Chapter 3
The Lattice Boltzmann Equation

Abstract After reading this chapter, you will know the basics of the lattice Boltzmann method, how it can be used to simulate fluids, and how to implement it in code. You will have insight into the derivation of the lattice Boltzmann equation, having seen how the continuous Boltzmann equation is discretised in velocity space through Hermite series expansion, before being discretised in physical space and time through the method of characteristics. In particular, you will be familiar with the various simple sets of velocity vectors that are available, and how the discrete BGK collision model is applied.

In this chapter we provide a simple overview of the LBM. After presenting a general introduction in Sect. 3.1, we briefly outline the LBM without derivation in order to give an initial understanding (cf. Sect. 3.2). Section 3.3 contains 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 by discretising the Boltzmann equation in two steps. First, in Sect. 3.4, we discretise velocity space by limiting the continuous particle velocity $\boldsymbol{\xi}$ to a discrete set of velocities $\{\boldsymbol{\xi}_i\}$. Secondly, we discretise physical space and time by *integrating along characteristics* in Sect. 3.5. The result of these two steps is the *lattice Boltzmann equation* (LBE). A thorough discussion of how the LBE is linked to fluid dynamics will be given in Chap. 4.

3.1 Introduction

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. Situations with more complex geometries and boundary conditions must typically be solved using numerical methods. However, as we have seen in Chap. 2, the numerical methods used to solve the equations of fluid mechanics can be difficult both to implement and to parallelise.

The Boltzmann equation in Sect. 1.3 describes the dynamics of a gas on a mesoscopic scale. From Sect. 1.3.5 we also know that the Boltzmann equation leads

T. Krüger et al., *The Lattice Boltzmann Method*, Graduate Texts in Physics,
DOI 10.1007/978-3-319-44649-3_3

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 NSE for the same case.[1]

The problem with this idea is that the Boltzmann equation is even more difficult to solve analytically than the NSE. 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 NSE.

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 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 iterative numerical schemes with approximation errors are introduced to deal with this. Contrarily, the discretised Boltzmann equation takes a very different approach that results in exact advection.

In this chapter we will first discretise velocity space (cf. Sect. 3.4) and then space-time (cf. Sect. 3.5). Historically, the LBE was not found along these lines. It rather evolved out of the lattice gas automata described in Sect. 2.2.2 and many early articles on the 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 LB scheme is covered in Chap. 6.

3.2 The Lattice Boltzmann Equation in a Nutshell

Before diving into the derivation of the LBE, we will first give a short summary. This serves two purposes: first, it is unnecessary to know all the details of the derivation

[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 Sect. 4.1.

in order to implement simple codes. Therefore, this section, along with Sect. 3.3, gives a quick introduction for readers who only wish to know the most relevant results. Secondly, readers who *are* interested in understanding the derivations will benefit from knowing the end results before going through the analysis.

3.2.1 Overview

The basic quantity of the LBM is the *discrete-velocity distribution function* $f_i(\boldsymbol{x}, t)$, often called the particle *populations*. Similar to the distribution function introduced in Sect. 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:

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

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 time step Δt and lattice spacing Δx respectively represent a time resolution and a space resolution in any set of units. One possible choice is SI units, where Δt is given in seconds and Δx in metres, and another possible choice would be Imperial units. The most common choice in the LB literature, however, is *lattice units*, a simple artificial set of units scaled such that $\Delta t = 1$ and $\Delta x = 1$. We can convert quantities between lattice units and physical units as easily as converting between two sets of physical units, such as SI and Imperial units. Alternatively, we can ensure that we simulate the same behaviour in two different systems of units by exploiting the *law of similarity* explained in Sect. 1.2. That way, we need only ensure that the relevant *dimensionless numbers*, such as the Reynolds number, are the same in both systems. We cover the topics of units and the law of similarity in more depth in Chap. 7.

The discrete velocities $\boldsymbol{c}_i$ require further explanation. Together with a corresponding set of weighting coefficients w_i which we will explain 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 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 Table 3.1. The velocity components $\{c_{i\alpha}\}$ and weights $\{w_i\}$ are also collected in Tables 3.2, 3.3, 3.4, 3.5 and 3.6.

Typically, we want to use as few velocities as possible to minimise memory and computing requirements. However, there is a tradeoff between smaller velocity sets[2] (e.g. D3Q15) and higher accuracy (e.g. D3Q27). In 3D, the most commonly used velocity set is D3Q19.

We can also find a constant c_s in each velocity set as in (3.60). In the basic isothermal LBE, c_s determines the relation $p = c_\mathrm{s}^2 \rho$ 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 $c_\mathrm{s}^2 = (1/3)\Delta x^2/\Delta t^2$.

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

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

This expresses that particles $f_i(\boldsymbol{x}, t)$ move with 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] At the same time, particles are affected by a collision operator Ω_i. This operator models particle collisions by redistributing particles among the populations f_i at each site.

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:

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

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

This equilibrium is given by

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

with the weights w_i specific to the chosen velocity set. 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 $\sum_i \boldsymbol{c}_i f_i^{\mathrm{eq}} =$

[2]There is a limit to how small a velocity set can be, though, as it has to obey the requirements shown in (3.60) 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 Sect. 12.1 on sound waves.

[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 (cf. Chap. 10).

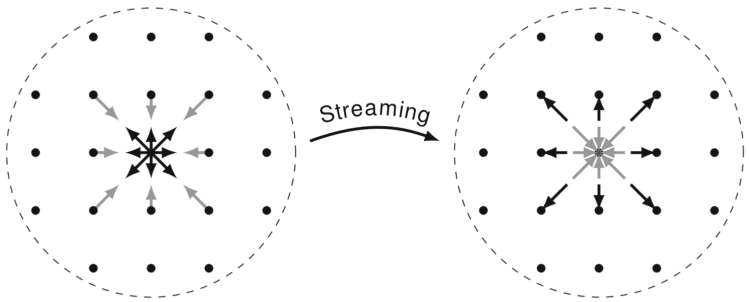

Fig. 3.1 Particles (*black*) streaming from the central node to its neighbours, from which particles (*grey*) are streamed back. To the left we see post-collision distributions $f_i^\star$ *before* streaming, and to the right we see pre-collision distributions f_i *after* streaming

$\sum_i \boldsymbol{c}_i f_i = \rho \boldsymbol{u}$.[6] The equilibrium f_i^{eq} depends on the local quantities density ρ and fluid velocity $\boldsymbol{u}$ only. These are calculated from the local values of f_i by (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 LBE and the NSE can be determined using the Chapman-Enskog analysis (cf. Sect. 4.1). Through this, we can show that the LBE results in macroscopic behaviour according to the NSE, with the kinematic shear viscosity given by the relaxation time τ as

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

and the kinematic bulk viscosity given as $\nu_\text{B} = 2\nu/3$. Additionally, the viscous stress tensor can be calculated from f_i as

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

where the *non-equilibrium* distribution $f_i^{\text{neq}} = f_i - f_i^{\text{eq}}$ 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.)

3.2.2 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

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

[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 $\sum_i \boldsymbol{c}_i \Omega_i = \boldsymbol{0}$.

We can decompose this equation 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 (3.4). Note that 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 $f_i^\star(\boldsymbol{x},t) = f_i^{\mathrm{eq}}(\boldsymbol{x},t)$.
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.

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 $\boldsymbol{u}$ to find the equilibrium distributions f_i^{eq} as in (3.4) and the post-collision distribution $f_i^\star$ as in (3.9). After collision, we *stream* the resulting distribution $f_i^\star$ to neighbouring nodes as in (3.10). When these two operations are complete, one time step has elapsed, and the operations are repeated.

3.3 Implementation of the Lattice Boltzmann Method in a Nutshell

We cover here some details of the implementation of the LBE to achieve an efficient and working algorithm. After discussing initialisation in Sect. 3.3.1 and the time step algorithm in Sect. 3.3.2, we provide details about the underlying memory structure and coding hints in Sect. 3.3.3.

The case covered here is the simplest force-free LB algorithm in the absence of boundaries. This core LB algorithm can be extended by including boundary conditions (cf. Chap. 5) and forces (cf. Chap. 6). More in-depth implementation advice can be found in Chap. 13.

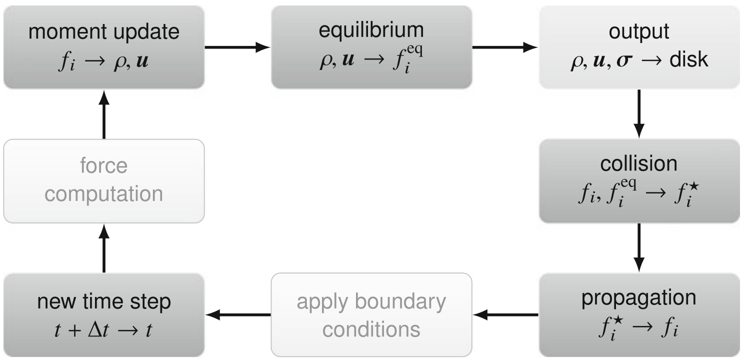
moment update
$f_i \to \rho, \boldsymbol{u}$
equilibrium
$\rho, \boldsymbol{u} \to f_i^{\text{eq}}$
output
$\rho, \boldsymbol{u}, \boldsymbol{\sigma} \to$ disk
collision
$f_i, f_i^{\text{eq}} \to f_i^\star$
propagation
$f_i^\star \to f_i$
apply boundary conditions
new time step
$t + \Delta t \to t$
force computation

Fig. 3.2 An overview of one cycle of the LB algorithm. The *dark grey boxes* show sub-steps that are necessary for the evolution of the solution. The *light grey box* indicates the optional output step. The *pale boxes* show steps whose details are given in later chapters (namely, Chap. 5 for boundary conditions, and Chap. 6 for forces)

3.3.1 Initialisation

The simplest approach to initialising the populations at the start of a simulation is to set them to $f_i^{\text{eq}}(\boldsymbol{x}, t = 0) = f_i^{\text{eq}}(\rho(\boldsymbol{x}, t = 0), \boldsymbol{u}(\boldsymbol{x}, t = 0))$ *via* (3.4). Often, the values $\rho(\boldsymbol{x}, t = 0) = 1$ and $\boldsymbol{u}(\boldsymbol{x}, t = 0) = \boldsymbol{0}$ are used. More details about different initialisation schemes are given in Sect. 5.5.

3.3.2 Time Step Algorithm

Overall, the core LBM algorithm consists of a cyclic sequence of substeps, with each cycle corresponding to one time step. These substeps are also visualised in Fig. 3.2:

1. Compute the macroscopic moments $\rho(\boldsymbol{x}, t)$ and $\boldsymbol{u}(\boldsymbol{x}, t)$ from $f_i(\boldsymbol{x}, t)$ *via* (3.1).
2. Obtain the equilibrium distribution $f_i^{\text{eq}}(\boldsymbol{x}, t)$ from (3.4).[7]
3. If desired, write the macroscopic fields $\rho(\boldsymbol{x}, t)$, $\boldsymbol{u}(\boldsymbol{x}, t)$ and/or $\boldsymbol{\sigma}(\boldsymbol{x}, t)$ to the hard disk for visualisation or post-processing. The viscous stress tensor $\boldsymbol{\sigma}$ can be computed from (3.6).
4. Perform collision (relaxation) as shown in (3.9).
5. Perform streaming (propagation) *via* (3.10).
6. Increase the time step, setting t to $t + \Delta t$, and go back to step 1 until the last time step or convergence has been reached.

[7]For those who want ready-to-program expressions, the unrolled equilibrium functions for D1Q3 and D2Q9 are shown in (3.64) and (3.65).

Note that the internal order of these sub-steps is important, as later steps depend on the results of earlier steps. However, the sub-step that is performed first can be chosen in different ways. Some people start a simulation with streaming rather than collision, and this does not make a significant difference in most cases. (Details on this are provided in Sect. 5.5.)

3.3.3 Notes on Memory Layout and Coding Hints

We provide here some useful coding hints for a successful LBM algorithm implementation. These considerations are the "common sense" and "common practice" implementations, and Chap. 13 contains more sophisticated explanations and advanced coding guidelines.

3.3.3.1 Initialisation

We have to allocate memory for the macroscopic fields ρ and $\boldsymbol{u}$ and the populations f_i. Usually, taking 2D as an example, the macroscopic fields are allocated in two-dimensional arrays $\rho[N_x][N_y]$, $u_x[N_x][N_y]$ and $u_y[N_x][N_y]$ and the populations are in a three-dimensional array $f[N_x][N_y][q]$. Here, N_x and N_y are the number of lattice nodes in x- and y-directions and q is the number of velocities. Other memory layouts are discussed in Chap. 13. Depending on the programming language and structure of the loops that update the simulation domain, it may be necessary to exchange the order of N_x, N_y, and q in the array dimensions to improve the speed at which memory is read and written.

3.3.3.2 Streaming

The streaming step has to be implemented in a way that ensures that the streamed populations do not overwrite memory that still contains unstreamed populations. In other words, as we sweep through the domain, we cannot overwrite data that we will need to use later. There are three common ways to efficiently implement streaming:

- Run through memory in "opposite streaming direction" and use a small temporary buffer for a single population. For example, for the direction $(0, 1)$, at each node we read the population for this direction from below, which is the direction opposite to the direction that is being streamed, and save it to the current node. The difficulty with this method is that we need to sweep through memory in a different direction for each discrete velocity, which is a common source of bugs. However, some programming languages provide convenient functions to do this, for example `circshift` in Matlab and Octave, `numpy.roll` in Python, and `CSHIFT` in Fortran.

- Allocate memory for two sets of populations, f_i^{old} and f_i^{new}, in order to store the populations for two consecutive time steps. During streaming, read data from $f_i^{\text{old}}(\boldsymbol{x})$ and write to $f_i^{\text{new}}(\boldsymbol{x} + \boldsymbol{c}_i\Delta t)$. Swap f_i^{old} and f_i^{new} after each time step so that the new populations at the end of each time step become the old populations for the next step. In this way, one does not have to care about how to stream populations without incorrectly overwriting data. The resulting code is significantly easier but requires twice the memory.[8]
- Avoid streaming altogether, and perform a combined streaming and collision step in which the necessary populations are read from adjacent nodes when they are needed to compute macroscopic variables and do the collision calculations at each node. In this case as in the previous, a second set of populations is used to store the result of the combined streaming and collision step. This approach is described in greater detail in Chap. 13.

3.3.3.3 Updating Macroscopic Variables

As seen in (3.1), the local density ρ and velocity $\boldsymbol{u}$ are (weighted) sums of the populations f_i. It is advisable to unroll these summations, which means writing out the full expressions rather than using loops. Most velocity components are zero, and we do not want to waste CPU time by summing zeros. For example, for the D1Q3 velocity set we have:

$$\rho = f_0 + f_1 + f_2, \quad u_x = \frac{f_1 - f_2}{\rho}. \tag{3.11}$$

For the D2Q9 velocity set as defined in Table 3.3, we would implement:

$$\begin{aligned} \rho &= f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8, \\ u_x &= \left[(f_1 + f_5 + f_8) - (f_3 + f_6 + f_7)\right]/\rho, \\ u_y &= \left[(f_2 + f_5 + f_6) - (f_4 + f_7 + f_8)\right]/\rho. \end{aligned} \tag{3.12}$$

3.3.3.4 Equilibrium

As for the macroscopic variable calculations, it is possible to accelerate the equilibrium computation in (3.4). Instead of computing all q distributions f_i^{eq} within a single for-loop over i, we should write separate expressions for each of the q distributions.

[8]For example, for a moderate simulation domain of $100 \times 100 \times 100$ lattice sites and the D3Q19 velocity set, one copy of the populations f_i requires about 145 MiB of memory, where we assume double precision (8 bytes per variable).

It is also strongly recommended to replace the inverse powers of the speed of sound, c_s^{-2} and c_s^{-4}, with new variables. For the standard lattices, the numerical values happen to be 3 and 9. However, it is better practice to keep distinct variables for those expressions as this will make it easier to switch to different velocity sets (with $c_\mathrm{s}^2 \neq (1/3)\Delta x^2/\Delta t^2$) if necessary. In any case, numerical divisions are expensive, and we can implement the equilibrium distributions without a single division.

3.3.3.5 Collision

From (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, an expensive numerical division can be removed from the collision operation:

$$f_i^\star(\boldsymbol{x}, t) = \omega' f_i(\boldsymbol{x}, t) + \omega f_i^\mathrm{eq}(\boldsymbol{x}, t). \tag{3.13}$$

This saves otherwise wasted CPU time. As detailed in Chap. 13, optimisations such as these can be performed automatically by the compiler if appropriate compilation options are selected. Running simulation codes compiled with no automatic optimisation is not recommended, and at least a basic level of optimisation should always be used. When compiled with no optimisation, the code in Chap. 13 took more than five times longer than when it was compiled with the highest level of optimisation.

3.4 Discretisation in Velocity Space

In Sect. 3.2 we gave an overview of LBM, now it is time to thoroughly derive the lattice Boltzmann equation from the Boltzmann equation. As a starting point we will first derive its velocity discretisation. One problem mentioned before 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 usually computationally expensive and requires large-scale computers and programming efforts.

This apprehension, however, is often not justified. As we found in Sect. 1.3.5, 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 the Mach number expansion [2] or the Hermite series expansion [3]. Both approaches give the same form of the equilibrium on the Navier-Stokes level. Although the Mach number expansion approach is simpler, we will follow the Hermite series approach as it provides a strong mathematical basis. Aside from delivering a variety of suitable discrete velocity sets, it can also correctly restore equations beyond the Navier-Stokes-Fourier level, i.e. Burnett-type equations [3].

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

We will see that a **simplified equilibrium** f^{eq} and a **discrete velocity space** are **sufficient to obtain the correct macroscopic conservation laws**.

In contrast to the unknown distribution function f, the *equilibrium* distribution function f^{eq} is a known function of 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 of a form which lets us evaluate them exactly as a *discrete sum* over the polynomial integrand evaluated at specific points $\boldsymbol{\xi}_i$ (*abscissae*). Thus, f^{eq} becomes discrete rather than continuous in velocity space. These techniques can also be applied to the particle distribution f itself.

Here we deal with the discretisation in velocity space. Later, in Sect. 3.5, we will perform the space-time discretisation.

3.4.1 Non-dimensionalisation

Before we start with the Hermite series expansion, we *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 Chap. 7.)

Let us recall the Boltzmann equation in continuous velocity space:

$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} + \frac{f_\alpha}{\rho}\frac{\partial f}{\partial \xi_\alpha} = \Omega(f) \tag{3.14}$$

where $\Omega(f)$ is the collision operator. Its specific form is not important here.[9]

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 (3.14) vanishes, and the solution of the Boltzmann equation becomes the equilibrium distribution function f^{eq}. As we found in Sect. 1.3.3, f^{eq} can be written in terms of the macroscopic quantities of density ρ, fluid velocity $\boldsymbol{u}$ and temperature T as

$$f^{\text{eq}}(\rho, \boldsymbol{u}, T, \boldsymbol{\xi}) = \frac{\rho}{(2\pi RT)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}-\boldsymbol{u})^2/(2RT)} \tag{3.15}$$

where d is the number of spatial dimensions and the gas constant $R = k_\text{B}/m$ 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 of 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 Sect. 1.2, we can analyse the properties of a fluid in terms of its characteristic length ℓ, velocity V and density ρ_0. A 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:

$$\frac{\partial}{\partial t^\star} = \frac{\ell}{V}\frac{\partial}{\partial t}, \quad \frac{\partial}{\partial x^\star} = \ell\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial \xi^\star} = V\frac{\partial}{\partial \xi}. \tag{3.16}$$

This leads to the non-dimensional continuous Boltzmann equation:

$$\frac{\partial f^\star}{\partial t^\star} + \xi^\star_\alpha \frac{\partial f^\star}{\partial x^\star_\alpha} + \frac{F^\star_\alpha}{\rho^\star}\frac{\partial f^\star}{\partial \xi^\star_\alpha} = \Omega^\star(f^\star) \tag{3.17}$$

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

$$f^{\text{eq}\star} = \frac{\rho^\star}{(2\pi\theta^\star)^{d/2}} \mathrm{e}^{-(\boldsymbol{\xi}^\star-\boldsymbol{u}^\star)^2/(2\theta^\star)} \tag{3.18}$$

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

[9] As we will see later in Chap. 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.

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

We will perform the Hermite series expansion in the force-free case ($\boldsymbol{f} = \boldsymbol{0}$) to reduce the level of complexity. The effect of forces on the Hermite series expansion will be covered in Sect. 6.3.1.

The **non-dimensional, continuous and force-free Boltzmann equation** is

$$\frac{\partial f}{\partial t} + \xi_\alpha \frac{\partial f}{\partial x_\alpha} = \Omega(f). \tag{3.19}$$

The **non-dimensional equilibrium distribution** reads

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

3.4.2 Conservation Laws

We have already discussed conservation laws in Sect. 1.3.4. 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:

$$\begin{aligned}
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right)\, \mathrm{d}^3\xi &&= \rho(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\boldsymbol{\xi}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \boldsymbol{\xi}\, \mathrm{d}^3\xi &&= \rho\boldsymbol{u}(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\frac{|\boldsymbol{\xi}|^2}{2}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \frac{|\boldsymbol{\xi}|^2}{2}\, \mathrm{d}^3\xi &&= \rho E(\boldsymbol{x}, t), \\
\int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\frac{|\boldsymbol{\xi}-\boldsymbol{u}|^2}{2}\, \mathrm{d}^3\xi &= \int f^{\mathrm{eq}}\left(\rho, \boldsymbol{u}, \theta, \boldsymbol{\xi}\right) \frac{|\boldsymbol{\xi}-\boldsymbol{u}|^2}{2}\, \mathrm{d}^3\xi &&= \rho e(\boldsymbol{x}, t).
\end{aligned} \tag{3.21}$$

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

All conserved quantities on the right-hand side of (3.21) 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 $\boldsymbol{\xi}$). We will now discuss the properties of Hermite polynomials that form the basis of the Hermite series expansion.

3.4.3 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 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 NSE) requires some knowledge of tensor notation. Readers without or with only little experience with tensors should take some time to learn the basics of tensor calculus [4, 5].

3.4.3.1 Definition and Construction of Hermite Polynomials

1D HPs are polynomials which can be obtained from the *weight function* (also called the *generating function*

$$\omega(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2}. \tag{3.22}$$

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

$$H^{(n)}(x) = (-1)^n \frac{1}{\omega(x)} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \omega(x) \tag{3.23}$$

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

$$\begin{aligned} &H^{(0)}(x) = 1, && H^{(1)}(x) = x, \\ &H^{(2)}(x) = x^2 - 1, && H^{(3)}(x) = x^3 - 3x, \\ &H^{(4)}(x) = x^4 - 6x^2 + 3, && H^{(5)}(x) = x^5 - 10x^3 + 15x. \end{aligned} \tag{3.24}$$