids **7**, 203 (1995)
22. T. Inamuro, M. Yoshino, F. Ogino, Phys. Fluids **7**, 2928 (1995)
23. Q. Zou, X. He, Phys. Fluids **9**, 1591 (1997)
24. I. Halliday, L.A. Hammond, C.M. Care, A. Stevens, J. Phys. A: Math. Gen. **35**, 157 (2002)
25. A.P. Hollis, I.H.H.M. Care, J. Phys. A: Math. Gen. **39**, 10589 (2006)
26. J.C.G. Verschaeve, Phys. Rev. E **80**, 036703 (2009)
27. R.S. Mayer, R.S. Bernard, D.W. Grunau, Phys. Fluids **8**, 1788 (1996)
28. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, 2001)
29. M.C. Sukop, D.T.T. Jr., *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers* (Springer, 2006)
30. S.H. Kim, H. Pitsch, Phys. Fluids **19**, 108101 (2007)
31. Q. Zou, S. Hou, S. Chen, G.D. Doolen, J. Stat. Phys. **81**, 35 (1995)
32. X. He, L.S. Luo, J. Stat. Phys. **88**, 927 (1997)
33. S.V. Patankar, C.H. Liu, E.M. Sparrow, ASME J. Heat Transfer **99**, 180 (1977)
34. J. Zhang, D.Y. Kwok, Phys. Rev. E **73**, 047702 (2006)
35. O. Gräser, A. Grimm, Phys. Rev. E **82**, 016702 (2010)
36. R. Cornubert, D. d'Humières, D. Levermore, Physica D **47**, 241 (1991)
37. D.P. Ziegler, J. Stat. Phys. **71**, 1171 (1993)
38. A.J.C. Ladd, R. Verberg, J. Stat. Phys. **104**(5-6), 1191 (2001)
39. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
40. C.K. Aidun, Y. Lu, E.J. Ding, J. Fluid Mech. **373**, 287 (1998)

41. N.Q. Nguyen, A.J.C. Ladd, Phys. Rev. E **66**(4), 046708 (2002)
42. A.J.C. Ladd, J. Fluid Mech. **271**, 311 (1994)
43. I. Ginzburg, J. Stat. Phys. **126**, 157 (2007)
44. D. d'Humières, In Rarefied Gas Dynamics: Theory and Simulations, ed. B. Shizgal, D Weaver **159**, 450 (1992)
45. P. Lallemand, L.S. Luo, Phys. Rev. E **61**(6), 6546 (2000)
46. D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, L.S. Luo, Phil. Trans. R. Soc. Lond. A **360**, 437 (2002)
47. D. d'Humières, I. Ginzburg, Comput. Math. Appl. **58**, 823 (2009)
48. A.A. Mohamad, S. Succi, Eur. Phys. J. **171**, 213 (2009)
49. M. Hecht, J. Harting, J. Stat. Mech.: Theory Exp. **P**, 01018 (2010)
50. H. Chen, Y. Qiao, C. Liu, Y. Li, B. Zhu, Y. Shi, D. Sun, K. Zhang, W. Lin, Appl. Math. Model **36**, 2031 (2012)
51. S. Izquierdo, N. Fueyo, Phys. Rev. E **78**, 046707 (2008)
52. S. Izquierdo, P. Martinez-Lera, N. Fueyo, Comput. Math. Appl. **58**, 914 (2009)
53. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 519 (2008)
54. M. Junk, Z. Yang, Phys. Rev. E **72**, 066701 (2005)
55. X. Yin, G. Le, J. Zhang, Phys. Rev. E **86**(2), 026701 (2012)
56. A.P. Hollis, I. HalIiday, C.M. Care, J. Comput. Phys. **227**, 8065 (2008)
57. T. Krüger, F. Varnik, D. Raabe, Phys. Rev. E **79**(4), 046704 (2009)
58. R. Mei, D. Yu, W. Shyy, L.S. Luo, Phys. Rev. E **65**(4), 041203 (2002)
59. D. Yu, R. Mei, L.S. Luo, W. Shyy, Prog. Aerosp. Sci. **39**, 329 (2003)
60. Z. Guo, T.S. Zhao, Y. Shi, Phys. Rev. E **70**(6), 066706 (2004)
61. L. Talon, D. Bauer, D. Gland, H. Auradou, I. Ginzburg, Water Resour. Res. , **48**, W04526 (2012)
62. A.M.M. Artoli, A.G. Hoekstra, P.M.A. Sloot, Int. J. Mod. Phys. C **14**(6), 835 (2003)
63. A. Caiazzo, J. Stat. Phys. **121**(1-2), 37 (2005)
64. R. Mei, L.S. Luo, P. Lallemand, D. d'Humières, Comput. Fluids **35**(8–9), 855 (2006)
65. P. Van Leemput, M. Rheinländer, M. Junk, Comput. Math. Appl. **58**(5), 867 (2009)
66. D.J. Holdych, D.R. Noble, J.G. Georgiadis, R.O. Buckius, J. Comput. Phys. **193**(2), 595 (2004)
67. P.J. Dellar, Comput. Math. Appl. **65**(2), 129 (2013)
68. U.D. Schiller, Comput. Phys. Commun. **185**(10), 2586 (2014)

Chapter 6
Forces

Forces play an important role in many hydrodynamic problems (section 6.1). Therefore, a proper discussion of force implementation in the lattice-Boltzmann algorithm is essential. In section 6.3 we derive an extension of the force-free lattice-Boltzmann equation to include forces. After performing a Chapman-Enskog analysis of the forcing term (section 6.4), we present benchmark tests to show the relevance of correct velocity and time discretisation schemes (section 6.7). Section **??** contains an overview of existing forcing schemes and a discussion of their differences and similarities. We also address the effect of forces on initial and boundary conditions (section 6.6). Section 6.2 summarises the most important results of this chapter and therefore acts as a reference and quick start for the implementation of a lattice-Boltzmann algorithm with forces.

TODO (TK): Further ideas for this chapter:

- We should mention somewhere that forces can be used to substitute a pressure gradient, even in complex geometries [1].

6.1 Motivation and background

Forces play a certain, if not a central role in many hydrodynamic problems. A prominent example is the gravitational acceleration $\boldsymbol{g}$ which can be cast into a force *density* $\boldsymbol{F}_g$ by multiplying it with the fluid density ρ:

$$\boldsymbol{F}_g = \rho \boldsymbol{g}. \tag{6.1}$$

In fact, in hydrodynamics we will mostly encounter force densities rather than forces since the momentum equation is actually a PDE for the momentum *density*. Forces are obtained by integrating surface stresses or bulk force densities. Mathematically, a force (density) is a momentum (density) source term, as can be seen from the Cauchy equation in eq. (1.52).

Gravity leads to a series of effects which have been successfully simulated with LBM. If two fluids with different densities are mixed or if the temperature in a fluid is non-homogeneous, density gradients in the gravitational field lead to buoyancy effects and phenomena like the *Rayleigh-Bénard* TODO (ALL): references or the *Rayleigh–Taylor instability* TODO (ALL): references. In the Rayleigh-Bénard instability, which is essential in studies of heat transfer, convection patterns develop when warmed fluid rises from a hot surface and falls after cooling. In the Rayleigh-Taylor instability, a layer of denser fluid descends as lower-density fluid below it rises. Gravitational waves at a free water surface are another example. TODO (ALL): references

Apart from gravity, there are several other cases where forces are important. Fluids in rotating reference frames are subject to radial and Coriolis forces. Charged or magnetic particles immersed in a fluid exert forces on each other, and they may also be forced by external electromagnetic fields. This is particularly important for modelling the effects of external electric fields on regions of unbalanced charges (the electrical double layer, EDL) in electrolytes near a charged solid surface or liquid-liquid interface. TODO (ALL): references

We will see in chapter 9 that forces are also commonly used to model multi-phase or multi-component flows, although a mathematical description of these phenomena is usually based on the stress tensor. Furthermore, some algorithms for fluid-structure interactions, *e.g.* the immersed boundary method, rely on forces mimicking boundary conditions. This will be discussed in section 11.4.

6.2 LBM with forces in a nutshell

This section summarises all relevant information about the implementation of forces in the LBM and how a complete time step with forces looks like. Technical details and further explanations are provided in the remainder of this chapter.

Assuming the BGK collision operator, we can write the order of operations in an LB time step including forces, also illustrated in fig. 6.1, in the following way:

1. Propagate populations.
2. Work out the force density $\boldsymbol{F}$ for the time step (*e.g.* gravity).
3. Compute the fluid density and velocity from

$$\rho = \sum_i f_i, \quad \boldsymbol{u} = \frac{1}{\rho} \sum_i f_i \boldsymbol{c}_i + \frac{\boldsymbol{F}\Delta t}{2\rho}. \tag{6.2}$$

4. Compute the equilibrium populations $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ to construct the collision operator

$$\Omega_i = -\frac{1}{\tau}\left(f_i - f_i^{\mathrm{eq}}\right). \tag{6.3}$$

5. If required, calculate the deviatoric stress:

$$\sigma_{\alpha\beta} \approx -\left(1-\frac{1}{2\tau}\right)\sum_i f_i^{\mathrm{neq}} c_{i\alpha}c_{i\beta} - \frac{1}{2}\left(1-\frac{1}{2\tau}\right)\left(F_\alpha u_\beta + u_\alpha F_\beta\right). \tag{6.4}$$

6. Compute the forcing term

$$F_i = \left(1-\frac{1}{2\tau}\right) w_i \left(\frac{F_\alpha c_{i\alpha}}{c_\mathrm{s}^2} + \frac{\left(c_{i\alpha}c_{i\beta} - c_\mathrm{s}^2\delta_{\alpha\beta}\right)F_\alpha u_\beta}{2c_\mathrm{s}^4}\right). \tag{6.5}$$

7. Apply collision and forcing to find the post-collision populations

$$f_i^\star = f_i + (\Omega_i + F_i)\Delta t \tag{6.6}$$

8. Increment time step and go back to step 1.

There are a few important remarks:

- The form of the force $\boldsymbol{F}$ depends on the underlying model. Gravity is the simplest example, but other forces, such as buoyancy, are also permitted.
- The velocity $\boldsymbol{u}$ in eq. (6.2) contains the so-called force correction. This velocity enters the equilibrium distributions and is also the macroscopic fluid velocity solving the Navier-Stokes equation. Using the *bare* velocity $\boldsymbol{u}^\star = \sum_i f_i \boldsymbol{c}_i/\rho$ would lead to first-order rather than second-order time accuracy (section 6.3.2). The velocity $\boldsymbol{u}$ can be interpreted as the average velocity during the time step, *i.e.* the average of pre- and post-collision values.
- The best place to write physical data (density ρ, velocity $\boldsymbol{u}$, stress $\boldsymbol{\sigma}$) to the hard disc is after step 5.
- The forcing scheme presented here is based on a Hermite expansion (section 6.3.1) and is the same as proposed by Guo *et al.* [2]. There are alternative ways to include forces as discussed in section **??**.

6.3 Discretisation

In chapter 3 we have shown how the LBE can be derived from the continuous Boltzmann equation in the absence of forces. Here, we will revisit that derivation, presented in section 3.3 and section 3.4, with the purpose of highlighting the required steps to include forces. The two main steps are the discretisation in velocity space (section 6.3.1) and the space-time discretisation (section 6.3.2), which will be presented next.

6.3.1 Discretisation in velocity space

Let us briefly recall the velocity space discretisation of the (force-free) Boltzmann equation as explained in section 3.3. The objective was to reduce the continuous velocity space $\boldsymbol{\xi}$ to a finite set of discrete velocities $\boldsymbol{c}_i$ while preserving the model's ability to capture the desired macroscopic physics *via* velocity moments.

A natural and systematic approach is to represent the equilibrium distribution function f^{eq} as a truncated Hermite expansion. This permits an *exact* evaluation of macroscopic quantities (*e.g.* density and velocity) through a Gauss-Hermite quadrature. This procedure led to two important results: (1) a polynomial representation of f^{eq} in velocity space (*cf.* section 3.3.5) and (2) the description of particles' motion through a discrete velocity set (*cf.* section 3.3.7). The question we aim to answer here is: *what is the equivalent polynomial representation in velocity space of the forcing term in the Boltzmann equation?*

Let us recall the continuous Boltzmann equation with a forcing term:

$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} + \frac{F_\alpha}{\rho} \frac{\partial f}{\partial \xi_\alpha} = \Omega(f). \tag{6.7}$$

Our goal is to find the discrete velocity structure of the forcing term F_α which aligns with the velocity space discretisation of f^{eq} in section 3.3.5. An evident problem is that F_α, contrarily to f^{eq}, does not appear as isolated term in eq. (6.7). Rather, to deal with F_α we have to discretise the full term $\frac{F_\alpha}{\rho}\frac{\partial f}{\partial \xi_\alpha}$.

Its discretisation in velocity space is simple if we have in mind the following two mathematical results:

1. The Hermite series expansion of the distribution function $f(\boldsymbol{\xi})$ is

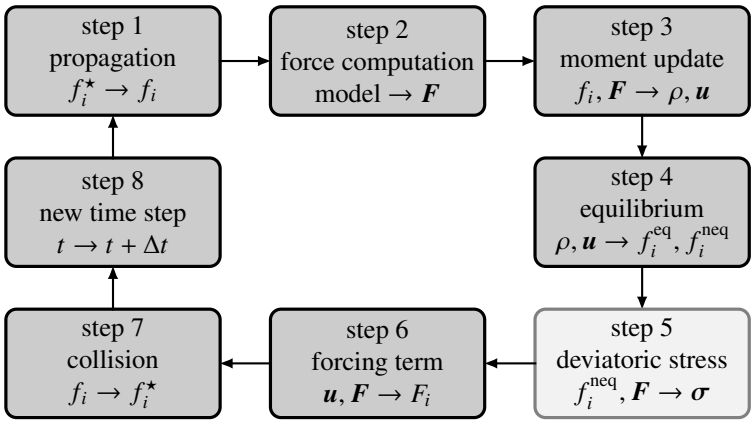


Fig. 6.1: Order of the LB algorithm in the presence of forces. Lighter boxes denote optional sub-steps which are not relevant for the evolution of the LB algorithm itself. Data can be written to the hard disc after step 5. See main text for more details.

$$f(\boldsymbol{x},\boldsymbol{\xi},t) \approx \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}^{(n)}(\boldsymbol{x},t) \cdot \boldsymbol{H}^{(n)}(\boldsymbol{\xi}). \tag{6.8}$$

2. The derivative property of Hermite polynomials reads

$$\omega(\boldsymbol{\xi}) \boldsymbol{H}^{(n)} = (-1)^n \nabla_{\boldsymbol{\xi}}^n \omega(\boldsymbol{\xi}). \tag{6.9}$$

With their help we can rewrite the Hermite expansion of $f(\boldsymbol{\xi}_i)$ as follows:

$$f \approx \sum_{n=0}^{N} \frac{(-1)^n}{n!} \boldsymbol{a}^{(n)} \cdot \nabla_{\boldsymbol{\xi}}^n \omega. \tag{6.10}$$

This representation allows us to simplify the forcing contribution in eq. (6.7) as follows:

$$\begin{aligned} \frac{\boldsymbol{F}}{\rho} \cdot \nabla_{\boldsymbol{\xi}} f &\approx \frac{\boldsymbol{F}}{\rho} \cdot \sum_{n=0}^{N} \frac{(-1)^n}{n!} \boldsymbol{a}^{(n)} \cdot \nabla_{\boldsymbol{\xi}}^{n+1} \omega \\ &\approx -\frac{\boldsymbol{F}}{\rho} \cdot \omega \sum_{n=1}^{N} \frac{1}{n!} n \boldsymbol{a}^{(n-1)} \cdot \boldsymbol{H}^{(n)}. \end{aligned} \tag{6.11}$$

The discretisation in velocity space can now be performed directly, by replacing the continuous $\boldsymbol{\xi}$ by a discrete set of $\boldsymbol{c}_i$. In this task, we rescale the velocities according to $\boldsymbol{c}_i = \boldsymbol{\xi}_i / \sqrt{3}$ and then renormalise the result by the lattice weights w_i (*cf.* section 3.3.5). Recalling section 3.3.5, this is similar to what was done in the construction of f^{eq} in the LB equation. Based on this procedure, the discrete form of the forcing term gets defined as follows:

$$F_i(\boldsymbol{x},t) = -\frac{w_i}{\omega(\boldsymbol{\xi})} \frac{\boldsymbol{F}}{\rho} \cdot \nabla_{\boldsymbol{\xi}} f \Big|_{\boldsymbol{\xi} \to \sqrt{3}\boldsymbol{c}_i}. \tag{6.12}$$

This way, the discrete velocity Boltzmann equation, equivalent to eq. (3.60), with a forcing term is written as

$$\partial_t f_i + c_{i\alpha} \partial_\alpha f_i = \Omega_i + F_i, \quad i = 0, \ldots, q-1. \tag{6.13}$$

The **truncation of the forcing term** up to second velocity order ($N = 2$), which follows the standard f^{eq} expansion, reads

$$F_i = w_i \left(\frac{F_\alpha c_{i\alpha}}{c_{\mathrm{s}}^2} + \frac{\left(c_{i\alpha} c_{i\beta} - c_{\mathrm{s}}^2 \delta_{\alpha\beta} \right) F_\alpha u_\beta}{2 c_{\mathrm{s}}^4} \right). \tag{6.14}$$

Its first three velocity moments are

$$\sum_i F_i = 0, \tag{6.15a}$$

$$\sum_i F_i c_{i\alpha} = F_\alpha, \tag{6.15b}$$

$$\sum_i F_i c_{i\alpha} c_{i\beta} = F_\alpha u_\beta + u_\alpha F_\beta. \tag{6.15c}$$

Exercise 6.1. Write down the explicit form of the forcing term F_i in eq. (6.14) for the velocity sets D1Q3 (*cf.* tab. 3.2) and D2Q9 (*cf.* tab. 3.3). Compare the results obtained for F_i with the structure of f_i^{eq} expressed by eq. (3.67) and eq. (3.68), respectively.

Physical interpretation of the force velocity moments

The zeroth-order moment, eq. (6.15a), denotes a mass source, which is zero in the present situation. The first-order moment, eq. (6.15b), is a momentum source, which appears as a body force in the Navier-Stokes equation. Finally, the second-order moment, eq. (6.15c), is an energy source which describes the power flux exerted by the body force on the fluid.

The role of the second-order moment eq. (6.15c) is subtle. Its appearance, at first glance, may seem surprising as LBM is typically built upon an isothermal assumption. However, the (weakly) compressible regime reproduced by LBM with standard equilibrium still preserves a (weak) link to energy transport, although an isothermal one [3]. The purpose of eq. (6.15c) is simply to remove the undesirable footprint left by this connection on the momentum equations. Otherwise, a spurious term, given by $F_\alpha u_\beta + u_\alpha F_\beta$, is recovered at the viscous stress level [2] TODO (GS): more references, *e.g.* Ladd, Luo. This error source is explained in more depth in section 6.4 TODO (TK): cross-reference.

On the other hand, energy transport is totally decoupled from the momentum equation in the incompressible regime. Given that for steady problems the LBM with the incompressible equilibrium is capable of reproducing the true incompressible Navier-Stokes equation, one should fulfil the condition: $\sum_i F_i c_{i\alpha} c_{i\beta} = 0$. According to the force discretisation process we have shown here, this is equivalent to saying F_i should be expanded only up to first-order terms in velocity space:

$$F_i = w_i \frac{F_\alpha c_{i\alpha}}{c_s^2}. \tag{6.16}$$

This duality in the expansion order of F_i in velocity space is explained in further details in []. TODO (GS): reference

6.3.2 Discretisation in space and time

The space and time discretisation of the force-free Boltzmann equation was discussed in section 3.4. The idea of this procedure was to replace the continuous space and time derivatives in the discrete velocity Boltzmann equation by equivalent difference operators, taken at discrete space and time steps (Δx and Δt). In the standard LB method these discretisation steps are linked with the velocity space discretisation to ensure that populations f_i, travelling with discrete velocities $\boldsymbol{c}_i$, always reach neighbouring lattice nodes within one time step Δt.

In the presence of forces, the space-time discretisation of the discrete velocity Boltzmann equation, eq. (6.13), seeks for a similar result. The task is simple and consists of two parts:

1. Advection, the left-hand side of eq. (6.13), is identical to the force-free case (*cf.* section 3.4). By applying the method of characteristics, *i.e.* defining $f_i = f_i(\boldsymbol{x}(\zeta), t(\zeta))$, where ζ parametrises a trajectory in space and time, the propagation step is obtained *exactly*, without any approximation:

$$\int_t^{t+\Delta t} \frac{\mathrm{d} f_i}{\mathrm{d}\zeta}\,\mathrm{d}\zeta = f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t). \tag{6.17}$$

2. The only approximation appears in the treatment of the right-hand side of eq. (6.13), *i.e.* the collision, which now includes the forcing term F_i:

$$\int_t^{t+\Delta t} (\Omega_i + F_i)\,\mathrm{d}\zeta. \tag{6.18}$$

   We can evaluated the integral in eq. (6.18) in different ways, *e.g.* []. As described in section 3.4, two numerical approximations are typically used.

First-order integration

The least accurate procedure employs a rectangular discretisation. Here, the integral of collision and forcing terms are approximated by just one point:

$$\int_t^{t+\Delta t} (\Omega_i + F_i)\,\mathrm{d}\zeta = \left[\Omega_i(\boldsymbol{x}, t) + F_i(\boldsymbol{x}, t)\right]\Delta t + O(\Delta t^2). \tag{6.19}$$

Using this first-order approximation and the BGK-type collision operator, $\Omega_i = -(f_i - f_i^{\mathrm{eq}})/\tau$, the LB equation with a force takes the expected form, where all terms on the right-hand-side are evaluated at $(\boldsymbol{x}, t)$.

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\tau}\left(f_i - f_i^{\mathrm{eq}}\right) + F_i\Delta t, \tag{6.20}$$

Aside from the inclusion of $F_i\Delta t$ in eq. (6.20), everything else remains identical to the unforced case discussed in chapter 3.

While fully explicit, this way of formulating the LB scheme is only first-order accurate. In the absence of forces, such a lower precision is not harmful as we can still obtain second-order accuracy at macroscopic level providing the $\Delta t/2$ shift is considered in the viscosity-relaxation relation [4], *i.e.* $\nu = c_s^2(\tau - \frac{\Delta t}{2})$ instead of $\nu = c_s^2\tau$. The reason for such an accuracy improvement is because both the 'physical' viscous term and its leading-order error have the same functional form. This enables the latter to be absorbed as a 'physical' contribution by redefining the viscosity. TODO (TK): cross-reference

However, in the force discretisation the same kind of 'trick' does not apply. Hence, a lower-order accuracy inevitably leads to macroscopic solutions corrupted by discrete lattice artifacts [5, 2]. Their form is shown in section 6.4, and their quantitative effect is illustrated in section 6.7. A solution to eliminate these undesired discretisation artifacts in the forcing form consists of employing a second-order space-time discretisation of the collision and forcing terms.

Second-order integration

The trapezoidal discretisation is more accurate:

$$\int_t^{t+\Delta t} (\Omega_i + F_i)\,\mathrm{d}\zeta = \left(\frac{\Omega_i(\boldsymbol{x},t) + \Omega_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)}{2} + \frac{F_i(\boldsymbol{x},t) + F_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)}{2}\right)\Delta t + O(\Delta t^3). \tag{6.21}$$

However, we obtain second-order accuracy at the expense of a time-implicit scheme. Fortunately, this does not constitute a problem since, as explained in section 3.4, the explicit form can be recovered by introducing a smart change of variables []. With the forcing term the new variable $\bar{f}_i$ is defined as follows:

$$\bar{f}_i = f_i - \frac{(\Omega_i + F_i)\Delta t}{2}. \tag{6.22}$$

Using eq. (6.22) and some simple algebra, the LB equation solving for $\bar{f}_i$ takes the familiar form:

$$\bar{f}_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) - \bar{f}_i(\boldsymbol{x},t) = \left[\Omega_i(\boldsymbol{x},t) + F_i(\boldsymbol{x},t)\right]\Delta t. \tag{6.23}$$

Considering the BGK-type collision operator, $\Omega_i = -(f_i - f_i^{\mathrm{eq}})/\tau$, then eq. (6.23) simplifies to:

$$\bar{f}_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) - \bar{f}_i(\boldsymbol{x},t) = -\frac{2\Delta t}{2\tau + \Delta t}\left(\bar{f}_i - f_i^{\mathrm{eq}} - \tau F_i\right) \tag{6.24}$$

where, once again, all terms on the right-hand-side of eq. (6.24) are given at $(\boldsymbol{x}, t)$. The extension to other collision operators is straightforward (section 10.5).

The **second-order accurate discretisation of the LBGK equation with forcing term** reads

$$\bar{f}_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - \bar{f}_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\bar{\tau}}\left(\bar{f}_i - f_i^{\mathrm{eq}}\right) + \left(1 - \frac{\Delta t}{2\bar{\tau}}\right) F_i \Delta t \tag{6.25}$$

with a redefined relaxation parameter $\bar{\tau} = \tau + \Delta t/2$. Based on the new variable $\bar{f}_i$, the leading macroscopic moments are

$$\rho = \sum_i \bar{f}_i + \frac{\Delta t}{2}\sum_i F_i, \tag{6.26a}$$

$$\rho\boldsymbol{u} = \sum_i \bar{f}_i \boldsymbol{c}_i + \frac{\Delta t}{2}\sum_i F_i \boldsymbol{c}_{i\alpha}, \tag{6.26b}$$

$$\boldsymbol{\Pi} = \left(1 - \frac{\Delta t}{2\bar{\tau}}\right)\sum_i \bar{f}_i \boldsymbol{c}_i\boldsymbol{c}_i + \frac{\Delta t}{2\bar{\tau}}\sum_i f_i^{\mathrm{eq}}\boldsymbol{c}_i\boldsymbol{c}_i + \frac{\Delta t}{2\bar{\tau}}\left(1 - \frac{\Delta t}{2\bar{\tau}}\right)\sum_i F_i\boldsymbol{c}_i\boldsymbol{c}_i. \tag{6.26c}$$

In most cases, the notation of the redefined variables is dropped for convenience and f_i and τ are written instead of $\bar{f}_i$ and $\bar{\tau}$. The equilibrium populations f_i^{eq} have the same functional form as before. However, the velocity entering $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ is now given by eq. (6.26b). (And in situations with non-zero mass sources the density entering $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ must be also redefined according to eq. (6.26a).)

The redefinition of the velocity in eq. (6.26b) can be interpret as the averaging of velocity before and after forcing. The use of this point of view, in alternative to the half-force correction in eq. (6.26b), has been adopted in some LB works []. [vanderSman,Walsh] An unified framework on the inclusion of the force term can be found in [] [GinzburgTRT2008]

A potential difficulty in dealing with eq. (6.26b) happens when $\boldsymbol{F}$ depends on $\boldsymbol{u}$. This relationship is identified in a number of cases, such as Brinkman models [] or Coriolis forces [], and it leads to an implicit form of eq. (6.26b) in $\boldsymbol{u}$. For linear relations, $\boldsymbol{F} \propto \boldsymbol{u}$, and other analytically invertible dependencies we can easily solve eq. (6.26b) for $\boldsymbol{u}$ []. In more general cases, however, a numerical routine may be required every time step to determine $\boldsymbol{u}$. TODO (GS): references

6.4 Chapman-Enskog analysis with forces

The Chapman-Enskog analysis (section 4.1) reveals the consistency of the LBM and the Navier-Stokes equations. We will now extend the Chapman-Enskog analysis to situations with forces.

Historically, the use of the Chapman-Enskog analysis applied to the forced LB model was pioneered in [6, 7]. Later, a number of authors [8, 9, 10, 11, 12] extended its formulation to include second-order terms, as given by eq. (6.14). A subsequent improvement [5, 2] showed the necessity of correcting discrete lattice effects. These effects were shown to be corrected in an *a priori* fashion through a systematic second-order discretisation of the LB equation (section 6.3.2) [13, 14, 4]. Even today, the study of a "clean" inclusion of forces in the LB equation remains an active topic of research, *e.g.* [15, 16, 17, 18, 19, 20, 21, 22, 23]. TODO (GS): Please break this list down into smaller chunks.

The Chapman-Enskog analysis of the forced LB equation evolves along the same lines as the force-free case in section 4.1. The difference is that now we are working with eq. (6.25) as evolution equation, together with eq. (6.26) for the velocity moments. Hence, the first question we need to address is at which expansion order we have to include the forcing term F_i.[1]

In order to be consistent with the remaining terms in the LB equation, the forcing term must scale as $F_i = O(\epsilon)$ [5] so that $F_i = \epsilon F_i^{(1)}$, at least.

Considering the sufficiency of $F_i = \epsilon F_i^{(1)}$, a valid assumption for most of hydrodynamic problems, the application of the steps described in section 4.1 to eq. (6.25) lead to the hierarchy of ϵ-perturbed equations, similar to eq. (4.8a) and eq. (4.8b), featuring the first-order expanded force term:

$$O(\epsilon): \qquad \left(\partial_t^{(1)} + c_{i\alpha}\partial_\alpha^{(1)}\right) f_i^{\mathrm{eq}} - \left(1 - \frac{\Delta t}{2\tau}\right) F_i^{(1)} = -\frac{1}{\tau} f_i^{(1)}, \quad (6.27a)$$

$$O(\epsilon^2): \quad \partial_t^{(2)} f_i^{\mathrm{eq}} + \left(\partial_t^{(1)} + c_{i\alpha}\partial_\alpha^{(1)}\right)\left(1 - \frac{\Delta t}{2\tau}\right)\left(f_i^{(1)} + \frac{\Delta t}{2} F_i^{(1)}\right) = -\frac{1}{\tau} f_i^{(2)}. \quad (6.27b)$$

In the presence of an external force, the hydrodynamic moments are no longer conserved. This leads to a redefinition of the solvability conditions for mass and momentum:

[1] Note that, in certain problems, the forcing term may require an higher-order expansion. For example, in formulating axisymmetric hydrodynamics via added LB source terms [], the formal expansion of the forcing term must extend up to the second order: $F_i = \epsilon F_i^{(1)} + \epsilon^2 F_i^{(2)}$. TODO (GS): reference

$$\sum_i f_i^{\text{neq}} = -\frac{\Delta t}{2} \sum_i F_i^{(1)}, \tag{6.28a}$$

$$\sum_i \boldsymbol{c}_i f_i^{\text{neq}} = -\frac{\Delta t}{2} \sum_i \boldsymbol{c}_i F_i^{(1)}. \tag{6.28b}$$

Likewise, the extension to "strengthened" order-by-order solvability conditions reads

$$\sum_i f_i^{(1)} = -\frac{\Delta t}{2} \sum_i F_i^{(1)} \qquad \text{and} \qquad \sum_i f_i^{(\text{k})} = 0, \tag{6.29a}$$

$$\sum_i \boldsymbol{c}_i f_i^{(1)} = -\frac{\Delta t}{2} \sum_i \boldsymbol{c}_i F_i^{(1)} \qquad \text{and} \qquad \sum_i \boldsymbol{c}_i f_i^{(\text{k})} = 0 \tag{6.29b}$$

with $k \geq 2$.

In order to proceed to the macroscopic level the form of F_i must be known. Therefore, we recall section 6.3.1 and consider the forcing term F_i given by eq. (6.14) whose moments are given by eq. (6.15). Such a forcing implies the absence of mass sources, *i.e.* the right-hand side in eq. (6.28a) and eq. (6.29a) is set to zero.

Next, by taking the zeroth and first moments of eq. (6.27a), we obtain at $O(\epsilon)$:

$$\partial_t^{(1)} \rho + \partial_\gamma^{(1)} (\rho u_\gamma) = 0, \tag{6.30a}$$

$$\partial_t^{(1)} (\rho u_\alpha) + \partial_\beta^{(1)} \Pi_{\alpha\beta}^{\text{eq}} = F_\alpha. \tag{6.30b}$$

Here, $\Pi_{\alpha\beta}^{\text{eq}} = \sum_i c_{i\alpha} c_{i\beta} f_i^{\text{eq}} = \rho u_\alpha u_\beta + \rho c_s^2 \delta_{\alpha\beta}$, according to eq. (4.10a). Similarly, by taking the zeroth and first moments of eq. (6.27b), the corresponding equations at $O(\epsilon^2)$ are

$$\partial_t^{(2)} \rho = 0, \tag{6.31a}$$

$$\partial_t^{(2)} (\rho u_\alpha) + \partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau}\right) \Pi_{\alpha\beta}^{(1)} = 0. \tag{6.31b}$$

By merging the mass and momentum equations in eq. (6.30) and eq. (6.31), respectively, we obtain

$$\left(\epsilon \partial_t^{(1)} + \epsilon^2 \partial_t^{(2)}\right) \rho + \epsilon \partial_\gamma^{(1)} (\rho u_\gamma) = 0, \tag{6.32a}$$

$$\left(\epsilon \partial_t^{(1)} + \epsilon^2 \partial_t^{(2)}\right) (\rho u_\alpha) + \epsilon \partial_\beta^{(1)} \Pi_{\alpha\beta}^{\text{eq}} = \epsilon F_\alpha^{(1)} - \epsilon^2 \partial_\beta^{(1)} \left(1 - \frac{\Delta t}{2\tau}\right) \Pi_{\alpha\beta}^{(1)}. \tag{6.32b}$$

The closure of the moment system in eq. (6.32) requires expressing $\Pi_{\alpha\beta}^{(1)}$ in terms of known quantities. This can be achieved by taking the second moment of eq. (6.27a):

$$\partial_t^{(1)} \Pi_{\alpha\beta}^{\text{eq}} + \partial_\gamma^{(1)} \Pi_{\alpha\beta\gamma}^{\text{eq}} - \left(1 - \frac{\Delta t}{2\tau}\right) \sum_i F_i^{(1)} c_{i\alpha} c_{i\beta} = -\frac{1}{\tau} \Pi_{\alpha\beta}^{(1)}, \tag{6.33}$$

where we have used the identity:

$$\Pi^{(1)}_{\alpha\beta} = \sum_i f_i^{(1)} c_{i\alpha} c_{i\beta} + \frac{\Delta t}{2} \sum_i F_i^{(1)} c_{i\alpha} c_{i\beta}, \tag{6.34}$$

which can be deduced by applying the Chapman-Enskog decomposition to eq. (6.26c).

Since $\Pi^{(1)}_{\alpha\beta}$ is the contribution responsible for the viscous stress at macroscopic level, the role of $\sum_i F_i^{(1)} c_{i\alpha} c_{i\beta}$ is clear: it aims at removing spurious forcing terms possibly appearing in $\Pi^{(1)}_{\alpha\beta}$ so that its form, given below, is kept identical to the unforced case (given by eq. (4.14))

$$\Pi^{(1)}_{\alpha\beta} = -\rho c_s^2 \tau \left(\partial^{(1)}_\beta u_\alpha + \partial^{(1)}_\alpha u_\beta \right) + O(u^3). \tag{6.35}$$

Based on eq. (6.35) we confirm that the computation of the viscous stress $\sigma_{\alpha\beta}$ remains given by $\sigma_{\alpha\beta} = -\left(1 - \frac{\Delta t}{2\tau}\right) \Pi^{(1)}_{\alpha\beta}$, as deduced for the force-free case (eq. (4.13)).

Finally, the closure of eq. (6.32b) is possible. By re-assembling $\partial_t = \epsilon \partial_t^{(1)} + \epsilon^2 \partial_t^{(2)}$ and using the content of $\Pi^{\text{eq}}_{\alpha\beta}$ and $\Pi^{(1)}_{\alpha\beta}$, we obtain from eq. (6.32) the correct form (up to $O(u^3)$ error terms) of the unsteady (compressible) Navier-Stokes equation with a forcing term:

$$\partial_t \rho + \partial_\gamma (\rho u_\gamma) = 0, \tag{6.36a}$$

$$\partial_t (\rho u_\alpha) + \partial_\beta \left(\rho u_\alpha u_\beta + \rho c_s^2 \delta_{\alpha\beta} \right) = \partial_\beta \left[\eta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta \right) \right] + F_\alpha. \tag{6.36b}$$

The dynamic viscosity is given by $\eta = \rho c_s^2 (\tau - \frac{\Delta t}{2})$ as usual (section 4.1).

6.5 Errors due to an incorrect LB force modelling

Now that we know how to perform the Chapman-Enskog analysis in the LB model with forces (section 6.4), we can evaluate whether any given forcing scheme brings errors to the recovered hydrodynamic model. According to section 6.3, the formulation of the LB force model comprises two steps: (1) the velocity space discretisation and (2) the space and time discretisation. Each of these discretisation steps comes with different error sources in case we do not deal with them properly. Our goal is to point out the form of these errors.

Discretisation of velocity space: the issue of unsteady and steady cases

The impact of the incorrect velocity space discretisation of the forcing term can be recognised by distinguishing between unsteady and steady phenomena.

In *unsteady* state, the term $\partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}}$ contains the spurious contribution $F_\alpha u_\beta + u_\alpha F_\beta$, see exercise 6.2 below. And this error term can be exactly cancelled by $\sum_i F_i c_{i\alpha} c_{i\beta}$ if the force term F_i is expanded up to the *second velocity order* as given in eq. (6.14) [11, 2]. With this velocity discretisation in the LB force model we can correctly recover the unsteady Navier-Stokes equation with a forcing term, eq. (6.36).

Exercise 6.2. Show that

$$\begin{aligned} \partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} &= \partial_t^{(1)} \left(\rho u_\alpha u_\beta + \rho c_{\mathrm{s}}^2 \delta_{\alpha\beta} \right) \\ &= -\partial_\gamma^{(1)} \left(\rho u_\alpha u_\beta u_\gamma \right) - c_{\mathrm{s}}^2 \left(u_\alpha \partial_\beta^{(1)} \rho + u_\beta \partial_\alpha^{(1)} \rho \right) \\ &\quad - c_{\mathrm{s}}^2 \delta_{\alpha\beta} \partial_\gamma^{(1)} (\rho u_\gamma) + u_\alpha F_\beta^{(1)} + F_\alpha^{(1)} u_\beta . \end{aligned} \tag{6.37}$$

Hint: apply the procedure in appendix A.2.2 including a forcing term.

NOTE (TK): I am not perfectly happy with the first sentence.NOTE (GS): Is it because of the word 'immaterial'?

On the other hand, in *steady* state the term $\partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}}$ is immaterial, recall discussion in section 4.2.3. Hence, we could expect that the spurious contribution $F_\alpha u_\beta + u_\alpha F_\beta$ no longer exists. This is not absolutely true, though. When using the standard LB equilibrium, an identical term is retrieved due to the requirement that the shear stress depends on the gradients of the velocity $\boldsymbol{u}$ rather than on the gradients of the momentum $\rho \boldsymbol{u}$. Consequently, to cancel that term, $\sum_i F_i c_{i\alpha} c_{i\beta}$ is still required as a correction. On the other hand, with the incompressible LB equilibrium, the steady incompressible NSE is recovered with no spurious terms as discussed in section 4.3. Hence, unlike previous cases, here we must set $\sum_i F_i c_{i\alpha} c_{i\beta} = 0$, meaning F_i must be expanded only to the first velocity order. A second-order expansion of F_i would lead to an incorrect steady incompressible Navier-Stokes equation affected by the divergence of $F_\alpha u_\beta + u_\alpha F_\beta$. In section 6.7 we illustrate this result through numerical examples. A more detailed explanation of this subtle point can be found in [20, 21].

Discretisation of space and time: the issue of discrete lattice effects

The impact of the inaccurate space and time discretisation of the LB forcing term can be understood by repeating the Chapman-Enskog analysis presented before, but this time for the LB model with a first-order time integration scheme, *cf.* section 6.3.1. Let us assume a time-dependent process, where the forcing term uses a second-order velocity discretisation, eq. (6.14). It can be shown, *e.g.* [5, 2], that the macroscopic flow equations reproduced in this case will have the following incorrect form:

$$\begin{aligned}\partial_t \rho + \partial_\gamma(\rho u_\gamma) &= -\frac{\Delta t}{2}\partial_\gamma F_\gamma,\\ \partial_t(\rho u_\alpha) + \partial_\beta\left(\rho u_\alpha u_\beta + \rho c_s^2 \delta_{\alpha\beta}\right) &= \partial_\beta\left[\eta\left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right)\right] + F_\alpha \\ &\quad - \frac{\Delta t}{2}\left[\partial_t F_\alpha + \partial_\beta\left(u_\alpha F_\beta + F_\alpha u_\beta\right)\right].\end{aligned} \tag{6.38}$$

The difference between eq. (6.38) and the 'true' NSEs with a force, eq. (6.36), lies in the added $O(\Delta t)$ error terms [5, 2]. They are called *discrete lattice artifacts* because, as acting upon the same scale of the viscous term, $\eta \sim O(\Delta t)$, they corrupt the macroscopic equations below $O(\Delta t^2)$, the leading truncation error in the LB model. These discrete artifacts, caused by the incorrect space and time discretisation, lead to *inconsistencies* in both mass and momentum macroscopic equations, making them more problematical than the errors caused by the incorrect velocity discretisation, which just corrupt momentum.

6.6 Boundary and initial conditions with forces

So far the discussion about forces has been limited to the bulk solution. In chapter 5 we have already discussed the topic of initial and boundary conditions, but without including the effect of forces. The purpose of this section is therefore to point out the required modifications of initial (section 6.6.1) and boundary conditions (section 6.6.2) in the presence of forces.

6.6.1 Initial conditions

Initial conditions are necessary for time-dependent problems. But even steady flows must be subject to a proper initialisation. Otherwise, initial errors may be conserved during the simulation and contaminate the steady-state solution. In section 5.4 we discussed two ways of initialising LB simulations. Let us revisit them for problems with forces and work out the necessary modifications.

The simplest strategy consists of prescribing the *equilibrium* state to the initial populations, *i.e.* $f_i(\boldsymbol{x}, t=0) = f_i^{\mathrm{eq}}\left(\rho_0(\boldsymbol{x}), \boldsymbol{u}_0(\boldsymbol{x})\right)$, where ρ_0 and $\boldsymbol{u}_0$ refer to known initial density and velocity fields. Recall from section 6.3.2 that for problems with forces the macroscopic velocity is computed from $\rho\boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i + \frac{\Delta t}{2}\boldsymbol{F}$. Therefore, to set an initial velocity $\boldsymbol{u}_0$ consistent with the force field, the initial populations must be set to [] TODO (GS): reference

$$f_i(\boldsymbol{x}, t=0) = f_i^{\mathrm{eq}}\left(\rho_0(\boldsymbol{x}), \boldsymbol{u}_0^\star(\boldsymbol{x})\right), \quad \boldsymbol{u}_0^\star = \boldsymbol{u}_0 - \frac{\boldsymbol{F}\Delta t}{2\rho_0}. \tag{6.39}$$

Obviously, for low-order forcing schemes, where the macroscopic velocity is computed from $\rho \boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i$ without considering the half-forcing correction, the equilibrium initialisation is constructed similarly to the unforced case, *i.e.* $\boldsymbol{u}_0^\star = \boldsymbol{u}_0$.

Note that this procedure is strictly only correct for forcing schemes for which the equilibrium velocity equals $\boldsymbol{u}$, *i.e.* $A = \frac{1}{2}$ in eq. (**??**), such as Guo forcing. NOTE (GS): I don't find the above statement necessary. It basically repeats the paragraph before eq. (6.39).

As discussed in section 5.4, a more accurate initialisation consists of adding the non-equilibrium populations f_i^{neq} to f_i^{eq}. Given that the leading order of f_i^{neq}, *i.e.* $f_i^{(1)}$, depends on $\boldsymbol{F}$, *cf.* eq. (6.27a), the non-equilibrium term added to eq. (6.39) shall be redefined as follows [] TODO (GS): reference

$$f_i^{\mathrm{neq}} \approx -\frac{w_i \tau}{c_\mathrm{s}^2} \rho Q_{i\alpha\beta} S_{\alpha\beta} - \frac{w_i \Delta t}{2 c_\mathrm{s}^2} \left(c_{i\alpha} F_\alpha + \frac{Q_{i\alpha\beta}}{2 c_\mathrm{s}^2} \left(u_\alpha F_\beta + F_\alpha u_\beta \right) \right), \tag{6.40}$$

where $Q_{i\alpha\beta} = c_{i\alpha} c_{i\beta} - c_\mathrm{s}^2 \delta_{\alpha\beta}$ and $\boldsymbol{S}$ is the velocity gradient tensor. TODO (GS): Please verify!

NOTE (TK): I have removed the comment about higher orders. I think it is too advanced.

6.6.2 Boundary conditions

Both link-wise and wet-node boundary conditions (*cf.* section 5.1.4) are affected differently by the presence of forces. Next, we discuss the consequences of each case.

Bounce-back

Although the working principle of the bounce-back rule is not changed by the inclusion of forces, its accuracy does depend on the force implementation. If we do not work with the second-order space-time discretisation of the LB equation, the macroscopic laws established by the bounce-back rule will be affected by discrete lattice artifacts of the force. Let us demonstrate this issue by looking at a simple example: an hydrostatic equilibrium where a constant force (*e.g.* gravity) is balanced by a pressure gradient.

To start the example let us consider the correct approach and adopt for the bulk dynamics the second-order space-time discretisation of the LB equation, given by eq. (6.25). For a time-independent process: $\partial_t f_i = 0$. Then, the Chapman-Enskog analysis up to $O(\epsilon)$ yields:[2]

[2] Eq. (6.41) results from omitting the time derivatives in eq. (6.27a).

$$f_i^{(1)} = -\tau c_{i\alpha}\partial_\alpha^{(1)} f_i^{\mathrm{eq}} + \left(\tau - \frac{\Delta t}{2}\right) F_i^{(1)}. \tag{6.41}$$

NOTE (TK): Careful: we cannot simple set the time-derivative terms equal to zero.
NOTE (GS): Why in steady-state we cannot set the time-derivative of f_i to zero?
Given that we are interested in the hydrostatic solution, *i.e.* $\boldsymbol{u} = \mathbf{0}$, the equilibrium for this problem can be reduced to $f_i^{\mathrm{eq}} = w_i\rho$ and the forcing term to $F_i = w_i \boldsymbol{c}_i \cdot \boldsymbol{F}/c_s^2$. Inserting f_i^{eq} and F_i into eq. (6.41), we have $f_i^{(1)} = -\tau w_i c_{i\alpha}\partial_\alpha^{(1)}\rho + (\tau - \Delta t/2)c_{i\alpha}F_\alpha/c_s^2$. The macroscopic behaviour of the populations for this hydrostatic problem is completely determined by $f_i = f_i^{\mathrm{eq}} + \epsilon f_i^{(1)}$, without any approximation.
TODO (GS): reference

The next step is transferring these results to the bounce-back formula applied at a resting wall, *i.e.* $f_{\bar{i}} = f_i^\star$, *cf.* eq. (5.23). This way one can describe the bounce-back rule in the form of a Chapman-Enskog decomposition as follows:

$$f_{\bar{i}}^{\mathrm{eq}} + \epsilon f_{\bar{i}}^{(1)} = f_i^{\mathrm{eq}} + \left(1 - \frac{\Delta t}{\tau}\right)\epsilon f_i^{(1)} + \left(\tau - \frac{\Delta t}{2}\right)\Delta t \epsilon F_i^{(1)}. \tag{6.42}$$

After substituting the content of f_i^{eq}, $f_i^{(1)}$ and $F_i^{(1)}$ into eq. (6.42), and undertaking some simplifications, we arrive at the hydrostatic solution established by the bounce-back rule:

$$\left(\tau - \frac{\Delta t}{2}\right)\left(c_s^2\partial_\alpha\rho - F_\alpha\right) = 0. \tag{6.43}$$

The first factor in eq. (6.43) is positive due to the stability requirement $\tau > \frac{\Delta t}{2}$ (section 4.4) and can be cancelled. Hence, we conclude that the LB equation with the bounce-back rule is *exact* for the hydrostatic pressure solution where we expect the balance $c_s^2\nabla_\alpha\rho = F_\alpha$.

The question that follows is whether the correct hydrostatic balance also holds when the LB equation with a force is formulated with a first-order space-time discretisation. Based on the bulk analysis presented in section 6.5, one may be led to conclude that nothing changes. The justification is that bulk errors have the form of force derivatives, thereby vanishing for a constant body force. However, the closure in bounce-back boundary conditions may induce discrete lattice artifacts even in the case of uniform forces. This conclusion is demonstrated in exercise 6.3. A complementary discussion on this result can be find in [24, 23] (and in [15] through a different framework).

Exercise 6.3. Repeat the Chapman-Enskog analysis for a first-order time discretisation of the LB equation. Show that the hydrostatic balance established by the bounce-back rule is then incorrectly predicted as

$$\left(\tau - \frac{\Delta t}{2}\right)\left(c_s^2\partial_\alpha\rho - F_\alpha\right) = \frac{\Delta t}{2}F_\alpha.$$

Non-equilibrium bounce-back

The fundamental principle underlying the wet-node technique is that boundary nodes follow the same LB rules as the bulk nodes. Hence, to be consistent with the bulk, the algorithm for boundary nodes needs to be reformulated to account for the presence of the force. We demonstrate this procedure in more detail by focusing on the non-equilibrium bounce-back (NEBB) method.

As we have seen in section 5.2.4, wet boundary nodes must satisfy the macroscopic laws of bulk nodes through the velocity moments. It follows that the first-order moment for the momentum is modified by the presence of a force when we use the second-order space-time discretisation in eq. (6.26b). This leads to a number of changes in the NEBB algorithm.

Consider the top wall depicted in fig. 5.21. While in the force-free case the determination of the unknown wall density is given by eq. (5.32), with a force it changes to

$$\begin{aligned} \rho &= \sum_i f_i = \underbrace{f_0 + f_1 + f_2 + f_3 + f_5 + f_6}_{\text{known}} + \underbrace{f_4 + f_7 + f_8}_{\text{unknown}}, \\ \rho u_y &= \sum_i f_i c_{iy} + \frac{\Delta t}{2} \sum_i F_i c_{iy} = \underbrace{f_2 + f_5 + f_6}_{\text{known}} - \underbrace{\left(f_4 + f_7 + f_8\right)}_{\text{unknown}} + \frac{F_y \Delta t}{2}. \end{aligned} \tag{6.44}$$

By combining these two equations we get

$$\rho = \frac{1}{1 + u_y} \left(f_0 + f_1 + f_3 + 2\left(f_2 + f_5 + f_6\right) + \frac{F_y \Delta t}{2} \right). \tag{6.45}$$

The unknown boundary populations remain determined by the bounce-back of their non-equilibrium components, *i.e.* eq. (5.38). Yet, compared to the force-free case, now it is necessary to consider both tangential and normal momentum corrections N_α. The reason for that will become clear shortly. For now, let us consider the top wall in fig. 5.21 and write the bounce-back of the non-equilibrium populations as[3]

$$\left.\begin{aligned} f_4^{\text{neq}} &= f_2^{\text{neq}} - N_y, \\ f_7^{\text{neq}} &= f_5^{\text{neq}} - N_x - N_y, \\ f_8^{\text{neq}} &= f_6^{\text{neq}} + N_x - N_y. \end{aligned}\right\} \Longrightarrow \left\{\begin{aligned} f_4 &= f_2 + \left(f_4^{\text{eq}} - f_2^{\text{eq}}\right) - N_y, \\ f_7 &= f_5 + \left(f_7^{\text{eq}} - f_5^{\text{eq}}\right) - N_x - N_y, \\ f_8 &= f_6 + \left(f_8^{\text{eq}} - f_6^{\text{eq}}\right) + N_x - N_y. \end{aligned}\right. \tag{6.46}$$

Using the known equilibrium distributions, we get

[3] The sign convention for the normal momentum correction goes in line with the tangential case, *cf.* eq. (5.39). If $\boldsymbol{n}$ and $\boldsymbol{t}$ denote the wall normal and the wall tangential vectors and if their positive sign coincides with the positive sign of the Cartesian axis, then the normal and tangential momentum corrections appear in the algorithm as $f_{\bar{i}}^{\text{neq}}(\boldsymbol{x}_B, t) = f_i^{\text{neq}}(\boldsymbol{x}_B, t) + (\boldsymbol{n} \cdot \boldsymbol{c}_i) N_n + (\boldsymbol{t} \cdot \boldsymbol{c}_i) N_t$.

$$
\begin{aligned}
f_4 &= f_2 - \frac{2\rho u_y}{3} - N_y, \\
f_7 &= f_5 - \frac{\rho}{6}(u_x + u_y) - N_x - N_y, \\
f_8 &= f_6 - \frac{\rho}{6}(-u_x + u_y) + N_x - N_y.
\end{aligned} \tag{6.47}
$$

We compute N_x by resorting to the first-order velocity moment along the boundary tangential direction:

$$
\begin{aligned}
\rho u_x &= \sum_i f_i c_{ix} + \frac{\Delta t}{2} \sum_i F_i c_{ix} \\
&= (f_1 + f_5 + f_8) - (f_3 + f_6 + f_7) + \frac{F_x \Delta t}{2} \\
&= (f_1 - f_3) - (f_7 - f_5) + (f_8 - f_6) + \frac{F_x \Delta t}{2} \\
&= (f_1 - f_3) - \frac{\rho u_x}{3} + 2N_x + \frac{F_x \Delta t}{2}.
\end{aligned} \tag{6.48}
$$

This gives

$$
N_x = -\frac{1}{2}(f_1 - f_3) + \frac{\rho u_x}{3} - \frac{F_x \Delta t}{4}. \tag{6.49}
$$

Similarly, we compute N_y based on the first-order velocity moment along the boundary normal direction:

$$
\begin{aligned}
\rho u_y &= \sum_i f_i c_{iy} + \frac{\Delta t}{2} \sum_i F_i c_{iy} \\
&= (f_2 + f_5 + f_6) - (f_4 + f_7 + f_8) + \frac{F_y \Delta t}{2} \\
&= (f_2 - f_4) - (f_7 - f_5) + (f_6 - f_8) + \frac{F_y \Delta t}{2} \\
&= \frac{\rho u_y}{3} + 3N_y + \frac{F_y \Delta t}{2}.
\end{aligned} \tag{6.50}
$$

We obtain

$$
N_y = -\frac{F_y \Delta t}{6}. \tag{6.51}
$$

Clearly, the normal momentum correction N_y is only relevant when forces are included.

In the end, the NEBB prescribes the unknown populations with forces:

$$
\begin{aligned}
f_4 &= f_2 - \frac{2\rho u_y}{3} + \frac{F_y \Delta t}{6}, \\
f_7 &= f_5 + \frac{1}{2}(f_1 - f_3) - \frac{\rho u_x}{2} - \frac{\rho u_y}{6} + \frac{F_x \Delta t}{4} + \frac{F_y \Delta t}{6}, \\
f_8 &= f_6 - \frac{1}{2}(f_1 - f_3) + \frac{\rho u_x}{2} - \frac{\rho u_y}{6} - \frac{F_x \Delta t}{4} + \frac{F_y \Delta t}{6}.
\end{aligned}
\tag{6.52}
$$

The extension of eq. (6.52) to other boundaries orientations is straightforward (*cf.* exercise 6.4).

The necessity for including force corrections in the NEBB method has been recognised in a number of works, *e.g.* []. These terms aim at preventing the appearance of discrete lattice artifacts in the macroscopic laws of wet boundary nodes. Those errors terms are $\propto \nabla \cdot \boldsymbol{F}$. Hence, they will be visible only in *spatially varying force fields*. We illustrate this conclusion with numerical examples in section 6.7. TODO (GS): Recall my referee report to complete refs.

Exercise 6.4. Show that the Dirichlet velocity condition prescribed with the NEBB method at the left boundary takes the following closure form in a flow subject to the force field $\boldsymbol{F} = (F_x, F_y)$:

$$
\begin{aligned}
\rho &= \frac{1}{1 - u_x}\left(f_0 + f_2 + f_4 + 2(f_3 + f_6 + f_7) - \frac{F_x \Delta t}{2}\right), \\
f_1 &= f_3 + \frac{2\rho u_x}{3} - \frac{F_x \Delta t}{6}, \\
f_5 &= f_7 - \frac{1}{2}(f_2 - f_4) + \frac{\rho u_y}{2} + \frac{\rho u_x}{6} - \frac{F_x \Delta t}{6} - \frac{F_y \Delta t}{4}, \\
f_6 &= f_8 + \frac{1}{2}(f_2 - f_4) - \frac{\rho u_y}{2} + \frac{\rho u_x}{6} - \frac{F_x \Delta t}{6} + \frac{F_y \Delta t}{4}.
\end{aligned}
\tag{6.53}
$$

6.7 Numerical example with LB forcing schemes

So far the discussion on LB force models (in bulk and boundaries) was based on theoretical analyses. While this allowed us to understand basic features of LB forcing schemes, such as their consistency, their role in actual LB simulations remains to be discussed. Therefore, the goal of this section is to illustrate the impact of using an incorrect LB force model. For concreteness, we will compare four possible strategies to implement the force model in the LB equation, which are summarised in tab. 6.1.

Benchmark solution

The benchmark adopted herein is the Poiseuille channel flow, already introduced in early chapters, *e.g.* chapter 5. The difference is that now, besides the pressure

Table 6.1: LB forcing schemes used in this section. They differ with respect to the discretisation orders of velocity space and space/time discretisations.

Scheme	Velocity space discret.	Space/time discret.	References
I	1^{st}-order [eq. (6.16)]	1^{st}-order [eq. (6.20)]	[]
II	2^{st}-order [eq. (6.14)]	1^{st}-order [eq. (6.20)]	[]
III	1^{st}-order [eq. (6.16)]	2^{st}-order [eq. (6.25)]	[]
IV	2^{st}-order [eq. (6.14)]	2^{st}-order [eq. (6.25)]	[]

gradient, we also consider the flow to be driven by a force F_x.

$$\rho\nu\frac{\partial u_x}{\partial y} = \frac{\partial p}{\partial x} - F_x. \tag{6.54}$$

The velocity solution u_x is:

$$u_x(y) = \frac{1}{2\rho\nu}\left(\frac{\partial p}{\partial x} - F_x\right)\left(y^2 - \left(\frac{H}{2}\right)^2\right), \tag{6.55}$$

where the no-slip condition $u_x = 0$ holds at bottom/top walls $y = \pm H/2$, respectively, *cf.* fig. 1.1 (b).

Numerical procedure

In bulk the modelling of eq. (6.54) adopts the BGK collision operator with the LB incompressible equilibrium (section 4.3.2). Later, some comments will be made on the application of the standard (compressible) equilibrium.

In boundaries two different wall boundary schemes are considered: the bounce-back and the non-equilibrium bounce-back methods. Each case will be discussed individually.

The simulations are initialised by setting $f_i(\boldsymbol{x}, t = 0) = f_i^{\text{eq}}(\rho = 1, \boldsymbol{u} = \boldsymbol{0})$, *cf.* section 6.6.1, and they are stopped when reaching the steady-state criterion $L_2 \leq 10^{-10}$ (for fluid velocity u_x), *cf.* section **??**. The channel domain is discretised in $N_x = 5$ and $N_y = 5$ grid nodes along streamwise and transversal directions, respectively.

The LB solutions, using each of the four strategies presented in tab. 6.1, are evaluated against the analytical solution eq. (6.55) and deviations are quantified through the L_2 error norm (section **??**). We will perform the numerical exercises below by increasing step-by-step the level of complexity of the physical force modelled.

6.7.1 Constant force

Let us start by considering a purely force-driven Poiseuille flow. This case is established by setting $\partial_x p = 0$ in eq. (6.54) (and consequently in eq. (6.55) as well). Inlet and outlet boundaries can be modelled with periodic conditions, *cf.* section 5.2.1. The absence of pressure (density) variations leads to identical results between incompressible and standard equilibrium models.

Since the force is uniform any *bulk errors*, due to an incorrect LB force modelling, vanish, *cf.* section 6.5. They are however still perceptible at bounce-back boundaries, *cf.* section 6.6.2.

Bounce-back

The solution of the LBGK model with bounce-back walls is summarised in tab. 6.2 for several τ values. The order of the velocity discretisation in the force model formulation plays no role in this problem. Differences exist however in the space-time discretisation order of the LB scheme. While both cases are able to exactly reproduce the parabolic solution, the exact halfway location of the no-slip condition is reached for different τ values. This difference is due to how spatial discretisation errors are cancelled out in each case, see more details in [] [He,Ladd,Ginzburg,Delft].

Table 6.2: Force-driven Poiseuille flow using periodic conditions at inlet/outlet and bounce-back at walls, *cf.* fig. **??**. Results are identical for standard and incompressible equilibrium. $N_y = 5$ etc...$L_2(\%)$

	ϵ_{u_x} (%)	
τ	schemes I and II	schemes III and IV
0.6	5.91	5.18
0.8	5.04	2.85
$\sqrt{3/16} + 1/2$	3.16	2.04×10^{-12}
1.0	1.82	1.82
$\sqrt{13/64} + 5/8$	1.42×10^{-12}	4.20
1.2	3.72	8.83
1.4	11.60	18.17

Non-equilibrium bounce-back

The non-equilibrium bounce-back (NEBB) method follows the same dynamical rules of the bulk solution, where no force errors exist for a constant forcing. Consequently, with the NEBB method the parabolic velocity solution, eq. (6.55), is always reproduced exactly, regardless the force scheme employed.

6.7.2 Constant force and pressure gradient

Let us now increase the complexity of the previous exercise and consider the simultaneous presence of a constant force and pressure-gradient in the Poiseuille flow. The problem remains described by eq. (6.54), with solution given by eq. (6.55). In terms of implementation, the only modification refers to the inlet and outlet boundaries, which are now modelled with pressure periodic boundary conditions, *cf.* section 5.2.2.

Similarly to the previous case, the force is uniform making any bulk error to vanish, *cf.* section 6.5. On the other hand, the macroscopic dynamics prescribed by the bounce-back rule may be affected by different force formulations, *cf.* section 6.6.2.

Bounce-back

The solution of the LBGK model with bounce-back walls is summarised in tab. 6.3. While the order of the velocity discretisation of the force model is immaterial, its space-time discretisation may be important. In fact, we see that only with a second-order space-time discretisation of the forcing term we guarantee the force-driven solution is unchanged by adding the pressure-gradient to the force term – compare tab. 6.3 to tab. 6.2 and observe that solutions are exactly equivalent for any τ value. From the physical point of view this result should be expected as the two conditions (constant force and constant pressure-gradient) are absolutely equivalent in incompressible hydrodynamics. However, this physical condition is no longer preserved when the force discretisation is less accurate than the rest of the LB scheme. As we see in tab. 6.3 even the τ value where the solution becomes exact, $\tau = 1$, is different from the pure force-driven case $\sqrt{13/64} + 5/8$ or the pure pressure-driven case $\sqrt{3/16} + 1/2$.

Table 6.3: Force- and pressure-driven Poiseuille flow using periodic pressure boundary conditions at inlet/outlet and bounce-back at walls, *cf.* fig. **??**. Incompressible equilibrium. $N_y = 5$ etc...$\|L_2\|(\%)$

	ϵ_{u_x} (%)	
τ	Schemes I and II	Schemes III and IV
0.6	5.55	5.18
0.8	3.94	2.85
$\sqrt{3/16} + 1/2$	1.58	1.37×10^{-11}
1.0	5.78×10^{-13}	1.82
$\sqrt{13/64} + 5/8$	2.10	4.20
1.2	6.27	8.83
1.4	14.89	18.17

Table 6.4: Identical to tab. **??** but with standard equilibrium.

τ	Scheme I	Scheme II	Scheme III	Scheme IV
0.6	9.96	9.97	9.65	9.65
0.8	4.53	4.53	3.45	3.45
$\sqrt{3/16} + 1/2$	2.21	1.89	0.35	0.35
1.0	0.30	0.30	1.58	1.58
$\sqrt{13/64} + 5/8$	1.91	1.91	4.00	4.00
1.2	6.13	6.13	8.68	8.68
1.4	14.77	14.77	18.06	18.06

Non-equilibrium bounce-back

Similarly to the pure force-driven case, also here no errors are present. The explanation is the same: both bulk and NEBB solutions follow identical dynamical rules, where any possible force error vanishes in the constant forcing case.

6.7.3 Linear force and pressure gradient

Finally, let us consider the most interesting case where the modelled force is spatially varying. This force increases linearly along the streamwise direction and is balanced by the pressure gradient (which decreases linearly by the same amount in order to guarantee the fully-developed condition). This test case is described in more details in [] [PhysicaA]. It is noted that the problem remains described by eq. (6.54), with solution given by eq. (6.55). The numerical implementation adopts the same setup of the last case.

The key difference from other cases is that now the force bulk error may be present, because the modelled force varies spatially – recall results from the Chapman-Enskog analysis in section 6.4. Consequently, both bulk and boundary errors coming from an incorrect force implementation will affect the LB solution. This case enables the unequivocal identification of the most accurate LB forcing scheme.

Bounce-back

An exact solution is impossible to reach when using a first-order time/space discretisation of the LB scheme (force included), regardless the τ value. This is due to the non-vanishing bulk error, see tab. 6.5. (The explicit form of this error can be recalled from eq. (6.38) in section 6.5.)

Furthermore, this bulk error also makes relevant the discretistaion of the forcing scheme in velocity space. Remembering the discussion provided in section 6.3 and section 6.5, when modelling steady incompressible hydrodynamics subject to an external force, the formulation of the LB forcing term should be restricted to the

first order in velocity space. This is confirmed in tab. 6.5, where only scheme III is able to be exact in this problem.

Finally, the reason for the exact solution taking place for $\tau = \sqrt{3/16}+1/2$ only is due to the bounce-back scheme. Remember it is this τ value that places the bounce-back wall *exactly* halfway from the boundary nodes when the velocity solution is parabolic – section 5.2.3 and section 5.3.1. Such a result is consistent throughout all exercises.

Table 6.5: Force- and pressure-driven Poiseuille flow using periodic pressure boundary conditions at inlet/outlet and bounce-back at walls, *cf.* fig. **??**. Incompressible equilibrium. $N_y = 5$ etc...$\|L_2\|(\%)$

	ϵ_{u_x} (%)			
τ	Scheme I	Scheme II	Scheme III	Scheme IV
0.6	6.10	5.44	5.18	5.07
0.8	4.12	3.84	2.85	2.74
$\sqrt{3/16}+1/2$	1.70	1.48	8.46×10^{-15}	0.12
1.0	0.51	0.51	1.82	1.94
$\sqrt{13/64}+5/8$	2.00	2.21	5.47	4.32
1.2	6.19	6.39	8.83	8.96
1.4	14.81	15.03	18.17	18.32

Table 6.6: Identical to tab. **??** but with standard equilibrium.

τ	Scheme I	Scheme II	Scheme III	Scheme IV
0.6	7.39	6.75	6.48	6.38
0.8	4.30	4.03	3.02	2.91
$\sqrt{3/16}+1/2$	1.82	1.59	0.13	0.12
1.0	0.43	0.38	1.74	1.85
$\sqrt{13/64}+5/8$	1.91	2.12	4.13	4.25
1.2	6.11	6.31	8.77	8.90
1.4	14.73	14.95	18.12	18.27

Non-equilibrium bounce-back

The outcomes using the NEBB scheme are essentially similar to those of the bounce-back case. Once again a bulk error may corrupt the LB solutions if the velocity space and space/time discretisations of the forcing term are not properly handled. When targeting steady incompressible flow solutions the correct force formulation is scheme III.

The fact that the exact solution is reached for any τ value in scheme III is explained by the use of the NEBB scheme. For parabolic flow solutions (as in the

present case) the NEBB prescribes the no-slip wall condition at the grid nodes *exactly* and independently of τ – recall section 5.2.4 and section 5.3.1.

Table 6.7: Force- and pressure-driven Poiseuille flow using periodic pressure boundary conditions at inlet/outlet and bounce-back at walls, *cf.* fig. **??**. Incompressible equilibrium. $N_y = 5$ etc...$\|L_2\|(\%)$

	ϵ_{u_x} (%)			
τ	Scheme I	Scheme II	Scheme III	Scheme IV
0.6	0.19	0.27	8.60×10^{-14}	0.05
0.8	0.42	0.41	4.41×10^{-15}	0.07
$\sqrt{3/16}+1/2$	0.63	0.63	1.24×10^{-14}	0.07
1.0	0.74	0.74	1.32×10^{-14}	0.07
$\sqrt{13/64}+5/8$	0.87	0.87	1.24×10^{-14}	0.07
1.2	1.08	1.08	2.39×10^{-14}	0.07
1.4	1.42	1.42	1.55×10^{-14}	0.07

TODO: Treat the compressible case briefly and substitute tables by plots.

References

1. S.H. Kim, H. Pitsch, Phys. Fluids **19**, 108101 (2007)
2. Z. Guo, C. Zheng, B. Shi, Phys. Rev. E **65**, 46308 (2002)
3. P. Dellar, Phys. Rev. E **64**(3) (2001)
4. S. Ubertini, P. Asinari, S. Succi, Phys. Rev. E **81**(1), 016311 (2010)
5. J.M. Buick, C.A. Greated, Phys. Rev. E **61**(5), 5307 (2000)
6. I. Ginzbourg, P.M. Adler, J. Phys. II France **4**(2), 191 (1994)
7. A.J.C. Ladd, J. Fluid Mech. **271**, 285 (1994)
8. L.S. Luo, Phys. Rev. Lett. **81**(8), 1618 (1998)
9. N.S. Martys, X. Shan, H. Chen, Phys. Rev. E **58**(5), 6855 (1998)
10. X. He, X. Shan, G.D. Doolen, Phys. Rev. E **57**(1), R13 (1998)
11. A.J.C. Ladd, R. Verberg, J. Stat. Phys. **104**(5-6), 1191 (2001)
12. A. Kupershtokh, in *Proc. 5th International EHD Workshop, University of Poitiers, Poitiers, France* (2004), p. 241–246
13. Z. Guo, C. Zheng, B. Shi, T. Zhao, Phys. Rev. E **75**(036704), 1 (2007)
14. R.W. Nash, R. Adhikari, M.E. Cates, Phys. Rev. E **77**(2), 026709 (2008)
15. R.G.M. Van der Sman, Phys. Rev. E **74**, 026705 (2006)
16. X. Nie, N.S. Martys, Phys. Fluids **19**, 011702 (2007)
17. I. Ginzburg, Phys. Rev. E **77**(066704), 1
18. Z. Guo, C. Zheng, B. Shi, Phys. Rev. E **83**, 036707 (2011)
19. A. Kuzmin, Z. Guo, A. Mohamad, Phil. Trans. Royal Soc. A **369**, 2219 (2011)
20. G. Silva, V. Semiao, Physica A **390**(6), 1085 (2011)
21. G. Silva, V. Semiao, J. Fluid Mech. **698**, 282 (2012)
22. D. Lycett-Brown, K.H. Luo, Phys. Rev. E **91**, 023305 (2015)
23. I. Ginzburg, G. Silva, L. Talon, Phys. Rev. E **91**, 023307 (2015)
24. S. Khirevich, I. Ginzburg, U. Tallarek, J. Comp. Phys. **281**, 708 (2015)

Chapter 7
Non-dimensionalisation and choice of simulation parameters

Being able to map the physical properties of a system to the lattice (and the results of a simulation back to a prediction about a physical system) is essential. A computer does not know about the concept of a physical dimension like length or time. Computers simply perform arithmetic operations on numbers, ignoring what they physically represent, such as a distance or time interval. This requires to convert any dimensional quantity into a non-dimensional or dimensionless *lattice* quantity (and *vice versa* to interpret the simulation results).

Unless the simulation program has a built-in functionality for performing unit conversion between physical and lattice units, the user is responsible for specififying simulation parameters in lattice units. Unit conversion is straightforward once the basic rules are clear (section 7.1). Unfortunately, few references in the LB literature cover this topic coherently and rigorously.

LB simulations require not only a good knowledge of unit conversion. Due to intrinsic restrictions of the LB algorithm, it is crucial to balance the simulation parameters in such a way that a suitable compromise of accuracy, stability and efficiency is achieved (section 7.2). We also provide numerous examples to demonstrate the non-dimensionalisation process (section 7.3). Finally, a summary with the most important rules is given in section 7.4.

7.1 Non-dimensionalisation

We introduce the underlying concepts and basic rules to non-dimensionalise physical parameters (section 7.1.1). The *law of similarity* plays a crucial role (section 7.1.2).

7.1.1 Unit scales and conversion factors

In order to indicate a dimensional physical quantity, one requires a reference scale. For example, a length ℓ can be reported in multiples of a unit scale with length $\ell_0 = 1\,\mathrm{m}$. If a given length ℓ is ten times as long as the unit scale ℓ_0, we say that its length is $10\,\mathrm{m}$: $\ell = 10\,\mathrm{m} = 10\,\ell_0$. This convention is not unique. One may wish to express the length compared to a different unit scale ℓ'_0, *e.g.* $\ell'_0 = 1\,\mathrm{ft}$. How long is ℓ in terms of ℓ'_0? To answer this question we have to know how long ℓ_0 is compared to ℓ'_0. We know that 1 ft corresponds to 0.3048 m: $\ell'_0/\ell_0 = 0.3048\,\mathrm{m/ft}$. Therefore, it is straightforward to report ℓ in feet, rather than metres:

$$\ell = 10\,\mathrm{m} = 10\,\ell_0 = 10\,\ell'_0\frac{\ell_0}{\ell'_0} = 10\,\mathrm{ft}\frac{1}{0.3048} \approx 32.81\,\mathrm{ft} = 32.81\,\ell'_0. \tag{7.1}$$

Although the numerical values are not the same, the length ℓ is identical in both representations. This calculation seems to be trivial, but it is the key to understand unit conversion and non-dimensionalisation.

Non-dimensionalisation is achieved by dividing a dimensional quantity by a chosen reference quantity of the same dimension. The result is a number which we will sometimes call the *lattice* value of the quantity when talking about the LBM. The reference quantity is called the *conversion factor*. Let us denote conversion factors by C and non-dimensionalised quantities by a star *[1], then we can write for any length ℓ:

$$\ell^\star = \frac{\ell}{C_\ell}. \tag{7.2}$$

The conversion factor C_ℓ has to be chosen appropriately. How to do this for a given LB simulation will be discussed in section 7.2; but let us first focus on the non-dimensionalisation itself.

Any mechanical quantity q has a dimension which is a combination of the dimensions of length ℓ, time t and mass m. In the following, let us only use SI units, *i.e.* metre (m) for length, second (s) for time and kilogramme (kg) for mass. The units of q, denoted as $[q]$, are therefore

$$[q] = [\ell]^{q_\ell}[t]^{q_t}[m]^{q_m}. \tag{7.3}$$

The exponents q_ℓ, q_t and q_m are numbers. Let us take the velocity u as an example. We know that the dimension of velocity is length over time:

$$[u] = \frac{\mathrm{m}}{\mathrm{s}} = [\ell]^1[t]^{-1} = [\ell]^{q_\ell}[t]^{q_t}[m]^{q_m}. \tag{7.4}$$

Therefore, we find $q_\ell = 1$, $q_t = -1$ and $q_m = 0$. It is a simple exercise to find the three numbers q_ℓ, q_t and q_m for any mechanical quantity q. Tab. 7.1 shows a few examples.

[1] Note that the star also denotes post-collision values. There is no danger of confusing both concepts in this chapter, though.

Table 7.1: Units of selected mechanical quantities.

quantity	symbol	unit	q_ℓ	q_t	q_m
length	ℓ	m	1	0	0
time	t	s	0	1	0
mass	m	kg	0	0	1
velocity	u	m/s	1	−1	0
gravity	g	$\mathrm{m/s^2}$	1	−2	0
force	F	$\mathrm{kg\,m/s^2}$	1	−2	1
force density	F	$\mathrm{kg/(m^2\,s^2)}$	−2	−2	1
density	ρ	$\mathrm{kg/m^3}$	−3	0	1

In practical situations, one does not only want to non-dimensionalise one quantity, but all of them. How can this be achieved in a consistent way? Since three fundamental dimensions are sufficient to generate the dimension of any mechanical quantity, one requires exactly three independent conversion factors in order to define a unique non-dimensionalisation scheme. Three conversion factors C_i ($i = 1, 2, 3$) are independent if the relations

$$[C_i] = [C_j]^{a_i}[C_k]^{b_i} \quad (i \neq j \neq k \neq i) \tag{7.5}$$

have no solution for the numbers a_i and b_i. In other words: none of the dimensions of the three conversion factors must be a combination of the other two. For example, the conversion factors for length, time and velocity are dependent because $[C_u] = [C_\ell]/[C_t]$. But the conversion factors for velocity, force and time are independent. This argumentation is tightly related to the *Buckingham π theorem* [1].

Exercise 7.1. What are the exponents q_u, q_t, q_ρ for a force F if velocity u, time t and density ρ are used? (Answer: $q_\rho = 1$, $q_u = 4$, $q_t = -2$)

Let us call any set of three independent conversion factors *basic conversion factors*. The first step is to construct a system of three basic conversion factors. All other required conversion factors (let us call them *derived conversion factors*) can then be easily constructed as we will see shortly.

It is completely arbitrary which basic conversion factors to choose, but in LB simulations one usually takes C_ℓ, C_t (or C_u) and C_ρ because length, time (or velocity) and density are natural quantities in any LB simulation. Using basic conversion factors for energy or force are possible but usually impractical.

Care must be taken when a 2D system is simulated. The dimension of density ρ is mass per volume, but in 2D it is formally mass per area. But how can a 3D density be mapped to a 2D density? This dilemma can be circumvented by pretending that a simulated 2D system is nothing more than a 3D system with thickness of one lattice constant, *i.e.* an $N_x \times N_y$ system is treated as an $N_x \times N_y \times 1$ system where N_x and N_y are the number of lattice sites along the

x- and y-axes. This enables us to use 3D conversion factors for 2D problems without any restrictions.

7.1.2 Law of similarity and derived conversion factors

Let us pretend that we have a set of three basic conversion factors. How do we obtain the derived conversion factors for other mechanical quantities? First of all we note that physics is independent of units, which are an arbitrary human construct. Ratios of physical phenomena are what matter: a pipe diameter of 1 m means that the diameter is the length of the path travelled by light in vacuum in $1/299\,792\,458$ of a second, with time intervals defined through periods of radiation from a caesium atom. In particular, the physical outcome should not depend on whether we use dimensional or dimensionless quantities.

Related to this is the *law of similarity* in fluid dynamics which is nicely explained by Landau and Lifshitz [2]: two fluid systems are dynamically similar if they have the same Reynolds number and geometry. The Reynolds number may be defined as

$$\mathrm{Re} = \frac{\ell u}{\nu} = \frac{\rho \ell u}{\eta} \tag{7.6}$$

where ℓ and u are typical length and velocity scales in the system and ρ, ν and η are the density, kinematic viscosity and dynamic viscosity of the fluid, respectively.[2]

The law of similarity is, for instance, regularly used in the automotive or aircraft industries where models of cars or planes are tested in wind tunnels. Since the models are usually smaller than the real objects, one has to increase the flow velocity or decrease the viscosity in such a way that the Reynolds numbers in both systems match. For example, if the model of a car is five times smaller than the car itself, the velocity of the air in the wind tunnel should be five times larger than in reality (given the same density and viscosity).

Another example is the flow in a T-junction. Consider two fluids: one with water-like viscosity and one with the same density but ten times more viscous. Both flow at the same average speed through a T-junction with circular pipes, but the more viscous fluid flows through pipes with ten times the diameter (and ten times the length of pipe before and after the junction). In this example, the Reynolds number is the same and the geometry is similar, and so we expect identical solutions (in non-dimensional space). Should we perform a simulation that matches the physical properties of the water flow or the more viscous flow? The answer is that the simula-

[2] Note that the definition of the Reynolds number is not unique: on the one hand, some people may choose ℓ for the length, others for the width of the considered system. On the other hand, u may be the average velocity or the maximum velocity — depending on the person defining the Reynolds number. This has to be kept in mind when comparing Reynolds numbers from different sources.

tion should match the Reynolds number in a way that optimises the accuracy of the solution. The results of the simulation can then be applied to *both* physical systems.

This means that the Reynolds number must be identical in both unit systems (the physical and the lattice system):

$$\frac{\ell^\star u^\star}{\nu^\star} = \frac{\ell u}{\nu}. \tag{7.7}$$

Plugging in the definition of the conversion factors (*e.g.* $C_\ell = \ell/\ell^\star$) simply yields

$$\frac{C_\ell C_u}{C_\nu} = 1 \tag{7.8}$$

or — if for example C_ν shall be computed from C_ℓ and C_u:

$$C_\nu = C_\ell C_u. \tag{7.9}$$

Based on the dimensions of the conversion factors alone, one could have come to the conclusion that $C_\nu \propto C_\ell C_u$ must hold, although this is only a proportionality rather than an equality. The law of similarity for the Reynolds number, however, uniquely defines the relation of the conversion factors for viscosity, length and velocity.

This result can be generalised to all missing conversion factors. Any derived conversion factor C_q can be constructed directly by writing down a suitable combination of basic conversion factors of the form

$$C_q = C_1^{q_1} C_2^{q_2} C_3^{q_3} \tag{7.10}$$

without any additional numerical prefactor.[3] The problem reduces to finding a suitable set of numbers q_i, which is usually not a difficult task. The resulting expression for C_q is unique and guarantees that the conversion is consistent: the physics of the system, *i.e.* the characteristic dimensionless numbers are kept invariant, and the law of similarity is satisfied.

Example 7.1. For the given basic conversion factors C_ℓ, C_t and C_ρ, one can easily construct the conversion factor for pressure p. First we observe that $[p] = \mathrm{N/m^2} = (\mathrm{kg/m^3})\,\mathrm{m^2/s^2}$. Therefore one directly finds

$$C_p = \frac{C_\rho C_\ell^2}{C_t^2}. \tag{7.11}$$

Some concrete examples of non-dimensionalisation procedures are shown in section 7.3.

Exercise 7.2. How does the pressure conversion factor look like if C_u rather than C_t is used? (Answer: $C_p = C_\rho C_u^2$)

[3] This is true as long as we stick to SI units. Mixing metres, kilometres and feet, for example, will generally lead to additional numerical prefactors. This is the beauty of using SI units!

It has to be emphasised that unit systems must not be mixed. This is a typical mistake made by most users at some point. Mixing unit systems causes inconsistencies in the definition of the conversion factors and a subsequent violation of the law of similarity. It is strongly advised to clearly mark quantities in different unit systems (*e.g.* by adding * or subscripts like p for 'physical' and l for 'lattice'). Additionally, it is recommended to perform all calculations in pure SI units without using prefixes like 'milli' or 'centi', which otherwise easily leads to confusion.

7.2 Parameter selection

We discuss the relevance of parameters in the LBM in general (section 7.2.1), the effect of accuracy, stability and efficiency (section 7.2.2) and the strategies how to find simulation parameters for a given problem in particular (section 7.2.3).

7.2.1 Parameters in the lattice Boltzmann method

Any LB simulation is characterised by a set of parameters:

- The lattice constant Δx is the distance between neighbouring lattice nodes in physical units, *i.e.* $[\Delta x] = \mathrm{m}$.
- The physical length of a time step is denoted Δt, therefore $[\Delta t] = \mathrm{s}$.
- The BGK relaxation parameter τ is usually understood as a number. However, strictly speaking, τ is a relaxation time with $[\tau] = \mathrm{s}$. In order to avoid ambiguities in this chapter, we write τ for the physical relaxation time and $\tau^\star$ for the dimensionless relaxation parameter.
- The dimensionless fluid density is $\rho^\star$. It is usually set to unity. This situation is slightly more complicated for multi-component or multi-phase simulations (chapter 9), where the density can be different from unity.
- Another important parameter is the typical simulated velocity $u^\star$. It is usually not an input parameter but rather part of the simulation output. However, some boundary conditions require specification of $u^\star$ on the boundaries as inlet and outlet velocities. Additionally, it is desired to estimate the magnitude of $u^\star$ before the simulation is started in order to avoid unstable situations or very long computing times.
- In the standard LBM, the lattice speed of sound $c_\mathrm{s}^\star$ is $\sqrt{1/3} \approx 0.577$. In order to operate in the quasi-incompressible limit, all simulated velocities have to be significantly smaller: $u^\star \ll c_\mathrm{s}^\star$. In practice this means that the maximum value of $u^\star$ should be between 0.1 and 0.2.

The first step is to relate the physical parameters Δx, Δt, τ, ρ and u to their lattice counterparts $\Delta x^\star$, $\Delta t^\star$, $\tau^\star$, $\rho^\star$ and $u^\star$. It is very common, though not necessary, to set $\Delta x^\star = 1$, $\Delta t^\star = 1$ and $\rho^\star = 1$.[4] This means that the conversion factors for length, time and density equal the dimensional values for the lattice constant, time step and density:

$$C_\ell = \Delta x, \quad C_t = \Delta t, \quad C_\rho = \rho. \tag{7.12}$$

It is extremely important to realise that Δx and $\Delta x^\star$ are the *same* quantity in *different* unit systems. Most non-dimensionalisation problems are caused by the fact that users confuse physical and non-dimensional quantities, *i.e.* Δx with $\Delta x^\star$, or Δt with $\Delta t^\star$. The parameters τ and $\tau^\star$ are connected through the conversion factor for Δt because τ has the dimension of time:

$$\tau = \tau^\star C_t = \tau^\star \Delta t. \tag{7.13}$$

However, the conversion factor for the velocity is $C_u = C_\ell / C_t = \Delta x / \Delta t$, and it is not independent. Thus, Δx, Δt and ρ form a complete set of basic conversion factors. Alternatively one may use Δx, C_u and ρ or other independent combinations. As we will see later, different choices of the set of conversion factors result in different strategies for non-dimensionalisation.

Viscosity

One of the important physical fluid properties is viscosity. From section 4.1 TODO (TK): Check reference we know that the kinematic lattice viscosity is related to the relaxation parameter according to

$$\nu^\star = c_\mathrm{s}^{\star 2} \left(\tau^\star - \tfrac{1}{2} \right) \tag{7.14}$$

with the lattice speed of sound $c_\mathrm{s}^\star$. A typical problem is to relate the *dimensionless* relaxation parameter $\tau^\star$ to the *physical* kinematic viscosity ν since the latter is usually given by an experiment and the former has to be defined for a simulation. The conversion factor for ν is $C_\nu = C_\ell^2 / C_t$, which directly follows from the dimension of ν: $[\nu] = \mathrm{m}^2/\mathrm{s}$. Therefore, we can write

$$\nu = c_\mathrm{s}^{\star 2} \left(\tau^\star - \tfrac{1}{2} \right) \frac{\Delta x^2}{\Delta t}. \tag{7.15}$$

This is a consistency equation for the three simulation parameters $\tau^\star$, Δx and Δt, which means that these three parameters are not independent. Only two of them can

[4] Some authors prefer to define the total system size and the total simulation time rather than the lattice constant and the time step to unity. This is nothing more than a different but valid unit system. As a consequence, the conversion factors C_ℓ and C_t would be different.

be chosen freely. We will soon consider the intrinsic limitations of the LB algorithm which further restrict the choice of parameters.

Pressure, stress and force

There are other additional quantities which are commonly encountered in LB simulations: pressure p, stress σ and force F. We know from section 4.1 that the equation of state of the LB fluid is

$$p^\star = c_\mathrm{s}^{\star 2}\rho^\star. \tag{7.16}$$

This is, however, not the entire truth. Only the pressure gradient ∇p rather than the pressure p by itself appears in the Navier-Stokes equation. While the total pressure p *does* appear in the energy equation, this equation is not relevant for non-thermal LB models. The reference pressure is thus irrelevant; only pressure *changes* matter.

To connect with the physical pressure, one decomposes the LB density into its constant average $\rho_0^\star$ and deviation $\rho'^\star$ from the average:

$$\rho^\star = \rho_0^\star + \rho'^\star. \tag{7.17}$$

It is often wrongly assumed that the reference density $\rho_0^\star$ has to correspond to a physical reference pressure p_0, *e.g.* atmospheric pressure. Generally, the LB density can be converted to the physical pressure for non-thermal models as

$$p = p_0 + p' = p_0 + p'^\star C_p, \quad p'^\star = c_\mathrm{s}^{\star 2}\rho'^\star, \tag{7.18}$$

where $C_p = C_\rho C_\ell^2/C_t^2 = C_\rho C_u^2$ is the conversion factor for pressure, and p_0 is the physical reference pressure which can be freely specified by the user. Thus, the LB equation of state can model a number of realistic equations of state [3]; see also Section 1.1.3 for more details.

Example 7.2. A simulation with $\rho_0^\star = 1$ yields a density fluctuation $\rho'^\star = 0.03$ at a given point. The pressure conversion factor is known to be $C_p = 1.2 \cdot 10^3$ Pa. What is the physical pressure at that point? First we can compute the physical value of the pressure fluctuation from $p' = c_\mathrm{s}^{\star 2}\rho'^\star C_p = \frac{1}{3} \cdot 0.03 \cdot 1.2 \cdot 10^3\,\mathrm{Pa} = 12\,\mathrm{Pa}$. However, since p_0 has not been specified, we do not know the absolute pressure in physical units. It may be wrong to compute it from $c_\mathrm{s}^{\star 2}\rho_0^\star C_p$ which would give 400 Pa. Atmospheric pressure, for example is roughly 10^5 Pa and orders of magnitude larger than the "wrong" reference pressure of 400 Pa. This misconception often leads to confusion and incorrectly computed real-world pressure values.

The components of the stress tensor have the same dimension as a pressure, therefore the conversion factor is always identical. For example, we can obtain the lattice deviatoric stress tensor *via* eq. (4.13) and then convert it to physical units by $\boldsymbol{\sigma} = \boldsymbol{\sigma}^\star C_\sigma$ with $C_\sigma = C_p$.

The procedure for forces is straightforward (*cf.* tab. 7.1). A force which is often encountered in hydrodynamic situations is the drag or lift force acting on the surface of obstacles in the flow. It is obtained by computing the surface integral of the stress (*cf.* section 5.3.3). The conversion factor for any force (no matter if body force or surface force) is $C_F = C_\rho C_\ell^4 / C_t^2$. One has to be careful when talking about body forces, though: authors often write "body force" but actually mean "body force density" as in body force per volume. The conversion factor for a body force *density* F is obviously $C_F = C_F / C_\ell^3 = C_\rho C_\ell / C_t^2$.

Additional confusion is commonly caused by gravity. In physical terms, gravity g is an acceleration, $[g] = \mathrm{m/s^2}$, not a force or force density. The gravitational force density F_g is given by $F_g = \rho g$. The conversion factor for gravity is C_ℓ / C_t^2. A precise language is therefore very helpful when talking about gravity (acceleration), force and force density.

The dimensions of pressure, stress and force density depend on the number of spatial dimensions. This has to be taken into account when 2D rather than 3D simulations are performed. Additionally it is always helpful to write down a table similar to tab. 7.1 with all quantities of interest, their dimensions and chosen conversion factors.

7.2.2 Accuracy, stability and efficiency

Eq. (7.15) tells us that there is an infinite number of possibilities to get the correct physical viscosity by balancing $\tau^\star$, Δx and Δt. The key question is: how does one choose these parameters? To answer this, we have to consider intrinsic limitations of the LB algorithm. In the end, one has to choose the parameters in such a way that simulation accuracy, stability and efficiency are reasonably considered. Here we will focus on the BGK collision operator. Note that some stability and accuracy problems of the BGK operator can be solved by using advanced collision operators (TRT or MRT, chapter 10) instead. We will see that there is no free lunch: increased accuracy and stability usually come at the expense of increased computing time. A similar, yet shorter discussion can be found in section 2.2 of [4].

Accuracy and parameter scaling

There are several error terms which affect the accuracy of an LB simulation (*cf.* section 4.5):

- The spatial discretisation error scales like Δx^2, *i.e.* LB is second-order accurate in space [5, 6].
- The time discretisation error scales like Δt^2 [5, 7, 8].
- There is a compressibility error $\propto \mathrm{Ma}^2 \propto u^{\star 2}$ for simulations in the incompressible limit. Since $u^\star$ decreases with increasing $C_u = C_\ell / C_t = \Delta x / \Delta t$, this error scales like $\Delta t^2 / \Delta x^2$.

- The BGK truncation error in space is proportional to $(\tau - 1/2)^2$ [9].

The user's task is to make sure that none of these error contributions plays too large a role. In fact, we observe that increasing the lattice resolution (decreasing Δx) alone does not necessarily reduce the error because the compressibility error will grow and dominate eventually. Reducing only the time step (decreasing Δt) does not decrease the spatial error. Thus, one needs to come up with certain relationships between Δx and Δt to control the error.

One particular relation between Δx and Δt is given by the *diffusive scaling*: $\Delta t \propto \Delta x^2$.[5] It guarantees that the leading order of the overall error scales like Δx^2. However, the LB algorithm then becomes effectively first-order accurate in time [6]. Additionally, since $u^{\star 2} \propto \Delta t^2/\Delta x^2 \propto \Delta x^2$ in the diffusive scaling, LB becomes second-order accurate in velocity.

The expression "diffusive scaling" stems from the apparent similarity of $\Delta t \propto \Delta x^2$ and the diffusion equation, but there is no physical relation between both. It is worth mentioning that the diffusive scaling leaves $\tau^\star$, and hence the non-dimensional viscosity $\nu^\star$, unchanged. This follows from eq. (7.15). The diffusive scaling is the standard approach to test if an LB algorithm is indeed second-order accurate: one performs a series of simulations, each with a finer resolution Δx than the previous. The overall velocity error should then decrease proportional to Δx^2.

The *acoustic scaling* $\Delta t \propto \Delta x$ keeps the compressibility error unchanged. If one is only interested in incompressible situations, the speed of sound does not have any physical significance and any compressibility effects are undesired. The diffusive scaling is then the method of choice to reduce compressibility effects proportional to Δx^2. If, however, the speed of sound is a physically relevant parameter (as in compressible fluid dynamics and acoustics), the acoustic scaling must be chosen as it maintains the correct scaling of the speed of sound. Holdych *et al.* [6] emphasised that the numerical solution can only converge to the solution of the *incompressible* Navier-Stokes equation when $\Delta t \propto \Delta x^\gamma$ with $\gamma > 1$ since the compressibility error remains constant for $\gamma = 1$.

It has to be noted that also the value of $\tau^\star$ affects the accuracy. $\tau^\star \gg 1$ should be avoided [6] because the error of the *bulk* BGK algorithm grows with $(\tau^\star - \frac{1}{2})^2$. Note that the presence of complex boundary conditions modelled with lower-order methods (such as simple bounce-back) leads to a non-trivial overall error dependence on $\tau^\star$. It is generally recommended to choose relaxation times around unity [6, 10, 11]. This is not feasible if the desired Reynolds number is large and the viscosity (and therefore $\tau^\star$) are small. We will get back to this point in section 7.2.3.

The above discussion can only give a recommendation as to how Δx and Δt should be changed at the same time, *e.g.* for a grid refinement study, but it does not provide any information about the initial choice of Δx and Δt. This will be discussed in section 7.2.3.

[5] Such a relation between the spatial and temporal scales is no special feature of the LB algorithm. It can also be found in typical time-explicit centred finite difference schemes, such as the DuFort-Frankel scheme.

Stability

The stability of LB algorithms has already been discussed in section 4.4. The essential results are that the relaxation parameter $\tau^\star$ should not be too close to $\frac{1}{2}$ and that the velocity $u^\star$ should not be larger than about 0.4 for $\tau^\star \geq 0.55$. Another result of section 4.4 was that the achievable maximum velocity is decreasing when $\tau^\star$ approaches $\frac{1}{2}$. For $\tau^\star < 0.55$, we can approximate this relation by

$$\tau^\star > \tfrac{1}{2} + \alpha u^\star_{\max} \tag{7.19}$$

where α is a numerical constant which is on the order of $\frac{1}{8}$.

Example 7.3. On the one hand, if $\tau^\star = 0.51$ is chosen, the maximum velocity should be below 0.08. On the other hand, for an expected maximum velocity of 0.01, $\tau^\star$ may be chosen as low as 0.50125.

The values reported above are only guidelines. The onset of instability also depends on the flow geometry and other factors. Simulations may therefore remain stable for smaller values of $\tau^\star$ or become unstable for larger values. In fact, instabilities are often triggered at boundaries rather than in the bulk [12].

Related to the stability considerations above is the *grid Reynolds number* $\mathrm{Re_g}$ that is defined by taking the lattice resolution Δx as length scale:

$$\mathrm{Re_g} = \frac{u^\star_{\max}\Delta x^\star}{\nu^\star} = \frac{u^\star_{\max}}{c_s^{\star 2}\left(\tau^\star - \frac{1}{2}\right)} \quad \Longrightarrow \quad \tau^\star = \tfrac{1}{2} + \frac{u^\star_{\max}}{c_s^{\star 2}\mathrm{Re_g}}. \tag{7.20}$$

A comparison with eq. (7.19) reveals that $\mathrm{Re_g}$ should not be much larger than $O(10)$. The physical interpretation is that the lattice should always be sufficiently fine to resolve local vortices. In other words: the simulation remains stable as long as all relevant hydrodynamic length scales are resolved.

Efficiency

The efficiency of an LB simulation can indicate two things: performance and optimisation level of the simulation code on the one hand and required number of lattice sites and iterations for a given physical problem on the other hand. Here, we will only address the latter. Remarks about code optimisation can be found in chapter 13.

The total number of site updates required to complete a simulation is $N_s N_t$ where N_s is the total number of lattice sites and N_t is the required number of time steps. The memory requirements are proportional to N_s (LB is a quite memory-hungry method, *cf.* chapter 13). It is clear that $N_s \propto 1/\Delta x^d$ in a situation with d spatial dimensions. Additionally, $N_t \propto 1/\Delta t$ holds independently of the chosen lattice. The finer space and time are resolved (*i.e.* the smaller Δx and Δt are), the more site updates are required. Assuming that the computing time for one lattice site and one time step is a fixed number (typical codes reach a few million site updates per second on modern

desktop CPUs), the total required runtime T and memory M obey

$$T \propto \frac{1}{\Delta x^d \Delta t}, \quad M \propto \frac{1}{\Delta x^d}. \tag{7.21}$$

It is therefore important to choose Δt and especially Δx as large as possible to reduce the computational requirements. However, as we have seen before, a coarser resolution usually reduces the accuracy and brings the system closer to the stability limit.

7.2.3 Strategies for parameter selection

The intrinsic limitations of the LB algorithm — as briefly discussed in the previous section — have to be considered when choosing the simulation parameters for the BGK collision operator:

- If the spatial or temporal resolution shall be refined or coarsened, one should obey the diffusive scaling $\Delta t \propto \Delta x^2$.
- $\tau^\star$ should be close to unity if somehow possible.
- For $\tau^\star$ close to $\frac{1}{2}$, $\tau^\star$ should obey eq. (7.19) which is equivalent to the requirement of having a sufficiently small grid Reynolds number Re_g.
- In any case, the maximum velocity $u^\star_{\max}$ should not exceed 0.4 to maintain stability. If accurate results are desired, $u^\star_{\max}$ should even be smaller (between 0.1 and 0.3) due to the truncation of the equilibrium distributions. Usually, a good strategy is to have velocity ${u^\star}_{\max}$ less than 0.1.
- Since the computing time and memory requirements increase very strongly with (powers of) the inverse of Δx, one should choose Δx as large as possible without violating the aforementioned limitations.

These intrinsic LB restrictions are also collected in tab. 7.2.

Mapping of dimensionless physical parameters

Any physical system can be characterised by dimensionless parameters like the Reynolds or Mach numbers. The first step before setting up a simulation is to identify these parameters and assess their relevance.

Inertia, for example, is relevant as long as Re is larger than order unity. Only flows with vanishing small Reynolds number (Stokes flow) do not depend on the actual value of Re: there is virtually no difference between the flow patterns for $\mathrm{Re} = 10^{-3}$ and $\mathrm{Re} = 10^{-6}$. The limit below which Re does not matter depends on the flow details and the definition of Re. For many applications in fluid dynamics,

Re is not small and, thus, it has to be properly mapped from the real world to the simulation (law of similarity).[6]

LB is mostly used for the simulation of incompressible fluids where the Mach number is small. It is not necessary to map the exact value of Ma then; it is sufficient to guarantee that Ma is "small" in the simulation.

A lattice Mach number, which is defined as $\mathrm{Ma}^{(\mathrm{l})} = u^\star / c_\mathrm{s}^\star$, is considered small if $\mathrm{Ma}^{(\mathrm{l})} < 0.3$. Larger Mach numbers lead to more significant compressibility errors. Note that many users try to match the lattice and real-world Mach numbers; this is not necessary in almost all situations where LBM is used.

Example 7.4. Let us investigate which effect a mapping of Re and Ma has on the simulation parameters. We have already found eq. (7.15) which is based on the law of similarity for the Reynolds number. This equation poses one relation for the three parameters $\tau^\star$, Δx and Δt. If we now additionally claim that the lattice and physical Mach numbers shall be identical, we obtain

$$\mathrm{Ma}^{(\mathrm{l})} = \mathrm{Ma}^{(\mathrm{p})} \quad \Longrightarrow \quad \frac{u^\star}{c_\mathrm{s}^\star} = \frac{u}{c_\mathrm{s}} \quad \Longrightarrow \quad \frac{\Delta x}{\Delta t} = C_u = \frac{u}{u^\star} = \frac{c_\mathrm{s}}{c_\mathrm{s}^\star}. \tag{7.22}$$

Since $c_\mathrm{s}^\star$ is known and c_s is defined by the physical system to be simulated, the ratio of Δx and Δt is fixed by the claim that the Mach numbers match (acoustic scaling). This leaves only one of the three parameters $\tau^\star$, Δx and Δt as an independent quantity.[7]

Luckily, in situations where compressibility effects are not desired, the Mach number mapping can be dropped. It is computationally more efficient to increase the lattice Mach number as much as possible, since the time step scales like $1/u^\star \propto 1/\mathrm{Ma}^{(\mathrm{l})}$.

Example 7.5. Typical wind speeds are a few metres per second while the speed of sound in air is about 330 m/s. The Mach number is then on the order of 0.01, and compressibility effects are not relevant. By choosing a simulation Mach number of $\mathrm{Ma}^{(\mathrm{l})} = 0.3$, while keeping the Reynolds number fixed, the simulation runs 30 times faster than a simulation with a mapping of the Mach number. Of course it is not automatically guaranteed that this simulation is sufficiently accurate and stable, but this example gives an idea about how useful it can be to increase the Mach number.

---

[6] Some phenomena remain (inversely) proportional to Re, for example the drag coefficient. Thus, there is still a significant difference in the result of a simulation between $\mathrm{Re} = 10^{-3}$ and 10^{-6}, although inertia may be irrelevant in both.

[7] As will be covered in chapter 12, treating c_s as physically relevant usually leads to small values of $\tau^\star$ or Δx and Δt, rendering LB an expensive scheme to simulate acoustic problems. The only way to decrease the lattice size is by reducing $\tau^\star$ which in turn can cause stability issues, though these can be alleviated, at least in part, by using other collision models than BGK.

Starting from eq. (7.6), the relation of the simulation parameters in terms of Reynolds and Mach numbers can be written in the useful form

$$\mathrm{Re}^{(\mathrm{l})} = \frac{\ell^\star u^\star}{\nu^\star} = \frac{\ell^\star u^\star}{c_\mathrm{s}^{\star 2}\left(\tau^\star - \frac{1}{2}\right)} \quad \Longrightarrow \quad \frac{\mathrm{Re}^{(\mathrm{l})}}{\mathrm{Ma}^{(\mathrm{l})}} = \frac{\ell}{c_\mathrm{s}^\star\left(\tau^\star - \frac{1}{2}\right)\Delta x}. \tag{7.23}$$

Here, $\ell^\star = \ell/\Delta x$ is a typical system length scale in lattice units. In particular, $\ell^\star$ may be chosen as the number of lattice sites along one axis of the system. The total number of lattice sites for the three-dimensional space is therefore on the order of $\ell^{\star 3}$. For a target Reynolds number $\mathrm{Re}^{(\mathrm{l})}$ and a given system size ℓ, this equation allows to balance the simulation parameters in such a way that $\mathrm{Ma}^{(\mathrm{l})}$, $\tau^\star$ and Δx are under control. If all these parameters are set, one can find the time step from eq. (7.15) which requires knowledge of the physical viscosity ν.

It should be noted that $\mathrm{Ma}^{(\mathrm{l})}/\mathrm{Re}^{(\mathrm{l})}$ is the lattice Knudsen number Kn. For $\tau^\star = O(1)$, Kn is basically $\Delta x/\ell$. Hydrodynamic behaviour is only expected for sufficiently small Knudsen numbers. This sets an upper bound for the lattice constant Δx.

Parameter selection strategies

The first step is to set the lattice density $\rho^\star$. It plays a special role in LB simulations because it is a pure scaling parameter which can be arbitrarily chosen, without any (significant) effect on accuracy, stability or efficiency.[8] The most obvious choice is $\rho^\star = 1$, as already mentioned. Other values are possible, but there is no good reason to deviate from unity. Changing $\rho^\star$ leads to a change of the conversion factor for the density ($\rho^\star \neq 1$ leads to $C_\rho \neq \rho$), and it will affect the lattice value of every quantity whose dimension contains the mass (*e.g.* force, stress or energy).

A typical scenario is that there is a maximum lattice size $\ell^\star$ which can be handled by the computer. This suggests that the lattice constant Δx should be set first. The lattice Mach number (or the non-dimensional velocity $u^\star$) is then reasonably chosen, *e.g.* $\mathrm{Ma}^{(\mathrm{l})} = 0.1$ or $u^\star = 0.1$. For given system size $\ell^\star$, velocity $u^\star$ and Reynolds number, $\tau^\star$ can then be computed from Eq. (7.20). Now it has to be checked via eq. (7.19) whether the chosen values for $u^\star = \mathrm{Ma}^{(\mathrm{l})} c_\mathrm{s}^\star$ and $\tau^\star$ provide stable simulations. If $\tau^\star$ is too small, eq. (7.23) reveals that Δx should be decreased (increase $\ell^\star$) making simulations more expensive or $\mathrm{Ma}^{(\mathrm{l})}$ (increase $u^\star$) should be increased (less accurate, less stable). Once the parameters $u^\star$ and Δx are fixed, one can calculate Δt.

This scenario reveals some problems arising when large Reynolds numbers are simulated: it requires either large lattices, small relaxation parameters or large Mach numbers. It is possible only to a limited extent to reach large Reynolds numbers by increasing the Mach number and decreasing the relaxation parameter due to accu-

[8] Round-off errors may become important if $\rho^\star$ deviates too strongly from unity.

racy and stability issues. In the end, the only way is to increase the resolution which becomes progressively more computationally expensive.

Example 7.6. What is the maximum Reynolds number which can be achieved for a given lattice size? Assuming that we choose $\tau^\star$ near the stability limit, say $\tau^\star = \frac{1}{2} + \frac{1}{4}u^\star$, we find

$$\mathrm{Re}^{(\mathrm{l})} = \frac{\ell^\star u^\star}{c_\mathrm{s}^{\star 2}\left(\tau^\star - \frac{1}{2}\right)} = \frac{4\ell^\star}{c_\mathrm{s}^{\star 2}}. \tag{7.24}$$

This shows that the achievable Reynolds number is limited by $O(10)\,\ell^\star$.

It is also possible to set the Mach number (or $u^\star$) and viscosity $\nu^\star$ (or $\tau^\star$) first. Here eq. (7.19) has to be considered for stability reasons. Consequently, one can find the system size $\ell^\star$, and therefore the required lattice resolution Δx, and finally the time step Δt.

Another approach is to set the resolution Δx (or non-dimensional system size $\ell^\star$) and viscosity $\nu^\star$ first. Matching Re in physical and lattice systems gives the velocity $u^\star$ which needs to be less than 0.4.

Overall, it is up to the user which strategy is most convenient. In any case, it has to be checked whether the accuracy, stability and efficiency conditions are reasonably satisfied. We will discuss several concrete examples in section 7.3 to illustrate the parameter selection strategies.

Small Reynolds numbers

We have seen above that it can be difficult to reach large Reynolds numbers in simulations. However, there is also an intrinsic limitation when it comes to small Reynolds numbers.

Eq. (7.23) shows that a small Reynolds number can be reached by choosing a large Δx, a large relaxation parameter $\tau^\star$ or a small lattice Mach number $\mathrm{Ma}^{(\mathrm{l})}$. The resolution cannot be arbitrarily decreased; at some point the lattice domain is so small that the details of the flow are finer than the spacing between lattice nodes. It is not advisable to use $\tau^\star \gg 1$ as the numerical errors increase strongly with $\tau^\star$ (this can be avoided by using advanced collision operators such as TRT or MRT). Therefore, the only unbounded way to reduce $\mathrm{Re}^{(\mathrm{l})}$ is to decrease $\mathrm{Ma}^{(\mathrm{l})}$ and therefore the flow velocity $u^\star$. This in turn means that the time step Δt becomes very small because $\Delta t = \Delta x / C_u$ and $C_u \propto 1/\mathrm{Ma}^{(\mathrm{l})}$ so that $\Delta t \propto \mathrm{Ma}^{(\mathrm{l})}$.

Example 7.7. How small can the Reynolds number be for a given resolution? A typical flow geometry has an extension of about 100 lattice constants, *i.e.* $\ell^\star = 100$. Assuming that we do not want to use $\tau^\star > 2$, eq. (7.23) yields

$$\mathrm{Re}^{(\mathrm{l})} = u^\star \frac{100}{\frac{3}{2}c_\mathrm{s}^{\star 2}}. \tag{7.25}$$

Reaching $\mathrm{Re}^{(\mathrm{l})} = 0.01$ therefore requires $u^\star \approx 10^{-4}$. The velocity is relatively small, and it takes a large number of time steps and therefore computing time to follow the development of the flow field. The situation becomes even worse when the spatial resolution is larger or smaller Reynolds numbers are required.

At this point it is advisable to consider a violation of the Reynolds number mapping and the law of similarity. As explained before, the flow field is not sensitive to the Reynolds number as long as it is sufficiently small. One of the examples where Reynolds number is not important (used to speed up simulations) are capillary flows [13, 14] or microfluidic applications [15]. In these situations the time scale is not given by viscosity [16]. It is therefore suggested to use a numerical Reynolds number which is larger than the physical Reynolds number: $\mathrm{Re}^{(\mathrm{l})} > \mathrm{Re}^{(\mathrm{p})}$. A word of warning: it is very tempting to speed up simulations by violating the law of similarity. One should always check if the simulation results are still valid, *e.g.* by benchmarking against analytical results or varying the numerical Reynolds number.

Concluding our achievements in this chapter so far, we have seen that LB is particularly useful for flow problems with intermediate Reynolds numbers, especially in the range between $O(1)$ and $O(100)$ [17].

7.3 Examples

The examples collected in this section shall underline the relevance of the parameter selection process for LB simulations. Poiseuille (section 7.3.1, section 7.3.2 and section 7.3.3) and Womersley flow (section 7.3.4) are covered in detail. Furthermore, we talk about surface tension in the presence of gravity (section 7.3.5). It is recommended to read all examples in order of appearance since additional important concepts are introduced.

7.3.1 Poiseuille flow I

Consider a force-driven 2D Poiseuille flow. The physical parameters for channel diameter w, kinematic viscosity ν, density ρ and gravity g are $w = 10^{-3}\,\mathrm{m}$, $\nu = 10^{-6}\,\mathrm{m}^2/\mathrm{s}$, $\rho = 10^3\,\mathrm{kg}/\mathrm{m}^3$ and $g = 10\,\mathrm{m}/\mathrm{s}^2$. How should we choose the simulation parameters?

First of all we compute the expected centre velocity $\hat{u}$ of the flow. This is possible since the flow field is known analytically:

$$\hat{u} = \frac{g w^2}{8\nu} = 1.25\,\mathrm{m}/\mathrm{s}. \tag{7.26}$$

We define the Reynolds number as

$$\mathrm{Re} = \frac{\hat{u}w}{\nu} = 1\,250, \tag{7.27}$$

but other definitions are possible, *e.g.* by taking the average velocity which is $\bar{u} = \hat{u}/2$ for 2D Poiseuille flow.

We may now set the resolution first, *e.g.* $\Delta x = 5 \cdot 10^{-5}\,\mathrm{m}$, which corresponds to $w^\star = 20$. The lattice density is taken as $\rho^\star = 1$ and therefore $C_\rho = 10^3\,\mathrm{kg/m^3}$. Furthermore we choose $\tau^\star = 0.6$. The system is now fully determined (the user may start with a different set of initial values, though), and we can compute all dependent parameters.

Let us first obtain the conversion factor Δt by applying eq. (7.15):

$$\Delta t = c_\mathrm{s}^{\star 2}\left(\tau^\star - \tfrac{1}{2}\right)\frac{\Delta x^2}{\nu} = 8.33 \cdot 10^{-5}\,\mathrm{s}, \tag{7.28}$$

which means that 12 000 time steps are required for 1 s physical time. We can now compute the expected value of the lattice velocity.[9] This can be done by either using the law of similarity and the relation $\mathrm{Re}^{(\mathrm{p})} = \mathrm{Re}^{(\mathrm{l})} = \hat{u}^\star w^\star/\nu^\star$ or by computing the conversion factor for the velocity, $C_u = \Delta x/\Delta t = 0.6\,\mathrm{m/s}$. In either case we find $\hat{u}^\star = 2.08$ which is an invalid value because it is much larger than 0.4.

We have seen that the initially chosen parameters do not lead to a proper set of simulation parameters. This is very common, but one should not be discouraged. The invalidity of the simulation parameters could have been guessed before: the Reynolds number is nearly two orders of magnitude larger than the length scale $w^\star$ which is in conflict with the findings in eq. (7.24). This means that we have to increase the spatial resolution. Let us try a finer resolution ($\Delta x = 1 \cdot 10^{-5}\,\mathrm{m}$ rather than $5 \cdot 10^{-5}\,\mathrm{m}$) and a reduced viscosity ($\tau^\star = 0.55$ instead of 0.6). The same calculation as before now yields $\hat{u}^\star = 0.208$, which is acceptable. The user may choose yet another parameter set; there is no general reason for sticking with the values provided here.

In order to run the simulation, the lattice value for the force density $F = \rho g$ has to be obtained. We first compute the conversion factor for g: $C_g = \Delta x/\Delta t^2 = 3.6 \cdot 10^6\,\mathrm{m/s^2}$ where we have used the time conversion factor $\Delta t = 1.667 \cdot 10^{-6}\,\mathrm{s}$. Therefore, we get $g^\star = 2.78 \cdot 10^{-6}$ and $F^\star = \rho^\star g^\star = 2.78 \cdot 10^{-6}$. The simulation can now be performed and any results on the lattice can be mapped back to the physical system by using the known conversion factors.

We could have pursued alternative routes to find the simulation parameters, *e.g.*:

1. Choose $w^\star$ and $\hat{u}^\star$, which gives Δx and C_u.
2. Find the viscosity conversion factor $C_\nu = \Delta x^2/\Delta t = \Delta x C_u$ and therefore $\nu^\star$.
3. Compute $\tau^\star$ from $\nu^\star$.
4. Find the remaining conversion factors and check the validity of the parameters.

Another different approach is this:

1. Select $\hat{u}^\star$ and $\tau^\star$ and therefore C_u and $\nu^\star$.

[9] If there is no analytical solution for the problem, one may run a test simulation and extract $\hat{u}^\star$.

2. Find the viscosity conversion factor C_ν.
3. Compute the lattice resolution $\Delta x = C_\nu / C_u$ and then $w^\star$ (if an integer value for $w^\star$ is required, the other parameters have to be slightly adapted).
4. Calculate all other required conversion factors and check the validity of the parameters.

Exercise 7.3. Find conversion factors for the above example by first selecting reasonable values for $\hat{u}^\star$ and $\tau^\star$.

If the initial guess gives invalid or otherwise unacceptable results, one has to modify one or two parameters (which are not necessarily the initially chosen parameters) while updating the dependent parameters until the desired level of accuracy, stability and efficiency is obtained. The Reynolds number has to be kept invariant, which can be enforced by exploiting eq. (7.23). This can be a frustrating process because one may find that the lattice becomes too large or the time step too small. However, this *a priori* analysis is necessary to assess whether the simulation is feasible at all. The alternative is to run a series of simulations blindly and adapt the parameters after each run. This very time consuming approach should be avoided.

After some trial and error the user will be able to make educated guesses for the initial parameter set based on the intrinsic limitations of the LB algorithm. It is absolutely justified to say that the parameter selection for LB simulations is an art which can be practised. We particularly recommend [16] for further reading.

7.3.2 Poiseuille flow II

In the previous example, the simulation parameters for a Poiseuille flow have been obtained after the physical set of parameters had been defined. Thanks to the law of similarity there is, however, another approach: one can set up simulations without any given dimensional parameter! For the Poiseuille flow it is absolutely sufficient to know the target Reynolds number.

Let us pursue this idea here. We assume that we only know the Reynolds number (*e.g.* Re = 1250 as in the previous example) but nothing else. We can then take advantage of eq. (7.23) (where $u^\star$ has to be replaced by $\hat{u}^\star$) and start with an initial guess for any two independent parameters. Since we know that the Reynolds number is relatively large, we may want to run a well-resolved simulation with $w^\star = 100$. We may wish to limit the velocity to $\hat{u}^\star = 0.1$. From this we can directly compute $\tau^\star = 0.524$, which should yield a stable simulation (*cf.* eq. (7.19)). The required force density $F^\star = \rho^\star g^\star$ can then be obtained from

$$\hat{u}^\star = \frac{g^\star w^{\star 2}}{8\nu^\star} \quad \Longrightarrow \quad g^\star = 6.4 \cdot 10^{-7} \tag{7.29}$$

after $\rho^\star$ has been chosen ($\rho^\star = 1$ in this case).

An interesting consequence is that we have all required simulation parameters, but not a single conversion factor. This is not a problem at all because the simulation

is valid for all similar physical systems, *i.e.* for all systems with the same Reynolds number. The conversion factors can be obtained *a posteriori* by setting the physical scales *after* a successful simulation. For example, if w is known, one can compute Δx. The full set of conversion factors can only be obtained when three independent physical parameters are given (*e.g.* w, $\hat{u}$, ρ or F, ρ, w).

In physics it is often the case that systems are only characterised by the relevant dimensionless parameters without giving the scales like length or velocity themselves. This is sufficient to set up and run simulations, at least as long as *all* required dimensionless parameters are known. This leads to the important question how many dimensionless parameters have to be known for a given physical system. The answer will be given in section 7.3.4.

7.3.3 Poiseuille flow III

It is very common to drive a Poiseuille flow by a pressure gradient rather than by gravity or a force density. The required boundary conditions for pressure-driven flow are provided in section 5.2.5. Here, we address how to obtain the required pressure values at the inlet and outlet and which implications this brings along.

First we note that the pressure gradient $p' = \Delta p/\ell$ (where $\Delta p = p_{\text{in}} - p_{\text{out}} > 0$ is the pressure difference between inlet and outlet and ℓ is the length of the channel) and the force density F is simply $p' = F$. From eq. (7.29) we immediately find

$$\hat{u}^\star = \frac{\Delta p^\star}{\rho_0^\star}\frac{w^{\star 2}}{8\ell^\star \nu^\star} \quad \Longrightarrow \quad \frac{\Delta\rho^\star}{\rho_0^\star} = \frac{8\ell^\star\nu^\star\hat{u}^\star}{c_{\text{s}}^{\star 2} w^{\star 2}} \tag{7.30}$$

for a 2D Poiseuille flow.[10] We have introduced the *average* density $\rho_0^\star$ to appreciate the fact that the density is not constant. In the second step, the pressure difference $\Delta p^\star$ has been replaced by the density difference *via* $c_{\text{s}}^{\star 2}\Delta\rho^\star$. Eq. (7.30) provides a direct expression for the required relative difference between the inlet and outlet *densities*. In the incompressible limit, we require $\Delta\rho^\star \ll \rho_0^\star$, and the right-hand-side of this equation has to be a small quantity. This is another restriction for an LB simulation and sets additional bounds, especially for the system length $\ell^\star$. Note that this problem does not exist for a force-driven flow.

Example 7.8. How long can we choose a channel for a typical set of simulation parameters ($\hat{u}^\star = 0.1$, $\nu^\star = 0.1$, $w^\star = 50$, $c_{\text{s}}^{\star 2} = \frac{1}{3}$)? Assuming that we allow density variations of up to 5% ($\Delta\rho^\star/\rho_0^\star = 0.05$), the maximum channel length becomes $\ell^\star \approx 520$. This is only ten times more than the channel diameter. If longer channels are required, the user has to re-balance the simulation parameters on the right-hand-side of eq. (7.30), keeping all relevant dimensionless parameters (here only the Reynolds number) invariant. In the end, this leads to an increased spatial resolution which allows larger ratios of $\ell^\star/w^\star$ but requires more expensive simulations.

[10] The numerical prefactor becomes 4 in a 3D Poiseuille flow with circular cross-section.

How can we now choose the inlet and outlet densities in lattice units? If $\Delta\rho^\star$ and the average density $\rho_0^\star$ are known, we set

$$\rho_{\text{in}}^\star = \rho_0^\star + \frac{\Delta\rho^\star}{2} \quad \text{and} \quad \rho_{\text{out}}^\star = \rho_0^\star - \frac{\Delta\rho^\star}{2}. \tag{7.31}$$

Obviously, $\Delta\rho^\star = \rho_{\text{in}}^\star - \rho_{\text{out}}^\star$ is satisfied. In turn, the local pressure $p^\star$ can be obtained from the local density $\rho^\star$ according to

$$p^\star = p_0^\star + c_{\text{s}}^{\star 2}\left(\rho^\star - \rho_0^\star\right). \tag{7.32}$$

The constant reference pressure $p_0^\star$ does not affect the simulation and is chosen by the user as discussed in eq. (7.18).

7.3.4 Womersley flow

Womersley flow denotes a Poiseuille-like flow (channel width w, viscosity ν) with an oscillating pressure drop along the length ℓ of the channel: $\Delta p(t) = \Delta p_0 \cos\omega t$ with frequency ω. Since there exists an analytical solution for this unsteady flow, it is an ideal benchmark for Navier-Stokes solvers. Hence, Womersley flow has often been simulated with the LB algorithm (*e.g.* [18, 19, 20]). We will not show the analytical solution and refer the reader to the literature instead. Here we will rather consider the generic implications arising from simulating non-steady flows.

The Reynolds number for an oscillatory flow is usually defined through the velocity one would observe for $\omega = 0$: $\text{Re} = \hat{u}_0 w/\nu$ where $\hat{u}_0$ is related to Δp_0 according to eq. (7.30):

$$\hat{u}_0 = \frac{\Delta p_0 w^2}{8\ell\rho\nu} \tag{7.33}$$

for a 2D channel. Due to the presence of the frequency ω — which is obviously zero for simple Poiseuille flow — an additional dimensionless number is required to characterise the flow. There are different ways to construct such a number: we just have to write down a dimensionless combination containing ω and other suitable parameters. One possibility is $\omega w^2/\nu$. The *Womersley number* is defined as the square root

$$\alpha = \sqrt{\frac{\omega}{\nu}} w. \tag{7.34}$$

Before we continue with the discussion of unsteady flows in LB simulations, let us first address the issue of finding dimensionless numbers for a physical system. The *Buckingham π theorem* [1] states that for Q independent quantities whose physical dimension can be constructed from D independent dimensions there are $N = Q - D$ independent dimensionless parameters. For example, for a simple Poiseuille flow, there are $Q = 4$ independent quantities: channel width, flow velocity, viscosity and density. We know that any mechanical system is characterised

by $D = 3$ independent dimensions: length, time and mass. This gives $N = 1$ parameter which is the Reynolds number. For each additional physical parameter, one dimensionless number is required. Womersley flow is characterised by $Q = 5$ quantities (the four Poiseuille parameters and the oscillation frequency) and therefore $N = 2$ dimensionless numbers.

Another very helpful way to look at this is the definition of characteristic time scales and their ratios. In a Poiseuille flow, there are two time scales: the advection time $t_\mathrm{a} \sim w/\hat{u}$ and the diffusion or viscous time $t_\nu \sim w^2/\nu$. $N + 1$ such time scales define N independent time scale ratios (*e.g.* $\mathrm{Re} \sim t_\nu/t_\mathrm{a}$ for the Poiseuille flow). For Womersley flow, a third time scale is given by $t_\omega \sim 1/\omega$. A second suitable dimensionless number is therefore t_ω/t_ν or t_ω/t_a or appropriate combinations thereof (like $\alpha \sim \sqrt{t_\nu/t_\omega}$). Any additional dimensionless number would not be independent. For example, one can also define the *Strouhal number* $\mathrm{St} = \omega w/(2\pi\hat{u}_0) \sim t_\mathrm{a}/t_\omega$. It can be expressed as a combination of Reynolds and Womersley numbers: $\alpha \sim \sqrt{\mathrm{Re}\,\mathrm{St}}$. Womersley flow is usually characterised by Re and α.

Let us now get back to the LBM. When setting up a Womersley flow (or any other unsteady flow) in an LB framework, one has to be aware of the subtleties of the LB algorithm. As LB is a local algorithm without a Poisson equation for the pressure, any information on the lattice propagates with a finite velocity comparable to the speed of sound. In particular, this means that all physical time scales which shall be resolved *must* be sufficiently long compared to the intrinsic sound wave (or acoustic) time scale $t_\mathrm{s}^\star \sim \ell^\star/c_\mathrm{s}^\star$. In other words, the acoustic time scale $t_\mathrm{s}^\star$ on the one hand and the advection, diffusion and oscillation time scales ($t_\mathrm{a}^\star$, $t_\nu^\star$ and $t_\omega^\star$) on the other hand have to be sufficiently separated (*cf.* section 3.1 in [21]) as long as one is interested in incompressible flows: $t_\mathrm{a}^\star, t_\nu^\star, t_\omega^\star \gg t_\mathrm{s}^\star$. This defines, for a given lattice size, a lower bound for the oscillation time scale and therefore an upper bound for the frequency (here we assume that $\ell^\star > w^\star$):

$$\omega^\star \ll \frac{c_\mathrm{s}^\star}{\ell^\star}. \tag{7.35}$$

Any unsteady incompressible flow violating this condition is non-physical, and the results will not be a good approximation of the incompressible Navier-Stokes solution.

Having clarified this, we provide a suggested procedure to set up Womersley flow with known Re and α:

1. Select initial values for $\ell^\star$ and $w^\star$ which are compatible with the Reynolds number, *e.g.* $w^\star > 0.1\,\mathrm{Re}$ (stability).
2. Estimate the sound propagation time scale $t_\mathrm{s}^\star \sim \ell^\star/c_\mathrm{s}^\star$ and therefore the recommended maximum for $\omega^\star$ via $2\pi/\omega^\star = T^\star \gg t_\mathrm{s}^\star$. It requires some trial and error to find a sufficient minimum ratio $T^\star/t_\mathrm{s}^\star$, but one should at least use a factor of ten.
3. Balance $\omega^\star$ and $\nu^\star$ to match α via eq. (7.34) by taking into account the law of similarity for the Womersley number: $\alpha^{(\mathrm{l})} = \alpha^{(\mathrm{p})}$. There is no unique way to

choose $\omega^\star$ and $\nu^\star$, but keep in mind already now that the relaxation parameter $\tau^\star$ should not be too close to $\frac{1}{2}$ or much larger than unity.
4. Estimate the velocity $\hat{u}_0^\star$ from Re using the selected values for $w^\star$ and $\nu^\star$.
5. Check the validity of $\hat{u}_0^\star$: if its value is not in the desired range (too large for reasonable stability or accuracy or too small for a feasible time step), the parameters have to be re-balanced while keeping Re and α invariant. This process is more complicated than for the Poiseuille flow because Re and α have to be considered simultaneously.
6. The required pressure difference $\Delta p_0^\star$ finally can be obtained from eq. (7.33).

A parameter optimisation scheme for unsteady LB simulations is thoroughly discussed in [19].

Example 7.9. Let us consider a flow with Re = 1000 and $\alpha = 15$ which is typical for blood flow in the human aorta. First we choose $\rho^\star = 1$, $\ell^\star = 500$ and $w^\star = 100$ and find that $T^\star \gg 870$ should hold. We choose $T^\star = 8700$ and therefore $\omega^\star = 7.22 \cdot 10^{-4}$. Hence, we get $\nu^\star = \omega^\star w^{\star 2}/\alpha^2 = 0.0321$ and $\tau^\star = 0.596$ for a lattice with $c_s^{\star 2} = \frac{1}{3}$. This is a reasonable value for the relaxation parameter. We find $\hat{u}_0^\star = \mathrm{Re}\,\nu^\star/w^\star = 0.321$, though, which seems quite large. However, the maximum velocity actually observed in a Womersley flow is always smaller than $\hat{u}_0^\star$, especially for $\alpha > 1$. The reason is that the flow lags behind the pressure gradient and does not have enough time to develop fully. The required pressure difference follows from eq. (7.33) and yields $\Delta p_0^\star = 4.12 \cdot 10^{-3}$ which is small compared to order unity. We can therefore assume that the present set of simulation parameters is suitable for a simulation of Womersley flow, but one may want to decrease the time step further to obtain a better separation of time scales.

LB simulations are usually subjected to initial unphysical or at least undesired transients (*cf.* section 5.4) which decay after some time. As a rule of thumb, the transient length in channel flow is given by the diffusion time scale $t_\nu = w^2/\nu$; for the above example we find $t_\nu^\star \approx 3.11 \cdot 10^5 \approx 36\,T^\star$! It may therefore be necessary to simulate about 40 full oscillations before the numerical solution converges. The actual transient length depends on the simulated flow and the chosen initial conditions. It is recommended to employ an on-the-fly convergence algorithm which aborts the simulation after a given convergence criterion has been satisfied.[11] Under the assumption that 40 oscillation periods are required, the simulation will run for 348 000 time steps. Further assuming a relatively slow D2Q9 code with 2 million site updates per second and taking the lattice size $\ell^\star \times w^\star = 500 \times 100$, the total simulation runtime will be 8 700 seconds or nearly three hours. A comparable 3D simulation would be roughly 100 times more expensive (assuming a circular cross-section with 100 lattice sites across the diameter), which is unfeasible for a serial code.

[11] One may check for temporal convergence by comparing the velocity profiles at times $t^\star$ and $t^\star - T^\star$ and applying a suitable error norm.

7.3.5 Surface tension and gravity

Let us now consider a droplet of a liquid in vapour (or the other way around: a vapour bubble in a liquid). The surface tension of the liquid-gas interface is γ. The droplet/bubble may be put on a flat substrate or into a narrow capillary. For the implementation of such a system, we refer to chapter 9. Here, we focus on the unit conversion in presence of surface tension and gravity, which is independent of the underlying numerical model.

The density difference of the liquid and vapour phases is defined as $\Delta\rho = \rho_\mathrm{l} - \rho_\mathrm{v} > 0$. In the presence of the gravitational acceleration with magnitude g we can define the dimensionless *Bond number*

$$\mathrm{Bo} = \frac{\Delta\rho g r^2}{\gamma}. \tag{7.36}$$

It characterises the relative strength of gravity and surface tension effects where r is a length scale typical for the droplet/bubble. A common definition for r is the radius of a sphere with the same volume V as the droplet/bubble[12]:

$$r = \left(\frac{3V}{4\pi}\right)^{1/3}. \tag{7.37}$$

A small Bond number means that gravity is negligible and the droplet/bubble shape is dominated by the surface tension. This will result in a spherical shape of the droplet/bubble if it is not in contact with any wall or a shape like a spherical cap if it is attached to a wall. For larger Bond numbers, gravity is important, and the droplet/bubble is generally deformed. This is illustrated in fig. 7.1.

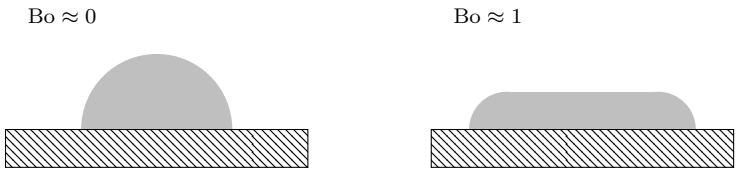


Fig. 7.1: Illustration of typical shapes of a droplet on a substrate (a) for small Bond number (Bo $\ll$ 1) with negligible gravity and (b) large Bond number (Bo $\gg$ 1) where gravity is important. In both cases the contact angle is 90 °. More details about contact angles are given in chapter 9 TODO (TK): more precise reference required.

[12] This definition appreciates the fact that the droplet/bubble may be deformed and therefore have a more complex shape than a section of a sphere.

How do we find the simulation parameters in lattice units for a given Bond number? The first step is to set the lattice densities which are usually tightly related to the numerical model (*cf.* chapter 9). Let us take the liquid density as reference and write

$$\rho_{\mathrm{v}}^{\star} = \frac{1}{\lambda}\rho_{\mathrm{l}}^{\star} =: \frac{1}{\lambda}\rho^{\star} \tag{7.38}$$

where $\lambda > 1$ is the achievable density ratio of the model. The next step is to choose the radius $r^{\star}$ properly. In lattice-based methods, interfaces between phases are always subject to a finite width of a few Δx ($d^{\star} = O(3)$) as will be detailed in chapter 9. We have to make sure that the radius $r^{\star}$ is significantly larger than this interface width, $r^{\star} \gg d^{\star}$, otherwise the simulated system will be dominated by undesired diffuse interface properties. Typical recommended thresholds are $r^{\star} \geq 8$, but one may obtain decent results for smaller resolutions. This is an important additional restriction in multi-phase or multi-component LB models.

Separating the known and unknown parameters, we get

$$\frac{\gamma^{\star}}{g^{\star}} \approx \frac{\rho^{\star} r^{\star 2}}{\mathrm{Bo}^{(\mathrm{l})}} \tag{7.39}$$

where we have approximated $\Delta\rho \approx \rho_{\mathrm{l}} = \rho$, which is valid for $\lambda \gg 1$. Furthermore, we have taken advantage of the law of similarity in the form $\mathrm{Bo}^{(\mathrm{l})} = \mathrm{Bo}^{(\mathrm{p})}$. All quantities on the right-hand-side of eq. (7.39) are known. This leaves us with one degree of freedom: it is up to the user to choose $\gamma^{\star}$ and $g^{\star}$ in such a way that eq. (7.39) is satisfied. On the one hand, based on the underlying LB model, the surface tension may be restricted to a certain numerical range (similar to the restriction of $\tau^{\star}$). On the other hand, if $g^{\star}$ is chosen too big, the lattice acceleration and therefore velocity may be too large, and the simulation is less accurate or even unstable. Since the time step is effectively defined by $g^{\star}$ via

$$g = g^{\star}\frac{\Delta x}{\Delta t^2}, \quad \Delta x = \frac{r}{r^{\star}}, \tag{7.40}$$

the user has to consider the total runtime of the simulation as well. Reducing $\gamma^{\star}$ and $g^{\star}$ will decrease the time step and result in a longer computing time.

We have not said anything about viscosity. The reason is that, so far, we have only taken advantage of the Bond number scaling. If for example the Reynolds number is a relevant parameter as well, *i.e.* if Re is not small, the simulation parameters have to be chosen simultaneously.

Example 7.10. Liquid glycerol with density $\rho_{\mathrm{l}} = 1\,260\,\mathrm{kg/m^3}$ and kinematic viscosity $\nu = 8.49 \cdot 10^{-4}\,\mathrm{m^2/s}$ (dynamic viscosity $\eta = 1.07\,\mathrm{Pa\,s}$) is flowing in a vertical pipe of diameter $w = 0.015\,\mathrm{m}$ in the gravitational field $g = 9.81\,\mathrm{m/s^2}$. We want to simulate an air bubble in the liquid with radius $r = 4 \cdot 10^{-3}\,\mathrm{m}$ and large density contrast, $\lambda \gg 1$. The Reynolds number is defined according to the flow we would observe in the absence of the bubble (*cf.* section 7.3.1):

$$\mathrm{Re} = \frac{\hat{u}w}{\nu} = \frac{gw^3}{4\nu^2} = 11.5. \tag{7.41}$$

The surface tension of glycerol in air at 20 °C is $\gamma = 6.34 \cdot 10^{-2}\,\mathrm{N/m}$ resulting in Bo = 3.12. Note that the *confinement* $\chi := 2r/w = 0.533$ is also a relevant dimensionless parameter which is, however, easily mapped to the lattice as it is merely the ratio of two length scales. How should we select the simulation parameters? We start by setting $w^\star = 30$ and $r^\star = 8$ obeying the confinement scaling. The Reynolds number $\mathrm{Re}^{(\mathrm{p})} = \mathrm{Re}^{(\mathrm{l})}$ therefore restricts the ratio of gravity and viscosity:

$$\frac{g^\star}{\nu^{\star 2}} = \frac{4\,\mathrm{Re}}{w^{\star 3}} = 1.70 \cdot 10^{-3}. \tag{7.42}$$

We choose $\rho^\star = 1$ and find from the Bond number:

$$\frac{\gamma^\star}{g^\star} = \frac{\rho^\star r^{\star 2}}{\mathrm{Bo}^{(\mathrm{l})}} = 20.5. \tag{7.43}$$

We have one degree of freedom left because the choice of the three parameters $g^\star$, $\nu^\star$ and $\gamma^\star$ is restricted by two conditions. We may now set the surface tension first: $\gamma^\star = 0.06$. This gives $g^\star = 2.93 \cdot 10^{-3}$ and therefore $\nu^\star = 1.31$ and $\tau^\star = 4.44$. Additionally, we find the maximum velocity $\hat{u}^\star = 0.50$. These parameters are not acceptable, and the velocity and viscosity have to be reduced. There are different strategies:

1. Keep $\gamma^\star$ fixed. This leads to the scalings $g^\star \propto r^{\star -2}$, $\nu^\star \propto r^{\star 1/2}$ and $\hat{u}^\star \propto r^{\star -1/2}$. Showing these relations is left as an exercise for the reader. We notice that we can either reduce the viscosity or the velocity by changing the resolution, but we cannot reduce both at the same time. It is therefore not possible to keep the surface tension unchanged.
2. Keep $r^\star$ fixed. Now we get $g^\star \propto \gamma^\star$, $\nu^\star \propto \gamma^{\star 1/2}$ and $\hat{u}^\star \propto \gamma^{\star 1/2}$. By reducing the surface tension we can therefore reduce the viscosity and the velocity simultaneously. But it depends on the selected numerical model if a decrease of surface tension is feasible at all.

The two above-mentioned approaches are usually combined to further balance the simulation parameters. One has to keep in mind, though, that every numerical method has intrinsic limitations which make certain combinations of dimensionless numbers (here: Re, Bo, χ) unfeasible if not impossible to access in a simulation.

7.4 Summary

We conclude this chapter by collecting the relevant results about unit conversion and parameter selection.

Table 7.2: Overview of intrinsic LB limitations and suggested remedies.

accuracy	1) $\tau^\star$ not much larger than unity 2) $u^\star_{\max} < 0.3$ 3) Δx sufficiently fine to resolve all flow features 4) Δt sufficiently fine to reduce time discretisation artifacts
stability	1) $u^\star_{\max} < 0.4$ 2) $\tau^\star$ not too close to $\frac{1}{2}$, *cf.* eq. (7.19); 3) sufficiently small grid Reynolds number, *cf.* eq. (7.20)
efficiency	lattice constant Δx and time step Δt not unnecessarily small (memory requirements $\propto \Delta x^{-d}$, simulation runtime $\propto \Delta t^{-1}\Delta x^{-d}$ in d dimensions)
interface width	droplet/bubble radius significantly larger than interface width: $r^\star \gg d^\star = O(3)$

- Purely mechanical systems require exactly three conversion factors. The first step is to define three independent (basic) ones. All other conversion factors can be derived according to eq. (7.10).
- Some dimensions (*e.g.* for pressure or density) are different in 2D and 3D. This problem can be circumvented by interpreting a 2D system as a 3D system with thickness Δx.
- The law of similarity (*e.g.* for the Reynolds, Womersley or Bond number) is the physical basis for consistent non-dimensionalisation: two systems with the same set of relevant dimensionless parameters are similar.
- Unit systems must not be mixed as this will lead to inconsistencies and wrong physical results.
- Typical basic conversion factors for LB simulations are those for length, time (or velocity) and density: C_ℓ, C_t (or C_u) and C_ρ. Other sets are possible but usually less practical.
- Relevant simulation parameters are the lattice spacing Δx, the time step Δt, the relaxation parameter $\tau^\star$, the average density $\rho^\star$ and the characteristic flow velocity $u^\star$ (a star * denotes non-dimensionalised parameters). They are not independent. Their relations are partially dictated by characteristic dimensionless numbers such as the Reynolds number. For example, the physical kinematic viscosity ν poses a condition for $\tau^\star$, Δx and Δt:

$$\nu = c_s^{\star 2}\left(\tau^\star - \tfrac{1}{2}\right)\frac{\Delta x^2}{\Delta t}. \tag{7.44}$$

- Choose simulations parameters in such a way that the intrinsic LB restrictions are considered (*cf.* tab. 7.2).
- The LB algorithm is second-order accurate in space and first-order accurate in time when choosing the diffusive scaling $\Delta t \propto \Delta x^2$. This is the preferred scaling for resolution refinement if compressibility effects are unimportant (otherwise the acoustic scaling $\Delta t \propto \Delta x$ is the way to go).
- For incompressible simulations the Mach number is an unimportant parameter, as long as it is not too large. The correct scaling of the Mach number is therefore not necessary and should actually be avoided to increase the simulation efficiency.

Another handy relation between the simulation parameters is given by

$$\frac{\mathrm{Re}^\star}{\mathrm{Ma}^\star} = \frac{\ell}{c_\mathrm{s}^\star \left(\tau^\star - \frac{1}{2}\right) \Delta x} \tag{7.45}$$

where $\mathrm{Re}^\star = \ell^\star u^\star / \nu^\star$ and $\mathrm{Ma}^\star = u^\star / c_\mathrm{s}^\star$.

- It is recommended to set the lattice density $\rho^\star$ first. There is usually no good reason to deviate from $\rho^\star = 1$.
- LB simulations can be performed without specifying any conversion factors. The results can later be mapped to a physical system with the same dimensionless parameters. In particular, one can find all LB parameters from the dimensionless parameters without the necessity to specify absolute scales.
- Any LB simulation for incompressible flows has to be set up in a way such that the local density variation $\delta\rho^\star$ is small compared to $\rho^\star$.
- The number of independent simulation parameters is reduced by each imposed dimensionless parameter. Simulation parameters have to be chosen such that all relevant dimensionless numbers are correct, *i.e.* that all laws of similarity are satisfied. This is tightly related to the Buckingham π theorem and can also be understood from defining all relevant time scales and their ratios. For example, the Reynolds number is the ratio of the characteristic viscous and advection time scales. The Strouhal number is the ratio of the advection and the oscillation time scale of an oscillating flow.
- Any relevant time scale (*e.g.* advection, diffusion, oscillation) has to be significantly larger than the sound propagation time scale $t_\mathrm{s}^\star \sim \ell^\star / c_\mathrm{s}^\star$. Otherwise the physical properties of the system change faster than the simulation can adapt.

References

1. E. Buckingham, Phys. Rev. **4**, 345 (1914)
2. L.D. Landau, E.M. Lifshitz, *Fluid Mechanics* (Pergamon Press, 1987)
3. E.M. Viggen, Phys. Rev. E **90**, 013310 (2014)
4. Y.T. Feng, K. Han, D.R.J. Owen, Int. J. Numer. Meth. Eng. **72**(9), 1111–1134 (2007)
5. P.A. Skordos, Phys. Rev. E **48**(6), 4823 (1993)
6. D.J. Holdych, D.R. Noble, J.G. Georgiadis, R.O. Buckius, J. Comput. Phys. **193**(2), 595 (2004)
7. S. Ubertini, P. Asinari, S. Succi, Phys. Rev. E **81**(1), 016311 (2010)
8. P.J. Dellar, Comput. Math. Appl. **65**(2), 129 (2013)
9. D. d'Humières, I. Ginzburg, Comput. Math. Appl. **58**, 823 (2009)
10. T. Krüger, F. Varnik, D. Raabe, Phys. Rev. E **79**(4), 046704 (2009)
11. I. Ginzburg, D. d'Humieres, A. Kuzmin, J. Stat. Phys. **139**, 1090 (2010)
12. J.C.G. Verschaeve, Phys. Rev. E **80**, 036703 (2009)
13. A. Kuzmin, M. Januszewski, D. Eskin, F. Mostowfi, J. Derksen, Chem. Eng. J. **171**, 646 (2011)
14. A. Kuzmin, M. Januszewski, D. Eskin, F. Mostowfi, J. Derksen, Chem. Eng. J. **178**, 306 (2011)
15. T. Krüger, D. Holmes, P.V. Coveney, Biomicrofluidics **8**(5), 054114 (2014)
16. M.E. Cates, J.C. Desplat, P. Stansell, A.J. Wagner, K. Stratford, R. Adhikari, I. Pagonabarraga, Philos. T. Roy. Soc. A **363**(1833), 1917 (2005)

17. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, 2001)
18. X. He, L.S. Luo, J. Stat. Phys. **88**(3-4), 927 (1997)
19. A.M.M. Artoli, A.G. Hoekstra, P.M.A. Sloot, Comput. Fluids **35**(2), 227 (2006)
20. R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, Phys. Rev. E **89**(2), 023303 (2014)
21. B. Dünweg, A.J.C. Ladd, in *Advances in Polymer Science* (Springer Berlin Heidelberg, 2008), pp. 1–78

Chapter 8
Lattice-Boltzmann for advection-diffusion problems

TODO (TK): Write chapter introduction when chapter is finished.

8.1 Advection-diffusion problems

Advection and diffusion are two common phenomena that can be observed in daily life and many hydrodynamic problems. For example, oil spilled in a river is dragged along with the water, therefore moving downstream with the current; this is an *advection* effect. Clouds moving in the sky are another example of advection: the clouds travel along with the wind in the atmosphere due to the drag force between both.

Diffusion is slightly less intuitive. At finite temperature, molecules show random motion, even if the average velocity in a medium is zero. A famous effect is the *Brownian motion* of dust particles on top of a water surface. The particles are constantly hit by water molecules, which leads to characteristic diffusive trajectories. Although the average velocity vanishes, the mean square displacement of the dust particles increases linearly with time. Two miscible fluids, for instance ethanol and water, mix themselves when put into the same container. The reason is that both molecule species diffuse until the concentration of both water and alcohol is constant.

Thermal diffusion is relevant in many industrial applications, for instance to keep the temperature in a reactor as constant and homogeneous as possible. Local heat sources, *e.g.* due to chemical reactions, or heat sinks, *e.g.* caused by the presence of cold walls, cause a temperature gradient. The diffusion of heat then tends to oppose the temperature gradient in the system.

Diffusion is often a slow process, in particular when the physical system is large. Since the mean square displacement grows linearly with time, diffusion is characterised by a $\sqrt{t}$ behaviour. Advection with a constant velocity u, however, scales linearly with time since the distance travelled at constant velocity is ut. It therefore

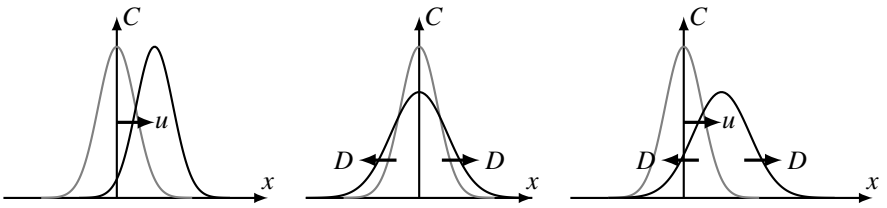


Fig. 8.1: Illustration of pure advection (left), pure diffusion (middle) and advection-diffusion (right). u and D are the advection velocity and diffusion coefficients, respectively. The grey curve is an initial concentration distribution $C(x, t = 0)$, the black curve shows the concentration at a later time.

often depends on the spatial scale whether advection or diffusion dominate. We will pick up this idea again shortly.

The *advection-diffusion equation* (ADE) or *convection-diffusion equation* (CDE) for a scalar field C (*e.g.* the concentration of a chemical species or temperature) with isotropic diffusion coefficient D and source term q reads

$$\frac{\partial C}{\partial t} + \nabla \cdot (C\boldsymbol{u}) = \nabla \cdot (D \nabla C) + q. \tag{8.1}$$

TODO (TK): Give some general reference. Eq. (8.1) shows that there are three mechanisms that lead to a local change of C:

- *Advection* is caused be a prescribed advection velocity $\boldsymbol{u}$. This could be the ambient flow velocity of a fluid.
- The scalar field shows intrinsic *diffusion* according to the divergence of the term $\boldsymbol{j} = -D\nabla C$. This linear relation between the *diffusion flux* $\boldsymbol{j}$ and the gradient ∇C is called *Fick's first law*. We see that diffusion is generally driven by the gradient of C. D is a material property that is normally temperature-dependent. Its physical unit is m^2/s.
- C may be locally produced or destroyed as indicated by the *source* term q. One possible mechanism is a chemical reaction that releases heat or consumes a given chemical species.

Advection and diffusion are illustrated in fig. 8.1.

If we assume that D is homogeneous, we can simplify eq. (8.1) further:

$$\frac{\partial C}{\partial t} + \nabla \cdot (C\boldsymbol{u}) = D\nabla^2 C + q. \tag{8.2}$$

There are certain situations where D is non-homogeneous (so that eq. (8.1) applies) or non-isotropic (where D becomes a direction-dependent diffusion tensor $\boldsymbol{D}$). TODO (TK): Give a few examples. If not otherwise stated, we only consider the simplified ADE in eq. (8.2).

Eq. (8.2) is general in the sense that it describes not only a concentration C, but for example also the temperature T. In this case, one often writes

$$\frac{\partial T}{\partial t} + \nabla \cdot (T\boldsymbol{u}) = \kappa \nabla^2 T + q \tag{8.3}$$

where κ is the *thermal diffusivity*. To highlight the generality of the ADE, we continue writing C and keep in mind that C may be replaced by T (and D by κ) if we want to consider thermal problems.

Like the Reynolds number for the NSE, we can define a characteristic dimensionless number for advection-diffusion problems: the *Péclet number*. Introducing a characteristic velocity U, length L and time $T = L/U$, we can rewrite eq. (8.2) without source term q as

$$\frac{\partial C}{\partial t'} + \nabla' \cdot (C\boldsymbol{u}') = \frac{D}{LU} \nabla'^2 C \tag{8.4}$$

where primed variables are dimensionless, *e.g.* $t' = t/T$ and $\nabla' = L\nabla$. The dimensionless prefactor on the right-hand side is the inverse Péclet number:

$$\mathrm{Pe} = \frac{LU}{D}. \tag{8.5}$$

Problems with large Péclet number are advection-dominated, *i.e.* the change of C due to the advection velocity $\boldsymbol{u}$ is more important than diffusive contributions. In the limit $\mathrm{Pe} \to 0$, for small or vanishing velocity or at small length scales, eq. (8.2) is dominated by diffusion.

There exist different numerical methods to solve advection-diffusion problems. The most commonly used is the finite-difference method (section 2.1.1). TODO (TK): More references for further reading? In the remainder of this chapter, we will focus on the LBM as advection-diffusion solver.

8.2 Lattice-Boltzmann for advection-diffusion

We show how the lattice-Boltzmann method can be used to simulate advection-diffusion problems. The governing equations are presented in section 8.2.1. This requires a modified equilibrium distribution function (section 8.2.2) and some comments about the lattice velocities (section 8.2.3). The Chapman-Enskog analysis in section 8.2.4 is similar to the known analysis of the standard lattice-Boltzmann method for fluid dynamics. We conclude the section by mentioning a few extensions of the lattice-Boltzmann advection-diffusion model (section 8.2.5).

8.2.1 Lattice-Boltzmann equation for advection-diffusion problems

There are different approaches to construct an LB algorithm for advection-diffusion problems. A straightforward way is to start from the incompressible NSE and rewrite it as

$$\frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u} + p\mathbf{1}) = \nu \nabla^2(\rho \boldsymbol{u}) + \boldsymbol{F}. \tag{8.6}$$

The similarity with the ADE in eq. (8.2) becomes obvious when the following substitutions are performed:

$$\begin{aligned} \rho \boldsymbol{u} &\to C, \\ \rho \boldsymbol{u}\boldsymbol{u} + p\mathbf{1} &\to C\boldsymbol{u}, \\ \nu &\to D, \\ \boldsymbol{F} &\to q. \end{aligned} \tag{8.7}$$

Therefore, instead of working with the momentum density vector $\rho \boldsymbol{u}$ as conserved quantity, we now have the scalar C as central observable. The kinematic viscosity ν is replaced by the diffusion coefficient D. There is no analogon for the fluid incompressibility in the advection-diffusion formalism. While the NSE involves two conserved quantities (density and momentum density), there is only one conserved quantity in the ADE (concentration).

Exercise 8.1. Convince yourself that the NSE is an advection-diffusion equation for the momentum density $\rho \boldsymbol{u}$. What is the meaning of the kinematic viscosity ν and the force density $\boldsymbol{F}$?

Since the functional forms of eq. (8.2) and eq. (8.6) are very similar, the next logical step is to use the same general LB algorithm as before:

$$g_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) - g_i(\boldsymbol{x}, t) = \Omega_i(\boldsymbol{x}, t) + Q_i(\boldsymbol{x}, t). \tag{8.8}$$

Here, we write g_i to denote populations for the scalar C, Ω_i is the collision operator, and Q_i is responsible for the source term q.

The simplest collision model for advection-diffusion problems is the BGK operator:

$$\Omega_i(\boldsymbol{x}, t) = -\frac{1}{\tau_D}\left(g_i(\boldsymbol{x}, t) - g_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right). \tag{8.9}$$

We write τ_D to distinguish the relaxation time from τ in the standard LBM for the NSE.

It will be shown in section 8.2.4 that the **diffusion coefficient** D in the BGK model is given by the relaxation time τ_D:

$$D = c_{\mathrm{s}}^2 \left(\tau_D - \frac{\Delta t}{2}\right). \tag{8.10}$$

The speed of sound c_s depends on the chosen lattice (section 8.2.3).

Now we have to find out how to construct the equilibrium populations g_i^{eq} to recover the ADE, rather than the NSE, in the macroscopic limit.

8.2.2 Equilibrium distribution

The first assumption is that the zeroth and first moments of g_i obey

$$\sum_i g_i = \sum g_i^{\mathrm{eq}}, \quad \sum_i g_i \boldsymbol{c}_i = \sum_i g_i^{\mathrm{eq}} \boldsymbol{c}_i, \tag{8.11}$$

just as in the LBM for NSE. To satisfy the substitutions in eq. (8.7), we claim

$$\sum_i g_i^{\mathrm{eq}} = C, \tag{8.12a}$$

$$\sum_i g_i^{\mathrm{eq}} \boldsymbol{c}_i = C\boldsymbol{u}. \tag{8.12b}$$

Now we have to construct an equilibrium in such a way that these moments are satisfied.

The ADE is linear in C and $\boldsymbol{u}$, so we start with a *linear ansatz* for the equilibrium:

$$g_i^{\mathrm{eq}} = w_i C(A + \boldsymbol{B}_i \cdot \boldsymbol{u}) \tag{8.13}$$

with unknown scalar A and vectors $\boldsymbol{B}_i$. We have included the lattice weights w_i of the so far unspecified lattice. They have the standard properties

$$\sum_i w_i = 1, \quad \sum_i w_i c_{i\alpha} = 0, \quad \sum_i w_i c_{i\alpha} c_{i\beta} = c_s^2 \delta_{\alpha\beta}. \tag{8.14}$$

Inserting eq. (8.13) into eq. (8.12a) yields

$$\sum_i w_i C(A + \boldsymbol{B}_i \cdot \boldsymbol{u}) = C. \tag{8.15}$$

Since the right-hand side of this equation does no depend on $\boldsymbol{u}$, the vector $\boldsymbol{B}$ has to satisfy $\sum_i B_{i\alpha} = 0$. Furthermore we find $\sum_i w_i CA = C$ or, exploiting eq. (8.14), $A = 1$. Combining eq. (8.13) with eq. (8.12b) gives

$$\sum_i w_i C u_\beta B_{i\beta} c_{i\alpha} = C u_\alpha. \tag{8.16}$$

From this and eq. (8.14) we can infer $B_{i\alpha} = c_{i\alpha}/c_s^2$.

The simplest **equilibrium distribution** that leads to the **advection-diffusion equation** is the linear function

$$g_i^{\mathrm{eq}} = w_i C \left(1 + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}}{c_\mathrm{s}^2}\right) \tag{8.17}$$

where the concentration C is obtained from

$$C = \sum_i g_i. \tag{8.18}$$

The velocity $\boldsymbol{u}$ is an external field that has to be provided. It could be (but does not have to be) the solution of the LBE for the Navier-Stokes equation.

In many applications, two LBEs are solved side by side: one for the momentum and therefore the velocity $\boldsymbol{u}$, the other for the field C. This is particularly straightforward if C is purely passive, *i.e.* if it is merely moving along with the fluid, without affecting the NSE in return. This assumption is typically justified if C indicates a dilute chemical species. However, if C is used for the temperature T and the fluid density ρ is a function of T, then the NSE itself depends on the outcome of the ADE and a fully coupled system of equations has to be solved. TODO (TK): Refer to later section.

Although the equilibrium is linear in C, it does not have to be linear in the velocity $\boldsymbol{u}$. An alternative quadratic equilibrium assumes the form

$$g_i^{\mathrm{eq}} = w_i C \left(1 + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}}{c_\mathrm{s}^2} + \frac{(\boldsymbol{c}_i \cdot \boldsymbol{u})^2}{2c_\mathrm{s}^4} - \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2c_\mathrm{s}^2}\right). \tag{8.19}$$

The choice of the equilibrium distribution has some subtle effects on the accuracy and convergence as we will briefly discuss in section 8.2.4.

Exercise 8.2. Show that eq. (8.19) fulfils the same moments, eq. (8.12), as the linear equilibrium in eq. (8.17). This requires the additional lattice isotropy condition $\sum_i w_i c_{i\alpha} c_{i\beta} c_{i\gamma} = 0$.

8.2.3 Choice of the lattice

The major difference between LBM for the NSE and the ADE is that the former requires velocity moments up to the second order, but the latter requires only the zeroth and first moments. We can therefore assume that we can get away with a lower-isotropy lattice when we are only interested in the ADE. Indeed this has been shown TODO (TK): Write about isotropy requirements ([1], section 5.8.2).

Although D2Q9 or D3Q19 are often used for the ADE, it is absolutely sufficient to employ the D2Q5 or D3Q7 instead. TODO (TK): Briefly define the lattices.

TODO (TK): Are there any disadvantages if a lower-order lattice is used? What is the difference between D2Q4 and D2Q5 for ADE?

TODO (TK): Speed of sound for lattices: $c_s^2 = 1/2$ for lattices without rest velocity and $c_s^2 = 1/3$ for lattices with rest velocity

TODO (TK): Here we could use an example simulation, *e.g.* diffusion of Gaussian hill and radial profile after a certain number of time steps.

8.2.4 Chapman-Enskog analysis

We discuss the relevant steps of the Chapman-Enskog analysis. The aim is to show that the BGK equation

$$g_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = g_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau_D}\left(g_i(\boldsymbol{x}, t) - g_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right) \tag{8.20}$$

indeed recovers the ADE. We will also show how the relaxation time τ_D and the diffusion coefficient D are connected. Finally, we point out the existence of an error term that can reduce the convergence of the ADE to first order.

Expansion

The Chapman-Enskog analysis shown here is very similar to that in section 4.1. Therefore we will not go into details and provide only the assumptions and key results.

We expand the populations g_i as

$$g_i = g_i^{\mathrm{eq}} + \epsilon g_i^{(1)} + \epsilon^2 g_i^{(2)} + \dots \tag{8.21}$$

and the time and space derivative operators as

$$\partial_t = \epsilon\partial_{t_1} + \epsilon^2\partial_{t_2} + \dots \tag{8.22}$$

and

$$\nabla_\alpha = \epsilon\nabla_{1,\alpha}. \tag{8.23}$$

Additionally, the usual Taylor expansion of the propagation step up to $O(\Delta t^2)$ gives

$$\begin{aligned} g_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - g_i(\boldsymbol{x}, t) = \Delta t\,&\left(\partial_t g_i + \nabla_\alpha g_i c_{i\alpha}\right) \\ &+ \frac{\Delta t^2}{2}\left(\partial_t^2 g_i + 2\partial_t\nabla_\alpha g_i c_{i\alpha} + \nabla_\alpha\nabla_\beta g_i c_{i\alpha}c_{i\beta}\right) \end{aligned} \tag{8.24}$$

where the right-hand side is evaluated at $(\boldsymbol{x}, t)$. We can now combine eq. (8.24) with eq. (8.20) to remove $g_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t)$. Furthermore we insert the expansions from eq. (8.21), eq. (8.22) and eq. (8.23) and keep only terms up to $O(\epsilon^2)$:

$$\begin{aligned} -\frac{\Delta t}{\tau_D}\left(\epsilon g_i^{(1)} + \epsilon^2 g_i^{(2)}\right) = {} & \epsilon\Delta t\left(\partial_{t_1} g_i^{\mathrm{eq}} + \nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha}\right) \\ & + \epsilon^2\Delta t\left(\partial_{t_1} g_i^{(1)} + \partial_{t_2} g_i^{\mathrm{eq}} + \nabla_{1,\alpha} g_i^{(1)} c_{i\alpha}\right) \\ & + \frac{\epsilon^2\Delta t^2}{2}\left(\partial_{t_1}^2 g_i^{\mathrm{eq}} + 2\partial_{t_1}\nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha} + \nabla_{1,\alpha}\nabla_{1,\beta} g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta}\right). \end{aligned} \tag{8.25}$$

Next we consider only terms of $O(\epsilon)$:

$$-\frac{1}{\tau_D} g_i^{(1)} = \partial_{t_1} g_i^{\mathrm{eq}} + \nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha}. \tag{8.26}$$

Summing over i and assuming the validity of the strong solvability conditions, $\sum_i g_i^{(n)} = 0$ for $n > 0$, the left-hand side vanishes. Furthermore, we know the first two moments of g_i^{eq}, eq. (8.12), and can write

$$\partial_{t_1} C + \nabla_{1,\alpha}(C u_\alpha) = 0. \tag{8.27}$$

Therefore, we see that the terms of $O(\epsilon)$ give rise to advection of C.

The terms of $O(\epsilon^2)$ are

$$\begin{aligned} -\frac{1}{\tau_D} g_i^{(2)} = {} & \left(\partial_{t_1} g_i^{(1)} + \partial_{t_2} g_i^{\mathrm{eq}} + \nabla_{1,\alpha} g_i^{(1)} c_{i\alpha}\right) \\ & + \frac{\Delta t}{2}\left(\partial_{t_1}^2 g_i^{\mathrm{eq}} + 2\partial_{t_1}\nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha} + \nabla_{1,\alpha}\nabla_{1,\beta} g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta}\right). \end{aligned} \tag{8.28}$$

We can replace $\partial_{t_1} g_i^{(1)}$ and $\nabla_{1,\alpha} g_i^{(1)}$ by taking appropriate derivatives of eq. (8.26):

$$\partial_{t_1} g_i^{(1)} = -\tau_D\left(\partial_{t_1}^2 g_i^{\mathrm{eq}} + \partial_{t_1}\nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha}\right), \tag{8.29a}$$

$$\nabla_{1,\alpha} g_i^{(1)} c_{i\alpha} = -\tau_D\left(\partial_{t_1}\nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha} + \nabla_{1,\alpha}\nabla_{1,\beta} g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta}\right). \tag{8.29b}$$

This way, eq. (8.28) becomes

$$-\frac{1}{\tau_D} g_i^{(2)} = \partial_{t_2} g_i^{\mathrm{eq}} - \left(\tau_D - \frac{\Delta t}{2}\right)\left(\partial_{t_1}^2 g_i^{\mathrm{eq}} + 2\partial_{t_1}\nabla_{1,\alpha} g_i^{\mathrm{eq}} c_{i\alpha} + \nabla_{1,\alpha}\nabla_{1,\beta} g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta}\right). \tag{8.30}$$

Now we can sum over i. From eq. (8.17) we find $\sum_i g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta} = C c_{\mathrm{s}}^2 \delta_{\alpha\beta}$ and therefore

$$0 = \partial_{t_2} C - \left(\tau_D - \frac{\Delta t}{2}\right)\left(\partial_{t_1}^2 C + 2\partial_{t_1}\nabla_{1,\alpha}(C u_\alpha) + c_{\mathrm{s}}^2\nabla_1^2 C\right). \tag{8.31}$$

Taking advantage of the time derivative of eq. (8.27), we can further write

$$\partial_{t_2} C = c_s^2 \left(\tau_D - \frac{\Delta t}{2} \right) \nabla_1^2 C + E \tag{8.32}$$

where

$$E = \left(\tau_D - \frac{\Delta t}{2} \right) \partial_{t_1} \nabla_{1,\alpha} (C u_\alpha) \tag{8.33}$$

is an undesired error term. Concluding, the terms $O(\epsilon^2)$ lead to the diffusion equation with an additional error.

Neglecting the error E for now, we can combine the results from eq. (8.27) and eq. (8.32):

$$\partial_t C + \nabla_\alpha (C u_\alpha) = D \nabla^2 C \tag{8.34}$$

which is obviously the desired ADE. Here we have already used the identity

$$D = c_s^2 \left(\tau_D - \frac{\Delta t}{2} \right) \tag{8.35}$$

by comparing the result of the Chapman-Enskog analysis with the ADE in eq. (8.2) for $q = 0$. As for the Navier-Stokes case, the additional term $-\Delta t/2$ in eq. (8.35) is a discrete lattice artifact that has been absorbed into the definition of D.

Error term

We have seen that, up to $O(\Delta t^2)$ and $O(\epsilon^2)$, we encounter an error term E as shown in eq. (8.33). This term interferes with the second-order convergence of the ADE based on the LBM. If the quadratic equilibrium in eq. (8.19) is used, the error gets an additional contribution proportional to $\nabla_\alpha \nabla_\beta (C u_\alpha u_\beta)$ [2]. Several improvements have been proposed to restore second-order convergence. The common approach is to add a source term that cancels the error. There exist non-local [2] and local [3] schemes to compute the derivatives of the scalar C that appear in eq. (8.33). We will not discuss these extended methods further.

Inhomogeneous diffusivity

In the Chapman-Enskog analysis we have treated τ_D and therefore D as constant. It is straightforward to extend this to a situation with inhomogeneous diffusivity, where D becomes $D(\boldsymbol{x})$. To achieve this, the gradient in eq. (8.29b) cannot be moved past the relaxation time so that instead

$$\nabla_{1,\alpha} g_i^{(1)} c_{i\alpha} = -\nabla_{1,\alpha} \left[\tau_D \left(\partial_{t_1} g_i^{\mathrm{eq}} c_{i\alpha} + \nabla_{1,\beta} g_i^{\mathrm{eq}} c_{i\alpha} c_{i\beta} \right) \right]. \tag{8.36}$$

To leading order, this gives the macroscopic equation

$$\partial_t C + \nabla_\alpha (C u_\alpha) = \nabla_\alpha (D \nabla_\alpha C), \tag{8.37}$$

i.e. the ADE with inhomogeneous diffusion coefficient. It is therefore possible to vary D by changing τ_D locally according to eq. (8.35) at each $\boldsymbol{x}$.

Exercise 8.3. Show that eq. (8.37) follows from eq. (8.36).

Source term and reactions

TODO (Also say something about the source term.):

Higher-order thermal lattice-Boltzmann models

TODO (TK): Thermal LBM and how to construct such models are covered by [1] in section 5.5.

8.2.5 Parameter selection and model extensions

TODO (TK): Not to be covered here but maybe mentioned for further reading: axisymmetric LBM for ADE [4]

TODO (TK): Discuss parameter selection: the Péclet number cannot be arbitrarily chosen. This would be too much for chapter 7, we have to cover this here. What about the stability criterion? Numerical cost? TODO (TK): Based on [5], say that a large variation of D can be simulated as well.

TODO (TK): Mention TRT for advection-diffusion and provide references [6].

8.3 Boundary conditions

In this section we will shortly present most used boundary conditions schemes for the advection-diffusion equation. We extensively covered different boundary conditions for flow in chapter 6 and chapter **??**. In most cases boundary conditions used for the advection-diffusion equation are the same as for flow but simplified. Here, we will cover some boundary conditions presented in chapter 6 from another angle and present a few new boundary conditions.

As we already covered, there are three major types of the boundary conditions: Dirichlet, von Neumann and Robin BCs. The third one is a linear combination of the first two.

8.3.1 Dirichlet BC

Dirichlet BC specifies a constant concentration $C = C^*$ at a boundary. For example, it could be a constant temperature or a concentration on the wall. The Dirichlet boundary condition for flow corresponds to a constant pressure BC, which is consequently connected with the density through the sound speed $p = c_s\rho$. Thus, for example the Zou-He scheme (*cf.* chapter 5) that fixes a constant pressure could be successfully used for constant concentration in the advection-diffusion equation. However, an application of the Zou-He BCs for the arbitrary geometries is rather complicated because of corners treatment.

A good alternative that will be presented below is the *pressure anti-bounce-back* conditions and the Inamuro method that can treat arbitrary geometries in a rather general way. Especially, the pressure anti-bounce-back is as simple as the bounce-back boundary treatment.

Pressure anti-bounce-back Pressure anti-bounce-back is used to specify a pressure/concentration at a boundary. It operates in the same manner as the bounce-back [7]:

$$\begin{aligned} g^*_{\bar{i},\text{wall}} &= -g^*_{i,\text{bulk}} + 2g^{\text{eq}}_{i,\text{wall}} \\ g^*_{\bar{i}}(\boldsymbol{x}, t) &= -g^*_i(\boldsymbol{x} + \boldsymbol{c}_i, t) + 2g^{\text{eq}}(\boldsymbol{x}, t), \end{aligned} \tag{8.38}$$

where again by superscript $*$ we denote postcollision populations, the subscript bulk is referred to the internal domain. If the boundary has zero velocity, the imposed wall concentation is C^*, then the pressure anti-bounce-back condition is transformed into:

$$g^*_{\bar{i}}(\boldsymbol{x}, t) = -g^*_i(\boldsymbol{x} + \boldsymbol{c}_i, t) + 2w_i C^*, \tag{8.39}$$

where w_i are the lattice weights. This method works with any arbitrary geometry, at it is as simple as the bounce-back condition. Notice that the anti-bounce-back wall nodes are not the part of the internal domain, they are not *wet* nodes (*cf.* chapter 5). We will provide an example of the imposing the constant concentration on the cylinder boundary approximated by the staircase in section 8.4.

Inamuro BC The Inamuro BC [8, 9] also allows imposing a concentration at a boundary. If the boundary wall has the normal $\boldsymbol{n}$, then the populations coming into the domain are $\boldsymbol{c}_i \cdot \boldsymbol{n} > 0$. Inamuro implementation assumes that those populations can be represented through the equilibrium populations with unknown yet-to-find concentration C':

$$g_i = w_i C', \tag{8.40}$$

where C' is to be specified. Note that for equilibrium populations in eq. (8.40) we assume velocity being zero at the BC, which is not the case for example for outflow BC or the thermal layer development with the uniform velocity. It is done only for the calculations simplification purposes. The velocity inclusion in the equilibrium does not affect results. NOTE (TK): It would be good to show the general case (with non-zero velocity) first and then simplify it for the example.

We assume that the sum of the populations equals the wall concentration specified by the BC:

$$\sum_i g_i = C_{\text{wall}} = \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} > 0} g_i + \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} \leq 0} g_i = \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} > 0} w_i C' + \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} \leq 0} g_i \tag{8.41}$$

From this equation one can find the unknown concentration:

$$C' = \frac{C_{\text{wall}} - \sum_{\boldsymbol{c} \cdot \boldsymbol{n} \leq 0} g_i}{\sum_{\boldsymbol{c}_i \cdot n > 0} w_i} \tag{8.42}$$

Knowing the unknown concentration one can specify the incoming to the domain populations via eq. (8.40). One very important note here is that the Inamuro BC nodes are the part of the internal domain (*wet* nodes), so the collision is done at these nodes. If the usual flow of the LBM algorithm is the collision followed by the propagation, then one should add the Inamuro wall treatment implementation right after the propagation. A

8.3.2 Neumann BC

NOTE (TK): This section requires references. Please add them as early as possible.

Neumann BC specifies a normal diffusion flux at a boundary: $D\frac{\partial C}{\partial n} = j_n$. Usually fluxes are specified at a wall or at an inlet. For example, a zero flux relates to an impenetrating wall, a non-zero flux is usually attributed to an inlet.

A wall impenetration can be easily implemented as the usual bounce-back condition. The bounce-back condition for flow assumes the zero velocity value between fluid and wall nodes. However, it is different for the advection-diffusion equation. The bounce-back for ADE implements zero *normal* flux. Thus, it is suitable to simulate any boundary condition with $\partial C/\partial n = 0$ and a tangential velocity to the boundary with the normal $\boldsymbol{n}$. For example, we successfully used the bounce-back condition to simulate the thermal layer development with the uniform velocity profile (*cf.* section 8.4).

If the velocity have non-tangential components to a boundary, then two procedures are widespread:

Inamuro flux BC If a normal flux is specified, then the summation of all populations is:

$$j_n = \sum_i g_i \boldsymbol{c}_i \cdot \boldsymbol{n}. \tag{8.43}$$

Along the same procedure described previously, the populations pointing into the internal domain can be represented through the equilibrium population with yet-to-find concentration C'. For simplicity we assume that $g_i^{\text{eq}} = w_i C'$ for $\boldsymbol{c}_i \cdot n > 0$:

$$\begin{aligned} j_n &= \sum_i g_i \boldsymbol{c}_i \cdot \boldsymbol{n} = \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} > 0} g_i \boldsymbol{c}_i \cdot \boldsymbol{n} + \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} \leq 0} g_i \boldsymbol{c}_i \cdot \boldsymbol{n} \\ &= \sum_i w_i C' \boldsymbol{c}_i \cdot \boldsymbol{n} + \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} \leq 0} g_i \boldsymbol{c}_i \cdot \boldsymbol{n}. \end{aligned} \tag{8.44}$$

After some algebra one can obtain:

$$C' = \frac{j_n - \sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} \leq 0} g_i \boldsymbol{c}_i \cdot \boldsymbol{n}}{\sum_{\boldsymbol{c}_i \cdot \boldsymbol{n} > 0} w_i \boldsymbol{c}_i \cdot \boldsymbol{n}}. \tag{8.45}$$

Transformation The second procedure is to transform the constant diffusion flux to the constant concentration boundary condition. For example, if the normal to a wall vector $\boldsymbol{n}$ is known, then the flux boundary condition is approximated in the following way:

$$D \frac{\partial C}{\partial n} \approx D \frac{C_{\text{bulk}} - C_{\text{wall}}}{\Delta n} = j_n, \tag{8.46}$$

where the bulk concentration is located at the location $\boldsymbol{x}_{\text{wall}} + \boldsymbol{n}$. With the complex boundaries not aligned with the underlying lattice, the bulk node as well can be not aligned with it. Here, we do not cover such cases. In the case of the staircase boundary approximation, the normal is a vector rotated from the major axes by the angle proportional to 45 degrees. The wall concentration can be approximated using eq. (8.46):

$$C_{\text{wall}} = C_{\text{bulk}} - \frac{j_n \Delta n}{D}, \tag{8.47}$$

where Δn depends on the angle

$$\Delta n = \begin{cases} 1 & \text{if } \boldsymbol{n} \text{ is along one of the axes} \\ \sqrt{2} & \text{otherwise.} \end{cases} \tag{8.48}$$

One of the widely used applications of the substitution of the Neumann boundary condition with the Dirichlet is the outflow boundary condition. If the outflow boundary is along one of the axes, then $C_{\text{boundary}} = C_{\text{bulk}}$ according to eq. (8.47). This condition is often implemented just by copying postcollion populations from the layer nearest to the outflow boundary. For example, outflow boundary conditions for two benchmark cases in section 8.4 are implemented by this procedure.

8.3.3 Robin BC

The Robin BC is a combination of the Neumann and Dirichlet boundary conditions:

$$j_n = h(C_{\text{bulk}} - C_{\text{wall}}), \tag{8.49}$$

where h is the proportionality coefficient between flux and the concentration species difference between bulk and wall concentrations. For the temperature ADE this condition is nothing else but the convective wall boundary condition with h being the heat transfer coefficient.

Note that C_{wall} is given externally, and C_{bulk} is the result of a solve procedure. Therefore, at each particular time step one can calculate the normal flux applied at the boundary and consecutively reduce this BC to previous cases. However, one caution needs to be taken into the account. If one decides to represent the Robin BC through the Dirichlet boundary condition, then it is necessary to distinguish the concentration at the boundary given by the external condition to calculate the normal flux through eq. (8.49) (*cf.* C_{wall} there) and the concentration imposed at the boundary to fulfill BC. NOTE (TK): I think this section is not sufficiently clear. An example would be great (maybe there is one in the benchmarks below?). Also, references are missing.

8.4 Benchmark problems

This section covers specifically chosen benchmarks that apply boundary conditions previously covered. Particularly, the aim of this section is two-fold: to present analytical solutions that could be useful when one designs and verifies its own solutions and to present numerical results of previously covered boundary conditions. Here we will present three different benchmarks: diffusion from the walls of the two-dimensional cylinder to inner region, the concentration layer development with the Poiseuille velocity profile, and finally the concentration layer development with the uniform velocity. NOTE (TK): This is an excellent idea, and the examples are very important. But what about simpler problems like advection-diffusion of a Gaussian hill? The benchmarks here do not have to be related to boundary problems (bulk benchmarks are also important).

8.4.1 Diffusion from the cylinder

The cylinder of radius a with the imposed constant concentration C^* at the surface diffuses species into inner domain with the initial concentration C_{init}.

The PDE system represented in cylindrical coordinates that describes this setup is the following:

$$\begin{aligned} \partial_t C(r,t) &= \frac{D}{r}\partial_r r \partial_r C(r,t) \\ C(a,t) &= C^*,\ C(r,0) = C_{\text{init}} \end{aligned} \tag{8.50}$$

The analytical solution with a lot of other examples for the advection-diffusion equation is presented [10]:

$$\frac{C(r,t)-C^*}{C_{\text{init}}-C^*} = \sum_{n=1}^{\infty} \frac{2}{\mu_n J_1(\mu_n)} \exp\left(-\mu_n^2 \frac{Dt}{a^2}\right) J_0\left(\mu_n \frac{r}{a}\right), \tag{8.51}$$

where μ_n is the n-th zero root of the 0th order Bessel polynomial $J_0(\mu_n) = 0$. Some of the corresponding roots are as follows: $\mu_1 = 2.4048$, $\mu_2 = 5.5201$, $\mu_3 = 8.6537$, $\mu_4 = 11.7915$, $\mu_5 = 14.9309$. By taking the initial concentration as $C_{\text{init}} = 0$, the analytical solution is simplified to:

$$C(r,t) = C^*\left(1 - \sum_{n=1}^{\infty} \frac{2}{\mu_n J_1(\mu_n)} \exp\left(-\mu_n^2 \frac{Dt}{a^2}\right) J_0(\mu_n \frac{r}{a})\right). \tag{8.52}$$

The numerical setup is the domain of 129×129 lattice spacings. The radius of the cylinder is $a = 40$ lattice spacings. The relaxation rate for the BGK collision operator D2Q9 model is chosen as $\omega = 1.939$ that corresponds to the diffusion coefficient $D = 1/3(1/\omega - 1/2) = 0.0052$. The cylinder boundary is approximated through a stair-case curve. The constant concentration for the cylinder boundary $C^* = 1$ is imposed by using two different boundary conditions: pressure anti-bounce-back and Inamuro BC (wet nodes). The comparison of two different Dirichlet BC implementations and diffusion propagation profiles along the x-axis at two different positions in time are shown in fig. 8.2. Notice that the time is non-dimensionalized as $\hat{\tau} = \frac{Dt}{a^2}$. Both approaches are able to accurately capture the diffusion.

NOTE (TK): This is a nice example (will have to be clarified a bit later). Since this is a problem with zero velocity, it is a good idea to show it as first benchmark.

8.4.2 Poiseuille velocity profile

Here we present the concentration layer development between two parallel plates subject to the concentration C^*. Velocity profile is the parabolic Poiseuille velocity profile. The following PDE formulates this problem:

$$\begin{aligned} &\frac{\partial C}{\partial x} U(y) = D \frac{\partial^2 C}{\partial y^2} \\ &C(0,y) = 0,\ C(x,\pm H) = C^*, \\ &U(y) = U_0\left(1 - \left(\frac{y}{H}\right)^2\right), \qquad y \in [-H, H]. \end{aligned} \tag{8.53}$$

Notice that the problem has a symmetry plane at $y = 0$. Thus, we will reformulate it by using only a half of the channel and by adding an additional symmetric condition (see fig. 8.3):

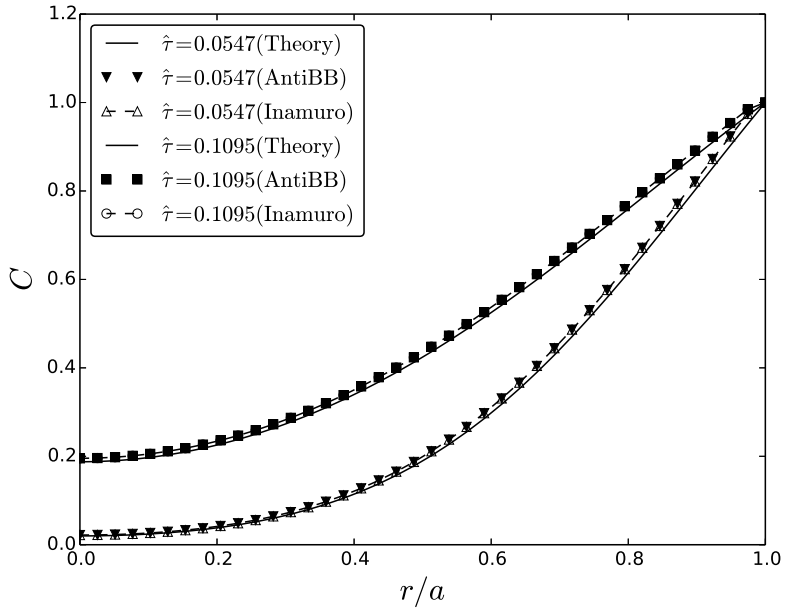


Fig. 8.2: Profiles for species diffusion along the x-axis from the cylinder boundary with the imposed concentration $C^* = 1$ are presented. The relaxation rate $\omega = 1.939$ corresponds to the diffusion coefficient $D = 0.0052$. Two different non-dimensional times $\hat{\tau} = Dt/a^2$ are chosen. r is the distance from the center. Constant concentration is imposed with the help of the anti-bounce-back boundary condition and the Inamuro boundary condition. The analytical solution was calculated using eq. (8.52). One can see that the diffusion from curved boundaries is captured accurately.

$$\begin{aligned} &\frac{\partial C}{\partial x} U(y) = D \frac{\partial^2 C}{\partial y^2} \\ &C(0, y) = 0,\ C(x, H) = C^*, \partial_y C(x, y)|_{y=0} = 0, \\ &U(y) = U_0 \Big(1 - \Big(\frac{y}{H}\Big)^2\Big), \qquad y \in [0, H]. \end{aligned} \tag{8.54}$$

The procedure to obtain the analytical solution is presented in details in the appendix of [11]. The final solution is the following:

$$C = C^* - C^* \sum_{m=0} C_m e^{-m^4 \frac{x}{H} \frac{1}{Pe}} e^{-m^2 y^2/(2H^2)} {}_1F_1\Big(-\frac{m^2}{4} + \frac{1}{4}, \frac{1}{2}, m^2 \frac{y^2}{H^2}\Big), \tag{8.55}$$

where ${}_1F_1$ is the hypergeometric function. Coefficients C_m can be found through the integrals of the hypergeometric function:

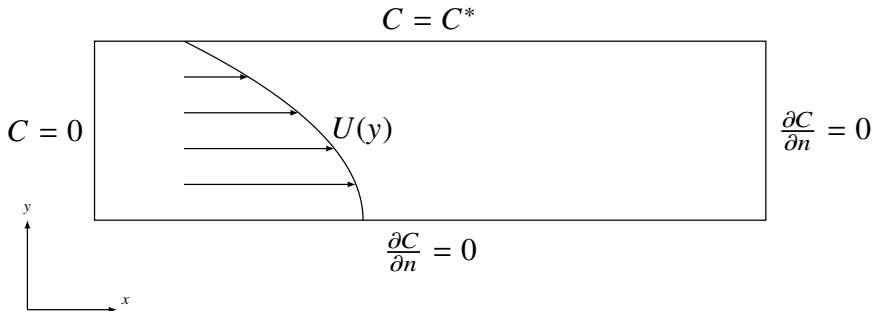


Fig. 8.3: The sketch for the benchmark with the Poiseuille velocity profile and boundary conditions applied is presented. Due to symmetry only a half of the channel is simulated.

$$C_m = -C^* \frac{\int_{\xi=0}^{1} (1-x^2) e^{-m^2\xi^2/2} {}_1F_1\left(-\frac{m^2}{4}+\frac{1}{4},\frac{1}{2},m^2\xi^2\right) \mathrm{d}\xi}{\int_{\xi=0}^{1} (1-\xi^2) e^{-m^2\xi^2/2} {}_1F_1\left(-\frac{m^2}{4}+\frac{1}{4},\frac{1}{2},m^2\xi^2\right)^2 \mathrm{d}\xi} \tag{8.56}$$

For the case $C^* = 1$, the first ten coefficients C_m are: 1.2008, −0.2991, 0.1608, −0.1074, 0.0796, −0.0627, 0.0515, −0.0435, 0.0375, −0.0329. Those first ten members were used to reconstruct the analytical solution. The comparison between contours of analytical and simulation results is presented in fig. 8.4. Parameters were taken as: $D = 0.0185$, the grid dimension is $N_x \times N_y = 400 \times 40$. The centerline velocity is $U_0 = 0.05$ which yields the Peclet number $Pe = U_0 N_y / D = 108.108$. The numerical domain is $N_y/2 \times 5N_y = 40 \times 400$ lattice units. The symmetric boundary condition is implemented via the bounce-back condition, the inlet and wall constant concentration are via the anti-bounce-back. The outflow boundary condition at $x = N_x$ is the populations copying from the layer at $x = N_x - 1$. The results are in good agreement. Note that the bounce-back condition for the wall is used even the flow velocity at the wall is not zero. As we indicated earlier, the bounce-back condition provides good results if the velocity is tangential to the boundary. NOTE (TK): Also a nice problem, but maybe a bit complicated. I think it would be good to have one example that has a relatively simple analytical solution and does not require boundary conditions (e.g. Gaussian hill in a sufficiently large periodic domain). The Poiseuille example could be shown as last benchmark.

8.4.3 Uniform velocity profile

Final example is the concentration layer development from the semi-infinite plate with the imposed constant concentration by uniformly developed flow. In terms of the corresponding PDE it can be represented:

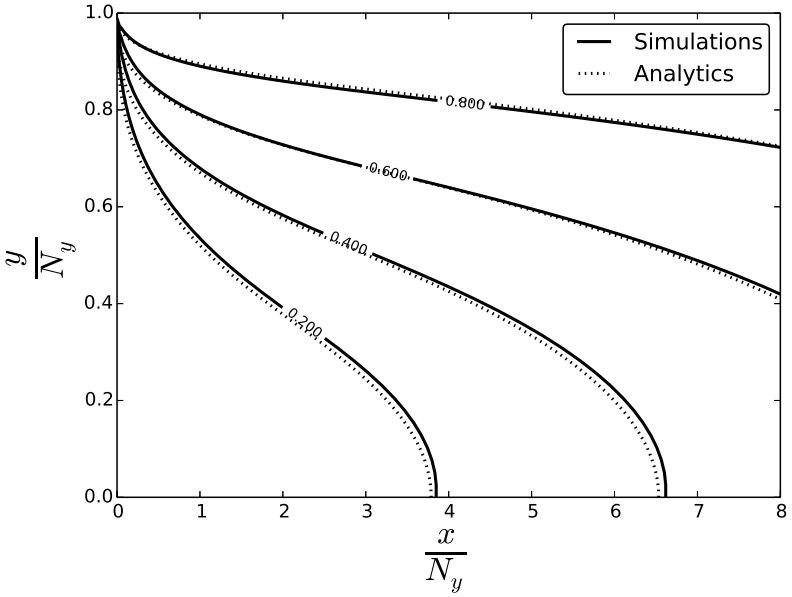


Fig. 8.4: Comparison between the analytical concentration contours and simulations with pressure anti bounce-back conditions for concentration at the inlet and the top wall. The simulation were done for the diffusion coefficient $D = 0.0185$ with a 40×400 grid. The centerline velocity is $U_0 = 0.05$, and the Peclet number is 108.108.

$$\begin{aligned} &\frac{\partial C}{\partial x} U_0 = D \frac{\partial^2 C}{\partial y^2} \\ &C(0, y) = 0,\ C(x, 0) = C^*. \end{aligned} \tag{8.57}$$

Here we assume that the advection component of species propagation along the x-axis is larger than the diffusion component (large Peclet number). That allowed neglecting $D\partial^2 C/\partial x^2$ in eq. (8.57). The solution of this PDE is the following [12]:

$$C(x, y) = C^* \mathrm{erfc}\left(\frac{y}{\sqrt{4 \frac{Dx}{U_0}}} \right), \tag{8.58}$$

where $\mathrm{erfc}(x)$ is the complementary error function:

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \mathrm{d}t. \tag{8.59}$$

Though this example looks simpler than the previous one with the parabolic velocity profile, there is one interesting thing what we want to present here. That concerns simulations with semi-infinite domains. For such domains it is often assumed that at big distances the solution is not changed which is represented by the normal flux being zero. While imposing zero-flux boundary conditions, one should verify that distances of walls where those BCs are imposed are large enough not to affect numerical solutions. For more precise estimates one can analyse eq. (8.58).The numerical setup sketch is presented in fig. 8.5.

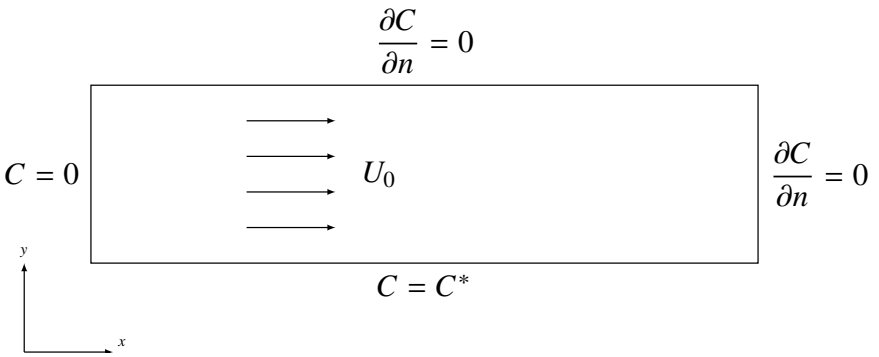


Fig. 8.5: Numerical setup sketch for eq. (**??**) that provides a precise solution if certain conditions are fulfilled: the channel length and height are chosen in such way as the diffusion is not able to reach top wall along the channel length.

The domain is of size $N_x \times N_y = 400 \times 40$ lattice spacings. The top wall is simulated by using a simple bounce-back rule, the upstream outlet BC is implemented by the outflow boundary condition where we copy post-collision populations close to it. The constant wall concentration is simulated by using the anti-bounce-back rule. Note that the anti-bounce-back and the bounce-back conditions assuming the velocity being zero are applied in places where there is a non-zero tangential velocity, and that does not affect the numerical solution. The relaxation rate is $\omega = 1.8$ which corresponds to the diffusion coefficient $D = 1/3(1/1.8 - 1/2) = 0.0185$, the uniform velocity magnitude is $U_0 = 0.05$. Overall, this corresponds to the large Peclet number $Pe = \frac{U_0 N_y}{D} = 108$. With this Peclet number and the domain size chosen we are guaranteed that the zero flux conditions at the top wall and at the outlet do not affect the numerical solution, which is shown in fig. 8.6. NOTE (TK): There is quite some deviation between simulations and the analytical result. What is the reason? I think a benchmark is only useful if the errors are properly analysed. Several potential explanations come to my mind, but the reader is left in the dark. We should have at least one grid refinement study in this chapter.

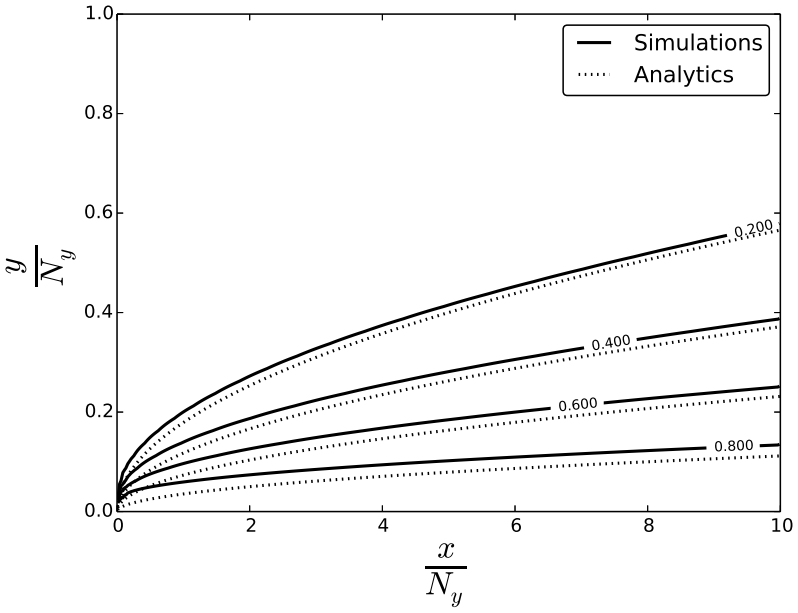


Fig. 8.6: Concentration layer development with the uniform velocity profile. The analytical solution is constructed from eq. (8.58). The domain is $N_x \times N_y = 400 \times 40$ lattice spacings. The relaxation rate $\omega = 1.8$ corresponds to the diffusion coefficient $D = 0.00185$. The uniform velocity magnitude is $U_0 = 0.05$, the Peclet number is $Pe = U_0 N_y / D = 108$.

References

1. D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models* (Springer, 2005)
2. B. Chopard, J.L. Falcone, J. Latt, Eur. Phys. J. Spec. Top. **171**, 245 (2009)
3. Z. Chai, T.S. Zhao, Phys. Rev. E **87**(6), 063309 (2013)
4. A. Mohamad, *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*, 1st edn. (Springer, 2011)
5. J. Perko, R.A. Patel, Phys. Rev. E **89**(5), 053309 (2014)
6. I. Ginzburg, D. d'Humières, A. Kuzmin, J. Stat. Phys. **139**(6), 1090 (2010)
7. I. Ginzburg, Adv. Water Res. **28**(11), 1196 (2005)
8. M. Yoshino, T. Inamuro, Int. J. Num. Meth. Fluids **43**(2), 183 (2003)
9. T. Inamuro, M. Yoshino, H. Inoue, R. Mizuno, F. Ogino, J. Comp. Phys. **179**(1), 201 (2002)
10. A. Polyanin, A. Kutepov, A. Vyazmin, D. Kazenin, *Hydrodynamics, Mass and Heat Transfer in Chemical Engineering* (Taylor and Francis, 2002)
11. A. Kuzmin, Z. Guo, A. Mohamad, Phil. Trans. Royal Soc. A **369**, 2219 (2011)
12. M. Ozisik, *Heat Conduction* (John Wiley and Sons, 1993)

Chapter 9
Multi-phase and multi-component flows

9.1 Introduction

One of the most popular applications of the lattice Boltzmann method is on multiphase and multicomponent flows. For a multiphase flow, the liquid and gas phases of the same substance are in coexistence. The two phases can interconvert from one to another: the gas can condense to form more liquid, and the liquid can evaporate to the gas phase. A typical example would be water and water vapour. On the other hand, there are two (or more) different substances in a multicomponent flow, such as water and oil. In a multicomponent flow, the substances do not interconvert. Instead, an additional physics that we have to account for is the diffusion between these components. In practice, the distinction between a multiphase and a multicomponent flow is quite blurry. Many flows are in fact a mixture between the two. In cases where there is no transfer of material between the different fluid domains and inertia plays negligible roles (e.g. low Reynolds number flow), equivalent results are obtained whether we are using a multiphase or a multicomponent model, assuming the material properties (e.g. viscosities, surface tension) are the same.

Multiphase and multicomponent flows are important for a wide range of structured products and engineering applications. One important example is emulsions, which are mixtures of several immiscible liquids. They are ubiquitously exploited in food industry, pharmaceuticals, personal care, and enhanced oil recovery. Another major interest in multiphase and multicomponent flows is the effect of confinement by solid surfaces, e.g. in porous media and microfluidics. In particular, the different fluids may have different affinities to the surface. This is usually quantified by a material property called the contact angle, θ, where

$$\cos\theta = \frac{\gamma_{s\beta} - \gamma_{s\alpha}}{\gamma_{\alpha\beta}}. \tag{9.1}$$

$\gamma_{s\alpha}$, $\gamma_{s\beta}$ and $\gamma_{\alpha\beta}$ are respectively the surface tensions between the solid and fluid phase α, the solid and fluid phase β and the two fluid phases. Here the contact angle is defined with respect to phase α. The wetting properties may significantly affect

the fluid flow near the boundary. In fact, it has been demonstrated that we can take advantage of surface patterning, both chemically and topographically, to control the motion of fluids.

There are two different approaches to model multiphase and multicomponent flows: (i) sharp and (ii) diffuse interface models. In the sharp interface model, the interface is a two-dimensional boundary, which is usually respresented by a distinct computational mesh. The motion of this interface needs to be explicitly tracked, and there are many publications devoted to perform this efficiently. One needs a Navier-Stokes solver on either side of the boundary, and suitable boundary conditions on the fluid velocity and stress tensor need to be applied at the boundary. The models employed in the lattice Boltzmann community, on the other hand, usually belong to the diffuse interface approach. Here the interface spreads over several lattice spacings and its evolution is calculated by solving an advection-diffusion (Cahn-Hilliard) equation for the order parameter which distinguishes one fluid from another. The width of the interface sets a lower bound to the length scale which can be resolved, and usually we work in a regime where the simulation results should not depend on the interface width. A key advantage of the diffuse interface model is that the motion of the interface need not be tracked explicitly. All fluid nodes can be treated at equal footing whether they are in the bulk of the fluid or at the interface, and as such, diffuse interface models are very convenient for studying problems with complex surface geometries.

Within the lattice Boltzmann community itself, there are a variety of multiphase and multicomponent models. Broadly speaking, these models can be categorized into a bottom-up or a top-down approach. In a bottom-up approch, the starting point is often kinetic theory and some form of interactions are postulated between the fluids at the level of the Boltzmann equation. Reminiscent to many other lattice and particle based simulation techniques, separation between different fluid phases and components can be induced by tuning the interaction potentials. In a top-down approach, we start with by writing down the free energy of the fluids instead. The form of the free energy functional should capture intended features of the thermodynamics of the system, e.g. phase separation, surface tension between different fluids. The corresponding chemical potential, pressure tensor, and Boltzmann equation can then subsequently be derived.

In this chapter we will consider three popular classes of lattice Boltzmann models for simulating multiphase and multicomponent flows: (i) free energy, (ii) color, and (iii) pseudopotential models. This chapter is not meant to give an exhaustive review of all the possible variations. Instead, it should equip the readers to comprehend the most important ideas behind the methods.

9.2 Free-Energy Lattice Boltzmann Model

In this section we will focus on the free energy lattice Boltzmann models, covering both multicomponent and multiphase systems. It is worth emphasizing that these models are not unique. In fact, they are chosen primarily due to their simplicity.

9.2.1 Binary fluid model

Thermodynamics

The simplest multicomponent system is a binary fluid. To model a binary fluid, a common approach in any diffuse interface model is to introduce an order parameter ϕ. This order parameter, often also called the phase field, distinguishes the bulk of one fluid from another and corresponds to the relative concentration of the fluids. If C_1 and C_2 are the fractions of fluid 1 and 2 at a given point in space, then $\phi = C_1 - C_2$. Thus, we will use the convention where $\phi = 1$ signifies fluid α (e.g. water) and $\phi = -1$ fluid β (e.g. oil). The isosurface $\phi = 0$ then corresponds to the interface between the two fluids (e.g. oil-water interface).

Given these conventions, we want to construct a free energy functional which has two minima at $\phi = \pm 1$, and provides an energy penalty which scales with the area of the fluid-fluid interfaces. The constant of proportionality is the surface tension. The simplest model which captures this physics is given by the following Landau free energy:

$$\Psi = \int_V \psi_b dV = \int_V \left[\frac{A}{4} \left(\phi^2 - 1 \right)^2 + \frac{\kappa}{2} (\boldsymbol{\nabla} \phi)^2 \right] dV. \tag{9.2}$$

We have assumed the density of the two fluids are the same and constant. Otherwise, we need terms in the free energy functional which depend explicitly on the densities of the fluids. We will discuss such a case for the multiphase system.

Let us consider the fluid–fluid interface and derive an analytical expression for both the surface tension and the interface width. Minimising the free energy functional in Eq. (9.2) with respect to ϕ leads to the condition for equilibrium

$$\mu \equiv \frac{\delta \psi_b}{\delta \phi} = -A\phi + A\phi^3 - \kappa \nabla^2 \phi = 0, \tag{9.3}$$

where μ is the chemical potential. For simplicity, we shall now assume the interface is flat and located at $x = 0$. Remember that the bulk behaviours at $x = \pm\infty$ are such that $\phi = \pm 1$. Eq. (9.3) allows an interface solution of the form

$$\phi = \tanh \left(\frac{x}{\sqrt{2}\xi} \right), \tag{9.4}$$

where $\xi = \sqrt{\kappa/A}$ is defined as the interface width. The variable ξ is typically chosen to be of order a few lattice spacings (the method is unstable if the interface width is too small), and this is why models of this type are often called diffuse interface models, in contrast to sharp interface models where $\xi \to 0$. The surface tension, $\gamma_{\alpha\beta}$, of the liquid–liquid interface can then be calculated by integrating the free energy density across the interface. Using Eq. (9.37), we obtain

$$\gamma_{\alpha\beta} = \int_{-\infty}^{\infty} \left[\frac{A}{4} \left(\phi^2 - 1\right)^2 + \frac{\kappa}{2} (\boldsymbol{\nabla}\phi)^2 \right] dx = \sqrt{8\kappa A/9}. \tag{9.5}$$

When the flow is bounded by a solid surface, we need to account for the interactions between the fluids and the solid surface. There are several approaches to implement the wetting boundary condition (i.e. the contact angle). Here we will follow a thermodynamic approach. Following Cahn, we can prescribe a surface energy contribution,

$$\Psi_s = \int_A -h\phi_s dA\,, \tag{9.6}$$

where ϕ_s is the value of the order parameter at the surface and the integral is taken over the system's solid surface. Minimisation of the total free energy functional with respect to ϕ at the solid boundary,

$$\left[\frac{\delta(\Psi + \Psi_s)}{\delta\phi} \right]_{\text{boundary}} = 0\,, \tag{9.7}$$

leads to a Neumann boundary condition in the gradient of order parameter ϕ, as given by

$$\kappa \nabla_\perp \phi = -h\,. \tag{9.8}$$

The important consequence of this equation is that the contact angle of the surface can be implemented by setting the perpendicular derivative of the order parameter.

NOTE (HK: Make an exercise to derive the wetting boundary condition):

The variable h can be used to tuned the contact angle. For $h > 0$, fluid α interacts favourably with the solid (more wetting) compared to fluid β. The opposite is true for $h < 0$. To derive an explicit relation between the variable h and the contact angle θ, we can exploit Noether's theorem

$$\frac{\delta\psi_b}{\delta\boldsymbol{\nabla}\phi} \cdot \boldsymbol{\nabla}\phi - \psi_b = \text{constant} \tag{9.9}$$

to show that

$$\frac{\kappa}{2}(\boldsymbol{\nabla}\phi)^2 - \frac{A}{4}\left(\phi^2 - 1\right)^2 = \text{constant} = 0\,. \tag{9.10}$$

In the last step, we have used the fact that far from the interface (in the bulk), $\phi = \pm 1$ and $\boldsymbol{\nabla}\phi = 0$. Substituting Eq. (9.8) to Eq. (9.10), we find that the values of the order parameter at the surface may take four possible values, $\phi_s = \pm(1 \pm (2h^2/\kappa A)^{1/2})^{1/2}$. To decide which solutions are relevant, let us consider the following arguments. For fluid α, the bulk solution is given by $\phi = 1$. If $h > 0$, there is an energy gain

for having more positive order parameter at the solid surface. As such, we expect $\phi_{s\alpha} > 1$. On the other hand, if $h < 0$, it is favourable to have $\phi_{s\alpha} < 1$. A solution which satisfies this requirement is

$$\phi_{s\alpha} = +\sqrt{1 + \left(\frac{2}{\kappa A}\right)^{1/2} h}. \tag{9.11}$$

Using similar arguments, we can conclude that for fluid β,

$$\phi_{s\beta} = -\sqrt{1 - \left(\frac{2}{\kappa A}\right)^{1/2} h}. \tag{9.12}$$

The fluid–solid surface tensions can be calculated in a similar way to the fluid–fluid surface tension, except that now we also have to take into account the contributions from the surface energy term. For the fluid α–solid surface tension, assuming a flat solid interface at $x = 0$,

$$\gamma_{s\alpha} = -h\phi_{s\alpha} + \int_{x=0}^{\infty} \left[\frac{A}{4}\left(\phi^2 - 1\right)^2 + \frac{\kappa}{2}(\boldsymbol{\nabla}\phi)^2\right] dx. \tag{9.13}$$

To evaluate this integral, we shall take advantage the Noether's theorem given in Eq. (9.10) and use a change of variable, such that

$$\int_{x=0}^{\infty} \left[\frac{A}{4}\left(\phi^2 - 1\right)^2 + \frac{\kappa}{2}\left(\frac{d\phi}{dx}\right)^2\right] dx = \int_{\phi_{s\alpha}}^{\phi=1} \left[\sqrt{\frac{A\kappa}{2}}(1 - \phi^2)\right] d\phi.$$

Using the definition of $\phi_{s\alpha}$ given in Eq. (9.11), it is straightforward to show that

$$\gamma_{s\alpha} = \frac{\gamma_{\alpha\beta}}{2}\left[1 - (1 + \Omega)^{3/2}\right]. \tag{9.14}$$

where $\Omega = \sqrt{\frac{2}{\kappa A}} h$ and $\gamma_{\alpha\beta} = \sqrt{8\kappa A/9}$. The fluid β–solid surface tension can be derived in a similar way, and we obtain

$$\gamma_{s\beta} = \frac{\gamma_{\alpha\beta}}{2}\left[1 - (1 - \Omega)^{3/2}\right]. \tag{9.15}$$

The contact angle follows from substituting the values of the surface tensions into Young's law, Eq. (9.1), to give (with θ defined as the contact angle of the α-phase)

$$\cos\theta = \frac{\gamma_{s\beta} - \gamma_{s\alpha}}{\gamma_{\alpha\beta}} = \frac{(1 + \Omega)^{3/2} - (1 - \Omega)^{3/2}}{2}. \tag{9.16}$$

Eq. (9.16) can be inverted to give a relation between the phenomenological parameter h and the equilibrium contact angle θ

$$\Omega = 2\ \mathrm{sign}\left(\frac{\pi}{2}-\theta\right)\sqrt{\cos\left(\frac{\alpha}{3}\right)\left\{1-\cos\left(\frac{\alpha}{3}\right)\right\}}\,, \tag{9.17}$$

where $\alpha = \cos^{-1}\left(\sin^2\theta\right)$ and the function sign returns the sign of its argument.

Equations of Motion

Before we write down the lattice Boltzmann equations for binary fluid, let us review the corresponding continuum equations of motion. As described in section **??**, the fluid motion is described by the continuity and Navier-Stokes equations,

$$\partial_t\rho + \partial_\gamma\left(\rho u_\gamma\right) = 0, \tag{9.18}$$

$$\partial_t(\rho u_\alpha) + \partial_\beta\left(\rho u_\alpha u_\beta\right) = -\partial_\alpha p + \partial_\beta\eta\left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right) - \phi\partial_\alpha\mu. \tag{9.19}$$

The key additional physics due to the thermodynamics of binary fluid is contained in the last term of Eq. (9.19). In equilibrium, the chemical potential has to be the same everywhere. Any inhomogeneity leads to a body force proportional to the gradient of the chemical potential. The application of this interfacial force directly follows the algorithm described in Chapter 6. It is worth noting that for numerical stabilities, the interfacial force term can be equivalently written as $\phi\partial_\alpha\mu \to -\mu\partial_\alpha\phi$. This is the implementation that is most often used in the literature.

The interfacial term can also be taken into account within the definition of the pressure tensor, and traditionally, this is the approach taken in the free energy lattice Boltzmann community. For concreteness, this is what we will do here. The pressure tensor thus needs to satisfy the condition

$$\partial_\beta P_{\alpha\beta} = \partial_\alpha p + \phi\partial_\alpha\mu. \tag{9.20}$$

In our notation, p corresponds to other isotropic contributions to the pressure tensor outside the binary fluid thermodynamics described above. In lattice Boltzmann method, this is usually $p = \rho c_\mathrm{s}^2$. Using the definition of μ in Eq. (9.3), it follows that

$$P_{\alpha\beta} = \left(p_b - \tfrac{\kappa}{2}(\partial_\gamma\phi)^2 - \kappa\phi\partial_{\gamma\gamma}\phi\right)\delta_{\alpha\beta} + \kappa(\partial_\alpha\phi)(\partial_\beta\phi)\,, \tag{9.21}$$

$$p_b = \rho c_\mathrm{s}^2 + A\left(-\tfrac{1}{2}\phi^2 + \tfrac{3}{4}\phi^4\right). \tag{9.22}$$

p_b can be interpreted as the bulk pressure far from the interface, where the gradient terms are zero. In this model, the value of ϕ usually deviates from ±1 in the bulk when there is a Laplace pressure difference between the two fluid domains.

The order parameter itself evolves through a advection-diffusion (Cahn-Hilliard) equation,

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} + \boldsymbol{\nabla}\cdot(\phi\boldsymbol{u}) = M\nabla^2\mu. \tag{9.23}$$

The second term on the left hand side is the advection term, where the order parameter moves with together with the fluid. The diffusive term on the right hand side accounts for motion of the order parameter due to inhomogeneities in the chemical potential.

The Lattice Boltzmann Algorithm

We now define a lattice Boltmann algorithm which solves Eq. (9.18), Eq. (9.19), and Eq. (9.23). For a binary fluid, we need to define two distribution functions, $f_i(\mathbf{r},t)$ and $g_i(\mathbf{r},t)$, corresponding to the density and relative concentration of the two fluids. The physical variables are related to the distribution functions by

$$\rho = \sum_i f_i, \qquad \rho u_\alpha = \sum_i f_i e_{i\alpha}, \qquad \phi = \sum_i g_i, \qquad \phi u_\alpha = \sum_i g_i e_{i\alpha}. \tag{9.24}$$

Similar to the single fluid case, the lattice Boltzmann algorithm for multicomponent flows can be broken into two steps. For simplicity, here we have used the standard BGK single relaxation time approach. Extensions to multiple relaxation time approaches follow the same route as described in Chapter 10.

$$\begin{aligned}
\text{Collision step}: \quad & f_i'(\mathbf{r},t) = f_i(\mathbf{r},t) - \tfrac{1}{\tau}\left[f_i(\mathbf{r},t) - f_i^{eq}(\mathbf{r},t)\right], \\
& g_i'(\mathbf{r},t) = g_i(\mathbf{r},t) - \tfrac{1}{\tau_\phi}\left[g_i(\mathbf{r},t) - g_i^{eq}(\mathbf{r},t)\right], \\
\text{Propagation step}: \quad & f_i(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t) = f_i'(\mathbf{r},t), \\
& g_i(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t) = g_i'(\mathbf{r},t).
\end{aligned} \tag{9.25}$$

f_i^{eq} and g_i^{eq} are local equilibrium distribution functions. The relaxation parameters τ and τ_ϕ in the lattice Boltzmann algorithm are related to the parameters in the hydrodynamic equations ν and M through

$$\nu = c_s^2\left(\tau - \tfrac{\Delta t}{2}\right), \tag{9.26}$$

$$M = \Gamma\left(\tau_\phi - \tfrac{\Delta t}{2}\right), \tag{9.27}$$

where Γ is a tunable parameter that appears in the equilibrium distribution. Since ν and M are positive quantities, the values of the relaxation times τ and τ_ϕ have to be larger than $1/2$.

The Chapman-Enskog analysis to derive the Navier-Stokes equation from the lattice Boltzmann equations, Eq. (9.25), is the same as before. The corresponding analysis for the order parameter is the subject of our next exercise. Importantly, following the discussions in Chapter 3, Eq. (9.25) reproduce Eq. (9.18), Eq. (9.19) and Eq. (9.23) in the continuum limit if the correct thermodynamic and hydrodynamic information is input to the simulation by a suitable choice of local equilibrium functions. By analogy to Eq. (3.69), the constraints that need to be satisfied are

$$\sum_i f_i^{eq} = \rho\,,\ \sum_i f_i^{eq} e_{i\alpha} = \rho u_\alpha\,,\ \sum_i f_i^{eq} e_{i\alpha} e_{i\beta} = P_{\alpha\beta} + \rho u_\alpha u_\beta, \tag{9.28}$$

$$\sum_i f_i^{eq} e_{i\alpha} e_{i\beta} e_{i\gamma} = \rho c_s^2 [u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\gamma\alpha} + u_\gamma \delta_{\alpha\beta}]\,, \tag{9.29}$$

$$\sum_i g_i^{eq} = \phi\,,\ \sum_i g_i^{eq} e_{i\alpha} = \phi u_\alpha\,,\ \sum_i g_i^{eq} e_{i\alpha} e_{i\beta} = \Gamma\mu\delta_{\alpha\beta} + \phi u_\alpha u_\beta\,. \tag{9.30}$$

Compared to the single phase lattice Boltzmann algorithm, we need to generalise the pressure term in Eq. (9.28) to account for thermodynamics of the binary fluid, which in return provide us the physics of Laplace pressure and surface tension. It is worth emphasizing that, if the $\phi\partial_\alpha\mu$ term in Eq. (9.19) is implemented as a forcing term, then we need not do this. Otherwise, we are double-counting the terms. For the order parameter, the zeroth and first moment of the equilibrium distribution functions need to satisfy mass and momentum conservations. The second moment of g^{eq}'s also has the same structure as for f^{eq}'s, except now it is related to the chemical potential rather than the pressure tensor. The forms of f_i^{eq} and g_i^{eq} that satisfy the above constraints are

$$\begin{aligned} f_i^{\mathrm{eq}} = w_i\rho\left(1 + \frac{c_{i\alpha}u_\alpha}{c_s^2} + \frac{u_\alpha u_\beta\left(c_{i\alpha}c_{i\beta} - c_s^2\delta_{\alpha\beta}\right)}{2c_s^4}\right) \\ + \frac{w_i}{c_s^2}\left(-\frac{A}{2}\phi^2 + \frac{3A}{4}\phi^4 - \kappa\phi\nabla^2\phi\right) + \frac{\kappa}{c^2}\sum_{\alpha,\beta} w_i^{\alpha\beta}\partial_\alpha\phi\partial_\beta\phi \end{aligned} \tag{9.31}$$

$$g_i^{\mathrm{eq}} = w_i\left(\frac{\Gamma\mu}{c_s^2} + \frac{c_{i\alpha}\phi u_\alpha}{c_s^2} + \frac{\phi u_\alpha u_\beta\left(c_{i\alpha}c_{i\beta} - c_s^2\delta_{\alpha\beta}\right)}{2c_s^4}\right), \tag{9.32}$$

for $i \neq 0$, and

$$f_0^{\mathrm{eq}} = \rho - \sum_{i\neq 0} f_i^{\mathrm{eq}}, \quad g_0^{\mathrm{eq}} = \phi - \sum_{i\neq 0} g_i^{\mathrm{eq}}. \tag{9.33}$$

The coefficients w_i's are the same as before. In fact, the first line in eq. (9.47) is identical to the equilibrium distribution function for a one-component lattice Boltzmann method. The $w_i^{\alpha\beta}$ coefficients are chosen to minimise the spurious velocities at the interface. One possible strategy was proposed by Pooley and Furtado, where they derived for D3Q19, [**?**]

$$
\begin{aligned}
w^{xx}_{1,2} = w^{yy}_{3,4} = w^{zz}_{5,6} &= \tfrac{5}{12},\\
w^{xx}_{3\text{-}6} = w^{yy}_{1,2,5,6} = w^{zz}_{1\text{-}4} &= -\tfrac{1}{3},\\
w^{xy}_{1\text{-}6} = w^{yz}_{1\text{-}6} = w^{zx}_{1\text{-}6} &= 0,\\
w^{xx}_{7\text{-}10,13\text{-}16} = w^{yy}_{7\text{-}8,11\text{-}14,17\text{-}18} = w^{zz}_{9\text{-}12,15\text{-}18} &= -\tfrac{1}{24},\\
w^{xx}_{11\text{-}12,17\text{-}18} = w^{yy}_{9\text{-}10,15\text{-}16} = w^{zz}_{7\text{-}8,13\text{-}14} &= \tfrac{1}{12},\\
w^{xy}_{7,8} = w^{yz}_{11,12} = w^{zx}_{9,10} &= \tfrac{1}{4},\\
w^{xy}_{13,14} = w^{yz}_{17,18} = w^{zx}_{15,16} &= -\tfrac{1}{4},\\
w^{xy}_{9\text{-}12,15\text{-}18} = w^{yz}_{7\text{-}10,13\text{-}16} = w^{zx}_{7\text{-}8,11\text{-}14,17\text{-}18} &= 0.
\end{aligned}
$$

Other lattices can be analysed in the same manner.

NOTE (HK: Exercise to show the equilibrium distribution functions do satisfy the constraints.):

NOTE (HK: Chapman-Enskog expansion for the order parameter):

9.2.2 Liquid-gas model

Thermodynamics

In this subsection we will turn our attention to a multiphase free energy lattice Boltzmann model. Compared to the multicomponent model where we have chosen the fluid concentration as the order parameter, a more suitable choice for a multiphase flow is the density. A wide variety of models are possible, including the classical van der Walls model. But here, for pedagogical reasons, we will focus on the following choice for the Landau free energy

$$
\Psi = \int_V \psi_b dV = \int_V \left[p_c(\nu_\rho^2 - \beta\tau_w)^2 + \mu_0\rho - p_0 + \frac{\kappa}{2}(\boldsymbol{\nabla}\rho)^2 \right] dV, \tag{9.34}
$$

subject to the constant total mass constraint $\int_V \rho dV$ = constant, and where $\nu_\rho = (\rho - \rho_c)/\rho_c$ and $\tau_w = (T_c - T)/T_c$.

In the above equation ρ_c, p_c and T_c are the density, pressure, and temperature at the critical point of the material. It is easy to see that below T_c, the system will favour two bulk solutions $\nu_\rho = (\rho - \rho_c)/\rho_c = \pm\sqrt{\beta\tau_w}$. The plus branch corresponds to the liquid state, while the minus branch is the gas state. Above the critical temperature, the system cannot exhibit liquid-gas coexistence. Mathematically speaking, the solution for ν_ρ becomes imaginary above T_c in this model. The constants μ_0 and p_0 are the reference chemical potential and pressure of the fluid. We will describe the relevance of these parameters in more details below.

The careful readers will notice that this choice of free energy has the same structure to that we chose for the binary model. As such, we can derive the relevant physical quantities in pretty much the same manner. Let us start by computing the

chemical potential

$$\mu \equiv \frac{\delta \psi_b}{\delta \rho} = \frac{4p_c}{\rho_c}\nu_\rho(\nu_\rho^2 - \beta\tau_w) + \mu_0 - \kappa\rho_c\nabla^2\nu_\rho. \tag{9.35}$$

When the system is in one of the bulk free energy minimum solutions, either in the liquid or in the gas phase, $\nu_\rho^2 - \beta\tau_w = 0$, and the gradient term vanishes. As such, $\mu = \mu_0$. This is the free energy density needed to add materials in the bulk liquid or gas phase.

In addition to the two homogeneous solutions, we can also look for a solution which contains a liquid-gas interface. Assuming the interface is flat and is located $x = 0$, the differential equation

$$\frac{4p_c}{\rho_c}\nu_\rho(\nu_\rho^2 - \beta\tau_w) - \kappa\rho_c\nabla^2\nu_\rho = 0, \tag{9.36}$$

with boundary conditions $\nu_\rho = \pm\sqrt{\beta\tau_w}$ for $x = \pm\infty$, has the following solution

$$\nu_\rho = \sqrt{\beta\tau_w}\tanh\left(\frac{x}{\sqrt{2}\xi}\right). \tag{9.37}$$

$\xi = \sqrt{(\kappa\rho_c^2)/(4\beta\tau_w p_c)}$ is defined as the interface width and is usually chosen to be a couple of lattice spacings. The surface tension, γ_{lg}, of the liquid–gas interface can then be calculated by integrating the free energy density across the interface,

$$\gamma_{lg} = \int_{-\infty}^{\infty}\left[p_c(\nu_\rho^2 - \beta\tau_w)^2 + \frac{\kappa}{2}(\boldsymbol{\nabla}\rho)^2\right]dx = \frac{4}{3}\sqrt{2\kappa p_c}(\beta\tau_w)^{3/2}\rho_c. \tag{9.38}$$

We have ignored the $\mu_0\rho$ and p_0 terms from the integral because they only contribute to a constant given a simulation setup. Remember that our lattice Boltzmann simulations are subject to a total mass constraint, $\int_V \rho dV = \text{constant}$.

The wetting boundary condition can be introduced in the same way as for the binary model. Following Cahn, we introduce a surface energy term

$$\Psi_s = \int_A -h\rho_s dA\,, \tag{9.39}$$

where ρ_s is the value of the density at the surface and the integral is taken over the system's solid surface. Minimisation of the total free energy functional with respect to ρ at the solid boundary,

$$\left[\frac{\delta(\Psi + \Psi_s)}{\delta\rho}\right]_{\text{boundary}} = 0\,, \tag{9.40}$$

leads to a Neumann boundary condition in the gradient of the density ρ, as given by

$$\kappa\nabla_\perp\rho = -h\,. \tag{9.41}$$

We will leave it as an exercise for the readers that the parameter h is related to the liquid contact angle θ via eq. (9.17), except now $\Omega = h/(\beta\tau_w\sqrt{2\kappa p_c})$.

Equations of Motion

In a multiphase model, the essential continuum equations of motion are described by the continuity and Navier-Stokes equations,

$$\partial_t \rho + \partial_\gamma \left(\rho u_\gamma\right) = 0, \tag{9.42}$$

$$\partial_t(\rho u_\alpha) + \partial_\beta \left(\rho u_\alpha u_\beta\right) = -\partial_\beta P_{\alpha\beta} + \partial_\beta \eta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right). \tag{9.43}$$

There is no need to solve an additional advection-diffusion (Cahn-Hilliard) equation. As before, the pressure tensor $P_{\alpha\beta}$ can be derived (up to a constant contribution in the pressure tensor) by requiring

$$\partial_\beta P_{\alpha\beta} = \rho\partial_\alpha \mu. \tag{9.44}$$

Using the definition of μ in Eq. (9.35), it follows that

$$P_{\alpha\beta} = \left(p_b - \tfrac{\kappa}{2}(\partial_\gamma\rho)^2 - \kappa\rho\partial_{\gamma\gamma}\rho\right)\delta_{\alpha\beta} + \kappa(\partial_\alpha\rho)(\partial_\beta\rho), \tag{9.45}$$

$$p_b = p_c(\nu_\rho + 1)^2(3\nu_\rho^2 - 2\nu_\rho + 1 - 2\beta\tau_w). \tag{9.46}$$

p_b can be interpreted as the bulk pressure far from the interface, where the gradient terms are zero. When there is a Laplace pressure difference between the liquid and the gas phases, the values of ρ deviate slightly from $\pm\beta\tau_w$ to accommodate this pressure difference.

The Lattice Boltzmann Algorithm

In the free energy approach, the lattice Boltzmann method for a multiphase flow differs from the standard single-phase (and single-component) algorithm only in the definition of the pressure tensor, and subsequently the equilibrium distribution functions, f_i^{eq}. The rest of the simulation algorithm is unchanged. The suitable form of f_i^{eq} is given by

$$\begin{aligned} f_i^{\mathrm{eq}} = w_i\rho\left(1 + \frac{c_{i\alpha}u_\alpha}{c_\mathrm{s}^2} + \frac{u_\alpha u_\beta\left(c_{i\alpha}c_{i\beta} - c_\mathrm{s}^2\delta_{\alpha\beta}\right)}{2c_\mathrm{s}^4}\right) \\ + \frac{w_i}{c_s^2}\left(p_b - \rho c_\mathrm{s}^2 - \kappa\rho\nabla^2\rho\right) + \frac{\kappa}{c^2}\sum_{\alpha,\beta} w_i^{\alpha\beta}\partial_\alpha\rho\partial_\beta\rho \end{aligned} \tag{9.47}$$

for $i \neq 0$, and

$$f_0^{\mathrm{eq}} = \rho - \sum_{i \neq 0} f_i^{\mathrm{eq}}. \tag{9.48}$$

9.3 Shan-Chen

9.4 Color Model

Chapter 10
MRT and TRT collision operators

The BGK model is an elegant way to simplify the collision operator of the Boltzmann equation. This comes at the cost of reduced accuracy (in particular for large viscosity) and stability (especially at small viscosity). Multi relaxation time (MRT) collision operators offer a larger number of free parameters that can be tuned to overcome these problems.

After a short introduction to the main concept of these operators in section 10.1, we will discuss the advantages of performing relaxation in momentum rather than in population space (section 10.2). These ideas lead to the general MRT algorithm (section 10.3). We discuss two different approaches, the Hermite polynomials and the Gram-Schmidt procedures, demonstrated on the basis of the D2Q9 lattice (section 10.4). Results for the common 3D lattices are given in appendix A.6. We also address the inclusion of forces into the MRT collision operator (section 10.5).

The two relaxation time (TRT) collision operator (section 10.6) can be viewed as a reduced MRT model which is nearly as simple as the BGK model but still offers significant advantages. We conclude the chapter with an overview of practical guidelines as how to choose the collision model and the relaxation rates (section 10.7).

10.1 Introduction

Under certain conditions, the BGK collision operator approaches its limits of accuracy and/or stability.

For example, stability issues can often occur for large Reynolds number simulations. In order to avoid increasing the grid resolution, a common approach is to reduce the viscosity or to increase the velocity of the magnitude. As discussed in section 4.4, small viscosities (relaxation times τ close to $1/2$) and large velocity magnitudes ($\boldsymbol{u} < c_s$) can lead to stability problems. In the end, the only viable solution for the BGK collision operator is to increase the grid size, which makes the simulations more expensive.

Furthermore, the accuracy of simulations based on the BGK collision operator varies with the relaxation time τ. This applies to both the bulk LB algorithm (section 4.5) and some boundary conditions. For example, the wall location depends on the numerical value of τ when the standard bounce-back algorithm is used (section 5.2.3). This issue can be easily demonstrated by simulating a Poiseuille flow [1, 2, 3]. The error is found to increase with τ [4, 5]. This problem is particularly challenging for porous media simulations where the apparent porosity strongly depends on the exact wall location [6]. Ideally, one wants to obtain so-called viscosity-independent numerical solutions [7] which only depend on the physical parameters (such as the Reynolds number), but not the relaxation time τ.

There is also a problem with *Galilean invariance*. Galilean invariance means that physical phenomena occur in the same way in all inertial systems, *i.e.* in systems moving with constant velocities relatively to each other. The LBM is known to violate exact Galilean invariance [8, 9]. The reason is that the second-order Hermite series expansion in eq. (3.56) is not sufficient to guarantee Galilean invariance [10], causing an $O(u^3)$ error term in the macroscopic dynamics as shown in Section 4.1.

Overall, these examples indicate that something else beyond the hydrodynamic level influences the overall accuracy and stability of the LBM as a Navier-Stokes solver. We have to understand that the relaxation time τ is a kinetic parameter, and its relation to viscosity has to be analysed *via* the Chapman-Enskog analysis (section 4.1). Contrarily, all other parameters (*e.g.* velocity, lattice size) are directly connected to macroscopic quantities. Essentially, this is the reason why the numerical BGK solutions (and their stability/accuracy) depend on τ [7, 11, 12, 13]. As a result, we require more insight to perform more accurate and stable LB simulations.

Since τ enters the BGK equation only through the collision operator, the underlying idea is to revise and improve the collision step. A general requirement for the collision operator to recover the Navier-Stokes equation is the conservation of density and momentum. So far, we have considered the simplest form of such an operator: the BGK model. However, as we will see in the following, it is possible to construct collision operators that have more degrees of freedom than the BGK operator and can be used to improve accuracy and stability. The models discussed here are the *multiple relaxation times* (MRT) and the *two relaxation times* (TRT) collision operators.

Let us consecutively present an idea behind MRT and TRT collision operators starting from the BGK collision operator. The BGK collision operator uses only one relaxation rate $\omega = 1/\tau$ for all populations, *i.e.* $\Omega_i = -\omega(f_i - f_i^{\mathrm{eq}})$. The first naive idea is to introduce different collision rates for each population: $\Omega_i = -\omega_i(f_i - f_i^{\mathrm{eq}})$. However, it is easy to see that one needs certain constraints to satisfy the conservation laws for density and momentum (in the absence of forces):

$$\begin{aligned} 0 = \sum_i \Omega_i = -\sum_i \omega_i \left(f_i - f_i^{\mathrm{eq}}\right) \qquad &\Longrightarrow \qquad \sum_i \omega_i f_i = \sum_i \omega_i f_i^{\mathrm{eq}}, \\ 0 = \sum_i \Omega_i \boldsymbol{c}_i = -\sum_i \omega_i \left(f_i - f_i^{\mathrm{eq}}\right) \boldsymbol{c}_i \qquad &\Longrightarrow \qquad \sum_i \omega_i f_i \boldsymbol{c}_i = \sum_i \omega_i f_i^{\mathrm{eq}} \boldsymbol{c}_i. \end{aligned} \tag{10.1}$$

Eq. (10.1) is satisfied in the case of the BGK collision operator ($\omega_i = \omega$). For a general situation with distinct values of ω_i, however, the situation is more complicated and not obvious. Thus, we skip this idea due to complexity [14].

The next, slightly more sophisticated, idea is to use velocity *moments* (section 1.3.4). Apart from the fact that Hermite velocity polynomial moments provide the mathematical foundation for the LBM, one can see that moments are already used in the BGK collision operator. Indeed, let us take another careful look at eq. (10.1), *i.e.* conservation laws rewritten for the BGK operator:

$$\begin{aligned} 0 &= \sum_i \Omega_i = -\sum_i \omega \left(f_i - f_i^{\mathrm{eq}}\right) = -\omega \left(\rho - \rho^{\mathrm{eq}}\right), \\ 0 &= \sum_i \Omega_i \boldsymbol{c}_i = -\sum_i \omega \left(f_i \boldsymbol{c}_i - f_i^{\mathrm{eq}} \boldsymbol{c}_i\right) = -\omega \left(\boldsymbol{j} - \boldsymbol{j}^{\mathrm{eq}}\right). \end{aligned} \tag{10.2}$$

Here, the density ρ and momentum $\boldsymbol{j}$ are the zeroth and first velocity moments. In the BGK model, all moments are relaxed with a single relaxation rate ω. However, the moments can in principle be relaxed with individual rates. This is the basic idea behind the MRT collision operators, *i.e.* to individually control hydrodynamic moments to achieve better accuracy and stability.

The major change compared to the BGK model is that MRT collisions are performed in *moment space*. Thus, we first have to map the populations f_i to moment space, then collide (*i.e.* relax) the moments toward equilibrium, and finally map the relaxed moments back to population space. We will investigate the underlying math in section 10.2.

TODO (TK): Here we should add a short paragraph about other collision operators which we will not cover (*e.g.* entropic, regularised). NOTE (EMV): We can cover regularised LB very quickly by saying that it reconstructs the entire distribution function in collisions in a way that is equivalent to choosing $\omega_i = 1$ (*i.e.* immediate relaxation) for moments whose relaxation rate do not affect the viscosity (*i.e.* nonhydrodynamic moments, though we have not defined that term yet) [15, 16].

10.2 Moment space and transformations

We will now sketch the general mathematical procedure behind the MRT collision operators. In particular, transformations from population to moment space and back will be covered.

In section 3.3 (*cf.* eq. (3.59)) we introduced moments as certain summations over populations with Hermite polynomials:

$$\boldsymbol{a}^{(n)} = \sum_i f_i \boldsymbol{H}_i^{(n)}. \tag{10.3}$$

We can use a similar definition here and say that the moment m_k in a DdQq lattice can be found through a $q \times q$ matrix as:

$$m_k = \sum_{i=0}^{q-1} M_{ki} f_k \qquad \text{for } k = 0, \ldots, q-1. \tag{10.4}$$

This equation can be rewritten in vector-matrix form:

$$\boldsymbol{m} = \boldsymbol{M}\boldsymbol{f}, \quad \boldsymbol{m} = \begin{pmatrix} m_0 \\ \vdots \\ m_{q-1} \end{pmatrix}, \quad \boldsymbol{M} = \begin{pmatrix} M_{0,0} & \cdots & M_{0,q-1} \\ \vdots & \ddots & \vdots \\ M_{q-1,0} & \cdots & M_{q-1,q-1} \end{pmatrix}, \quad \boldsymbol{f} = \begin{pmatrix} f_0 \\ \vdots \\ f_{q-1} \end{pmatrix}. \tag{10.5}$$

The matrix $\boldsymbol{M}$ with elements M_{ki} can also be interpreted as a q-tuple of row vectors $\boldsymbol{M}_i$, where $\boldsymbol{M}_k = (M_{k,0} \cdots M_{k,q-1})$ and $m_k = \boldsymbol{M}_k \boldsymbol{f}$. Generally speaking, one obtains q moments m_k from the q populations f_i through the *linear* transformation from *population* to *moment space* in eq. (10.5).

The basic idea behind the MRT collision operator is to relax *moments* (rather than populations) with individual rates. By carefully choosing the transformation matrix $\boldsymbol{M}$, the obtained moments m_k can be made to directly correspond to hydrodynamic moments. Thus, it is possible to affect those hydrodynamic terms (density, momentum, momentum flux tensor, etc) individually by choosing different distinct relaxation rates. Thus, the MRT collision operator has more flexibility to affect hydrodynamics through tuning its moments individually. In contrast, the BGK collision operator tunes all hydrodynamic moments with one relaxation rate ω.

So far, we have not yet specified how the MRT operator looks like in detail, but we do know that the BGK operator is the special case of the MRT collision operator where all relaxation rates are identical, *i.e.* $\omega_i = \omega$. We can use this relation to derive the MRT operator from the BGK operator.

First we write the LBGK equation in vector form:

$$\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{c}_i, t+1) - \boldsymbol{f}(\boldsymbol{x}, t) = -\omega \left(\boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{f}^{\mathrm{eq}}(\boldsymbol{x}, t) \right). \tag{10.6}$$

The collision step is not changed when multiplied by the identity matrix $\boldsymbol{I} = \boldsymbol{M}^{-1}\boldsymbol{M}$ (assuming that $\boldsymbol{M}$ can be inverted):

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{c}_i, t+1) - \boldsymbol{f}(\boldsymbol{x}, t) &= -\boldsymbol{M}^{-1}\boldsymbol{M}\omega \left(\boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{f}^{\mathrm{eq}}(\boldsymbol{x}, t) \right) \\ &= -\boldsymbol{M}^{-1}\omega \left(\boldsymbol{M}\boldsymbol{f}(\boldsymbol{x}, t) - \boldsymbol{M}\boldsymbol{f}^{\mathrm{eq}}(\boldsymbol{x}, t) \right) \\ &= -\boldsymbol{M}^{-1}\omega\boldsymbol{I} \left(\boldsymbol{m}(\boldsymbol{x}, t) - \boldsymbol{m}^{\mathrm{eq}}(\boldsymbol{x}, t) \right) \\ &= -\boldsymbol{M}^{-1}\boldsymbol{S} \left(\boldsymbol{m}(\boldsymbol{x}, t) - \boldsymbol{m}^{\mathrm{eq}}(\boldsymbol{x}, t) \right). \end{aligned} \tag{10.7}$$

We have introduced a diagonal matrix $\boldsymbol{S} = \omega\boldsymbol{I} = \mathrm{diag}(\omega, \ldots, \omega)$ and the equilibrium moment vector $\boldsymbol{m}^{\mathrm{eq}} = \boldsymbol{M}\boldsymbol{f}^{\mathrm{eq}}$.

Let us take a detailed look at eq. (10.7). The expression $\boldsymbol{S}(\boldsymbol{m} - \boldsymbol{m}^{\mathrm{eq}})$ is the relaxation of all *moments* with one rate ω. The result is then multiplied by the inverse matrix $\boldsymbol{M}^{-1}$. This step represents the transformation from moment space back to population space. The left-hand-side of eq. (10.7) is the usual streaming step.

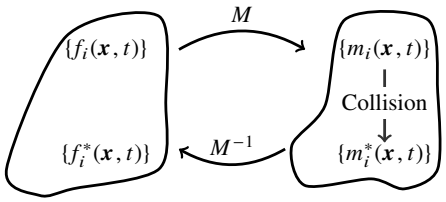


Fig. 10.1: Schematic representation of MRT LBM. Populations are mapped to moment space where collision (relaxation) takes place. Relaxed moments are transformed back to population space before streaming. The superscript * denotes post-collision values. TODO (TK): Revise figure: adapt symbols and notation. NOTE (EMV): You may want to represent streaming on the LHS, too.TODO (TK): Please change m_i to m_k

The fact that the matrix $\boldsymbol{S}$ contains only one parameter ω reflects that the BGK operator is a single-relaxation model. It is straightforward to introduce individual collision rates for every moment, which leads to the *relaxation matrix*

$$\boldsymbol{S} = \begin{pmatrix} \omega_0 & 0 & \dots & 0 \\ 0 & \omega_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \omega_{q-1} \end{pmatrix}. \tag{10.8}$$

The relaxation matrix $\boldsymbol{S}$ together with the operation $\boldsymbol{S}(\boldsymbol{m} - \boldsymbol{m}^{\mathrm{eq}})$ is already a general MRT collision operator we can write.

The basic idea behind the MRT collision operator is a mapping from population to moment space *via* a transformation matrix $\boldsymbol{M}$. This allows to relax moments rather than populations with individual rates (in form of a relaxation matrix $\boldsymbol{S}$). The relaxed moments are then transformed back to population space where streaming is performed as usual. This procedure is shown in fig. 10.1.

In section 10.3 we will present the MRT algorithm more thoroughly, but first we must discuss the moments and their meaning. We know that the density and the momentum are conserved. Those conserved quantities could act as moments in the MRT framework:

For example, the zeroth moment corresponds to the density:

$$m_0 = \rho = \sum_i f_i. \tag{10.9}$$

By expanding the zeroth order in terms of populations $m_0 = \sum_i M_{0,i} f_i$, one can find the entire first row of the matrix $\boldsymbol{M}$ as $M_{0,i} = 1$. Since the first row is related to the density ρ, we introduce the notation $M_i^{(\rho)} = M_{0,i}$ $(i = 0, \ldots, q-1)$.

Next we have momentum, which has two components in 2D and three in 3D. For the x-component we can write:

$$m_1 = j_x = \rho u_x = \sum_i f_i c_{ix}. \tag{10.10}$$

A similar equation holds for $m_2 = j_y$ and, if in 3D, also for $m_3 = j_z$. From this we can find the components of the matrix $\boldsymbol{M}$ corresponding to the momentum: $M_i^{(j_\alpha)} = c_{i\alpha}$ with $\alpha = x, y$ in 2D and $\alpha = x, y, z$ in 3D.

For the matrix $\boldsymbol{M}$ we have specified so far only three row vectors in 2D and four row vectors in 3D. To specify the other moments (and therefore the other missing components of the matrix $\boldsymbol{M}$), two approaches are typically used:

- **Hermite polynomials:** The most straightforward approach is to use Hermite polynomials which are also used in the discretisation of the continuous Boltzmann equation. We defined moments in eq. (3.59) as $\boldsymbol{a}^{(n)} = \sum_i \boldsymbol{H}_i^{(n)} f_i$ (recall that $\boldsymbol{H}^{(n)}$ is the tensor with n components $H^{(n)}_{\alpha_1 \ldots \alpha_n}$ where $\alpha_{1,\ldots,n} = \{x, y, z\}$). Comparing this definition with the general definition of moments in eq. (10.4), the missing matrix components of $\boldsymbol{M}$ are the values of Hermite polynomials $M_{ki} = H^{(n)}_{\alpha_1 \alpha_2 \ldots \alpha_n}(\boldsymbol{c}_i) = H^{(n)}_{\alpha_1 \alpha_2 \ldots \alpha_n i}$, where indices $\alpha_{1,\ldots,n}$ are uniquely corresponded to the index k. Density corresponds to the zeroth order polynomial, and momentum to the first (two vectors in 2D correspond to x and y, three vectors in 3D correspond to x, y and z components of the Hermite polynomial $\boldsymbol{H}^{(1)}$). Other Hermite moments are at least of second order in velocity. Despite its straightforwardness, this approach is not very common. It is mainly used for the D2Q9 lattice [17, 18, 19].
- **Gram-Schmidt procedure:** This is based on the idea of constructing a set of orthogonal vectors. We start the process with the vectors for the known moments (density ρ and momentum $\boldsymbol{j}$). The next step is to take a combination of the the velocity vectors $c_{i\alpha}$ of appropriate order and find the coefficients in such a way that the resulting vector is orthogonal to all previously found ones. By repeating this process, one can construct the entire matrix M_{ki} consisting of orthogonal row vectors $\boldsymbol{M}_k$.

Both approaches will be thoroughly presented in section 10.4 where we will construct $\boldsymbol{M}$ for the D2Q9 lattice. The corresponding Gram-Schmidt matrices for D3Q15 and D3Q19 can be found in appendix A.6.

Both approaches provide consistent (but different) ways to obtain all moments m_k. The linear independence of vectors guarantees that $\boldsymbol{M}$ can be inverted, *i.e.* $\boldsymbol{M}^{-1}$ exists. Moreover, the corresponding vectors $\boldsymbol{M}_k$ are mutually orthogonal. Orthogonality guarantees that the hydrodynamic moments m_k are uniquely defined. It means they can be tuned independently of each other. For example, performing the collision for a moment corresponding to the momentum flux tensor does not change

density which is also the moment of $\boldsymbol{M}$. As a consequence, moments can be independently relaxed, which in turn provides the necessary flexibility to tune accuracy and stability.

Though vectors are orthogonal in both bases, there is a difference. For the Gram-Schmidt vectors (denoted by G), orthogonality means

$$\sum_{i=0}^{q-1} {}^{\mathrm{G}}M_{ki}\, {}^{\mathrm{G}}M_{li} = 0 \quad (k \neq l). \tag{10.11}$$

In the case of the Hermite polynomials (denoted by H), the vectors are orthogonal when weighted with the lattice weights w_i:

$$\sum_{i=0}^{q-1} w_i\, {}^{\mathrm{H}}M_{ki}\, {}^{\mathrm{H}}M_{li} = 0 \quad (k \neq l). \tag{10.12}$$

Having discussed the moment space, in the next section we will discuss step-by-step instructions of how to perform the MRT algorithm.

10.3 General MRT algorithm

Once we know the transformation matrix $\boldsymbol{M}$ and the relaxation matrix $\boldsymbol{S}$, which includes all the relaxation rates ω_i, we can perform the MRT algorithm as sketched in fig. 10.1. In the following we will explain the force-free algorithm. (The inclusion of forces is discussed in section 10.5.)

1. **Compute conserved moments:** The density ρ and momentum $\boldsymbol{j} = \rho\boldsymbol{u}$ can be obtained from the pre-collision populations as usual:

$$\rho = \sum_j f_j, \quad \rho\boldsymbol{u} = \sum_j f_j \boldsymbol{c}_j. \tag{10.13}$$

2. **Transform to moment space:** The moments m_k are obtained from the populations f_i *via*

$$\boldsymbol{m} = \boldsymbol{M}\boldsymbol{f}, \quad m_k = M_{ki} f_i. \tag{10.14}$$

3. **Compute equilibrium moments:** Knowing conserved moments ρ and $\rho\boldsymbol{u}$, one can construct equilibrium moments through a general polynomial representation:

$$m_k^{\mathrm{eq}} = \rho \sum_{n,m} a_k{}_{\alpha\beta\gamma}^{nml} u_\alpha^n u_\beta^m u^l \gamma. \tag{10.15}$$

   The discussion of how to obtain coefficients $a_k{}_{\alpha\beta\gamma}^{nml}$ is presented after the algorithm outline. Exact form of the equilibrium moments is given in section 10.4.1

for Hermite polynomials approach and in section 10.4.2 for the Gram-Schmidt approach.

4. **Collide:** Once the moments $\boldsymbol{m}$ and $\boldsymbol{m}^{\text{eq}}$ have been calculated, collision is performed in moment space in a BGK relaxation manner:

$$m_k^\star = m_k - \omega_k \left(m_k - m_k^{\text{eq}}\right). \tag{10.16}$$

Note that collision does not have to be performed for the conserved moments since they are not changed during collision, *i.e.* $\rho = \rho^{\text{eq}}$ and $\boldsymbol{j} = \boldsymbol{j}^{\text{eq}}$. An exception is the presence of mass or momentum sources (*e.g.* forces which will be discussed in section 10.5).

5. **Transform to population space:** After relaxation, the post-collision populations are obtained using the inverse transformation:

$$f_i^\star = M_{ik}^{-1} m_k^\star. \tag{10.17}$$

The inverse $\boldsymbol{M}^{-1}$ for D2Q9 is given in section 10.4.

6. **Stream:** Finally, streaming is performed in population space:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i, t+1) = f_i^\star(\boldsymbol{x}, t). \tag{10.18}$$

Let us provide a few comments for the algorithm above. For the equilibrium moment m_k^{eq} one needs to find the coefficients $a_k{}_{\alpha\beta\gamma}^{nml}$. Recall that the expresion for equilibrium populations f^{eq} from eq. (3.56) contains conserved moments ρ and $\rho\boldsymbol{u}$:

$$f_i^{\text{eq}} = w_i \rho \Big(1 + 3c_{i\alpha}u_\alpha + \frac{9}{2}\Big(c_{i\alpha}c_{i\beta} - \frac{1}{3}\Big)u_\alpha u_\beta\Big). \tag{10.19}$$

Using the expression above one computes the equilibrium moments in the same manner as moments:

$$\boldsymbol{m}^{\text{eq}} = \boldsymbol{M}\boldsymbol{f}^{\text{eq}}, \quad m_k^{\text{eq}} = \sum_j M_{ki} f_i^{\text{eq}}. \tag{10.20}$$

This "brute force" matrix-vector multiplication [20] provides the macroscopic equilibrium moments up to second order in velocity, *i.e.* the form of the equilibrium moments is $m_k^{\text{eq}} = \sum_{n+m\leq 2} a_k{}_{\alpha\beta}^{nm} \rho u_\alpha^n u_\beta^m$. Notice that the multiplication of matrix $\boldsymbol{M}$ on equilibrium populations f^{eq} can be done only once, so coefficients $a_k{}_{\alpha\beta}^{nm}$ are known. Then, density and momentum are substituted directly to eq. (10.15) to calculate the equilibrium moments.This is much more efficient than applying eq. (10.20) over and over again.

One limitation with the "brute force" method is that velocity terms in equilibrium moments are upto the second order because the equilibrium populations contain terms upto the second order in velocity. Sometimes it is desirable to include terms of third or even higher orders in velocity, for example to improve accuracy, stability or Galilean invariance [10]. In chapter 3 we presented the expression for Hermite polynomial moments (*cf.* eq. (3.59)). We also mentioned that only Hermite

polynomials upto the second order in velocity are required to satisfy conservation rules. However, one can calculate higher order Hermite polynomials equilibrium moments. Those equilibrium moments and their corresponding Hermite polynomials can be included into the MRT approach. We underline here that those equilibrium moments are known exactly due to the Gauss quadrature rule.

However, there is one caveat. Only a few higher-order discretised Hermite polynomials are distinct. This is connected to the degeneracy of the velocity set: $\boldsymbol{c}_i^3 = \boldsymbol{c}$. Thus, as we will see later in section 10.4.1 only equilibrium moments upto the fourth order in velocity are constructed.

In general, any MRT vector $\boldsymbol{M}_k$ (not necessarily obtained through Hermite polynomials or Gram-Schmidt procedures) can be represented as a linear combination of Hermite vectors: $\boldsymbol{M}_l = \sum_k b_{lk}\, {}^{\mathrm{H}}\boldsymbol{M}_k$. For each of the Hermite vectors, we know exactly the associated equilibrium moment from section 3.3.4 (see also eq. (3.69)). This allows constructing the equilibrium moments for an arbitrary matrix $\boldsymbol{M}$. The inclusion of higher velocity orders in moments can improve stability [19] and accuracy of simulations, though this effect might be not significant.

Let us tell a few words about a numerical implementation. Surprisingly, the MRT model can be implemented with only 15–20% computational overhead compared to the BGK implementation [21]. For example, the calculation of the equilibrium moments for MRT has the same overhead as the calculation of the equilibrium functions for BGK. This is because of the density and the momentum being calculated through populations. Those conserved moments are then used to calculate the equilibrium moments which are functions of density and momentum only. Also, the components of $\boldsymbol{M}$ and $\boldsymbol{M}^{-1}$ are known and not changing in time, so the calculation of moments $\boldsymbol{M}\boldsymbol{f}$ and reverting collided moments $\boldsymbol{M}^{-1}\boldsymbol{m}$ back to populations can be done efficiently through vector-matrix multiplication. Streaming is the same operation for the BGK and the MRT collision operators.

10.4 MRT for the D2Q9 lattice

We will now present the MRT model for D2Q9 in detail. First we will discuss the Hermite polynomial procedure (section 10.4.1) that allows us to find the exact form of equilibrium moments beyond second order in $\boldsymbol{u}$. In the second part, we will investigate the Gram-Schmidt procedure (section 10.4.2). Finally we will compare both methods in section 10.4.3.

10.4.1 Hermite polynomials

In the Hermite picture (denoted by H), the moment vectors ${}^{\mathrm{H}}\boldsymbol{M}_i$ correspond to Hermite polynomials of the LBM velocities $\boldsymbol{c}_i$. For D2Q9, we can write

$$\begin{aligned}
{}^{\mathrm{H}}M_i^{(\rho)} &= H^{(0)} = 1,\\
{}^{\mathrm{H}}M_i^{(j_x)} &= H_x^{(1)} = c_{ix},\\
{}^{\mathrm{H}}M_i^{(j_y)} &= H_y^{(1)} = c_{iy},\\
{}^{\mathrm{H}}M_i^{(p_{xx})} &= H_{xx}^{(2)} = c_{ix}c_{ix} - c_{\mathrm{s}}^2,\\
{}^{\mathrm{H}}M_i^{(p_{yy})} &= H_{yy}^{(2)} = c_{iy}c_{iy} - c_{\mathrm{s}}^2,\\
{}^{\mathrm{H}}M_i^{(p_{xy})} &= H_{xy}^{(2)} = c_{ix}c_{iy},\\
{}^{\mathrm{H}}M_i^{(\gamma_x)} &= H_{xyy}^{(3)} = c_{ix}c_{iy}^2 - c_{\mathrm{s}}^2 c_{ix},\\
{}^{\mathrm{H}}M_i^{(\gamma_y)} &= H_{yxx}^{(3)} = c_{iy}c_{ix}^2 - c_{\mathrm{s}}^2 c_{iy},\\
{}^{\mathrm{H}}M_i^{(\gamma)} &= H_{xxyy}^{(4)} = c_{ix}^2c_{iy}^2 - c_{\mathrm{s}}^2c_{ix}^2 - c_{\mathrm{s}}^2c_{iy}^2 + c_{\mathrm{s}}^4.
\end{aligned} \tag{10.21}$$

We have introduced several superscripts denoting the nine moments ρ, j_x, j_y, p_{xx}, p_{xy}, p_{yy}, γ_x, γ_y and γ. While the first six are the components of the macroscopic density, momentum vector and stress (or pressure) tensor, the additional moments γ_x, γ_y and γ are higher-order moments that do not affect the Navier-Stokes level hydrodynamics. They are sometimes called "non-hydrodynamic" or "ghost" moments [22, 23]. The nine vectors ${}^{\mathrm{H}}\boldsymbol{M}_i$ in eq. (10.21) are orthogonal according to the condition specified in eq. (10.12).

Substituting the numerical values of the D2Q9 velocity set, including $c_{\mathrm{s}}^2 = \frac{1}{3}$, eq. (10.21) can be written in form of the transformation matrix ${}^{\mathrm{H}}\boldsymbol{M}$:

$${}^{\mathrm{H}}\boldsymbol{M} = \begin{pmatrix}
{}^{\mathrm{H}}M_i^{(\rho)}\\ {}^{\mathrm{H}}M_i^{(j_x)}\\ {}^{\mathrm{H}}M_i^{(j_y)}\\ {}^{\mathrm{H}}M_i^{(p_{xx})}\\ {}^{\mathrm{H}}M_i^{(p_{yy})}\\ {}^{\mathrm{H}}M_i^{(p_{xy})}\\ {}^{\mathrm{H}}M_i^{(\gamma_x)}\\ {}^{\mathrm{H}}M_i^{(\gamma_y)}\\ {}^{\mathrm{H}}M_i^{(\gamma)}
\end{pmatrix} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1\\
0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1\\
-\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3}\\
-\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3}\\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1\\
0 & -\frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & \frac{2}{3}\\
0 & 0 & -\frac{1}{3} & 0 & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} & -\frac{2}{3} & -\frac{2}{3}\\
\frac{1}{9} & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & \frac{4}{9} & \frac{4}{9} & \frac{4}{9} & \frac{4}{9}
\end{pmatrix}. \tag{10.22}$$

The equilibrium moments are simple functions of the density ρ and velocity $\boldsymbol{u}$. Using eq. (3.45) the equilibrium moments are calculated exactly:

$$\begin{aligned}
\rho^{\mathrm{eq}} &= \rho, & j_x^{\mathrm{eq}} &= \rho u_x, & j_y^{\mathrm{eq}} &= \rho u_y,\\
p_{xx}^{\mathrm{eq}} &= \rho u_x^2, & p_{yy}^{\mathrm{eq}} &= \rho u_y^2, & p_{xy}^{\mathrm{eq}} &= \rho u_x u_y,\\
\gamma_x^{\mathrm{eq}} &= \rho u_x u_y^2, & \gamma_y^{\mathrm{eq}} &= \rho u_x^2 u_y, & \gamma^{\mathrm{eq}} &= \rho u_x^2 u_y^2.
\end{aligned} \tag{10.23}$$

Therefore, there is no need to compute $\boldsymbol{m}^{\mathrm{eq}}$ from $\boldsymbol{M}\boldsymbol{f}^{\mathrm{eq}}$.

The matrix ${}^{\mathrm{H}}\boldsymbol{M}^{-1}$ can be easily obtained by reverting ${}^{\mathrm{H}}\boldsymbol{M}$ in any computational package as Mathematica, Matlab, MathCAD, etc. However, because of the orthogonality condition eq. (10.12) one can represent each column of $\boldsymbol{M}^{-1}$ through the corresponding row vector of $\boldsymbol{M}$. We leave it as an exercise for an interested reader. Finally, the inverse transformation matrix for Hermite polynomials approach is the following:

$${}^{\mathrm{H}}\boldsymbol{M}^{-1} = \begin{pmatrix}
\frac{4}{9} & 0 & 0 & -\frac{2}{3} & -\frac{2}{3} & 0 & 0 & 1\\
\frac{1}{9} & \frac{1}{3} & 0 & \frac{1}{3} & -\frac{1}{6} & 0 & -\frac{1}{2} & 0 & -\frac{1}{2}\\
\frac{1}{9} & 0 & \frac{1}{3} & -\frac{1}{6} & \frac{1}{3} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2}\\
\frac{1}{9} & -\frac{1}{3} & 0 & \frac{1}{3} & -\frac{1}{6} & 0 & \frac{1}{2} & 0 & -\frac{1}{2}\\
\frac{1}{9} & 0 & -\frac{1}{3} & -\frac{1}{6} & \frac{1}{3} & 0 & 0 & \frac{1}{2} & -\frac{1}{2}\\
\frac{1}{36} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\
\frac{1}{36} & -\frac{1}{12} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\\
\frac{1}{36} & -\frac{1}{12} & -\frac{1}{12} & \frac{1}{12} & \frac{1}{12} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4}\\
\frac{1}{36} & \frac{1}{12} & -\frac{1}{12} & \frac{1}{12} & \frac{1}{12} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4}
\end{pmatrix}, \tag{10.24}$$

The last missing piece of the puzzle is the relaxation matrix ${}^{\mathrm{H}}\boldsymbol{S}$. It is almost diagonal. One can show through CE procedure that in order to recover the Navier-Stokes equation ${}^{\mathrm{H}}\boldsymbol{S}$ has the following form for D2Q9 [24]:

$${}^{\mathrm{H}}\boldsymbol{S} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \frac{\omega_\zeta+\omega_\nu}{2} & \frac{\omega_\zeta-\omega_\nu}{2} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \frac{\omega_\zeta-\omega_\nu}{2} & \frac{\omega_\zeta+\omega_\nu}{2} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & \omega_\nu & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \omega_{\gamma_\alpha} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \omega_{\gamma_\alpha} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \omega_\gamma
\end{pmatrix}. \tag{10.25}$$

TODO (EMV): Please correct the stuff here if I missed something. The relaxation rates for conserved moments (density and momentum) are arbitrary; they are chosen as zero here. The rates ω_ν and ω_ζ are connected to the kinematic and bulk viscosities, respectively [24]:

$$\nu = \frac{1}{3}\left(\frac{1}{\omega_\nu} - \frac{1}{2}\right), \quad \zeta = \frac{2}{9}\left(\frac{1}{\omega_\zeta} - \frac{1}{2}\right). \tag{10.26}$$

The macroscopic momentum equation recovered by the MRT model is

$$\partial_t(\rho u_\alpha)+\partial_\beta(\rho u_\alpha u_\beta) = -c_s^2\partial_\alpha\rho+\partial_\beta\Big(\nu\big(\partial_\beta u_\alpha+\partial_\alpha u_\beta\big)+\big(\zeta-\frac{2}{3}\nu\big)\partial_\gamma u_\gamma\delta_{\alpha\beta}\Big). \quad (10.27)$$

Unlike in the BGK model, kinematic and bulk viscosity can be chosen independently. This is a clear advantage of MRT over BGK. In particular when the Reynolds number is large (and the kinematic viscosity small), increasing the bulk viscosity might stabilise simulations. The mechanism behind that is spurious pressure waves are damped more quickly as will be covered in Section 12.1, where the effect of bulk and shear viscosity on pressure waves is discussed.

For the higher order moments γ_x and γ_y, the same relaxation rate ω_{γ_α} is taken to satisfy isotropy. Therefore, there are four independent relaxation rates. Only two of them are related to physical quantities. The relaxation rates ω_{γ_α} and ω_γ do not appear in the leading-order terms, but they can be tuned to increase accuracy and stability [10, 22, 19].

Concluding, we have expressions for the matrices ${}^{\mathrm{H}}\boldsymbol{M}$, ${}^{\mathrm{H}}\boldsymbol{M}^{-1}$ and ${}^{\mathrm{H}}\boldsymbol{S}$ for Hermite polynomials approach. Those can be used in the implementation details outlined in section 10.3.

10.4.2 Gram-Schmidt procedure

The Gram-Schmidt approach is more widely used in the LB literature than the method based on the Hermite polynomials. The underlying idea of the Gram-Schmidt approach is to construct a set of orthogonal vectors to form the transformation matrix $\boldsymbol{M}$. Note that we seek these vectors to be orthogonal without any corresponding weights as for the Hermite approach as in eq. (10.12).

We start with a set of q linearly independent vectors $\{\boldsymbol{v}_n\}$ with $n = 0, \ldots, q-1$. These vectors are generally not mutually orthogonal. The Gram-Schmidt procedure maps these vectors to a set of q other vectors $\{\boldsymbol{u}_n\}$ which are all mutually orthogonal. The construction process is represented through expressions:

$$\begin{aligned}
\boldsymbol{u}_0 &= \boldsymbol{v}_0,\\
\boldsymbol{u}_1 &= \boldsymbol{v}_1 - \boldsymbol{u}_0\frac{\boldsymbol{u}_0\cdot\boldsymbol{v}_1}{\boldsymbol{u}_0\cdot\boldsymbol{u}_0},\\
\boldsymbol{u}_2 &= \boldsymbol{v}_2 - \boldsymbol{u}_0\frac{\boldsymbol{u}_0\cdot\boldsymbol{v}_2}{\boldsymbol{u}_0\cdot\boldsymbol{u}_0} - \boldsymbol{u}_1\frac{\boldsymbol{u}_1\cdot\boldsymbol{v}_2}{\boldsymbol{u}_1\cdot\boldsymbol{u}_1},\\
&\vdots\\
\boldsymbol{u}_{q-1} &= \boldsymbol{v}_{q-1} - \sum_{i=0}^{q-2}\boldsymbol{u}_i\frac{\boldsymbol{u}_i\cdot\boldsymbol{v}_{q-1}}{\boldsymbol{u}_i\cdot\boldsymbol{u}_i}.
\end{aligned} \quad (10.28)$$

This algorithm essentially projects each vector $\boldsymbol{v}_n$ onto a direction which is orthogonal to all previous directions.

Exercise 10.1. Show that all vectors $\{\boldsymbol{u}_n\}$ are mutually orthogonal.

We seek the initial vector set $\{\boldsymbol{v}_n\}$ in the form $c_{ix}^n c_{iy}^m$. Notice that the pair of coefficients $(n, m) = (0, 0)$ is the vector (1, 1, 1, 1, 1, 1, 1, 1, 1) related to density ρ. Pairs $(n, m) = (0, 1)$ and $(n, m) = (1, 0)$ give us vectors (0, 1, 0, −1, 0, 1, −1, −1, 1) and (0, 0, 1, 0, −1, 1, 1, −1, −1) related to x- and y-components of momentum $\boldsymbol{j}$. Those vectors are already orthogonal in the sense of eq. (10.11).

The next step is to find the remaining six vectors for D2Q9. We've already shown in Section 4.2.1 that there are three non-zero distinct polynomials of second order according to $(n, m) = (1, 1),\ (2, 0),\ (0, 2)$. We also consider the third-order polynomials according to $(n, m) = (2, 1),\ (1, 2)$ and the fourth-order polynomial $(n, m) = (2, 2)$. This gives nine linearly independent vectors in total.

Now we can perform the Gram-Schmidt procedure to reduce these initial vectors to a set of nine mutually orthogonal vectors. We will only show the result of the procedure. Taking the notation ${}^{\mathrm{G}}\boldsymbol{M}$ rather than $\boldsymbol{u}$ for the final Gram-Schmidt vectors, the set of orthogonal vectors is as follows [9]:

$$
\begin{aligned}
{}^{\mathrm{G}}M_i^{(\rho)} &= 1, \\
{}^{\mathrm{G}}M_i^{(e)} &= -4 + 3\left(c_{ix}^2 + c_{iy}^2\right), \\
{}^{\mathrm{G}}M_i^{(\epsilon)} &= 4 - \tfrac{21}{2}\left(c_{ix}^2 + c_{iy}^2\right) + \tfrac{9}{2}\left(c_{ix}^2 + c_{iy}^2\right)^2, \\
{}^{\mathrm{G}}M_i^{(j_x)} &= c_{ix}, \\
{}^{\mathrm{G}}M_i^{(q_x)} &= \left(-5 + 3\left(c_{ix}^2 + c_{iy}^2\right)\right) c_{ix}, \\
{}^{\mathrm{G}}M_i^{(j_y)} &= c_{iy}, \\
{}^{\mathrm{G}}M_i^{(q_y)} &= \left(-5 + 3\left(c_{ix}^2 + c_{iy}^2\right)\right) c_{iy}, \\
{}^{\mathrm{G}}M_i^{(p_{xx})} &= c_{ix}^2 - c_{iy}^2, \\
{}^{\mathrm{G}}M_i^{(p_{xy})} &= c_{ix} c_{iy}.
\end{aligned}
\tag{10.29}
$$

The moments ${}^{\mathrm{G}}\boldsymbol{Mf}$ correspond to the physical quantities (sometimes precisely, sometimes roughly), *i.e.* ρ is the density, j_x and j_y are components of momentum flux, q_x and q_y correspond to the energy flux components, e correspond to the energy and ϵ correspond to the energy squared, p_{xx} and p_{xy} correspond to the diagonal and off-diagonal components of the stress tensor. Replacing the velocities $c_{i\alpha}$ by their numerical values, we can write the Gram-Schmidt transformation matrix:

$$
{}^{\mathrm{G}}\boldsymbol{M} = \begin{pmatrix} {}^{\mathrm{G}}M_i^{(\rho)} \\ {}^{\mathrm{G}}M_i^{(e)} \\ {}^{\mathrm{G}}M_i^{(\epsilon)} \\ {}^{\mathrm{G}}M_i^{(j_x)} \\ {}^{\mathrm{G}}M_i^{(q_x)} \\ {}^{\mathrm{G}}M_i^{(j_y)} \\ {}^{\mathrm{G}}M_i^{(q_y)} \\ {}^{\mathrm{G}}M_i^{(p_{xx})} \\ {}^{\mathrm{G}}M_i^{(p_{xy})} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -4 & -1 & -1 & -1 & -1 & 2 & 2 & 2 & 2 \\ 4 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & -2 & 0 & 2 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \\ 0 & 0 & -2 & 0 & 2 & 1 & 1 & -1 & -1 \\ 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \end{pmatrix}. \tag{10.30}
$$

The next step is to find out the equilibrium moments $\boldsymbol{m}^{\mathrm{eq}}$ in the Gram-Schmidt basis. One way is the rather complicated linear wave number analysis as presented in [9]. Another route followed here is to represent the Gram-Schmidt vectors by Hermite vectors whose equilibrium moments are already known. In fact, the Gram-Schmidt vectors can be written as linear combinations of the Hermite vectors:

$$
\begin{aligned}
{}^{\mathrm{G}}M_i^{(\rho)} &= {}^{\mathrm{H}}M_i^{(\rho)}, \\
{}^{\mathrm{G}}M_i^{(e)} &= -2 + 3\left({}^{\mathrm{H}}M_i^{(p_{xx})} + {}^{\mathrm{H}}M_i^{(p_{yy})}\right), \\
{}^{\mathrm{G}}M_i^{(\epsilon)} &= 9\,{}^{\mathrm{H}}M_i^{(\gamma)} - 3\left({}^{\mathrm{H}}M_i^{(p_{xx})} + {}^{\mathrm{H}}M_i^{(p_{yy})}\right) + {}^{\mathrm{H}}M_i^{(\rho)}, \\
{}^{\mathrm{G}}M_i^{(j_x)} &= {}^{\mathrm{H}}M_i^{(j_x)}, \\
{}^{\mathrm{G}}M_i^{(q_x)} &= 3\,{}^{\mathrm{H}}M_i^{(\gamma_x)} - {}^{\mathrm{H}}M_i^{(j_x)}, \\
{}^{\mathrm{G}}M_i^{(j_y)} &= {}^{\mathrm{H}}M_i^{(j_y)}, \\
{}^{\mathrm{G}}M_i^{(q_y)} &= 3\,{}^{\mathrm{H}}M_i^{(\gamma_y)} - {}^{\mathrm{H}}M_i^{(j_y)}, \\
{}^{\mathrm{G}}M_i^{(p_{xx})} &= {}^{\mathrm{H}}M_i^{(p_{xx})} - {}^{\mathrm{H}}M_i^{(p_{yy})}, \\
{}^{\mathrm{G}}M_i^{(p_{xy})} &= {}^{\mathrm{H}}M_i^{(p_{xy})}.
\end{aligned} \tag{10.31}
$$

Thus, one can work out the corresponding Gram-Schmidt equilibrium moments as function of density and velocity:

$$
\begin{aligned}
&\rho^{\mathrm{eq}} = \rho, && e^{\mathrm{eq}} = \rho - 3\rho\left(u_x^2 + u_y^2\right), && \epsilon^{\mathrm{eq}} = 9\rho u_x^2 u_y^2 - 3\rho\left(u_x^2 + u_y^2\right) + \rho, \\
&j_x^{\mathrm{eq}} = \rho u_x, && q_x^{\mathrm{eq}} = 3\rho u_x^3 - \rho u_x, && j_y^{\mathrm{eq}} = \rho u_y, \\
&q_y^{\mathrm{eq}} = 3\rho u_y^3 - \rho u_y, && p_{xx}^{\mathrm{eq}} = \rho\left(u_x^2 - u_y^2\right), && p_{xy}^{\mathrm{eq}} = \rho u_x u_y.
\end{aligned} \tag{10.32}
$$

The inverse matrix ${}^{\mathrm{G}}\boldsymbol{M}^{-1}$ can be obtained by the help of any mathematical package:

$$
{}^{\mathrm{G}}\boldsymbol{M}^{-1} = \begin{pmatrix}
\frac{1}{9} & -\frac{1}{9} & \frac{1}{9} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{9} & -\frac{1}{36} & -\frac{1}{18} & \frac{1}{6} & -\frac{1}{6} & 0 & 0 & \frac{1}{4} & 0 \\
\frac{1}{9} & -\frac{1}{36} & -\frac{1}{18} & 0 & 0 & \frac{1}{6} & -\frac{1}{6} & -\frac{1}{4} & 0 \\
\frac{1}{9} & -\frac{1}{36} & -\frac{1}{18} & -\frac{1}{6} & \frac{1}{6} & 0 & 0 & \frac{1}{4} & 0 \\
\frac{1}{9} & -\frac{1}{36} & -\frac{1}{18} & 0 & 0 & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{4} & 0 \\
\frac{1}{9} & \frac{1}{18} & \frac{1}{36} & \frac{1}{6} & \frac{1}{12} & \frac{1}{6} & \frac{1}{12} & 0 & \frac{1}{4} \\
\frac{1}{9} & \frac{1}{18} & \frac{1}{36} & -\frac{1}{6} & -\frac{1}{12} & \frac{1}{6} & \frac{1}{12} & 0 & -\frac{1}{4} \\
\frac{1}{9} & \frac{1}{18} & \frac{1}{36} & -\frac{1}{6} & -\frac{1}{12} & -\frac{1}{6} & -\frac{1}{12} & 0 & \frac{1}{4} \\
\frac{1}{9} & \frac{1}{18} & \frac{1}{36} & \frac{1}{6} & \frac{1}{12} & -\frac{1}{6} & -\frac{1}{12} & 0 & -\frac{1}{4}
\end{pmatrix}. \tag{10.33}
$$

The relaxation matrix in the Gram-Schmidt basis assumes the diagonal form

$$
{}^{\mathrm{G}}\boldsymbol{S} = \mathrm{diag}\left(0, \omega_e, \omega_\epsilon, 0, \omega_q, 0, \omega_q, \omega_\nu, \omega_\nu\right). \tag{10.34}
$$

Zero relaxation rates above are for the conserved moments (density and momentum), ω_e and ω_ν are connected with bulk and dynamic viscosities, ω_ϵ and ω_q are free parameters to tune. With those relaxation rates, the restored Navier-Stokes equation is

$$
\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = -c_\mathrm{s}^2\partial_\alpha\rho + \partial_\beta\Big(\nu(\partial_\beta u_\alpha + \partial_\alpha u_\beta) + \Big(\zeta - \frac{2}{3}\eta\Big)\partial_\gamma u_\gamma\Big), \tag{10.35}
$$

where $\nu = \frac{1}{3}\left(\frac{1}{\omega_\nu} - \frac{1}{2}\right)$ and $\zeta = \frac{2}{9}\left(\frac{1}{\omega_e} - \frac{1}{2}\right)$. TODO (EMV): Check the consistency between equations.

10.4.3 Similarities and differences

The Hermite and Gram-Schmidt approaches are similar, but they also show a few differences:

MRT vectors: Each lattice velocity set has spurious invariants which are not found for their continuous counterparts. For example, we mentioned in Section 4.2.1 that $c_{i\alpha}^3 = c_{i\alpha}$ for D2Q9 (and for common 3D lattices). As a result of this invariant, there is only a certain number of independent Hermite polynomials in discretised velocity that can be obtained. This number does not necessarily coincide with the number of velocity directions [25]. For example, for D2Q9 and D3Q27 the number of Hermite polynomials equals the number of velocities, but for D3Q15 or D3Q19 this is not the case. Missing vectors have to be constructed, and this is usually done with the Gram-Schmidt procedure.This is one of the reasons why the Gram-Schmidt approach is more widely used than the Hermite approach.

Equilibrium moments: Only the Hermite approach provides the set of equilibrium moments in a relatively easy way and also beyond the second-order Navier-Stokes level. Thus, any arbitrary set of MRT vectors can be represented as linear

combinations of Hermite vectors to obtain the equilibrium moments. This approach is significantly easier than obtaining the moments from a linear wavenumber analysis [9].

Another difference is that the equilibrium moments for the Hermite polynomials are in increasing order in the macroscopic velocity: ρ, $\rho\boldsymbol{u}$, $\rho\boldsymbol{u}^2$, $\rho\boldsymbol{u}^3$, $\rho\boldsymbol{u}^4$. In the Gram-Schmidt approach, however, polynomials of different order are mixed, as can be seen in eq. (10.32).

As well for the Hermite polynomials approach, one can see that the equilibrium moments contain third and fourth order polynomials in $\boldsymbol{u}$. Thus, equilibrium moments for any other MRT approach can contain higher-order terms as one can represent its matrix $\boldsymbol{M}$ through ${}^{\mathrm{H}}\boldsymbol{M}$. We should not forget that only polynomials up to the second order in $\boldsymbol{u}$ are required to restore the Navier-Stokes equation. Thus, one common approach is to neglect all moments of order $\boldsymbol{u}^3$ and $\boldsymbol{u}^4$ [9, 26]. Although there are some indications that keeping higher orders can improve stability [19], these are only minor improvements. If higher-order moments are neglected, the equilibrium moments can also be calculated from $\boldsymbol{m}^{\mathrm{eq}} = \boldsymbol{M}\boldsymbol{f}^{\mathrm{eq}}$ with f_i^{eq} taken from the quadratic equilibrium in eq. (3.56).

Relaxation matrix: We have seen that the Gram-Schmidt relaxation matrix ${}^{\mathrm{G}}\boldsymbol{S}$ in eq. (10.34) is diagonal, while the Hermite relaxation matrix ${}^{\mathrm{H}}\boldsymbol{S}$ in eq. (10.25) is nearly diagonal. This is an additional reason why the Gram-Schmidt approach is normally preferred.

We will discuss the choice of the relaxation rates for both relaxation matrices ${}^{\mathrm{H}}\boldsymbol{S}$ and ${}^{\mathrm{G}}\boldsymbol{S}$ and provide more practical advice in section 10.7 to improve stability and accuracy. However, one advantage is already clearly seen: the MRT collision operator allows tuning shear and bulk viscosities independently. The increase of the bulk viscosity often allows improving stability by damping underresolved hydrodynamic artifacts [27, 26, 21].

10.5 Inclusion of forces

So far we have assumed that the momentum is conserved during collision. This is no longer the case when external forces are present. We will now give a brief overview of how to include forces in the MRT collision model.

Let us start with a revision of forcing in the BGK model:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i, t+1) = f_i(\boldsymbol{x}, t) - \frac{1}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)\right) + \Delta f_i(\boldsymbol{x}, t). \tag{10.36}$$

As thoroughly discussed in chapter 6, the effect of an external force density $\boldsymbol{F}^{\mathrm{ext}}$ can be included in the equilibrium distributions f_i^{eq} and/or the additional term Δf_i. This depends on the chosen forcing model. For MRT, the following implementations are common:

Simple forcing

The simplest approach is to approximate Δf_i by

$$\Delta f_i^{\text{simple}} = 3 w_i \boldsymbol{F}^{\text{ext}} \cdot \boldsymbol{c}_i. \tag{10.37}$$

One can easily add this force contribution to the right-hand-side of the MRT equation:

$$f_i^{\star}(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - M_{ij}^{-1} S_{jk} \left(m_k - m_k^{\text{eq}} \right) + \Delta f_i^{\text{simple}}. \tag{10.38}$$

Note that the equilibrium velocity used to calculate the equilibrium moments m_k^{eq} is not shifted, *i.e.* $\rho \boldsymbol{u}^{\text{eq}} = \sum_i f_i \boldsymbol{c}_i$. However, the Navier-Stokes equation is reproduced *via* the Chapman-Enskog analysis with the shifted macroscopic physical velocity according to $\rho \boldsymbol{u}^{\text{p}} = \sum_i f_i \boldsymbol{c}_i + \frac{1}{2} \boldsymbol{F}^{\text{ext}}$ (*cf.* chapter 6).

Guo forcing

Guo's model [28] for the BGK collision operator leads to

$$\Delta f_i^{\text{Guo}} = \left(1 - \frac{1}{2\tau} \right) w_i \left(\frac{\boldsymbol{c}_i - \boldsymbol{u}^{\text{p}}}{c_{\text{s}}^2} + \frac{\left(\boldsymbol{c}_i \cdot \boldsymbol{u}^{\text{p}} \right) \boldsymbol{c}_i}{c_{\text{s}}^4} \right) \cdot \boldsymbol{F}^{\text{ext}} = \left(1 - \frac{1}{2\tau} \right) F_i, \tag{10.39}$$

where we have introduced the abbreviation F_i. Again, the physical velocity is given by $\rho \boldsymbol{u}^{\text{p}} = \sum_i f_i \boldsymbol{c}_i + \frac{1}{2} \boldsymbol{F}^{\text{ext}}$. Furthermore, the equilibrium velocity is also shifted in the same way: $\rho \boldsymbol{u}^{\text{eq}} = \sum_i f_i \boldsymbol{c}_i + \frac{1}{2} \boldsymbol{F}^{\text{ext}}$. In the MRT model, one can use a similar form for the force contribution [2, 29]. The relaxed moments then become

$$m_k^{\star} = m_k - \omega_k \left(m_k - m_k^{\text{eq}} \right) + \left(1 - \frac{\omega_k}{2} \right) M_{ki} F_i. \tag{10.40}$$

The forcing term F_i is first transformed to momentum space, and then its contribution to the corresponding moment is multiplied by $1 - \omega_k/2$. After relaxation, the moments are transformed back to population space *via* $f_i^{\star} = M_{ik}^{-1} m_k^{\star}$, followed by the normal streaming. For D2Q9, the result of the matrix-vector multiplication $M_{ki} F_i$ in eq. (10.40) is given by [30, 2]:

$${}^{\text{H}}M_{ki} F_i = \begin{pmatrix} 0 \\ F_x^{\text{ext}} \\ F_y^{\text{ext}} \\ 2 F_x^{\text{ext}} u_x^{\text{p}} \\ 2 F_y^{\text{ext}} u_y^{\text{p}} \\ F_x^{\text{ext}} u_y^{\text{p}} + F_y^{\text{ext}} u_x^{\text{p}} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad {}^{\text{G}}M_{ki} F_i = \begin{pmatrix} 0 \\ 6 \left(F_x^{\text{ext}} u_x^{\text{p}} + F_y^{\text{ext}} u_y^{\text{p}} \right) \\ -6 \left(F_x^{\text{ext}} u_x^{\text{p}} + F_y^{\text{ext}} u_y^{\text{p}} \right) \\ F_x^{\text{ext}} \\ -F_x^{\text{ext}} \\ F_y^{\text{ext}} \\ -F_y^{\text{ext}} \\ 2 F_x^{\text{ext}} u_x^{\text{p}} - 2 F_y^{\text{ext}} u_y^{\text{p}} \\ F_y^{\text{ext}} u_x^{\text{p}} + F_x^{\text{ext}} u_y^{\text{p}} \end{pmatrix}. \tag{10.41}$$

Other forcing schemes

An implicit formulation of the force inclusion is given in [31, 32]. This approach is similar to Guo's. Another MRT force implementation can be found in [33] where only the momentum is shifted without a need to introduce the force populations δf_i.

10.6 TRT collision operator

The two relaxation time (TRT) model (section 10.6.1) combines the algorithmic simplicity of the BGK operator and improved accuracy and stability properties of the more general MRT model. We discuss the TRT implementation in section 10.6.2 and provide an example that shows its superiority over BGK in section 10.6.3.

10.6.1 Introduction

The MRT model provides a great deal of flexibility in tuning the relaxation of individual moments. However, the number of free MRT parameters may be too large and confusing from the perspective of an average LBM user. For example, for MRT D2Q9 there are two free relaxation rates to tune, for MRT D3Q15 and D3Q19 presented in appendix A.6 there are even more free parameters to tune. As far as those relaxation rates do not correspond to physical quantities on the isothermal Navier-Stokes level, it is not obvious how to tune those rates to achieve better accuracy or stability.

Moreover, so far MRT optimal parameters can be obtained only numerically by performing parameters study. One of such examples is the three-dimensional shear wave propagation [21], where the parameters are $\omega_e = 1.19$, $\omega_\epsilon = \omega_\pi = 1.4$, $\omega_q = 1.2$ and $\omega_m = 1.98$ to attain optimal stability for D3Q19 model (*cf.* appendix A.6). In general situations it is almost impossible or at least impractical to perform a parameters study or a linear stability analysis to optimise accuracy and stability.

Fortunately, there exists a collision model that combines the accuracy and stability advantages of the MRT model and the calculation simplicity of the BGK model: the two relaxation times (TRT) model [7, 34, 1] requires only two relaxation rates. The first relaxation rate is related to the shear viscosity, the other is a free parameter. Having only two rates significantly simplifies the mathematical analysis of accuracy and stability.

The **TRT model** is a simplification of the MRT model. All moments related to even-order polynomials in velocity (*i.e.* ρ, p_{xx}, p_{yy}, p_{xy} etc) are relaxed with a rate ω^+, all other moments (*e.g.* j_x, j_y, γ_x, γ_y, etc) with another rate ω^-.

Two relaxation rates can solve the BGK collision operator 'disease', which is the dependency of accuracy errors on the relaxation rate ω. Previously in section 4.5 we presented the complicated form of the accuracy error on $\tau = \frac{1}{\omega}$. The spatial accuracy error is proportional to $(\tau - 1/2)^2$; time accuracy error has terms containing τ. Because the viscosity depends on ω (τ), the error is said to be dependent on viscosity. Especially, high viscosities ($\omega \to 0$) drastically deteriorate accuracy. Overall, if one performs simulations across different τ while all other parameters are fixed (grid number, macroscopic velocity, etc), then it is difficult to guarantee same accuracy errors across simulations. Usually, to guarantee consistent errors across simulations one needs to simultaneously tune the relaxation rate ω, grid number N and to a smaller extent the macroscopic velocity $\boldsymbol{u}$.

In contrast, for the TRT collision operator it can be shown that the accuracy error depends on a certain combination of the two relaxation rates ω^+ and ω^- [7].

The so-called **magic parameter** Λ characterises the truncation error and stability properties of the TRT model. It is a function of the TRT relaxation rates ω^+ and ω^-:

$$\Lambda = \left(\frac{1}{\omega^+} - \frac{1}{2}\right)\left(\frac{1}{\omega^-} - \frac{1}{2}\right). \tag{10.42}$$

With the TRT collision operator control of accuracy is drastically simplified. If one needs to guarantee the same accuracy errors for simulations where the viscosity changes, then there is no need to tune number of parameters as the grid number and the macroscopic velocity. The only thing required is to fix Λ to be a constant across simulations with the help of the free parameter ω^- while changing the viscosity which is represented through another relaxation rate ω^+.

Therefore, unlike in the BGK model, one can modify the viscosity by changing ω^+, but simultaneously keep higher-order errors under control by fixing the numerical values of Λ [14]. We will get back to specific numerical values of Λ in section 10.7.

One can show [7, 11, 34] that the higher-order truncation errors of the steady-state TRT model scale with $\Lambda - \frac{1}{12}$ (third-order error) and $\Lambda - \frac{1}{6}$ (fourth-order error). Thus, there exist preferable values of Λ that can provide better accuracy and stability (*cf.* section 10.7).

10.6.2 Implementation

We will now provide the algorithmic details and show that the TRT model is as computationally efficient as the BGK model, while offering more flexibility in terms of accuracy and stability.

Naive implementation: As we have already mentioned, one can use the MRT model with ω^+ for all even-order moments and ω^- for all odd-order moments. This implementation as computationally "heavy" as the original MRT model, but it is a good starting point to optimise the relaxation rates of the MRT model.

Standard implementation: The TRT model is based on the decomposition of populations into symmetric and antisymmetric parts. Any LB velocity set is always symmetric, *i.e.* for any given velocity $\boldsymbol{c}_i$ there is always a velocity $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$. This is also true for $\boldsymbol{c}_0 = \boldsymbol{0}$. Using these notations one can decompose the populations and equilibrium populations into their symmetric and antisymmetric parts:

$$\begin{aligned} f_i^+ &= \frac{f_i + f_{\bar{i}}}{2}, & f_i^- &= \frac{f_i - f_{\bar{i}}}{2}, \\ f_i^{\mathrm{eq}+} &= \frac{f_i^{\mathrm{eq}} + f_{\bar{i}}^{\mathrm{eq}}}{2}, & f_i^{\mathrm{eq}-} &= \frac{f_i^{\mathrm{eq}} - f_{\bar{i}}^{\mathrm{eq}}}{2} \end{aligned} \tag{10.43}$$

with equilibrium distributions from eq. (3.56). For the rest population we have $f_0^+ = f_0$ and $f_0^- = 0$ and accordingly for the equilibrium value f_0^{eq}. From eq. (10.43) we can now decompose the populations in terms of their symmetric and antisymmetric parts:

$$\begin{aligned} f_i &= f_i^+ + f_i^-, & f_{\bar{i}} &= f_i^+ - f_i^-, \\ f_i^{\mathrm{eq}} &= f_i^{\mathrm{eq}+} + f_i^{\mathrm{eq}-}, & f_{\bar{i}}^{\mathrm{eq}} &= f_i^{\mathrm{eq}+} - f_i^{\mathrm{eq}-}. \end{aligned} \tag{10.44}$$

The **standard TRT model** can be implemented similarly to the BGK model:

$$\begin{aligned} f_i^\star(\boldsymbol{x}, t) &= f_i - \omega^+ \left(f_i^+ - f_i^{\mathrm{eq}+} \right) - \omega^- \left(f_i^- - f_i^{\mathrm{eq}-} \right), \\ f_i(\boldsymbol{x} + \boldsymbol{c_i}, t+1) &= f_i^\star(\boldsymbol{x}, t). \end{aligned} \tag{10.45}$$

This algorithm is independent of the chosen lattice (*e.g.* D2Q9, D3Q19).

The TRT model solves the Navier-Stokes equation with kinematic viscosity

$$\nu = c_\mathrm{s}^2 \left(\frac{1}{\omega^+} - \frac{1}{2} \right). \tag{10.46}$$

Thus, ω_- is a free parameter. We have already seen that it can be used to choose certain values for Λ in order to improve accuracy or stability (*cf.* chapter 4 and section 10.7).

The standard implementation is equivalent to the naive implementation but it is much more computationally efficient [35].

Efficient implementation: The most efficient implementation is as fast as the BGK model. It is similar to the standard implementation, except that one directly uses symmetric and antisymmetric populations and equilibrium populations, without explicitly computing them at every time step [6].

10.6.3 Example: diffusion of Gaussian hill

To show the advantages of the TRT over the BGK model, we simulate the diffusion of a Gaussian hill with a relatively coarse resolution ($N = 50\,\Delta x$) on a 1D lattice. Instead of solving the Navier-Stokes equation, we use the TRT model to solve the advection-diffusion equation. Therefore, we follow the common convention and denote the populations as g_i rather than f_i (*cf.* chapter 8).

The employed D1Q3 lattice has the three velocities $\{c_i\} = \{0, 1, -1\}$. We use the equilibrium distributions

$$g_0^{\mathrm{eq}} = \frac{2\rho}{3}\left(1 - \frac{3}{2}u^2\right), \quad g_1^{\mathrm{eq}} = \frac{\rho}{6}\left(1 + 3u + 3u^2\right), \quad g_2^{\mathrm{eq}} = \frac{\rho}{6}\left(1 - 3u + 3u^2\right). \tag{10.47}$$

The only difference of the advection-diffusion equation simulation with the Navier-Stokes equation simulation is that the velocity $\boldsymbol{u}$ is externally provided rather than being recovered from $\rho\boldsymbol{u} = \sum_i g_i \boldsymbol{c}_i$. The populations g_i and g_i^{eq} are decomposed into symmetric and antisymmetric parts according to eq. (10.43).

The above equilibrium and the TRT equation in eq. (10.45), applied to g_i, restore the following advection-diffusion equation for ρ (*cf.* chapter 8):

$$\partial_t \rho + \partial_x(\rho u) = D\partial_x^2 \rho. \tag{10.48}$$

The diffusion coefficient is given by [34]

$$D = \frac{1}{3}\left(\frac{1}{\omega^-} - \frac{1}{2}\right). \tag{10.49}$$

Note that the diffusion coefficient in the advection-diffusion equation is connected to ω^-, but not to ω^+ which is connected to the viscosity in the Navier-Stokes equation. The reason is that the momentum is not conserved in the advection-diffusion equation, and from the Chapman-Enskog analysis one finds that the diffusion coefficient is connected to ω^-.

We impose the initial Gaussian density profile

$$\rho(\boldsymbol{x}, t = 0) = \exp\left(-\frac{(x - x_0)^2}{2\sigma_0^2}\right) \tag{10.50}$$

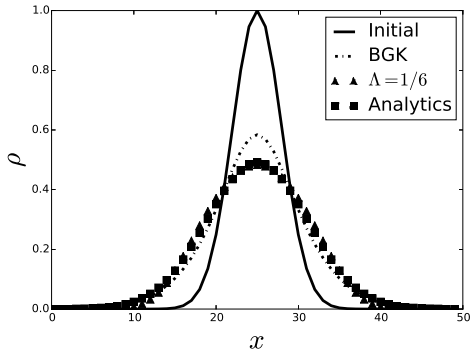


Fig. 10.2: Density profiles of the purely diffusive ($u = 0$) Gaussian hill after 15 time steps for the BGK ($\tau = 3$) and TRT ($\tau^- = 3$, $\Lambda = \frac{1}{6}$) models. TODO (TK): Figure should be revised (size, text and line style).

with the centre position $x_0 = 25\,\Delta x$ and the width $\sigma_0 = 3\,\Delta x$. The analytical solution for the time evolution is [14]

$$\rho(\boldsymbol{x}, t) = \frac{\sigma_0}{\sqrt{\sigma_0^2 + \sigma_D^2}} \exp\left(-\frac{(x - x_0 - ut)^2}{2\left(\sigma_0^2 + \sigma_D^2\right)}\right), \quad \sigma_D = \sqrt{2Dt}. \tag{10.51}$$

For the sake of simplicity, we set $u = 0$ (pure diffusion, no advection). Furthermore we choose a large relaxation time $\tau^- = 1/\omega^- = 3$ corresponding to $D = \frac{5}{6}$. By this choice, we emphasise the effect of the choice of collision operator on the accuracy error, which is described in section 4.5.

Fig. 10.2 shows the analytical profile and the simulated results for BGK ($\tau = 3$) and TRT ($\tau^- = 3$, $\Lambda = \frac{1}{6}$) after 15 time steps. On the one hand, we can clearly see a mismatch for the BGK simulation. On the other hand, the TRT model with the magic parameter $\Lambda = \frac{1}{6}$ (which cancels the fourth-order truncation error, see section 10.7 for details) yields a much more accurate result.

10.7 Choice of collision model and relaxation rates

We recall that there is a hierarchy of collision operators. MRT, as the most general relaxation model, can be reduced to the TRT model by taking one relaxation rate for all odd-order moments and another for all even-order moments. The TRT model can be further reduced to the BGK model if only one relaxation rate is used.

However, which model should be chosen in which situation, and how should the relaxation rates be chosen? We will now provide a simple guide on how to choose the collision model and the relaxation rates.

BGK model

There is only a single relaxation rate $\omega = 1/\tau$ in the BGK model. It is the simplest version that is most commonly employed. The chosen viscosity already fixes τ and therefore ω. This model has accuracy and stability deficiencies thoroughly discussed in chapter 4.

We recommend that LBM newcomers should start with the BGK model. The first step is to master non-dimensionalisation and parameter selection strategies for the BGK model (chapter 7). The next step is to understand its limitations. If the BGK deficiencies are severe for the given problem, the user should move on to TRT.

TRT model

The TRT model has the advantage that it provides more flexibility and control than the BGK model. At the same time it is still relatively easy to analyse the mathematical properties of the TRT algorithm.

There are some situations where TRT should definitely be chosen instead of BGK. One example is a system with large boundary areas (*e.g.* porous media). However, one has to understand that the TRT model itself is limited, and that it cannot cure all deficiencies of the BGK model.

As central part of the TRT model we have two relaxation rates ω^+ and ω^- and the magic parameter

$$\Lambda = \left(\frac{1}{\omega^+} - \frac{1}{2}\right)\left(\frac{1}{\omega^-} - \frac{1}{2}\right). \tag{10.52}$$

While ω^+ controls the kinematic viscosity *via* eq. (10.46), Λ can be used to control accuracy and stability. There are certain values of Λ that show distinctive properties:

- $\Lambda = \frac{1}{12}$ cancels the third-order spatial error, leading to optimal results for pure advection problems.
- $\Lambda = \frac{1}{6}$ cancels the fourth-order spatial error, providing the most accurate results for the pure diffusion equation.
- $\Lambda = \frac{3}{16}$ results in the boundary wall location implemented *via* bounce-bak for the Poiseuille flow exactly in the middle between wall and fluid nodes [1].
- $\Lambda = \frac{1}{4}$ provides the most stable simulations [36, 37].

We recommend that the user should experiment with different values of Λ for a given problem. The list above provides a starting point. When changing the viscosity *via* ω^+, Λ should be kept fixed by adapting the value of ω^-. If the underlying problems can still not be solved, the next step is to move on to MRT.

MRT model

MRT is the most general and advanced relaxation model with the largest number of free parameters to tune accuracy and stability. It also give ready-to-use ability to tune the bulk viscosity independently from the shear viscosity. Although, it is also more difficult to code and understand, and it requires more computational resources.

The full power of the MRT model can only be exploited after careful testing. For example, the optimal relaxation rates to simulate lid-driven cavity flow with the D3Q15 model are $\omega_e = 1.6$, $\omega_\epsilon = 1.2$, $\omega_q = 1.6$ and $\omega_m = 1.2$ as numerically obtained through a linear von Neumann analysis [21] (more details for the D3Q15 Gram-Schmidt model are in appendix A.6). Such an analysis for more complex problems is impractical. As general advice, the initial set of MRT relaxation rates could be the equivalent rates obtained from a tested TRT model. Thus, once ω^+ and ω^- are determined, all even-order moments are relaxed with ω^+ and all odd-order moments with ω^-. After this, individual rates can be changed to further improve the simulation accuracy and stability.

Unfortunately there is no ready-to-use recipe for choosing MRT relaxation rates different from the TRT rates. A few things, though, can be suggested in general: decrease the corresponding relaxation rate (usually ω_e in the Gram-Schmidt approach) to increase the bulk viscosity. This can suppress some underresolved numerical artifactsto improve stability and accuracy of simulations [27], and spurious sound waves are additionally suppressed (*cf.* Chapter 12).

As a final suggestion, in a typical situation when one wants to keep Reynolds number constant while changing viscosity (shear and bulk), one can achieve viscosity independent errors by keeping some parameters constant. Defining a general parameter $\Gamma_\alpha = \frac{1}{\omega_\alpha} - \frac{1}{2}$, all possible products need to be kept constant [6]. For example, the following parameter products should be kept constant for D2Q9:

- Hermite approach: $\Gamma_\nu\Gamma_{\gamma_\alpha}$, $\Gamma_\gamma\Gamma_{\gamma_\alpha}$, and $\Gamma_\zeta\Gamma_{\gamma_\alpha}$
- Gram-Schmidt approach: $\Gamma_\nu\Gamma_q$, $\Gamma_e\Gamma_q$, and $\Gamma_\epsilon\Gamma_q$

References

1. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 519 (2008)

2. A. Kuzmin, Multiphase simulations with lattice Boltzmann scheme. Ph.D. thesis, University of Calgary (2010)
3. X. He, Q. Zou, L.S. Luo, M. Dembo, J. Stat. Phys. **87**(1-2), 115 (1997)
4. I. Ginzbourg, P.M. Adler, J. Phys. II France **4**(2), 191 (1994)
5. I. Ginzburg, D. d'Humières, Phys. Rev. E **68**, 066614 (2003)
6. S. Khirevich, I. Ginzburg, U. Tallarek, J. Comp. Phys. **281**, 708 (2015)
7. D. d'Humières, I. Ginzburg, Comput. Math. Appl. **58**, 823 (2009)
8. Y.H. Qian, Y. Zhou, Europhysics Letters **42**(4), 359 (1998)
9. P. Lallemand, L.S. Luo, Phys. Rev. E **61**(6), 6546 (2000)
10. P.J. Dellar, J. of Comput. Phys. **259**, 270 (2014)
11. G. Silva, V. Semiao, J. Comput. Phys. **269**, 259 (2014)
12. B. Servan-Camas, F. Tsai, Adv. Water Res. **31**, 1113 (2008)
13. I. Ginzburg, Phys. Rev. E **77**(066704), 1
14. I. Ginzburg, Adv. Water Res. **28**(11), 1171 (2005)
15. J. Latt, Hydrodynamic limit of lattice Boltzmann equations. Ph.D. thesis, University of Geneva (2007)
16. J. Latt, B. Chopard, Mathematics and Computers in Simulation **72**(2-6), 165 (2006)
17. R. Adhikari, S. Succi, Phys. Rev. E **78**(066701), 1 (2008)
18. P.J. Dellar, J. Comp. Phys. **190**, 351 (2003)
19. A. Kuzmin, A. Mohamad, S. Succi, Int. J. Mod. Phys. C **19**(6), 875 (2008)
20. F. Higuera, S. Succi, R. Benzi, Europhysics Lett. **9**(4), 345
21. D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, L.S. Luo, Phil. Trans. R. Soc. Lond. A **360**, 437 (2002)
22. P.J. Dellar, Physical Review E **65**(3) (2002)
23. R. Benzi, S. Succi, M. Vergassola, Physics Report **222**(3), 145 (1992)
24. P. Asinari, Phys. Rev. E **77**(056706), 1 (2008)
25. R. Rubinstein, L.S. Luo, Phys. Rev. E **77**(036709), 1 (2008)
26. P. Dellar, Phys. Rev. E **64**(3) (2001)
27. P. Asinari, I. Karlin, Phys. Rev. E **81**(016702), 1 (2010)
28. Z. Guo, C. Zheng, B. Shi, Phys. Rev. E **65**, 46308 (2002)
29. A. Kuzmin, Z. Guo, A. Mohamad, Phil. Trans. Royal Soc. A **369**, 2219 (2011)
30. G. Silva, V. Semiao, J. Fluid Mech. **698**, 282 (2012)
31. S. Mukherjee, J. Abraham, Comp. & Fluids **36**, 1149 (2007)
32. K. Premnath, J. Abraham, J. Comput. Phys. **224**, 539 (2007)
33. P. Lallemand, L.S. Luo, Phys. Rev. E **68**, 1 (2003)
34. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
35. I. Ginzburg, Adv. Water Res. **28**(11), 1196 (2005)
36. I. Ginzburg, D. d'Humieres, A. Kuzmin, J. Stat. Phys. **139**, 1090 (2010)
37. A. Kuzmin, I. Ginzburg, A. Mohamad, Comp. Math. Appl. **61**, 1090 (2011)

Chapter 11
Boundary conditions for fluid-structure interaction

Boundary conditions play a paramount role in hydrodynamics. Chapter 5 concerns itself with the definition and conceptual introduction of boundary conditions, and it provides an overview of boundary conditions for relatively simple geometries, flow inlets and outlets and periodic systems. Here, we turn our attention to resting and moving boundaries with complex shapes (section 11.1). It is impossible to give an exhaustive overview of all available boundary conditions for fluid-structure interaction in the LBM. We will therefore focus on the most prominent examples: bounce-back methods in section 11.2, extrapolation methods in section 11.3 and immersed-boundary methods in section 11.4. We provide a list of comparative benchmark studies and an overview of the strengths and weaknesses of the discussed boundary conditions in section 11.5.

11.1 Why we need boundary conditions for complex geometries

Many works about boundary conditions in the LBM assume flat, resting and rigid boundaries. We have reviewed a selection of those methods in chapter 5. But our experience tells us that only a small number of boundaries in fluid dynamics obey these assumptions. In reality, most boundaries are curved, some can move and others are deformable. Prominent examples are porous media, curved surfaces of cars and planes in aerodynamics, suspensions (*e.g.* clay, slurries) or deformable objects (*e.g.* cells, wings, compliant containers). Analytical solutions are often impossible to obtain, which makes computer simulations an indispensible tool. This challenge led to a remarkable variety of proposed methods to model complex boundaries in LB simulations.

In order to accurately describe a complex domain, there are essentially two options (fig. 11.1). The first approach is to formulate the problem in a coordinate system which fits the shape of the boundary. This leads to body-fitted meshes where the boundary treatment itself is trivial. However, this way we lose the advantages of the simple cartesian grid. For example, if the boundary shape changes in time,

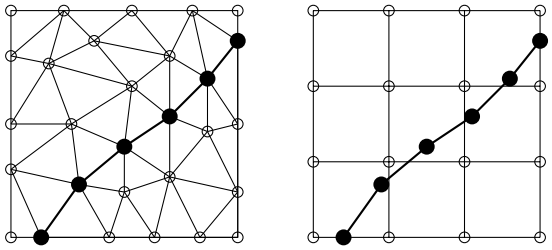


Fig. 11.1: Unstructured and structured meshes. The same boundary problem (solid black circles connected with thick lines) can be treated by an unstructured (left) or structured (right) approach. The former requires remeshing if the boundary moves, the latter leads to interpolations or extrapolations near the boundary.

remeshing becomes necessary. This can be a challenging and time-consuming task [1]. The alternative is to retain the cartesian structure of the bulk geometry, but then we have to introduce special procedures to account for the complex shape of the boundary which does generally not conform with the underlying lattice structure. In the end, this leads to interpolation and extrapolation boundary schemes.

Since most LB algorithms take advantage of the cartesian grid, the second route is usually preferred. First, it is easier to correct only the behaviour of the boundary nodes than touching all bulk nodes. Secondly, remeshing of the bulk involves interpolations in the entire numerical domain, which can lead to detrimental numerical viscosities (hyperviscosities) which in turn can reduce the accuracy and stability. TODO (GS): Do you have a citation for this statement? It is therefore less harmful to use interpolations only in the vicinity of the boundaries. In this chapter we will exclusively address boundary treatments of the second kind, where the underlying lattice structure is not changed.

There are different types of problems which are typically encountered in connection with off-lattice boundaries. We can identify three main categories:

- stationary rigid obstacles (*e.g.* porous media, microfluidic devices, flow over stationary cylinder)
- moving rigid obstacles (*e.g.* suspensions of non-deformable particles, oscillating cylinder, rotating turbine blades)
- moving deformable obstacles (*e.g.* flexible wings, red blood cells, compliant channels)

No single numerical boundary treatment works best for all of them. It is therefore worth to properly categorise the problem first, identify the main challenge and then "shop around" and look for the most suitable boundary treatment for the problem at hand. This chapter helps the reader to understand what the differences of the available methods are, when they are applicable and what their advantages and disadvantages are.

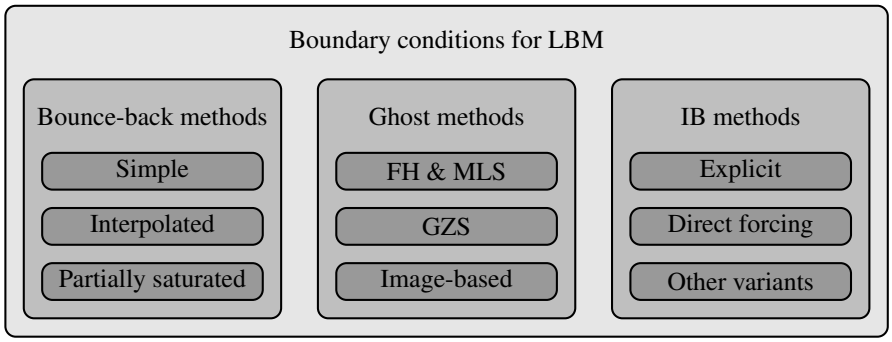


Fig. 11.2: Overview of boundary conditions for complex geometries in LB simulations as presented in this chapter. We can roughly distinguish between bounce-back, ghost and immersed-boundary (IB) methods. Each of them has several flavours. A large selection of those is covered in the following.

There exists a zoo of curved boundary conditions for LB simulations. We can only cover the most popular ones in any depth, but we will provide references to a wider range of boundary conditions in passing. Fig. 11.2 shows an overview of the boundary conditions discussed here. For the sake of compactness, we only consider single-phase fluids. Note that everything said in this chapter does equally apply to 2D and 3D systems.

11.2 Bounce-back methods

The most famous and certainly easiest boundary condition for LB simulations is bounce-back (section 5.2.3). Many researchers believe that its locality, simplicity and efficiency should be retained even in the presence of complex boundary shapes. Therefore, the obvious way is to approximate a curved boundary by a staircase (section 11.2.1). This can lead to some problems, in particular a reduction of the numerical accuracy. For that reason, improved and interpolated bounce-back schemes have been proposed (section 11.2.2). Another variant to account for complex geometries is the partially saturated method (section 11.2.3). A problem related to staircase and interpolated bounce-back BCs is the destruction and creation of fluid sites if the boundaries move. We will discuss the creation of so-called *fresh nodes* in section 11.2.4. Finally we will elaborate on the calculation of the wall shear stress in the presence of complex boundaries (section 11.2.5). We recommend reading [2, 3, 4, 5, 6, 7] to understand bounce-back methods in greater detail.

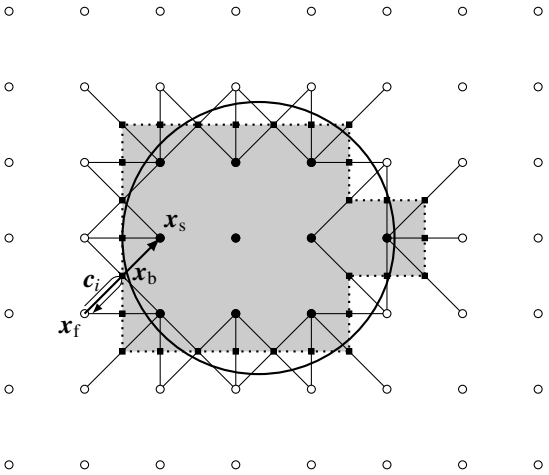


Fig. 11.3: Staircase approximation of a circle. A circle (here with an unrealistically small radius $r = 1.8\Delta x$) can be discretised on the lattice by identifying exterior fluid nodes (open circles) and interior solid nodes (solid circles) first. Any lattice link $\boldsymbol{c}_i$ connecting a fluid and a solid node is a cut link (lines) with a boundary node (solid squares) in the middle. The resulting staircase shape is shown as a grey-shaded area. Populations moving along cut links $\boldsymbol{c}_i$ (defined as pointing inside the solid) from $\boldsymbol{x}_\mathrm{f}$ to $\boldsymbol{x}_\mathrm{s}$ are bounced back at $\boldsymbol{x}_\mathrm{b}$.

11.2.1 Simple bounce-back and staircase approximation

One of the motivations to simulate complex geometries is to study flows in porous media. The simplest way to introduce curved or inclined boundaries in LB simulations is through a *staircase approximation* of the boundary and the bounce-back scheme, often called *simple bounce-back* (SBB, section 5.2.3). This is illustrated in fig. 11.3. The advantages are obvious: everything lives on the lattice, and SBB is fast and easy to implement. The problem becomes more complex when the boundaries can move, which requires the destruction and creation of fluid sites (section 11.2.4).

Revision of the half-way bounce-back method

In the following we will only consider the half-way bounce-back scheme: an incoming (post-collision) population $f_i^\star(\boldsymbol{x}_\mathrm{f}, t)$ which would propagate through a boundary from a fluid node $\boldsymbol{x}_\mathrm{f}$ to a solid node $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{f} + \boldsymbol{c}_i\Delta t$ is instead reflected half-way to the solid node at the boundary location $\boldsymbol{x}_\mathrm{b} = \boldsymbol{x}_\mathrm{f} + \frac{1}{2}\boldsymbol{c}_i\Delta t$ at time $t + \frac{1}{2}\Delta t$ and returns to $\boldsymbol{x}_\mathrm{f}$ as

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t) = f_i^\star(\boldsymbol{x}_\mathrm{f}, t) - 2w_i\rho\boldsymbol{u}_\mathrm{b} \cdot \boldsymbol{c}_i \tag{11.1}$$

where $\boldsymbol{u}_\mathrm{b} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{b}, t + \frac{1}{2}\Delta t)$ is the velocity of the boundary, ρ is the fluid density at $\boldsymbol{x}_\mathrm{b}$ and $\bar{i}$ is defined by $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$. In practical implementations, ρ is often taken as the fluid density at $\boldsymbol{x}_\mathrm{f}$ or the average fluid density instead.

The half-way bounce-back scheme requires detection of all lattice links $\boldsymbol{c}_i$ intersecting the boundary. If the boundary is stationary, this has to be done only once.

We can compute the momentum exchange at the wall based on the incoming and bounced back populations alone by using the momentum exchange algorithm (MEA, section 5.3.3). Here we will briefly revise the MEA for the simple bounce-back method. The first step is to evaluate the incoming and bounced back populations $f_i^\star$ and $f_{\bar{i}}$ at each boundary link. The total momentum exchange between the fluid and the solid during one time step is given by eq. (5.67):

$$\begin{aligned} \Delta\boldsymbol{P} &= \Delta x^3 \sum_{\boldsymbol{x}_\mathrm{b},i} \left(f_i^\star(\boldsymbol{x}_\mathrm{b} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) + f_{\bar{i}}(\boldsymbol{x}_\mathrm{b} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t + \Delta t) \right) \boldsymbol{c}_i \\ &= \Delta x^3 \sum_{\boldsymbol{x}_\mathrm{b},i} \left(2f_i^\star(\boldsymbol{x}_\mathrm{b} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) - 2w_i\rho\boldsymbol{u}_\mathrm{b}\cdot\boldsymbol{c}_i \right) \boldsymbol{c}_i \end{aligned} \tag{11.2}$$

where the sum runs over all incoming links $\boldsymbol{c}_i$ (pointing from the fluid into the solid) intersecting the boundary at $\boldsymbol{x}_\mathrm{b}$. Accordingly, the total angular momentum exchange during one time step is

$$\Delta\boldsymbol{L} = \Delta x^3 \sum_{\boldsymbol{x}_\mathrm{b},i} \left(2f_i^\star(\boldsymbol{x}_\mathrm{b} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) - 2w_i\rho\boldsymbol{u}_\mathrm{b}\cdot\boldsymbol{c}_i \right) (\boldsymbol{x}_\mathrm{b} - \boldsymbol{R}) \times \boldsymbol{c}_i. \tag{11.3}$$

$\boldsymbol{R}$ is a reference point. If the torque acting on a particle is computed, the reference point is the particle's centre of mass.

The MEA works for any geometry approximated by the simple bounce-back scheme, including the staircase shown in fig. 11.3. The tedious part is the identification of all cut links pointing from a fluid to a solid node and obtaining the boundary velocity $\boldsymbol{u}_\mathrm{b}$ at each of the boundary locations $\boldsymbol{x}_\mathrm{b}$.

Stationary boundaries

There is a large number of applications where the flow in a complex stationary geometry has to be simulated. Examples are flows in porous media or blood flow in the vascular system (fig. 11.4). Those geometries can by obtained by, for example, CT or MRI scans. Due to the large surface and complex shape of those geometries, it is preferable to use a simple and fast boundary condition algorithm, such as SBB.

Back in the 90s, Ginzbourg and Adler [2] presented a very careful analysis of half-way SBB with several important conclusions. Apart from developing general theoretical tools to study boundary conditions, one contribution was the understanding of the numerical mechanism leading to the velocity slip at the wall. Furthermore, the violation of the no-slip condition is anisotropic with respect to the underlying lattice structure, *i.e.* the slip velocity depends on the way the boundary is inclined. In particular, for a Poiseuille flow in a straight channel, the wall can only be *exactly* lo-

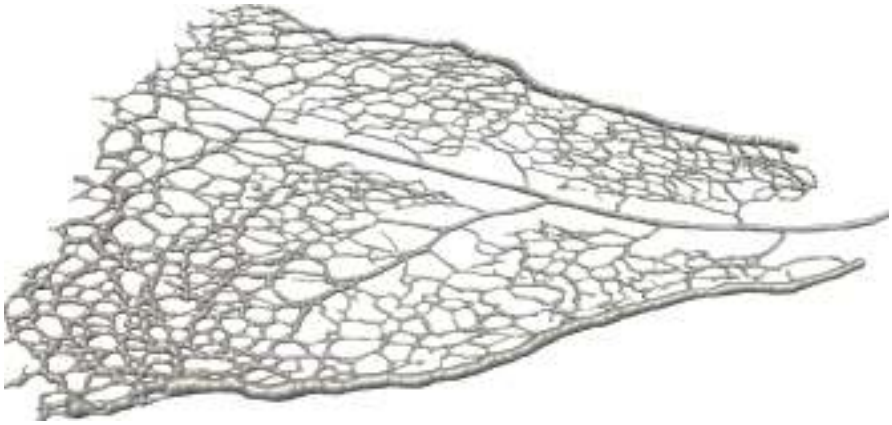


Fig. 11.4: Visualisation of a segment of the blood vessel network in a murine retina which has been used for LB simulations. This example shows the complexity of the involved boundaries. Original confocal microscope images courtesy of Claudio A. Franco and Holger Gerhardt. Luminal surface reconstruction courtesy of Miguel O. Bernabeu and Martin L. Jones. For more details see [8].

cated midway between lattice nodes if $\tau = 1/2 + \sqrt{3/16} \approx 0.933$ (*cf.* section 5.2.3). Contrarily, for a diagonal channel, $\tau = 1/2 + \sqrt{3/8} \approx 1.11$ has to be chosen. The slip velocity is viscosity-dependent and therefore a highly undesired numerical effect. This shows that the no-slip condition will generally not be satisfied in complex geometries approximated by a staircase and that the numerical error will depend on the choice of the relaxation time τ. In other words, the wall is apparently not exactly at the expected location.

Additionally, the combination of SBB with the BGK collision operator leads to strongly τ-dependent results when the pore or channel size is small (a few Δx), in particular when τ is large. This is problematic for porous media with small pore sizes where the effective porosity becomes a function of τ. Using MRT with a properly tuned set of relaxation rates [5], rather than BGK, improves the situation significantly. When τ is chosen close to 1 and the average pore size is large, SBB with BGK leads to acceptable results, though.

We also have to note that biological geometries obtained from CT or MRI scans are usually not very accurate in the first place. The resolution of those imaging techniques can be of the same order as the pore or channel size so that it does not make much sense trying to increase the resolution of the numerical domain or choose more accurate boundary conditions. This means that SBB, although generally not the most recommendable solution, is still a good choice given the large modelling error in many applications. Furthermore, it is worth mentioning that SBB is exactly mass-conserving when used for stationary geometries of any shape; an advantage only a few second-order accurate boundary conditions can claim. Therefore, before

setting up a simulation, one should always ask whether the boundary condition is really the limiting factor in terms of accuracy.

Using **simple bounce-back (SBB)** for complex geometries generally leads to two sources of error:

1. geometrical discretisation error (modelling error) by approximating a complex shape by a staircase
2. artificial and anisotropic slip caused by the combination of BGK and SBB.

The advantages of SBB (mass conservation, ease of implementation, locality) explain why it is still a popular method.

Rigid moving particles

So far we have only addressed stationary boundaries with SBB. In the 90s researchers became interested in the simulation of suspensions *via* LBM. This requires the treatment of multiple rigid particles with translational and rotational degrees of freedom. One of the problems of earlier computational suspension models was the numerical cost which scaled with the square or cube of the particle number [3]. Ladd [9, 3, 10] introduced an LB-based model for suspensions of rigid particles with hydrodynamic interactions whose numerical cost scales linearly with the particle number.

Particle suspensions give rise to a plethora of physical effects and phenomena. In this section we will only focus on the algorithmic details. For physical results we refer to review articles about LB-based suspension simulations [11, 12] and the references therein.

For the sake of brevity we will not discuss lubrication forces which become necessary at high particle volume fractions. There exist several articles dealing with lubrication corrections in LB simulations [13, 14, 12]. The review by Ladd and Verberg [11], which we generally recommend to read, also descibes the use of thermal fluctuations for the simulation of Brownian motion in suspensions. Aidun and Clausen [12] have published a review about LBM for complex flows, which is an excellent starting point to learn about more recent developments.

In the following we will outline Ladd's [3, 10] idea of how to use SBB for suspensions. See also [13] for a compact and [11] for an extensive review of Ladd's method. Note that the particles in Ladd's algorithm are filled with fluid in order to avoid destruction and recreation of fluid nodes when the particles move on the lattice. The dynamics of the interior fluid is therefore fully captured. One can imagine this like a can filled with liquid concrete in an exterior fluid rather than the same can with set (and therefore solid) concrete. This is different compared to Aidun's model [15] which we will briefly describe at the end of this section.

The first step is to start with a distribution of suspended spherical particles on the lattice. For each particle, it is straightforward to work out which lattice nodes

are located inside and outside of a particle (fig. 11.3). There is no conceptual difficulty in extending the model to non-spherical particles; but it will generally be more demanding to identify interior and exterior lattice nodes.

The next step is to identify all lattice links between fluid and solid nodes, *i.e.* those links cut by any particle surface. For moving boundaries, the list of those links has to be updated whenever the boundary configuration on the lattice changes. Generally one has to update the list every time step before propagation is performed. In the following, let $\boldsymbol{x}_\mathrm{f}$ be the location of a fluid node just outside the particle and $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{f} + \boldsymbol{c}_i \Delta t$ the location of a solid node just inside the particle. The boundary link is then located at $\boldsymbol{x}_\mathrm{b} = \boldsymbol{x} + \frac{1}{2}\boldsymbol{c}_i \Delta t$ (fig. 11.3).

Now we have to compute the boundary velocity $\boldsymbol{u}_\mathrm{b}$ at each boundary link $\boldsymbol{x}_\mathrm{b}$. From the known linear velocity $\boldsymbol{U}$ and angular velocity $\boldsymbol{\Omega}$ of the particle we can obtain

$$\boldsymbol{u}_\mathrm{b} = \boldsymbol{U} + \boldsymbol{\Omega} \times (\boldsymbol{x}_\mathrm{b} - \boldsymbol{R}) \tag{11.4}$$

where $\boldsymbol{R}$ is the particle's centre of mass.

With the known boundary velocity at each link, we can compute the momentum exchange and therefore the value of all bounced-back populations. We have to take into account that a particle in Ladd's method is filled with fluid, as explained earlier, and all interior nodes participate in collision and propagation as well. This means that there are also populations streaming from the interior nodes at $\boldsymbol{x}_\mathrm{s}$ to exterior nodes at $\boldsymbol{x}_\mathrm{f}$. These populations have to be bounced back at $\boldsymbol{x}_\mathrm{b}$, too. While the populations streaming from the outside to the particle's interior are described by eq. (11.1), we now also have to consider those populations streaming from the inside towards the exterior:

$$f_i(\boldsymbol{x}_\mathrm{s}, t + \Delta t) = f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{s}, t) - 2w_{\bar{i}}\rho \boldsymbol{u}_\mathrm{b} \cdot \boldsymbol{c}_{\bar{i}} = f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{s}, t) + 2w_i \rho \boldsymbol{u}_\mathrm{b} \cdot \boldsymbol{c}_i. \tag{11.5}$$

Eq. (11.1) and eq. (11.5) express that the two populations hitting a boundary link from both sides exchange a certain amount of momentum, $2w_i\rho\boldsymbol{u}_\mathrm{b}\cdot\boldsymbol{c}_i$. This operation is obviously mass-conserving since f_i gains exactly the loss of $f_{\bar{i}}$ (or the other way around) so that the sum of both populations moving along the same link in different directions is not changed by the interaction with the boundary, at least as long the same density ρ is used in both equations. Ladd uses the average fluid density, and not the local density, for ρ.

Effectively, we can view the momentum transferred from the exterior to the interior fluid as the momentum transferred from the fluid to the particle. In order to obey the global momentum and angular momentum conservation, we therefore have to update the particle momentum and angular momentum by summing up all transferred contributions. Eq. (11.2) and eq. (11.3) provide the total momentum $\Delta\boldsymbol{P}$ and angular momentum $\Delta\boldsymbol{L}$ transferred during one time step, but we have to take into account that each link has to be counted twice: once for all populations coming from the outside and once for all populations coming from the inside. This is necessary because the interior fluid participates in collision and propagation and therefore the momentum exchange.

The simplest way to update the particle properties is the forward Euler method, but more accurate and more stable methods are available, *e.g.* implicit time integration [11]. At each time step, the velocity and angular velocity are updated according to

$$\boldsymbol{U}(t+\Delta t) = \boldsymbol{U}(t) + \frac{\Delta \boldsymbol{P}}{M}, \quad \boldsymbol{\Omega}(t+\Delta t) = \boldsymbol{\Omega}(t) + \boldsymbol{I}^{-1} \cdot \Delta \boldsymbol{L} \tag{11.6}$$

where M and $\boldsymbol{I}$ are the particle's mass and tensor of inertia. The centre of mass is then moved according to the old or new velocity.[1] If the particles are spherical, their orientation does not have to be updated. For non-spherical particles, however, the situation is different, and several authors have suggested algorithms for this case. Aidun *et al.* [15], for example, use a fourth-order Runge-Kutta integration to update the particle orientation. Qi [16] employed quaternions to capture the particle orientation and a leap-frog time integration. It is noteworthy that Ladd [3] does not follow the simple scheme in eq. (11.6). He instead averages the momentum and angular momentum transfer over two time steps before updating the particle properties. The reason for this is to reduce the undesired effect of so-called *staggered momenta* which are an artefact of lattice-based methods. We refer to [3] for a more thorough discussion of this issue.

Ladd's algorithm [3] can be summarised in the following way:

1. Find the particle discretisation on the lattice (fig. 11.3).
2. Identify all boundary links and compute $\boldsymbol{u}_\mathrm{b}$ by applying eq. (11.4).
3. Perform collision on *all* nodes since particles are filled with fluid.
4. Propagate the populations. If a population moves along a boundary link, bounce-back this population *via* eq. (11.1) or eq. (11.5).
5. Compute the total momentum and angular momentum exchange according to eq. (11.2) and eq. (11.3).
6. Update the particle configuration, for example *via* eq. (11.6).
7. Go back to step 1 for the next time step. There is no need to treat nodes crossing a boundary in a special way.

It is interesting to note why Ladd has chosen a link-based (half-way) rather than a node-based (full-way) bounce-back method. The simple explanation is that the link-based bounce-back leads to a "somewhat higher resolution" [3] for the same discretisation since there are more cut links than solid nodes near the particle surface. This can be easily seen in fig. 11.3.

Ladd [3] pointed out that his method has a few disadvantages. First, the dynamics of the fluid inside the particles can affect the particle dynamics at higher Reynolds numbers where the interior fluid cannot any more be approximated by an effectively rigid medium. Furthermore, Ladd's method is limited to situations where the particle density is larger than the fluid density. Aidun *et al.* [15] proposed an alternative method with one major distinctive feature: the absence of fluid inside the particles. Therefore, in Aidun's approach, only the exterior fluid contributes to the momentum and angular momentum exchange in eq. (11.2) and eq. (11.3). Removing the fluid from the interior solves both disadvantages of Ladd's method, but it also introduces a

[1] The velocities are usually small so that the exact form of the position update is not very important.

new complexity: what happens when lattice nodes change their identity (fluid nodes become solid nodes and the other way around) when the particles move? We will come back to this point in section 11.2.4. In contrast to Ladd's approach, Aidun's method does not obey global mass conservation [15].

We emphasise that several researchers have further improved the methods presented above. For example, Lorenz *et al.* [17], Clausen and Aidun [18] and Wen *et al.* [19] proposed modified versions of the momentum exchange to improve Galilean invariance.

We have only discussed *link-based* BB schemes in this section. It is possible to implement node-based BB schemes for complex geometries where the boundary velocity is enforced directly on lattice nodes, though. Behrend [20] and Gallivan *et al.* [21] provide discussions of the node-based BB approach. In section 11.2.3 we will present partially saturated methods which are also built on the node-based BB scheme.

As pointed out by Han and Cundall [22], the **simple bounce-back (SBB) applied to moving boundaries** has its limitations compared to higher-order schemes, such as the partially saturated method (section 11.2.3). This becomes most obvious when the particles are rather small (a few Δx in diameter) and move on the lattice. Eventually, the user has to decide whether the focus lies on the ease of implementation or level of accuracy. In the former case, SBB can be recommended. In the latter, a smoother boundary condition should be implemented.

11.2.2 Interpolated bounce-back

We will now cover interpolated bounce-back (IBB) methods which are suitable to describe curved and inclined boundaries with a second-order accuracy. We emphasise the conceptual difference between IBB schemes and extrapolation-based methods (section 11.3). While the idea of the former is to interpolate populations in the fluid region to perform bounce-back at a curved wall, the motivation for the latter is to create a virtual (ghost) fluid node inside the solid to compute the populations streaming out of the wall. We generally recommend reading [5, 23, 24] for thorough reviews of IBB methods.

Basic algorithm

In 2001, Bouzidi *et al.* [4] proposed the IBB approach for curved boundaries. The IBB is second-order accurate for arbitrary boundary shapes and therefore reduces the modelling error of the staircase bounce-back method which is only first-order accurate for non-planar boundaries.

Fig. 11.5: Interpolated bounce-back cases. For $q < \frac{1}{2}$ (left), the distance d between the wall (vertical line) and the next fluid node at $\boldsymbol{x}_\mathrm{f}$ is smaller than half a lattice spacing. Interpolations are required to construct the post-collision population (small black circle). For $q \geq \frac{1}{2}$ (right), d is larger than half a lattice spacing and the end-point of the streaming population (small black circle) lies between the wall and $\boldsymbol{x}_\mathrm{f}$. $\boldsymbol{x}_\mathrm{s}$ denotes a solid node behind the wall, and $\boldsymbol{x}_\mathrm{ff}$ is a second fluid node required for the interpolation.

The basic idea of IBB is to include additional information about the wall location during the bounce-back process. A boundary link $\boldsymbol{c}_i$ generally intersects the wall at a distance d between 0 and $|\boldsymbol{c}_i|\Delta t$ measured from the fluid node (fig. 11.5). We define $q = d/(|\boldsymbol{c}_i|\Delta t) \in [0, 1[$ as the reduced wall location and note that $q = \frac{1}{2}$ holds for simple bounce-back. In the following, we introduce three lattice nodes with locations $\boldsymbol{x}_\mathrm{f}$, $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{f} + \boldsymbol{c}_i\Delta t$ and $\boldsymbol{x}_\mathrm{ff} = \boldsymbol{x}_\mathrm{f} - \boldsymbol{c}_i\Delta t$ as shown in fig. 11.5. $\boldsymbol{x}_\mathrm{f}$ and $\boldsymbol{x}_\mathrm{s}$ are neighbouring fluid and solid nodes which are located on either side of the wall, and $\boldsymbol{x}_\mathrm{ff}$ is the nearest fluid node beyond $\boldsymbol{x}_\mathrm{f}$.

The starting point of IBB is to assume that any population f_i moves a distance $|\boldsymbol{c}_i|\Delta t$ during propagation. If the population hits a wall which is modelled by the half-way bounce-back, f_i first travels a distance $|\boldsymbol{c}_i|\Delta t/2$ from the original fluid node to the wall and then another distance $|\boldsymbol{c}_i|\Delta t/2$ back to the fluid node after bounce-back. If the wall is not located half-way between lattice nodes, $q \neq \frac{1}{2}$, f_i cannot reach another lattice node. Therefore, the origin of the population is chosen such that f_i exactly reaches a lattice node. This requires interpolation to find the post-collision value of f_i as illustrated in fig. 11.5.

Bouzidi *et al.* [4] proposed a linear interpolation to construct the *a priori* unknown bounced back population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t)$ from the post-collision values of the known populations at $\boldsymbol{x}_\mathrm{f}$ and $\boldsymbol{x}_\mathrm{ff}$. The algorithm for any cut link at a resting wall reads

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t) = \begin{cases} 2q f_i^\star(\boldsymbol{x}_\mathrm{f}, t) + (1 - 2q) f_i^\star(\boldsymbol{x}_\mathrm{ff}, t) & q \leq \frac{1}{2} \\ \frac{1}{2q} f_i^\star(\boldsymbol{x}_\mathrm{f}, t) + \frac{2q-1}{2q} f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{f}, t) & q \geq \frac{1}{2} \end{cases} . \tag{11.7}$$

Exercise 11.1. Show that both cases in eq. (11.7) reduce to simple bounce-back for $q = \frac{1}{2}$.

There are several remarks:

- The reason for having different expressions for $q < \frac{1}{2}$ and $q \geq \frac{1}{2}$ is to ensure that $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t)$ is always non-negative (given that the post-collision populations on

the right-hand-side of eq. (11.7) are positive), which improves the stability of the algorithm.
- Bouzidi *et al.* [4] have also proposed a quadratic interpolation which leads to more accurate results. This extension requires a third fluid node $\boldsymbol{x}_{\mathrm{fff}}$.
- The boundary slip for linear and quadratic interpolations still depends on viscosity; IBB therefore does not eliminate this shortcoming of simple bounce-back.
- The IBB algorithm, like simple bounce-back, is completely decoupled from the collision step and can therefore be combined with any collision operator.
- IBB leads to problems in very narrow gaps (*e.g.* in porous media simulations) where there are not enough fluid nodes between neighbouring walls to apply eq. (11.7) (two fluid nodes required) or the quadratic interpolation (three fluid nodes required). Chun and Ladd [23] have addressed this issue and proposed an alternative.
- Due to its interpolations IBB is generally not mass-conserving. There exist approaches to (partially) remedy this shortcoming, *e.g.* [24].

The interpolated bounce-back algorithm can be summarised as follows:

1. Identify all links penetrating a boundary and compute their reduced distance q. If the boundary configuration does not change in time, this has to be done only once.
2. Collide on all fluid nodes. This will provide $f_i^\star(\boldsymbol{x}_{\mathrm{f}}, t)$, $f_{\bar{i}}^\star(\boldsymbol{x}_{\mathrm{f}}, t)$ and $f_i^\star(\boldsymbol{x}_{\mathrm{ff}}, t)$.
3. Compute all $f_{\bar{i}}(\boldsymbol{x}_{\mathrm{f}}, t + \Delta t)$ from eq. (11.7).
4. Propagate all remaining populations.
5. Go back to step 1.

The extension of the momentum exchange algorithm to the IBB is straightforward. According to Bouzidi *et al.* [4], the momentum exchange for a resting boundary link in fig. 11.5 is

$$\Delta \boldsymbol{p}_i = \Delta x^3 \left(f_i^\star(\boldsymbol{x}_{\mathrm{f}}, t) - f_{\bar{i}}(\boldsymbol{x}_{\mathrm{f}}, t + \Delta t) \right) \boldsymbol{c}_i, \tag{11.8}$$

no matter the value q of the link.

Moving boundaries

Most IBB applications in the literature deal with rigid boundaries (which can either be stationary or move on the lattice). There is only a small number of works featuring the IBB for deformable boundaries, *e.g.* [25]. The immersed boundary method (section 11.4) is a more common approach for deformable boundaries. We will therefore not discuss deformable boundaries here.

Lallemand and Luo [26] extended the IBB to moving boundaries. The first ingredient is Ladd's algorithm: the term $-2w_i\rho\boldsymbol{u}_{\mathrm{b}} \cdot \boldsymbol{c}_i$ has to be added to the right-hand-side of eq. (11.7), where $\boldsymbol{u}_{\mathrm{b}}$ is the boundary velocity at the intersection point. Also, since there is generally no fluid inside the solid in the IBB, a refill mechanism has to be implemented when boundaries move and uncover new (fresh) fluid nodes. We

will get back to this in section 11.2.4. Lallemand and Luo's important finding is that the motion of a cylinder on the lattice and the subsequent destruction and creation of fresh fluid nodes leads to some fluctuations of the drag coefficient. This is one of the largest disadvantages of the IBB for moving boundaries; a problem which can be reduced by using an advanced fresh node treatment (section 11.2.4) or the immersed boundary method (section 11.4).

Extended and alternative methods

Several other second-order accurate bounce-back-based boundary conditions for arbitrary geometries have been proposed. The following list shows a selection of those methods and their most notable properties.

- Yu *et al.* [27] presented a unified scheme of Bouzidi's algorithm which does not require separate treatment of the regions $q < \frac{1}{2}$ and $q \geq \frac{1}{2}$. Otherwise Yu's and Bouzidi's approaches lead to similar results, including viscosity-dependent slip and violation of mass conservation.
- Ginzburg and d'Humières [28] proposed the so-called *multireflection* boundary condition as an enhanced IBB. Due to a post-collision correction, the multireflection method is formally third-order accurate. It is often claimed that this boundary condition is the most accurate one for LB simulations. However, the rigorous analysis in [28] is only valid for steady flows. Mass conservation is generally violated and the multireflection method requires three fluid nodes between neighbouring boundaries.
- Chun and Ladd [23] proposed a method based on the interpolation of the equilibrium distribution. It has the advantage that it requires only one fluid node — in contrast to two or three fluid nodes in the linear and quadratic IBB scheme. The exact boundary location is viscosity-independent. This approach is therefore suitable for the simulation of geometries with narrow gaps between solids, *e.g.* for porous media. Mass conservation is generally violated, but the authors claim that this does not significantly affect the simulated velocity profiles.
- Kao and Yang [24] suggested an interpolation-free method based on the idea of local grid refinement in order to improve the mass conservation of the boundary condition.
- Yin and Zhang [29] presented another improved bounce-back scheme. Their idea was to use Ladd's momentum correction term and linearly interpolate the fluid velocity between a nearby fluid node and the boundary location (which can be anywhere between two lattice nodes) to obtain the fluid velocity midway between fluid and solid nodes. This promising method is easy to implement, but also shares common disadvantages with other interpolated bounce-back schemes: violation of mass conservation and viscosity-dependent wall location.

The **interpolated bounce-back (IBB) method** is a common extension of the simple bounce-back scheme for rigid resting or moving obstacles with com-

plex shapes. IBB is second-order accurate but inherits an important weakness of the standard bounce-back: viscosity-dependent boundary slip. Furthermore, due to the involved interpolations, IBB is not mass-conserving. Even so, due to its relatively simple implementation and accuracy IBB is often the method of choice for stationary complex geometries.

11.2.3 Partially saturated bounce-back

Now we present the so-called *partially saturated* method (PSM), also known as *grey* LB model or *continuous bounce-back*, where a lattice node can be a pure fluid, a pure solid or a mixed (partially saturated) node as shown in fig. 11.6. Interestingly, there exist two research communities which do not seem to interact strongly. The first applies the PSM to simulations of flows in porous media with sub-grid resolution and heterogeneous permeability[30, 31, 32]. The other community is interested in suspension flows; they employ the PSM to map the sharp surface of an immersed structure onto the lattice [33, 6, 22, 34]. For the sake of brevity and since both approaches are technically similar, we only elaborate on the latter application. We emphasise that the PSM must not be confused with immersed boundary schemes (section 11.4) which are, according to our definition, fundamentally different in nature.

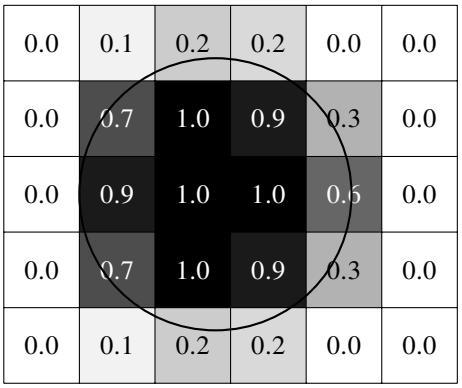


Fig. 11.6: Partially saturated bounce-back. A spherical particle (circle) covers a certain amount of each lattice cell. White corresponds to no coverage, black to full coverage. The solid fraction $0 \leq \epsilon \leq 1$ for each cell is shown up to the first digit. Lattice nodes (not shown here) are located at the centre of lattice cells.

Basic algorithm

In 1998, Noble and Torczynski [33] presented a bounce-back-based approach, later investigated more thoroughly by Strack and Cook [6], to approximate complex boundaries *on lattice nodes*. The central part of the PSM algorithm is a modified LBGK equation:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + (1 - B)\Omega_i^{\mathrm{f}} + B\Omega_i^{\mathrm{s}} \tag{11.9}$$

where

$$\Omega_i^{\mathrm{f}} = -\frac{f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)}{\tau} \tag{11.10}$$

is the standard BGK collision operator for fluid (f) nodes and

$$\Omega_i^{\mathrm{s}} = \left(f_{\bar{i}}(\boldsymbol{x}, t) - f_{\bar{i}}^{\mathrm{eq}}(\rho, \boldsymbol{u})\right) - \left(f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\rho, \boldsymbol{u}_{\mathrm{s}})\right) \tag{11.11}$$

is the collision operator for solid (s) nodes. $\boldsymbol{u}$ is the local fluid velocity and $\boldsymbol{u}_{\mathrm{s}}$ is the velocity of the boundary at point $\boldsymbol{x}$. B is a weighting parameter defined by [33]

$$B(\epsilon, \tau) = \frac{\epsilon\left(\tau - \frac{1}{2}\right)}{(1 - \epsilon) + \left(\tau - \frac{1}{2}\right)} \tag{11.12}$$

where $0 \leq \epsilon \leq 1$ is the solid fraction of the node. It can be shown that $B(\epsilon)$ increases monotonically between 0 for $\epsilon = 0$ (pure fluid node) and 1 for $\epsilon = 1$ (pure solid node) for any fixed value of $\tau > \frac{1}{2}$. The essential idea is to surrender any shape details of the off-lattice boundary and use an on-lattice volume fraction ϵ instead.

Exercise 11.2. Show that the collision operator in eq. (11.9) is mass-conserving by computing $\sum_i \Omega_i^{\mathrm{f}}$ and $\sum_i \Omega_i^{\mathrm{s}}$.

Note that the PSM assumes the standard BGK form for $B = 0$ and describes a BB of the non-equilibrium for $B = 1$. A mixed collision and BB scheme is performed for partially saturated nodes ($0 < \epsilon < 1$), which are only found in direct boundary neighbourhood (fig. 11.6).

Force and torque acting on the boundary can be computed from

$$\begin{aligned} \boldsymbol{F} &= \frac{\Delta x^3}{\Delta t} \sum_{\boldsymbol{x}_n} B(\boldsymbol{x}_n) \sum_i \Omega_i^{\mathrm{s}}(\boldsymbol{x}_n)\boldsymbol{c}_i, \\ \boldsymbol{T} &= \frac{\Delta x^3}{\Delta t} \sum_{\boldsymbol{x}_n} B(\boldsymbol{x}_n)(\boldsymbol{x}_n - \boldsymbol{R}) \times \sum_i \Omega_i^{\mathrm{s}}(\boldsymbol{x}_n)\boldsymbol{c}_i, \end{aligned} \tag{11.13}$$

respectively, where the $\boldsymbol{x}_n$ are all lattice nodes in contact with the solid (including all interior nodes), *i.e.* those nodes with $\epsilon > 0$, i runs over all lattice directions at a given position $\boldsymbol{x}_n$ and $\boldsymbol{R}$ is the location of the centre of mass of the solid. Updating the solid's momentum and angular momentum according to eq. (11.13) guarantees overall momentum conservation.

It is worth mentioning that Zhou *et al.* [35] have combined the node-based method with Lees-Edwards BCs, which is relevant for the simulation of large bulk systems. Furthermore, Chen *et al.* [34] have recently proposed a combination of the PSM and a ghost method (section 11.3) to improve the no-slip condition at the boundary surface. Yu *et al.* [36] proposed another variant taking into account a mass-conserving population migration process in the vicinity of moving walls.

Advantages and limitations

The implementation of this algorithm is relatively straightforward. If the boundary is stationary, as for example encountered for a porous medium, ϵ can be computed once on each lattice site. For moving boundaries, ϵ has to be updated, which is the most challenging aspect of the PSM. For example, Han and Cundall [22] use a sub-cell method to estimate ϵ for a given lattice site while Chen *et al.* [34] employ a cut-cell approach. Apart from updating ϵ, no additional measures have to be taken when objects are moving on the lattice. In particular, fresh nodes appearing on the rear of a moving obstacle do not have to be treated in a special way. Neither are fluid nodes destroyed when they are covered by the advancing boundary. Since the interior fluid is never destroyed and still participates in the collisions described by Ω_i^{s}, mass and momentum are conserved.

Strack and Cook [6] performed careful 3D benchmark tests of the PSM. The authors report a significantly smoother motion when the weight $B(\epsilon, \tau)$ is used, rather than just falling back to a staircase approximation of the boundary. This is mostly due to the smooth uncovering of fluid nodes which have previously been solid nodes and *vice versa*. However, the smoothness of the observables (velocity, force and torque) depends on the accurate computation of the solid ratio ϵ.

Later, Han and Cundall [22] investigated the resolution sensitivity of the PSM and Ladd's BB scheme (section 11.2.1) in 2D. They found that both methods are comparably accurate in terms of the drag coefficient of relatively large circles (diameter $\approx 10\Delta x$). However, for diameters as small as $4 - 5\Delta x$, the PSM is superior, in particular when the objects are moving on the lattice.

The PSM has a number of advantages. The first is that one does not face the fresh node problem (section 11.2.4) which causes some trouble in most of the other BB variants. Moreover, the PSM, unlike IBB, is exactly mass conserving. Another advantage is the absence of interpolations to enforce the boundary condition. This makes the PSM a promising candidate for dense suspensions and porous media with small pore sizes.

However, when used for suspension flow, the PSM has so far mostly been applied to very simple geometries like circles in 2D or spheres in 3D. Although it is possible to construct more complex geometries by assembling several circles or spheres [22], additional work is necessary to make the PSM more attractive for moving, arbitrarily shaped boundaries. (Chen *et al.* [34] provide a short discussion of algorithms which can be used for more complex shapes.) Also, by sacrificing the treatment of the exact boundary shape, one cannot expect that the no-slip condition is accurately satisfied

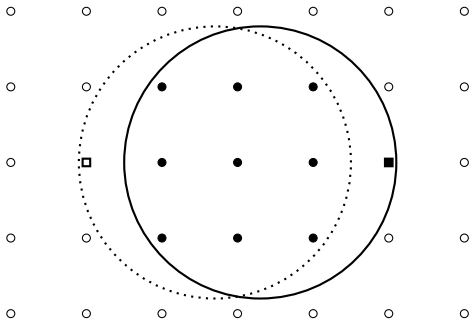


Fig. 11.7: Creation and destruction of fluid nodes. A particle is moving from its previous position (dotted circle) to its new position (solid circle). As a consequence, one fresh fluid node (open square) appears behind the particle and a fluid node is destroyed (solid square) at the front of the particle. Fluid and solid nodes are shown as open and solid symbols, respectively.

at the boundary. Furthermore, the PSM in its present form is not suitable for the simulation of deformable boundaries or thin shells with fluid on both sides (unlike interpolated bounce-back, for example).

The **partially saturated method (PSM)** is a node-based method. PSM is exactly mass-conserving and does not require the treatment of fresh fluid nodes. The disadvantage is the difficulty of finding correct values for the solid fraction near solid boundaries, which is effectively limiting this method to stationary geometries (where the solid fraction has to be computed only once) or to spherical particles.

11.2.4 Destruction and creation of fluid nodes

When boundaries move, it happens from time to time that lattice nodes cross the boundary, either from the fluid to the inside of the boundary or *vice versa*. If the interior of the boundary is not filled with a fluid, the former event requires the *destruction*, the latter the *creation* of a fluid site as shown in fig. 11.7. Newly created nodes are also called *fresh nodes*. This does not apply to Ladd's method (section 11.2.1) or the partially saturated method (section 11.2.3) where nodes are neither created nor destroyed. Generally the number of fluid and solid nodes is not conserved when boundaries move on the lattice.

Destruction is straightforward [15]: the state of the site is switched from 'fluid' to 'solid', and its momentum and angular momentum are transferred to the solid. For a destroyed fluid node at $\boldsymbol{x}_\mathrm{d}$ with density ρ and velocity $\boldsymbol{u}$, the particles receives a

momentum contribution $\rho \boldsymbol{u} \Delta x^3$ and an angular momentum contribution $(\boldsymbol{x}_\mathrm{d} - \boldsymbol{R}) \times (\rho \boldsymbol{u}) \Delta x^3$ where $\boldsymbol{R}$ is the particle's centre of mass. Finally, the fluid information of the site at $\boldsymbol{x}_\mathrm{d}$ is omitted.

The inverse process, creation of a fluid site, is more difficult because the density, velocity and even all populations are unknown at first. The simplest approach to initialise a fresh node at point $\boldsymbol{x}_\mathrm{f}$ and time t is to estimate the density as the average of the neighbouring fluid sites [15],

$$\rho_\mathrm{f} = \rho(\boldsymbol{x}_\mathrm{f}, t) = \frac{1}{N_\mathrm{f}} \sum_i \rho(\boldsymbol{x}_\mathrm{f} + \boldsymbol{c}_i \Delta t, t), \tag{11.14}$$

where the sum runs only over those N_f neighbouring sites which are fluid. The velocity $\boldsymbol{u}_\mathrm{f}$ of the fresh node is computed from the known boundary velocity of the obstacle at the same point *via* eq. (11.4). The populations f_i are then initialised with their equilibrium values $f_i^\mathrm{eq}(\rho_\mathrm{f}, \boldsymbol{u}_\mathrm{f})$. As additional step, the momentum and angular momentum of the fresh node have to be subtracted from the solid.

Although this approach is easy to implement, two disadvantages are obvious: the total mass in the system is generally not conserved, and the non-equilibrium contribution of the fresh node is missing, which can lead to distortions of the flow field.

Chen *et al.* [7] compared the above-mentioned approach and three other algorithms to initialise fresh nodes. One of those relies on extrapolation of populations from neighbouring fluid sites [26]. The other two approaches, both first described in [7], are based on the consistent initialisation [37] (see also section 5.4). From benchmark tests involving moving cylinders, the authors come to the conclusion that the consistent initialisation methods are usually more accurate than interpolation [15] or extrapolation [26].

The **destruction and creation of fluid nodes** is necessary when the standard or interpolated bounce-back method (or certain other boundary conditions) are used for moving boundaries. Boundary treatments without the need for this consideration are Ladd's method for suspended particles and the partially saturated method. It has been observed that the treatment of fresh nodes is crucial to reduce oscillations of the particle drag and creation of detrimental sound waves.

11.2.5 Wall shear stress

Several diseases of the circulatory system are assumed to be linked to pathological levels or changes of the shear stress at the arterial wall (see, *e.g.*, [38] and references therein). Two prominent examples are atherosclerosis or aneursym formation. In recent years, a growing number of scientists became interested in the LB modelling

of blood flow in realistic blood vessel geometries. Apart from finding and implementing reasonable boundary conditions, a key question is how the wall shear stress (WSS) can be computed and how accurate the obtained values are.

We will provide a brief review of the comparatively small number of publications in this field, but before that a few words about the WSS are necessary. The WSS is tightly connected to the momentum exchange at the boundary. Evaluating eq. (11.2) is not sufficient to get the WSS, though. The reason is that WSS is a local quantity and not an integrated property of the entire surface of the boundary.

In order to find the WSS, the boundary location *and* orientation have to be known at each point of interest. Assuming that $\boldsymbol{x}_\mathrm{b}$ is a point on the boundary, we denote $\hat{\boldsymbol{n}}$ the boundary normal vector at $\boldsymbol{x}_\mathrm{b}$ pointing inside the fluid domain. For a given fluid stress tensor $\boldsymbol{\sigma}$ at $\boldsymbol{x}_\mathrm{b}$, we first define the *traction* vector as $\boldsymbol{T} = \boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}}$. It is the force acting on an infinitesimal, oriented wall area element $\mathrm{d}\boldsymbol{A} = \mathrm{d}A\,\hat{\boldsymbol{n}}$. The WSS vector $\boldsymbol{\tau}$ is the tangential component of the traction vector, *i.e.* we have to subtract the normal component of the traction:

$$\boldsymbol{\tau} = \boldsymbol{T} - (\boldsymbol{T} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}, \quad \tau_\alpha = \sigma_{\alpha\beta}\hat{n}_\beta - (\sigma_{\beta\gamma}\hat{n}_\beta\hat{n}_\gamma)n_\alpha. \tag{11.15}$$

The subtracted normal component $(\boldsymbol{T} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$ contributes to the wall pressure. It is common to report only the magnitude of the WSS vector, simply called the WSS $\tau = |\boldsymbol{\tau}|$ (not to be confused with the BGK relaxation time).

We distinguish three typical situations:

1. The normal vector $\hat{\boldsymbol{n}}$ is known everywhere on the boundary, but the geometry is approximated by a staircase boundary (simple bounce-back).
2. The geometry is only known as a staircase, and no additional information about the normal vector $\hat{\boldsymbol{n}}$ is available.
3. A higher-order boundary conditions is used for the boundaries (*e.g.* IBB). We will not consider this case here.

An example for the first case is the discretisation of a known boundary geometry where a surface tessellation is converted to a staircase surface. Here we still have access to the original normal vectors, but the LB simulation is only aware of the staircase. The second case is important when voxel data is directly converted into a staircase geometry, without *a priori* knowledge of the boundary normals. This means that we first need to estimate $\hat{\boldsymbol{n}}$ from the known data before we can compute the WSS.

Stahl *et al.* [39] were the first authors to provide a careful investigation of the behaviour of the WSS in LB simulations with staircase geometries. They presented a scheme to obtain the unknown normal vectors $\hat{\boldsymbol{n}}$ from the flow field. This is straightforward in 2D where the no-penetration condition requires $\boldsymbol{u} \cdot \hat{\boldsymbol{n}} = 0$ for the fluid velocity $\boldsymbol{u}$ at the wall. From the known velocity $\boldsymbol{u}$ we can easily compute the unknown $\hat{\boldsymbol{n}}$ up to its sign. This is more complicated but possible in 3D, where additional information is required (see [39] for details). The authors then investigated the accuracy of the stress computation near the boundary in an inclined Poiseuille flow. They found strongly anisotropic behaviour: the error is minimum for walls aligned with one of the major lattice directions (*i.e.* when the wall is flat), but larger

errors for arbitrarily inclined walls. This error, however, decreases when the stress is evaluated farther away from the wall. Therefore the authors suggested to measure the fluid stress a few lattice sites away from the wall. All in all, it requires relatively high resolutions (several $10\Delta x$) to estimate the WSS reasonably well in a staircase geometry, which makes this method unfeasible when the average channel diameter is small.

Later, Matyka *et al.* [40] proposed a different scheme to obtain a better estimate of the unknown normal vectors $\hat{\boldsymbol{n}}$. Their idea was to compute a weighted average of the staircase information of the neighbouring lattice nodes. The advantage of their approach is that it is based on the geometry alone; it is independent of the flow field. Furthermore, the authors showed that the WSS error is dominated by the flow field error and not by an inaccurate approximation of the normal vector. The flow field error in turn is caused by the staircase approximation (modelling error), which can only be decreased by using a more accurate boundary condition for the LBM.

Pontrelli *et al.* [38] used a finite-volume LBM to compute the WSS in a small artery with a realistic endothelial wall profile. Unfortunately the authors did not provide a benchmark test of the WSS accuracy in their setup. It would be interesting to investigate whether the finite-volume LBM is able to mitigate the shortcomings of the regular lattice with staircase approximation.

Very recently, Kang and Dun [41] studied the accuracy of the WSS in inclined 2D Poiseuille and Womersley flows for BGK and MRT collision operators and for SBB and IBB at the walls. One of the basic results is that, in channels *aligned* with a major lattice axis, the WSS converges with a first-order rate upon grid refinement when it is evaluated at the fluid layer closest to the wall and with a second-order rate when the result is extrapolated to the wall location. This is no surprise since the distance of the last fluid layer and the wall itself converges to zero with first-order rate. The authors report similar results for BGK and MRT for their chosen parameter range. When the flow in an *inclined* channel is simulated, the choice of the boundary condition plays a significant role. IBB leads to errors which are about one order of magnitude smaller than for SBB. Moreover, MRT leads to slightly better results than BGK.

The choice of wall boundary condition critically affects the quality of **wall shear stress** estimates. The best results are obtained when a curved boundary condition is used as this increases the accuracy of the flow field and the fluid stress tensor in direct vicinity of the wall. Further research is necessary to develop improved boundary conditions for more accurate WSS computations in complex geometries.

11.3 Extrapolation methods

We will now present LB boundary methods which require extrapolations. A typical scenario is the extrapolation of fluid properties at virtual nodes within a solid body. These so-called *ghost* nodes then participate in collision and propagation like normal fluid nodes. This process produces those populations which stream out of the solid and would otherwise be unknown. After providing some definitions in section 11.3.1, we discuss three distinct classes of extrapolation-based boundary conditions: the Filippova-Hänel and Mei-Luo-Shyy methods (section 11.3.2), the Guo-Zheng-Shi method (section 11.3.3) and comparably novel image-based ghost methods (section 11.3.4). Some recommended articles are [42, 43, 44, 45, 46].

11.3.1 Definitions

In order to understand the motivation for the boundary conditions presented in this section, we first have to define certain terms and understand their implications.

- *Extrapolation* in the present context means that the known information of a quantity (*e.g.* velocity) within a geometrical region is used to approximate the quantity outside this region. The known region is typically the fluid region whereas the interior of a solid is unknown. For example, if we know the velocity u at points x and $x' = x + \Delta x$ (which may be inside the fluid) but not at $x'' = x' + \Delta x'$ (which may be inside the solid), we can still approximate it by *assuming* a linear behaviour and write

$$u(x'') = u(x') + \frac{u(x') - u(x)}{\Delta x}\Delta x'. \tag{11.16}$$

  Higher-order extrapolations require more known data points (usually $n+1$ points for extrapolation of order n). Extrapolations can often lead to instability (in particular when Δx in eq. (11.16) is small) and loss of accuracy.
- *Fictitious domain* methods are based on the idea that the solution of a problem in a given (usually complex) domain Ω can be simpler when instead a substitute problem in a larger (and simpler) domain Ω' with $\Omega \subset \Omega'$ is solved. This obviously means that information in the complement region $\Omega' \setminus \Omega$ has to be constructed. This usually involves extrapolations.
- *Ghost* methods are a special case of fictitious domain methods where virtual fluid nodes (ghost nodes) are created inside the solid region close to the boundary. Extrapolations of fluid and boundary properties are used to reconstruct the ghost nodes. These nodes then normally participate in collision and propagation in order to supply the boundary nodes with otherwise missing populations. Unfortunately these methods are sometimes denoted as *sharp-interface immersed-boundary methods*, although they hardly share any similarities with the immersed boundary method originally introduced by Peskin (section 11.4).

Revisiting the bounce-back boundary conditions presented in section 11.2, we can make the following comments:

- Standard and interpolated bounce-back do not involve any extrapolation or ghost nodes. Although one can implement both methods with the help of nodes which are on the solid side of the boundary, these nodes are only used for memory storage purposes and do not qualify these methods as ghost methods.
- The partially saturated method uses nodes which are inside the solid, but no extrapolations are required to create them. The reason is that the fluid is simply kept in the interior without the need to reconstruct it at every time step.

Therefore, neither of the bounce-back-based boundary conditions in section 11.2 is an extrapolation or ghost method.

We present three extrapolation-based methods in more detail: the Filippova-Hänel and Mei-Luo-Shyy methods, the Guo-Zheng-Shi method and image-based ghost methods. Apart from this, the method by Verschaeve and Müller [47] as an extension of [48] to curved boundaries is yet another alternative which we will, however, not discuss in detail. In short, the underlying idea of [47] is to have boundary nodes in both the fluid and solid regions and to interpolate and extrapolate fluid properties, respectively. The equilibrium distributions are reconstructed from the density and velocity, the non-equilibrium distributions from the velocity gradient.

11.3.2 Filippova-Hänel (FH) and Mei-Luo-Shyy (MLS) methods

In 1998, Filippova and Hänel [49] proposed the first LB boundary condition for curved geometries (FH method) using extrapolations. They assume the following situation: a population $f_i^\star(\boldsymbol{x}_\mathrm{f}, t)$ propagates towards a boundary located between a fluid node at $\boldsymbol{x}_\mathrm{f}$ and a solid node at $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{s} + \boldsymbol{c}_i\Delta t$. The boundary is located at $\boldsymbol{x}_\mathrm{b} = \boldsymbol{x}_\mathrm{f} + q\boldsymbol{c}_i\Delta t$ as illustrated in fig. 11.8. The question is how to find the missing population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t)$.

Original method by Filippova and Hänel (FH)

Filippova and Hänel [49] suggested the equation

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t) = (1 - \chi) f_i^\star(\boldsymbol{x}_\mathrm{f}, t) + \chi f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t) - 2w_i \frac{\boldsymbol{u}_\mathrm{b} \cdot \boldsymbol{c}_i}{c_\mathrm{s}^2} \tag{11.17}$$

where $\boldsymbol{u}_\mathrm{b}$ is the boundary velocity (*i.e.* the velocity of the boundary at the intersection point $\boldsymbol{x}_\mathrm{b}$, *cf.* fig. 11.8) and χ a weighting factor (with the dimensionless BGK relaxation time τ):

$$\chi = \begin{cases} \frac{1}{\tau}\frac{2q-1}{1-\frac{1}{\tau}} & q < \frac{1}{2} \\ \frac{1}{\tau}(2q-1) & q \geq \frac{1}{2} \end{cases}. \tag{11.18}$$

Exercise 11.3. Show that eq. (11.17) reduces to simple bounce-back for $q = \frac{1}{2}$.

Eq. (11.17) is essentially an interpolation of populations at $\boldsymbol{x}_\mathrm{f}$ and $\boldsymbol{x}_\mathrm{s}$, but we still have to investigate the shape of the required equilibrium term $f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t)$. Filippova and Hänel [49] construct the "equilibrium distribution in the rigid nodes" from

$$f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t) = w_i \left(\frac{p(\boldsymbol{x}_\mathrm{f}, t)}{c_\mathrm{s}^2} + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}_\mathrm{s}}{c_\mathrm{s}^2} + \frac{(\boldsymbol{c}_i \cdot \boldsymbol{u}_\mathrm{f})^2}{2c_\mathrm{s}^4} - \frac{\boldsymbol{u}_\mathrm{f} \cdot \boldsymbol{u}_\mathrm{f}}{2c_\mathrm{s}^2} \right). \tag{11.19}$$

where we have used the abbreviations $\boldsymbol{u}_\mathrm{f} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{f}, t)$ and $\boldsymbol{u}_\mathrm{s} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{s}, t)$. This is nearly the standard incompressible equilibrium evaluated at $\boldsymbol{x}_f$, with the only exception that the fluid velocity $\boldsymbol{u}_\mathrm{f}$ is replaced by the solid node velocity $\boldsymbol{u}_\mathrm{s}$ in the linear term. The authors suggested

$$\boldsymbol{u}_\mathrm{s} = \begin{cases} \boldsymbol{u}_\mathrm{f} & q < \frac{1}{2} \\ \frac{q-1}{q}\boldsymbol{u}_\mathrm{f} + \frac{1}{q}\boldsymbol{u}_\mathrm{b} & q \geq \frac{1}{2} \end{cases} \tag{11.20}$$

to find the missing velocity at $\boldsymbol{x}_\mathrm{s}$.

This deserves a few comments:

- For $q \geq \frac{1}{2}$, the solid node velocity is obtained by *extrapolating* the velocity at $\boldsymbol{x}_\mathrm{s}$ from $\boldsymbol{x}_\mathrm{f}$ and $\boldsymbol{x}_\mathrm{b}$.
- Since the extrapolation would lead to unstable results for $q \to 0$, the authors fall back to $\boldsymbol{u}_\mathrm{s} = \boldsymbol{u}_\mathrm{f}$ for $q < \frac{1}{2}$.
- The choice of the incompressible equilibrium also explains why the fluid density does not appear in the momentum exchange term on the right-hand-side of eq. (11.17): in the incompressible method the density is constant and typically set to unity. TODO (ALL): Is the method still applicable for the normal compressible LBM?

Although the FH method reduces to simple bounce-back for $q = \frac{1}{2}$, it is conceptually different from interpolated bounce-back (IBB, section 11.2.2) which also

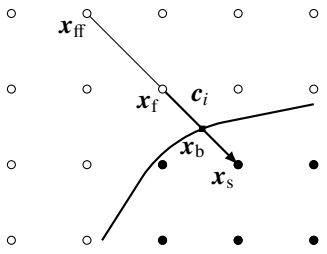


Fig. 11.8: Filippova and Hänel boundary condition. A link $\boldsymbol{c}_i$ is cut by a curved boundary at $\boldsymbol{x}_\mathrm{b}$ (solid square). Fluid and solid nodes are shown as open and solid circles, respectively. Information about the velocity at the solid node $\boldsymbol{x}_\mathrm{s}$ is required to find the post-streaming population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f})$. The original method [49] requires the velocity at $\boldsymbol{x}_\mathrm{f}$ only, while the improved method [42] uses the velocity at $\boldsymbol{x}_\mathrm{ff}$ as well.

reduces to simple bounce-back for $q = \frac{1}{2}$. For any q-value, only one fluid node is required in eq. (11.17), which makes the FH method more local than IBB. The FH method requires an extrapolation for $q \geq \frac{1}{2}$, IBB does not.

Improvements by Mei, Luo and Shyy (MLS)

The FH method has the major disadvantage that the weight χ diverges for $\tau \to 1$ and $q < \frac{1}{2}$, which leads to instability. Mei *et al.* [42] therefore analysed the FH method and its stability properties in detail and proposed an improved version (MLS method). The starting point for the improvement is to realise that there are different ways to construct the term $f_i^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t)$. The authors proposed new expressions for $q < \frac{1}{2}$:

$$\boldsymbol{u}_\mathrm{s} = \boldsymbol{u}_\mathrm{ff}, \quad \chi = \frac{2q-1}{\tau - 2}, \tag{11.21}$$

where $\boldsymbol{u}_\mathrm{ff} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{ff}, t)$ and $\boldsymbol{x}_\mathrm{ff} = \boldsymbol{x}_\mathrm{f} - \boldsymbol{c}_i \Delta t$ is the location of a second fluid node (fig. 11.8). The expressions for $q \geq \frac{1}{2}$ remain untouched.

Mei *et al.* [42] showed that this modification indeed improves the stability of the original FH method, but they are also sacrificing its locality as a second fluid node is required. The authors further mention that the above expressions are only strictly valid for stationary flows. They therefore suggested a higher-order extrapolation at $\boldsymbol{u}_\mathrm{s}$ for transient flows.

It is important to note that most follow-up works in the literature employ the improved [42] rather than the original [49] implementation. In the following we summarise some notable progress:

- Mei *et al.* [50] were the first to perform a thorough comparative evaluation of the momentum exchange algorithm (MEA) and stress integration to obtain drag and lift coefficients at stationary curved boundaries. They found that the stress integration is much more demanding in terms of implementation effort and computing time while the MEA is still relatively accurate. Mei *et al.* therefore recommend to use the MEA.
- Like other interpolation- and extrapolation-based approaches, the FH and MLS methods suffer from a violation of mass conservation. Therefore, Bao *et al.* [51] analysed the mechanism responsible for the mass leakage in those boundary treatments and presented an improved mass-conserving method.
- Wen *et al.* [43] extended the MEA [50] to moving boundaries.

11.3.3 Guo-Zheng-Shi (GZS) method

Guo *et al.* [44] proposed yet another extrapolation-based LB boundary condition for curved boundaries (GZS). The problem is the same as in section 11.3.2 and fig. 11.8. In particular, the cut link $\boldsymbol{c}_i$ points into the solid.[2]

The question is how to find $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t)$. The GZS method uses a fictitious fluid node at $\boldsymbol{x}_\mathrm{s}$ which is assigned an equilibrium value

$$f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\rho_\mathrm{s}, \boldsymbol{u}_\mathrm{s}) \tag{11.22}$$

where $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ is the standard incompressible equilibrium. Note that the only fictitious nodes are those solid nodes which are directly connected to a fluid node by a lattice vector $\boldsymbol{c}_i$.

The authors approximate the density at the solid site by its neighbour value: $\rho_\mathrm{s} = \rho(\boldsymbol{x}_\mathrm{s}, t) = \rho(\boldsymbol{x}_\mathrm{f}, t)$. For the velocity, they suggested

$$\boldsymbol{u}_\mathrm{s} = \begin{cases} q\boldsymbol{u}^{(1)} + (1-q)\boldsymbol{u}^{(2)} & q < \frac{3}{4} \\ \boldsymbol{u}^{(1)} & q \geq \frac{3}{4} \end{cases} \tag{11.23}$$

where $\boldsymbol{u}^{(1)}$ and $\boldsymbol{u}^{(2)}$ are extrapolations using the first and second fluid node at $\boldsymbol{x}_\mathrm{f}$ and $\boldsymbol{x}_\mathrm{ff}$, respectively:

$$\begin{aligned} \boldsymbol{u}^{(1)} &= \frac{(q-1)\boldsymbol{u}_\mathrm{f} + \boldsymbol{u}_\mathrm{b}}{q}, \\ \boldsymbol{u}^{(2)} &= \frac{(q-1)\boldsymbol{u}_\mathrm{ff} + 2\boldsymbol{u}_\mathrm{b}}{1+q}. \end{aligned} \tag{11.24}$$

When q is large enough, an extrapolation from the closest fluid node at $\boldsymbol{x}_\mathrm{f}$ is sufficiently stable, but for smaller q-values an extrapolation from the second fluid node at $\boldsymbol{x}_\mathrm{ff}$ becomes necessary.

Now, apart from the equilibrium, the GZS method also involves the non-equilibrium populations at the solid node. The authors proposed the extrapolation

$$f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t) = \begin{cases} q f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{f}, t) + (1-q) f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{ff}, t) & q < \frac{3}{4} \\ f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{f}, t) & q \geq \frac{3}{4} \end{cases}. \tag{11.25}$$

The GZS algorithm includes the following steps:

1. Find q for a cut link connecting a fluid node $\boldsymbol{x}_\mathrm{f}$ and a solid node $\boldsymbol{x}_\mathrm{s}$.
2. Reconstruct the populations of the fictitious nodes by

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) + f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t), \tag{11.26}$$

   where the equilibrium and non-equilibrium parts are computed from eq. (11.22) and eq. (11.25).

[2] Guo *et al.* [44] defined $\boldsymbol{c}_i$ exactly the other way around.

3. Collide on all fluid nodes *and* fictitious nodes.[3]
4. Stream populations from all fluid nodes *and* fictitious nodes to their fluid neighbours. In particular, $f_{\bar{i}}^{\star}$ streams from the fictitious to the fluid node and provides the missing value for $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t + \Delta t)$.
5. Go back to step 1.

According to Guo *et al.* [44], the present method has advantages over the methods in section 11.3.2. First, while the FH and MLS methods assume a slowly varying flow field, the GZS method only requires a low Mach number flow which can be unsteady. Secondly, the GZS scheme is more stable than the MLS approach.

We would also like to mention that Guo *et al.* [44] view FH and MLS as improved bounce-back methods. Although that statement is certainly not wrong (the functional form for the missing population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{f}, t+\Delta t)$ is similar to the standard and interpolated bounce-back expressions), the FH, MLS and GZS methods all require extrapolations and are ghost-like methods. They are therefore conceptually different from the bounce-back methods presented in section 11.2 which are all extrapolation-free.

11.3.4 Image-based ghost methods

Only in 2012, Tiwari and Vanka [45] developed a ghost-fluid boundary condition for the LBM which is based on the so-called *image* method. The idea of their boundary condition is illustrated in fig. 11.9.

The algorithm consists of the following steps:

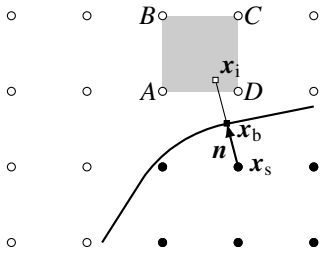


Fig. 11.9: Image-based ghost method. For each ghost node $\boldsymbol{x}_\mathrm{s}$, the closest boundary point $\boldsymbol{x}_\mathrm{b}$ is computed. The corresponding image point $\boldsymbol{x}_\mathrm{i}$ in the fluid (open circles) is constructed along the normal vector $\boldsymbol{n}$. Fluid properties at the image point are obtained from an interpolation in the gray region (fluid nodes A, B, C, D). A different interpolation is required if not all interpolation support points are located within the fluid.

[3] In the original paper [44], the fictitious populations are already constructed in their post-collision state $f_{\bar{i}}^{\star}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) + (1 - \frac{1}{\tau}) f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t)$. In this case, collision on fictitious nodes is of course not additionally performed.

1. Identify all required ghost nodes $\boldsymbol{x}_\mathrm{s}$. Those are all solid nodes which are connected to at least one fluid node along a lattice vector $\boldsymbol{c}_i$.
2. For each ghost node $\boldsymbol{x}_\mathrm{s}$ find the closest point $\boldsymbol{x}_\mathrm{b}$ on the boundary. We define $\boldsymbol{n} = \boldsymbol{x}_\mathrm{b} - \boldsymbol{x}_\mathrm{s}$ as the outward-pointing normal vector at the boundary. Note that $|\boldsymbol{n}|$ is the distance of the ghost node from the boundary and $\boldsymbol{n}$ is generally not aligned with any of the lattice vectors $\boldsymbol{c}_i$.
3. The next step is to find the *image point* $\boldsymbol{x}_\mathrm{i}$ in the fluid:

$$\boldsymbol{x}_\mathrm{i} = \boldsymbol{x}_\mathrm{s} + 2\boldsymbol{n} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{n}. \tag{11.27}$$

   For a stationary boundary, steps 1–3 have to be performed only once.
4. Interpolate the required fluid properties (velocity and density) at the image point $\boldsymbol{x}_\mathrm{i}$ to obtain $\boldsymbol{u}_\mathrm{i}$ and ρ_i. The interpolation process is somewhat tedious as it depends on whether interpolation support points are located in the fluid or on the boundary. We refer to [45] for a detailed discussion.
5. Extrapolate velocity and density along the normal $\boldsymbol{n}$ to the ghost node. Tiwari and Vanka [45] used

$$\boldsymbol{u}_\mathrm{s} = 2\boldsymbol{u}_\mathrm{b} - \boldsymbol{u}_\mathrm{i}, \quad \rho_\mathrm{s} = \rho_\mathrm{i}. \tag{11.28}$$

   The first equation refers to a linear extrapolation of the velocity, the second to a zero density gradient at the boundary.
6. Compute the equilibrium distributions at the ghost nodes from $f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t) = f_i^\mathrm{eq}(\rho_\mathrm{s}, \boldsymbol{u}_\mathrm{s})$.
7. The non-equilibrium distributions $f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{s}, t)$ are obtained like the fluid density: interpolate them at $\boldsymbol{x}_\mathrm{i}$ first, then apply $f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{s}, t) = f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{i}, t)$.
8. Combine the equilibrium and non-equilibrium distributions at the ghost nodes and perform the propagation step, followed by the collision step.
9. Go back to step 1.

This algorithm deserves a few remarks:

- According to [45], the extrapolated values of density, velocity and non-equilibrium distributions are post-collision rather than pre-collision. This is unusual since the moments (density, velocity, stress) are normally computed after the previous streaming and before the next collision step.
- The boundary condition is based on the hydrodynamic fields rather than the populations. This allows to implement Neumann boundary conditions. The authors for example demonstrated the applicability of their method for inlet and outlet boundary conditions [45].
- Extrapolation along normal vectors as in step 5 avoids typical stability issues encountered with other extrapolation methods. TODO (ALL): Citation?
- Although being trivial for circular or spherical boundary segments, finding the image point can be tedious for complex boundary shapes. Also the interpolation at the image points is complicated if not all interpolation support points are within the fluid domain. The application of this boundary condition to moving boundaries of complex shape, in particular in 3D, is therefore difficult and expensive. Tiwari and Vanka [45] simulated only circular boundaries in 2D.

- The assumption of a zero density gradient across the boundary is a gross oversimplification. For example, it fails when a force density along the extrapolation direction exists which is balanced by a pressure gradient [46]. Since errors in the pressure gradient are of higher order, the velocity profile may still be second-order accurate, though. A similar objection can be made for the non-equilibrium distributions. At least a linear extrapolation for the density and the non-equilibrium distributions are required to accurately capture second-order flows like the Poiseuille flow.

Several extensions and improvements of the algorithm have been proposed in the meantime:

- Khazaeli *et al.* [52] followed a similar route as Tiwari and Vanka [45] to impose higher-order boundary conditions for coupled fluid-heat problems in the two-population LBM.
- Mohammadipoor *et al.* [46] followed the same line as [45] and extended the approach of Zou and He [53] to curved boundaries.
- Pellerin *et al.* [54] proposed an image-based method that relies only on equilibrium distributions.

There exist several **extrapolation-based** boundary conditions for the LBM. These methods are conceptually more difficult than bounce-back-based methods. A common algorithmic complication all these methods share with the interpolated bounce-back method is the detection of boundary points (either on cut links or closest points to ghost nodes). In practice, these methods are quite unhandy for moving boundaries of complicated shape although they are promising candidates for highly accurate boundary conditions when properly applied. More research is required to make extrapolation-based method more attractive for moving objects with non-trivial shape.

11.4 Immersed boundary methods

The immersed boundary method (IBM) [55, 56, 57] is older than LBM, but the combination of both was not suggested before 2004 [58] (section 11.4.1). The basic idea of the IBM is to approximate a boundary by a set of off-lattice marker points that affect the fluid only *via* a force field. An interpolation stencil is introduced to couple the lattice and the marker points (section 11.4.2). This allows a relatively simple implementation of complex boundaries. There are several IBM variants, for example explicit (section 11.4.3) or direct-forcing (section 11.4.4) for rigid boundaries and explicit IBM for deformable boundaries (section 11.4.5). We also mention a series of other related boundary conditions which are less commonly used (section 11.4.6). We recommend reading [59, 60] for introductions and investigations of the IBM in conjunction with the LBM.

11.4.1 Introduction

Boundary conditions in the LBM are usually treated on the population level, *i.e.* the populations f_i are manipulated or constructed in such a way that the desired values for pressure and velocity (or their derivatives) are obtained at the boundary. This applies to all boundary conditions discussed in chapter 5 and in the present chapter up to this point.

There is, however, a completely different way to enforce boundary conditions which was available long before anybody knew of the LBM. In 1972, Peskin proposed the *immersed boundary method* (IBM) in his dissertation [55], followed by an article in 1977 [56]. Peskin's idea was to use the force density $\boldsymbol{F}(\boldsymbol{x},t)$ in the Navier-Stokes equation to mimic a boundary condition. To this end, $\boldsymbol{F}(\boldsymbol{x},t)$ has to be computed such that the fluid behaves *as if* there was a boundary with desired properties (*e.g.* no-slip). When correctly applied, this approach can be used to recover immersed rigid or deformable objects with nearly arbitrary shape. Since the boundary condition exists only on the Navier-Stokes level (*via* the force density $\boldsymbol{F}(\boldsymbol{x},t)$), IBM is not aware of the populations f_i.

The IBM provides a number of advantages. The main advantage is its front-tracking character, *i.e.* the shape of the boundary is directly known and does not have to be reconstructed (as in phase-field or level-set approaches). Neither do intersection points have to be computed (as required for nearly all boundary conditions presented in this chapter so far). The IBM can be combined with any Navier-Stokes solver which supports external forcing, such as the LBM. The IBM is relatively simple to implement and, if done so properly, its numerical overhead is small. Moving and deformable boundaries can be realised without remeshing. It has to be noted that fluid exists on both sides of an IBM surface. In particular, closed surfaces are filled with fluid.

The original IBM does not take any consideration of the kinetic origin of the LBM as it only operates on the Navier-Stokes level. Still, the combination of the IBM and the LBM, also called immersed-boundary-lattice-Boltzmann method (IB-LBM), first proposed by Feng and Michaelides [58], has become a popular application. It therefore deserves a somewhat thorough introduction in this book, together with some recent developments and related approaches. We cannot provide an exhaustive coverage of the IBM in general, though. Readers who are interested in the IBM independently of the LBM should read the seminal paper by Peskin [57] and the review by Mittal and Iaccarino [61].

There is some dissent in the literature what "immersed boundary method" actually means and how it is defined. Some people use it for nearly all methods where a boundary is immersed in a fluid, including, for example, fictitious domain methods (section 11.3). Here, we define those methods as immersed boundary methods which involve, on the one hand, an Eulerian grid and Lagrangian markers and, on the other hand, some kind of velocity interpolation and force spreading as devised by Peskin [57], but there is no clear distinction between the IBM and related methods.

In the remainder of this section we will focus on the IBM combined with LBM using the BGK collision operator. However, several authors recently pointed out that

the MRT or TRT collision operators can bring additional advantages by reducing undesired velocity slip at the immersed boundaries [62, 63]. We will not discuss those extensions further.

11.4.2 Mathematical basis

We will now review the original IBM, discuss its mathematical properties and show its basic numerical algorithm.

Eulerian and Lagrangian systems

Mathematically, the basis of the IBM is an *Eulerian* and a *Lagrangian* system. The former is represented by a fixed regular grid on which the fluid lives and the Navier-Stokes equations are solved. The latter is an ensemble of marker points $\{\boldsymbol{r}_j\}$. They can be (nearly) arbitrarily distributed in space, as long as they are sufficiently dense (see below). These markers represent discrete surface points of the boundary and are generally allowed to move: $\boldsymbol{r}_j = \boldsymbol{r}_j(t)$. We therefore have to distinguish between two node systems (fig. 11.10) with the following properties:

1. The Eulerian grid defined by the LBM lattice nodes (coordinates designated by $\boldsymbol{x}$) is regular and stationary.
2. The immersed boundary marker points $\boldsymbol{r}_j(t)$ are Lagrangian nodes. They are not bound to the Eulerian grid and can move in space.

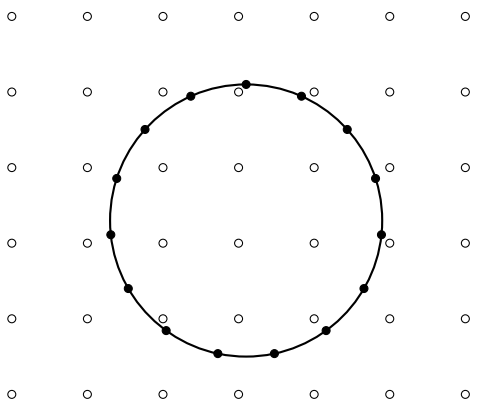


Fig. 11.10: Cylinder with boundary markers arbitrarily positioned in the regular fluid domain. The Eulerian mesh (fluid nodes, open circles) and the Lagrangian mesh (boundary, solid nodes) are independent. No intersections of lattice links with the boundary have to be computed.

If the boundary is rigid, one would ideally fix relative distances such that $|\boldsymbol{r}_{jk}(t)| = |\boldsymbol{r}_j(t) - \boldsymbol{r}_k(t)| = \text{const}$. This is often not achievable, and a somewhat softened condition $|\boldsymbol{r}_{jk}(t)| \approx \text{const}$ is used instead. For deformable boundaries, a relative marker motion is actually desired.

It may or may not be necessary to connect neighbouring markers. Most implementations of rigid boundary conditions do not require connected markers, while all deformable algorithms require some kind of surface tessellation which involves defining the markers *and* their connectivity (surface mesh). This mesh is called *non-conforming* as it does not have to be aligned with the lattice of the LBM. The main advantage of the IBM is that the complex shape of the boundary is not related to the lattice structure and no intersection points have to be computed. This makes the IBM particularly useful for deformable boundaries (section 11.4.5).

The decomposition of the geometry into two coordinate systems brings up the important question how to couple the dynamics of the boundary and the fluid. We need a bi-directional coupling where the fluid has to know about the presence of the boundary and *vice versa*. It is therefore required to communicate some information between both node systems through *velocity interpolation* and *force spreading*.

Continuous governing equations

We start with a fully continuous description and later turn our attention to the discretised version. In the following we assume the validity of the no-slip boundary condition, which is the first key idea of the IBM. It implies that each point of the surface $\boldsymbol{r}(t)$ and the ambient fluid at position $\boldsymbol{r}$ have to move with the same velocity:

$$\dot{\boldsymbol{r}}(t) = \boldsymbol{u}(\boldsymbol{r}(t), t). \tag{11.29}$$

The time dependence on the right-hand side of eq. (11.29) is, on the one hand, caused by the variation of $\boldsymbol{u}$ itself and, on the other hand, by the boundary moving and therefore seeing different parts of the flow field. We can rewrite eq. (11.29) as

$$\dot{\boldsymbol{r}}(t) = \int \mathrm{d}^3x\, \boldsymbol{u}(\boldsymbol{x}, t)\delta(\boldsymbol{x} - \boldsymbol{r}(t)) \tag{11.30}$$

where $\delta(\boldsymbol{x}-\boldsymbol{r}(t))$ is Dirac's delta distribution. Eq. (11.30) is the first of two governing equations of the as yet continuous IBM. We will later see that the discretised version of eq. (11.30) requires *velocity interpolation*. Note that we write all equations for 3D applications, but everything said (unless otherwise stated) can be directly applied in 2D as well.

The second governing equation describes the momentum exchange between the boundary and the fluid. In the IBM picture, we are interested in the force the boundary surface exerts on the nearby fluid, rather than the other way around. Let us assume we know the force density (per area) $\boldsymbol{F}_A(\boldsymbol{r}(t), t)$ everywhere on the bound-

ary surface. Therefore, $\boldsymbol{F}_A \, \mathrm{d}^2 r$ is the force acting on a small area element $\mathrm{d}^2 r$.[4] The force density (per volume) that the fluid feels due to the presence of the immersed boundary can then be written as

$$\boldsymbol{F}(\boldsymbol{x}, t) = \int \mathrm{d}^2 r \, \boldsymbol{F}_A(\boldsymbol{r}(t), t) \delta(\boldsymbol{x} - \boldsymbol{r}(t)). \tag{11.31}$$

The delta distribution is the same as in eq. (11.30). Eq. (11.31) essentially means that the Lagrangian boundary force is spread to the Eulerian fluid. Therefore this equation is called *force spreading*.

Note that $\boldsymbol{F}(\boldsymbol{x}, t)$ is singular. Since the delta distribution $\delta(\boldsymbol{x} - \boldsymbol{r}(t))$ is 3D, but the integration is only 2D along the boundary, $\boldsymbol{F}(\boldsymbol{x}, t)$ is singular when crossing the boundary in normal direction. This marks the defining difference between velocity interpolation (which is non-singular) and force spreading.

Eq. (11.30) and eq. (11.31) are the basic IBM equations in their continuous form. We will now show their discretised versions which can be used in computer simulations.

Discretised governing equations

Since the velocity field is only known at discrete lattice sites, the integral in eq. (11.30) cannot be exactly computed in a lattice-based simulation. The same holds for eq. (11.31). Instead, both integrals have to be replaced by sums with a suitably chosen discretisation of the delta distribution.

Peskin [57] provided a full derivation of a general set of equations for the IBM. We will restrict ourselves, for the sake of brevity and clarity, to the final set of equations based on the assumption that the fluid in the entire volume is homogeneous, in particular its density and viscosity. We further assume that the markers are massless, which means that the boundary has the same density as the surrounding fluid.

The **discretised IBM equations** read

$$\dot{\boldsymbol{r}}_j(t) = \sum_{\boldsymbol{x}} \Delta x^3 \boldsymbol{u}(\boldsymbol{x}, t) \Delta(\boldsymbol{r}_j(t), \boldsymbol{x}) \tag{11.32}$$

and

$$\boldsymbol{F}(\boldsymbol{x}, t) = \sum_j \boldsymbol{F}_j(t) \Delta(\boldsymbol{r}_j(t), \boldsymbol{x}). \tag{11.33}$$

The fluid is discretised as an Eulerian lattice with coordinates $\boldsymbol{x}$, the boundary is approximated by an ensemble of markers at $\boldsymbol{r}_j(t)$. Here, $\boldsymbol{u}$ and $\boldsymbol{F}$ are velocity and force density (per volume) on the lattice, $\dot{\boldsymbol{r}}_j$ and $\boldsymbol{F}_j$ are the velocity of and the total force acting on the markers. Velocity interpolation in eq. (11.32) and

[4] Remember that the boundary in 3D is a 2D surface.

force spreading in eq. (11.33) are the central IBM equations. Both require an appropriate kernel function (or stencil) Δ, as discussed below.

It is important to realise that $\boldsymbol{F}_j(t)$ is the total force (not force density) acting on node j at position $\boldsymbol{r}_j(t)$. Apart from the no-slip condition discussed above, it is one of the key ideas of the IBM that the force $\boldsymbol{F}_j(t)$ is first computed in the Lagrangian system and then spread to the lattice. This brings up the central question how $\boldsymbol{F}_j(t)$ can be found in the first place. In fact, this depends strongly on the chosen kind of the IBM. We will discuss this problem in the upcoming sections. Let us for now simply assume that all forces $\boldsymbol{F}_j(t)$ are known at each time step.

We further emphasise that $\boldsymbol{F}(\boldsymbol{x}, t)$ is the only mechanism through which the fluid is aware of the presence of the boundary; there is otherwise no direct boundary condition for the fluid. Once we know $\boldsymbol{F}(\boldsymbol{x}, t)$, we can use one of the forcing schemes described in chapter 6 to update the fluid. To the LBM, the IBM force density is not at all different from gravity (although gravity is usually homogeneous and constant).

Kernel functions

The function $\Delta(\boldsymbol{r}_j, \boldsymbol{x})$ is a suitably discretised version of the Dirac delta distribution and another key ingredient of any IBM. It is in most cases simplified by assuming $\Delta(\boldsymbol{r}_j, \boldsymbol{x}) = \Delta(\boldsymbol{r}_j - \boldsymbol{x})$, *i.e.* it is only a function of the distance vector $\boldsymbol{r}_j - \boldsymbol{x}$ rather than a more general function of $\boldsymbol{r}_j$ and $\boldsymbol{x}$ (see [64] for a kernel without this simplification). It is not directly obvious which functions $\Delta(\boldsymbol{r}_j - \boldsymbol{x})$ qualify as valid interpolation and spreading kernels. While Peskin [57] explains the procedure to find suitable discretisations in detail and an overview of interpolation function can be found in [65, 60], we will only provide the basic ideas and final results.

The fundamental claims and restrictions are:

- Interpolation and spreading should be short-ranged. This is required to reduce computational overhead by making the number of summands in eq. (11.32) and eq. (11.33) as small as possible.
- Momentum and angular momentum have to be identical when evaluated either in the Eulerian or the Lagrangian system (same speed and rotation in both systems).
- Lattice artifacts ("bumpiness" of the interpolation when boundaries move) should be suppressed as much as possible.
- The kernel has to be normalised: $\sum_{\boldsymbol{x}} \Delta x^3 \Delta(\boldsymbol{x}) = 1$.

It is convenient to factorise the kernel function as $\Delta(\boldsymbol{x}) = \phi(x)\phi(y)/\Delta x^2$ in 2D and $\Delta(\boldsymbol{x}) = \phi(x)\phi(y)\phi(z)/\Delta x^3$ in 3D, *i.e.* each major coordinate axis contributes independently. This is not essential but simplifies the procedure. Peskin [57] derived a series of stencils which are also shown in fig. 11.11. Those kernels read

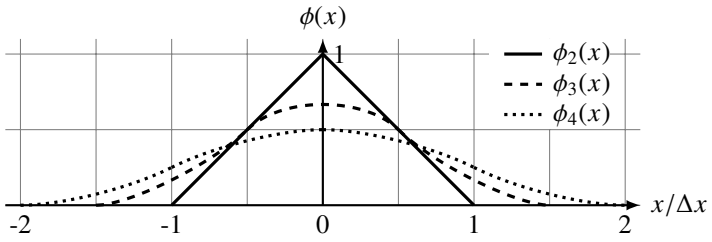


Fig. 11.11: IBM interpolation stencils ϕ_2, ϕ_3, and ϕ_4. The total kernel range is two, three and four lattice sites, respectively.

$$\phi_2(x) = \begin{cases} 1 - |x| & (0 \le |x| \le \Delta x) \\ 0 & (\Delta x \le |x|) \end{cases}, \tag{11.34}$$

$$\phi_3(x) = \begin{cases} \frac{1}{3}\left(1 + \sqrt{1 - 3x^2}\right) & 0 \le |x| \le \frac{1}{2}\Delta x \\ \frac{1}{6}\left(5 - 3|x| - \sqrt{-2 + 6|x| - 3x^2}\right) & \frac{1}{2}\Delta x \le |x| \le \frac{3}{2}\Delta x \\ 0 & \frac{3}{2}\Delta x \le |x| \end{cases}, \tag{11.35}$$

$$\phi_4(x) = \begin{cases} \frac{1}{8}\left(3 - 2|x| + \sqrt{1 + 4|x| - 4x^2}\right) & 0 \le |x| \le \Delta x \\ \frac{1}{8}\left(5 - 2|x| - \sqrt{-7 + 12|x| - 4x^2}\right) & \Delta x \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}. \tag{11.36}$$

The integer index denotes the number of lattice nodes required for interpolation and spreading along each coordinate axis (fig. 11.11). Therefore, the stencils require 2^d, 3^d and 4^d lattice sites in d dimensions, respectively.

ϕ_4 fulfills all of Peskin's requirements, but it also leads to a diffuse boundary since the interpolation range is rather large. ϕ_3 also fulfills all requirements, but it is less smooth. ϕ_2 is most efficient in terms of computing time and leads to the sharpest boundaries, but the lattice structure is not well hidden, *i.e.* the resulting flow field is generally more bumpy. Sometimes, $\phi_4(x)$ is replaced by another stencil which has nearly the same shape but does not exactly satisfy all of the requirements mentioned above:

$$\phi_4'(x) = \begin{cases} \frac{1}{4}\left(1 + \cos(\frac{\pi x}{2})\right) & 0 \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}. \tag{11.37}$$

General IB-LBM algorithm

Without specifying yet how the forces $\boldsymbol{F}_j$ acting on the boundary nodes are obtained, we can still jot down a simple IB-LBM algorithm. It consists of the following substeps:

1. Compute the Lagrangian forces $\boldsymbol{F}_j(t)$ from the current boundary configuration $\{\boldsymbol{r}_j(t)\}$. This is a model-dependent step which still remains to be discussed.
2. Spread the Lagrangian forces $\boldsymbol{F}_j(t)$ to the lattice *via* eq. (11.33) to obtain the Eulerian force density $\boldsymbol{F}(\boldsymbol{x}, t)$. See also fig. 11.12.
3. Compute the uncorrected (pre-collision) velocity $\boldsymbol{u}(\boldsymbol{x}, t)$ from $\boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i / \rho$.
4. Perform the LB algorithm (computing equilibrium distributions, collision and propagation) with forcing (chapter 6), using $\boldsymbol{u}(\boldsymbol{x}, t)$ and $\boldsymbol{F}(\boldsymbol{x}, t)$ as input. If other forces, such as gravity, are present, the total force is the sum of all these contributions. Note that the choice of an accurate forcing scheme (e.g. Guo *et al.* [66]) is important. This is often ignored in the literature.
5. As we know from chapter 6, the physical fluid velocity during the time step is given by the first moment of the populations and a force correction:

$$\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}, t) = \boldsymbol{u}(\boldsymbol{x}, t) + \frac{\boldsymbol{F}(\boldsymbol{x}, t)\Delta t}{2\rho(\boldsymbol{x}, t)}. \tag{11.38}$$

   Leaving the force correction out can lead to significant stability (and accuracy) problems.
6. Interpolate the fluid velocity $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}, t)$ at the Lagrangian node positions *via* eq. (11.32) to obtain $\dot{\boldsymbol{r}}_j(t)$. See also fig. 11.12.
7. Advect the boundary nodes (usually by the explicit forward Euler method) to find the new boundary configuration:

$$\boldsymbol{r}_j(t + \Delta t) = \boldsymbol{r}_j(t) + \dot{\boldsymbol{r}}_j(t)\Delta t. \tag{11.39}$$

   There exist different explicit time integration schemes [67, 68], though.
8. Go back to step 1 for the next time step.

Note that the time steps for the LBM and the marker position update are identical, *i.e.* in the standard IB-LBM there can only be one marker update per LB time step.

Not all IBM flavours follow this algorithm. There are several approaches (and algorithms) to deal with rigid boundaries. We will get back to those later.

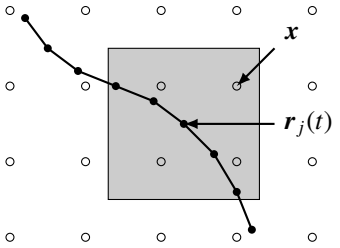


Fig. 11.12: Interpolation and spreading. The lattice velocity is interpolated at $\boldsymbol{r}_j(t)$. For this operation, all lattice nodes within the grey region are required (here ϕ_2 is used). The force density at a given lattice node $\boldsymbol{x}$ is the sum of all contributions from those nodes $\boldsymbol{r}_j(t)$ whose interpolation box covers $\boldsymbol{x}$.

Once the discretised kernel functions $\Delta(\boldsymbol{x})$ have been implemented, the rules for computing the forces $\boldsymbol{F}_j$ have been defined and the initial boundary node locations $\boldsymbol{r}_j(t = 0)$ are known, the simulation can be executed. The real challenge is normally hidden behind the models providing the required forces $\boldsymbol{F}_j$. We will get back to this point in the following sections.

For the sake of **efficiency**, note that is it very easy to implement a naive IBM which is, despite being mathematically correct, horribly inefficient. Eq. (11.32) clearly shows that the sum should run over the lattice neighbours of a given boundary node. For a boundary node $\boldsymbol{r}_j$, it is easy to identify the neighbouring lattice sites. However, eq. (11.33) suggests to go the other way around and to identify all boundary markers in the vicinity of a given lattice site. This can be extremely expensive, in particular when the Eulerian lattice is large. A small trick can make the computational effort for interpolation and spreading identical though. In order to do so, we run over all known boundary markers $\boldsymbol{r}_j$ and compute the fraction of the force density a neighbouring lattice site $\boldsymbol{x}$ would receive:

$$\delta_j \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}_j(t)\Delta(\boldsymbol{r}_j(t), \boldsymbol{x}). \tag{11.40}$$

Here, $\delta_j \boldsymbol{F}(\boldsymbol{x})$ is the contribution to $\boldsymbol{F}(\boldsymbol{x})$ due to the presence of $\boldsymbol{r}_j$ alone. All these contributions are simply summed and the correct total force density $\boldsymbol{F}(\boldsymbol{x}) = \sum_j \delta_j \boldsymbol{F}(\boldsymbol{x})$ is automatically obtained in the end. This also highlights the conceptual difference between interpolation and spreading. In fact, Tryggvason *et al.* [69] give the helpful advice (where "front" means "boundary" in our case):

> When information is transferred between the front and the fixed grid, it is always easier to go from the front to the grid and not the other way around. Since the fixed grid is structured and regular, it is very simple to determine the point on the fixed grid that is closest to a given front position.

Implications of the combination of IBM and LBM

Although the IBM is just another way to impose boundary conditions on the Navier-Stokes level, the populations f_i are completely unimpressed by the presence of the Lagrangian marker points. In particular, the f_i simply penetrate any closed IB surface. This is not problematic as long as one is only interested in the no-slip condition of the velocity $\boldsymbol{u}$ and one does not care what the populations are doing. But we can already see that the IBM is not an ideal approach when one wants to keep, for example, two fluids separate on two sides of a membrane.

Another observation is that the fluid usually fills the entire space, including the regions inside any boundary. This significantly simplifies things but can also lead

to additional difficulties. For example, it has been shown that the dynamics of the interior fluid can have an effect on the dynamics of the exterior fluid if the immersed boundaries are rotating [70]. There are ways around this, for example by adding *interior* marker points. In the following, we will not discuss methods with interior markers, such as direct-forcing/fictitious-domain methods [71].

Distribution of markers in space

One open question is how to distribute the markers $\boldsymbol{r}_j$ in space initially. For 2D problems this answer is easy to answer: define a 1D chain of markers with a given mutual distance d on the boundary. Each marker knows which one is its left and right neighbour.

The choice of d is a more delicate issue. On the one hand, intuitively, d cannot be too large because otherwise there are "holes" in the boundary and fluid can flow between markers. On the other hand, too small a value for d can lead to problems as well [72]. This is due to the peculiarities of the IBM algorithm: the marker position update relies on the interpolated fluid velocity. If two markers are very close, $d \ll \Delta x$, they essentially see the same fluid environment and move with the same velocity. Markers which are too close can therefore not be separated again (or only with a lot of effort) and they can stick together. It is usually recommended to choose d somewhere between 0.5 and one lattice constant, but some authors even choose $d \approx 2\Delta x$. We will get back to this point in section 11.4.3.

The situation is much more complicated in 3D where boundaries are generally curved 2D surfaces. One has to distribute the markers such that the mutual distance of any pair of neighbours is approximately the same. This can be a tedious task for general surface shapes and is one of the biggest challenges when applying the IBM in 3D. Furthermore, the node connectivity (*i.e.* an unstructured mesh) is required for deformable boundaries and additional constraints may apply in those situations (*e.g.* the resulting triangular face elements should be as equilateral as possible). Here, we can only give some starting points for further literature studies:

- For simple geometries of high symmetry (spheres, red blood cells), one can start from an *icosahedron* and subdivide each triangular surface element into n^2 ($n > 1$ being an integer) triangular elements [73, 68]. The markers are radially or tangentially shifted to approximate the desired boundary shape.
- In the *minimum potential approach* [74] a fixed number of markers is initially randomly distributed on the surface. Markers interact via repulsive forces and move along the surface until the system has found an energetic minimum. The resulting marker configuration can then be used in simulations as initial boundary discretisation.
- Feng and Michaelides [74] presented a different approach to distribute markers on a sphere.
- There exist free meshing tools which can cope with more complicated boundary shapes, for example [75, 76].

Accuracy and convergence

One shortcoming of the IBM is that the velocity interpolation does not generally maintain the solenoidal properties of the fluid. Even if the fluid solver is perfectly divergence-free (which LBM usually cannot claim), the interpolated velocity may not be divergence-free. The consequence is that the volume of an enclosed region can change in time.

Furthermore, the IBM is formally a first-order accurate boundary condition [57]. There seems to be some dispute in the literature about the actual convergence rate though. While Peng and Luo [77] report second-order convergence, other authors observed only first-order convergence for the velocity field [78, 79].

Related to the question of accuracy and convergence is the apparent size of particles and/or apparent location of walls modelled with the IBM. Several authors, *e.g.* [74, 68, 80], have reported that particles appear to be larger than they actually are. Instead of the input radius r, a larger radius $r + \delta r$ is observed where δr is somewhere between $0.2\Delta x$ and $0.5\Delta x$, depending on the chosen stencil (a kernel with wider support usually leads to a larger δr). In order to model a sphere of actual radius r, Feng and Michaelides [74] suggested to distribute markers on a sphere with radius

$$r_\mathrm{b} = \sqrt[3]{\frac{r^3 + (r - \Delta x)^3}{2}} \tag{11.41}$$

instead. This finding is important for the modelling of particle suspensions or porous media where the rheology strongly depends on the volume fraction and porosity. We will get back to the convergence and apparent wall location in section 11.4.3.

Once again, we see that there is no free lunch. The advantages of the standard IBM (ease of implementation, no need to find boundary intersections, no treatment of fresh fluid nodes required), which explain the IBM's popularity, are challenged by inferior accuracy and convergence compared to other boundary conditions. Note, however, that there have been efforts to make the IB-LBM more accurate [62, 63, 80].

11.4.3 Explicit feedback IBM for rigid boundaries

We show a simple way to compute the nodal forces $\boldsymbol{F}_j$ for (nearly) rigid boundaries. This *explicit* IBM is easy to implement but shows weak stability properties. After discussing the algorithm we use the explicit IBM to model Poiseuille flow and demonstrate the convergence and boundary location issues within the IBM. Note that the explicit IBM does not work very well for unsteady flows as it takes some time for the marker points to respond to the flow.

Algorithm

A rigid body is defined by $|\boldsymbol{r}_j(t) - \boldsymbol{r}_k(t)| = \text{const}$ for any two points $\boldsymbol{r}_j$ and $\boldsymbol{r}_k$ of the body. The simplest way to approximate rigid objects with the IBM is to model the boundary as a collection of marker points $\boldsymbol{r}_j(t)$ which are individually connected by an elastic spring to their reference locations $\boldsymbol{r}_j^{(0)}(t)$. Feng and Michaelides [58] first proposed this idea within the framework of IB-LBM in 2004. While the virtual reference locations obey the rigidity condition exactly, $|\boldsymbol{r}_j^{(0)}(t) - \boldsymbol{r}_k^{(0)}(t)| = \text{const}$, the real markers are allowed to deviate slightly from this condition.

The magnitude of the undesired body deformation can be controlled by springs with strength κ. We can then explicitly compute the marker "penalty" force $\boldsymbol{F}_j$ from a function like

$$\boldsymbol{F}_j(t) = -\kappa \delta \boldsymbol{r}_j(t), \quad \delta \boldsymbol{r}_j(t) = \boldsymbol{r}_j(t) - \boldsymbol{r}_j^{(0)}(t) \tag{11.42}$$

at each time step so that the required nodal forces $\boldsymbol{F}_j$ are known. Contrarily, Feng and Michaelides [58] proposed a form similar to

$$\boldsymbol{F}_j(t) = \begin{cases} 0 & |\delta \boldsymbol{r}_j(t)| = 0 \\ -\kappa \frac{\delta \boldsymbol{r}_j(t)}{|\delta \boldsymbol{r}_j(t)|} & |\delta \boldsymbol{r}_j(t)| > 0 \end{cases}. \tag{11.43}$$

In the example shown below, we use another penalty force:

$$\boldsymbol{F}_j(t) = -\kappa \frac{d}{\Delta x} \delta \boldsymbol{r}_j(t). \tag{11.44}$$

The difference to eq. (11.42) is that the *force per node* is weighted by the average distance d between the nodes. This guarantees that increasing the number of markers (and therefore decreasing d) does not increase the *total force* at the boundary. In any case, the IBM algorithm in section 11.4.1 is employed: in step 1, the forces $\boldsymbol{F}_j(t)$ are obtained *via* one of the approaches shown above.

Using the explicit penalty IBM, each marker point is allowed to be slightly carried away from its reference position. Each point applies a penalty force as discussed above. This force then tends to pull the marker back towards its reference position. After a few time steps (given a steady flow), a marker point will reach an "equilibrium position" where the force it exerts on the fluid is just enough to keep the fluid, and therefore itself, in place. It has then achieved a no-slip condition locally.

Ideally, the exact form of the penalty force should not be important, but it depends on the chosen parameter values whether this is actually the case. For example, if κ is too small, the undesired deformation becomes too large, and if κ is too large, the simulation can become unstable. A clear disadvantage of this method is that the optimum range for κ has to be obtained and that a small time step may be necessary. It is not possible to achieve perfectly rigid boundaries with an explicit IBM algorithm.

Finally, we distinguish between three fundamental cases:

1. The rigid body is fixed in space. All reference points $\boldsymbol{r}_j^{(0)}$ are stationary, and their positions do not have to be updated. The Poiseuille flow in the example below belongs to this category.

2. The body is rigid, and its motion is externally prescribed. This is similar to the first case, but the marker point positions $\boldsymbol{r}_j^{(0)}$ are updated according to the *a-priori* known velocity.
3. The body is rigid but can move *freely* in space. This means that the reference points $\boldsymbol{r}_j^{(0)}$ have to be updated according to the equations of motion of a rigid body. In contrast to the second case, this requires the momentum and angular momentum exchange to be integrated on the surface of the body to find the total force and torque acting on the body. Updating the marker positions of rigid bodies can be complicated. We will not discuss details here and instead refer to the literature [58, 74, 70, 81].

Stationary boundary: Poiseuille flow

We simulate a force-driven Poiseuille flow along the x-axis in 2D with (nearly) rigid boundaries as shown in fig. 11.13. The gravitational force density driving the flow is $F = 10^{-5}$, and the fluid domain consists of $N_x \times N_y = 19 \times 20$ nodes on a D2Q9 lattice. The BGK collision operator with $\tau = 1$ is used. We approximate both walls by lines of markers with mutual distance d as free parameter and employ the penalty force in eq. (11.44). The distance between the IBM walls is $D = 15.3\Delta x$. The spring constant κ is the second free parameter. Simulations are run until the velocity profile is stationary.

We have chosen a prime number for N_x and a non-integer for D to reduce the symmetry of the problem and therefore avoid situations which may accidentally have small numerical errors. Note, however, that the chosen benchmark problem is still highly idealised. Typical IBM applications involve moving curved boundaries

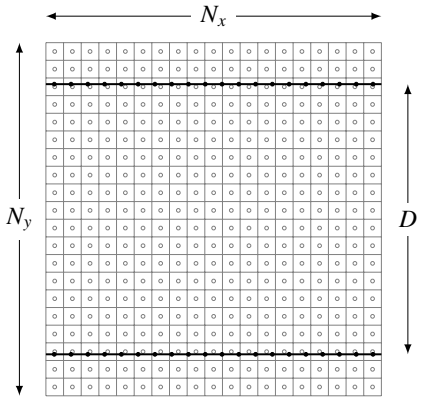


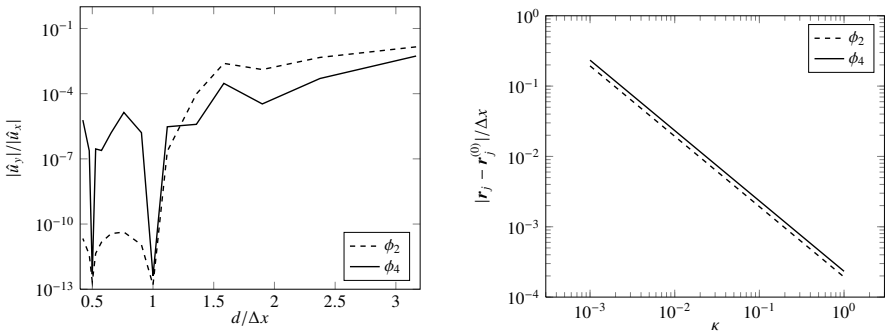
Fig. 11.13: Setup of the Poiseuille flow problem. The lattice size is $N_x \times N_y = 19\Delta x \times 20\Delta x$, and the distance between the IBM walls is $D = 15.3\Delta x$. In this particular example, $d = 0.95\Delta x$ is chosen.

with complex shapes. The purpose of this exercise is to get an initial feeling for the IBM simulation parameters.

The first task is to investigate the effect of the Lagrangian mesh spacing $d/\Delta x$. We keep $\kappa = 1$ fixed and vary d for two interpolations stencils, ϕ_2 and ϕ_4. As error measure we take the largest value of u_y in the simulation, normalised by the Poiseuille peak velocity $\hat{u}_x$. Note that ideally we expect $u_y = 0$ everywhere. The results are shown in fig. 11.14a. The ϕ_2-errors are larger than the ϕ_4-errors for $d > \Delta x$, but they are smaller for $d < \Delta x$. A resonance effect with vanishing u_y can be seen for $d = \Delta x$ and $d = 0.5\Delta x$. In those situations the problem is highly symmetric as the system is x-periodic after a single lattice unit. Generally we conclude that d should not be larger than $1.5\Delta x$. Cheng *et al.* [60] reported a similar observation. For this simple example, ϕ_2 provides significantly better results than ϕ_4, but this observation should certainly not be generalised to arbitrary situations. The resonance effect is expected to disappear for more complex geometries with curved boundaries.

In the second test, we set $d = \Delta x$ and vary the penalty parameter κ. Due to the explicitness of the algorithm, the Lagrangian nodes are slightly dragged by the fluid along the x-axis until the penalty force balances the drag force. We show the displacement of the Lagrangian nodes as function of penalty parameter κ in fig. 11.14b. As expected, the displacement is inversely proportional to the penalty parameter. For $\kappa = 1$, the displacement is less than 0.1% of a lattice spacing Δx, which should be sufficient for most applications. We found that $\kappa > 3$ leads to instability. Concluding, $d = \Delta x$ and $\kappa = 1$ are reasonable choices for the current problem; we will keep these values for the final tests. Note, however, that different flow configurations may require different parameter values for optimum results.

We now investigate the apparent boundary location and the convergence rate of the IBM. For that purpose, we perform a grid refinement study. We only vary the system size, but keep $d = \Delta x$, $\kappa = 1$ and $\tau = 1$ fixed (diffusive scaling). As a consequence, the gravitational force density F scales with $(D/\Delta x)^{-3}$ and the ex-



(a) Sensitivity to mesh spacing

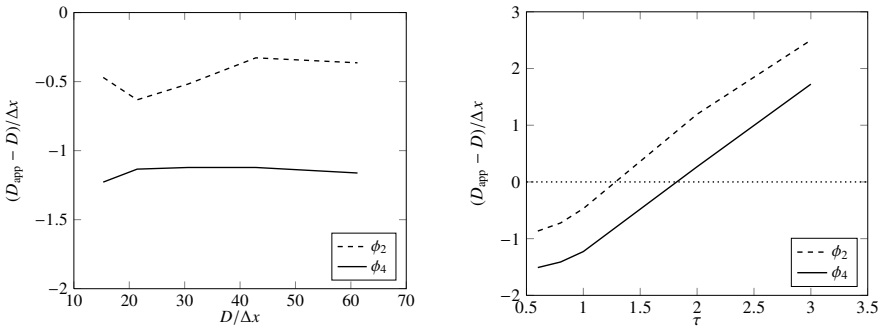
(b) Sensitivity to penalty parameter

Fig. 11.14: Benchmark results showing the effect of mesh spacing d and penalty parameter κ on the accuracy of the explicit IBM for Poiseuille flow.

pected peak velocity $\hat{u}_x$ with $(D/\Delta x)^{-1}$ (*cf.* chapter 7). For each simulation, we fit a parabola to the flow field in the central region between $\pm D/2$ and compute the appararent channel diameter D_{app}. Fig. 11.15a shows the mismatch of the channel diameter, $D_{\text{app}} - D$, as function of resolution. Obviously the channel appears to be smaller than expected, which also leads to a reduced peak velocity compared to its expected values (not shown here). The mismatch is larger for ϕ_4 than for ϕ_2. Futhermore, the diameter mismatch does not significantly depend on the resolution. This means that the mismatch cannot be removed by increasing the resolution, which leads only to a first-order convergence rate of the velocity error. We can therefore conclude that the IBM is generally a first-order accurate velocity boundary condition.

In the final test we investigate how the wall location mismatch depends on the relaxation time τ. We now vary τ at fixed resolution. Fig. 11.15b reveals that D_{app} is a function of τ. Depending on the value of τ, the channel can appear smaller or larger than expected. While the exact values depend on the choice of the interpolation stencil, we can conclude that the appararent channel diameter increases roughly linearly with τ for $\tau > 1$. This is a highly undesirable effect that has been discussed by several authors [82, 60, 63]. It has recently been suggested to use the MRT [62] or TRT [63] collision operators to resolve this problem. We will not discuss these approaches here.

As already concluded at the end of section 11.4.2, the IBM accuracy is typically inferior to other available boundary conditions. Care has to be taken when the exact channel diameter or particle size (for suspension simulations) is important. In the end, it can take a significant amount of work to make sure that an IBM code is working reliably.



(a) Sensitivity to resolution

(b) Sensitivity to relaxation time

Fig. 11.15: Channel diameter mismatch as function of spatial resolution and LBM relaxation time τ.

11.4.4 Direct-forcing IB-LBM for rigid boundaries

The explicit penalty IBM for rigid boundaries has a major disadvantage: it involves a free parameter whose choice affects the stability and accuracy. We seek an alternative implementation without a free parameter. This means that the IB force has to be computed directly from the flow field. Therefore, we call this class of methods *direct-forcing IB-LBM*.

Feng and Michaelides [83, 74] originally combined the direct forcing IBM with the LBM. A number of alternative direct-forcing IB-LBMs have been proposed since then. We can distinguish between three different approaches, each with different levels of accuracy and numerical cost:

1. implicit IBM
2. multi-direct-forcing IBM (iterative)
3. direct-forcing IBM (explicit)

We cannot present and compare all available approaches in depth. Instead, we will start from the underlying hydrodynamic problem and show which steps are necessary to construct a reliable parameter-free IB-LBM.

Background

We assume a rigid boundary described by a number of marker points at positions $\boldsymbol{r}_j$. The markers have known velocities $\dot{\boldsymbol{r}}_j = \boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$, the desired boundary velocity. In most situations, the boundary is resting, but there is no fundamental difficulty with moving (translating and rotating) boundaries.

Before collision, the fluid velocity on a lattice node $\boldsymbol{x}$ is

$$\boldsymbol{u}(\boldsymbol{x}) = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i(\boldsymbol{x}) \boldsymbol{c}_i. \tag{11.45}$$

In the absence of a force, collision leaves the momentum and velocity invariant. The only mechanism that can change the fluid velocity during collision is a body force $\boldsymbol{F}$:

$$\boldsymbol{u}^\star(\boldsymbol{x}) = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i^\star(\boldsymbol{x}) \boldsymbol{c}_i = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i(\boldsymbol{x}) \boldsymbol{c}_i + \frac{\boldsymbol{F}(\boldsymbol{x}) \Delta t}{\rho(\boldsymbol{x})}. \tag{11.46}$$

As usual, a star denotes post-collision quantities. Furthermore, we drop the time t because everything which follows is happening within a single time step.

We know that the physical fluid velocity during a time step is the average of the pre- and post-collision velocities [66]:

$$\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}) = \frac{\boldsymbol{u}(\boldsymbol{x}) + \boldsymbol{u}^\star(\boldsymbol{x})}{2} = \boldsymbol{u}(\boldsymbol{x}) + \frac{\boldsymbol{F}(\boldsymbol{x}) \Delta t}{2\rho(\boldsymbol{x})}. \tag{11.47}$$

The central idea of any direct-forcing IB-LBM is to construct the force $\boldsymbol{F}(\boldsymbol{x})$ in such a way that $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ matches the known boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ at the marker positions to satisfy the no-slip condition. As we will now see, this is a non-trivial task and explains why there is a number of direct-forcing variants in the literature.

Since we are working in the framework of the IBM, the boundary velocity is known at the positions of the boundary markers: $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \dot{\boldsymbol{r}}_j$. This means that we have to interpolate $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ at the boundary markers $\boldsymbol{r}_j$ to obtain $\boldsymbol{u}_\mathrm{f}(\boldsymbol{r}_j)$ first, then find the required boundary force $\boldsymbol{F}_j$ and finally spread the force back to the lattice to obtain $\boldsymbol{F}(\boldsymbol{x})$.

The difficulty is caused be the non-local velocity interpolation and force spreading. In order to compute $\boldsymbol{F}_j$, we require the velocity on all lattice nodes $\boldsymbol{x}$ close to $\boldsymbol{r}_j$. In return, we have to spread $\boldsymbol{F}_j$ back to those lattice nodes. However, many lattice nodes $\boldsymbol{x}$ participate in interpolation and spreading of more than one marker $\boldsymbol{r}_j$ *at the same time*. This means that eq. (11.47) has to be solved *simultaneously* on all lattice nodes to guarantee a consistent solution.

Implicit IB-LBM

In 2009, Wu and Shu [84] proposed the *implicit velocity correction-based IB-LBM* which is probably the most accurate and consistent way to enforce the no-slip condition at a rigid boundary with the IB-LBM. We will only provide the derivation of the algorithm. Benchmark tests can be found in [84, 85, 86].

The basic idea of the implicit IB-LBM is to consider the required force density $\boldsymbol{F}(\boldsymbol{x})$ as the unknowns for which the problem has to be solved in such a way that the no-slip condition is satisfied. Since the unknowns $\boldsymbol{F}(\boldsymbol{x})$ depend on the current and the desired flow field, the problem is implicit.

The first step is to write the physical fluid velocity in eq. (11.47) as

$$\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}) = \boldsymbol{u}(\boldsymbol{x}) + \delta\boldsymbol{u}(\boldsymbol{x}) \tag{11.48}$$

where $\boldsymbol{u}(\boldsymbol{x})$, given by eq. (11.45), is the known uncorrected velocity and $\delta\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{F}(\boldsymbol{x})\Delta t)/(2\rho(\boldsymbol{x}))$ is the unknown velocity correction required to achieve the desired no-slip condition.

Now, we use the IBM relations eq. (11.32) and eq. (11.33) to link Eulerian and Lagrangian quantities. We can express the Eulerian correction terms $\delta\boldsymbol{u}(\boldsymbol{x})$ by their Lagrangian counterparts $\delta\boldsymbol{u}(\boldsymbol{r}_j)$:

$$\delta\boldsymbol{u}(\boldsymbol{x}) = \sum_j \delta\boldsymbol{u}(\boldsymbol{r}_j)\Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.49}$$

Note that $\delta\boldsymbol{u}(\boldsymbol{r}_j)$ is proportional to the unknown Lagrangian force $\boldsymbol{F}_j$.

Let $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ be the desired boundary velocity imposed on the Lagrangian nodes. The aim is construct an Eulerian flow field $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ which, interpolated at the boundary nodes, equals the boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$:

$$\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \sum_{\boldsymbol{x}} \boldsymbol{u}_\mathrm{f}(\boldsymbol{x}) \Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.50}$$

To achieve this, we combine eq. (11.48), eq. (11.49) and eq. (11.50):

$$\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \sum_{\boldsymbol{x}} \left[\boldsymbol{u}(\boldsymbol{x}) + \sum_k \delta\boldsymbol{u}(\boldsymbol{r}_k) \Delta(\boldsymbol{r}_k, \boldsymbol{x}) \right] \Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.51}$$

The only unknowns in this equation are the desired correction terms $\delta\boldsymbol{u}(\boldsymbol{r}_k)$. We can rewrite this equation in the following way:

$$\sum_k \underbrace{\left[\sum_{\boldsymbol{x}} \Delta(\boldsymbol{r}_k, \boldsymbol{x}) \Delta(\boldsymbol{r}_j, \boldsymbol{x}) \right]}_{A_{jk}} \underbrace{\delta\boldsymbol{u}(\boldsymbol{r}_k)}_{X_k} = \underbrace{\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) - \sum_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}) \Delta(\boldsymbol{r}_j, \boldsymbol{x})}_{B_j} \tag{11.52}$$

or, in simple matrix-vector notation, as $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$.

The vectors $\boldsymbol{X}$ and $\boldsymbol{B}$ have N elements, and $\boldsymbol{A}$ is an $N \times N$-matrix where N is the number of marker points, *i.e.* j and k run from 1 to N. The elements of $\boldsymbol{A}$ are functions of the node positions $\boldsymbol{r}_j$ only, depending on the choice of the IBM stencil Δ. Finding the unknowns $\boldsymbol{X}$ *via* $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$ requires inversion of $\boldsymbol{A}$. Obviously the matrix A can be large, with N typically ranging from several 10^2 to several 10^4 or 10^5. Still, the matrix is usually sparse because only nearby Lagrangian nodes affect each other.

Concluding, the implicit IB-LBM algorithm works as follows:

1. Compute the matrix $\boldsymbol{A}$ and its inverse $\boldsymbol{A}^{-1}$ from the known node positions $\boldsymbol{r}_j$. See [84] for details.
2. Stream the populations to obtain $f_i(\boldsymbol{x})$ and compute the density and uncorrected velocity from $\rho\boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i$.
3. Using the known boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ and the uncorrected fluid velocity $\boldsymbol{u}(\boldsymbol{x})$, solve the matrix equation, eq. (11.52), for the unknown corrections $\delta\boldsymbol{u}(\boldsymbol{r}_j)$.
4. Spread $\delta\boldsymbol{u}(\boldsymbol{r}_j)$ to the Eulerian grid *via* eq. (11.49).
5. Compute the desired force density from $\delta\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{F}(\boldsymbol{x})\Delta t)/(2\rho(\boldsymbol{x}))$.
6. Perform collision with forcing.
7. If the boundary is stationary, *i.e.* all boundary velocities obey $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \boldsymbol{0}$, go back to step 2 for the next time step.
8. If the boundary is not stationary, update the positions $\boldsymbol{r}_j$ and velocities $\dot{\boldsymbol{r}}_j = \boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$. The position update may be enforced (*e.g.* oscillating cylinder) or a consequence of fluid stresses (*e.g.* freely moving cylinder). In the latter case, a suitable time integrator has to be chosen [58, 74, 70, 81].
9. Go back to step 1 for the next time step.

Note that the re-computation of the matrix $\boldsymbol{A}$ and its inverse at each time step for non-stationary boundaries can be expensive when N is large. Therfore, alternative approaches, such as multi-direct forcing, which are computationally more efficient and conceptually simpler, have been suggested.

Multi direct-forcing IB-LBM

The aim of *multi direct-forcing IB-LBM* is to avoid the construction and inversion of the matrix $\boldsymbol{A}$ of the implicit IB-LBM, while keeping its consistency. Kang and Hassan [59] provided an exhaustive overview of the multi direct-forcing IB-LBM. Since the underlying idea is similar to that of the implicit IB-LBM, we only provide the algorithm and a few comments.

Instead of constructing and inverting a large matrix $\boldsymbol{A}$, the multi direct-forcing method relies on an iterative approach to satisfy the no-slip condition at all markers $\boldsymbol{r}_j$ simultaneously. Again, the underlying idea is to take advantage of the velocity correction in eq. (11.47). The algorithm of the multi direct-forcing approach can be summarised as follows:

1. Set iteration counter m to 0.
2. Stream the populations to obtain $f_i(\boldsymbol{x})$ and compute the density and uncorrected velocity from $\rho\boldsymbol{u}^{(m)} = \sum_i f_i \boldsymbol{c}_i$.
3. Interpolate $\boldsymbol{u}^{(m)}(\boldsymbol{x})$ at the boundary marker locations $\boldsymbol{r}_j$ to obtain $\boldsymbol{u}^{(m)}(\boldsymbol{r}_j)$.
4. Increment iteration counter m by 1.
5. Compute the Lagrangian correction force from [59]

$$\boldsymbol{F}_j^{(m)} = 2\rho \frac{\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) - \boldsymbol{u}^{(m-1)}(\boldsymbol{r}_j)}{\Delta t}. \tag{11.53}$$

6. Spread $\boldsymbol{F}_j^{(m)}$ to the Eulerian lattice to obtain $\boldsymbol{F}^{(m)}(\boldsymbol{x})$.
7. Correct previous Eulerian velocity according to

$$\boldsymbol{u}^{(m)}(\boldsymbol{x}) = \boldsymbol{u}^{(m-1)}(\boldsymbol{x}) + \frac{\boldsymbol{F}^{(m)}(\boldsymbol{x})\Delta t}{2\rho(\boldsymbol{x})}. \tag{11.54}$$

8. Repeat steps 3–7 until m reaches a pre-defined limit m_max or until $\boldsymbol{u}^{(m)}(\boldsymbol{r}_j)$ converges to $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$.
9. Use the total correction force

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{m=1}^{m_\mathrm{max}} \boldsymbol{F}^{(m)}(\boldsymbol{x}) \tag{11.55}$$

   in the collision step.
10. Go back to step 1 for the next time step.

Kang and Hassan [59] compared results of benchmark tests for different iteration numbers up to $m_\mathrm{max} = 20$. They found that $m_\mathrm{max} = 5$ is a reasonable compromise of accuracy and efficiency. Since the iteration involves only those lattice nodes close to the boundary, the additional computational cost is relatively low.

Explicit, non-iterative direct-forcing IB-LBM

As pointed out by Kang and Hassan [59], a non-iterative direct-forcing scheme can be obtained as special case of the multi direct-forcing method in the previous section. Setting $m_{\max} = 1$ leads to a simple explicit scheme that does not require expensive matrix inversions or iterations. This special case is commonly denoted "direct-forcing" IB-LBM, although the implicit and iterative methods are, strictly speaking, also direct-forcing methods.[5]

There exist different flavours of non-iterative direct-forcing IB-LBM (see for example [63]). However, it is obvious that this method will generally not give results of a comparable accuracy and consistency compared to implicit or iterative schemes.

11.4.5 Explicit IBM for deformable boundaries

The first works utilising the deformable IB-LBM for flowing deformable red blood cells were published in 2007 by several groups [87, 88, 89]. The overall algorithm follows the layout described in section 11.4.2. The step from the IBM algorithm for rigid boundaries as presented in section 11.4.3 to deformable boundaries is straightforward. Instead of finding suitable penalty forces to keep the boundary deformation as small as possible, one has to use forces which arise from elastic surface stresses due to the (desired) deformation of the boundary. This requires two additional ingredients: i) a constitutive model for the boundary deformation and ii) a surface mesh (*i.e.* markers *and* their connectivity) to evaluate the boundary deformation.

Sui *et al.* [90] were the first to present a 3D model for elastic particles (capsules, red blood cells) in an LB simulation with well-defined constitutive behaviour and a finite-element method to find the elastic membrane forces. Krüger *et al.* [68] later investigated the effect of the choice of interpolation stencil and distance between neighbouring Lagrangian nodes on the deformation of a capsule in shear flow. The IB-LBM for elastic problems has been applied to, for example, viscous flow over a flexible sheet [79] and dense suspensions of red blood cells [91] (see also fig. 11.16).

The IBM provides a major advantage over other boundary conditions for the LBM when it comes to deformable objects. Since IBM boundaries in the original implementation are intrinsically deformable, it is relatively simple to turn this presumed disadvantage into an advantage for problems where the deformability is actually desired. Allowing Lagrangian markers to move with the fluid and distributing forces to the fluid is the natural algorithm of the IBM and lends itself to problems where the fluid causes structure deformation and the structure "reacts" elastically. Applying any of the other boundary condi-

[5] Remember that "direct forcing" means that there are no free parameters, such as the elaticity κ of the explicit method in section 11.4.3.

tions presented in this chapter to deformable boundaries is significantly more difficult.

Constitutive models

The *constitutive model* contains all the physics of the boundary deformation. Its choice is independent of the IBM algorithm itself and has to be defined by the user. In the end, the IBM expects the marker forces $\boldsymbol{F}_j$, but the IBM itself is unable to provide them. Boundaries are mostly considered hyperelastic (*i.e.* the dynamics can be fully described in terms of an energy density) or viscoelastic. In the former case, the marker forces depend only on the current deformation state, in the latter case the forces depend both on the deformation state and its rate of change.

There exists a large variety of hyperelastic and viscoelastic models for deformable boundaries. The problem of finding and implementing an appropriate constitutive model is highly problem-specific. We cannot delve into details here; this could easily fill a book on its own. The most commonly used hyperelastic models for red blood cells are briefly discussed in [90].

For simplicity, we will assume that hyperelastic models can be written in the form

$$\boldsymbol{F}_j(t) = \boldsymbol{F}_j(\{\boldsymbol{r}_k(t)\}) \tag{11.56}$$

and viscoelastic models as

$$\boldsymbol{F}_j(t) = \boldsymbol{F}_j(\{\boldsymbol{r}_k(t)\}, \{\dot{\boldsymbol{r}}_k(t)\}). \tag{11.57}$$

This means that the instantaneous marker forces are (arbitrarily complicated) functions of all current boundary marker positions and, if viscoelastic, of all current boundary marker velocities. Once these laws have been specified, they can be hard-

Fig. 11.16: Flow of a single (left) and multiple (right) red blood cells in a straight tube (indicated by solid horizontal lines) with circular cross-section. The tube diameter is slightly larger than that of an undeformed red blood cell. The Lagrangian mesh consists of 998 nodes and 2,000 triangular elements. The simulations are based on the model presented in [91].

coded and used to find the forces $\boldsymbol{F}_j$ for a given deformation state at every time step.

Example 11.1. A simple hyperelastic constitutive model which can be used in 2D and 3D is (dropping the time dependence for simplicity)

$$\boldsymbol{F}_j(\{\boldsymbol{r}_k\}) = -\kappa \sum_{k \neq j} \frac{d_{jk} - d_{jk}^{(0)}}{d_{jk}^{(0)}} \frac{\boldsymbol{d}_{jk}}{d_{jk}}, \quad \boldsymbol{d}_{jk} = \boldsymbol{r}_k - \boldsymbol{r}_j, \quad d_{jk} = |\boldsymbol{d}_{jk}| \tag{11.58}$$

where κ is an elastic modulus, the sum runs over all next neighbours of marker j and $d_{jk}^{(0)}$ is the equilibrium distance between markers j and k. This example shows that not only the markers, but also their connectivity is an important part of the problem description. In many situations, additional constraints are necessary, for instance conservation of the total volume or surface of a boundary.

In most cases of hyperelastic boundaries one first defines an elastic energy density $\epsilon(\{\boldsymbol{r}_j(t)\})$. The force acting on node j can then be recovered by applying the principle of virtual work,

$$\boldsymbol{F}_j = \frac{\partial \epsilon(\{\boldsymbol{r}_k\})}{\partial \boldsymbol{r}_j} A_j, \tag{11.59}$$

where A_j is the area related to marker j, *e.g.* its Voronoi area. More details are provided in [90, 68].

11.4.6 Additional variants and similar boundary treatments

The previous sections cover the most prominent flavours of the IB-LBM. This is, however, not the end of the rope. There are more variations on the market, some of which we want to mention in the following.

There are a number of fluid-structure interaction approaches which share some features with the IBM (in particular the existence of off-lattice markers or a Lagrangian mesh and kernel functions for velocity interpolation and/or force spreading), but their algorithms reveal distinct differences. Schiller [92] recently revisited those algorithms and pointed out their mathematical similarity.

1. Ahlrichs and Dünweg [93] introduced a dissipative coupling method for LBM and molecular dynamics (MD), which has been further analysed by Caiazzo and Maddu [78] and recently reviewed by Dünweg and Ladd [94] and is used in the open-source package ESPResSo [95], mostly for polymer simulations. Lagrangian markers are allowed to move with a different velocity than the velocity of the fluid at the location of the marker (obtained by velocity interpolation). A finite slip velocity results in a drag force acting on the marker whose magnitude is controlled *via* a numerical drag coefficient. An equal but opposite force is exerted on the fluid by spreading it to the Eulerian lattice. Additionally, the markers may

experience external or interaction forces. The marker update is treated by higher-order MD, which requires the introduction of another model parameter, a finite marker mass. An advantage of this approach is that the time step for the update of the markers is decoupled from the LB time step, which can be exploited to implement more stable time-integration schemes; a freedom which is not available for the conventional IB-LBM algorithm. Disadvantages are that the no-slip condition is not strictly satisfied and that two model parameters are required (drag coefficient and marker mass).

2. The momentum-exchange-based IB-LBM, as proposed for rigid boundaries [96, 97] and recently extended to flexible boundaries [98], uses a different approach to obtain the force density acting on the fluid. The basic idea is to interpolate the LB populations (rather than the velocity) to find their value at the location of the Lagrangian markers. The bounce-back scheme is then applied on the Lagrangian mesh to find the momentum exchange and therefore the marker force $\boldsymbol{F}_j$ which is then distributed to the lattice *via* the standard force spreading operation. As a consequence, there is no need for user-defined penalty parameters (for rigid boundaries) and the markers can move independently of the fluid motion. However, this may lead to a violation of the no-slip condition.[6]
3. Wu and Aidun [99] proposed the so-called *external-boundary force* (EBF) which can be used both as alternative to the direct-forcing IBM for rigid boundaries and for deformable objects. Similar to the other two examples above, the most notable difference to the IBM is that the markers are not directly advected by the fluid. Instead, a relative slip velocity is permitted which is counteracted by a fluid-solid interaction force, which is essentially a penalty force. Note that this force does not require a free parameter such as the dissipative coupling does. Also here, the allowance of a relative slip velocity may lead to problems with the no-slip condition.

Concluding, we can say that, although it is desirable to decouple the motion of the marker points from the fluid motion (as this allows higher-order and therefore potentially more stable time integration schemes), the no-slip condition at the boundary cannot be strictly enforced at the same time. The reason is that all the methods discussed above [93, 96, 98, 99] are explicit with respect to the fluid velocity computation. Still, if the precise realisation of the no-slip condition is not the primary goal, the above methods are attractive alternatives and overcome some of the disadvantages of the more conventional IB schemes.

It is also worth mentioning that the IBM can be used to model thermal boundary conditions. In 2010, Jeong *et al.* [100] combined the IBM with a thermal LBM to simulate flows around bluff bodies with heat transfer. Seta [101] later improved the thermal IB-LBM by analysing the governing equations through a Chapman-Enskog analysis. Another IBM variant that has apparently not yet been combined with the

[6] While the authors of [96] use bounce-back to obtain the momentum exchange at the boundary markers, the populations on the lattice are not directly affected by this bounce-back procedure. Although the momentum exchange is correctly obtained, there is no strong mechanism enforcing the local no-slip condition. Therefore, streamlines may penetrate the boundary.

LBM is the so-called penalty IBM (p-IBM) [102, 103, 104] for flexible boundaries. It involves two set of Lagrangian markers: one interacting with the fluid, the other used for the calculation of the Lagrangian forces. Both marker sets are coupled by springs generating penalty forces.

11.5 Concluding remarks

The number of available boundary conditions for the LBM is overwhelming, and it can be a daunting task to grasp the implications of those schemes. In the following we list a series of publications which provide comparative studies of boundary conditions for curved geometries. This should help to understand the relative performance of certain boundary methods for a given flow geometry.

- Ginzburg and d'Humières [28] compared simple (section 11.2.1) and interpolated bounce-back (section 11.2.2) with the multireflection method [28] in a number of stationary situations (inclined Couette and Poiseuille flows, flow over single cylinder and array of cylinders, impulsively started cylinder and moving sphere in a cylinder). They conclude that the multireflection method is more accurate than the linear interpolated bounce-back method.
- Pan *et al.* [5] compared the performance of simple bounce-back, interpolated bounce-back (both linear and quadratic interpolations) and multireflection boundary conditions for porous media simulations. They also included an analysis of the effect of collision operator (BGK *vs.* MRT) on the permeability of idealised porous media. Their main finding is that the permeability is generally viscosity-dependent. Especially the combination of BGK and simple bounce-back leads to a strongly increasing permeability with viscosity, an effect caused by the increasing slip velocity at the boundary. They conclude that the combination of simple bounce-back with MRT is consistently better than with BGK. The reason is that the no-slip condition can be much better controlled in the MRT framework when the viscosity is large. Using interpolated rather than simple bounce-back can also improve the accuracy of the simulations.
- Peng and Luo [77] investigated the relative performance of interpolated bounce-back and a direct-forcing immersed boundary method (IBM, section 11.4.4). They considered steady and unsteady flows about a stationary rigid cylinder in 2D. Their major findings are that both methods require roughly the same computing time, the interpolated bounce-back is more accurate, but that the IBM is easier to implement.
- Chen *et al.* [34] have combined the ghost method (section 11.3.4) with the partially saturated method (PSM, section 11.2.3) in order to remove spurious pressure oscillations. The immediate consequence is that the algorithm becomes significantly more complicated than Noble's and Torczynski's [33] original one: interpolations become necessary to find the ghost node properties and a treatment of fresh nodes is required. This is a clear disadvantage over the original method [33] where the fresh node problem is naturally avoided. The conclu-

sion is that the combined method yields better results than the original PSM or the interpolated bounce-back method, especially for moving obstacles at smaller resolutions. According to Chen *et al.*, PSM is recommended when code simplicity and efficiency are desired, while the combined method should be favoured for high-accuracy applications.

- Chen *et al.* [7] recently conducted a thorough comparison of three bounce-back schemes (standard, interpolated and unified interpolation), two IBM variants (explicit and implicit direct forcing) and three additional methods. The authors were primarily interested in acoustic problems involving sound wave generation from moving bodies due to the fresh node treatment. The authors found that the IBM is more suitable for moving boundaries than the interpolated bounce-back when fresh nodes are involved.
- Nash *et al.* [105] compared the accuracy of simple and interpolated bounce-back, the Guo-Zheng-Shi extrapolation method (GZS, section 11.3.3) and the Junk-Yang method in non-grid-aligned Poiseuille, Womersley and Dean flows at moderate Reynolds numbers (up to 300). The authors found that the Junk-Yang method shows poor stability in the selected parameter range. The linear interpolated bounce-back and the GZS methods have comparable accuracy (with a second-order convergence) although the latter becomes unstable for the highest Reynolds numbers tested. For the situation of interest (flow in inclined channels with moderate Reynolds number), the authors conclude that interpolated bounce-back is the best all-around option, although simple bounce-back (despite its first-order convergence) may be the method of choice when code development time is at a premium.

Everything said up to this point applies to *rigid* boundaries. It seems that the IBM and its related methods is still the most convenient approach for *deformable* boundaries (*cf.* section 11.4.5 and section 11.4.6). The reason is that all other boundary conditions presented in this chapter require an accurate *local* momentum exchange algorithm to compute the local stresses in the deformable material.[7] This is normally a very challenging and expensive problem that is elegantly circumvented by the IBM.

We can generally conclude that all existing boundary conditions claim their own compromise of accuracy, stability and efficiency/ease of implementation. Futhermore, some boundary conditions perform better in stationary situations, others when the boundaries are moving. It is up to the user to identify the requirements before choosing one of the many available boundary conditions. There is no best boundary treatment for all possible scenarios. We hope that this chapter sheds some light on the plethora of boundary conditions and helps the reader to find a suitable scheme for a given problem.

NOTE (TK): I thought of a table collecting the most important information about the presented boundary conditions, but this seems to be a very difficult task. What

[7] For rigid objects it is sufficient to compute the total momentum and angular momentum transfer. Soft objects, however, deform locally. This local deformation depends on the local stresses.

are the criteria? How can complicated restrictions and advantages be shown in a compact way?

References

1. S. Haeri, J.S. Shrimpton, Int. J. Multiphase Flow **40**, 38 (2012)
2. I. Ginzbourg, P.M. Adler, J. Phys. II France **4**(2), 191 (1994)
3. A.J.C. Ladd, J. Fluid Mech. **271**, 285 (1994)
4. M. Bouzidi, M. Firdaouss, P. Lallemand, Phys. Fluids **13**, 3452 (2001)
5. C. Pan, L.S. Luo, C.T. Miller, Comput. Fluids **35**(8–9), 898 (2006)
6. O.E. Strack, B.K. Cook, Int. J. Numer. Meth. Fluids **55**(2), 103 (2007)
7. L. Chen, Y. Yu, J. Lu, G. Hou, Int. J. Numer. Meth. Fluids **74**(6), 439 (2014)
8. M.O. Bernabeu, M.L. Jones, J.H. Nielsen, T. Krüger, R.W. Nash, D. Groen, S. Schmieschek, J. Hetherington, H. Gerhardt, C.A. Franco, P.V. Coveney, J. R. Soc. Interface **11**(99), 20140543 (2014)
9. A.J.C. Ladd, Phys. Rev. Lett. **70**(9), 1339 (1993)
10. A.J.C. Ladd, J. Fluid Mech. **271**, 311 (1994)
11. A.J.C. Ladd, R. Verberg, J. Stat. Phys. **104**(5-6), 1191 (2001)
12. C.K. Aidun, J.R. Clausen, Annu. Rev. Fluid Mech. **42**, 439 (2010)
13. N.Q. Nguyen, A.J.C. Ladd, Phys. Rev. E **66**(4), 046708 (2002)
14. E.J. Ding, C.K. Aidun, J. Stat. Phys. **112**(3-4), 685 (2003)
15. C.K. Aidun, Y. Lu, E.J. Ding, J. Fluid Mech. **373**, 287 (1998)
16. D. Qi, J. Fluid Mech. **385**, 41 (1999)
17. E. Lorenz, A. Caiazzo, A.G. Hoekstra, Phys. Rev. E **79**(3), 036705 (2009)
18. J.R. Clausen, C.K. Aidun, Int. J. Multiphas. Flow **35**(4), 307 (2009)
19. B. Wen, C. Zhang, Y. Tu, C. Wang, H. Fang, J. Comput. Phys. **266**, 161 (2014)
20. O. Behrend, Phys. Rev. E **52**(1), 1164 (1995)
21. M.A. Gallivan, D.R. Noble, J.G. Georgiadis, R.O. Buckius, Int. J. Numer. Meth. Fluids **25**(3), 249–263 (1997)
22. Y. Han, P.A. Cundall, Int. J. Numer. Meth. Fluids **67**(3), 314–327 (2011)
23. B. Chun, A.J.C. Ladd, Phys. Rev. E **75**, 066705 (2007)
24. P.H. Kao, R.J. Yang, J. Comput. Phys. **227**(11), 5671 (2008)
25. X. Descovich, G. Pontrelli, S. Melchionna, S. Succi, S. Wassertheurer, Int. J. Mod. Phys. C **24**(05), 1350030 (2013)
26. P. Lallemand, L.S. Luo, J. Comput. Phys. **184**(2), 406 (2003)
27. D. Yu, R. Mei, W. Shyy, in *41st Aerospace Sciences Meeting and Exhibit*, 2003-953 (AIAA, New York, 2003)
28. I. Ginzburg, D. d'Humières, Phys. Rev. E **68**, 066614 (2003)
29. X. Yin, J. Zhang, J. Comput. Phys. **231**(11), 4295 (2012)
30. O. Dardis, J. McCloskey, Phys. Rev. E **57**(4), 4834 (1998)
31. S.D.C. Walsh, H. Burwinkle, M.O. Saar, Comput. Geosci. **35**(6), 1186 (2009)
32. J. Zhu, J. Ma, Adv. Water Resour. **56**, 61 (2013)
33. D.R. Noble, J.R. Torczynski, Int. J. Mod. Phys. C **09**(08), 1189 (1998)
34. L. Chen, Y. Yu, G. Hou, Phys. Rev. E **87**(5), 053306 (2013)
35. G. Zhou, L. Wang, X. Wang, W. Ge, Phys. Rev. E **84**(6), 066701 (2011)
36. H. Yu, X. Chen, Z. Wang, D. Deep, E. Lima, Y. Zhao, S.D. Teague, Phys. Rev. E **89**(6), 063304 (2014)
37. R. Mei, L.S. Luo, P. Lallemand, D. d'Humières, Comput. Fluids **35**(8–9), 855 (2006)
38. G. Pontrelli, C.S. König, I. Halliday, T.J. Spencer, M.W. Collins, Q. Long, S. Succi, Med. Eng. Phys. **33**(7), 832 (2011)
39. B. Stahl, B. Chopard, J. Latt, Comput. Fluids **39**(9), 1625–1633 (2010)

40. M. Matyka, Z. Koza, Ł. Mirosław, Comput. Fluids **73**, 115 (2013)
41. X. Kang, Z. Dun, Int. J. Mod. Phys. C p. 1450057 (2014)
42. R. Mei, L.S. Luo, W. Shyy, J. Comput. Phys. **155**(2), 307 (1999)
43. B. Wen, H. Li, C. Zhang, H. Fang, Phys. Rev. E **85**(1), 016704 (2012)
44. Z.L. Guo, C.G. Zheng, B.C. Shi, Phys. Fluids **14**, 2007 (2002)
45. A. Tiwari, S.P. Vanka, Int. J. Numer. Meth. Fluids **69**(2), 481 (2012)
46. O.R. Mohammadipoor, H. Niazmand, S.A. Mirbozorgi, Phys. Rev. E **89**(1), 013309 (2014)
47. J.C.G. Verschaeve, B. Müller, J. Comput. Phys. **229**, 6781 (2010)
48. J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Phys. Rev. E **77**(5), 056703 (2008)
49. O. Filippova, D. Hänel, J. Comput. Phys. **147**, 219 (1998)
50. R. Mei, D. Yu, W. Shyy, L.S. Luo, Phys. Rev. E **65**(4), 041203 (2002)
51. J. Bao, P. Yuan, L. Schaefer, J. Comput. Phys. **227**(18), 8472 (2008)
52. R. Khazaeli, S. Mortazavi, M. Ashrafizaadeh, J. Comput. Phys. **250**, 126 (2013)
53. Q. Zou, X. He, Phys. Fluids **9**, 1591 (1997)
54. N. Pellerin, S. Leclaire, M. Reggio, Comput. Fluids **101**, 126 (2014)
55. C.S. Peskin, Flow patterns around heart valves: A digital computer method for solving the equations of motion. Ph.D. thesis, Sue Golding Graduate Division of Medical Sciences, Albert Einstein College of Medicine, Yeshiva University (1972)
56. C.S. Peskin, J. Comput. Phys. **25**(3), 220 (1977)
57. C.S. Peskin, Acta Numerica **11**, 479–517 (2002)
58. Z.G. Feng, E.E. Michaelides, J. Comput. Phys. **195**(2), 602 (2004)
59. S.K. Kang, Y.A. Hassan, Int. J. Numer. Meth. Fluids **66**(9), 1132 (2011)
60. Y. Cheng, L. Zhu, C. Zhang, Commun. Comput. Phys. **16**(1), 136 (2014)
61. R. Mittal, G. Iaccarino, Annu. Rev. Fluid Mech. **37**, 239 (2005)
62. J. Lu, H. Han, B. Shi, Z. Guo, Phys. Rev. E **85**(1), 016711 (2012)
63. T. Seta, R. Rojas, K. Hayashi, A. Tomiyama, Phys. Rev. E **89**(2), 023307 (2014)
64. B.E. Griffith, X. Luo, D.M. McQueen, C.S. Peskin, Int. J. Appl. Mechanics **01**, 137 (2009)
65. X. Wang, L.T. Zhang, Comput. Mech. **45**(4), 321 (2010)
66. Z. Guo, C. Zheng, B. Shi, Phys. Rev. E **65**, 46308 (2002)
67. S.K. Doddi, P. Bagchi, Int. J. Multiphas. Flow **34**(10), 966 (2008)
68. T. Krüger, F. Varnik, D. Raabe, Comput. Method. Appl. **61**(12), 3485 (2011)
69. G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.J. Jan, J. Comput. Phys. **169**(2), 708 (2001)
70. K. Suzuki, T. Inamuro, Comput. Fluids **49**(1), 173 (2011)
71. D. Nie, J. Lin, Commun. Comput. Phys. **7**(3), 544 (2010)
72. S.K. Doddi, P. Bagchi, Phys. Rev. E **79**(4), 046318 (2009)
73. S. Ramanujan, C. Pozrikidis, J. Fluid Mech. **361**, 117 (1998)
74. Z.G. Feng, E.E. Michaelides, Comput. Fluids **38**(2), 370 (2009)
75. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org
76. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://www.geuz.org/gmsh
77. Y. Peng, L.S. Luo, Prog. Comput. Fluid Dyn. **8**(1), 156 (2008)
78. A. Caiazzo, S. Maddu, Comput. Math. Appl. **58**(5), 930 (2009)
79. L. Zhu, G. He, S. Wang, L. Miller, X. Zhang, Q. You, S. Fang, Comput. Math. Appl. **61**(12), 3506 (2011)
80. Q. Zhou, L.S. Fan, J. Comput. Phys. **268**, 269 (2014)
81. O. Shardt, J.J. Derksen, Int. J. Multiphase Flow **47**, 25 (2012)
82. G. Le, J. Zhang, Phys. Rev. E **79**(2), 026701 (2009)
83. Z.G. Feng, E.E. Michaelides, J. Comput. Phys. **202**(1), 20 (2005)
84. J. Wu, C. Shu, J. Comput. Phys. **228**(6), 1963 (2009)
85. J. Wu, C. Shu, Y.H. Zhang, Int. J. Numer. Meth. Fluids **62**(3), 327 (2010)
86. J. Wu, C. Shu, Int. J. Numer. Meth. Fluids **68**(8), 977 (2012)
87. P. Bagchi, Biophys. J. **92**(6), 1858 (2007)

88. M.M. Dupin, I. Halliday, C.M. Care, L. Alboul, L.L. Munn, Phys. Rev. E **75**(6), 066707 (2007)
89. J. Zhang, P.C. Johnson, A.S. Popel, Phys. Biol. **4**(4), 285 (2007)
90. Y. Sui, Y. Chew, P. Roy, H. Low, J. Comput. Phys. **227**(12), 6351 (2008)
91. T. Krüger, M. Gross, D. Raabe, F. Varnik, Soft Matter **9**(37), 9008 (2013)
92. U.D. Schiller, Comput. Phys. Commun. **185**(10), 2586 (2014)
93. P. Ahlrichs, B. Dünweg, Int. J. Mod. Phys. C **09**(08), 1429 (1998)
94. B. Dünweg, A.J.C. Ladd, in *Advances in Polymer Science* (Springer Berlin Heidelberg, 2008), pp. 1–78
95. I. Cimrák, M. Gusenbauer, I. Jančigová, Comput. Phys. Commun. **185**(3), 900 (2014)
96. X.D. Niu, C. Shu, Y.T. Chew, Y. Peng, Phys. Lett. A **354**(3), 173 (2006)
97. Y. Hu, H. Yuan, S. Shu, X. Niu, M. Li, Comput. Math. Appl. **68**(3), 140 (2014)
98. H.Z. Yuan, X.D. Niu, S. Shu, M. Li, H. Yamaguchi, Comput. Math. Appl. **67**(5), 1039 (2014)
99. J. Wu, C.K. Aidun, Int. J. Numer. Meth. Fl. **62**(7), 765–783 (2009)
100. H.K. Jeong, H.S. Yoon, M.Y. Ha, M. Tsutahara, J. Comput. Phys. **229**(7), 2526 (2010)
101. T. Seta, Phys. Rev. E **87**(6), 063304 (2013)
102. Y. Kim, M.C. Lai, J. Comput. Phys. **229**(12), 4840 (2010)
103. W.X. Huang, C.B. Chang, H.J. Sung, J. Comput. Phys. **230**(12), 5061 (2011)
104. W.X. Huang, C.B. Chang, H.J. Sung, J. Comput. Phys. **231**(8), 3340 (2012)
105. R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, Phys. Rev. E **89**(2), 023303 (2014)

Chapter 12
Sound waves

We previously saw in Chapter 4.1 that the lattice Boltzmann method behaves on the macroscopic scale according to the continuity equation and the *compressible* Navier-Stokes equation. As a natural consequence of this, any fluid phenomenon that is in keeping with these equations can also be captured in a lattice Boltzmann simulation.

One such phenomenon is that of sound waves, which follows from the continuity equation and the Euler equation, the latter of which is an approximated version of the compressible Navier-Stokes equation. Indeed, if we plot the pressure field of an LB simulation right after its startup we can often reveal sound waves caused by the initial conditions of the simulation. Most simulations contain sound waves to some degree or other, even though that may be unintended. For this reason, everyone who performs LB simulations can benefit from knowing a little about sound waves.

Even so, sound in LB simulations has been a neglected subfield compared to *e.g.* incompressible and multiphase/multicomponent flows. The first proper treatment of LB sound waves came as late as 1998 [1]. However, since 2007 several theses have been published on this topic [2, 3, 4, 5, 6, 7, 8, 9], especially on the subtopics of aeroacoustics[1] and boundary conditions that do not reflect sound waves.

This chapter starts with a brief general introduction to acoustics in Section 12.1, as some knowledge on acoustics is required to read the rest of the chapter. Section 12.2 deals with how sound propagates in LB simulations, Section 12.3 covers how sound may appear or be deliberately imposed in simulations, and Section 12.4 introduces boundary conditions that do not reflect sound waves.[2]

[1] Typically in the wider sense of the word *aeroacoustics*, *i.e.* as the interaction of sound and flow.

[2] These non-reflecting BCs are in contrast to pressure and velocity BCs, which reflect sound waves back into the system.

12.1 Background: Sound in viscous fluids

This section will give a quick introduction to where sound waves come from and how they propagate, with the aim of giving readers without a special background in acoustics some required insight for the rest of this chapter.[3] For those readers desiring a deeper insight, there are many books available that all provide a thorough grounding in acoustics [10, 11, 12, 13, 14].

Sound waves follow as a consequence of the conservation equations of a compressible fluid. The scientific field of acoustics is largely based on the linearised Euler equations, which result in a linear, ideal and lossless wave equation. Here we will instead base the discussion on the linearised Navier-Stokes equations, which gives a wave equation with viscous sound absorption.[4] The reason for this slightly more complex choice is that the sound waves that we observe in LB simulations are typically heavily affected by fluid viscosity. Indeed, we will soon see that we will typically need to choose a relaxation time τ very close to 0.5 when simulating sound below the high ultrasonic range. Otherwise, the simulations require an enormous numerical grid, or the sound waves will be absorbed far too quickly.

To linearise the conservation equations, we assume that the field variables of density ρ, pressure p, and velocity $\boldsymbol{u}$ are on the form of an infinitesimal perturbation around a constant rest state,[5]

$$\begin{aligned} \rho(\boldsymbol{x},t) &= \rho_0 + \rho'(\boldsymbol{x},t), \\ p(\boldsymbol{x},t) &= p_0 + p'(\boldsymbol{x},t), \\ \boldsymbol{u}(\boldsymbol{x},t) &= \boldsymbol{0} + \boldsymbol{u}'(\boldsymbol{x},t). \end{aligned} \tag{12.1}$$

Here, the subscripted zeros indicate the rest state constants while the perturbations are indicated as primed quantities.

Inserting these linearised variables into the conservation equation and neglecting terms that are nonlinear in the infinitesimal fluctuations, we end up with

$$\partial_t \rho + \rho_0 \partial_\alpha u_\alpha = 0, \tag{12.2a}$$

$$\rho_0 \partial_t u_\alpha + \partial_\alpha p = \partial_\beta \sigma'_{\alpha\beta}. \tag{12.2b}$$

The viscous stress tensor is unchanged by the linearisation:

$$\sigma'_{\alpha\beta} = \eta \left(\partial_\beta u_\alpha + \partial_\alpha u_\beta - \tfrac{2}{3} \delta_{\alpha\beta} \partial_\gamma u_\gamma \right) + \eta_{\mathrm{B}} \delta_{\alpha\beta} \partial_\gamma u_\gamma. \tag{12.2c}$$

---

[3] Deeper insight in acoustics is often missing in the LB literature, as most LB researchers are more interested in incompressible fluid mechanics than in acoustics.

[4] The basic LB model which gives isothermal Navier-Stokes behaviour is only affected by this viscous sound absorption. It does not allow for absorption due to heat conduction or molecular relaxation, the latter of which dominates at audible frequencies in *e.g.* air and seawater. We will touch on these absorption mechanisms in Section 12.1.5.

[5] An LB model for the linearised conservation equations was discussed briefly in Section 4.3.1.

The equation of state is also very important for sound waves. We use here the isothermal equation of state,

$$p = \rho c_s^2; \tag{12.2d}$$

not only is it exact for the basic LBE, but as we discussed previously in Section 1.1.3, it also works as a linear approximation to other equations of state given that entropy changes are small.

12.1.1 The viscous wave equation

The wave equation is quite simple to derive from these equations in the ideal Euler equation case (*i.e.* $\sigma'_{\alpha\beta} = 0$), and the derivation is only slightly more difficult if we carry along the viscous stress tensor. To begin, we find the sum ∂_t(12.2a)$-\partial_\alpha$(12.2b), which results in

$$\partial_t^2 \rho - \nabla^2 p = -\left(\tfrac{4}{3}\eta + \eta_B\right) \nabla^2 \left(\partial_\alpha u_\alpha\right) = \left(\tfrac{4}{3}\nu + \nu_B\right) \nabla^2 \left(\partial_t \rho\right). \tag{12.3}$$

The simplification of the viscous stress in the last equality uses Eq. (12.2a). In addition, we can assume that the viscosities are nearly constant, with only an infinitesimal variation.

At this point we nearly have the wave equation already. The final piece of the puzzle is the equation of state, Eq. (12.2d), with which we can replace the density ρ with p/c_s^2.[6]

Applying the equation of state in this way, we find the viscous wave equation

$$\frac{1}{c_s^2}\partial_t^2 p - \left(1 + \tau_{vi}\partial_t\right) \nabla^2 p = 0, \tag{12.4a}$$

where the effect of viscosity on sound propagation is expressed through the *viscous relaxation time*

$$\tau_{vi} = \frac{1}{c_s^2}\left(\frac{4}{3}\nu + \nu_B\right). \tag{12.4b}$$

The ideal wave equation, which is used more widely in the literature and is typically derived from the Euler equation, can be found from this simply by assuming $\tau_{vi} \simeq 0$,

$$\frac{1}{c_s^2}\partial_t^2 p - \nabla^2 p = 0. \tag{12.4c}$$

The viscous relaxation time can be seen as a characteristic time for the stabilisation of viscous processes. For physical fluids, it is typically on the order of 10^{-10} s

[6] It is also possible to use the equation of state the other way around, which results in a wave equation with the density as the only remaining variable instead of the pressure.

for gases and 10^{-12} s for liquids [10]. Inserting the LBE viscosities into Eq. (12.4b) using the isothermal bulk viscosity $\eta_B = 2\eta/3$ found in Section 4.1.3, we can relate the viscous relaxation time with the BGK relaxation time as

$$\tau_{vi} = 2(\tau - 1/2). \tag{12.5}$$

Exercise 12.1. (a) Show that the generic one-dimensional function $p(c_s t \mp x)$, known as d'Alembert's solution, is a solution of the ideal wave equation, Eq. (12.4c). *Hint: Insert the solution into the wave equation and apply the chain rule.*
(b) Show that this solution propagates at a speed of $\pm c_s$.

12.1.2 The complex-valued representation of waves

The linearity of the wave equation lets us employ some neat tricks. Steady-state solutions of a single angular frequency ω, related to the wave period T as $\omega = 2\pi/T$, are commonly expressed using complex-valued notation. This complex-valued approach has been widely used in acoustics since Lord Rayleigh introduced it in 1877 [15, 16]. If we know the solution of a particular case for any frequency ω, a corresponding time-domain solution can be found through an inverse Fourier transform [10, 11].

The simplest solution to the wave equation is an infinite plane wave, which is a one-dimensional solution that is not localised in space or time. On complex form, the solution of the ideal (*i.e.* $\tau_{vi} = 0$) wave equation is

$$\hat{p}(x,t) - p_0 = \hat{p}'(x,t) = \hat{p}^\circ \mathrm{e}^{\mathrm{i}(\omega_0 t \mp k_0 x)}. \tag{12.6}$$

In this notation, we indicate complex-valued variables by carets and plane wave amplitudes by circles. As can be found by comparison with Exercise 12.1, the angular frequency ω_0 and the wavenumber k_0 are related as $\omega_0/k_0 = c_s$. (The real-valued frequency and wavenumber for the ideal wave equation are indicated by subscripted zeroes.) The complex amplitude $\hat{p}^\circ$ has both a magnitude $|\hat{p}^\circ|$ and a phase φ_p, *i.e.* $\hat{p}^\circ = |\hat{p}^\circ| \mathrm{e}^{\mathrm{i}\varphi_p}$.

Of course, the physical pressure is not complex. It can be found directly from the real part of the complex-valued pressure,

$$p'(x,t) = \Re\left(\hat{p}'(x,t)\right) = \left|\hat{p}^\circ\right| \cos\left(\omega_0 t \mp k_0 x + \varphi_p\right). \tag{12.7}$$

The motivation for using complex notation is that these complex quantities are much easier to handle than the corresponding real expressions. We can augment an originally real-valued field with an imaginary part, perform calculations on the result, and retrieve the real part of the result, because the wave equation is *linear*,

and taking the real part is also a linear operation. These operations commute. On the other hand, we must be careful when using the complex notation in nonlinear equations. As an example, $\Re(\hat{p}^2) \neq (\Re(\hat{p}))^2$.

We can also simulate complex quantities directly in LB simulations by letting artificial sound sources (which will be described later in Section 12.3) emit complex-valued distribution functions. However, this should only be used with the linearised equilibrium described earlier in Section 4.3.1 as linearity is a prerequisite for the complex wave notation. This approach has been used to simplify the analysis of sound wave simulations [17].

Exercise 12.2. Show by insertion that Eq. (12.6) and Eq. (12.7) are both valid solutions of the ideal wave equation, Eq. (12.4c).

Exercise 12.3. Show that the plane wave expressed in Eq. (12.6) has a *period* (i.e. the time before the wave repeats itself) of $T = 2\pi/\omega_0$ and a *wavelength* (i.e. the distance between successive wave peaks) of $\lambda = 2\pi/k_0$, and that it propagates at the speed of sound $c_\mathrm{s} = \omega_0/k_0$.

The corresponding plane wave solution to the viscous wave equation (*i.e.* for $\tau_\mathrm{vi} \neq 0$) is similar, but has a complex wavenumber $\hat{k}$ and/or frequency $\hat{\omega}$. For a wave propagating in the $+x$ direction in a viscous fluid, a plane wave looks like

$$\hat{p}'(x,t) = \hat{p}^\circ \mathrm{e}^{\mathrm{i}\left(\hat{\omega}t - \hat{k}x\right)} = \hat{p}^\circ \mathrm{e}^{-\alpha_t t} \mathrm{e}^{-\alpha_x x} \mathrm{e}^{\mathrm{i}(\omega t - kx)}. \tag{12.8}$$

Here, the complex wavenumber and frequency have been split into their real and imaginary parts,

$$\hat{\omega} = \omega + \mathrm{i}\alpha_t, \quad \hat{k} = k - \mathrm{i}\alpha_x. \tag{12.9}$$

The αs represent attenuation coefficients, while the phase speed of the sound wave is given by $c_\mathrm{p} = \omega/k$, which may differ from the ideal sound speed $c_\mathrm{s} = \omega_0/k_0$. This means that sound propagation in viscous fluids is dispersive: the discrepancy between c_p and c_s increases with frequency and τ_vi, as found in Exercise 12.4.

Wave dispersion is often seen in LB simulations; unintended sound waves usually move at a speed noticably different from c_s. However, the dispersion in LB simulations is also partly due to numerical error, and the rest of the dispersion differs from what is predicted by the Navier-Stokes equation, as we will see in Section 12.2.

Exercise 12.4. (a) Show by insertion that Eq. (12.8) being a solution of Eq. (12.4a) implies a *dispersion relation* that relates the frequency and wavenumber as

$$\frac{\hat{\omega}^2}{\hat{k}^2} = c_\mathrm{s}^2 \left(1 + \mathrm{i}\hat{\omega}\tau_\mathrm{vi}\right). \tag{12.10}$$

(b) In which limits can this dispersion relation become the ideal dispersion relation $c_\mathrm{s} = \omega_0/k_0$?

12.1.3 Simple one-dimensional solutions: Free and forced waves

Typically, only the frequency or the wavenumber is complex for an attenuated sound wave. For instance, let us take the semi-infinite system $0 \leq x$. If the boundary condition at $x = 0$ is $\hat{p}'(0,t) = \hat{p}^\circ e^{i\omega_0 t}$, *i.e.* the pressure oscillates with a frequency ω_0 and constant amplitude $|\hat{p}^\circ|$, the resulting wave in the rest of the system must have the same frequency $\hat{\omega} = \omega_0$,

$$\hat{p}'(x,t) = \hat{p}^\circ e^{-\alpha_x x} e^{i(\omega_0 t - kx)}.$$

Following the nomenclature of Truesdell [18], this is a *forced wave*. In general, forced waves are radiated (or *forced*) by a source oscillating at a constant amplitude.

Forced waves are an idealisation of physically attainable waves. For instance, the wave above could be a high-frequency sound wave in a duct (neglecting the marginal effect of no-slip boundaries).

Using the same nomenclature, a *free wave* is a wave which is not emitted by any source, but which is absorbed while it propagates. Let us say that at $t = 0$ we have a plane wave of constant amplitude and infinite extent, *i.e.* $\hat{p}'(x,0) = \hat{p}^\circ e^{-ik_0 x}$. For such a case, the resulting plane wave is

$$\hat{p}'(x,t) = \hat{p}^\circ e^{-\alpha_t t} e^{i(\omega t - k_0 x)}. \tag{12.11}$$

Free waves are not as relevant or even physically realisable in the same way as forced waves. Even so, they may serve as useful simulation benchmarks since their periodicity in space can be captured perfectly using a periodic boundary condition.[7]

Exercise 12.5. (a) Show from Eq. (12.10) that the complex wavenumber for the forced plane wave and the complex frequency for the free plane wave are respectively

$$\frac{\hat{k}}{k_0} = \frac{1}{\sqrt{1 + i\omega_0 \tau_{vi}}}, \tag{12.12a}$$

$$\frac{\hat{\omega}}{\omega_0} = i\frac{\omega_0 \tau_{vi}}{2} + \sqrt{1 - \left(\frac{\omega_0 \tau_{vi}}{2}\right)^2}, \tag{12.12b}$$

with k_0 and ω_0 related through $c_s = \omega_0 / k_0$.

(b) Show from Taylor expansion of these equations that the nondimensionalised absorption and dispersion for forced and free waves are, to lowest order,

$$\text{Forced:} \qquad \frac{\alpha_x}{k_0} = \frac{\omega_0 \tau_{vi}}{2}, \qquad \frac{c_p}{c_s} = \frac{k_0}{k} = 1 + \frac{3}{8}(\omega_0 \tau_{vi})^2, \tag{12.13a}$$

$$\text{Free:} \qquad \frac{\alpha_t}{\omega_0} = \frac{\omega_0 \tau_{vi}}{2}, \qquad \frac{c_p}{c_s} = \frac{\omega}{\omega_0} = 1 - \frac{1}{8}(\omega_0 \tau_{vi})^2. \tag{12.13b}$$

[7] Free waves are also key in the von Neumann stability analyses described in Section 4.4.

Generally, the dimensionless number $\omega_0\tau_{\mathrm{vi}}$, which we can call the *acoustic viscosity number*, indicates the effect of viscosity on sound propagation at a given frequency. The higher the value of $\omega_0\tau_{\mathrm{vi}}$, the higher the degree of viscous absorption and dispersion.

Exercise 12.6. Let us take a look at how the acoustic viscosity number scales. We know that $\omega_0 \sim c_{\mathrm{s}}/\lambda$ and that $\nu \sim \nu_{\mathrm{B}} \sim c_{\mathrm{s}}\ell_{\mathrm{mfp}}$. From this, show that

$$\omega_0\tau_{\mathrm{vi}} \sim \frac{\ell_{\mathrm{mfp}}}{\lambda}, \tag{12.14}$$

i.e. that the acoustic viscosity number represents an acoustic Knudsen number.

From Eq. (12.13) we see that for both forced and free waves, the absorption coefficient (α_x and α_t, respectively) to lowest-order increases with the square of the frequency ω_0. (Due to the typically low values of $\omega_0\tau_{\mathrm{vi}}$, the lowest order is typically the only one felt.) However, free and forced waves experience different dispersion according to the isothermal Navier-Stokes; the wave speed *increases* with frequency for *forced* waves, and *decreases* with frequency for *free* waves.

The absorption and dispersion given above are derived from the Navier-Stokes equation, which can be seen as a continuum approximation to the Boltzmann equation. Higher-order fluid models such as the Burnett equation generally predict the same viscous absorption to lowest order, but make different predictions for terms of higher order in $\omega_0\tau_{\mathrm{vi}}$ [19, 7]. One thing that generally holds for all these fluid models is that in a series expansion of $\hat{k}/k_0$ or $\hat{\omega}/\omega_0$ odd powers in $\omega_0\tau_{\mathrm{vi}}$ are related to absorption while even powers are related to dispersion.

Let's take a quick look at typical values of $\omega_0\tau_{\mathrm{vi}}$ in everyday cases. At everyday conditions, the ideal speed of sound in air is around $c_{\mathrm{s}} \simeq 340\,\mathrm{m/s}$, the kinematic shear viscosity is $\nu \simeq 1.5 \times 10^{-5}\,\mathrm{m^2/s}$ and the bulk viscosity is $\nu_{\mathrm{B}} \simeq 0.61\nu$ [14, 13, 11]. For a high but audible frequency of 10 kHz, we get an acoustic viscosity number of $\omega_0\tau_{\mathrm{vi}} = 1.6 \times 10^{-5}$. From Chapter 7, this dimensionless number will be the same in any system of units.

Now, what would it require to achieve such a number in lattice units? Just to have a number, let's assume a reasonable lattice wavelength of $\lambda = 10$, which corresponds to an angular frequency of $\omega_0 = 2\pi c_{\mathrm{s}}/\lambda \approx 0.36$. Relating τ_{vi} to τ using Eq. (12.5), we find that this acoustic viscosity number, given $\lambda = 10$, requires $\tau = 0.500022$. This incredibly low number for τ is hard to achieve. In principle it can be increased by increasing the lattice wavelength, but that is equivalent with increasing the resolution which may lead to an unfeasibly large simulation domain. On the other hand, it has been shown possible to simulate at least *some* acoustic phenomena for the inviscid case of $\tau = 0.5$ with both accuracy and stability using MRT or MRT-like collisions [7, 17]. Generally, low values of τ are very important for LB acoustics.

12.1.4 Time-harmonic waves: the Helmholtz equation

The wave equation can be simplified considerably in the steady-state case where the entire field is harmonically varying in time, which means that the field varies with time as $e^{i\omega_0 t}$. Such a time variance allows the simplifying substitution $\partial_t \rightarrow i\omega_0$. Consequently, the viscous wave equation, Eq. (12.4a), can be reduced to the viscous Helmholtz equation

$$\nabla^2 \hat{p}' + \hat{k}^2 \hat{p}' = 0, \tag{12.15}$$

with the wavenumber $\hat{k} = k_0/\sqrt{1 + i\omega_0 \tau_{vi}}$. (In the more commonly seen inviscid case, $\tau_{vi} \rightarrow 0$ and the wavenumber is real, *i.e.* $\hat{k}^2 = k_0^2$.)

The Helmholtz equation holds universally for a harmonic time variation no matter the spatial shape of the wave; it may be plane, cylindrical, spherical, a sound beam, or any other shape, and the complex wavenumber $\hat{k}$ remains the same. This shows that the wavenumber of a forced wave equals that of any viscously affected wave.

The Helmholtz equation is a steady-state equation and time does not enter into it except through the underlying assumption that the fields vary as $e^{i\omega_0 t}$. On the other hand, LB simulations are inherently time-dependent. This means that LB simulations can only approximate Helmholtz equation solutions, and then only in cases with time-harmonic sound sources that are left to radiate sound until a near-steady state is reached.

12.1.5 Other attenuation and absorption mechanisms

In isothermal lattice Boltzmann simulations, shear and bulk viscosity are typically the only causes of sound wave attenuation. However, in real fluids, there are other mechanisms that cause attenuation and absorption of sound waves, both in free fields and for wave-surface interaction. We will not go into these mechanisms in too much depth here; they are described more thoroughly in the acoustics literature [11, 10, 13, 7].

One free-field mechanism is heat conduction, which does not appear in isothermal LB fluids. In short, sound wave peaks experience an increase not only in pressure and density, but also in temperature. Sound wave troughs experience a corresponding *decrease* in temperature. This temperature inhomogenity results in heat conduction. This reduces the temperature differences, and consequently the pressure differences, between peaks and troughs; the wave amplitude decreases.

Heat conduction causes a wave attenuation which, similarly to the viscous attenuation, is proportional to the square of the frequency. In gases, this heat conduction typically causes an attenuation of similar strength to that from viscosity. In liquids, attenuation from heat conduction is typically negligible. For free-field attenuation, the importance of viscosity compared to heat conduction scales with the Prandtl number as $O(\mathrm{Pr})$ [10].

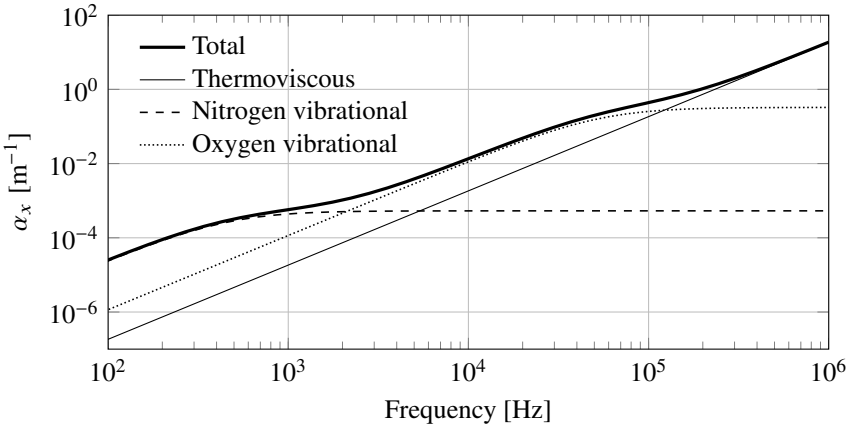


Fig. 12.1: Contributions to the total sound attenuation for air at conditions of 293.15 K, atmospheric pressure, and 70 % relative humidity [11], according to ISO 9613-1:1993.

Additionally, when a sound wave interacts with a solid boundary, the viscous friction and heat transfer between the two will also cause a partial absorption of the sound wave, in particular at high frequencies [13, 10].

Another free-field mechanism is molecular relaxation processes, which is linked to transfer of energy between various forms. In short, a passing sound wave will temporarily change the temperature of the fluid. As we saw in section 1.3, the temperature is directly proportional to the translational energy of the fluid molecules. Increasing the translational energy in this way puts it out of equilibrium with inner forms of energy in the fluid. In a polyatomic gas, these are the vibrational and rotational energies of the molecules. In seawater, the increase in temperature changes the equilibrium point of a chemical reaction between solutes in the water. In both cases, translational energy is equilibrated with these inner energies through a relaxation process. This reduces the temperature differences (and consequently the pressure differences) between peaks and troughs, weakening the pressure wave as it propagates.

At audible frequencies, this tends to completely dominate viscous absorption in typically encountered fluids such as air or seawater. However, this mechanism is dependent on the rate at which translational energy is converted to inner energies and back.[8] Therefore, it is less effective at very high frequencies where the inner energies cannot keep up and remain nearly constant, or *frozen*.

[8] These rates can depend on the presence of other molecules. For instance, water molecules act as a catalyst for the transfer between translational and vibrational energy in air. Humidity therefore has a surprisingly large effect on sound absorption in air.

The relative importance of vibrational energy to sound attenuation in air, compared to a combined attenuation due to viscosity and heat conduction, is shown in Fig. 12.1. In the audible range, the thermoviscous attenuation is typically *negligible*.

The most important source of sound absorption in day-to-day situations, however, is sound waves' interaction with surfaces that are not perfectly rigid. This can be represented through the surface property of *normal impedance* $\hat{Z}_n = \hat{p}/\hat{u}_n$, where $\hat{u}_n$ is the normal velocity of the surface for an applied pressure $\hat{p}$ [10]. Higher impedance, corresponding to a harder and less moveable surface, typically results in the surface absorbing less sound. For instance, a room with hard concrete walls will be more reverberant (*i.e.*, sound will linger for a longer time) than a similar room with softer wooden walls.

Now, how can these mechanisms be captured in LB simulations? Thermal attenuation is something we get for free in thermal LB models, and the attenuation in such LB simulations matches theory [20]. Vibrational relaxation, on the other hand, is a more complex effect that cannot be captured so easily. The only published attempt at this time of writing was very simple and limited itself to a single relaxation process and monofrequency sound. It achieved good accuracy but poor stability [7]. Similarly, only a little has been published on impedance BCs for the LBM [21].

12.1.6 Simple multidimensional waves: The Green's function

The wave and Helmholtz equations discussed above only describe how *existing* waves propagate, not how they are *generated*. One simple mathematical description of sound wave generation is given through *Green's functions* $\hat{G}(\boldsymbol{x}, t)$. In general, a Green's function is the response to a delta function inhomogenity in a partial differential equation.

For the viscous wave equation in Eq. (12.4a), the time-harmonic Green's function is defined through

$$\left(\frac{1}{c_\mathrm{s}^2}\partial_t^2 - (1 + \tau_\mathrm{vi}\partial_t)\nabla^2\right)\hat{G}(\boldsymbol{x}, t) = \delta(\boldsymbol{x})\,\mathrm{e}^{\mathrm{i}\omega_0 t}, \tag{12.16}$$

$\delta(\boldsymbol{x})$ being the Dirac delta function. We'll see how such inhomogenities can appear in the wave equation in Section 12.3.

The solution to this inhomogeneous equation depends on the number of spatial dimensions and represents the simplest type of wave for that number of dimensions. For a one-dimensional space, the Green's function is a plane wave. For two-dimensional space, it is a cylindrical wave. For three-dimensional space, it is a spherical wave. Explicitly, these solutions are [12]

$$\text{1D:} \qquad \hat{G}(x,t) = \frac{1}{2\mathrm{i}\hat{k}} \mathrm{e}^{\mathrm{i}(\omega_0 t - \hat{k}|x|)}, \tag{12.17a}$$

$$\text{2D:} \qquad \hat{G}(\boldsymbol{x},t) = \frac{1}{4\mathrm{i}} H_0^{(2)}(\hat{k}|\boldsymbol{x}|)\, \mathrm{e}^{\mathrm{i}\omega_0 t}, \tag{12.17b}$$

$$\text{3D:} \qquad \hat{G}(\boldsymbol{x},t) = \frac{1}{4\pi|\boldsymbol{x}|} \mathrm{e}^{\mathrm{i}(\omega_0 t - \hat{k}|\boldsymbol{x}|)}. \tag{12.17c}$$

In these equations $|\boldsymbol{x}|$ is the distance to the origin (often denoted as r in the literature), and $H_n^{(2)}$ is an n-th order Hankel function of the second kind, which is a particular superposition of Bessel functions.

It is also possible to define time-impulsive Green's functions through the alternate inhomogenity term $\delta(\boldsymbol{x})\delta(t)$. However, these have no similarly simple analytical solutions for two dimensions or for sound waves affected by viscosity. In any case, such time-impulsive Green's functions can be indirectly found through an inverse Fourier transformation of time-harmonic Green's functions.

Knowing the response of a point inhomogenity also lets us find the pressure field from a spatially distributed inhomogenity, meaning that the equation

$$\left(\frac{1}{c_\mathrm{s}^2}\partial_t^2 - (1 + \tau_\mathrm{vi}\partial_t)\nabla^2\right)\hat{p}(\boldsymbol{x},t) = \hat{\mathcal{T}}(\boldsymbol{x})\, \mathrm{e}^{\mathrm{i}\omega_0 t}, \tag{12.18a}$$

is solved by

$$\hat{p}'(\boldsymbol{x},t) = \int \hat{\mathcal{T}}(\boldsymbol{y})\, \hat{G}(\boldsymbol{x}-\boldsymbol{y},t)\, \mathrm{d}^3 y, \tag{12.18b}$$

where $\boldsymbol{x}$ is the 'receiver' point and we integrate over all possible 'source' points $\boldsymbol{y}$.

These delta function inhomogenities radiate waves equally in every direction, which means that they are *monopole* sources. However, inhomogenities that radiate directively, for instance as *dipoles* and *quadrupoles*, are also possible. Such inhomogenities are represented as tensor terms with spatial derivatives applied to them [22, 23, 7], *i.e.* as

$$\begin{aligned}\left(\frac{1}{c_\mathrm{s}^2}\partial_t^2 - (1 + \tau_\mathrm{vi}\partial_t)\nabla^2\right)\hat{p}(\boldsymbol{x},t) = \mathrm{e}^{\mathrm{i}\omega_0 t}\Big[&\hat{\mathcal{T}}(\boldsymbol{x}) + \partial_\alpha \hat{\mathcal{T}}_\alpha(\boldsymbol{x}) \\ &+ \partial_\alpha\partial_\beta \hat{\mathcal{T}}_{\alpha\beta}(\boldsymbol{x})\Big].\end{aligned} \tag{12.19a}$$

It can be shown [22, 23, 7] that this equation is solved by

$$\begin{aligned}\hat{p}'(\boldsymbol{x},t) = \int \Big[&\hat{\mathcal{T}}(\boldsymbol{y})\hat{G}(\boldsymbol{x}-\boldsymbol{y},t) + \hat{\mathcal{T}}_\alpha(\boldsymbol{y})\partial_\alpha \hat{G}(\boldsymbol{x}-\boldsymbol{y},t) \\ &+ \hat{\mathcal{T}}_{\alpha\beta}(\boldsymbol{y})\partial_\alpha\partial_\beta \hat{G}(\boldsymbol{x}-\boldsymbol{y},t)\Big]\, \mathrm{d}^3 y,\end{aligned} \tag{12.19b}$$

where the spatial derivatives in the integrand are always operating on $\boldsymbol{x}$. The first spatial derivative of the Green's function corresponds to dipole radiation and the second spatial derivative of the Green's function to quadrupole radiation.

Exercise 12.7. (a) Using the 3D Green's function in Eq. (12.17c), show for the far field (*i.e.* for large $|\boldsymbol{x}|$), that $\partial_\alpha G(\boldsymbol{x}, t)$ has a dipole variation in space, *i.e.* that it varies as $x_\alpha/|\boldsymbol{x}|$.
(b) Similarly, show that $\partial_\alpha \partial_\beta G(\boldsymbol{x}, t)$ has a quadrupole variation in space, *i.e.* that it varies as $x_\alpha x_\beta/|\boldsymbol{x}|^2$.

12.2 Sound propagation in LB simulations

If we would look at the sound waves that typically appear by accident in LB simulations, one of the first two things we would notice is that they are attenuated fairly quickly and that they do not propagate at a speed c_s. For the latter point, we already established in Section 12.1.3 that waves in a viscous medium are dispersive at high frequencies.[9]

Additionally, sound waves do not propagate in numerical solvers exactly as predicted by the Navier-Stokes equations. There are several reasons for this:

- Due to discretisation error, waves (or Fourier components of waves) that are more heavily discretised, meaning that they have fewer points per wavelength, will propagate differently to well-resolved waves.
- Numerical solvers typically discretise space into elements or a grid, and it is not possible to discretise space in a fully isotropic way. Therefore, sound propagation is typically slightly anisotropic, again especially at short wavelengths.

These points hold fully for the lattice Boltzmann method, though there are some LB-specific issues as well:

- The LBM discretises velocity space, and it is not possible to do this fully isotropically. Even without the discretisation in space and time, this makes sound propagation anisotropic, though this effect is only significant for $\omega_0 \tau_{\mathrm{vi}} > 0.1$ [26].
- At second order and higher in series expansions in $\omega_0 \tau_{\mathrm{vi}}$, the Boltzmann equation predicts different sound propagation to the Navier-Stokes equations [19]. It is similar for the LBE: To $O(\omega_0 \tau_{\mathrm{vi}})$ the absorption is the

[9] For strong sound waves nonlinear effects also affect the local wave speed. These nonlinear effects can be reproduced in LB simulations [24, 25], though we will not go into nonlinear sound in this book.

same as for Navier-Stokes, but the dispersion (which to lowest order is $O(\omega_0\tau_{\mathrm{vi}})^2$) is different.[10]

Consequently, the sound absorption and dispersion in LB simulations depends both on the physical model (*i.e.* the discrete-velocity Boltzmann equation) and on the purely numerical discretisation. As we will see in the following sections, it is possible to tell these effects apart. Additionally, the anisotropy of each velocity set may also uniquely affect LB sound propagation.

One classic method for analysing the sound propagation predicted by fluid models goes back to Stokes' first article on his fluid momentum equation that we now know as the Navier-Stokes equation [27]. In this method, sometimes known as *linearisation analysis*, we assume that the unknowns (*i.e.* density, pressure, and fluid velocity) are varying on the form of an infinitesimal plane wave around a rest state, *e.g.* $\hat{p} = p_0 + \hat{p}^\circ \mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)}$. In the fluid model's governing partial differential equations, space and time derivatives can be applied directly to these unknown quantities since their behaviour in space and time is known. From the resulting derivative-free equations, dispersion relations such as Eq. (12.10) which relate the frequency $\hat{\omega}$ and wavenumber $\hat{k}$ can be found.

Such linearisation analyses have also been applied to the Boltzmann equation [28, 29, 30]. However, they were not able to determine a closed-form dispersion relation, instead ending up with a non-convergent power series in quantities of $O(\omega_0\tau_{\mathrm{vi}})$. Thankfully, the finite velocity space of the discrete-velocity Boltzmann equation and the lattice Boltzmann method make them more amenable to this kind of analysis.

12.2.1 Linearisation method

To analyse LB sound propagation, a linearisation analysis similar to the von Neumann analysis mentioned in Section 4.4 can be used. Similarly to what we saw in Section 12.1.2, the unknowns f_i are assumed to be varying on plane wave form. However, exact space and time derivatives do not exist in any numerical solver with discretised space and time, which complicates the analysis somewhat. Instead, we must relate the unknowns f_i over several nodes and time steps.

A number of such analyses have been performed in the literature [31, 32, 33, 34, 35, 36, 37, 26, 25, 7]. However, the most thorough and detailed analyses have unfortunately only been performed for free waves. As discussed in Section 12.1.3, free waves are not usually relevant in practice beyond benchmarks and stability analyses, and knowing how free waves behave is not sufficient to know how forced waves behave [18].

In this section we will go through a simple linearisation analysis of sound propagation for the D1Q3 velocity set given in Tab. 3.2. This will result in expressions for

the absorption and dispersion of sound in the LBM. This analysis does not take into account the anisotropy of each higher-dimensional velocity set. Even so, the end results hold for propagation along the x, y, or z axis in higher-dimensional velocity sets of which D1Q3 is a one-dimensional projection as discussed in Section 3.3.7.

The process behind this analysis, which will now be presented, is not really necessary to understand the results, which are presented in Section 12.2.2.

In this linearisation analysis we assume that the solution is an infinitesimally weak plane sound wave on top of a rest state. Consequently, we can use a linearised equilibrium like in Section 4.3.1,

$$f_i^{\mathrm{eq}} = w_i \left(\rho_0 + \hat{\rho}' + \frac{\rho_0 \hat{u}'}{c_\mathrm{s}^2} c_i \right). \tag{12.20}$$

Since we are basing this analysis on the one-dimensional D1Q3 velocity set, this linearised equilibrium distribution is also one-dimensional.

We define the distribution function $\{f_i\}$ to be split into two parts:

$$\hat{f}_i(x,t) = F_i^{\mathrm{eq}} + \hat{f}_i'(x,t) = F_i^{\mathrm{eq}} + \hat{f}_i^{\circ} \mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)}. \tag{12.21}$$

Here, we define F_i^{eq} to be the quiescent (*i.e.* rest state) component of the distribution function. $\hat{f}_i'$ is defined to be the infinitesimal plane-wave disturbance on top, and $\hat{f}_i^{\circ}$ is the plane-wave amplitude. As the F_i^{eq} component is constant and unaffected by the fluctuations $\hat{f}_i'$ due to linearity, it remains in a permanent state of both macroscopic and mesoscopic equilibrium. Mathematically, these definitions can be expressed as

$$\begin{aligned} \begin{bmatrix} \sum_i \hat{f}_i(x,t) \\ \sum_i c_i \hat{f}_i(x,t) \end{bmatrix} &= \begin{bmatrix} \sum_i F_i^{\mathrm{eq}} \\ \sum_i c_i F_i^{\mathrm{eq}} \end{bmatrix} + \begin{bmatrix} \sum_i \hat{f}_i^{\circ}(x,t) \\ \sum_i c_i \hat{f}_i^{\circ}(x,t) \end{bmatrix} \mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)} \\ &= \begin{bmatrix} \hat{\rho}(x,t) \\ \rho_0 \hat{u}(x,t) \end{bmatrix} = \begin{bmatrix} \rho_0 \\ 0 \end{bmatrix} + \begin{bmatrix} \hat{\rho}^{\circ} \\ \rho_0 \hat{u}^{\circ} \end{bmatrix} \mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)}. \end{aligned} \tag{12.22}$$

The distribution function $\{f_i\}$ has the same zeroth and first moments as its corresponding equilibrium $\{f_i^{\mathrm{eq}}\}$. Therefore, the linearised equilibrium in Eq. (12.20) can similarly be split into a quiescent component and a plane wave fluctuation component,

$$F_i^{\mathrm{eq}} = \rho_0 w_i, \qquad \hat{f}_i^{\circ \mathrm{eq}} = w_i \left(\hat{\rho}^{\circ} + \frac{\rho_0 \hat{u}^{\circ}}{c_\mathrm{s}^2} c_i \right). \tag{12.23}$$

Inserting this split equilibrium distribution into the LBE, the quiescent component is cancelled out everywhere and we are left with an LBE only for the fluctuation itself,

$$\hat{f}_i'(x + c_i, t + 1) = \left(1 - \tfrac{1}{\tau}\right) \hat{f}_i'(x,t) + \tfrac{1}{\tau} \hat{f}_i'^{\mathrm{eq}}(x,t).$$

From the above definitions we can perform the substitution $\hat{f}_i'(x,t) = \hat{f}_i^{\circ} \mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)}$. Subsequently explicitly expanding the moments in the equilibrium distribution and dividing the resulting LBE by $\mathrm{e}^{\mathrm{i}(\hat{\omega}t - \hat{k}x)}$, we find

$$\hat{f}_i^\circ \mathrm{e}^{\mathrm{i}(\hat{\omega}-\hat{k}c_i)} = \left(1 - \tfrac{1}{\tau}\right) \hat{f}_i^\circ + \frac{w_i}{\tau}\left[\hat{f}_1^\circ + \hat{f}_0^\circ + \hat{f}_2^\circ + 3\left(\hat{f}_1^\circ - \hat{f}_2^\circ\right) c_i\right]. \tag{12.24}$$

This corresponds to three explicit equations, one for each i. These relate the three unknown f_is with each other and with $\hat{\omega}$ and $\hat{k}$. After some algebra, these can be put in the form of an eigenvalue problem $\hat{\boldsymbol{A}}\boldsymbol{f}^\circ = \mathrm{e}^{\mathrm{i}\hat{\omega}}\boldsymbol{f}^\circ$, which in explicit form is

$$\hat{\boldsymbol{A}}\hat{\boldsymbol{f}}^\circ = \frac{1}{3}\begin{bmatrix} \mathrm{e}^{-\mathrm{i}\hat{k}}(3-1/\tau) & \mathrm{e}^{-\mathrm{i}\hat{k}}/2\tau & -\mathrm{e}^{-\mathrm{i}\hat{k}}/\tau \\ 2/\tau & 3-1/\tau & 2/\tau \\ -\mathrm{e}^{\mathrm{i}\hat{k}}/\tau & \mathrm{e}^{\mathrm{i}\hat{k}}/2\tau & \mathrm{e}^{\mathrm{i}\hat{k}}(3-1/\tau) \end{bmatrix} \begin{bmatrix} \hat{f}_2^\circ \\ \hat{f}_0^\circ \\ \hat{f}_1^\circ \end{bmatrix} = \mathrm{e}^{\mathrm{i}\hat{\omega}} \begin{bmatrix} \hat{f}_2^\circ \\ \hat{f}_0^\circ \\ \hat{f}_1^\circ \end{bmatrix}. \tag{12.25}$$

Now, let us remember what we are really looking for here: a dispersion relation that relates the unknowns $\hat{\omega}$ and $\hat{k}$. This can be found from the characteristic polynomial of the above eigenvalue problem,

$$\det\left(\hat{\boldsymbol{A}} - \boldsymbol{I}\mathrm{e}^{\mathrm{i}\hat{\omega}}\right) = g(\hat{\omega}, \hat{k}, \tau) = 0. \tag{12.26}$$

Here, $g(\hat{\omega}, \hat{k}, \tau) = 0$ is the analytical dispersion relation that is our goal. However, this relation is very cumbersome; it should not be dealt with by hand, nor will it be reproduced explicitly here.

12.2.2 Linearisation results

With the help of a computer algebra system, the dispersion relation in Eq. (12.26) can be solved for either $\hat{k}$ or $\hat{\omega}$, given some assumption on the relation between the two. In particular, we can determine exactly how D1Q3 waves propagate for the aforementioned forced and free wave cases, where the frequency or the wavenumber are real-valued, respectively.

Forced waves

For forced waves, the frequency is real-valued, *i.e.* $\hat{\omega} = \omega_0$, and we can solve the dispersion relation to find an exact, though extremely unwieldy, analytical wavenumber $\hat{k}$. There are two solutions for $\hat{k}$, corresponding to propagation in the $+x$ and the $-x$ direction. The $+x$-propagating solution is:

$$\hat{k} = \mathrm{i}\ln\left[\frac{3\tau(\zeta^2 - \zeta + 1 - \zeta^{-1}) + \zeta - 2 + \zeta^{-1}(3 + \sqrt{3\Xi})}{4 + 6\tau(\zeta - 1) - 2\zeta}\right], \tag{12.27}$$

where the shorthands $\zeta = \mathrm{e}^{\mathrm{i}\omega_0}$ and

$$\Xi = (\zeta + 1)(\zeta - 1)^2(\tau\zeta + 1 - \tau)(3\tau\zeta^2 - \zeta + 3 - 3\tau)$$

have been used.

While this solution is exact for D1Q3 and a good approximation[11] for those velocity sets that can be projected to D1Q3, it is a mess to look at. It is difficult to get any feeling from this solution for how the wavenumber behaves.

Instead, we can find something simpler and clearer by series expanding this in $k_0 = \omega_0/c_s$. By cleverly arranging the series expansion we find

$$\begin{aligned}\frac{\hat{k}}{k_0} = &\left[1 + \tfrac{1}{12}\omega_0^2 + \tfrac{13}{480}\omega_0^4 + O(\omega_0^6)\right] - \mathrm{i}\tfrac{1}{2}(\omega_0\tau_{\mathrm{vi}})\left[1 + \tfrac{5}{12}\omega_0^2 + O(\omega_0^4)\right] \\ &- \tfrac{5}{8}(\omega_0\tau_{\mathrm{vi}})^2\left[1 + \tfrac{13}{20}\omega_0^2 + O(\omega_0^4)\right] + \mathrm{i}\tfrac{13}{16}(\omega_0\tau_{\mathrm{vi}})^3\left[1 + O(\omega_0^2)\right] \\ &+ O([\omega_0\tau_{\mathrm{vi}}]^4).\end{aligned} \tag{12.28}$$

This arrangement nests a series expansion in ω_0, which here represents the numerical resolution, in a series expansion in the dimensionless acoustic viscosity number $\omega_0\tau_{\mathrm{vi}}$, which represents the physical effect of viscosity on sound propagation in the model. From Eq. (12.9) we know that the imaginary terms represent sound wave attenuation while the real terms represent dispersion.

To make the expansion more clear, consider letting $\omega_0 \to 0$ while keeping $\omega_0\tau_{\mathrm{vi}}$ constant. This is equivalent to having an infinitely refined resolution for sound waves. This limit removes the discretisation error in space and time, resulting in the same wave propagation as for the discrete-velocity Boltzmann equation [26]. Conversely, letting $\omega_0\tau_{\mathrm{vi}} \to 0$ while keeping ω_0 finite is equivalent to ideal sound propagation unaffected by dissipative effects. Only the discretisation error in space and time remains.

Free waves

For free waves, the wavenumber is real-valued, *i.e.* $\hat{k} = k_0$, and we can solve the dispersion relation for the unknown $\hat{\omega}$. This actually admits three solutions [7]: two for plane wave propagation in each direction and one non-propagating dissipiative mode. All three solutions are significantly more cumbersome than even Eq. (12.27), and we will jump straight to the approximate series expansion form,

$$\begin{aligned}\frac{\hat{\omega}}{\omega_0} = &\left[1 - \tfrac{1}{36}k_0^2 - \tfrac{1}{1440}k_0^4 + O(k_0^6)\right] + \mathrm{i}\tfrac{1}{2}(\omega_0\tau_{\mathrm{vi}})\left[1 + O(k_0^4)\right] \\ &+ \tfrac{1}{8}(\omega_0\tau_{\mathrm{vi}})^2\left[1 - \tfrac{1}{4}k_0^2 + O(k_0^4)\right] + \mathrm{i}\tfrac{1}{8}(\omega_0\tau_{\mathrm{vi}})^3\left[1 + O(k_0^2)\right] \\ &+ O([\omega_0\tau_{\mathrm{vi}}]^4).\end{aligned} \tag{12.29}$$

In this case, $\omega_0\tau_{\mathrm{vi}}$ still represents the physical effect of viscosity in the model while numerical resolution is represented by k_0.

[11] Again, it is exact if we disregard anisotropy.

Discussion

For both forced and free waves, we've seen that the time and space discretisation error first appears at second order in the parameter describing the numerical resolution. This shows that LB sound wave propagation is second order accurate.

The results presented from this analysis are highly dependent on the specifics of the LBE. Take for example the model described in Section 4.3.3, where the equilibrium f_i^{eq} can be altered in order to change the equation of state. A linearisation analysis of this model found that changing the equilibrium affects every order of the time and space discretisation error [25]. Additionally, this change affects the continuous model, represented by $\omega_0\tau_{\mathrm{vi}}$ for $\omega_0 \to 0$, at $O(\omega_0\tau_{\mathrm{vi}})^2$ and presumably also higher orders. Similar results have also been reported elsewhere in the literature [32, 35].

On the other hand, the results from this derivation accurately describe the wave propagation for the classic isothermal equilibrium, even for non-plane waves. This will be demonstrated in Section 12.3 for a case with 2D cylindrical waves and a simple MRT operator.

In the same continuous-model limit, we can compare Eq. (12.28) and Eq. (12.29) with the corresponding results in Eq. (12.13) for the isothermal Navier-Stokes model. We see that to lowest order, the absorption is consistent while the dispersion is not. In fact, it can be found that all terms above $O(\omega_0\tau_{\mathrm{vi}})$ deviate from the isothermal Navier-Stokes model [7].

The analysis performed here has some clear limits: it assumes that there is no background flow, it is performed for one specific equilibrium, it is limited to D1Q3 (*i.e.* the anisotropy of other velocity sets[12] is lost), and it does not take into account various possible choices of collision operator. There exist articles that take these effects into account in D2Q9-based linearisation analyses [32, 35], though their analyses are unfortunately limited to free waves and are by necessity *far* more complex.

12.3 Sources of sound

There are many reasons why sound waves may appear in LB simulations:

- **Intentional setup:** The initial conditions of the simulation can be set up to contain sound waves. Free waves can be initialised with one wavelength in a periodic system [1, 24, 33, 38, 39, 7, 25]. Forced waves can be initialised with an inhomogeneous (e.g. Gaussian) density distribution [40, 41, 42, 43, 5, 44, 45].
- **Unintentional setup:** When setting up a flow field as an initial condition for a simulation, this field is seldom perfect for the case at hand. Typically, this initial condition will be a mix of an incompressible flow field and a sound wave

[12] For forced waves in the continuous model using the D2Q9 velocity set, anisotropy first occurs at $O(\omega_0\tau_{\mathrm{vi}})^3$ [26]. Thus, absorption and dispersion are both isotropic to lowest order.

field. However, generally separating a low-Ma field into incompressible flow and sound is an unsolved, and perhaps *unsolvable*, problem,[13] which makes it hard to avoid including sound waves in initial flow conditions of simulations.[14]

- **Aerodynamic noise:** Sound can come from unsteady flow fields and their interaction with surfaces. Indeed, this is the main source of *e.g.* aircraft noise. As this behaviour is a consequence of the fluid conservation equations, it can also be captured in LB simulations.
- **Artificial sound sources:** It is also possible to make sound sources in simulations that generate sound artificially during the simulation. These can either be inside the domain or be implemented as boundary conditions.

In this section we will focus on the last two cases. They are tied together through the theory of sound sources which we will now look into.

12.3.1 Example: The pulsating sphere

One of the most basic examples of a sound source within a fluid is a pulsating sphere which continuously expands and contracts at a single frequency ω_0. For a sphere with a rest radius of a centred at $\boldsymbol{x} = 0$, such a pulsation is typically modelled by a velocity boundary condition $u_r(|\boldsymbol{x}| = a, t) = u^\circ \mathrm{e}^{\mathrm{i}\omega_0 t}$, u_r being the fluid velocity in the radial direction and u° being the sphere's velocity amplitude. The sphere's pulsation is sketched in Fig. 12.2.

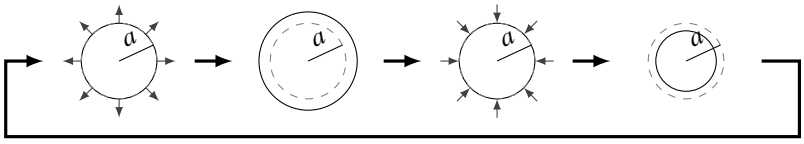


Fig. 12.2: Exaggerated sketch of four stages of a pulsating sphere of rest radius a. From left to right: Expanding, fully expanded, contracting, and fully contracted. The surface velocity is indicated by arrows.

This velocity boundary condition leads to a mass displacement per time unit of $Q\mathrm{e}^{\mathrm{i}\omega t} = 4\pi a^2 \rho_0 u_0 \mathrm{e}^{\mathrm{i}\omega t}$. In three dimensions it also leads to the generation of the pressure wave [10]

$$\hat{p}'(\boldsymbol{x}, t) = \frac{\mathrm{i}\omega Q}{4\pi|\boldsymbol{x}|} \mathrm{e}^{\mathrm{i}(\omega_0 t - \hat{k}|\boldsymbol{x}|)}. \tag{12.30}$$

[13] Even so, some filtering techniques have been developed to approximately separate the sound field from the total flow field [46, 47, 48, 49].

[14] Initial flow conditions for LB simulations are described in Section 5.4.

Alternatively, we can model this mass displacement as a *mass source* at $\boldsymbol{x} = 0$ through an inhomogeneous linearised continuity equation

$$\partial_t \rho + \rho_0 \partial_\alpha u_\alpha = Q\delta(\boldsymbol{x})\mathrm{e}^{\mathrm{i}\omega t}, \tag{12.31}$$

which leads to a wave equation

$$\left(\frac{1}{c_\mathrm{s}^2}\partial_t^2 - (1 + \tau_\mathrm{vi}\partial_t)\nabla^2\right)\hat{p}'(\boldsymbol{x}, t) = \mathrm{i}\omega Q\delta(\boldsymbol{x})\mathrm{e}^{\mathrm{i}\omega t}, \tag{12.32}$$

which, from the Green's functions in Section 12.1.6, has a solution identical to Eq. (12.30).

From this we can tell that pulsating objects scattered throughout the fluid can be modelled as mass sources. Similarly, oscillating objects (i.e. objects moving side-to-side) can be modelled as forces throughout the fluid.

Now, how is this relevant to LB simulations? In Section 12.3.3 we will come back to how such a simple source can be implemented in LB simulations.

12.3.2 The inhomogeneous wave equation

In the last subsection we saw how a pulsating sphere may be modelled as a mass source, causing an inhomogeneous term to appear in the wave equation. Indeed, there are several possible sources of such inhomogeneous terms which we will now look at.

Instead of using approximate, linearised, forceless conservation equations, the wave equation may also be derived from the Navier-Stokes-level conservation equations

$$\partial_t \rho + \partial_\alpha \rho u_\alpha = Q(\boldsymbol{x}, t), \tag{12.33a}$$

$$\partial_t\left(\rho u_\alpha\right) + \partial_\beta\left(\rho u_\alpha u_\beta\right) = -\partial_\alpha p + \partial_\beta \sigma'_{\alpha\beta} + F_\alpha(\boldsymbol{x}, t), \tag{12.33b}$$

where $Q(\boldsymbol{x}, t)$ is a time-varying distribution of mass sources throughout the physical domain.

From these, an *exact* inhomogeneous wave equation for the *density* can be derived

$$\left(\frac{1}{c_\mathrm{s}^2}\partial_t^2 - \nabla^2\right) c_\mathrm{s}^2 \rho'(\boldsymbol{x}, t) = \partial_t Q(\boldsymbol{x}, t) - \partial_\alpha F_\alpha(\boldsymbol{x}, t) + \partial_\alpha \partial_\beta T_{\alpha\beta}(\boldsymbol{x}, t). \tag{12.34}$$

Each of the terms on the right-hand side represents a type of sound source. The last term contains the *Lighthill stress tensor*

$$T_{\alpha\beta} = \rho u_\alpha u_\beta + (p' - c_\mathrm{s}^2 \rho')\delta_{\alpha\beta} - \sigma'_{\alpha\beta}. \tag{12.35}$$

As a source of sound, the latter two terms in the Lighthill stress tensor do not typically contribute [23]. However, the second is linked to the deviation from the linearised isentropic equation of state,[15] while the third is linked to viscous attenuation as discussed earlier in this chapter.

The first term, which comes from the nonlinear term on the left-hand side of the momentum equation, *does* contribute. The contribution from the inhomogeneous term $\partial_\alpha \partial_\beta (\rho u_\alpha u_\beta)$ is typically strongest when there is a strongly fluctuating underlying flow field, such as a turbulent field.

From an impulsive Green's function approach, similar to the time-harmonic Green's function approach described in Section 12.1.6, an exact solution for the inhomogeneous density wave equation Eq. (12.34) can be found as an integral of the inhomogeneous terms over the entire domain [22, 7]:

$$\begin{aligned} c_\mathrm{s}^2 \rho'(\boldsymbol{x}, t) = \int \Big[&\left(\partial_t Q(\boldsymbol{y}, t)\right) G(\boldsymbol{x} - \boldsymbol{y}, t) - F_\alpha(\boldsymbol{y}, t) \partial_\alpha G(\boldsymbol{x} - \boldsymbol{y}, t) \\ &+ T_{\alpha\beta}(\boldsymbol{y}, t) \partial_\alpha \partial_\beta G(\boldsymbol{x} - \boldsymbol{y}, t) \Big] \, \mathrm{d}^3 y. \end{aligned} \tag{12.36}$$

From Section 12.1.6 we know that these three terms can be interpreted as monopole, dipole, and quadrupole terms, respectively. For the case where all the inhomogeneous terms in Eq. (12.34) are time-harmonic, the corresponding Green's function is shown in Eq. (12.17).[16]

The sound generated by the interaction of the flow field, in particular the pressure, with extended surfaces is a field of study in of itself. For more on this this we refer to the literature [50, 51, 52, 22].

12.3.3 Point source monopoles in LB simulations

At this point we have introduced three sources of sound: The mass displacement Q from small pulsating structures makes monopole sound sources, time-varying inhomogeneous forces are dipole sound sources, and a strongly fluctuating fluid velocity represents a distributed quadrupole sound source.

Now, how do these inhomogeneous sound waves appear in LB simulations? The forcing can be imposed using any of the forcing schemes described in Chapter 6. The quadrupole source is a direct consequence of the Navier-Stokes equation, which means that even the most basic LBE is therefore able to simulate such sources without modification if the flow is resolved finely enough [53, 54, 55].

On the other hand, the mass fluctuation must be imposed. In this section we will describe how this can be done through an approach that can also be generalised further to dipoles and quadrupoles.

---

[15] This term is zero for the isothermal equation of state.

[16] Note again that there *is* no simple impulsive Green's function available for 2D cases [11].

An early approach to monopole point sources in LB simulations was to replace the entire particle distribution in a node with an equilibrium distribution at an imposed density that fluctuates around the rest density ρ_0 [56, 5]. However, this method is not advisable for several reasons [17]: It will disturb the flow unphysically, and the relationship between the source and the radiated wave's amplitude and phase is unknown.

A newer approach to LB sound sources is based on adding a particle source term j_i to the lattice Boltzmann equation,

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = j_i(\boldsymbol{x}, t) + \Omega_i(\boldsymbol{x}, t)\Delta t. \tag{12.37}$$

This term represents particles being added to or removed from the distribution function with a certain distribution in velocity space, physical space, and time. This can be contrasted with the mass source in Eq. (12.33a), which is distributed in space and time but not in *velocity* space.

The effect of the particle source term j_i is best seen through its moments, which we define similarly to the moments of f_i,

$$J = \sum_i j_i, \quad J_\alpha = \sum_i c_{i\alpha} j_i, \quad J_{\alpha\beta} = \sum_i c_{i\alpha} c_{i\beta} j_i. \tag{12.38}$$

Naturally, adding a source term to the LBE will affect its macroscopic behaviour. This behaviour can be determined by a Chapman-Enskog analysis as detailed in Section 4.1, resulting in mass and momentum conservation equations

$$\partial_t \rho + \partial_\beta(\rho u_\beta) = J - \tfrac{1}{2}\left(\partial_t J + \partial_\beta J_\beta\right), \tag{12.39a}$$

$$\begin{aligned}\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = {} & -\partial_\alpha p + \partial_\beta \sigma'_{\alpha\beta} + (1 - \tfrac{1}{2}\partial_t) J_\alpha - \partial_\beta(\tau J_{\alpha\beta}) \\ & + \partial_\beta(\tau - \tfrac{1}{2})\left(\delta_{\alpha\beta} c_s^2 J + u_\alpha J_\beta + u_\beta J_\alpha - u_\alpha u_\beta J\right),\end{aligned} \tag{12.39b}$$

where the viscous stress tensor is, as usual, $\sigma'_{\alpha\beta} = \rho c_s^2(\tau - 1/2)(\partial_\beta u_\alpha + \partial_\alpha u_\beta)$. These conservation equations have gained source terms given by the moments of the particle source term j_i. The derivative terms with the $1/2$ prefactor are a result of the velocity space discretisation error of the LBE [7]. Disregarding these terms, the moment J directly takes the place of Q in Eq. (12.33a).

Many of these additional terms in the conservation equations are small, being of order $(\tau - 1/2)uj$. Neglecting these, we can derive a viscous wave equation as done in Section 12.1.1, resulting in

$$\begin{aligned}\left[\frac{1}{c_s^2}\partial_t^2 - (1 + \tau_{\mathrm{vi}}\partial_t)\,\nabla^2\right] p = {} & \left(1 - \tfrac{1}{2}\partial_t\right)\partial_t J - \partial_\alpha J_\alpha \\ & + \tau\partial_\alpha\partial_\beta J_{\alpha\beta} - \left(\tau - \tfrac{1}{2}\right)\partial_\alpha\partial_\beta\left(3\delta_{\alpha\beta} c_s^2 J\right).\end{aligned} \tag{12.40}$$

Example 12.1. Let us look at the radiated wave from a point source at $\boldsymbol{x} = 0$, which from Eq. (12.34) and Eq. (12.36) is

$$\begin{aligned}\hat{p}'(\boldsymbol{x},t) = &\left[\left(\mathrm{i}\omega_0 + \frac{\omega_0^2}{2}\right) J\right] \hat{G}(\boldsymbol{x},t) - J_\alpha \partial_\alpha \hat{G}(\boldsymbol{x},t) \\ &+ \left[\tau J_{\alpha\beta} - \left(\tau - \tfrac{1}{2}\right) 3\delta_{\alpha\beta} c_\mathrm{s}^2 J\right] \partial_\alpha \partial_\beta \hat{G}(\boldsymbol{x},t).\end{aligned} \tag{12.41}$$

The J moment affects monopoles and quadrupoles, the J_α moment only affects dipoles, and the $J_{\alpha\beta}$ moment only affects quadrupoles.

In general, we can choose j_i cleverly to tailor which moments are non-zero and which are zero. For monopoles, a simple assumption is that mass appears at equilibrium, *i.e.*

$$j_i(\boldsymbol{x},t) = w_i Q(\boldsymbol{x},t), \tag{12.42}$$

which from Eq. (3.62) results in the moments

$$J = \sum_i j_i = Q, \quad J_\alpha = \sum_i c_{i\alpha} j_i = 0, \quad J_{\alpha\beta} = \sum_i c_{i\alpha} c_{i\beta} j_i = c_\mathrm{s}^2 \delta_{\alpha\beta} Q. \tag{12.43}$$

It might seem contradictory to choose a monopole source that also affects the quadrupole moment. However, this choice is ideal as it causes the quadrupole moment to cancel against the discretisation error term for low values of τ. (As explained in Section 12.1.3, such low values are generally necessary to use in LB acoustics.)

Let us show where this cancellation comes from: using the harmonic 2D and 3D Green's functions in Eq. (12.17), it can be found that Eq. (12.41) becomes

$$\partial_\alpha \partial_\alpha \hat{G}(\boldsymbol{x},t) = -\hat{k}^2 \hat{G}(\boldsymbol{x},t) \tag{12.44}$$

in both cases. Consequently, since Eq. (12.43) shows that $J_{\alpha\beta} = c_\mathrm{s}^2 \delta_{\alpha\beta} J$ and $J_\alpha = 0$, and since $(\hat{k} c_\mathrm{s}) \approx \omega_0$, for $j_i = w_i Q(t)\delta(\boldsymbol{x})$ and $\tau \to \frac{1}{2}$ Eq. (12.41) becomes simply

$$\hat{p}'(\boldsymbol{x},t) = \mathrm{i}\omega_0 Q \hat{G}(\boldsymbol{x},t), \tag{12.45}$$

just as it should be for a monopole point source.

To summarise, we can add sound sources to LB simulations through a particle source term j_i as in Eq. (12.37). By setting $j_i(\boldsymbol{x},t) = w_i Q(\boldsymbol{x},t)$, we can create monopole sound sources. These sources can either be single point sources that radiate as Eq. (12.45) or distributions of such point sources around the domain.

The same approach can be expanded to create dipole and quadrupole sources. Indeed, j_i can be decomposed into a basis of a monopole and various dipole and quadrupole sources. We will not go into this further here, but rather refer to the literature [17, 7].

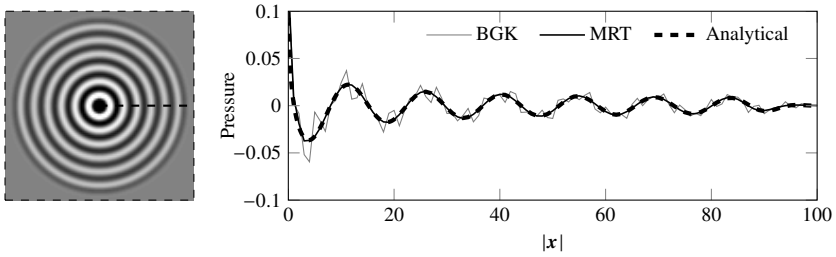


Fig. 12.3: Left: Snapshot of the pressure wave radiated by a smoothly switched-on monopole source, using a simple MRT operator at $\tau = 1/2$. Right: Comparison of the radial pressure along the dashed line on the left with the pressure predicted by theory, for MRT and BGK operators at $\tau = 1/2$. Taken from [7].

As an example of the monopole point source in action, Fig. 12.3 compares the field of a D2Q9 monopole sound source simulated at $\tau = 1/2$ using both the BGK operator and a simple MRT operator with immediate relaxation (*i.e.* relaxation times of 1) of the non-hydrodynamic moments. Taking the analytical wavenumber from Eq. (12.27), we find excellent agreement for MRT collisions while the BGK case shows spurious oscillations around the analytical solution. It should also be pointed out that there is a singularity in the analytical solution at $\boldsymbol{x} = \mathbf{0}$, which means that the simulated solution can never match the analytical solution in the immediate area around the source point.

Finally, we should briefly mention *precursors*, a wave phenomenon that appears in dispersive media. If sources are immediately switched on at reasonably high viscosities, precursors will appear [57], manifesting as bumps at the first wavefront. Additionally, if sources are immediately turned on in simulations at very low viscosities, errors may be visible along the first wavefront. In such cases, sound sources should be turned on smoothly. More details on this can be found elsewhere [7].

12.4 Non-reflecting boundary conditions

The most commonly used boundary conditions for inlets and outlets in lattice Boltzmann simulations specify the flow field's velocity or density. Such boundary conditions are far from ideal for flows containing sound waves, as they will reflect sound waves back into the fluid [58]. Consequently, any sound waves generated in the simulated system will not *exit* the system, and this may pollute the simulation results. In particular, the density field may be strongly polluted [6] as its variations are typically quite weak for steady flows, where they scale as $\rho'/\rho_0 = O(\mathrm{Ma}^2)$ [59]. (And of course, when the density field is polluted, the pressure field is polluted accordingly, as $p' = c_\mathrm{s}^2 \rho'$.)

This is not an LB-specific problem, but a general problem for compressible flow solvers. To solve it, a variety of different BC approaches that reflect much less sound have been proposed [60]. Some work has been done on adapting such non-reflecting BCs (or *NRBCs*) to lattice Boltzmann simulations [43, 61, 62, 63, 6, 45], but at this point there has been little work done on comparing these against each other [9].

There are two main approaches to NRBCs. The first is characteristic boundary conditions (CBCs), where the fluid equations are decomposed in boundary nodes in a way that allows supressing waves being reflected back from the boundary. The second approach is absorbing layers, where the simulated domain is bounded by a layer several nodes thick. In an absorbing layer, the LBM is modified in such a way that incoming waves are absorbed as they pass through the layer without being reflected back.

Absorbing layers are generally more demanding than CBCs, as they add more nodes to the system that need to be computed similarly to the existing ones. However, better results may be achieved by absorbing layers [9].

12.4.1 Reflecting boundary conditions

To understand the point of NRBCs, we must first comprehend how sound waves are reflected by simpler boundary conditions. These BCs typically impose a constant value for the fluid pressure or the fluid velocity. In this section we will look at what happens when an incoming sound wave, which represents a fluctuation around this constant pressure or velocity, meets such a BC. For simplicity, we will restrict ourselves to the basic one-dimensional case of a plane sound wave hitting the boundary at normal incidence. The reflection of non-plane waves is a more difficult topic which we will not go into here.

First, however, we must know how the pressure and the velocity in a sound wave are coupled. From Exercise 12.1 we already know that the pressure of a one-dimensional wave propagating in the $\pm x$ direction can be represented as $p'(c_s t \mp x)$. From Euler's momentum equation, the relation to the fluid velocity $u'(c_s t \mp x)$ can be determined:

Exercise 12.8. From the linearised one-dimensional Euler's equation $\rho_0 \partial_t u' = -\partial_x p$, show that the pressure and velocity are coupled as

$$p'(c_s t \mp x) = \pm \rho_0 c_s u'(c_s t \mp x). \tag{12.46}$$

Hint: Use the chain rule with the argument $c_s t \mp x$.

In the following examples, we will consider boundary conditions imposed at $x = 0$, with a known incoming sound wave $p'_i(c_s t - x)$ and an unknown reflected sound wave $p'_r(c_s t + x)$, the physical domain being $x \leq 0$.

Let us first consider a *pressure* boundary condition, which imposes $p = p_0$ at $x = 0$. Imposing this constant pressure means imposing a zero pressure fluctuation,

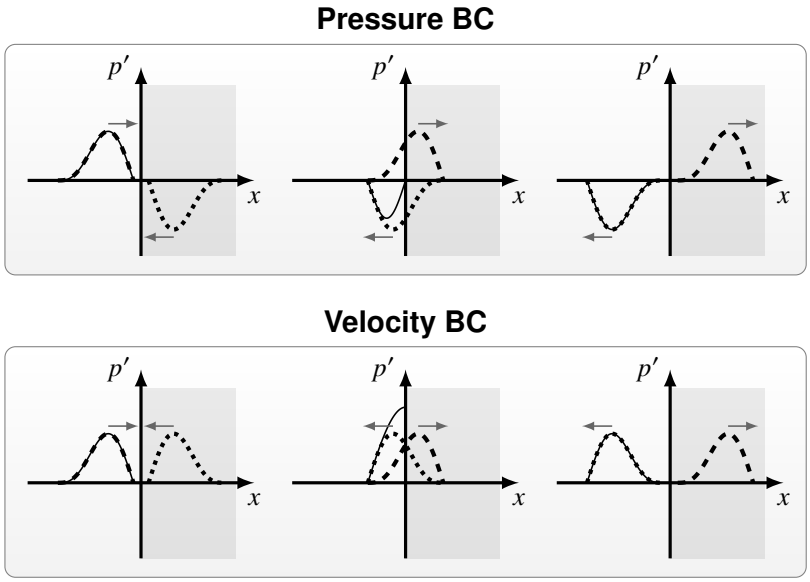


Fig. 12.4: For two different BCs, the incoming pressure pulse $p'_\mathrm{i}(c_\mathrm{s}t - x)$ (dashed) hits a boundary at $x = 0$ and a reflected pulse $p'_\mathrm{r}(c_\mathrm{s}t + x)$ (dotted) is sent back. The total pressure $p'_\mathrm{i} + p'_\mathrm{r}$ (solid) in the physical domain ($x \leq 0$) is also shown. (The non-physical 'mirror' domain ($x > 0$) is indicated by a darker colour.)

$p' = 0$, at $x = 0$, so that

$$p'(0,t) = p'_\mathrm{i}(c_\mathrm{s}t - 0) + p'_\mathrm{r}(c_\mathrm{s}t + 0) = 0. \tag{12.47a}$$

The unknown reflected wave p_r must therefore be the inverse of the incoming wave, *i.e.*

$$p'_\mathrm{r}(c_\mathrm{s}t + x) = -p'_\mathrm{i}(c_\mathrm{s}t - x). \tag{12.47b}$$

Second, let us consider a *velocity* boundary condition, which imposes $u = u_0$, *i.e.* $u' = 0$, at $x = 0$. The incoming and reflected fluid velocities are linked as

$$u'(0,t) = u'_\mathrm{i}(c_\mathrm{s}t - 0) + u'_\mathrm{r}(c_\mathrm{s}t + 0) = 0, \tag{12.48a}$$

and from Eq. (12.46) this links the incoming and reflected pressures as

$$p'_\mathrm{r}(c_\mathrm{s}t + x) = p'_\mathrm{i}(c_\mathrm{s}t - x). \tag{12.48b}$$

Both for a pressure and a velocity BC, the incoming wave is transformed into an identically shaped reflected wave of the same amplitude. For the pressure

BC, the reflected wave has the opposite sign, while it has the same sign for the velocity BC. These two cases are shown in Fig. 12.4, which also shows the non-physical 'mirror' domain $x > 0$ from which the reflected waves come.

Consequently, any LB BCs that enforce constant pressure or constant velocity along a boundary will reflect sound waves back into the system, regardless of the specific implementation of these BCs.

In a more realistic case, such as a sound wave hitting a building wall, there will not just be an incoming and a reflected sound wave: as most will have noticed, sound can also be *transmitted* into the wall (and transmitted again from the wall to the indoor air). Considering only normal incidence, the efficiency of this transmission between two media depends on their *characteristic impedance* $Z = \rho_0 c_s = \pm p'/u'$ [10]. If the impedances of both media are equal, the transmission is perfect, meaning that the incoming and transmitted waves are equal and that nothing is reflected at the boundary. The same reflections as for the pressure and velocity BCs are recovered if the impedance of the second medium is $Z = 0$ and $Z \to \infty$, respectively.[17]

12.4.2 Characteristic boundary conditions

A characteristic BC (CBC) is a type of NRBC where the macroscopic variables on the boundary nodes are determined in such a way that no waves are reflected. This is done by separating the macroscopic flow equations on the boundary into various components or *characteristics*, typically representing outgoing waves, pure advection, and incoming waves. Setting the amplitude of the incoming wave component to zero results in known macroscopic variables at the boundary in the next time step. These must then be implemented through another boundary condition.

Classic CBC approaches [65, 66] have in recent years been adapted to the LB method [61, 45]. In the following, we will follow the simplest exposition [45].

We first assume that the conservation equations at the boundary are nearly Eulerian, *i.e.* we neglect the effect of viscosity. For the force-free two-dimensional case, the Euler conservation equations can be expressed in vector and matrix form as

$$\partial_t \boldsymbol{m} + \boldsymbol{X}\partial_x \boldsymbol{m} + \boldsymbol{Y}\partial_y \boldsymbol{m} = 0, \tag{12.49a}$$

using the fluid variable vector $\boldsymbol{m} = (\rho, u_x, u_y)^{\mathrm{T}}$ and the matrices

[17] Such transmission between two media of different impedances has also been simulated correctly using the LBM, including weak interfacial effects not predicted by the hydrodynamic equations [64].

$$\boldsymbol{X} = \begin{bmatrix} u_x & \rho & 0 \\ c_\mathrm{s}^2/\rho & u_x & 0 \\ 0 & 0 & u_x \end{bmatrix}, \qquad \boldsymbol{Y} = \begin{bmatrix} u_y & 0 & \rho \\ 0 & u_y & 0 \\ c_\mathrm{s}^2/\rho & 0 & u_y \end{bmatrix}. \tag{12.49b}$$

As this system of equations is hyperbolic, these matrices are diagonalisable as $\boldsymbol{X} = \boldsymbol{P}_x^{-1}\boldsymbol{\Lambda}_x\boldsymbol{P}_x$ and $\boldsymbol{Y} = \boldsymbol{P}_y^{-1}\boldsymbol{\Lambda}_y\boldsymbol{P}_y$, where $\boldsymbol{\Lambda}_x$ and $\boldsymbol{\Lambda}_y$ are diagonal matrices containing the eigenvalues of $\boldsymbol{X}$ and $\boldsymbol{Y}$,

$$\begin{aligned} \boldsymbol{\Lambda}_x &= \mathrm{diag}(\lambda_{x,1}, \lambda_{x,2}, \lambda_{x,3}) = \mathrm{diag}(u_x - c_\mathrm{s}, u_x, u_x + c_\mathrm{s}), \\ \boldsymbol{\Lambda}_y &= \mathrm{diag}(\lambda_{y,1}, \lambda_{y,2}, \lambda_{y,3}) = \mathrm{diag}(u_y - c_\mathrm{s}, u_y, u_y + c_\mathrm{s}). \end{aligned} \tag{12.50a}$$

The diagonalisation matrices are given by

$$\begin{aligned} \boldsymbol{P}_x &= \begin{bmatrix} c_\mathrm{s}^2 & -c_\mathrm{s}\rho & 0 \\ 0 & 0 & 1 \\ c_\mathrm{s}^2 & c_\mathrm{s}\rho & 0 \end{bmatrix}, \quad \boldsymbol{P}_x^{-1} = \begin{bmatrix} \frac{1}{2c_\mathrm{s}^2} & 0 & \frac{1}{2c_\mathrm{s}^2} \\ -\frac{1}{2\rho c_\mathrm{s}} & 0 & \frac{1}{2\rho c_\mathrm{s}} \\ 0 & 1 & 0 \end{bmatrix}, \\ \boldsymbol{P}_y &= \begin{bmatrix} c_\mathrm{s}^2 & 0 & -c_\mathrm{s}\rho \\ 0 & 1 & 0 \\ c_\mathrm{s}^2 & 0 & c_\mathrm{s}\rho \end{bmatrix}, \quad \boldsymbol{P}_y^{-1} = \begin{bmatrix} \frac{1}{2c_\mathrm{s}^2} & 0 & \frac{1}{2c_\mathrm{s}^2} \\ 0 & 1 & 0 \\ -\frac{1}{2\rho c_\mathrm{s}} & 0 & \frac{1}{2\rho c_\mathrm{s}} \end{bmatrix}. \end{aligned} \tag{12.50b}$$

The physical meaning of this diagonalisation can be seen in a y-invariant case where $\partial_y \boldsymbol{m} = \boldsymbol{0}$. Left-multiplying eq. (12.49a) with $\boldsymbol{P}_x$ results in

$$\boldsymbol{P}_x\partial_t\boldsymbol{m} + \boldsymbol{\Lambda}_x\boldsymbol{P}_x\partial_x\boldsymbol{m} = 0. \tag{12.51}$$

A subsequent definition $\mathrm{d}\boldsymbol{n} = \boldsymbol{P}_x\mathrm{d}\boldsymbol{m}$ leads to the three equations

$$\partial_t n_i + \lambda_{x,i}\partial_x n_i = 0. \tag{12.52}$$

These equations are mathematically identical to the advection equation, describing propagation of the quantities n_i, the propagation speeds being given by the eigenvalues $\lambda_{x,i}$. From Eq. (12.50), these propagation speeds in turn describe the combined effect of advection and sound propagation in the $-x$ direction, pure advection, and advection and sound propagation in the $+x$ direction. The quantities n_i can consequently be interpreted as amplitudes of the corresponding components of the flow field.

Why this decomposition into characteristics is relevant for non-reflecting BCs can now be seen. If we can enforce $n_i = 0$ at the boundary for the characteristics that represent sound coming into the system, we can in principle ensure that sound waves smoothly exit the system without reflection. Let us now look at how to implement this.

The x-derivative term in eq. (12.49a) can be expressed as

$$\boldsymbol{X}\partial_x\boldsymbol{m} = \boldsymbol{P}_x^{-1}\boldsymbol{\Lambda}_x\boldsymbol{P}_x\partial_x\boldsymbol{m} = \boldsymbol{P}_x^{-1}\boldsymbol{\mathcal{L}}_x, \tag{12.53}$$

defining a characteristic vector as $\mathcal{L}_{x,i} = \lambda_{x,i} \sum_j P_{x,ij} \partial_x m_j$. Each component $\mathcal{L}_{x,i}$ is proportional to the amplitude of one of the three characteristics. Explicitly, the components of $\boldsymbol{\mathcal{L}}_x$ are

$$\begin{aligned} \mathcal{L}_{x,1} &= (u_x - c_\mathrm{s}) \left[c_\mathrm{s}^2 \partial_x \rho - c_\mathrm{s} \rho \partial_x u_x \right], \\ \mathcal{L}_{x,2} &= (u_x) \left[\partial_x u_y \right], \\ \mathcal{L}_{x,3} &= (u_x + c_\mathrm{s}) \left[c_\mathrm{s}^2 \partial_x \rho + c_\mathrm{s} \rho \partial_x u_x \right]. \end{aligned} \tag{12.54}$$

At x-boundaries we want to enforce a *modified* characteristic vector with elements

$$\mathcal{L}'_{x,i} = \begin{cases} \mathcal{L}_{x,i} & \text{for outgoing characteristics,} \\ 0 & \text{for incoming characteristics.} \end{cases} \tag{12.55}$$

Outgoing characteristics are those where the eigenvalue $\lambda_{x,i}$ corresponds to propagation out of the system (*e.g.* $\lambda_{x,3} = u_x + c_\mathrm{s}$ at the rightmost boundary). For incoming characteristics, the eigenvalue corresponds to propagation into the system (*i.e.* $\lambda_{x,1} = u_x - c_\mathrm{s}$ at the same boundary). In this way, we ensure that the outgoing characteristics are undisturbed while the incoming characteristics carry nothing back into the system.

A similar characteristic vector can be defined for the y-derivatives as $\mathcal{L}_{y,i} = \lambda_{y,i} \sum_j P_{y,ij} \partial_y m_j$. The only difference to $\mathcal{L}_{x,i}$ given in Eq. (12.54) is that all the x and y indices are switched. We similarly want to enforce a modified characteristic vector $\boldsymbol{\mathcal{L}}'_y$ at the y boundaries.

There are several different approaches to evolving the fluid variables at the x-boundary that enforce $\boldsymbol{\mathcal{L}}_x$. They can be generalised using a modified version of Eq. (12.49a) as

$$x\text{-boundary:} \qquad \partial_t \boldsymbol{m} = -\boldsymbol{P}_x^{-1} \boldsymbol{\mathcal{L}}'_x - \gamma \boldsymbol{Y} \partial_y \boldsymbol{m}. \tag{12.56a}$$

Here, $\gamma = 1$ corresponds to the original approach of Thompson [65], while the LB CBC of Izquierdo and Fueyo used an one-dimensional approach with $\gamma = 0$ that does not include any y-contribution [61]. Heubes et al. found the choice of $\gamma = 3/4$ to be superior [45]. In the same generalised approach, y-boundaries and the x- and y-boundary conditions at corners may be treated as

$$y\text{-boundary:} \qquad \partial_t \boldsymbol{m} = -\gamma \boldsymbol{X} \partial_x \boldsymbol{m} - \boldsymbol{P}_y^{-1} \boldsymbol{\mathcal{L}}'_y, \tag{12.56b}$$

$$\text{corner:} \qquad \partial_t \boldsymbol{m} = -\boldsymbol{P}_x^{-1} \boldsymbol{\mathcal{L}}'_x - \boldsymbol{P}_y^{-1} \boldsymbol{\mathcal{L}}'_y. \tag{12.56c}$$

To determine the time derivatives $\partial_t \boldsymbol{m}$, the characteristic vectors must be estimated according to Eq. (12.55) and Eq. (12.54). This requires estimating the spatial derivatives. For derivatives *across* a boundary (*e.g.* x-derivatives at an x-boundary), this can be done through the one-sided second-order finite difference approximations

$$(\partial_x m_i)(x) \approx \frac{\mp 3 m_i(x) \pm 4 m_i(x \pm \Delta x) \mp m_i(x \pm 2\Delta x)}{2\Delta x},$$

where the upper and lower signs correspond to forward and backward difference approximations, respectively. For derivatives *along* a boundary (*i.e.* y-derivatives at an x-boundary), the second-order central difference approximation

$$(\partial_x m_i)(x) \approx \frac{m_i(x + \Delta x) - m_i(x - \Delta x)}{2\Delta x},$$

is appropriate.

With the spatial derivatives in place, we have a known approximation of the macroscopic variables' time derivative $\partial_t \boldsymbol{m}$ on every edge of the system. To determine these macroscopic variables for the next time step, a simple approach is a Forward Euler one where

$$m_i(\boldsymbol{x}, t + \Delta t) \approx m_i(\boldsymbol{x}, t) + \Delta t \partial_t m_i(\boldsymbol{x}, t).$$

In one benchmark, this simple first-order approach performed near-identically to a higher-order Runge-Kutta approach [45], which suggests that the error of this approximation is typically dominated by other sources of error in the CBC method.

With a known approximation of each macroscopic variable m_i on the non-reflecting boundary at the next time step, these macroscopic variables may be implemented through any boundary condition that allows specifying ρ, u_x, and u_y. The simplest choice, which has also been made in the literature [45, 9], is to simply replace the distribution function f_i at the boundary with an equilibrium distribution f_i^{eq} determined by the macroscopic variables ρ, u_x, and u_y, like in the equilibrium scheme discussed in Section 5.2.4.

When implementing this boundary condition in code, no collisions are required in the CBC boundary nodes. After the macroscopic variables have been determined, but before streaming, the f_is in the CBC boundary nodes are replaced using predetermined macroscopic variables $m_i(t)$, and the macroscopic variables $m_i(t + \Delta t)$ for the next time step are determined.

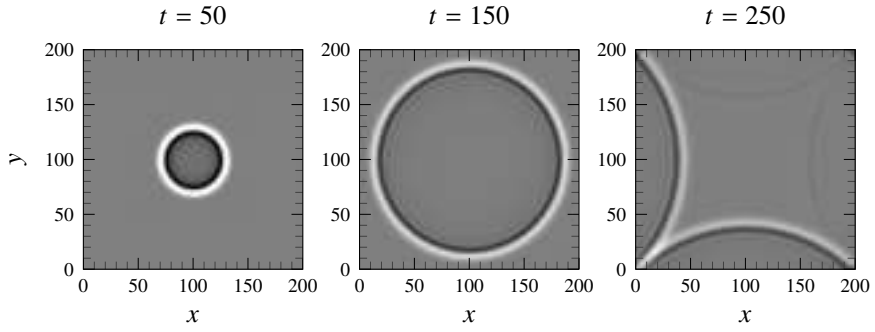


Fig. 12.5: Reflection of a sound pulse from bounce-back boundaries (bottom and left) and characteristic BC boundaries (top and right).

As an example of how CBCs are significantly less reflective than velocity boundaries, CBC boundaries as described above are compared against mid-grid bounce-back in Fig. 12.5. In the simulation, a pulse was initialised as a Gaussian-distributed density $\rho'(\boldsymbol{x}, t = 0) = 10^{-4}\mathrm{e}^{-(\boldsymbol{x}-\boldsymbol{x}_0)^2/10}$, $\boldsymbol{x}_0$ being the centre of the system. The simulation was run using the BGK collision operator with $\tau = 0.51$, and the CBC used $\gamma = 0.75$. Note however that the CBCs are not perfect; weak reflections can be seen from these boundaries.

12.4.3 Absorbing layers

A characteristic boundary condition (CBC), as described in Section 12.4.2, is a type of a non-reflecting BC (NRBC) that only affects the nodes adjacent to a boundary. A different approach is taken by *absorbing layer*-type NRBCs, where the simulation domain is bounded by a layer of many nodes of thickness. In this layer, the LBM is modified in a way that attenuates sound waves as they pass through the layer, with as much attenuation and as little reflection as possible. The performance of an absorbing layer typically increases with its thickness.

At the outer edge of the layer, another BC must be chosen to close the system. Ideally this should be a CBC or similar in order to reduce the reflection at the edge of the system (in addition to the wave absorption in the layer) [60]. In this case, waves are not only attenuated as they pass through the layer, but only a small part of the wave that hits the edge of the system is reflected back in, and that part is further attenuated as it passes *back* through the layer into the simulation domain.

We will not treat absorbing layers in any depth here, as the worthwhile methods are somewhat complex and do not offer as much additional physical insight as CBCs.

The simplest absorbing layer is the *viscous sponge layer*, where the simulated system is bounded by a layer with a higher viscosity than the simulation domain [44, 67]. This viscosity is typically smoothly increasing from the viscosity of the interior domain to a very high viscosity near the edge of the system. Viscous sponge layers are very simple to implement; the value of τ must be made a local function of space inside the layer, but otherwise the LBM proceeds as normal. On the other hand, viscous sponge layers are not as effective at being non-reflective as other types of NRBCs [9]; sound waves may be reflected as they pass through the layer itself [60].

A more complex but more effective absorbing layer is the *perfectly matched layer* (PML), which is designed not only to damp waves passing through it, but also to be non-reflecting for the waves passing through it; the governing equations of a PML are *perfectly matched* to the equations of the interior in such a way that sound is not reflected.

PMLs are implemented by adding terms to the governing equations. These terms are determined using the deviation of the field from some nominal state [60], such as an equilibrium rest state defined by ρ_0 and $\boldsymbol{u}_0$. The amplitude of these additional attenuating terms is increased smoothly into the layer.

LBM adaptations of PMLs [63, 62] and similar absorbing layers [43] can be found in the literature. A comparison of one LB PML implementation with a viscous sponge layer and a CBC as described in Section 12.4.2 found that PMLs perform the best and sponge layers the worst, using a layer thickness of 30 nodes in the comparison [9].

12.5 Summary

Fluid models represented by a set of conservation equations, such as the Euler model or Navier-Stokes model, support sound waves if they fulfill certain criteria: they must have a mass conservation equation on the form $\partial\rho/\partial t + \nabla\cdot(\rho\boldsymbol{u})$, a momentum equation with a pressure gradient, and an equation of state relating the changes in pressure and density. If those criteria are fulfilled, a wave equation can be derived as in Section 12.1.1, which indicates that the fluid model in question supports sound waves.

The Euler and Navier-Stokes models both fulfill these criteria and therefore support sound waves. On the other hand, the *incompressible* Navier-Stokes model presented in Section 1.1.2 does *not*: its assumption of constant density modifies the continuity equation to $\nabla\cdot\boldsymbol{u}$ and breaks the link between changes in pressure and changes in density. (Interestingly, the incompressible LB model described in Section 4.3.2 *does* fulfill the criteria and therefore supports sound waves.)

As discussed in Section 12.3, there are several ways that sound waves can appear in simulations. Sound waves can be present in the initial conditions of a simulation, either intentionally or unintentionally. Additionally, sound can be generated by unsteady flow fields, either by their interaction with surfaces or in areas of rapid fluid velocity variation such as turbulent areas. As these sound generation mechanisms follow from the fluid model's equations, they are not in any way specific to LB simulations; sound waves can generally appear spontaneously in compressible fluid simulations. (Artificial sound sources can also be put in LB simulations, as discussed in Section 12.3.)

These sound waves are typically reflected from velocity or density boundary conditions back into the simulated system as discussed in Section 12.4. This pollutes the flow field of the solution; in particular, the density and pressure fields. In order to ensure that the sound waves exit the system smoothly, special *non-reflecting* boundary conditions must be applied such as characteristic boundary conditions, viscous layers, or perfectly matched layers.

Sound waves in LB simulations do not propagate in quite the same way as in theory, as discussed in Section 12.2. There are two reasons for this. The first is the second-order discretisation error in space and time. The second is the fact that the Boltzmann equation, and its discrete-velocity version that the LBM is based on, predicts a different absorption and dispersion in some cases. This deviation can be quantified through the acoustic viscosity number $\omega_0\tau_{\mathrm{vi}} \sim \mathrm{Kn}$, where ω_0 is the angular frequency and $\tau_{\mathrm{vi}} = (4\nu/3 + \nu_{\mathrm{B}})/c_{\mathrm{s}}^2$ is the viscous relaxation time.

Disregarding the discretisation error, the sound wave propagation of the LBM starts to deviate significantly from that predicted by the Navier-Stokes model at $\omega_0\tau_{\text{vi}} \approx 0.1$ [26]. Similarly, this is also the point at which anisotropy starts becoming significant for the D2Q9 velocity set.

Example 12.2. We have from Eq. (12.5) that $\tau_{\text{vi}} = 2(\tau - 1/2)$ for the BGK collision operator, so that for $\tau = 1$ we get $\tau_{\text{vi}} = 1$. Having $\omega_0\tau_{\text{vi}} < 0.1$, so that waves still propagate more or less according to the Navier-Stokes model, thus requires a sound wave period of $T = 2\pi/\omega_0 > 62.8$.

Generally, choosing τ values as close to $1/2$ as possible is important for LB sound simulations; this was also discussed in Section 12.1.3. However, as shown in Fig. 12.3, this can push the limits of the BGK collision operator and can thus require choosing another collision operator.

As a concluding remark, sound waves are to some degree inevitable in unsteady LB simulations, and it therefore benefits researchers who use the LBM to know something about sound waves, regardless of whether they are interested in sound-related applications or not.

References

1. J.M. Buick, C.A. Greated, D.M. Campbell, Europhys. Lett. (EPL) **43**(3), 235 (1998)
2. A. Wilde, Anwendung des lattice-Boltzmann-verfahrens zur berechnung strömungsakustischer probleme. Ph.D. thesis, TU Dresden (2007)
3. S. Marié, Etude de la méthode Boltzmann sur réseau pour les simulations en aéroacoustique. Ph.D. thesis, Institut Jean le Rond d'Alembert, Paris (2008)
4. A.R. da Silva, Numerical studies of aeroacoustic aspects of wind instruments. Ph.D. thesis, McGill University, Montreal (2008)
5. E.M. Viggen, The lattice Boltzmann method with applications in acoustics. Master's thesis, Norwegian University of Science and Technology (NTNU), Trondheim (2009)
6. M. Schlaffer, Non-reflecting boundary conditions for the lattice Boltzmann method. Ph.D. thesis, TU München (2013)
7. E.M. Viggen, The lattice Boltzmann method: Fundamentals and acoustics. Ph.D. thesis, Norwegian University of Science and Technology (NTNU), Trondheim (2014)
8. M. Hasert, Multi-scale lattice Boltzmann simulations on distributed octrees. Ph.D. thesis, RWTH Aachen University (2014)
9. S.J.B. Stoll, Lattice Boltzmann simulation of acoustic fields, with special attention to non-reflecting boundary conditions. Master's thesis, Norwegian University of Science and Technology (NTNU), Trondheim (2014)
10. L.E. Kinsler, A.R. Frey, A.B. Coppens, J.V. Sanders, *Fundamentals of acoustics*, 4th edn. (John Wiley & Sons, 2000)
11. D.T. Blackstock, *Fundamentals of physical acoustics* (John Wiley & Sons Inc., 2000)
12. P.M. Morse, K.U. Ingard, *Theoretical acoustics* (McGraw-Hill Book Company, 1968)
13. A.D. Pierce, *Acoustics* (The Acoustical Society of America, 1989)
14. S. Temkin, *Elements of acoustics* (John Wiley & Sons, 1981)
15. Lord Rayleigh, *The Theory of Sound*, vol. 1, 1st edn. (Macmillan and Co., London, 1877)
16. Lord Rayleigh, Proc. London Math. Soc. **s1-9**(1), 21 (1877)
17. E.M. Viggen, Physical Review E **87**(2) (2013)
18. C. Truesdell, J. Rat. Mech. Anal. **2**(4), 643 (1953)

19. M. Greenspan, in *Physical Acoustics*, vol. IIA, ed. by W.P. Mason (Academic Press, 1965), pp. 1–45
20. Y. Li, X. Shan, Phil. Trans. R. Soc. A **369**(1944), 2371 (2011)
21. C. Sun, F. Pérot, R. Zhang, D.M. Freed, H. Chen, Communications in Computational Physics **13**(1), 757 (2013)
22. M.S. Howe, *Theory of vortex sound* (Cambridge University Press, 2003)
23. M.J. Lighthill, Proc. R. Soc. London A **211**(1107), 564 (1952)
24. J.M. Buick, C.L. Buckley, C.A. Greated, J. Gilbert, J. Phys. A **33**, 3917 (2000)
25. E.M. Viggen, Phys. Rev. E **90**, 013310 (2014)
26. E.M. Viggen, Communications in Computational Physics **13**(3), 671 (2013)
27. G.G. Stokes, Trans. Cambridge Phil. Soc. **8**, 287 (1845)
28. C.S. Wang Chang, G.E. Uhlenbeck, in *Studies in statistical mechanics*, vol. V (North-Holland Publishing Company, 1970)
29. J. Foch, G. Uhlenbeck, Phys. Rev. Lett. **19**(18), 1025 (1967)
30. J. Foch, M. Losa, Phys. Rev. Lett. **28**(20), 1315 (1972)
31. J.D. Sterling, S. Chen, J. Comput. Phys. **123**(1), 196 (1996)
32. P. Lallemand, L.S. Luo, Phys. Rev. E **61**(6), 6546 (2000)
33. P. Dellar, Phys. Rev. E **64**(3) (2001)
34. A. Wilde, Comp. & Fluids **35**, 986 (2006)
35. T. Reis, T.N. Phillips, Phys. Rev. E **77**(2), 026702 (2008)
36. S. Marié, D. Ricot, P. Sagaut, J. Comput. Phys. **228**(4), 1056 (2009)
37. E.M. Viggen, Phil. Trans. R. Soc. A **369**(1944), 2246 (2011)
38. D. Haydock, J.M. Yeomans, J. of Phys. A **34**, 5201 (2001)
39. J.M. Buick, J.A. Cosgrove, J. Phys. A **39**(44), 13807 (2006)
40. X.M. Li, R.C.K. Leung, R.M.C. So, AIAA Journal **44**(1), 78 (2006)
41. X.M. Li, R.M.C. So, R.C.K. Leung, AIAA Journal **44**(12), 2896 (2006)
42. A.R. da Silva, G.P. Scavone, J. Phys. A **40**(3), 397 (2007)
43. E.W.S. Kam, R.M.C. So, R.C.K. Leung, AIAA J. **45**(7), 1703 (2007)
44. E. Vergnault, O. Malaspinas, P. Sagaut, J. of Comput. Phys. **231**(24), 8070 (2012)
45. D. Heubes, A. Bartel, M. Ehrhardt, J. Comput. Appl. Math. **262**, 51 (2014)
46. W. de Roeck, M. Baelmans, W. Desmet, AIAA J. **46**(2), 463 (2008)
47. C. Silva, F. Nicoud, S. Moreau, in *16th AIAA/CEAS Aeroacoustics Conference* (2010)
48. S. Sinayoko, A. Agarwal, Z. Hu, J. Fluid Mech. **668**, 335 (2011)
49. S. Sinayoko, A. Agarwal, J. Acoust. Soc. Am. **131**(3), 1959 (2012)
50. N. Curle, Proc. R. Soc. A **231**, 505 (1955)
51. J.E. Ffowcs Williams, D.L. Hawkings, Phil. Trans. R. Soc. A **264**(1151), 321 (1969)
52. K.S. Brentner, F. Farassat, AIAA Journal **36**(8), 1379 (1998)
53. T. Colonius, S.K. Lele, Progress in Aerospace Sciences **40**(6), 345 (2004)
54. M. Wang, J.B. Freund, S.K. Lele, Annu. Rev. Fluid Mech. **38**(1), 483 (2006)
55. E. Garnier, N. Adams, P. Sagaut, *Large Eddy Simulation for Compressible Flows*. Scientific Computation (Springer, 2009)
56. H. Yu, K. Zhao, Phys. Rev. E **61**(4), 3867 (2000)
57. D.T. Blackstock, J. Acoust. Soc. Am. **41**(5), 1312 (1967)
58. S. Izquierdo, P. Martínez-Lera, N. Fueyo, Comp. Math. Appl. **58**(5), 914 (2009)
59. P.A. Thompson, *Compressible-Fluid Dynamics* (McGraw-Hill, 1972)
60. T. Colonius, Annu. Rev. Fluid Mech. **36**, 315 (2004)
61. S. Izquierdo, N. Fueyo, Phys. Rev. E **78**(4) (2008)
62. M. Tekitek, M. Bouzidi, F. Dubois, P. Lallemand, Comp. Math. Appl. **58**(5), 903 (2009)
63. A. Najafi-Yazdi, L. Mongeau, Comp. & Fluids **68**, 203 (2012)
64. M.M. Tekitek, M. Bouzidi, F. Dubois, P. Lallemand, Prog. Comput. Fluid Dyn. **8**(1-4), 49 (2008)
65. K.W. Thompson, J. of Comput. Phys. **68**(1), 1 (1987)
66. T. Poinsot, S. Lele, J. Comput. Phys. **101**, 104 (1992)
67. E. Vergnault, O. Malaspinas, P. Sagaut, J. Acoust. Soc. Am. **133**(3), 1293 (2013)

Chapter 13
Numerical Implementations

Abstract This chapter presents a tutorial-style guide to high performance computing and the efficient implementation of LBM algorithms. After a brief introduction to general topics in scientific computing, including floating point arithmetic and processor architecture, the chapter reviews common strategies for optimizing code. Optimization of memory access and the efficient use of cache memory are particularly important, and one section is dedicated to these topics. An example code for studying the decay of a two-dimensional Taylor-Green vortex is then presented and discussed. Subsequent sections describe parallelization of this code using OpenMP for shared memory systems, MPI for distributed memory systems, and CUDA for NVIDIA general purpose graphics processing units. Full code examples are provided with this book.

13.1 Introduction

13.1.1 Languages and architectures

13.2 Optimization

13.2.1 Basic optimization

13.2.2 Automatic optimization during compilation

13.3 Memory caches

13.4 Taylor-Green vortex decay

13.5 Introductory code

13.5.1 Optimizing the introductory code

13.5.2 Data storage, post-processing, and results

13.5.3 Exercises

13.6 LBM algorithm optimizations

13.7 Parallel computing

13.7.1 Multithreading and OpenMP

On computers with one single-core CPU, multitasking operating systems provide the illusion that several programs are executing simultaneously. Such operating systems achieve this illusion by rapidly switching between programs many times per second: they interrupt the running program, save the state of the CPU, load the state of the CPU for another program, and then resume execution of that program. On a system with one or more multi-core CPUs, the operating system can schedule the programs to run truly simultaneously on the available processors. To utilize multiple cores for computational tasks, one may run several programs that communicate with each other. This approach, however, could be inefficient and introduces several programming complexities because operating systems isolate programs to maintain

security and allocate system resources. Many tasks that benefit from parallel execution, including the computational tasks we are interested in, do not require this isolation and are hampered by it. Operating systems therefore provide alternative mechanisms for performing multitasking with less overhead. Threads are units of code that, like a program, are scheduled by the operating system to run simultaneously or share time on the available processors. Unlike a program, they inherit some of the resources of their parent program, such as its memory address space. The common address space is particularly important because it allows threads to share pointers and easily access the same regions of memory.

To facilitate programming and hide operating system-specific details of setting up and managing multiple threads, the OpenMP Architecture Review Board (ARB) [1] publishes and maintains the specifications of a collection of tools called OpenMP (Open MultiProcessing) for C, C++, and Fortran. The specifications describe a collection of special statements called directives, a library of functions, and a set of environment variables that together describe how code execution is split among threads and how data is shared between them. The OpenMP ARB does not provide implementations of these tools. Instead, compiler authors choose whether to support OpenMP directives and provide the required libraries. Support for OpenMP is included in many compilers, both commercial and free, for a wide range of operating systems and computer architectures. In this chapter, we consider features that are available at least since version 3.1 of OpenMP.[2] At the time of writing, a more recent version with additional features is available[**?**].

13.7.1.1 OpenMP directives

The OpenMP directives that are used to describe when and how threads are created to execute blocks of code are structured as special pre-processor directives in C/C++ and comments in Fortran. In this section we consider OpenMP for C/C++.[3] Preprocessor directives are special statements in code that are not a part of the language but rather provide additional information to the compiler. OpenMP directives begin with `#pragma omp`, which is followed by the name of the directive and then a list of options that are called clauses. A long directive, such as one with many clauses, can be split across several lines by using the line continuation character `\`.

This and the following sections introduce the directives and clauses that are essential to using OpenMP and those that are used in a sample parallel LBM code (Section 13.7.1.4). Readers may consult the OpenMP specifications[**?**] or publications about them for a full reference of all features.[4]

---

[1] `http://openmp.org/wp/about-openmp/`

[2] `http://www.openmp.org/mp-documents/OpenMP3.1.pdf`

[3] For convenience, the code in this section is not strictly standard C and uses the convenience of some features of C++ such as variable scope.

[4] One useful online tutorial is provided by Blaise Barney of Lawrence Livermore National Laboratory: `https://computing.llnl.gov/tutorials/openMP/`.

The most important directive in OpenMP is the `parallel` directive that indicates that the subsequent block of code is to be run by several threads. Regions of sequential code, i.e. code that is outside a `parallel` block, are executed by what is called the master thread. Upon starting the `parallel` block, OpenMP creates additional threads to form a team of threads, all of which execute that block of code. The threads are assigned an identification number starting with zero for the master thread. Logic within the `parallel` block can use the thread's identification number, provided by the function `int omp_get_thread_num()`, to perform a different task in each thread. The `int omp_get_num_threads()` function returns the number of threads that are executing the block. Listing 13.1 provides a minimal example of the use of these functions and the `parallel` directive. Upon reaching the end of the `parallel` section, the master thread by default waits until all threads finish and then continues with the sequential code after the `parallel` block.

Listing 13.1: Example code showing the use of the `parallel` directive

```
// ...
// sequential code

#pragma omp parallel
{
    // block of code to be executed by several threads

    // get identification number of thread
    int id = omp_get_thread_num();

    // get number of threads executing block
    int threadcount = omp_get_num_threads();

    // determine task to perform
    // based on id and threadcount
}

// sequential code
// ...
```

The number of threads used to execute a `parallel` section may be specified in several ways. In order of precedence from highest to lowest, the methods are: the `num_threads` clause of the `parallel` section, the number specified by the most recent call to `void omp_set_num_threads(int)`, the value of the `OMP_NUM_THREADS` environment variable, and finally an implementation-specific default that is often the number of available processors. Since it is unlikely that a specific number of threads is best for all systems that a program may run on, the use of the `num_threads` clause is generally not advised. It is more convenient to be able to specify the number of threads at the time the code is executed either by using the environment variable `OMP_NUM_THREADS` or through a configuration option of the program (perhaps a command line option or an option in a configuration file) and a call to `omp_set_num_threads(int)`.

A multithreaded version of the previous LBM code (Section **??**) could now be implemented using only the `parallel` directive. For example, the `for` loops that iterate over the nodes in the domain could be modified so that each thread simultaneously updates a portion of the domain as determined by its identification number. However, it is more convenient, and perhaps also more elegant, to use specialized directives to achieve this task. The `parallel for` directive, shown in Listing 13.2, automates the allocation of iterations of `for` loops to different threads.[5]

Listing 13.2: Example of OpenMP `parallel for` directive

```
#pragma omp parallel for
for(int n = 0; n < N_MAX; ++n)
{
    // iterations executed by different threads
}
```

Like specifying the number of threads for a `parallel` block, clauses or environment variables can be used to indicate the way that the iterations of the `for` loop are split among the threads. With these options, programmers can ensure that the threads are allocated an equal share of the work. When work is shared evenly, processor time is not wasted waiting for some threads to finish. The clauses for the three types of scheduling are:

- `schedule(static,chunk)` where `chunk` is an integer. With `static` scheduling, the threads are allocated `chunk` iterations in order by their identification number. If `chunk` is not specified, the iterations are split as evenly as possible among the threads. This type of scheduling is useful when the workload in each iteration is known to be equal, in which case the threads can be expected to finish their work in the same amount of time when given the same number of iterations to perform.
- `schedule(dynamic,chunk)` In `dynamic` scheduling, each thread first performs the number of iterations specified by `chunk`. When a thread completes its initial allocation, it is assigned the next set of `chunk` iterations until all have been completed. The default value for `chunk` in this case is one. This type of scheduling is useful when workload varies between iterations because it allows threads that happen to be assigned iterations that require less work to perform more iterations.
- `schedule(guided,chunk)` This type of scheduling is similar to `dynamic` scheduling, but the number of iterations allocated to a thread that completes its initial allocation decreases over time, staying proportional to the number of iterations that remain to be performed. For this scheduling option, the parameter `chunk` specifies the minimum number of iterations that can be allocated to a thread and defaults to one.

Another possible clause, `schedule(auto)` defers selection of the iteration allocation scheme to the compiler or OpenMP libraries. For our purposes, `static` scheduling is sufficient because every iteration of the outermost `for` loops that

[5] The `omp parallel for` directive is a shortcut for an `omp for` directive inside a `parallel` block.

we will parallelize performs an equal amount of computations. If the `schedule(runtime)` clause is used, the `OMP_SCHEDULE` environment variable determines how the threads execute the iterations of the loop. The format of the `OMP_SCHEDULE` variable is a string with the name of the scheduling method optionally followed by a comma and the chunk size. The third alternative for setting the scheduling method is the `void omp_set_schedule(omp_sched_t,int)` function. As for setting the number of threads, the clause has the highest precedence, followed by the function call, and the environment variable has lowest precedence.

Now that we have seen how to designate blocks of code that will be executed in parallel by several threads, we need to consider how the threads use variables that they share access to. This is the topic of the next section, Section 13.7.1.2.

13.7.1.2 Data sharing

The `parallel` and `parallel for` directives presented in Section 13.7.1.1 accept clauses that specify how threads handle variables that all threads can access because they are declared outside a `parallel` block. Wrong specifications are a common reason for incorrect behaviour of a multithreaded OpenMP program, and therefore this topic deserves special attention.

Each thread has its own copy of every variable that is declared inside a `parallel` block. Two options are available for variables that are declared before a `parallel` region and are used inside it: the threads may all share access to the one copy of that variable that is used in the sequential regions of the code, or they may each have their own copy. Variables are designated as being shared among threads with the `shared(variable_list)` clause, where `variable_list` is a comma separated list of variable names. When variables are shared, threads may read or write to these variables at any time. Therefore care must be taken to ensure that the memory accesses occur in the correct order. Consider, for example, the statement `x = x+1` as it is being executed by two threads with `x` being a shared variable. The outcome is ill-defined. The two threads might initially read the same value then both write the same result, or one thread might read `x` after the other has already written it, causing the variable to be incremented twice. To specify the intended outcome of such situations, OpenMP has a `#pragma omp critical` directive that defines sections of code than can only be executed by one thread at a time. Threads that reach a `critical` section while another is executing it must wait until that thread finishes. An example that shows the use of this directive is given in Listing 13.3.

Listing 13.3: Example showing the use of the `critical` directive. This code computes the sum of the threads' identification numbers.

```
int x = 0;
#pragma omp parallel shared(x)
{
    int id = omp_get_thread_num();
    #pragma omp critical
    {
```

```
            x = x+id;
        }
    }
    printf("x = %d\n",x);
```

The `private(variable_list)` clause is the second option for variables declared outside a `parallel` block. Threads have their own copies of all variables in the `variable_list`, and all such variables must be initialized in every thread. Alternatively, variables may be listed in a `firstprivate(variable_list)` clause to request that the value in each thread be initialized with the value of the variable prior to entering the `parallel` section.

For convenience, one may also specify that variables not listed in other clauses are by default `shared` using the clause `default(shared)`. However, this is not recommended because it may lead to an incorrect classification for an overlooked variable. Instead, beginners are advised to use `default(none)` which requires that each variable must be explicitly listed in a data sharing clause. If a variable is accidentally omitted, compilation will end with an error, forcing the programmer to evaluate how that variable should be handled.

The data sharing clauses of `parallel for` directives are the same as those for `parallel` directives together with two additional clauses. The first is the `lastprivate(variable_list)` clause that indicates that variables will be treated as private in all threads but the value computed in the final iteration of the loop will be saved to the corresponding variable with the same name outside the `parallel for` block. In other words, after execution of the `for` loop in parallel, the variable has the same value as it would have if the loop had been executed sequentially. Note that the final iteration is not necessarily the iteration that is executed chronologically last by the threads: the thread that is allocated the final iteration might finish earlier than other threads.

The second additional clause that is available in `parallel for` directives simplifies the implementation of many algorithms. In a simulation, one might need to compute perhaps the sum, product, or minimum/maximum of a quantity at every node in the domain, thereby summarizing or "reducing" the values at every node to one value for the whole domain. Though one could write the required code to perform this task with the previously-presented OpenMP concepts, use of the `reduction` memory sharing method simplifies the task and helps avoid common errors. The syntax for the clause is `reduction(op:variable_list)` where `op` is one of the operators: `+`, `*`, `-`, `&`, `|`, `^`, `&&`, `||`, `min`, or `max`. The meanings of these operators are the same as the corresponding C/C++ arithmetic, bitwise, and logical operators. The latter two operators return the minimum or maximum value, respectively. Each thread receives a private variable that is initialized appropriately for the chosen reduction operator (0 for `+` and 1 for `*`; the required initializer for each operator is intuitive and readers may consult the OpenMP standard[**?**] for the details for other operators). At the end of the execution of the `parallel for` section, the value of the variable for the subsequent sequential code is computed by applying the specified operator to the initial value of the variable (before the start of the `parallel for` section) and all the private variables of each thread. Note that re-

ductions involving floating point arithmetic will not necessarily provide exactly the same result as a sequential reduction due to rounding error and the different order of evaluation.

13.7.1.3 Compiling and running OpenMP code

In GCC, support for OpenMP directives and linking with the OpenMP libraries is enabled with the `-fopenmp` compiler flag. To compile a program contained in only one C++ source file, a possible command is

```
g++ -Ofast -fopenmp simulation.cpp -o sim
```

When the source code for a program is split among several files, all files that use OpenMP directives must be compiled with the `-fopenmp` option and the command that links the object files must also have `-fopenmp` to link with the OpenMP libraries. For example, the commands

```
g++ -Ofast -fopenmp source_openmp.cpp -o source_openmp.o
g++ -Ofast source_no_openmp.cpp -o source_no_openmp.o
g++ -fopenmp source_openmp.o source_no_openmp.o -o sim
```

compile one C++ file with OpenMP directives (`source_openmp.cpp`) and one without (`source_no_openmp.cpp`) then link the resulting object files and OpenMP libraries to generate the executable `sim`.

As discussed in the preceding sections (13.7.1.1 and 13.7.1.2), several environment variables can be used to affect the behaviour of programs that use OpenMP. For reference, these may be set on the command line or in a shell script by using the command

```
export OMP_ENV_VAR=value
```

in the Bourne family of shells, which includes `sh` and `bash`, or the command

```
setenv OMP_ENV_VAR value
```

In both these examples, `OMP_ENV_VAR` is the name of the OpenMP environment variable, and `value` is its new value. With the environment variables set as desired, the program may then be run as usual:

```
./sim
```

13.7.1.4 Multithreaded LBM implementation

Modifying the previous simulation code (Section **??**) to support multithreading with OpenMP is fairly straightforward. First, we `#include` the `omp.h` header file that declares the OpenMP functions and variables. We then add some informative output about the OpenMP runtime environment to the `main` function:

```
printf("OpenMP information\n");
printf("  maximum threads: %d\n", omp_get_max_threads());
printf("       processors: %d\n", omp_get_num_procs());
printf("\n");
```

Next, we indicate which `for` loops will be executed by several threads. The most important loop to parallelize in this way is the loop that updates the density distributions every time step in the `stream_collide_save` function:

```
void stream_collide_save(double *f0, double *f1, double *f2,
    double *r, double *u, double *v, bool save)
{
    const double tauinv = 2.0/(6.0*nu+1.0); // 1/tau
    const double omtauinv = 1.0-tauinv;     // 1 - 1/tau

    #pragma omp parallel for default(none) \
        shared(f0,f1,f2,r,u,v,save) schedule(static)
    for(unsigned int y = 0; y < NY; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            // same code as in serial version
        }
    }
}
```

All the pointers and the boolean variable have been specified as `shared` because the same values need to be visible to all threads. Alternatively, these variables could also have been declared as `firstprivate` because their values are never changed. The performance implications of this choice are left to interested readers to investigate. Static scheduling is selected because the time to complete of each iteration of the loop is expected to be nearly the same though caching issues may cause some variation between iterations. Note that the `const` variables `tauinv` and `omtauinv` do not require a data sharing specification. Since they cannot be written to, they are automatically treated as shared. Note that because the pointer variables themselves are not changed (only the memory at the location they point to is altered) they could also be declared `const` in the function declaration. With this qualifier, the type of data sharing for the pointer variables would not need to be indicated.

Though parallelization of additional loops does not have a significant impact on the performance of the whole program, we do so to illustrate several features. The function that initializes the density distribution, `init_equilibrium`,

runs only once, and therefore decreasing its runtime provides a negligible improvement in the total runtime of the program. Similarly, the functions that compute the exact solution and the error in the numerical solution (`taylor_green` and `compute_flow_properties`, respectively) run infrequently compared to the main domain update function. The outer `for` loops in the `taylor_green` and `init_equilibrium` are parallelized identically to the `for` loop of `stream_collide_save` : all variables are shared and the scheduling is static. The parallelization of the `compute_flow_properties` illustrates the use of `reduction` variables. For reference, the complete multithreaded version of this function is shown in Listing13.4. Seven variables in this function are used to compute cumulative sums of quantities at every position in the domain. These variables are therefore listed in a `reduction` clause with the `+` operator. Each thread has a set of private variables, all automatically initialized to zero, that are used to compute the required sums for the nodes handled by each thread. The remainder of the function is the same as in the sequential case (c.f. Listing **??**).

Listing 13.4: Multithreaded version of the `compute_flow_properties` function that illustrates the use of `reduction` variables.

```
void compute_flow_properties(unsigned int t,
                             double *r, double *u, double *v,
                             double *prop)
{
    // prop must point to space for 4 doubles:
    // 0: energy
    // 1: L2 error in rho
    // 2: L2 error in ux
    // 3: L2 error in uy

    double E = 0.0; // kinetic energy

    double sumrhoe2 = 0.0; // sum of error squared in rho
    double sumuxe2  = 0.0; //                          ux
    double sumuye2  = 0.0; //                          uy

    double sumrhoa2 = 0.0; // sum of analytical rho squared
    double sumuxa2  = 0.0; //                    ux
    double sumuya2  = 0.0; //                    uy

    #pragma omp parallel for default(none) shared(t,r,u,v) \
        reduction(+:E,sumrhoe2,sumuxe2,sumuye2, \
                    sumrhoa2,sumuxa2,sumuya2) \
        schedule(static)
    for(unsigned int y = 0; y < NY; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            double rho = r[scalar_index(x,y)];
            double ux  = u[scalar_index(x,y)];
            double uy  = v[scalar_index(x,y)];
```

```
                E += rho*(ux*ux + uy*uy);

                double rhoa, uxa, uya;
                taylor_green(t,x,y,&rhoa,&uxa,&uya);

                sumrhoe2 += (rho-rhoa)*(rho-rhoa);
                sumuxe2  += (ux-uxa)*(ux-uxa);
                sumuye2  += (uy-uya)*(uy-uya);

                sumrhoa2 += (rhoa-rho0)*(rhoa-rho0);
                sumuxa2  += uxa*uxa;
                sumuya2  += uya*uya;
            }
        }

        prop[0] = E;
        prop[1] = sqrt(sumrhoe2/sumrhoa2);
        prop[2] = sqrt(sumuxe2/sumuxa2);
        prop[3] = sqrt(sumuye2/sumuya2);
    }
```

TODO: add a section about running code on a cluster? In next section only show how it works on a workstation with multi cores and CPUs. Refer to MPI section.

13.7.1.5 Performance results

Figure 13.1 presents the speed of the OpenMP version of the simulation code. These simulations were run on the same quad-core Intel Xeon W3550 CPU that was used for the single core benchmarks (Section **??**). To avoid competition for resources with other running programs, the system was first restarted and then only the simulations were run. Representative results are shown for four domain sizes, and each point is an average of three runs. For reference, a dashed line shows ideal parallel performance for which the speed would be proportional to the number of threads. The actual improvement achieved by increasing the number of threads is lower than in the ideal case. Understanding the performance of multithreaded code on shared memory systems is complex because it depends on many system-specific details such as: the number of CPUs present, the number of cores per CPU, cache sizes, the sharing of caches between cores, the sharing of data between caches to avoid access to RAM, the number of memory channels between the CPUs and RAM, and how the operating system schedules the threads on the available cores and CPUs. While the details of how performance depends on the domain size and number of threads are architecture-dependent, a generalization is possible for memory-intense algorithms like LBM simulations: on typical contemporary systems, the available memory bandwidth will be exhausted when simulating large domains before the number of threads equals the number of cores. In Figure 13.1, we see that the speed increases nearly ideally for the two smaller domain sizes, but for the two larger do-

main sizes, the use of four threads provides only a minimal improvement over three threads. The loss of performance appears to coincide with the domain size exceeding the capacity of the L3 cache, which is shared by all cores on the CPU that was used. The lower speed of the smallest domain (32×32) compared to the next smallest (128×128) suggests an inefficiency in the sharing of data between the cores' caches.

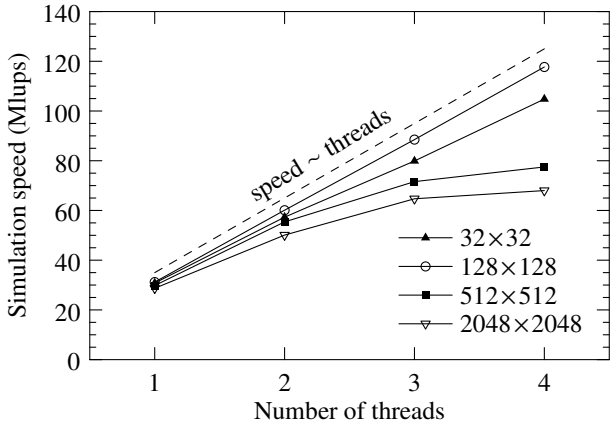


Fig. 13.1: Speed of multithreaded simulations with several domain sizes using 1 to 4 threads running on a quad-core Intel Xeon W3550 CPU.

A detailed discussion of the reasons for the observed performance of the multithreaded simulations is beyond the scope of this chapter. To reach higher speeds, we require multiple CPUs that do not compete for memory bandwidth. This is commonly achieved by connecting many otherwise independent computers, such as small form factor servers, through a high-speed and low-latency network. Programming such systems is the subject of Section 13.7.2.

13.7.1.6 Exercises

13.7.2 Computing clusters and MPI

Computing clusters can be conceptually understood as collections of many computers, often called nodes,[6] that communicate with each other but are otherwise independent. The requirements for high data transfer rates and low latencies, which are the delays in starting a data transfer, have led to the development of specialized interconnect hardware to connect the nodes of a cluster. The most common

[6] Where the term "node" is potentially ambiguous, the terms "computing node" and "lattice node" are used for clarity.

choice of interconnect on contemporary supercomputers, InfiniBand, provides data transfer rates exceeding 10 Gbps (10^{10} bits per second) and latencies on the order of one microsecond. Since a variety of vendors provides InfiniBand and other high-performance interconnects for clusters, programming such systems and adapting code to run on different systems would be rather complex in the absence of a standard programming interface.

The Message Passing Interface (MPI) standards describe a collection of protocols, data structures, and routines for processes to exchange data, called messages, with each other and synchronize their operations. Developed by the MPI Forum[7], the standards provide a uniform method for performing these tasks in a way that hides the details of the hardware devices and operating systems. Code written using MPI is therefore highly portable, and when written carefully, it can efficiently use the resources of an individual shared memory machine, a small clusters of computers, or the world's largest supercomputing clusters[8]. Since the first version of MPI (MPI-1.0) was released in 1994, the standard has undergone many revisions and updates to add new features that facilitate the efficient use of continually evolving hardware capabilities. The most recent version is MPI-3.1 (June 2015). In this chapter, however, we consider only features available since MPI-1, specifically version MPI-1.3[9] from 2008. Implementations of this version are widely available, and it is sufficient for demonstrating the use of clusters for parallel computing and the core capabilities of MPI.

Many implementations of MPI are available from hardware and software vendors as well as open source versions by several groups. This chapter uses only standard MPI features, and the code that is presented has been tested with the open source Open MPI implementation[10]. This chapter does not cover the installation and configuration of clusters or MPI, and we assume readers have access to an operating cluster. Interested readers without access to a corporate or academic research cluster can try the code examples by purchasing resources from commercial cloud computing providers, such as Amazon[11].

In addition to the original reference documents from the MPI Forum[12], documentation is available for Open MPI[13], and from online tutorials[14] and numerous books, such as[**?**].

---

[7] `http://www.mpi-forum.org/`

[8] `http://www.top500.org/`

[9] `http://www.mpi-forum.org/docs/mpi-1.3/mpi-report-1.3-2008-05-30.pdf`

[10] `http://www.open-mpi.org/`

[11] `http://aws.amazon.com/hpc/`

[12] `http://www.mpi-forum.org/docs/docs.html`

[13] `http://www.open-mpi.org/doc/`

[14] Blaise Barney: `https://computing.llnl.gov/tutorials/mpi/`

13.7.2.1 MPI concepts

The MPI standard defines a set of functions, constants, and data structures.[**?**] All names defined by MPI begin with `MPI_`. The functions can be divided into several categories: point-to-point communications, collective communications, and functions for querying and modifying the state of MPI systems. Of the two types of communications functions, point-to-point functions involve sending and receiving messages between two processes. Collective communications operations involve data transfers between a group of processes, such as sending data from one or all processes to all others, or one process may receive data from the others. Examples of these functions will be presented in Section 13.7.2.2.

In MPI-1, the number of processes that execute in parallel is fixed and determined by how many times the MPI-capable executable is launched. During execution, processes may be grouped to help manage work sharing and communication. The MPI functions for communication between processes include a parameter that selects the group of processes that will be involved in the transfers. This parameter is a reference to an MPI object called a communicator that manages communications between the processes in a group. In this chapter we do not perform any grouping of processes, and we use the `MPI_COMM_WORLD` communicator that allows communication between all processes.

Most MPI functions return error codes. Though useful for debugging, these values are discarded in the example code.

13.7.2.2 MPI LBM implementation

We now proceed to modify the serial version of the Taylor-Green vortex decay LBM code (Section **??**) to create a parallel version using MPI. When working with MPI, we typically write one program that is then run several times simultaneously on one or more computers. Each instance of the program is assigned an identification number and performs different tasks based on this number. The first step in programming is to add `#include<mpi.h>` at the beginning of any source files that use MPI. This loads declarations for MPI functions and variables. Next, at the start of the `main` function, we initialize the MPI library (`MPI_Init`; this must be matched by a call to `MPI_Finalize(void)` in every process at the end of the program) and then save both the rank of the process and the number of processes that are running.

```
int rank, nprocs;

// initialize MPI
MPI_Init(&argc,&argv);

// save rank of this process
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

// save number of processes
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
```

Here `MPI_COMM_WORLD` refers to the communicator for the group of all running processes (Section 13.7.2.1). We are therefore requesting the number of processes in this group, and each running instance of the program requests its rank within this group. These ranks range from `0` to `nprocs-1`. We next display the number of processes being used.

```
if(rank == 0)
{
    printf("Simulating Taylor-Green vortex decay\n");
    // ...
    printf("\n");
    printf("MPI information\n");
    printf("        processes: %d\n",nprocs);
    printf("\n");
}
```

Without the initial conditional statement `if(rank == 0)` the output would be unnecessarily repeated by each process. We now synchronize all the processes with the function

```
MPI_Barrier(MPI_COMM_WORLD);
```

In each process, this function does not return until all processes have called it. This synchronization can be thought of as a barrier because no process may proceed until all processes reach this point in the code. The reason for the synchronization is purely cosmetic. Without it, all processes with nonzero rank would skip the initial display statments and continue, causing their output could appear in between the initial output from rank 0.

Next, we use the rank and number of processes to calculate which portion of the domain each process will handle. For simplicity, we consider only one dimensional domain decomposition into subdomains consisting of several complete rows of the domain. Two variables keep track of the allocated portion: `rank_ny` stores the number of rows and `rank_ystart` is the y coordinate of the first row. The simulation domain is split as evenly as possible. If the size of the domain in the y direction is not divisible by the number of processes, each process with rank less than the remainder is assigned one additional row. The code for computing `rank_ny` and `rank_ystart` is:

```
if(rank < NY%nprocs) // ranks that are less than remainder
{
    rank_ny = NY/nprocs+1;
    rank_ystart = rank*rank_ny;
}
else
{
    rank_ny = NY/nprocs;
    rank_ystart = NY-(nprocs-rank)*rank_ny;
}
printf("Rank %d: %d nodes from %d to %d\n",
       rank,rank_ny,rank_ystart,rank_ystart+rank_ny-1);
```

In this domain decomposition strategy, the processes are assigned consecutive blocks of rows in order by their ranks. The final line of the code shows which rows of the simulation domain were allocated to each rank. It is important to note that in general the output will not appear in order by rank. Even on a cluster with identical computational nodes, the processes may run at different speeds for various reasons including the effects of other users' tasks running simultaneously on the same node or even the temperature of the system (CPUs may decrease their clock rates when they heat up to avoid overheating).

With the number of rows for each rank determined, the amount of memory required is calculated and then allocated. Each process allocates two additional rows, one below the assigned portion of the simulation domain, and one above. As will be described later, these rows are used to share data between processes. The memory required by the three types of variables (see Section **??**) is

```
size_t mem_size_0dir   = sizeof(double)*NX*rank_ny;
size_t mem_size_n0dir  = sizeof(double)*NX*(rank_ny+2)
                                       *(ndir-1);
size_t mem_size_scalar = sizeof(double)*NX*rank_ny;
```

Additional layers are not needed for the zero direction and the density and velocity fields because they are not shared across subdomain boundaries. After allocating the memory required for each array in the same way as for the serial code, we proceed with initialization.

The calculation of the initial flow and pressure fields requires knowledge about the absolute location of each lattice node in the domain. Two additional parameters are therefore required in the function that initializes these scalars:

```
void taylor_green(unsigned int t,
                  double *r, double *u, double *v,
                  unsigned int ystart, unsigned int ny)
{
    for(unsigned int y = 0; y < ny; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            taylor_green(t,x,ystart+y,
                         &r[scalar_index(x,y)],
                         &u[scalar_index(x,y)],
                         &v[scalar_index(x,y)]);
        }
    }
}
```

Comparing with Listing **??**, the only differences are that the loop for `y` counts up to the number of layers computed by each process and `ystart` is added to the value of `y` that is passed to the function that calculates the Taylor-Green flow. The density distributions for each lattice node can be computed from the corresponding velocity

and density values without knowledge of the absolute positions of the nodes. Therefore only the number of nodes handled by the process is added as a parameter to the initialization function:

```
void init_equilibrium(double *f0, double *f1,
                      double *r, double *u, double *v,
                      unsigned int ny)
{
    for(unsigned int y = 0; y < ny; ++y)
    {
        // ... same code as in serial version
    }
}
```

The omitted code is identical to that in Listing **??**. However, a detail has been hidden in this implementation. Due to the additional layer below each process's subdomain, the memory indexes are computed with `y+1`:

```
inline size_t fieldn_index(unsigned int x, int y,
                           unsigned int d)
{
    return (ndir-1)*(NX*(y+1)+x)+(d-1);
}
```

One could alternatively rewrite the `for` loop in the `init_equilibrium` function as

```
for(unsigned in y = 1; y < ny+1; ++y)
{
    // ...
}
```

and leave `fieldn_index` unchanged. However, with the method we employ, the same indices `x` and `y` can be used to refer to the same locations in the density distribution arrays as the density and velocity arrays. We leave optimization of the memory address calculations to the compiler, specifically the avoidance of an extra addition operation to compute the address for each memory access.

With the arrays initialized, we turn our attention now to the main loop of the simulation. Though the saving of the scalar fields and computation of the error are also performed in this loop, we defer discussion of the implementation of these features to Sections 13.7.2.4 and 13.7.2.5 because they involve special MPI features.

13.7.2.3 Blocking and non-blocking communications

Before the density distributions in the bottom (`y=0`) and top (`y=ny-1`) rows of each process's subdomain can be updated, the values in these rows must be shared with the processes that update the adjacent subdomains. More specifically, to update its bottom row, each process requires data from the top row of the process that handles

the lower subdomain. Due to the chosen domain decomposition strategy, this data is handled by the process with rank `rank-1`. Similarly, updating the top row requires data from the bottom row of the process with rank `rank+1`. Due to the periodic domain, the ranks of these adjacent processes must be computed as

```
int rankp1 = (rank+1) % nprocs;
int rankm1 = (nprocs+rank-1) % nprocs;
```

The MPI specification describes two main classes of communication functions. The first class consists of functions that wait until after the transfer has been completed before allowing further execution. More specifically, a function that sends data returns only once it is safe to modify the memory that was sent without affecting the transfer. Note that the transfer to the recipient need not have been completed: the data may have been only copied to a temporary buffer. On the receiving side, a function that receives data only returns once the transfer is truly finished and the destination memory may be read. Functions in the second class only initiate the transfer process and return, allowing execution to proceed before the transfer is completed. Functions in the first class are called blocking functions because further execution is blocked until the communication is completed. Though slightly more complicated to use, the second class of communications functions, nonblocking functions, offer two advantages. First, they allow calculations that do not depend on the transferred data to be performed during the transfer. Second, programmers do not need to ensure that one process receives while the other sends to avoid deadlock situations in which the programs cannot proceed. These two advantages will be clarified once code for the two cases has been presented.

The two standard blocking communication functions are `MPI_Send` and `MPI_Recv`, which send and receive data:

```
int MPI_Send(const void *buf,
             int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf,
             int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status)
```

In these functions, the first three parameters describe the nature of the transferred data. The source and destination memory locations of the data are specified by `buf`, the type of data transferred is given by `datatype`, and the number of elements of this type is `count`. In `MPI_Send`, `buf` is a pointer to the first variable that will be sent, while in `MPI_Recv` it is the address where the first received variable will be stored. Among other options given in the MPI standard, common choices for the data type are `MPI_INT`, `MPI_UNSIGNED`, `MPI_FLOAT`, and `MPI_DOUBLE`, which respectively match the C data types `int`, `unsigned int`, `float`, and `double`. In the code that follows, we transfer only `double`s and therefore use `MPI_DOUBLE`. In `MPI_Send`, the parameter `count` specifies the number of variables of the specified

type that are sent; in it specifies the maximum number that can be received. In the code we present, the two `count`s are equal. The next three parameters in the blocking communications functions provide additional information about the transfer. The parameters `dest` and `source` indicate the ranks of the processes within the communicator `comm` to which the data will be sent (`dest` for `MPI_Send`) and from which it will be received (`source` for `MPI_Recv`). On the receiving side, the special source `MPI_ANY_SOURCE` allows receiving data from any rank that is sending data to the receiving rank. In both functions, `tag` may be used by programmers to identify the nature of the transfer: a receive with a particular `tag` can only obtain data that was sent by a call to the sending function with the same `tag`. Alternatively, the tag `MPI_ANY_TAG` allows a function to receive data irrespective of the tag specified on the sending side. Finally, in `MPI_Recv`, `status` is a pointer to an `MPI_Status` structure that the function modifies to provide information about the transfer that was performed.

How can we use these two functions to perform the transfers required by the LBM implementation? Specifically, in each rank we need to: receive the bottom layer from `rankp1`, send the top layer to `rankp1`, receive the top layer from `rankm1`, and send the bottom layer to `rankm1`. Figure 13.2 illustrates the transfers that need to be performed.

A first attempt to implement one pair of the required transfers might be written as follows:

```
// receive from above, i.e. rank+1
MPI_Recv(recv_buffer, count, MPI_DOUBLE,
         rankp1, tag, MPI_COMM_WORLD,
         status);
// send below, i.e. rank-1
MPI_Send(send_buffer, count, MPI_DOUBLE,
         rankm1, tag, MPI_COMM_WORLD);
```

However, if this code were run, each process would first attempt to receive data. Since no process would be sending data, the result would be a deadlock because none of the `MPI_Recv` function calls would return. Switching the order of the calls to `MPI_Recv` and `MPI_Send` would also be problematic:

```
// send below, i.e. rank-1
MPI_Send(send_buffer, count, MPI_DOUBLE,
         rankm1, tag, MPI_COMM_WORLD);
// receive from above, i.e. rank+1
MPI_Recv(recv_buffer, count, MPI_DOUBLE,
         rankp1, tag, MPI_COMM_WORLD,
         status);
```

Depending on details of the MPI implementation used, this version might succeed provided that the size of each transfer is smaller than an internal MPI buffer. In this case, each `MPI_Send` returns after copying its data to this temporary buffer. The `MPI_Recv`s may then run and complete the transfers. If, however, the size of the data to be sent exceeds the buffer size, the result is a deadlock: the `MPI_Send`s

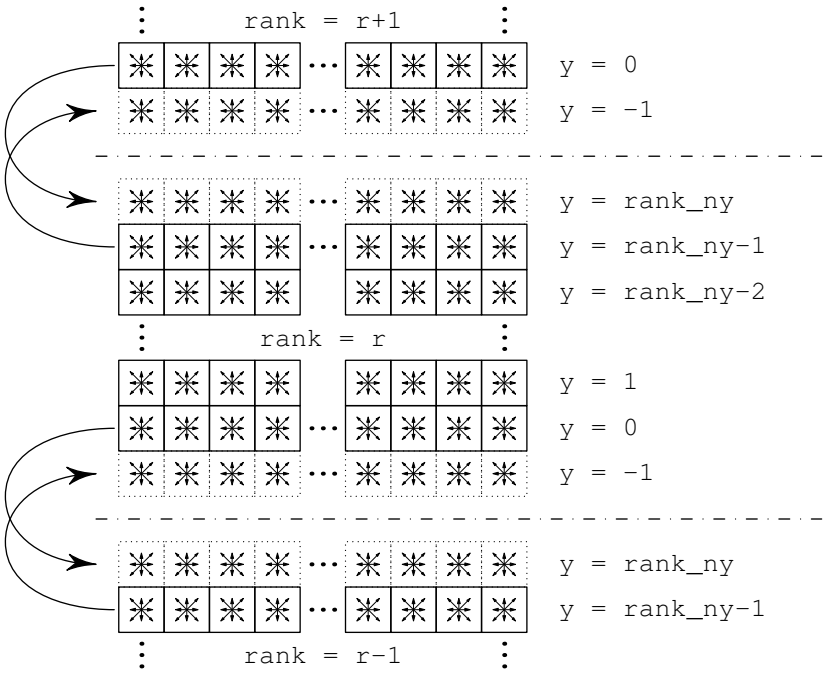


Fig. 13.2: Diagram showing the rows of data that need to be transferred across subdomain boundaries between processes. Boxes with solid outlines denote the nodes of the subdomain updated by each rank, while boxes with dotted outlines denote the extra layers used to store data from adjacent subdomains that are handled by different ranks.

cannot place all the data into a buffer and they therefore stall, waiting for data to be removed from the buffers.

To avoid such deadlocks, we must ensure that some processes receive while others send data. One way to achieve this is to have processes with even ranks first call `MPI_Send` while processes with odd ranks call `MPI_Recv`. Then the roles are reversed, and the even ranks receive data from the odd ranks. Neglecting the details of specifying all parameters, sample code for achieving this is:

```
if(rank % 2 == 0) // even ranks send then receive
{
    // send below, i.e. rank-1
    MPI_Send(send_buffer, count, MPI_DOUBLE,
             rankm1, tag, MPI_COMM_WORLD);
    // receive from above, i.e. rank+1
    MPI_Recv(recv_buffer, count, MPI_DOUBLE,
             rankp1, tag, MPI_COMM_WORLD,
             status);
}
else // odd ranks receive then send
```

```
{
    // receive from above, i.e. rank+1
    MPI_Recv(recv_buffer, count, MPI_DOUBLE,
             rankp1, tag, MPI_COMM_WORLD,
             status);
    // send below, i.e. rank-1
    MPI_Send(send_buffer, count, MPI_DOUBLE,
             rankm1, tag, MPI_COMM_WORLD);
}
```

MPI provides a simpler way to carry out the required transfers. The `MPI_Sendrecv` function combines a send and receive operation and is implemented in a way that prevents deadlocks. This function's parameters are those of `MPI_Send` (excluding the communicator) followed by those of `MPI_Recv`:

```
int MPI_Sendrecv(const void *sendbuf,
                 int sendcount, MPI_Datatype sendtype,
                 int dest, int sendtag,
                 void *recvbuf,
                 int recvcount, MPI_Datatype recvtype,
                 int source, int recvtag,
                 MPI_Comm comm, MPI_Status *status)
```

Using this function, the body of the main simulation loop starts with

```
MPI_Sendrecv(&f1[fieldn_index(0,rank_ny-1,1)],
             transfer_doubles,MPI_DOUBLE,
             rankp1,rank,
             &f1[fieldn_index(0,-1,1)],
             transfer_doubles,MPI_DOUBLE,
             rankm1,rankm1,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);

MPI_Sendrecv(&f1[fieldn_index(0, 0,1)],
             transfer_doubles,MPI_DOUBLE,
             rankm1,rank,
             &f1[fieldn_index(0,rank_ny,1)],
             transfer_doubles,MPI_DOUBLE,
             rankp1,rankp1,
             MPI_COMM_WORLD,MPI_STATUS_IGNORE);

stream_collide_save(f0,f1,f2,rho,ux,uy,save,rank_ny);
```

In this code, `transfer_doubles` is the number of `double`s that are sent and received, which is initialized prior to the `for` loop as

```
size_t transfer_doubles = (ndir-1)*NX;
```

First each rank sends the top layer of its subdomain "upward" to the process with rank `rankp1` and receives this layer from rank `rankm1`, storing it in the layer below the subdomain (`y=-1`). Next the bottom layer (`y=0`) is sent downward to `rankm1` and

received from `rankp1` and stored in the layer above the subdomain (`y=rank_ny`). The tags for these transfers are the ranks of the sending processes. Once these transfers have completed, the whole subdomain is updated. The only changes required in `stream_collide_save()` are an the additional parameter and a modified `for` loop, both changed exactly as for `init_equilibrium` (Listing 13.7.2.2).

TODO: deal with `MPI_STATUS_IGNORE` in text and code

Note that for simplicity we transfer more data than is necessary: only the density distributions of directions that cross subdomain boundaries actually need to be transferred. One way to avoid the unnecessary transfers is to copy the required directions into a temporary buffer, send this buffer, and then load the transferred directions into their corresponding locations on the receiving side. We leave implementing this and assessing the performance benefits as an exercise for interested readers.

The code that accompanies this book also includes an MPI version of the simulation code that uses nonblocking communications functions. In this version, we use the nonblocking alternatives to `MPI_Send()` and `MPI_Recv()`, which are `MPI_Isend()` and `MPI_Irecv()`:

```
// Start a nonblocking send
int MPI_Isend(const void *buf,
              int count, MPI_Datatype datatype,
              int dest, int tag,
              MPI_Comm comm, MPI_Request *request)

// Start a nonblocking receive
int MPI_Irecv(void *buf,
              int count, MPI_Datatype datatype,
              int source, int tag,
              MPI_Comm comm, MPI_Request *request)
```

The parameters are the same as those for the corresponding blocking versions, but exclude a pointer to an `MPI_Status` structure and include a pointer `request` to an `MPI_Request` structure that is used to identify initiated transfers, query their status, and wiat for their completion. These nonblocking functions return as soon as they have completed whatever tasks are needed to initiate the transfers and fill in the fields of the `MPI_Request` structure. To use the nonblocking communication functions, we first create an array of four `MPI_Request`s that are used to identify the four transfers that are subsequently performed, and also an array of four `MPI_Status` objects for the status of each transfer once it is completed:

```
MPI_Request reqs[4];
MPI_Status  stats[4];
```

The nonblocking version of the simulation's main `for` loop starts by initiating the required transfers:

```
MPI_Isend(&f1[fieldn_index(0,rank_ny-1,1)],
          transfer_doubles,MPI_DOUBLE,
          rankp1,rank,  MPI_COMM_WORLD,&reqs[0]);
```

```
MPI_Irecv(&f1[fieldn_index(0,-1,1)],
          transfer_doubles,MPI_DOUBLE,
          rankm1,rankm1,
          MPI_COMM_WORLD,&reqs[1]);

MPI_Isend(&f1[fieldn_index(0,0,1)],
          transfer_doubles,MPI_DOUBLE,
          rankm1,rank,
          MPI_COMM_WORLD,&reqs[2]);
MPI_Irecv(&f1[fieldn_index(0,rank_ny,1)],
          transfer_doubles,MPI_DOUBLE,
          rankp1,rankp1,
          MPI_COMM_WORLD,&reqs[3]);
```

Each call uses its own `MPI_Request` structure in the array `reqs`. With nonblocking functions unlike blocking functions, the order of the calls is irrelevant and deadlocks do not occur if the order is changed. We may now overlap computations with the these communication operations, and update of the interior layers (`y` from `1` to `rank_ny-2` inclusive):

```
stream_collide_save(f0,f1,f2,rho,ux,uy,save,1,rank_ny-1);
```

The `stream_collide_save` function in the nonblocking version differs slightly from the one in the blocking version because the entire domain cannot be updated at once. Therefore the nonblocking version has two parameters, `ystart` and `yend`, that specify the range of layers to update as `ystart` to `yend-1` inclusive:

```
void stream_collide_save(double* f0, double *f1, double *f2,
                         double *r, double *u, double *v,
                         bool save,
                         unsigned int ystart,
                         unsigned int yend)
{
    for(unsigned int y = ystart; y < yend; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            // ...
        }
    }
}
```

Before updating layers `0` and `rank_ny-1`, we must ensure that the transfers have finished. This can be achieved in several ways. The first is to use one call to `MPI_Wait` for every transfer that was initiated. The declaration of this function is

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

This function returns when the transfer identified by `request` has completed. If the completed operation involved receiving data, the supplied `MPI_Status` structure is

updated accordingly. Several variants of `MPI_Wait` are useful when working with multiple pending transfers. These functions are `MPI_Waitany`, `MPI_Waitsome`, and `MPI_Waitall`. Both `MPI_Waitany` and `MPI_Waitsome` wait until at least one transfer has finished; the difference between them is that `MPI_Waitany` provides status information for only one completed transfer, while `MPI_Waitsome` provides information for all the completed transfers. The third variant, `MPI_Waitall` is the one we use:

```
int MPI_Waitall(int count,
                MPI_Request *array_of_requests,
                MPI_Status *array_of_statuses)
```

The parameter `count` indicates how many `MPI_Request`s have been supplied by the pointer `array_of_requests`, and `array_of_statuses` is an array of `MPI_Status`es that must be able to hold at least `count` of them. This variant waits until all the transfers complete and updates the `MPI_Status` array accordingly. We use this function as follows:

```
MPI_Waitall(4,reqs,stats);
```

With the transfers completed, we finally update the bottom and top layers:

```
stream_collide_save(f0,f1,f2,rho,ux,uy,save,0,1);
stream_collide_save(f0,f1,f2,rho,ux,uy,save,
                    rank_ny-1,rank_ny);
```

One subtle but important performance issue deserves particular attention here. Although `MPI_Isend` and `MPI_Irecv` initiate the data transfers, MPI implementations do not necessarily continue the transfers concurrently with subsequent computations. In implementations that do not perform the communications automatically in the background (perhaps by the network hardware devices or a separate thread), the communications delay is incurred when `MPI_Waitall` is called. In this case, communications and computations do not truly overlap, eliminating any possible speed up compared to the blocking version. To avoid this problem, we must ensure that some MPI functions are called in between computations to allow the communications to progress. For this purpose we use the `MPI_Testall` function whose task is similar to that of`MPI_Waitall`[15]: it checks the status of the specified transfers. Unlike `MPI_Waitall`, it only indicates if all transfers have been completed and does not wait for them to finish. This function is declared as:

```
int MPI_Testall(int count,
                MPI_Request *array of requests,
                int *flag,
                MPI_Status *array of statuses)
```

[15] `MPI_Testall` is a variant of `MPI_Test`. The variants of `MPI_Test` are analogous to those of `MPI_Wait`: `MPI_Testany`, `MPI_Testsome`, and `MPI_Testall`.

The parameters are the same as for `MPI_Waitall` except for `flag`, which the function sets to `0` unless all transfers have been completed. If all the transfers are finished, `flag` is set to `1` and the `MPI_status` variables are set accordingly. We use this function after every layer is updated by modifying `stream_collide_save` as follows:

```
void
stream_collide_save_test(double *f0, double *f1, double *f2,
                         double *r, double *u, double *v,
                         bool save,
                         unsigned int ystart,
                         unsigned int yend,
                         int nr,
                         MPI_Request *reqs,
                         MPI_Status *stats)
{
    int com_finished = 0;
    for(unsigned int y = ystart; y < yend; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            // ...
        }
        if(com_finished == 0)
        {
            MPI_Testall(nr,reqs,&com_finished,stats);
        }
    }
}
```

TODO: update code in repo Here `nr` is the number of requests, `reqs` points to an array of `MPI_Request` structures, and `stats` points to an array of `MPI_Status` structures. We use this modified function `stream_collide_save_test` instead of `stream_collide_save` after the calls to `MPI_ISend` and `MPI_IRecv` and before `MPI_Waitall`. We still include `MPI_Waitall` in case the transfers are not completed during the computations. The `com_finished` variable is used to avoid wasting time by calling `MPI_Testall()` after the communications have already been completed. In this code, the calls to `MPI_Testall()` appear to be superfluous: their purpose however, is not to check the status of the communications. Rather, it is ensure that the communications proceed as a side-effect of checking their status. In some MPI implementations, this is not required, so programmers need to consult the documentation for the implementation they use and test their code.

13.7.2.4 Collective communications

In addition to point-to-point communications functions (Section 13.7.2.3), MPI defines a set of functions that facilitate efficient sharing of data between all processes in a group. Some of these functions only transfer data between processes, while oth-

ers also compute a function of the data received from other processes. These latter operations are called reductions (see also Section 13.7.1.2) and can be used, for example, to compute the sum of a value at every point in a simulation domain that has been split among the processes. In such an application of a reduction, each process would compute the sum over the nodes it handles, and then perform a reduction to compute the global sum of all the local sums.

Since the example MPI implementation of an LBM simulation does not use collective operations that only perform communication, in this section we examine what functions MPI defines for these features but do not consider the details of how to use them. Several types of transfers between the member processes of a group are possible. A transfer of a dataset from one process to all others is called a broadcast, and it is performed by the `MPI_Bcast` function. A scatter operation (`MPI_Scatter`) is similar to a broadcast in that one process sends data to all others, but the data sent to each process may be different. The inverse of a scatter, in which one process receives data from each other process, is called a gather (`MPI_Gather`). When every process requires the same set of data from each other process, the `MPI_Allgather` function can be used to perform the same operation as gather, but with every process receiving the combined data set. Finally, every process can send different data to each other process through the use of the `MPI_Alltoall` function. Several variants of these functions allow each process to send a different amount of data to the recipients.

The example code uses reductions in two places. The first place is in the computation of the total energy of the flow field and the error between the numerical and exact solutions for the flow. For this purpose, we use the `MPI_Allreduce` function that performs a reduction and provides the result to every process. Its definition is

```
int MPI_Allreduce(void* sendbuf, void* recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op,
                  MPI_Comm comm)
```

Except for the first two parameters, all processes must call the function with the same parameter values. The parameter `sendbuf` points to the first of `count` variables of type `datatype`. The communicator `comm` specifies which processes participate in the reduction. In every process, the n^{th} value of `recvbuf` receives the result of the reduction applied to the n^{th} values of `sendbuf` from every process. Users may specify a custom reduction operation or use one of the predefined options for `MPI_Op op`: `MPI_SUM` for a sum, `MPI_PROD` for a product, `MPI_MIN` for the minimum, `MPI_MAX` for the maximum, and several others that perform logical operations, bitwise operations, and minimum/maximum computations that also provide the location of the extrema.

The MPI version of the `compute_flow_properties` function is shown in Listing 13.5 (see Listing **??** for the serial version). It requires several additional parameters (the rank of the process as `rank`, the y coordinate of the first row in the subdomain as `ystart`, and the number of rows in the subdomain as `ny`), computes local sums, and then performs a reduction to obtain the global sums.

Listing 13.5: The MPI version of the `compute_flow_properties` function uses a reduction to compute the kinetic energy in the flow domain and the L2 errors of the density and velocity fields.

```
void compute_flow_properties(unsigned int t,
                             double *r, double *u, double *v,
                             double *prop,
                             int rank,
                             unsigned int ystart,
                             unsigned int ny)
{
    // prop must point to space for 4 doubles:
    // 0: energy
    // 1: L2 error in rho
    // 2: L2 error in ux
    // 3: L2 error in uy

    // sums over nodes belonging to this process
    double local_sumdata[7];
    // global sums
    double global_sumdata[7];

    // initialize local sum values
    for(int i = 0; i < 7; ++i)
        local_sumdata[i] = 0.0;

    for(unsigned int y = 0; y < ny; ++y)
    {
        for(unsigned int x = 0; x < NX; ++x)
        {
            double rho = r[scalar_index(x,y)];
            double ux  = u[scalar_index(x,y)];
            double uy  = v[scalar_index(x,y)];

            // add to local sum of energy
            local_sumdata[0] += rho*(ux*ux + uy*uy);

            // compute exact solution at this location
            double rhoa, uxa, uya;
            taylor_green(t,x,ystart+y,&rhoa,&uxa,&uya);

            // add to local sums of errors
            local_sumdata[1] += (rho-rhoa)*(rho-rhoa);
            local_sumdata[2] += (ux-uxa)*(ux-uxa);
            local_sumdata[3] += (uy-uya)*(uy-uya);

            // add to local sums of exact solution
            local_sumdata[4] += (rhoa-rho0)*(rhoa-rho0);
            local_sumdata[5] += uxa*uxa;
            local_sumdata[6] += uya*uya;
        }
    }

    // compute global sums
```

```
    MPI_Allreduce(local_sumdata,global_sumdata,
                  7,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);

    // compute and store final values
    prop[0] = global_sumdata[0];
    prop[1] = sqrt(global_sumdata[1]/global_sumdata[4]);
    prop[2] = sqrt(global_sumdata[2]/global_sumdata[5]);
    prop[3] = sqrt(global_sumdata[3]/global_sumdata[6]);
}
```

The second use of a reduction is in the `main` function, where one is used to compute the total memory allocated by all processes. In this case we use `MPI_Reduce`:

```
int MPI_Reduce(void* sendbuf, void* recvbuf,
               int count, MPI_Datatype datatype,
               MPI_Op op, int root, MPI_Comm comm)
```

This function provides the result of the reduction in `recvbuf` only in the process whose rank is specified by the parameter `root`. The other parameters are the same as those for `MPI_Allreduce`. Prior to starting the main simulation loop, each process computes the amount of memory it allocated:

```
double bytesPerGiB = 1024.0*1024.0*1024.0;
size_t total_mem_bytes = mem_size_0dir + 2*mem_size_n0dir +
    3*mem_size_scalar;
```

After the main simulation loop ends, we use `MPI_Reduce`:

```
size_t global_total_mem_bytes = 0;
MPI_Reduce(&total_mem_bytes,&global_total_mem_bytes,1,
    MPI_LONG_LONG_INT,MPI_SUM,0,MPI_COMM_WORLD);

if(rank == 0)
{
    printf("memory allocated: %.1f (MiB)\n",
           global_total_mem_bytes/bytesPerMiB);
}
```

Every process sends one `MPI_LONG_LONG_INT`[16] to rank 0, which receives the sum in the variable `global_total_mem_bytes`. Finally, rank 0 displays the total memory.

For completeness, two other reduction-type operations are available in MPI. These are `MPI_Reduce_scatter` that performs a reduction and scatters the result to all processes and `MPI_Scan` that computes cumulative reductions in which the value returned to rank n is the result of the reduction on the values from ranks 0 to n inclusive.

[16] This matches the size of `size_t` on the systems used for testing, but it is not portable and may need to be changed for other systems.

13.7.2.5 I/O

If having a single combined output file is absolutely required in an application and the necessary data transfers are sufficiently small or infrequent, the full data sets from every rank could be sent to rank one that writes the data it receives to a file. One fairly obvious pitfall must be avoided when implementing this approach. The writing process most likely cannot allocate enough memory at once to store the whole data set that is spread across all processes. Instead, the process should allocate as much data as is needed for each process individually (to avoid multiple allocations/deallocations, one may use a "maximum" reduction and allocate only once the maximum memory required by any process), and write the data once it is received. As an alternative to this approach, each process could sequentially write its data to the same file. In this case, however, the data structures that represent an open file on one process cannot be shared with other processes. Futhermore, special care must be taken to synchronize the processes and ensure that each correctly adds its data after the data from the previous process.

For efficient performance, it is preferable to avoid unnecessary communication and synchronization. In the code for this section, we take the simplest approach: each process opens its own files to which it saves its portion of the density and velocity fields separately. Though the results could be stored in one file per process, keeping the data separate makes combining the results from all the processes easier. To analyze the data after the simulation has run, one may write a program that loads the results for each process and then computes desired values, generates visualizations, or creates the input files required to perform these tasks with other software. Instead of using a custom program to combine the data, the "raw" data that the program saves can be combined with the common Unix command `cat`, which stands for "concatenate," as follows:

```
cat data_rank0.bin data_rank1.bin data_rank2.bin > data.bin
```

This command combines the data from a hypothetical run of a program with three processes and stores the result in `data.bin`.

The MPI version of the function `save_scalar` that saves data differs minimally from the serial version (see Listing **??**). The differences are: the addition of the process rank as a parameter, the inclusion of the process rank in the output file's name, the addition of a parameter for the number of bytes of memory to be saved (since it is not necessarily the same in each process), and the cosmetic use of `MPI_Barrier` at the end of the function to synchronize the text output from the processes.

This approach for saving data from parallel programs, in which each process saves its own file, has a special efficiency advantage. Each node of a cluster typically has fast local filesystems and shared network filesystems. When one combined data file is created for all processes, it must be in saved in a slower shared filesystem that every process can access. In comparison, writes to local filesystems are significantly faster, and the local files could be examined during a simulation or after it finishes to determine whether a transfer to a shared network filesystem is justified (for example when the simulation was performed at the correct conditions for

a phenomenon to occur). Transfers to a network filesystem could also be entirely avoided if postprocessing software can also be run in a distributed fashion, like the simulations are.

Though it is not the case for the code in this chapter, many simulations need to read data from files during initialization or execution. As for writing, the data to be read by each process can be split into separate files for each process. When this is inconvenient, one rank can read the required data and share it with other processes, using calls to `MPI_Scatter` or `MPI_Send`/`MPI_Isend`.

As an alternative to the file access methods presented so far, MPI-2 includes features for parallel reading and writing of shared files. The interface for using these features, which we do not cover in further detail, is similar to the interface for the communications functions described in this section. The interface provices, for example, functions that allow each process to read from and write to different locations in a file.

13.7.2.6 Compilation and execution

MPI implementations and cluster administrators usually provide tools to assist the compilation and parallel execution of MPI programs. The `mpic++` program is useful for the compilation of C++ code. For example, the MPI progam `sim` can be generated from the source file `sim_mpi.cpp` using the command

```
mpic++ -O3 sim_mpi.cpp -o sim
```

The MPI compilation program is called a wrapper because it does not perform the compilation itself but rather runs a compiler and passes it the required options for finding header files and linking with the MPI libraries. To see which compiler is invoked and all the options that would be used to compile the code, we can use the option `-showme`. For example the command

```
mpic++ -showme -O3 sim_mpi.cpp -o sim
```

shows output that is similar to

```
g++ -O3 sim_mpi.cpp -o sim -I/path/to/include -pthread \
    -L/path/to/lib -lmpi_cxx -lmpi
```

This shows which additional libraries are linked and the paths that are searched for include and library files. Wrappers are also available for other languages, including C (`mpicc`) and Fortran (`mpif90`).

The tasks that users must perform to an MPI program on a cluster depend on the configuration of the cluster and the software that is used to share the computing resources with many users. Readers are therefore advised to consult the documentation for their cluster or ask a system administrator. In general, cluster users submit

requests for computing resources to resource management and job scheduling software that then run their program on a group of the cluster's nodes once the requested processing and memory resources are available. For example, on systems that use TORQUE (Terascale Open-source Resource and QUEue Manager)[17], the command

```
qsub -l nodes=32:ppn=2,pmem=2gb,walltime=12:00:00 \
     ./simulation_script.sh
```

requests two processors each on 32 nodes, 2 GiB of RAM per process for a maximum of 12 hours. The `qsub` command does not wait for the script to run and only places it in a queue for it to wait until resources are available. The scheduler runs the job script `simulation_script.sh` at a time when the requested resources are available for the specified duration. If the script does not finish within the chosen time limit, the scheduler terminates it. A minimal job submission script is:

```
#!/bin/bash

# go to directory from which the script was submitted
cd $PBS_O_WORKDIR

# run MPI program
mpirun ./sim
```

The last command in this script, `mpirun`[18] sets up the execution environment for the MPI program `sim` and starts it as many times as is needed. On many systems the number of times is automatically determined from the options specified to `qsub`, in our case the product of the `nodes` and `ppn` settings. On other systems, it is necessary to explicitly specify the number of processes to start by using `mpirun -np N ./sim` where `N` is the number of processes.

The requested resources can also be specified through the use of special comments that start with `PBS` in the script file. For example, this job script requests the same resources as the previous `qsub` command:

```
#!/bin/bash
#PBS -l nodes=32:ppn=2
#PBS -l pmem=2gb
#PBS -l walltime=12:00:00

# go to directory from which the script was submitted
cd $PBS_O_WORKDIR

# run MPI program
mpirun ./sim
```

This script would be submitted for scheduling using the command `qsub simulation_script.sh`. Options specified in the `qsub` command that submits the script override those inside it.

[17] `http://www.adaptivecomputing.com/products/open-source/torque/`

[18] `mpirun`, `mpiexec`, and `orterun` are synonyms in Open MPI.

In both cases, the job script `simulation_script.sh` must be executable, i.e. the user must have permission to execute the script file. This script file may be created as any plain text file, and then given execution permission with the command

```
chmod u+x simulation_script.sh
```

Commands for setting up the simulation environment and postprocessing the results can also be included in the job script.

In addition to requesting resources, job configuration settings also allow users to choose to receive status updates about the job, such as when it starts and ends, by email. The output of the program is saved to default files or those specified in the `qsub` command or job script.

13.7.2.7 Performance results

13.7.2.8 Exercises

13.7.3 General purpose graphics processing units

13.7.3.1 Exercises

13.8 Concluding remarks

End of chapter discussion.

References

Appendix A
Appendices

NOTE (EMV): There's several possible approaches to appendices:

1. We have just one appendix chapter with a lot of sections
2. We form different appendix chapters, with each covering several subtopics of one overarching topic
3. We have one appendix chapter for each possible topic

I personally prefer option 2, as it is clean and tidy. Option 3 wastes space, while with option 1 we can potentially end up with sections numbered like A.17, which looks silly in my opinion.

A.1 Index notation

Many equations in physics deal with vector quantities, which have both a magnitude and an orientation in physical space. For instance, the simplified form of Newton's second law,

$$\boldsymbol{F} = m\boldsymbol{a}, \tag{A.1}$$

connects the vector quantity of force $\boldsymbol{F}$ and the vector quantity of acceleration $\boldsymbol{a}$, both having the same orientation.

Equations such as this can also be expressed more explicitly as three scalar equations, one for each spatial direction:

$$F_x = ma_x, \quad F_y = ma_y, \quad F_z = ma_z. \tag{A.2}$$

However, it is cumbersome to write all three equations in this way, especially when their only difference is that their index changes between x, y, and z. Instead, we can represent the same equation using only one generic index $\alpha \in \{x, y, z\}$ as

$$F_\alpha = ma_\alpha. \tag{A.3}$$

This style of notation, called *index notation*, retains the explicitness of the notation in Eq. (A.2) while remaining as brief as Eq. (A.1).

With simple vector equations like this, the advantage of index notation might not seem all that great. However, vectors are only first-order *tensors* (scalars being zeroth-order tensors). We can also apply index notation to a second-order tensor (or matrix) $\boldsymbol{A}$ by pointing to a generic element as $A_{\alpha\beta}$, α and β being two generic indices that may or may not be different. Higher-order tensors are equally explicit: A generic element of the third-order tensor $\boldsymbol{R}$ is $R_{\alpha\beta\gamma}$. Indeed, this style of notation lets us immediately see the order of the tensor from the number of unique indices.

Another strength of index notation is that it allows use of the *Einstein summation convention*, where repeating the same index twice in a single term implies summation over all possible values of that index. Thus, the dot product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ can be expressed as

$$a_\alpha b_\alpha = \sum_\alpha a_\alpha b_\alpha = a_x b_x + a_y b_y + a_z b_z = \boldsymbol{a} \cdot \boldsymbol{b}. \tag{A.4}$$

The dot product can be expressed equally briefly in index and vector notation.

The dot product is expressed in index notation using only the Einstein summation convention, while the vector notation uses a specific, dedicated symbol '$\cdot$' to express it. For the dyadic product,

$$\boldsymbol{A} = \boldsymbol{a} \otimes \boldsymbol{b} \quad \Leftrightarrow \quad A_{\alpha\beta} = a_\alpha b_\beta, \tag{A.5}$$

the vector notation requires yet another specific, dedicated symbol '$\otimes$' while the index notation is explicit and clear: The α, β component of the second-order tensor $\boldsymbol{A}$ equals the product of the α component of the vector $\boldsymbol{a}$ and the β component of the vector $\boldsymbol{b}$.

We may also use index notation to generalise coordinate notation: A general component of the spatial coordinate vector $\boldsymbol{x} = (x, y, z) = (x_1, x_2, x_3)$ can be written as x_α. In this way, we can also express *e.g.* gradients in index notation,

$$\nabla\lambda(\boldsymbol{x}) \quad \Leftrightarrow \quad \frac{\partial\lambda(\boldsymbol{x})}{\partial x_\alpha} \quad \Leftrightarrow \quad \partial_\alpha\lambda(\boldsymbol{x}),$$

the third option being a common shorthand for derivatives, used throughout the literature and this book. (Similarly, the time derivative can be expressed using the shorthand $\partial\lambda(t)/\partial t = \partial_t\lambda(t)$.)

Most common vector and tensor operations can be conveniently expressed in index notation, as shown in Tab. A.1. One exception to this convenience is the always inconvenient cross product, which must be expressed using the *Levi-Civita symbol*

$$\varepsilon_{\alpha\beta\gamma} = \begin{cases} +1 & \text{if } (\alpha, \beta, \gamma) \text{ is } (1,2,3), (3,1,2) \text{ or } (2,3,1), \\ -1 & \text{if } (\alpha, \beta, \gamma) \text{ is } (3,2,1), (1,3,2) \text{ or } (2,1,3), \\ 0 & \text{if } \alpha = \beta, \beta = \gamma, \text{ or } \gamma = \alpha. \end{cases} \tag{A.6}$$

Table A.1: Examples of common operations in vector and index notation, including an index notation shorthand for derivatives

Operation	Vector notation	Index notation	Shorthand
Vector dot product	$\lambda = \boldsymbol{a} \cdot \boldsymbol{b}$	$\lambda = a_\alpha b_\alpha$	
Vector outer product	$\boldsymbol{A} = \boldsymbol{a} \otimes \boldsymbol{b}$	$A_{\alpha\beta} = a_\alpha b_\beta$	
Vector cross product	$\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}$	$c_\alpha = \varepsilon_{\alpha\beta\gamma} a_\beta b_\gamma$	
Tensor contraction	$\lambda = \boldsymbol{A} : \boldsymbol{B}$	$\lambda = A_{\alpha\beta} B_{\alpha\beta}$	
Gradient	$\boldsymbol{a} = \nabla\lambda$	$a_\alpha = \partial\lambda/\partial x_\alpha$	$a_\alpha = \partial_\alpha \lambda$
Laplacian	$\Lambda = \nabla^2\lambda$	$\Lambda = \partial^2\lambda/\partial x_\alpha \partial x_\alpha$	$\Lambda = \partial_\alpha \partial_\alpha \lambda$
1st order tensor divergence	$\lambda = \nabla \cdot \boldsymbol{a}$	$\lambda = \partial a_\alpha/\partial x_\alpha$	$\lambda = \partial_\alpha a_\alpha$
2nd order tensor divergence	$\boldsymbol{a} = \nabla \cdot \boldsymbol{A}$	$a_\alpha = \partial A_{\alpha\beta}/\partial x_\beta$	$a_\alpha = \partial_\beta A_{\alpha\beta}$
3rd order tensor divergence	$\boldsymbol{A} = \nabla \cdot \boldsymbol{R}$	$A_{\alpha\beta} = \partial R_{\alpha\beta\gamma}/\partial x_\gamma$	$A_{\alpha\beta} = \partial_\gamma R_{\alpha\beta\gamma}$

However, while the cross product is widely used in fields like electromagnetics, it is far less used for the topics covered in this book.

In this book, we use Greek indices for the Cartesian indices x, y, and z. We also use Roman indices such as i, j, and k for non-Cartesian indices; typically to index discrete velocities as *e.g.* $\boldsymbol{\xi}_i$. Einstein's summation convention is used *only* for the Cartesian indices.

A.2 Details in the Chapman-Enskog analysis

A.2.1 Higher-order terms in the Taylor expanded LBE

In Eq. (4.5) we found the Taylor expansion of the $f_i(\boldsymbol{x} + \boldsymbol{c_i}\Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t)$ terms in the lattice Boltzmann equation to be

$$\sum_{n=1}^{\infty} \frac{\Delta t^n}{n!} \left(\partial_t + c_{i\alpha}\partial_\alpha\right)^n . \tag{A.7}$$

The terms at third order and higher in this series were neglected in the subsequent analysis. If we can show that these terms are at least two orders higher in Kn than the lowest-order terms, this neglection is justified from the ansatz of that section which states that it is only necessary to keep the two lowest orders in Kn. Let us take a closer look.

Recall that the Knudsen number is $\mathrm{Kn} = \ell_{\mathrm{mfp}}/\ell$, where ℓ_{mfp} is the mean free path and ℓ is a macroscopic length scale. It can be related to a similar ratio in times instead of lengths using the speed of sound c_s. As c_s is on the order of the mean particle speed in the gas [1], we have that the mean time between collisions is $\mathcal{T}_{\mathrm{mfp}} = \mathcal{O}(\ell_{\mathrm{mfp}}/c_\mathrm{s})$. Additionally, we can define an acoustic time scale as $\mathcal{T}_{c_\mathrm{s}} = \ell/c_\mathrm{s}$, this

being the time it takes for an acoustic disturbance to be felt across the length scale ℓ. Together, these two relations show that $\mathrm{Kn} = \ell_{\mathrm{mfp}}/\ell = O(\mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_{c_\mathrm{s}})$.

Collisions are the mechanism by which the distribution function is returned to equilibrium, and relatively few collisions are required for this[1] meaning $\tau = O(\mathcal{T}_{\mathrm{mfp}})$. From the space and time discretisation of section 3.4, we know that $\Delta t = O(\tau)$: $\tau/\Delta t$ must be larger than 0.5 for reasons of linear stability, while we should avoid choosing $\tau/\Delta t \gg 1$ for reasons of accuracy, explained in section 4.5.

The *acoustic* time scale $\mathcal{T}_{c_\mathrm{s}} = \ell/c_\mathrm{s}$ is typically shorter than the *advective* time scale $\mathcal{T}_u = \ell/u$, where u is a characteristic fluid velocity. Indeed, it can readily be found that $\mathcal{T}_{c_\mathrm{s}}/\mathcal{T}_u = u/c_\mathrm{s} = \mathrm{Ma}$. Therefore, $\mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_{c_\mathrm{s}} = O(\mathrm{Kn})$ and $\mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_u = O(\mathrm{Kn} \times \mathrm{Ma})$; these two ratios scale at the same order in the Knudsen number.

With this said, we can now take another look at the terms in Eq. (4.5) and examine their order in the Knudsen number, allowing the characteristic time scale to be either advective or acoustic:

$$\begin{aligned}
\text{Advective:} \quad & O(\Delta t \partial_t f_i) \sim O(\tau/\mathcal{T}_u) \sim O(\mathrm{Ma}\, \mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_{c_\mathrm{s}}) \sim O(\mathrm{Ma} \times \mathrm{Kn}) \\
\text{Acoustic:} \quad & O(\Delta t \partial_t f_i) \sim O(\tau/\mathcal{T}_{c_\mathrm{s}}) \sim O(\mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_{c_\mathrm{s}}) \sim O(\mathrm{Kn}) \\
& O(\Delta t c_{i\alpha} \partial_\alpha f_i) \sim O(\tau c_\mathrm{s}/\ell) \sim O(\mathcal{T}_{\mathrm{mfp}}/\mathcal{T}_{c_\mathrm{s}}) \sim O(\mathrm{Kn})
\end{aligned} \tag{A.8}$$

Consequently, we have that $\Delta t^n(\partial_t + c_{i\alpha}\partial_\alpha)^n f_i$ scales with Kn^n. Neglecting the $O\left(\Delta t^3[\partial_t + c_{i\alpha}\partial_\alpha]^3\right)$ terms and higher-order terms in Eq. (4.5) is therefore consistent with our ansatz of keeping only terms of the two lowest orders in Kn.

A.2.2 The moment perturbation

To be able to find the macroscopic momentum equation through the Chapman-Enskog analysis in section 4.1, we must determine the moment $\Pi^{(1)}_{\alpha\beta}$. This can be found from Eq. (4.9c) as

$$\Pi^{(1)}_{\alpha\beta} = -\tau\left(\partial^{(1)}_t \Pi^{\mathrm{eq}}_{\alpha\beta} + \partial^{(1)}_\gamma \Pi^{\mathrm{eq}}_{\alpha\beta\gamma}\right). \tag{A.9}$$

From this we would like to find an explicit expression for $\Pi^{(1)}_{\alpha\beta}$ through the macroscopic quantities ρ and $\boldsymbol{u}$ and their derivatives.

We already know the two equilibrium moments in Eq. (A.9) explicitly,

$$\Pi^{\mathrm{eq}}_{\alpha\beta} = \rho u_\alpha u_\beta + \rho c_\mathrm{s}^2 \delta_{\alpha\beta}, \quad \Pi^{\mathrm{eq}}_{\alpha\beta\gamma} = \rho c_\mathrm{s}^2\left(u_\alpha \delta_{\beta\gamma} + u_\beta \delta_{\alpha\gamma} + u_\gamma \delta_{\alpha\beta}\right). \tag{A.10}$$

Since we would like the resulting momentum equation to be similar to the Euler and Navier-Stokes equations, all time derivatives in $\Pi^{(1)}_{\alpha\beta}$ should be eliminated. This can be done using Eq. (4.9), which we can write more explicitly as

[1] We show in Section 12.1.1 that the quantity of *viscous relaxation time* τ_{vi} is $O(\tau)$, and it is shown elsewhere [2] that $\tau_{\mathrm{vi}} = O(\mathcal{T}_{\mathrm{mfp}})$.

$$\partial_t^{(1)}\rho = -\partial_\alpha^{(1)}\left(\rho u_\alpha\right), \quad \partial_t^{(1)}(\rho u_\alpha) = -\partial_\beta^{(1)}\left(\rho u_\alpha u_\beta + \rho c_\mathrm{s}^2\delta_{\alpha\beta}\right). \tag{A.11}$$

We also need to make use of a corollary of the product rule; if ∂_* is a generic derivative and a, b, and c are generic variables, then

$$\partial_* abc = a\partial_* bc + b\partial_* ac - ab\partial_* c. \tag{A.12}$$

The following derivation is simplified by our use of the isothermal equation of state $p = c_\mathrm{s}^2\rho$, where the pressure and density are linearly related through the constant c_s^2. In other cases, we would have to treat the pressure in a more complicated fashion. For monatomic gases, we would need to express pressure changes using the conservation equation for translational energy [3].

We will now resolve the two equilibrium moment derivatives in Eq. (A.9) separately, starting with the one which is the simplest to resolve:

$$\begin{aligned}\partial_\gamma^{(1)}\Pi_{\alpha\beta\gamma}^{\mathrm{eq}} &= \partial_\gamma^{(1)}\left(\rho c_\mathrm{s}^2\left[u_\alpha\delta_{\beta\gamma} + u_\beta\delta_{\alpha\gamma} + u_\gamma\delta_{\alpha\beta}\right]\right)\\ &= c_\mathrm{s}^2\left(\partial_\beta^{(1)}\rho u_\alpha + \partial_\alpha^{(1)}\rho u_\beta\right) + c_\mathrm{s}^2\delta_{\alpha\beta}\partial_\gamma^{(1)}(\rho u_\gamma).\end{aligned} \tag{A.13}$$

The other equilibrium moment derivative is more complicated, and we will resolve it in steps. First of all, we apply Eq. (A.12) and find

$$\begin{aligned}\partial_t^{(1)}\Pi_{\alpha\beta}^{\mathrm{eq}} &= \partial_t^{(1)}(\rho u_\alpha u_\beta + \rho c_\mathrm{s}^2\delta_{\alpha\beta})\\ &= u_\alpha\partial_t^{(1)}(\rho u_\beta) + u_\beta\partial_t^{(1)}(\rho u_\alpha) - u_\alpha u_\beta\partial_t^{(1)}\rho + c_\mathrm{s}^2\delta_{\alpha\beta}\partial_t^{(1)}\rho.\end{aligned} \tag{A.14a}$$

Then we apply Eq. (A.11) to replace the time derivatives and subsequently rearrange:

$$\begin{aligned}\partial_t^{(1)}\Pi_{\alpha\beta}^{\mathrm{eq}} &= -u_\alpha\partial_\gamma^{(1)}\left(\rho u_\beta u_\gamma + \rho c_\mathrm{s}^2\delta_{\beta\gamma}\right) - u_\beta\partial_\gamma^{(1)}\left(\rho u_\alpha u_\gamma + \rho c_\mathrm{s}^2\delta_{\alpha\gamma}\right)\\ &\quad + u_\alpha u_\beta\partial_\gamma^{(1)}(\rho u_\gamma) - c_\mathrm{s}^2\delta_{\alpha\beta}\partial_\gamma^{(1)}(\rho u_\gamma)\\ &= -\left[u_\alpha\partial_\gamma^{(1)}(\rho u_\beta u_\gamma) + u_\beta\partial_\gamma^{(1)}(\rho u_\alpha u_\gamma) - u_\alpha u_\beta\partial_\gamma^{(1)}(\rho u_\gamma)\right]\\ &\quad - c_\mathrm{s}^2\left(u_\alpha\partial_\beta^{(1)}\rho + u_\beta\partial_\alpha^{(1)}\rho\right) - c_\mathrm{s}^2\delta_{\alpha\beta}\partial_\gamma^{(1)}(\rho u_\gamma).\end{aligned} \tag{A.14b}$$

Finally, the bracketed terms can be simplified by using Eq. (A.11) in reverse, giving

$$\partial_t^{(1)}\Pi_{\alpha\beta}^{\mathrm{eq}} = -\partial_\gamma^{(1)}\left(\rho u_\alpha u_\beta u_\gamma\right) - c_\mathrm{s}^2\left(u_\alpha\partial_\beta^{(1)}\rho + u_\beta\partial_\alpha^{(1)}\rho\right) - c_\mathrm{s}^2\delta_{\alpha\beta}\partial_\gamma^{(1)}(\rho u_\gamma). \tag{A.14c}$$

Now that we have explicit forms of the two equilibrium moment derivative terms in Eq. (A.13) and Eq. (A.14c), we insert them into Eq. (A.9). After using the product rule and having some terms cancel, we end up with the explicit expression

$$\Pi_{\alpha\beta}^{(1)} = -\tau\left[\rho c_\mathrm{s}^2\left(\partial_\beta^{(1)}u_\alpha + \partial_\alpha^{(1)}u_\beta\right) - \partial_\gamma^{(1)}\left(\rho u_\alpha u_\beta u_\gamma\right)\right]. \tag{A.15}$$

The last term is a kind of error term: It would have been entirely cancelled if $\Pi^{\text{eq}}_{\alpha\beta\gamma}$ in Eq. (A.10) had contained the $\rho u_\alpha u_\beta u_\gamma$ term which it includes in the exact kinetic theory. The reason why this term is missing is that the equilibrium distribution f_i^{eq} has been truncated to $O(u^2)$. This truncation was done to allow using smaller velocity sets like D2Q9, D3Q15, D3Q19, and D3Q27 without any undesirable anisotropy. This point is discussed further in Section 4.2.1.

In other words, removing the $O(u^3)$ error term in Eq. (A.15) would require using an extended lattice. This would slow down computations and make boundary conditions more difficult to deal with. However, recent work suggests that this error term can also be nearly cancelled by using a modified collision operator where τ depends on $\boldsymbol{u}$ [4].

A.2.3 Chapman-Enskog analysis for the MRT collision operator

The Chapman-Enskog analysis covered in Section 4.1 assumes the use of the BGK collision operator. However, it is not that much more difficult to perform the analysis for the very general multiple-relaxation-time (MRT) collision operator described in Chapter 10. We will here show how its Chapman-Enskog analysis differs from that in Section 4.1. (We will not give a full analysis here, only highlight the differences to the one given previously.) The approach we describe here can be used for any collision operator that can be expressed using the MRT formalism.

The fundamental difference to the BGK analysis is that MRT collision operators can have different moments. The analysis relies on the three collision operator moments

$$\sum_i \Omega_i = 0, \quad \sum_i c_{i\alpha} = 0, \quad \sum_i c_{i\alpha} c_{i\beta} \Omega_i. \tag{A.16}$$

Of these moments, the first two are zero due to mass and momentum conservation in collisions. For the BGK collision operator, the third moment becomes $-(\Delta t/\tau)\Pi^{\text{neq}}_{\alpha\beta}$, with $\Pi_{\alpha\beta} = \sum_i c_{i\alpha} c_{i\beta} f_i$. As we shall see, the results of the corresponding MRT analysis hinge on the differences in this moment.

We can find this moment by left-multiplying an MRT collision operator $\boldsymbol{\Omega} = -\boldsymbol{M}^{-1}\boldsymbol{S}\boldsymbol{M}\,(\boldsymbol{f} - \boldsymbol{f}^{\text{eq}})$ individually with the row vectors $\boldsymbol{M}^{(\Pi_{xx})}$, $\boldsymbol{M}^{(\Pi_{yy})}$, and $\boldsymbol{M}^{(\Pi_{xy})}$, where $M_i^{(\Pi_{\alpha\beta})} = c_{i\alpha} c_{i\beta}$. (While it is feasible to find $\boldsymbol{S}\boldsymbol{M}\,(\boldsymbol{f} - \boldsymbol{f}^{\text{eq}})$ analytically, *e.g.* $-\boldsymbol{M}^{(\Pi_{xx})}\boldsymbol{M}^{-1}$ is best found numerically.)

Thus, using the Hermite polynomial-based MRT approach from Section 10.4.1, we can find after some algebra that

$$\sum_i c_{i\alpha} c_{i\beta} \Omega_i = -\omega_\nu \Pi^{\text{neq}}_{\alpha\beta} - \frac{\omega_\zeta - \omega_\nu}{2} \delta_{\alpha\beta} \Pi^{\text{neq}}_{\gamma\gamma}. \tag{A.17}$$

Thus, only the relaxation rates ω_ν and ω_ζ affect the macroscopic momentum equation at the Navier-Stokes level, and if $\omega_\nu = \omega_\zeta$ this equation is equivalent with that of the BGK collision operator with $\omega_\nu = 1/\tau$. Using the Gram-Schmidt approach

in Section 10.4.2 instead, we find the same result as in Eq. (A.17) with $\omega_\zeta \to \omega_e$. Except for this tiny change in notation, the Chapman-Enskog procedure for the Hermite and Gram-Schmidt approaches are therefore identical.

Using a generic collision operator Ω_i, the LBE after Taylor expansion and some algebra becomes

$$\Delta t \left(\partial_t + c_{i\alpha}\partial_\alpha\right) f_i = \Omega_i - \Delta t \left(\partial_t + c_{i\alpha}\partial_\alpha\right) \frac{\Delta t}{2} \Omega_i. \tag{A.18}$$

instead of Eq. (4.6). Expanding f_i and the derivatives, the different moments at different orders in ϵ of this equation become as in Eq. (4.9) and Eq. (4.11), except that two equations have a few extra terms stemming from Eq. (A.17):

$$\partial_t^{(1)} \Pi_{\alpha\beta}^{\mathrm{eq}} + \partial_\gamma^{(1)} \Pi_{\alpha\beta\gamma}^{\mathrm{eq}} = -\omega_\nu \Pi_{\alpha\beta}^{(1)} - \frac{\omega_\zeta - \omega_\nu}{2} \delta_{\alpha\beta} \Pi_{\gamma\gamma}^{(1)}, \tag{A.19a}$$

$$\partial_t^{(2)} (\rho u_\alpha) = -\partial_\beta \left[\left(1 - \frac{\omega_\nu \Delta t}{2}\right) \Pi_{\alpha\beta}^{(1)} - \frac{(\omega_\zeta - \omega_\nu)\Delta t}{4} \delta_{\alpha\beta} \Pi_{\gamma\gamma}^{(1)} \right]. \tag{A.19b}$$

Using the same procedure as in Section A.2.2, we can find from Eq. (A.19a) that

$$\Pi_{\alpha\beta}^{(1)} = -\frac{\rho c_\mathrm{s}^2}{\omega_\nu} \left(\partial_\alpha u_\beta + \partial_\beta u_\alpha\right) - \frac{1}{2} \left(\frac{\omega_\zeta}{\omega_\nu} - 1\right) \delta_{\alpha\beta} \Pi_{\gamma\gamma}^{(1)}. \tag{A.20}$$

This can be made more explicit by multiplying with $\delta_{\alpha\beta}$. As we are using the two-dimensional D2Q9 lattice, $\delta_{\alpha\beta}\delta_{\alpha\beta} = \delta_{\gamma\gamma} = 2$, and after some rearranging we find

$$\Pi_{\gamma\gamma}^{(1)} = -\frac{2\rho c_\mathrm{s}^2}{\omega_\zeta} \partial_\gamma u_\gamma. \tag{A.21}$$

When re-assembling the different orders in ϵ the momentum equation, we find that the resulting viscous stress tensor $\sigma'_{\alpha\beta}$ is given by the right-hand side of Eq. (A.19b). After some algebra we find

$$\begin{aligned} \sigma'_{\alpha\beta} &= -\left(1 - \frac{\omega_\nu \Delta t}{2}\right) \Pi_{\alpha\beta}^{(1)} + \frac{(\omega_\zeta - \omega_\nu)\Delta t}{4} \delta_{\alpha\beta} \Pi_{\gamma\gamma}^{(1)} \\ &= \rho c_\mathrm{s}^2 \left(\frac{1}{\omega_\nu} - \frac{\Delta t}{2}\right) \left(\partial_\alpha u_\beta + \partial_\beta u_\alpha - \frac{2}{3} \delta_{\alpha\beta} \partial_\gamma u_\gamma\right) \\ &\quad + \rho c_\mathrm{s}^2 \left[\left(\frac{1}{\omega_\zeta} - \frac{\Delta t}{2}\right) - \frac{1}{3} \left(\frac{1}{\omega_\nu} - \frac{\Delta t}{2}\right) \right] \delta_{\alpha\beta} \partial_\gamma u_\gamma. \end{aligned} \tag{A.22}$$

Comparing this with Eq. (1.14) gives us the dynamic shear and bulk viscosities

$$\eta = \rho c_\mathrm{s}^2 \left(\frac{1}{\omega_\nu} - \frac{\Delta t}{2}\right), \quad \eta_\mathrm{B} = \rho c_\mathrm{s}^2 \left(\frac{1}{\omega_\zeta} - \frac{\Delta t}{2}\right) - \frac{\eta}{3}. \tag{A.23}$$

This result can be easily confirmed by simulating a free wave as described in Section 12.1.3 and verifying that its attenuation varies with η and η_B as predicted.

A.3 Taylor-Green vortex flow

The decaying *Taylor-Green vortex flow* (TGVF) solves the incompressible Navier-Stokes equations TODO (TK): add reference to NSE. As the TGVF is known analytically, it is often used as benchmark test for Navier-Stokes solvers.

The TGVF is unsteady and fully periodic in a domain of size $\ell_x \times \ell_y$. Formulated in two spatial dimensions its velocity and pressure fields read

$$\boldsymbol{u}(\boldsymbol{x}, t) = u_0 \begin{pmatrix} -\sqrt{k_y/k_x}\cos(k_x x)\sin(k_y y) \\ \sqrt{k_x/k_y}\sin(k_x x)\cos(k_y y) \end{pmatrix} \mathrm{e}^{-t/t_\mathrm{d}} \tag{A.24}$$

and

$$p(\boldsymbol{x}, t) = p_0 - \rho\frac{u_0^2}{4}\left[\frac{k_y}{k_x}\cos(2k_x x) + \frac{k_x}{k_y}\cos(2k_y y)\right]\mathrm{e}^{-2t/t_\mathrm{d}}. \tag{A.25}$$

Here, u_0 is the initial velocity scale, $k_{x,y} = 2\pi/\ell_{x,y}$ are the components of the wave vector $\boldsymbol{k}$ and

$$t_\mathrm{d} = \frac{1}{\nu(k_x^2 + k_y^2)} \tag{A.26}$$

is the vortex decay time. The pressure average p_0 is arbitrary and does not enter the Navier-Stokes equations. The initial state is defined by $\boldsymbol{u}(\boldsymbol{x}, 0)$ and $p(\boldsymbol{x}, 0)$.

Exercise A.1. Show that the velocity field in eq. (A.24) leads to a deviatoric stress tensor with components

$$\begin{aligned} \sigma_{xx} &= 2\rho\nu u_0\sqrt{k_x k_y}\sin(k_x x)\sin(k_y y)\,\mathrm{e}^{-t/t_\mathrm{d}}, \\ \sigma_{xy} &= \rho\nu u_0\left(\sqrt{k_x^3/k_y} - \sqrt{k_y^3/k_x}\right)\cos(k_x x)\cos(k_y y)\,\mathrm{e}^{-t/t_\mathrm{d}}, \\ \sigma_{yx} &= \sigma_{xy}, \\ \sigma_{yy} &= -\sigma_{xx}. \end{aligned} \tag{A.27}$$

In particular this means that the stress tensor is symmetric and traceless. Furthermore, σ_{xy} and σ_{yx} vanish if $k_x = k_y$, *i.e.* if $\ell_x = \ell_y$. This means that $\ell_x \neq \ell_y$ should be chosen if one wants to investigate the accuracy of the off-diagonal stress tensor components.

Exercise A.2. Show that the velocity and pressure in eq. (A.24) and eq. (A.25) solve the incompressible Navier-Stokes equations. Which pairs of terms cancel each other?

A.4 Gauss-Hermite quadrature

One of the most useful features of Hermite polynomials for numerical integration is the Gauss-Hermite quadrature rule [5]: the integral of any 1D function $f(x)$ multiplied by the weight function $\omega(x)$ (*cf.* eq. (3.24)) can be approximated by a finite series of function values in certain points x_i, also called *abscissae*:

$$\int_{-\infty}^{\infty} \omega(x) f(x)\,\mathrm{d}x \approx \sum_{i=1}^{q} w_i f(x_i). \tag{A.28}$$

The accuracy of the integration depends on the values and number q of point values x_i. If one chooses x_i as the n roots of the Hermite polynomials of order n, *i.e.* $H^{(n)}(x_i) = 0$ and $q = n$, then it is guaranteed that any polynomial $P^{(N)}(x)$ of order $N = 2n - 1$ can be integrated exactly:

$$\int_{-\infty}^{\infty} \omega(x) P^{(N)}(x)\,\mathrm{d}x = \sum_{i=1}^{n} w_i P^{(N)}(x_i). \tag{A.29}$$

The above weights can be found as [6]:

$$w_i = \frac{n!}{\left(nH^{(n-1)}(x_i)\right)^2}. \tag{A.30}$$

The 1D abscissae and weigths required to integrate polynomials up to fifth order are shown in tab. A.2.

Table A.2: Abscissae x_i and weights w_i for exact integration of 1D polynomials up to fifth order.

number of abscissae	polynomial degree	abscissae	weights
n	$N = 2n - 1$	x_i	w_i
1	1	0	1
2	3	± 1	1/2
3	5	0	2/3
		$\pm\sqrt{3}$	1/6

Example A.1. To integrate a third-order polynomial $P^{(3)}(x)$, one needs $n = 2$, and therefore the polynomial $H^{(2)}(x)$ with two abscissae points at ± 1 (*cf.* tab. A.2). Together with the weights $w_{1,2} = 1/2$ for $H^{(2)}$, one obtains

$$\int_{-\infty}^{\infty} \omega(x) P^{(3)}(x)\,\mathrm{d}x = \frac{1}{2} P^{(3)}(+1) + \frac{1}{2} P^{(3)}(-1). \tag{A.31}$$

Those abscissae allow to obtain the most common lattices for the LBM as will be described below. More elaborate examples and advanced lattices can be found in the seminal work of Shan *et al.* [7] or in [8].

In order to generalise the above-mentioned procedure to d spatial dimensions, we recall some basic properties of tensor products. The product of two tensorial structures $\boldsymbol{A}^{(n)}$ and $\boldsymbol{B}^{(m)}$ of orders n and m and indices $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_m$, respectively, is the tensorial structure $\boldsymbol{C}^{(n+m)}$ of order $n+m$ with indices $\alpha_1, \ldots, \alpha_n$, $\beta_1, \ldots, \beta_m$:

$$\boldsymbol{C}^{(n+m)} = \boldsymbol{A}^{(n)}\boldsymbol{B}^{(m)}, \quad C^{(n+m)}_{\alpha_1\ldots\alpha_n\beta_1\ldots\beta_m} = \sum A^{(n)}_{\alpha'_1\ldots\alpha'_n} B^{(m)}_{\beta'_1\ldots\beta'_m}, \tag{A.32}$$

where all indices form all possible combinations, *i.e.* $\{\alpha'_i, \beta'_j\} \in [\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m]$ $(i = 1 \ldots n,\, j = 1 \ldots m)$.

With this, the extension to multiple dimensions is straightforward. Any polynomial of order N in d-dimensional space can be written in the form

$$P^{(N)}(\boldsymbol{x}) = \sum_{N_1+\ldots+N_d\leq N} a_{N_1\ldots N_d} x_1^{N_1} \cdots x_d^{N_d}, \tag{A.33}$$

where the $a_{N_1\ldots N_d}$ are real coefficients and $\{N_1, \ldots, N_d\}$ are integers.

The integral of such a polynomial multiplied by the multidimensional weight function $\omega(\boldsymbol{x})$ can be decomposed into the summation of integrals with 1D weight functions $\omega(x)$:

$$\begin{aligned}\int \omega(\boldsymbol{x}) P^{(N)}(\boldsymbol{x})\,\mathrm{d}^d x &= \int \frac{1}{(2\pi)^{d/2}} \mathrm{e}^{-\left(x_1^2+\ldots+x_d^2\right)/2} \sum a_{N_1\ldots N_d} x_1^{N_1}\cdots x_d^{N_d}\,\mathrm{d}^d x \\ &= \sum a_{N_1\ldots N_d} \prod_{j=1}^{d} \int \frac{x_j^{N_j}}{\sqrt{2\pi}} \mathrm{e}^{-x_j^2/2}\,\mathrm{d}x_j.\end{aligned} \tag{A.34}$$

Each of the 1D integrals can now be decomposed using the Gauss-Hermite quadrature rule from eq. (A.29):

$$\sum a_{N_1\ldots N_d} \prod_{j=1}^{d} \int \frac{x_j^{N_j}}{\sqrt{2\pi}} \mathrm{e}^{-x_j^2/2}\,\mathrm{d}x_j = \sum_{N_1+\ldots+N_d\leq N} a_{N_1\ldots N_d} \prod_{j=1}^{d} \sum_{i=1}^{n_j} w_{i,j} x_{i,j}^{N_j}. \tag{A.35}$$

Let us assume that all 1D integrals are discretised using the same 1D Hermite polynomial, *i.e.* $n_1 = \ldots = n_d = n$, $x_{i,1} = \ldots = x_{i,d} = x_i$ and $w_{i,1} = \ldots = w_{i,d} = w_i$:

$$\prod_{j=1}^{d} \sum_{i=1}^{n_j} w_{i,j} x_{i,j}^{N_j} = \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} w_{i_1}\cdots w_{i_d} x_{i_1}^{N_1}\cdots x_{i_d}^{N_d}. \tag{A.36}$$

Introducing new multi-dimensional abscissae as $\boldsymbol{x}_i = (x_{i_1}, \ldots, x_{i_D})$ and weights $w_i = w_{i_1}\cdots w_{i_D}$ allows us to obtain the multi-dimensional Gauss-Hermite quadrature rule:

$$\int \omega(\boldsymbol{x}) P^{(N)}(\boldsymbol{x}) \, \mathrm{d}^d x = \sum_{i=1}^{n^d} w_i P^{(N)}(\boldsymbol{x}_i). \tag{A.37}$$

Example A.2. Let us demonstrate the multi-dimensional Gauss-Hermite quadrature rule by integrating the polynomial $P^{(3)}(\boldsymbol{x}) = x^2 y$ in 2D. Since $N = 3$, we need two abscissae ($n = 2$) for each dimension, *i.e.* $n^d = 2^2 = 4$ in total. First we note that, for a 1D polynomial of third order, we get

$$\int w(x) x^N \, \mathrm{d}x = w_1 x_1^N + w_2 x_2^N = \frac{1}{2}(-1)^N + \frac{1}{2} 1^N \tag{A.38}$$

for $N \leq 3$. In the last step we have used the known weights and abscissae from tab. A.2. It follows that

$$\begin{aligned} \int \frac{1}{2\pi} \mathrm{e}^{-(x^2+y^2)/2} x^2 y \, \mathrm{d}x \, \mathrm{d}y &= \int \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2} x^2 \, \mathrm{d}x \int \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-y^2/2} y \, \mathrm{d}y \\ &= \left(w_1 x_1^2 + w_2 x_2^2 \right) \left(w_1 y_1 + w_2 y_2 \right) . \end{aligned} \tag{A.39}$$

Using $w_1 = w_2 = \frac{1}{2}$, $x_1 = y_1 = -1$ and $x_2 = y_2 = 1$, it is straightforward to show that the result is zero.

We have seen in section 3.3 that one needs to integrate fifth-order polynomials in order to obtain Navier-Stokes behaviour. This implies $N = 5$ and $n = 3$. From tab. A.3 we find that $n = 3$ leads to the abscissae points 0 and $\pm\sqrt{3}$. After rescaling the velocities to get rid of the factor $\sqrt{3}$ (*cf.* section 3.3.5) one obtains the D1Q3 lattice in eq. (**??**).

Table A.3: Abscissae x_i and weights w_i for exact integration of 1D polynomials of up to fifth order.

number of abscissae	polynomial order	abscissae	weights
$q = n$	$N = 2n - 1$	x_i	w_i
1	1	0	1
2	3	± 1	1/2
3	5	0	2/3
		$\pm\sqrt{3}$	1/6

It is now straightforward to construct the corresponding 2D and 3D lattices *via* eq. (A.36). In 2D and 3D we require $q = n^d = 3^2 = 9$ and $q = n^d = 3^3 = 27$ abscissae, respectively. As a result, we obtain the D2Q9 (*cf.* eq. (**??**)) and the D3Q27 (*cf.* eq. (**??**)) lattices [9]. The corresponding multi-dimensional abscissae and weights are shown in tab. A.4 and tab. A.5.

Based on symmetry considerations, one can construct other lattices than D2Q9 and D3Q27. First of all, any integral of the form in eq. (A.29) vanishes if the polynomial $P^{(N)}(x)$ contains only odd orders, *e.g.* $P^{(5)}(x) = 2x^5 - x^3 + x$. This can be

Table A.4: Abscissae x_i and weights w_i for exact integration of 2D polynomials of fifth order.

number of abscissae q	abscissae x_i	weights w_i	
7	$(0,0)$	$1/2$	
	$2\left(\cos\frac{m\pi}{3}, \sin\frac{m\pi}{3}\right)$	$1/12$	$m = 1,\ldots,6$
9	$(0,0)$	$4/9$	
	$\left(0, \pm\sqrt{3}\right), \left(\pm\sqrt{3}, 0\right)$	$1/9$	
	$\left(\pm\sqrt{3}, \pm\sqrt{3}\right)$	$1/36$	

Table A.5: Abscissae x_i and weights w_i for exact integration of 3D polynomials of fifth order.

number of abscissae q	abscissae x_i	weights w_i	
13	$(0,0,0)$	$2/5$	
	$(\pm r, \pm s, 0)$	$1/20$	$r^2 = (5+\sqrt{5})/2$
	$(0, \pm r, \pm s)$	$1/20$	$s^2 = (5-\sqrt{5})/2$
	$(\pm s, 0, \pm r)$	$1/20$	
15	$(0,0,0)$	$2/9$	
	$(\pm\sqrt{3},0,0), (0,\pm\sqrt{3},0), (0,0,\pm\sqrt{3})$	$1/9$	
	$(\pm\sqrt{3},\pm\sqrt{3},\pm\sqrt{3})$	$1/72$	
19	$(0,0,0)$	$1/3$	
	$(\pm\sqrt{3},0,0), (0,\pm\sqrt{3},0), (0,0,\pm\sqrt{3})$	$1/18$	
	$(\pm\sqrt{3},\pm\sqrt{3},0), (\pm\sqrt{3},0,\pm\sqrt{3}), (0,\pm\sqrt{3},\pm\sqrt{3})$	$1/36$	
27	$(0,0,0)$	$8/27$	
	$(\pm\sqrt{3},0,0), (0,\pm\sqrt{3},0), (0,0,\pm\sqrt{3}$	$2/27$	
	$(\pm\sqrt{3},\pm\sqrt{3},0), (\pm\sqrt{3},0,\pm\sqrt{3}), (0,\pm\sqrt{3},\pm\sqrt{3})$	$1/54$	
	$(\pm\sqrt{3},\pm\sqrt{3},\pm\sqrt{3})$	$1/216$	

generalised: if all monomials of a multi-dimensional polynomial contain at least one odd-order term, *e.g.* xy^2z^2 (which is odd in x) or $x^5y^3z^2$ (which is odd in x and y), the integral vanishes. This means that we can directly abandon all of those monomials since they do not contribute to the integral anyway. As a result, we keep only monomials of the form $x^{2a}y^{2b}z^{2c}$ where a, b and c are non-negative integers. Examples are x^2y^2 ($a = 1$, $b = 1$, $c = 0$) or $x^2y^4z^2$ ($a = 1$, $b = 2$, $c = 1$).

Additionally we know that we only have to care about polynomials up to the fifth order. The lowest-order monomial containing x, y and z, however, is $x^2y^2z^2$ and therefore already of sixth order. It should therefore be possible to devise a lattice without the $(\pm1, \pm1, \pm1)$-velocities. This is indeed the case: D3Q15 and D3Q19 in eq. (**??**).

It is possible to obtain different lattices with even fewer abscissae from other methods than symmetry arguments. For example, by using the abscissae of inte-

grals with another weight function, one can construct D2Q7 (which is not on a square but on a hexagonal lattice) and D3Q13. These are the smallest possible sets in 2D and 3D, which can still be used to solve the Navier-Stokes equations. We skip the mathematical details and refer to [7, 8] instead, where thorough derivations are provided.

Concluding, the abscissae in tab. A.4 and tab. A.5 are suitable for LB simulations of the Navier-Stokes equation after an appropriate renormalisation to obtain integer lattice velocity components (*cf.* section 3.3.7).

A.5 BGK characteristics integration

In Section 3.4.1, we have presented a rather general scheme for the integration along characteristics where the collision operator has not yet been specified: eq. (**??**) and eq. (**??**).

With the known BGK collision operator it is possible to partially solve the continuos Boltzmann equation of the form

$$\frac{\partial f_i}{\partial t} + c_{i\alpha}\frac{\partial f_i}{\partial x_\alpha} = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau}. \tag{A.40}$$

To be able to find the solution for f_i one needs to solve the following system in terms of the newly introduced variable ζ:

$$\frac{\mathrm{d}f_i}{\mathrm{d}\zeta} = \frac{\partial f_i}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}\zeta} + \frac{\partial f_i}{\partial x_\alpha}\frac{\mathrm{d}x_\alpha}{\mathrm{d}\zeta} = -\frac{f_i(\zeta) - f_i^{\mathrm{eq}}(\zeta)}{\tau} \tag{A.41}$$

with

$$\frac{\mathrm{d}t}{\mathrm{d}\zeta} = 1, \quad \frac{\mathrm{d}x_\alpha}{\mathrm{d}\zeta} = c_{i\alpha}. \tag{A.42}$$

The dependence of the equilibrium distribution f_i^{eq} on ζ enters *via* $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ where $\rho = \rho(\boldsymbol{x}(\zeta), t(\zeta))$ and $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}(\zeta), t(\zeta))$.

Eq. (A.42) can be easily integrated to obtain the characteristics equation:

$$t = \zeta + t_0, \quad \boldsymbol{x} = \boldsymbol{c}_i\zeta + \boldsymbol{x}_0 \tag{A.43}$$

with $t_0 = t(\zeta = 0)$ and $\boldsymbol{x}_0 = \boldsymbol{x}(\zeta = 0)$.

To integrate eq. (A.41), we consider the ordinary differential equation (ODE)

$$\frac{\mathrm{d}y(\zeta)}{\mathrm{d}\zeta} = g(\zeta)y(\zeta) + h(\zeta) \tag{A.44}$$

for $y(\zeta)$ with given coefficient functions $g(\zeta)$ and $h(\zeta)$. The solution can be obtained following the well-known *variation of constants*:

$$y(\zeta) = \mathrm{e}^{G(\zeta)} \left[C + \int_{\zeta_0}^{\zeta} \mathrm{e}^{-G(\zeta')} h(\zeta') \, \mathrm{d}\zeta' \right] \tag{A.45}$$

with

$$G(\zeta) = \int_{\zeta_0}^{\zeta} g(\zeta') \, \mathrm{d}\zeta' \tag{A.46}$$

and integration constants C and ζ_0.

Exercise A.3. Show that eq. (A.45) solves eq. (A.44).

We can recast eq. (A.41) into the form of eq. (A.44):

$$\frac{\mathrm{d}\, f_i(\zeta)}{\mathrm{d}\zeta} = -\frac{1}{\tau} f_i(\zeta) + \frac{f_i^{\mathrm{eq}}}{\tau}. \tag{A.47}$$

Now we identify $f_i(\zeta)$ with $y(\zeta)$, $-1/\tau$ with $g(\zeta)$ and $f_i^{\mathrm{eq}}(\zeta)/\tau$ with $h(\zeta)$. This leads to $G(\zeta) = -(\zeta - \zeta_0)/\tau$.

Using the integration limits ζ_0 and $\zeta = \zeta_0 + \Delta t$, *i.e.* integrating over one time step, we first obtain

$$f_i(\zeta_0 + \Delta t) = \mathrm{e}^{-\Delta t/\tau} \left[C + \frac{1}{\tau} \int_{\zeta_0}^{\zeta_0 + \Delta t} \mathrm{e}^{\zeta'/\tau} f_i^{\mathrm{eq}}(\zeta') \, \mathrm{d}\zeta' \right]. \tag{A.48}$$

Taking $C = f_i(\zeta_0)$ and replacing the ζ-dependence by the dependence on $\boldsymbol{x}$ and t again gives:

$$\begin{aligned} f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) =& \mathrm{e}^{-\Delta t/\tau} \left[f_i(\boldsymbol{x}, t) + \frac{1}{\tau} \int_{t}^{t + \Delta t} \mathrm{e}^{(t' - t)/\tau} f_i^{\mathrm{eq}}(\boldsymbol{x} + \boldsymbol{c}_i (t' - t), t') \, \mathrm{d}t' \right] \\ f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) =& \mathrm{e}^{-\Delta t/\tau} \left[f_i(\boldsymbol{x}, t) + \frac{1}{\tau} \int_{0}^{\Delta t} \mathrm{e}^{t'/\tau} f_i^{\mathrm{eq}}(\boldsymbol{x} + \boldsymbol{c}_i t', t + t') \, \mathrm{d}t' \right]. \end{aligned} \tag{A.49}$$

This is the integral-form solution of the LBGK equation. Note that we omitted index 0 for x_0 and t_0 as they can be chosen arbitrarily.

Notice that Eq. (A.49) can be discretized using the first order approximation (backward Euler method) or the second order approximation (trapezoidal method). The first order integral approximation yields the lattice Boltzmann equation:

$$\begin{aligned} f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) &= \frac{e^{-\Delta t/\tau}}{\tau} f_i^{eq}(\boldsymbol{x}, t) \Delta t + e^{-\Delta t/\tau} f_i(\boldsymbol{x}, t) \\ f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) &= \frac{\Delta t}{\tau} f_i^{eq}(\boldsymbol{x}, t) + \left(1 - \frac{\Delta t}{\tau}\right) f_i(\boldsymbol{x}, t) \\ f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) &= f_i(\boldsymbol{x}, t) - \frac{f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)}{\tau} \Delta t \end{aligned} \tag{A.50}$$

The second order trapezoidal method yields the implicit formulation of the lattice Boltzmann equation:

$$\begin{aligned}
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& \frac{e^{-\Delta t/\tau}}{\tau}\left(\left(1+\frac{\Delta t}{\tau}\right) f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) + f_i^{\mathrm{eq}}(\boldsymbol{x},t) + O(\Delta t^2)\right)\frac{\Delta t}{2} \\
&+ e^{-\Delta t/\tau} f_i(\boldsymbol{x},t) \\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& \frac{1-\frac{\Delta t}{\tau}}{\tau}\left(\left(1+\frac{\Delta t}{\tau}\right) f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) + f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2} \\
&+ \left(1-\frac{\Delta t}{\tau}+\frac{\Delta t^2}{2\tau^2}\right) f_i(\boldsymbol{x},t) + O(\Delta t^3) \\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{\tau} \\
&+ \left(f_i(\boldsymbol{x},t) - 2f_i^{\mathrm{eq}}(\boldsymbol{x},t) + f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\right)\frac{\Delta t^2}{2\tau^2} + O(\Delta t^3)
\end{aligned} \tag{A.51}$$

We want to represent eq. (A.51) in the form of eq. (3.81) with the BGK collision operator $\Omega_i = -(f_i - f_i^{\mathrm{eq}})/\tau$:

$$\begin{aligned}
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) + \frac{\Omega_i(\boldsymbol{x},t)}{2}\Delta t + \frac{\Omega_i(\boldsymbol{x}+\boldsymbol{c}_i, t+\Delta t)}{2}\Delta t \\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2\tau} \\
&- \left(f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i, t+\Delta t)\right)\frac{\Delta t}{2\tau}
\end{aligned} \tag{A.52}$$

Let us transform eq. (A.51) accordingly:

$$\begin{aligned}
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{\tau} \\
&+ \left(f_i(\boldsymbol{x},t) - 2f_i^{\mathrm{eq}}(\boldsymbol{x},t) + f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\right)\frac{\Delta t^2}{2\tau^2} + O(\Delta t^3) \\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2\tau} \\
&- \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2\tau} + \left(f_i(\boldsymbol{x},t) - 2f_i^{\mathrm{eq}}(\boldsymbol{x},t) + f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\right)\frac{\Delta t^2}{2\tau^2} + O(\Delta t^3) \\
f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2\tau} \\
&- \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\right)\frac{\Delta t}{2\tau} + \left(f_i(\boldsymbol{x},t) - 2f_i^{\mathrm{eq}}(\boldsymbol{x},t) + f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\right)\frac{\Delta t^2}{2\tau^2} + O(\Delta t^3)
\end{aligned} \tag{A.53}$$

By rearranging eq. (A.53) one can further simplify eq. (**??**):

$$\begin{aligned} f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{2\tau} \\ &-\Bigg(\Big(f_i(\boldsymbol{x},t) - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{\tau}\Big) \\ &-\Big(f_i^{\mathrm{eq}}(\boldsymbol{x},t) - \Big(f_i^{\mathrm{eq}}(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\Big)\frac{\Delta t}{\tau}\Big)\Bigg)\frac{\Delta t}{2\tau} + O(\Delta t^3) \end{aligned} \tag{A.54}$$

The last two terms in eq. (A.54) represent $f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)$ and $f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)$. For the evolution of the population $f_i$one can use the first order approximation LBE (*cf.* eq. (A.50)):

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) = f_i(\boldsymbol{x},t) - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{\tau} + O(\Delta t^2) \tag{A.55}$$

The last term represents the evolution of the equilibrium population:

$$f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) = f_i^{\mathrm{eq}}(\boldsymbol{x},t) - \Big(f_i^{\mathrm{eq}}(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\Big)\frac{\Delta t}{2\tau} + O(\Delta t^2). \tag{A.56}$$

The eq. (A.56) requires more elaboration. The equilibrium function depends on density ρ and macroscopic velocity $\boldsymbol{u}$, each of them is summation of populations $\rho = \sum_i f_i$ and populations with velocities $\rho\boldsymbol{u} = \sum_i f_i\boldsymbol{c}_i$. Each of the population is evolving at least of the order of $O(\Delta t^2)$. Thus, one can conclude:

$$\begin{aligned} f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i^{\mathrm{eq}}(\boldsymbol{x},t) + O(\Delta t^2) \\ f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i^{\mathrm{eq}}(\boldsymbol{x},t) + \Big(f_i^{\mathrm{eq}}(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\Big)\frac{\Delta t}{2\tau} + O(\Delta t^2) \end{aligned} \tag{A.57}$$

Substituting $f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)$ and $f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i, t+\Delta t)$ into eq. (A.53) one can obtain the following expression:

$$\begin{aligned} f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) =& f_i(\boldsymbol{x},t) - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{2\tau} \\ &-\Big(f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\Big)\frac{\Delta t}{2\tau} + O(\Delta t^3) \end{aligned} \tag{A.58}$$

By introducing new variable $\bar{f}_i = f_i + \Big(f_i - f_i^{\mathrm{eq}}\Big)\Delta t/(2\tau)$ and rearranging terms in latter equation one can obtain the following expression:

$$\begin{aligned} &f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) + \Big(f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) - f_i^{\mathrm{eq}}(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t)\Big)\frac{\Delta t}{2\tau} \\ =& f_i(\boldsymbol{x},t) - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{2\tau} - \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{\tau} + O(\Delta t^3) \end{aligned} \tag{A.59}$$

The last step is to transform BGK collision operator through newly introduced variable:

$$
\begin{aligned}
\bar{f}_i(\boldsymbol{x},t) =& f_i(\boldsymbol{x},t) + \Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{2\tau} \\
\bar{f}_i(\boldsymbol{x},t) =& f_i(\boldsymbol{x},t)\Big(1+\frac{\Delta t}{2\tau}\Big) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\frac{\Delta t}{2\tau} \\
f_i(\boldsymbol{x},t) =& f_i^{\mathrm{eq}}(\boldsymbol{x},t)\frac{\frac{\Delta t}{2\tau}}{1+\frac{\Delta t}{2\tau}} + \bar{f}_i(\boldsymbol{x},t)\frac{1}{1+\frac{\Delta t}{2\tau}} \\
-\Big(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)\Big)\frac{\Delta t}{\tau} =& -\frac{\bar{f}_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)}{\tau+\frac{\Delta t}{2}}.
\end{aligned}
\tag{A.60}
$$

Combining altogether all terms one can proof that LBE is the second order accurate in time:

$$
\bar{f}_i(\boldsymbol{x}+\boldsymbol{c}\Delta t, t+\Delta t) = \bar{f}_i(\boldsymbol{x},t) - \frac{\bar{f}_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\boldsymbol{x},t)}{\bar{\tau}} + O(\Delta t^3), \tag{A.61}
$$

with $\bar{f}_i = f_i + (f_i - f_i^{\mathrm{eq}})\Delta t/2\tau$ and $\bar{\tau} = \tau + \Delta t/2$.

A.6 MRT D3Q15, D3Q19, D3Q27 models

Only matrices M, equilibrium moments and the collision rates to restore the Navier-Stokes equation are specified in this section for the most spread three-dimensional models. The MRT D3Q15 and D3Q19 models are based on the Gramm-Schmidt procedure [10].[2] The MRT D3Q27 is not wide spread, and only references are given.

MRT D3Q15 model

The moment space is $\boldsymbol{m} = (\rho, e, \epsilon, j_x, q_x, j_y, q_y, j_z, q_z, p_{xx}, p_{ww}, p_{xy}, p_{yz}, p_{zx}, m_{xyz})$. Those are corresponding to density, energy, energy squared, momentum, heat flux, and momentum flux. The lattice velocity vectors are as in chapter 3.3.7. The collision rates are the following:

$$
\boldsymbol{S} = \mathrm{diag}(0, \omega_e, \omega_\epsilon, 0, \omega_q, 0, \omega_q, 0, \omega_q, \omega_\nu, \omega_\nu, \omega_\nu, \omega_\nu, \omega_\nu, \omega_m), \tag{A.62}
$$

where collision rates 0 are specified for four conserved moments, *i.e.* density and momentum. Note that if one wants to simulate the external force $\boldsymbol{F}$ in the Navier-Stokes equation, then collision rates 0 for momentum are not suitable [11].

The vectors for the moments are the following:

[2] We underline that with the superscript G across throughout this section.

$$
\begin{aligned}
&{}^{\mathrm{G}}M_i^{(\rho)} = 1, \quad {}^{\mathrm{G}}M_i^{(e)} = c_{ix}^2 + c_{iy}^2 + c_{iz}^2 - 2 \\
&{}^{\mathrm{G}}M_i^{\epsilon} = \frac{1}{2}(15(c_{ix}^2 + c_{iy}^2 + c_{iz}^2)^2 - 55(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) + 32) \\
&{}^{\mathrm{G}}M_i^{(j_x)} = c_{ix}, \quad {}^{\mathrm{G}}M_i^{(q_x)} = \frac{1}{2}(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 13)c_{ix} \\
&{}^{\mathrm{G}}M_i^{(j_y)} = c_{iy}, \quad {}^{\mathrm{G}}M_i^{(q_y)} = \frac{1}{2}(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 13)c_{iy} \\
&{}^{\mathrm{G}}M_i^{(j_z)} = c_{iz}, \quad {}^{\mathrm{G}}M_i^{(q_z)} = \frac{1}{2}(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 13)c_{iz} \\
&{}^{\mathrm{G}}M_i^{(p_{xx})} = 3c_{ix}^2 - (c_{ix}^2 + c_{iy}^2 + c_{iz}^2), \quad {}^{\mathrm{G}}M_i^{(p_{ww})} = c_{iy}^2 - c_{iz}^2 \\
&{}^{\mathrm{G}}M_i^{(p_{xy})} = c_{ix}c_{iy}, \quad {}^{\mathrm{G}}M_i^{(p_{yz})} = c_{iy}c_{iz}, \quad {}^{\mathrm{G}}M_i^{(p_{xz})} = c_{ix}c_{iz} \\
&{}^{\mathrm{G}}M_i^{(m_{xyz})} = c_{ix}c_{iy}c_{iz},
\end{aligned}
\tag{A.63}
$$

which are combined in the following matrix M:

$$
{}^{\mathrm{G}}\boldsymbol{M} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
-2 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
16 & -4 & -4 & -4 & -4 & -4 & -4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & -4 & 4 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -4 & 4 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
0 & 2 & 2 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1
\end{pmatrix}
\tag{A.64}
$$

with the equilibrium moments as:

$$
\begin{aligned}
&e^{\mathrm{eq}} = -\rho + \rho(u_x^2 + u_y^2 + u_z^2) \\
&\epsilon^{\mathrm{eq}} = \rho - 5\rho(u_x^2 + u_y^2 + u_z^2) \\
&q_x^{\mathrm{eq}} = -\frac{7}{3}\rho u_x, \quad q_y^{\mathrm{eq}} = -\frac{7}{3}\rho u_y, \quad q_z^{\mathrm{eq}} = -\frac{7}{3}\rho u_z \\
&p_{xx}^{\mathrm{eq}} = 2\rho u_x^2 - \rho(u_y^2 + u_z^2), p_{ww}^{\mathrm{eq}} = \rho(u_y^2 - u_z^2) \\
&p_{xy}^{\mathrm{eq}} = \rho u_x u_y, \quad p_{yz}^{\mathrm{eq}} = \rho u_y u_z, \quad p_{xz}^{\mathrm{eq}} = \rho u_x u_z \\
&m_{xyz}^{\mathrm{eq}} = 0
\end{aligned}
\tag{A.65}
$$

Note that the equilibrium moment ϵ^{eq} is not uniquely defined but can be tuned [10].

Exercise A.4. Show that the equilibrium moments can be calculated as $\alpha^{\mathrm{eq}} = M^{\alpha}_{ij} f^{\mathrm{eq}}_j$ with $\alpha = \{\rho, e, \epsilon, j_x, q_x, j_y, q_y, j_z, q_z, p_{xx}, p_{ww}, p_{xy}, p_{yz}, p_{xz}, m_{xyz}\}$ and f^{eq} from eq. (3.56).

The kinematic viscosity ν and the bulk viscosity ζ are connected with the relaxation rates as:

$$\begin{aligned} \nu &= \frac{1}{3}\Big(\frac{1}{\omega_\nu} - \frac{1}{2}\Big) \\ \zeta &= \frac{2}{9}\Big(\frac{1}{\omega_e} - \frac{1}{2}\Big) \end{aligned} \tag{A.66}$$

The last detail which is needed to construct the inverse matrix ${}^{\mathrm{G}}\boldsymbol{M}^{-1} = {}^{\mathrm{G}}\boldsymbol{M}^{T}\,{}^{\mathrm{G}}\boldsymbol{N}^{-1}$ is the matrix ${}^{\mathrm{G}}\boldsymbol{N}^{-1}$ (see eq. (10.33)):

$${}^{\mathrm{G}}\boldsymbol{N}^{-1} = \mathrm{diag}\Big(\frac{1}{15}, \frac{1}{18}, \frac{1}{360}, \frac{1}{10}, \frac{1}{40}, \frac{1}{10}, \frac{1}{40}, \frac{1}{10}, \frac{1}{40}, \frac{1}{12}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\Big). \tag{A.67}$$

MRT D3Q19 model

The 19 discrete velocities for the D3Q19 model are specified in chapter 3.3.7. The moments are the following:

$$\boldsymbol{m} = (\rho, e, \epsilon, j_x, q_x, j_y, q_y, j_z, q_z, p_{xx}, \pi_{xx}, p_{ww}, \pi_{ww}, p_{xy}, p_{yz}, p_{xz}, m_x, m_y, m_z). \tag{A.68}$$

In comparison with D3Q15 model, there are a few moments added. Those are moments corresponding to third order polynomials (m_x, m_y, m_z) and fourth order polynomials (π_{xx}, π_{ww}). The vector basis is the following:

$$\begin{aligned}
&{}^{\mathrm{G}}M^{(\rho)}_i = 1, \quad {}^{\mathrm{G}}M^{e}_i = 19(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 30 \\
&{}^{\mathrm{G}}M^{(\epsilon)}_i = \Big(21(c_{ix}^2 + c_{iy}^2 + c_{iz}^2)^2 - 53(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) + 24\Big)/2 \\
&{}^{\mathrm{G}}M^{(j_x)}_i = c_{ix}, \quad {}^{\mathrm{G}}M^{(q_x)}_i = \Big(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 9\Big)c_{ix} \\
&{}^{\mathrm{G}}M^{(j_y)}_i = c_{iy}, \quad {}^{\mathrm{G}}M^{(q_y)}_i = \Big(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 9\Big)c_{iy} \\
&{}^{\mathrm{G}}M^{(j_z)}_i = c_{iz}, \quad {}^{\mathrm{G}}M^{(q_z)}_i = \Big(5(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 9\Big)c_{iz} \\
&{}^{\mathrm{G}}M^{(p_{xx})}_i = 3c_{ix}^2 - (c_{ix}^2 + c_{iy}^2 + c_{iz}^2), \quad {}^{\mathrm{G}}M^{(\pi_{xx})}_i = \Big(3(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 5\Big)\Big(3c_{ix}^2 - (c_{ix}^2 + c_{iy}^2 + c_{iz}^2)\Big) \\
&{}^{\mathrm{G}}M^{(p_{ww})}_i = c_{iy}^2 - c_{iz}^2, \quad {}^{\mathrm{G}}M^{(\pi_{ww})}_i = \Big(3(c_{ix}^2 + c_{iy}^2 + c_{iz}^2) - 5\Big)\Big(c_{iy}^2 - c_{iz}^2\Big) \\
&{}^{\mathrm{G}}M^{(p_{xy})}_i = c_{ix}c_{iy}, \quad {}^{\mathrm{G}}M^{(p_{yz})}_i = c_{iy}c_{iz}, \quad {}^{\mathrm{G}}M^{(p_{xz})}_i = c_{ix}c_{iz} \\
&{}^{\mathrm{G}}M^{(m_x)}_i = (c_{iy}^2 - c_{iz}^2)c_{ix}, \quad {}^{\mathrm{G}}M^{(m_y)}_i = (c_{iz}^2 - c_{ix}^2)c_{iy}, \quad {}^{\mathrm{G}}M^{(m_z)}_i = (c_{ix}^2 - c_{iy}^2)c_{iz}.
\end{aligned} \tag{A.69}$$

The collision matrix is:

$$\boldsymbol{S} = \text{diag}(0, \omega_e, \omega_\epsilon, 0, \omega_q, 0, \omega_q, 0, \omega_q, \omega_\nu, \omega_\pi, \omega_\nu, \omega_\pi, \omega_\nu, \omega_\nu, \omega_\nu, \omega_m, \omega_m, \omega_m). \tag{A.70}$$

The equilibrium moments are represented as:

$$\begin{aligned}
&e^{\text{eq}} = -11\rho + 19\rho(u_x^2 + u_y^2 + u_z^2), \quad \epsilon^{\text{eq}} = 3\rho - \frac{11}{2}\rho(u_x^2 + u_y^2 + u_z^2) \\
&q_x^{\text{eq}} = -\frac{2}{3}\rho u_x, \quad q_y^{\text{eq}} = -\frac{2}{3}\rho u_y, \quad q_z^{\text{eq}} = -\frac{2}{3}\rho u_z \\
&p_{xx}^{\text{eq}} = 2\rho u_x^2 - \rho(u_y^2 + u_z^2), \quad \pi_{xx}^{\text{eq}} = -\frac{1}{2}\Big(2\rho u_x^2 - \rho(u_y^2 + u_z^2)\Big) \\
&p_{ww}^{\text{eq}} = \rho(u_y^2 - u_z^2), \quad \pi_{ww}^{\text{eq}} = -\frac{1}{2}\rho(u_y^2 - u_z^2) \\
&p_{xy}^{\text{eq}} = \rho u_x u_y, \quad p_{yz}^{\text{eq}} = \rho u_y u_z, \quad p_{xz}^{\text{eq}} = \rho u_x u_z \\
&m_x^{\text{eq}} = 0, \quad m_y^{\text{eq}} = 0, \quad m_z^{\text{eq}} = 0
\end{aligned} \tag{A.71}$$

Finally, the matrix ${}^{\text{G}}\boldsymbol{M}$ is:

$${}^{\text{G}}\boldsymbol{M} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
-30 & -11 & -11 & -11 & -11 & -11 & -11 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
12 & -4 & -4 & -4 & -4 & -4 & -4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & -4 & 4 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 2 & 2 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 \\
0 & -4 & -4 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 \\
0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -2 & -2 & 2 & 2 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1
\end{pmatrix} \tag{A.72}$$

The last detail is to specify how to invert the moments matrix $\boldsymbol{M}$:

$${}^{\text{G}}\boldsymbol{M}^{-1} = {}^{\text{G}}\boldsymbol{M}^{T}\,{}^{\text{G}}\boldsymbol{N}^{-1}, \tag{A.73}$$

with ${}^{\text{G}}\boldsymbol{N}^{-1} = \text{diag}(1/19, 1/2394, 1/252, 1/10, 1/40, 1/10, 1/40, 1/10, 1/40, 1/36, 1/72, 1/12, 1/24, 1/4, 1/4, 1/4, 1/8, 1/8, 1/8)$.

Kinematic viscosity is defined as:

$$\nu = \frac{1}{3}\left(\frac{1}{\omega_\nu} - \frac{1}{2}\right) \tag{A.74}$$

The bulk viscosity is defined as:

$$\zeta = \frac{2}{9}\left(\frac{1}{\omega_e} - \frac{1}{2}\right) \tag{A.75}$$

MRT D3Q27

The D3Q27 model has the largest memory footprint and the least computational performance among common three-dimensional models. The MRT formulation of the collision operator only adds additional computational overhead. Thus, though D3Q27 model has the best rotational symmetry, its MRT counterpart is not well spread. Moreover, it is difficult to find the MRT D3Q27 model in the literature. Despite the absence of the ready-to-use examples, one can construct the MRT D3Q27 model based on the polynomials in velocity sets [12] and using Gramm-Schmidt approach as discussed in chapter 10. Another alternative is to use a variation of the MRT D3Q27 model, so-called cascaded lattice Boltzmann D3Q27 model [13, 14].

A.7 Non-equilibrium bounce-back in D3Q19 model

In complement to section **??**, we summarize the NEBB implementation in the D3Q19 model. These results follow the work of Hecht and Harting [15], which extends for the D3Q19 model the original procedure of Zou and He [16], developed for the D3Q15 model. The D3Q27 model has been treated in [].

For concreteness let us consider the cubic geometry depicted in fig. **??**. Below we explicitly describe the implementation of the NEBB for its 6 plane surfaces, 12 edges and 8 corners. While the formulas are given for the LB incompressible model [17], the standard model is easily recovered by setting $\rho_0 = \rho$.

Fig. A.1: Cubic cell using the D3Q19 lattice model. NOTE (AK): Can I borrow here your Fig. 3.2 from Chap. 3?

Plane surfaces

Bottom plane ($y = 0$)

Velocity boundary condition :

$\rho = (f_0 + f_1 + f_2 + f_5 + f_6 + f_9 + f_{10} + f_{15} + f_{16}) + 2(f_4 + f_8 + f_{12} + f_{13} + f_{18}) + \rho_0 u_y$

Pressure boundary condition :

$u_y = \frac{1}{\rho_0}[\rho - (f_0 + f_1 + f_2 + f_5 + f_6 + f_9 + f_{10} + f_{15} + f_{16}) - 2(f_4 + f_8 + f_{12} + f_{13} + f_{18})]$

Populations :

$$f_3 = f_4 + \frac{1}{3}\rho_0 u_y$$

$$f_7 = f_8 + \frac{1}{6}\rho_0(u_x + u_y) - N^z_{yz}$$

$$f_{11} = f_{12} + \frac{1}{6}\rho_0(u_y + u_z) - N^y_{yz}$$

$$f_{14} = f_{13} + \frac{1}{6}\rho_0(-u_x + u_y) + N^z_{yz}$$

$$f_{17} = f_{18} + \frac{1}{6}\rho_0(u_y - u_z) + N^y_{yz}$$

Top plane ($y = LY$)

Velocity boundary condition :

$\rho = (f_0 + f_1 + f_2 + f_5 + f_6 + f_9 + f_{10} + f_{15} + f_{16}) + 2(f_3 + f_7 + f_{11} + f_{14} + f_{17}) - \rho_0 u_y$

Pressure boundary condition :

$u_y = \frac{1}{\rho_0}[-\rho + (f_0 + f_1 + f_2 + f_5 + f_6 + f_9 + f_{10} + f_{15} + f_{16}) + 2(f_3 + f_7 + f_{11} + f_{14} + f_{17})]$

Populations :

$$f_4 = f_3 - \frac{1}{3}\rho_0 u_y$$

$$f_8 = f_7 - \frac{1}{6}\rho_0(u_x + u_y) + yz^z$$

$$f_{12} = f_{11} - \frac{1}{6}\rho_0(u_y + u_z) + N^y_{yz}$$

$$f_{13} = f_{14} - \frac{1}{6}\rho_0(-u_x + u_y) - N^z_{yz}$$

$$f_{18} = f_{17} - \frac{1}{6}\rho_0(u_y - u_z) - N^y_{yz}$$

Left plane ($x = 0$)

Velocity boundary condition :

$\rho = (f_0 + f_3 + f_4 + f_5 + f_6 + f_{11} + f_{12} + f_{17} + f_{18}) + 2(f_2 + f_8 + f_{10} + f_{14} + f_{16}) + \rho_0 u_x$

Pressure boundary condition :

$u_x = \frac{1}{\rho_0}[\rho - (f_0 + f_3 + f_4 + f_5 + f_6 + f_{11} + f_{12} + f_{17} + f_{18}) - 2(f_2 + f_8 + f_{10} + f_{14} + f_{16})]$

Populations :

$$f_1 = f_2 + \frac{1}{3}\rho_0 u_x$$

$$f_7 = f_8 + \frac{1}{6}\rho_0(u_x + u_y) - N_{zx}$$

$$f_9 = f_{10} + \frac{1}{6}\rho_0(u_x + u_z) - N_{yx}$$

$$f_{13} = f_{14} + \frac{1}{6}\rho_0(u_x - u_y) + N_{zx}$$

$$f_{15} = f_{16} + \frac{1}{6}\rho_0(u_x - u_z) + N_{yx}$$

Right plane ($x = LX$)

Velocity boundary condition :

$\rho = (f_0 + f_3 + f_4 + f_5 + f_6 + f_{11} + f_{12} + f_{17} + f_{18}) + 2(f_1 + f_7 + f_9 + f_{13} + f_{15}) - \rho_0 u_x$

Pressure boundary condition :

$u_x = \frac{1}{\rho_0}[-\rho + (f_0 + f_3 + f_4 + f_5 + f_6 + f_{11} + f_{12} + f_{17} + f_{18}) + 2(f_1 + f_7 + f_9 + f_{13} + f_{15})]$

Populations :

$$f_2 = f_1 - \frac{1}{3}\rho_0 u_x$$

$$f_8 = f_7 - \frac{1}{6}\rho_0(u_x + u_y) + N_{zx}$$

$$f_{10} = f_9 - \frac{1}{6}\rho_0(u_x + u_z) + N_{yx}$$

$$f_{14} = f_{13} - \frac{1}{6}\rho_0(u_x - u_y) - N_{zx}$$

$$f_{16} = f_{15} - \frac{1}{6}\rho_0(u_x - u_z) - N_{yx}$$

Back plane ($z = 0$)

Velocity boundary condition :

$\rho = (f_0 + f_1 + f_2 + f_3 + f_4 + f_7 + f_8 + f_{13} + f_{14}) + 2(f_6 + f_{10} + f_{12} + f_{15} + f_{17}) + \rho_0 u_z$

Pressure boundary condition :

$u_z = \frac{1}{\rho_0}[\rho - (f_0 + f_1 + f_2 + f_3 + f_4 + f_7 + f_8 + f_{13} - f_{14}) - 2(f_6 + f_{10} + f_{12} + f_{15} + f_{17})]$

Populations :

$$f_5 = f_6 + \frac{1}{3}\rho_0 u_z$$

$$f_9 = f_{10} + \frac{1}{6}\rho_0(u_x + u_z) - N_{xy}^x$$

$$f_{12} = f_{11} + \frac{1}{6}\rho_0(u_y + u_z) - N_{xy}^y$$

$$f_{16} = f_{15} + \frac{1}{6}\rho_0(-u_x + u_z) + N_{xy}^x$$

$$f_{18} = f_{17} + \frac{1}{6}\rho_0(-u_y + u_y) + N_{xy}^y$$

Front plane $(z = LZ)$

Velocity boundary condition :

$\rho = (f_0 + f_1 + f_2 + f_3 + f_4 + f_7 + f_8 + f_{13} + f_{14}) + 2(f_5 + f_9 + f_{11} + f_{16} + f_{18}) - \rho_0 u_z$

Pressure boundary condition :

$u_z = \frac{1}{\rho_0}[-\rho + (f_0 + f_1 + f_2 + f_3 + f_4 + f_7 + f_8 + f_{13} - f_{14}) + 2(f_5 + f_9 + f_{11} + f_{16} + f_{18})]$

Populations :

$$f_6 = f_5 - \frac{1}{3}\rho_0 u_z$$

$$f_{10} = f_9 - \frac{1}{6}\rho_0(u_x + u_z) + N_{xy}^x$$

$$f_{11} = f_{12} - \frac{1}{6}\rho_0(u_y + u_z) + N_{xy}^y$$

$$f_{15} = f_{16} - \frac{1}{6}\rho_0(-u_x + u_z) - N_{xy}^x$$

$$f_{17} = f_{18} - \frac{1}{6}\rho_0(-u_y + u_y) - N_{xy}^y$$

Transversal momentum corrections

$$N_{xy}^{x} = \frac{1}{2}[(f_1 + f_7 + f_{13}) - (f_2 + f_8 + f_{14})] - \frac{1}{3}\rho_0 u_x$$
$$N_{xy}^{y} = \frac{1}{2}[(f_3 + f_7 + f_{14}) - (f_4 + f_8 + f_{13})] - \frac{1}{3}\rho_0 u_y$$
$$N_{xz}^{x} = \frac{1}{2}[(f_1 + f_9 + f_{15}) - (f_2 + f_{10} + f_{16})] - \frac{1}{3}\rho_0 u_x$$
$$N_{xz}^{z} = \frac{1}{2}[(f_5 + f_9 + f_{16}) - (f_6 + f_{10} + f_{15})] - \frac{1}{3}\rho_0 u_z$$
$$N_{yz}^{z} = \frac{1}{2}[(f_5 + f_{11} + f_{18}) - (f_6 + f_{12} + f_{17})] - \frac{1}{3}\rho_0 u_z$$
$$N_{yz}^{y} = \frac{1}{2}[(f_3 + f_{11} + f_{17}) - (f_4 + f_{12} + f_{18})] - \frac{1}{3}\rho_0 u_y$$

Subscripts indicate tangential plane and superscripts the corresponding momentum component, *e.g.* N_{xz}^{x} refers to plane xz applying along momentum component x.

Edges

Bottom-left edge ($x = 0$ and $y = 0$)

Normal populations

$f_1 = f_2 \quad f_3 = f_4$

Diagonal populations

$f_7 = f_8 \quad f_9 = f_{10} - N_z \quad f_{11} = f_{12} - N_z f_{15} = f_{16} + N_z \quad f_{17} = f_{18} + N_z$

Transversal momentum correction $\quad N_z = \frac{1}{4}(f_5 - f_6)$

Buried links

$f_0 = f_{13} = f_{14} = 0$

Velocity boundary condition :

$$f_{13} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{14} = f_{13} \quad f_0 = 12 f_{13}$$

Pressure boundary condition :

$$f_{13} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{14} = f_{13} \quad f_0 = 12 f_{13}$$

Top-left edge ($x = 0$ and $y = LY$)

Normal populations

$f_1 = f_2 \quad f_4 = f_3$

Diagonal populations

$f_{13} = f_{14} \quad f_9 = f_{10} + N_z \quad f_{12} = f_{11} - N_z \quad f_{15} = f_{16} + N_z \quad f_{18} = f_{17} - N_z$

Transversal momentum correction $\quad N_z = \frac{1}{4}(f_5 - f_6)$

Buried links

$f_0 = f_7 = f_8 = 0$

Velocity boundary condition :

$$f_7 = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_8 = f_7 \quad f_0 = 12 f_7$$

Pressure boundary condition :

$$f_7 = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_8 = f_7 \quad f_0 = 12 f_7$$

Bottom-right edge ($x = LX$ and $y = 0$)

Normal populations

$f_2 = f_1 \quad f_3 = f_4$

Diagonal populations

$f_{14} = f_{13} \quad f_{10} = f_9 + N_z \quad f_{11} = f_{12} - N_z \quad f_{16} = f_{15} - N_z \quad f_{17} = f_{18} + N_z$

Transversal momentum correction $\quad N_z = \frac{1}{4}(f_5 - f_6)$

Buried links

$f_0 = f_7 = f_8 = 0$

Velocity boundary condition :

$$f_7 = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_8 = f_7 \quad f_0 = 12 f_7$$

Pressure boundary condition :

$$f_7 = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_8 = f_7 \quad f_0 = 12 f_7$$

Top-right edge ($x = LX$ and $y = LY$)

Normal populations

$f_2 = f_1 \quad f_4 = f_3$

Diagonal populations

$f_8 = f_7 \quad f_{10} = f_9 + N_z \quad f_{12} = f_{11} + N_z \quad f_{16} = f_{15} - N_z \quad f_{18} = f_{17} - N_z$

Transversal momentum correction $\quad N_z = \frac{1}{4}(f_5 - f_6)$

Buried links

$f_0 = f_{13} = f_{14} = 0$

Velocity boundary condition :

$$f_{13} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{14} = f_{13} \quad f_0 = 12 f_{13}$$

Pressure boundary condition :

$$f_{13} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{14} = f_{13} \quad f_0 = 12 f_{13}$$

Bottom-front edge ($y = 0$ and $z = LZ$)

Normal populations

$f_6 = f_5 \quad f_3 = f_4$

Diagonal populations

$f_{17} = f_{18} \quad f_{10} = f_9 + N_x \quad f_7 = f_8 - N_x \quad f_{14} = f_{13} + N_x \quad f_{15} = f_{16} - N_x$

Transversal momentum correction $\quad N_x = \frac{1}{4}(f_1 - f_2)$

Buried links

$f_0 = f_{11} = f_{12} = 0$

Velocity boundary condition :

$$f_{11} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{12} = f_{11} \quad f_0 = 12 f_{11}$$

Pressure boundary condition :

$$f_{11} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{12} = f_{11} \quad f_0 = 12 f_{11}$$

Left-front edge ($x = 0$ and $z = LZ$)

Normal populations

$f_1 = f_2 \quad f_6 = f_5$

Diagonal populations

$f_{15} = f_{16} \quad f_7 = f_8 - N_y \quad f_{12} = f_{11} + N_y \quad f_{13} = f_{14} + N_y \quad f_{17} = f_{18} - N_y$

Transversal momentum correction $\quad N_y = \frac{1}{4}(f_3 - f_4)$

Buried links

$f_0 = f_9 = f_{10} = 0$

Velocity boundary condition :

$$f_9 = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{10} = f_9 \quad f_0 = 12 f_9$$

Pressure boundary condition :

$$f_9 = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{10} = f_9 \quad f_0 = 12 f_9$$

Top-front edge ($y = LY$ and $z = LZ$)

Normal populations

$f_6 = f_5 \quad f_4 = f_3$

Diagonal populations

$f_{12} = f_{11} \quad f_8 = f_7 + N_x \quad f_{13} = f_{14} - N_x \quad f_{10} = f_9 + N_x \quad f_{15} = f_{16} - N_x$

Transversal momentum correction $\quad N_x = \frac{1}{4}(f_1 - f_2)$

Buried links

$f_0 = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_{17} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{18} = f_{17} \quad f_0 = 12 f_{17}$$

Pressure boundary condition :

$$f_{17} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{18} = f_{17} \quad f_0 = 12 f_{17}$$

Right-front edge ($x = LX$ and $z = LZ$)

Normal populations

$f_2 = f_1 \quad f_6 = f_5$

Diagonal populations

$f_{10} = f_9 \quad f_8 = f_7 + N_y \quad f_{12} = f_{11} + N_y \quad f_{14} = f_{13} - N_y \quad f_{17} = f_{18} - N_y$

Transversal momentum correction $\quad N_y = \frac{1}{4}(f_3 - f_4)$

Buried links

$f_0 = f_{10} = f_{15} = 0$

Velocity boundary condition :

$$f_{10} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{15} = f_{10} \quad f_0 = 12 f_{10}$$

Pressure boundary condition :

$$f_{10} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{15} = f_{10} \quad f_0 = 12 f_{10}$$

Bottom-back edge ($y = 0$ and $z = 0$)

Normal populations

$f_5 = f_6 \quad f_3 = f_4$

Diagonal populations

$f_{11} = f_{12} \quad f_7 = f_8 - N_x \quad f_9 = f_{10} - N_x \quad f_{14} = f_{13} + N_x \quad f_{16} = f_{15} + N_x$

Transversal momentum correction $\quad N_x = \frac{1}{4}(f_1 - f_2)$

Buried links

$f_0 = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_{17} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{18} = f_{17} \quad f_0 = 12 f_{17}$$

Pressure boundary condition :

$$f_{17} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{18} = f_{17} \quad f_0 = 12 f_{17}$$

Left-back edge ($x = 0$ and $z = 0$)

Normal populations

$f_1 = f_2 \quad f_5 = f_6$

Diagonal populations

$f_9 = f_{10} \quad f_7 = f_8 - N_y \quad f_{11} = f_{12} - N_y \quad f_{13} = f_{14} + N_y \quad f_{18} = f_{17} + N_y$

Transversal momentum correction $\quad N_y = \frac{1}{4}(f_3 - f_4)$

Buried links

$f_0 = f_{15} = f_{16} = 0$

Velocity boundary condition :

$$f_{15} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{16} = f_{15} \quad f_0 = 12 f_{15}$$

Pressure boundary condition :

$$f_{15} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{16} = f_{15} \quad f_0 = 12 f_{15}$$

Top-back edge ($y = LY$ and $z = 0$)

Normal populations

$f_5 = f_6 \quad f_4 = f_3$

Diagonal populations

$f_{18} = f_{17} \quad f_7 = f_8 + N_x \quad f_9 = f_{10} - N_x \quad f_{13} = f_{14} - N_x \quad f_{16} = f_{15} + N_x$

Transversal momentum correction $\quad N_x = \frac{1}{4}(f_1 - f_2)$

Buried links

$f_0 = f_{11} = f_{12} = 0$

Velocity boundary condition :

$$f_{11} = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{12} = f_{11} \quad f_0 = 12 f_{11}$$

Pressure boundary condition :

$$f_{11} = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{12} = f_{11} \quad f_0 = 12 f_{11}$$

Right-back edge ($x = LX$ and $z = 0$)

Normal populations

$f_2 = f_1 \quad f_5 = f_6$

Diagonal populations

$f_{16} = f_{15} \quad f_8 = f_7 + N_y \quad f_{11} = f_{12} - N_y \quad f_{14} = f_{13} - N_y \quad f_{18} = f_{17} + N_y$

Transversal momentum correction $\quad N_y = \frac{1}{4}(f_3 - f_4)$

Buried links

$f_0 = f_9 = f_{10} = 0$

Velocity boundary condition :

$$f_9 = \frac{1}{22}\sum_{i=0}^{18} f_i \quad f_{10} = f_9 \quad f_0 = 12 f_9$$

Pressure boundary condition :

$$f_9 = \frac{1}{14}\left(\rho - \sum_{i=0}^{18} f_i\right) \quad f_{10} = f_9 \quad f_0 = 12 f_9$$

Corners

Left-bottom-front edge ($x = 0$, $y = 0$ and $z = LZ$)

Normal populations

$f_1 = f_2 \quad f_3 = f_4 \quad f_6 = f_5$

Diagonal populations

$f_7 = f_8 \quad f_{15} = f_{16} \quad f_{17} = f_{18}$

Buried links

$f_0 = f_9 = f_{10} = f_{11} = f_{12} = f_{13} = f_{14} = 0$

Velocity boundary condition :

$$f_9 = \frac{1}{18}\sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_9 = \frac{1}{18}\left(\rho - \sum_{i=0}^{18} f_i\right)$$

$f_{10} = f_9 \quad f_{11} = f_9 \quad f_{12} = f_9 \quad f_{13} = f_9 \quad f_{14} = f_9 \quad f_0 = 12 f_9$

Left-top-front edge ($x = 0$, $y = LY$ and $z = LZ$)

Normal populations

$f_1 = f_2 \quad f_4 = f_3 \quad f_6 = f_5$

Diagonal populations

$f_{12} = f_{11} \quad f_{13} = f_{14} \quad f_{15} = f_{16}$

Buried links

$f_0 = f_8 = f_9 = f_{10} = f_{12} = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_8 = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_8 = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_9 = f_8 \quad f_{10} = f_8 \quad f_{12} = f_8 \quad f_{17} = f_8 \quad f_{18} = f_8 \quad f_0 = 12 f_8$

Left-top-back edge ($x = 0$, $y = LY$ and $z = 0$)

Normal populations

$f_1 = f_2 \quad f_4 = f_3 \quad f_5 = f_6$

Diagonal populations

$f_9 = f_{10} \quad f_{13} = f_{14} \quad f_{18} = f_{17}$

Buried links

$f_0 = f_7 = f_8 = f_{11} = f_{12} = f_{15} = f_{16} = 0$

Velocity boundary condition :

$$f_7 = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_7 = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_8 = f_7 \quad f_{11} = f_7 \quad f_{12} = f_7 \quad f_{15} = f_7 \quad f_{16} = f_7 \quad f_0 = 12 f_7$

Left-bottom-back edge ($x = 0$, $y = 0$ and $z = 0$)

Normal populations

$f_1 = f_2 \quad f_3 = f_4 \quad f_5 = f_6$

Diagonal populations

$f_7 = f_8 \quad f_9 = f_{10} \quad f_{11} = f_{12}$

Buried links

$f_0 = f_{13} = f_{14} = f_{15} = f_{16} = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_{13} = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_{13} = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_{14} = f_{13} \quad f_{15} = f_{13} \quad f_{16} = f_{13} \quad f_{17} = f_{13} \quad f_{18} = f_{13} \quad f_0 = 12 f_{13}$

Right-bottom-front edge ($x = LX$, $y = 0$ and $z = LZ$)

Normal populations

$f_2 = f_1 \quad f_3 = f_4 \quad f_6 = f_5$

Diagonal populations

$f_{10} = f_9 \quad f_{14} = f_{13} \quad f_{17} = f_{18}$

Buried links

$f_0 = f_7 = f_8 = f_{11} = f_{12} = f_{15} = f_{16} = 0$

Velocity boundary condition :

$$f_7 = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_7 = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_8 = f_7 \quad f_{11} = f_7 \quad f_{12} = f_7 \quad f_{15} = f_7 \quad f_{16} = f_7 \quad f_0 = 12 f_7$

Right-top-front edge ($x = LX$, $y = LY$ and $z = LZ$)

Normal populations

$f_2 = f_1 \quad f_4 = f_3 \quad f_6 = f_5$

Diagonal populations

$f_8 = f_7 \quad f_9 = f_{10} \quad f_{12} = f_{11}$

Buried links

$f_0 = f_{13} = f_{14} = f_{15} = f_{16} = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_{13} = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_{13} = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_{14} = f_{13} \quad f_{15} = f_{13} \quad f_{16} = f_{13} \quad f_{17} = f_{13} \quad f_{18} = f_{13} \quad f_0 = 12 f_{13}$

Right-top-back edge ($x = LX$, $y = LY$ and $z = 0$)

Normal populations

$f_2 = f_1 \quad f_4 = f_3 \quad f_5 = f_6$

Diagonal populations

$f_8 = f_7 \quad f_{16} = f_{15} \quad f_{18} = f_{17}$

Buried links

$f_0 = f_9 = f_{10} = f_{11} = f_{12} = f_{13} = f_{14} = 0$

Velocity boundary condition :

$$f_9 = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_9 = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_{10} = f_9 \quad f_{11} = f_9 \quad f_{12} = f_9 \quad f_{13} = f_9 \quad f_{14} = f_9 \quad f_0 = 12 f_9$

Right-bottom-back edge ($x = LX$, $y = 0$ and $z = 0$)

Normal populations

$f_2 = f_1 \quad f_3 = f_4 \quad f_5 = f_6$

Diagonal populations

$f_{11} = f_{12} \quad f_{14} = f_{13} \quad f_{16} = f_{15}$

Buried links

$f_0 = f_7 = f_8 = f_9 = f_{10} = f_{17} = f_{18} = 0$

Velocity boundary condition :

$$f_7 = \frac{1}{18} \sum_{i=0}^{18} f_i$$

Pressure boundary condition :

$$f_7 = \frac{1}{18} \left(\rho - \sum_{i=0}^{18} f_i \right)$$

$f_8 = f_7 \quad f_9 = f_7 \quad f_{10} = f_7 \quad f_{17} = f_7 \quad f_{18} = f_7 \quad f_0 = 12 f_7$

References

1. P.A. Thompson, *Compressible-Fluid Dynamics* (McGraw-Hill, 1972)
2. L.E. Kinsler, A.R. Frey, A.B. Coppens, J.V. Sanders, *Fundamentals of acoustics*, 4th edn. (John Wiley & Sons, 2000)
3. P. Dellar, Phys. Rev. E **64**(3) (2001)
4. P.J. Dellar, J. of Comput. Phys. **259**, 270 (2014)
5. T. Shao, T. Chen, R. Frank, Math. Comp. **18**, 598 (1964)
6. M. Abramowitz, I. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables* (U.S. Government Printing Office, 1964)
7. X. Shan, X.F. Yuan, H. Chen, J. Fluid Mech. **550**, 413 (2006)
8. W.P. Yudistiawan, S.K. Kwak, D.V. Patil, S. Ansumali, Phys. Rev. E **82**(4), 046701 (2010)
9. X. He, L.S. Luo, Phys. Rev. E **56**(6), 6811 (1997)
10. D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, L.S. Luo, Phil. Trans. R. Soc. Lond. A **360**, 437 (2002)
11. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
12. R. Rubinstein, L.S. Luo, Phys. Rev. E **77**(036709), 1 (2008)
13. M. Geier, A. Greiner, J. Korvink, Phys. Rev. E **73**(066705), 1 (2006)
14. K. Premnath, S. Banerjee, J. Stat. Phys. **143**, 747 (2011)
15. M. Hecht, J. Harting, J. Stat. Mech.: Theory Exp. **P**, 01018 (2010)
16. Q. Zou, X. He, Phys. Fluids **9**, 1591 (1997)
17. X. He, L.S. Luo, J. Stat. Phys. **88**(3-4), 927 (1997)

Index