NTNU – Trondheim
Norwegian University of
Science and Technology

Lattice Boltzmann Simulation of Acoustic Fields, with Special Attention to Non-reflecting Boundary Conditions

Sigurd Johannes Benedikt Stoll

Electronics System Design and Innovation
Submission date: February 2014
Supervisor: Ulf R Kristiansen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

SIGURD STOLL

LATTICE BOLTZMANN SIMULATION OF ACOUSTIC FIELDS, WITH SPECIAL ATTENTION TO NON-REFLECTING BOUNDARY CONDITIONS

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATIONS
SUPERVISOR: ULF KRISTIANSEN

FEBRUARY 1, 2014

Abstract

Non-reflecting boundary conditions (NRBCs) are of great importance in computational fluid dynamics. Ideal NRBCs will fully absorb every incoming wave, thereby eliminating reflections from the boundaries which would otherwise interfere with the simulation inside the domain.

In this thesis, three different types of existing NRBCs, perfectly matching layers (PMLs), characteristic boundary conditions (CBCs) and sponge layers, for the lattice Boltzmann method are summarized and compared against each other and an ideal case. In addition, tweaking of performance related parameters in each NRBC is performed to achieve the best performance. Simulations are done with two types of acoustic field excitations, a Gaussian pulse and a sinusoidally varying single point source.

In both cases the lattice Boltzmann method with PMLs performed slightly better than CBCs, and gave the least reflections, while the lattice Boltzmann method with sponge layers gave the most reflections.

Preface

> *Easy reading is damned hard writing.*
>
> Nathaniel Hawthorne

In addition to the obvious goal of writing a thesis that merits a good grade, I had a goal of a purely pedagogic nature, I wanted to write a thesis which conveyed its contents in a simple manner, making it as easy as possible to digest for readers who are as unfamiliar with the lattice Boltzmann method and fluid dynamics in general, as I were when I first started this thesis.

Many a times have I read a scientific article which not only presents a complex topic, but does so without giving any explanation of specific jargon[1] used or providing the reader with detailed explanations. This is most likely due to the fact that restrictions, such as article length, are often placed upon the author(s) by the publishers, and of course the fact that most articles are aimed at an audience with pre-existing knowledge of the field/topic. However, being a bit freer as regards to length and style, I wanted to create something that provides the reader with numerous explanations and details regarding the contents of this thesis. The idea is that the reader will find explanations on possibly unfamiliar jargon as well as details regarding procedures and methods used herein. However, including too many details creates the risk of obfuscating the main topics and purposes, and may end up confusing the reader even more. Therefore, I have tried to restrict myself in situations where I have felt that an inclusion of all the details would have detracted from the whole. Instead I have given concrete references to books or articles containing the details wherever possible.

[1] Jargon consists of words and expressions that are used in special or technical ways by particular groups of people.

As for the style of this thesis, it has been carefully chosen to conform with my second goal. The style is based on the books by Edward Tufte[2] [1, 2, 3, 4]. One of the most prominent and distinctive features of the Tufte style is the extensive use of sidenotes, as can bee seen from the jargon example above. The sidenotes are placed in a wider

[2] Edward Tufte is an American professor emeritus of political science, statistics and computer science, he is noted for his writings on information design and data visualization.

than usual margin, this has two benefits. One: Experienced readers who are familiar with the jargon and the different methods are presented with a continuous, flowing text without, for them, unnecessary explanations. Two: The wide sidemargins automatically gives us a shorter line length, approximately between 60 and 70 characters long, which is recognized by many as the optimal line length (given a reasonable choice of font size, e.g. 12pt font size) for legibility [5].

Lastly I would like to acknowledge, and say a big thank you to the people who have helped me during the process of writing this thesis. My supervisor Ulf Kristiansen who has kindly given of his time and answered every question that was asked. Erlend Magnus Viggen whose master and PhD thesis I have relied heavily upon. Especially his PhD thesis have both influenced and inspired me, as well as served as a great source of knowledge about the lattice Boltzmann method. Lastly I would like to thank Alireza Najafi-Yazdi and Daniel Heubes who have patiently answered all my emails and explained some of the finer points regarding non-reflecting boundary conditions.

Contents

1 Introduction

In this chapter we briefly mention the phenomenon sound as well as introducing and explaining the microscopic, mesoscopic and macroscopic scale. In the end, a short history of the lattice Boltzmann method is given.

1.1 Briefly on acoustics and sound in general

Acoustics was originally the study of small pressure waves in air which can be detected by the human ear, i.e. sound. Nowadays the scope of acoustics is much wider and include areas like ultrasound and infrasound which are inaudible to the human ear. Keeping with the original definition, acoustics is a part of fluid dynamics as the pressure waves which make up what we as humans perceive as sound, propagates through some fluid (usually air).

When it comes to the creation of sound waves, the most common example is a loudspeaker which produces sound waves by means of rapidly vibrating surfaces, and indeed any surface vibrating in a certain manner will produce sound waves. Other sources of sound include aeroacoustic processes like turbulent flow[1] and sound created by aerodynamic forces, i.e. air in motion interacting with solid objects.

[1] Turbulence, or turbulent flow, is characterized by its chaotic properties, among which are rapidly varying pressure and velocity in space and time.

1.2 Scales

We will start off by introducing the three different scale levels that we operate with in this thesis. Different equations and simulation methods produce information/results pertaining to different scales, an understanding of the different scale levels are therefore beneficial. A short introduction and what one can expect to find at these scales are presented below.

- **The microscopic scale:** The system is described in its entirety through equations describing the interactions among each particle in the system. This is the most complete description of a system, but also the least computationally feasible as the amount of particles and phenomena affecting them are staggering. Tan-

gible concepts like density or fluid velocity does not exist on this level. Examples of simulation methods performed on this level are molecular dynamic methods, among them is the LGA method later mentioned in subsection 1.3.2.

- **The mesoscopic scale:** This is the scale between the microscopic and the macroscopic scale. The system can be described statistically. Rather than describing particles, we are now using particle distributions to describe the evolution of the system. The lattice Boltzmann operates on this level.
- **The macroscopic scale:** On this level we are not using particles at all to describe the system, rather we describe the system as a continuum. This is the scale we are able to observe with our own eyes. As an example, a body of water will look continuous and smooth at this level, although actually composed of a myriad of loosely connected, tiny particles. The concept of density will now cease to be an abstract principle and becomes something tangible, the same for fluid velocity, which was only an idea or abstract principle at the microscopic level.
  The term macroscopic variable (which is a term we will encounter frequently) is often used to emphasize that the variable in question relates to something tangible, rather than some variable found only at microscopic or mesoscopic level with no conceptual interpretation in the macroscopic scale. The Navier-Stokes-Foureir model mentioned in subsection 2.2.2 operates at this level.

Although these three scopes offer very different descriptions of a system, they still describe the same system, a body of water is still a body of water no matter if it is described by its microscopic or its macroscopic properties. What links them together is the expectation value[2] of different characteristics found in the microscopic description. As an example we use the density. Density ρ has the dimension of $\mathrm{kg\,m^{-3}}$ so it is reasonable to expect that the definition of density must involve some mass m in some specified volume V, indeed this is the case as we have

[2] The expectation value implies the use of the expectation operator $\mathbb{E}(\cdot)$ and can in this case be be thought of as an ideal average.

$$\rho(\mathbf{x}, t) = \lim_{V \to 0} \mathbb{E}\left(\frac{1}{V} \sum_{\mathbf{x}_i \in V} m_i \right). \tag{1.1}$$

This equation tells us that the macroscopic density at a given point $\mathbf{x}$ in space, at a given time t is found by averaging the sum of each particle mass m_i found in an arbitrary small volume V. Thus the tangible quantity that we perceive as density is actually the averaged mass of every particle in an arbitrarily small volume. Equation (1.1) can therefore be said to link the microscopic and macroscopic scales.

1.3 The history of the lattice Boltzmann method

1.3.1 Cellular automata

The very first step towards what would become the lattice Boltzmann method (LBM) was taken with the cellular automata (CA), which was proposed by Ulam and van Neumann in the late 1940s to simulate life. Conway's Game of Life[3] is probably the most famous implementation of a CA and demonstrates beautifully how, with suitable rules, a CA can be used to simulate complex physical phenomenon in the real world, as was shown by Wolfram in [6, 7].

[3] The Game of Life is one of the simplest examples of what can be described as "emergent complexity" and can be studied to give insight into how elaborate patterns and behaviors can emerge from very simple rules, e.g. how the stripes of a Zebra can arise from a tissue of living cells growing together.

1.3.2 Lattice gas automata

The next step towards the LBM was the lattice gas automata (LGA), which is a particular class of the CA. The LGA is a discrete macroscopic model of a fluid where fictitious particles resides in nodes on a regular lattice and the particles can only move in accordance with the lattice directions. An example of lattice directions can be found in Figure 3.1 which represents the particle velocities in a D2Q9 lattice[4]. In addition, particles collide when they meet in the same node, and mass and momentum of the system is conserved, this is a very important feature lacking in the CA.

[4] The term DnQm indicates that the lattice in question is n dimensional and m velocities are applied in the lattice. There exists many such lattices, e.g. D2Q9, D3Q15, D3Q27 etc.

This led to a fully discrete model for a fluid (which is now commonly called the HPP model) and it was shown by Hardy, Pazzis and Pomeau in [8] that the model simulate flow equations. However, the model was built on the 4-speed square lattice which resulted in anisotropic[5] flow equations. This anisotropic property means that the system will not be able to behave in accordance with the Navier-Stokes equations[6]. Not until ten years later was there developed a model which could recover the Navier-Stokes model correctly, this model was know as the FHP model after its authors Frisch, Hasslacher and Pomeau [9]. The FHP model was based on a 6-speed hexagonal lattice as it was discovered that the symmetry of the lattice plays a vital role in recovering the Navier-Stokes equations.

[5] Anisotropy is the property of being directionally dependent, e.g. having different magnitudes when measured in different directions.

[6] The Navier-Stokes equations are some of the most important equations in fluid dynamics and are used to describe the motion of fluid substances and can therefore be used to simulate everything from weather and ocean currents, to water flow in a pipe and air flow around a wing.

The LGA comprises two steps which is worth explaining as the LBM can be called a direct descendant of the LGA, and therefore share some similarities. These two steps are streaming and collision which describes the streaming of the particles in the system and collision between said particles. Streaming happens along the lattice lines connecting the different nodes in the system, while collision is represented by a collision operator. These two steps represents two macroscopic phenomena, namely convection[7] and diffusion[8] respectively. These two phenomena determine the basic features of the LGA, and the equation for the LGA is given by

[7] Convection is actually the sum of both the diffusion and the advection phenomenon, but it suffices to think of convection as the movement of particles due to large scale motion of currents in the fluid.

[8] Diffusion is a transport phenomena and can be described as "the spreading out" of particles, it is important to note that diffusion does not depend on movement of the surrounding fluid to spread the particles. The particles diffuse due to the Brownian motion of

$$n_i(\mathbf{x} + \mathbf{c}_i, t + 1) = n_i(\mathbf{x}, t) + \Omega_i(\mathbf{x}, t) \tag{1.2}$$

Where $n_i(\mathbf{x}, t)$ is a Boolean variable that is used to indicate the presence ($n_i(\mathbf{x}, t) = 1$), or lack thereof ($n_i(\mathbf{x}, t) = 0$), of a particle residing in node $\mathbf{x}$ with velocity $\mathbf{c}_i$, and $\Omega_i(\mathbf{x}, t)$ is the collision operator. For an explanation of the notation used, namely $\mathbf{x}$ and the subscript i, see subsection 2.1.1.

Using this we can recover the macroscopic variables density and fluid velocity:

$$\rho(\mathbf{x}, t) = \sum_i n_i(\mathbf{x}, t), \quad u_i(\mathbf{x}, t) = \frac{1}{\rho(\mathbf{x}, t)} \sum_i \mathbf{c}_i n_i(\mathbf{x}, t). \tag{1.3}$$

Unfortunately, simulations with a LGA suffer from statistical noise. Macroscopic variables, e.g. density, recovered from the microscopic simulation will be noisy due to the fact that the microscopic system is subject to random fluctuations that disappear in the continuum limit[9] [10]. This problem spurred the development of the lattice Boltzmann method.

[9] The continuum limit in this context describes the transition from a discrete system with a countable number of particles N, into a continuous system of $N \to \infty$ particles. The transition from discrete to continuous alters some of the mathematics involved, as an example summations signs are converted into integrals (with and appropriate normalizing factor).

1.3.3 Lattice Boltzmann method

A solution to the problem inherent in the LGA was found by McNamara and Zanetti [11], where the authors replaced the Boolean variable $n_i(\mathbf{x}, t)$ with its ensemble average[10]

$$f_i(\mathbf{x}, t) = \langle n_i(\mathbf{x}, t) \rangle, \quad 0 \leq f_i(\mathbf{x}, t) \leq 1. \tag{1.4}$$

Using this, (1.2) becomes

$$f_i(\mathbf{x} + \mathbf{c}_i, t + 1) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{x}, t). \tag{1.5}$$

This is the lattice Boltzmann evolution equation. A more thorough discussion surrounding this equation is given in chapter 3.

[10] Ensemble average can be explained thus: Time averaging concerns itself with a single system over a given time interval. However, noise is a stochastic process consisting of a randomly varying function of time and space, and as such can only be characterized statistically. Therefore it makes no sense to single out specific events during a given time interval and averaging them. This is where the ensemble average comes in as this is the averaged quantity of many identical (random) systems at a certain time.

2 Theory

In this chapter we will cover specific notation and definitions used, as well as introducing two of the most common models in fluid dynamics, the Euler model and The Navier-Stokes-Fourier model.

2.1 Notation and useful definitions

2.1.1 Vector notation

In this thesis, every vector is presented with a boldfaced letter. An example is position, where the vector $\mathbf{x} = [x\hat{\mathbf{e}}_x, y\hat{\mathbf{e}}_y]$ is used, here $\hat{\mathbf{e}}_x$ and $\hat{\mathbf{e}}_y$ are unit vectors. The vector $\mathbf{x}$ represents a point in two dimensional Cartesian space, which at first might be a bit confusing since the letter x usually represents some position on the horizontal axis in a Cartesian system, nevertheless, the use of $\mathbf{x}$ is commonly found in the literature surrounding the LBM and we will use this definition in this thesis.

2.1.2 Index notation

Index notation is a compact way of writing an equation which acts in different spatial directions, instead of writing

$$\mathbf{F}(x, y, z) = m\mathbf{a}(x, y, z), \tag{2.1}$$

or even

$$[F_x, F_y, F_z] = m[a_x, a_y, a_z], \tag{2.2}$$

we simply write

$$F_\alpha = ma_\alpha. \tag{2.3}$$

Equation (2.3) uses index notation, the fact that F_α and a_α are vector components are conveyed through the index α. Note that Greek indices (α, β, γ etc.) are used for spatial components, e.g. x, y and z in Cartesian space. Components of general non-spatial tensors[1] use the indices i, j, k etc. instead.

Another neat feature of index notation is that it allows us to directly address elements in a tensor, in (2.3), the single index α tells

[1] A tensor can be seen as a generalization of scalars, vectors and matrices, a zeroth order tensor is a scalar, a first order tensor is a vector, second order represents a matrix etc.

us that we are dealing with element α of a vector, we know it is a vector because a vector can be uniquely addressed with only one index. Likewise, $A_{\alpha\beta}$ would indicate element α, β in matrix $\mathbf{A}$.

2.1.3 Material derivative

The material derivative is a common concept used in fluid dynamics and can be interpreted as a time derivative of a particle moving through a fluid with velocity $\mathbf{u}$. It is commonly defined as

$$\frac{\mathrm{D}\lambda}{\mathrm{D}t} = \frac{\partial\lambda}{\partial t} + \mathbf{u}\cdot\nabla\lambda, \tag{2.4}$$

where $\lambda(\mathbf{x}, t)$ is a generic quantity. Note that the material derivative is sometimes confusingly called the convection derivative or the advection derivative.

2.1.4 Miscellaneous

For convenience, functions of one or more variables will from time to time appear in a shorter form, e.g. $\rho(\mathbf{x}, t)$ will be presented as ρ. This is to prevent the equations from becoming too messy, however, the proper form will still be used where it is deemed necessary for sake of completeness or emphasis.

2.2 Fluid dynamics

As the name implies, fluid dynamics is the branch of physics that concerns itself with the studies of the flow of fluids[2]. This is a huge topic which we will not delve deeply into here. However, fluid dynamics serves as the backbone of any fluid simulation and a few key principles and equations are therefore presented here in order to provide some bare minimum of theoretical background.

[2] Fluids does not only comprise of liquids, but also of gases and plasmas.

A fundamental part of fluid dynamics is the conservation equations which tells us that in a closed system, mass, momentum and total energy is always conserved. There are several sets of equations, or models as they are sometimes called, which include these conservation laws as well as other important terms. We will now take a closer look on the first two of these models and then briefly mention the other models available to us.

2.2.1 The Euler model

The Euler model for fluids consists of the following set of equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u_\alpha) = 0, \tag{2.5a}$$

$$\rho \frac{\mathrm{D}u_\alpha}{\mathrm{D}t} = -\nabla p + F_\alpha, \tag{2.5b}$$

$$\rho \frac{\mathrm{D}e}{\mathrm{D}t} = -p \nabla \cdot u_\alpha. \tag{2.5c}$$

Where F_α is the external body force density, this consists typically of gravity forces and p is the pressure. Equations (2.5a), (2.5b) and (2.5c) are the conservation equations for mass, momentum and energy respectively.

Acoustic waves in general are subject to energy loss/damping as a result of the viscosity of the fluid in which the sound wave propagates, and as a result of heat transfer due to said viscosity. This level of description is lacking from the Euler model and hence it is best suited to model inviscid flow[3]. Despite this, the Euler model can sometimes provide accurate enough results for fluids with low viscosity as they behave like inviscid fluids.

[3] Inviscid flow is the flow of an ideal fluid, it is considered ideal since it does not have viscosity.

2.2.2 Navier-Stokes-Fourier model

A more detailed model is the Navier-Stokes-Fourier (NSF) model which consists of the following equations

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u_\alpha) = 0, \tag{2.6a}$$

$$\rho \frac{\mathrm{D}u_\alpha}{\mathrm{D}t} = -\frac{\partial p}{\partial x_\alpha} + \frac{\partial \sigma'_{\alpha\beta}}{\partial x_\beta} + F_\alpha, \tag{2.6b}$$

$$\rho \frac{\mathrm{D}e}{\mathrm{D}t} = \left(-\delta_{\alpha\beta} p + \sigma'_{\alpha\beta}\right) \frac{\partial u_\beta}{\partial x_\alpha} - \frac{\partial q_\alpha}{\partial x_\alpha}. \tag{2.6c}$$

Equation (2.6a) is the familiar mass conservation equation also found in the Euler model, but the conservation equation for momentum (2.6b) and energy (2.6c) are presented in a new form. Equation (2.6b) is now given as the general Cauchy momentum equation:

$$\rho \frac{\mathrm{D}u_\alpha}{\mathrm{D}t} = \frac{\partial \sigma_{\alpha\beta}}{\partial x_\beta} + F_\alpha, \tag{2.7}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor[4], which in (2.6b) has been split into $-p\mathbf{I} + \boldsymbol{\sigma}'$, $\mathbf{I}$ is the identity matrix, and the deviatoric stress tensor $\boldsymbol{\sigma}'$ for a simple fluid is given by

$$\sigma'_{\alpha\beta} = \mu \left(\frac{\partial u_\beta}{\partial x_\alpha} + \frac{\partial u_\alpha}{\partial x_\beta} - \frac{2}{3} \delta_{\alpha\beta} \frac{\partial u_\gamma}{\partial x_\gamma} \right) + \mu' \delta_{\alpha\beta} \frac{\partial u_\gamma}{\partial x_\gamma}. \tag{2.8}$$

Here, $\delta_{\alpha\beta}$ is the Kroenecker delta, μ is the dynamic shear viscosity[5]

[4] The Cauchy stress tensor $\boldsymbol{\sigma}$ is a second order tensor which describes the normal and shear stress in the x, y and z directions at any point in the fluid.

[5] The dynamic shear viscosity μ tells us something about a fluid's resistance to shearing flows. A shearing flow can be described as adjacent fluid layers moving in parallel with different velocities.

and μ' is the bulk viscosity[6]. The point of this splitting is simply to divide total stress into contributions from the pressure and from the viscosity.

[6] The bulk viscosity μ' is important when a fluid is rapidly expanding or contracting (e.g. sound waves), it reduces to zero for incompressible fluids. Note that this parameter is also called the volume viscosity.

The last equation in the NSF model is the energy conservation equation, which also uses the split Cauchy stress tensor, as well as the term q_α which is the heat flux given by Fourier's law,

$$q_\alpha = -\kappa \frac{\partial T}{\partial x_\alpha}, \tag{2.9}$$

where κ is the thermal conductivity of a material and T is the temperature.

Equation (2.6b) with the deviatoric stress tensor given in (2.8) yields one form of the famous compressible Navier-Stokes equation, but as it is sometimes useful to represent compressible Navier-Stokes without the use of the material derivative, another very common representation is:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\right) + \nabla p - \mu\Delta\mathbf{u} - \left(\frac{\mu}{3} + \mu'\right)\Delta\cdot\mathbf{u} = \mathbf{f}, \tag{2.10}$$

where Δ is the Laplace operator and $\mathbf{f}$ is the external body force (if any) given by

$$\mathbf{f} = \frac{\mathrm{d}\mathbf{F}}{\mathrm{d}v} = \rho\mathbf{a}. \tag{2.11}$$

Here $\mathbf{F}$ is again the external body force density and $\mathbf{a}$ is the acceleration caused by the force and V is the volume of the body the force acts upon.

As we have seen, there is at least two ways to write the compressible Navier-Stokes equation and more can be found in the literature. As a small addendum to (2.10) and (2.6b), the dynamic shear and bulk viscosity μ and μ' are sometimes replaced with the kinematic shear and bulk viscosity given as $\nu = \mu/\rho$ and $\nu' = \mu'/\rho$ respectively.

The Navier-Stokes-Fourier equations are very important in fluid dynamics as they describes the motion of a viscous, heat conducting fluid, and in fact, it is this equation that is simulated (with some modifications) by the lattice Boltzmann method used in this thesis. This is most commonly proven by way of Chapman-Enskog expansion. The main idea behind the Chapman-Enskog expansion is briefly mentioned in section 3.4.

2.2.3 Beyond Navier-Stokes-Fourier

Using what is called a zeroth order Chapman-Enskog expansion, we get conservation laws in accordance with the Euler model, a first order Chapman-Enskog expansion produces the NSF model. Higher order Chapman-Enskog (CE) expansions can also be calculated, which

produce even more detailed models. None of these higher order CE expansions are used in this thesis, but they are nevertheless interesting, this subsection is therefore a short, but hopefully interesting digression.

- *The Burnett, super-Burnett and the augmented Burnett equations:* Second and third order CE expansion produces the Burnett and super-Burnett equations. These equations, or models, are applicable for flows where the Knudsen number[7] lies in the continuum-transition regime, this implies $Kn \geq 1$. Typical processes where such Knudsen numbers are encountered include high altitude flight, shock waves and flows in micro-channels of micro-electromechanical devices.

  The reason why Burnett and super-Burnett is necessary is because the NSF model is only valid for $Kn \ll 1$ [12]. However, despite their increased level of detail, there is a drawback with these models as they display some instability issues, as shown by Bobylev in [13].

  To combat the stability issues in the Burnett and super-Burnett equations, Zhong, Maccormack and Chapman [14] came up with the augmented Burnett equations where some terms of super-Burnett were added to Burnett to stabilize the equations. Although the Burnett and augmented Burnett equations has shown some success for fluid simulations [12], [15] and [16], the sheer complexity of these equations and the fact that even the augmented Burnett show instability issues [17] means that these models should be used with care or altogether avoided.

- *R13 equations:* In addition to the CE method, the best known alternative to produce models describing fluid flow, is Grad's method of moments. Unlike the Burnett and super-Burnett equations produced by the CE method, Grad's equations are always stable [18]. The R13 equations are derived using a combination of CE expansion and Grad's method of moments and incorporates the benefits from both methods while avoiding the shortcomings. The R13 equations has been shown to give accurate and meaningful results, for example for Couette[8] and Poiseuille[9] flows with heat transfer, as was shown by Struchtrup and Torrilhon in [19].

[7] The Knudsen number Kn is a dimensionless number and is given as the ratio of the molecular mean free path and a relevant characteristic length scale. It can serve as an indicator of whether statistical mechanics or continuum mechanics should be applied to the problem, depending on the value of Kn.

[8] Couette flow is the flow of a viscous fluid situated between two parallel plates, one moving relative to the other. An often used test scenario in CFD simulations.

[9] A Poiseuille flow is a flow which obeys the Hagen–Poiseuille equation which describes the pressure drop in a fluid moving through a long cylindrical pipe, e.g. the flow through a drinking straw or a hypodermic needle. Poiseuille flow is an often used test scenario in CFD simulations.

3 The lattice Boltzmann method

Here we cover the lattice Boltzmann method as well as some important elements surrounding it like boundary conditions and lattice units.

As the topic of this thesis in not the lattice Boltzmann method itself, importer equations, relations and quantities are presented rather than derived, but references will be given so the curious reader can indulge themselves in the details found therein. In addition, it is assumed that we are using a D2Q9 model.

3.1 The lattice Boltzmann equation

We begin by introducing two very important concepts in kinetic theory[1]. Firstly, the phase density, $f(\mathbf{x}, t, \boldsymbol{\xi})$, which when known gives us an (almost) complete description of the state of the gas in question [18, Ch. 2]. The function $f(\mathbf{x}, t, \boldsymbol{\xi})$ is the distribution of a single particle at position $\mathbf{x}$, with particle velocity $\boldsymbol{\xi}$, at time t, this is therefore often called the particle distribution function instead of the phase density. Secondly, the continuous Boltzmann equation[2] given as

$$\frac{\partial f}{\partial t} + \xi \cdot \nabla f + \frac{\mathbf{F}}{\rho} \cdot \nabla f = \Omega(f). \tag{3.1}$$

Here, $\mathbf{F}$ the body force density acting on the particle and $\Omega(f)$ is the collision operator which when applied to f gives the rate of change due to collision between particles. A derivation of (3.1) can be found in [20, Ch. 3].

The collision operator $\Omega(f)$ deserves some further mentioning as this is usually a very complex term that severely complicates the solving of (3.1), as an example of its complexity, by the Stosszahlansatz[3], the collision operator $\Omega(f)$ can be written as

$$\Omega(f) = \int \int_0^{2\pi} \int_0^{\pi/2} [f(\mathbf{x}, t, \mathbf{c}') f(\mathbf{x}, t, \mathbf{c}^{1\prime}) - f(\mathbf{x}, t, \mathbf{c}) f(\mathbf{x}, t, \mathbf{c}^1)] g\sigma \sin(\Theta) \, \mathrm{d}\Theta \, \mathrm{d}\epsilon \, \mathrm{d}\mathbf{c}^1.$$

For more information on this equation and the Stosszahlansatz, see [18, Ch. 3.1].

[1] The kinetic theory of gases describes a gas consisting of a larger number of particles. The particles are in constant, random motion and are also constantly colliding with each other and the walls of their container. In addition, kinetic theory explains the macroscopic properties of gases, such as pressure, temperature, viscosity, thermal conductivity and volume.

[2] The Boltzmann equation is the central equation in kinetic theory of gases and describes the time and space evolution of the phase density $f(\mathbf{x}, t, \boldsymbol{\xi})$.

[3] The Stosszahlansatz is also called molecular chaos and is the assumption that the velocities of colliding particles are uncorrelated and independent of position.

To reduce complexity, an often used simplification is the BGK collision operator Ω_{BGK} by Bhatnagar, Gross, and Krook [21], which when used gives us a continuous Boltzmann equation on the form

$$\frac{\partial f}{\partial t} + \xi \cdot \nabla f = \Omega_{\mathrm{BGK}}(f) = -\frac{1}{\tau}(f - f_{\mathrm{M}}). \tag{3.2}$$

Here, f_{M} is the Maxwell-Boltzmann equilibrium distribution and τ is the relaxation time that indicates how fast the particle distribution function relaxes towards its equilibrium, sometimes a relaxation frequency $\omega = 1/\tau$ is used instead. Note that the force term $\mathbf{F}$ found in (3.1) has been set to zero in (3.2) as it is not of importance in this thesis.

As the continuous Boltzmann equation, even with a simplified collision operator as in (3.2), usually is too complex to solve analytically, we have to find a way around. One way would be to find a discrete version of this equation, which in turn can be solved numerically, and this is exactly what the lattice Boltzmann (LB) equation is, a finite difference form of the continuous Boltzmann equation [22].

The first step towards a fully discrete lattice Boltzmann equation from (3.2) is to discretize the velocity space[4]. There are several ways to do this, one is to approximate the Maxwell-Boltzmann distribution f_{M} up to second order via a Taylor series expansion and restricting particle velocities $\boldsymbol{\xi}$ to a finite set of velocities $\mathbf{c}_i$, thereby effectively reducing $f(\mathbf{x}, t, \boldsymbol{\xi})$ to $f_i(\mathbf{x}, t)$. In other words, the particle distribution function f_i can now only propagate in i directions, where $i \in [0, 1, \ldots, 8]$ in the D2Q9 model.

[4] To explain what velocity space is, we first need to describe something called phase space. Phase space is a space in which all possible states of a system are represented, in our case the phase space comprises of position $\mathbf{x}$, time t and particle velocity $\boldsymbol{\xi}$. The velocity space is then a subset of phase space. Generally speaking, the coordinates of velocity space are the velocities in each of the three spatial directions x, y and z.

Together with the discrete collision operator $\Omega_{\mathrm{BGK}}(f_i)$ we now have what is called the discrete velocity Boltzmann equation:

$$\frac{\partial f_i}{\partial t} + \mathbf{c}_i \cdot \nabla f_i = -\frac{1}{\tau}\left(f_i - f_i^{\mathrm{eq}}\right), \tag{3.3}$$

where the function f_i^{eq} is the equilibrium distribution. For more details on the derivation, see [20, Ch. 4].

Next, discretization of both time and space has to be performed. Again there are more than one way to do this, and some ways produce a more accurate model than others. The model, or rather, the lattice Boltzmann equation presented here is, according to Lätt [23], second order accurate both in time and in space for simulations of compressible, isothermal[5] flows at small Mach numbers. However the author mentions that this has never been proven formally (as of 2007). For details on the various discretization steps, see [20, Ch. 4], [22] or [24, Ch. 5]. In conclusion, the lattice Boltzmann equation[6] is given as

$$f_i(\mathbf{x} + \mathbf{c}_i\Delta t, t + \Delta t) = -\frac{1}{\tau}[f_i(\mathbf{x}, t) - f_i^{\mathrm{eq}}(\mathbf{x}, t)] + f_i(\mathbf{x}, t), \tag{3.4}$$

where Δt and Δx (not included in (3.4)) are the time resolution and lattice resolution respectively. During simulation the time and lattice

[5] An isothermal flow describes a flow in which the temperature stays constant. Under the isothermal assumption, the density ρ is related to pressure p via the linear equation $p = c_{\mathrm{s}}^2\rho$, where c_{s} is the speed of sound in $\mathrm{m\,s^{-1}}$.

[6] The lattice Boltzmann equations is sometimes given the more specific name BGK lattice Boltzmann equation because of the BGK collision operator simplification.

resolution are usually set equal to one for convenience, and gives us a lattice Boltzmann equation on the form

$$f_i(\mathbf{x}+\mathbf{c}_i, t+1) = -\frac{1}{\tau}[f_i(\mathbf{x},t) - f_i^{\text{eq}}(\mathbf{x},t)] + f_i(\mathbf{x},t). \tag{3.5}$$

The isothermal assumption was only mentioned briefly in passing, but deserves some more explanation. Most lattice Boltzmann simulations only simulate the continuity (2.6a) and momentum (2.6b) equations in the Navier-Stokes-Fourier model, this is also the case with the simulations performed in this thesis since we assume that the gas (air) is isothermal. This means that the equilibrium distribution function f_i^{eq}, given in subsection 3.1.1 no longer conserves energy and effectively acts as a thermostat [25], in other words, the energy equation (2.6c) in the NSF model is not taken into account in the simulations. Strictly speaking, the lattice Boltzmann method used in this thesis should specifically be called the isothermal lattice Boltzmann method, but we will stick to just "lattice Boltzmann method" for convenience.

3.1.1 The equilibrium distribution function and particle velocities

The discrete equilibrium distribution f_i^{eq} used in (3.3), (3.4) and (3.5) is expressed as

$$f_i^{\text{eq}} = \rho w_i \left(1 + \frac{\mathbf{c}_i \cdot \mathbf{u}}{c_{\text{s}}^2} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_{\text{s}}^2} + \frac{(\mathbf{c}_i\cdot\mathbf{u})^2}{2c_{\text{s}}^4}\right), \tag{3.6}$$

with the weights w_i given by

$$w_i = \begin{cases} 4/9, & \text{if} \quad i = 0 \\ 1/9, & \text{if} \quad i = 1,2,3,4 \\ 1/36, & \text{if} \quad i = 5,6,7,8 \end{cases} \tag{3.7}$$

and particle velocities given by

$$\begin{aligned} &\mathbf{c}_1 = [0,0], && \mathbf{c}_2 = [1,0], && \mathbf{c}_3 = [0,1], \\ &\mathbf{c}_3 = [-1,0], && \mathbf{c}_5 = [0,-1], && \mathbf{c}_6 = [1,1], \\ &\mathbf{c}_7 = [-1,1], && \mathbf{c}_8 = [-1,-1], && \mathbf{c}_9 = [1,-1]. \end{aligned}$$

Lastly, c_{s} is the speed of sound in the lattice and has the value

$$c_{\text{s}} = \frac{\Delta x}{\Delta t\sqrt{3}}, \tag{3.8}$$

which reduces to $c_{\text{s}} = 1/\sqrt{3}$, given that we set the space and time resolution equal to one in our simulations. A graphical representation of the $\mathbf{c}_i$ vectors can be found in Figure 3.1. Derivations of different equilibrium functions with according weights and particle velocities can be found in [26].

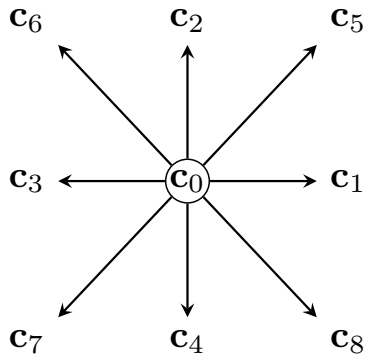


Figure 3.1: A graphical representation of a node and the nine different directions in which particle distribution functions can stream in the D2Q9 model. Note that the distribution function with velocity $\mathbf{c}_0$ always remains stationary in the node.

An interesting fact, but not really relevant to this thesis, is that the choice of the equilibrium distribution function affects which equations can be recovered from the LB equation. With equilibrium distribution function chosen as in (3.6), only the Navier-Stokes equations can be recovered [27]. A more general and powerful approach is to express the equilibrium function as a power series in macroscopic velocity [28]:

$$f_i^{\text{eq}} = A_i + B_i c_{i,\alpha} u_\alpha + C_i c_{i,\alpha} c_{i,\beta} u_\alpha u_\beta + D_i u_\alpha u_\alpha. \tag{3.9}$$

However, using this is not the same as saying that more detailed flow models e.g. super-Burnett model can be recovered from the LB equation, but rather that different types of flow equations can be recovered. An example is given in [27, Ch. 3] where the shallow water equations[7] are recovered.

[7] The shallow water equations describe flow below a pressure surface in a fluid where the horizontal length scale is much greater than the vertical length scale. Often used to model waves on rivers, lakes and oceans.

3.1.2 Recovering the hydrodynamic variables

For the lattice Boltzmann method (LBM) to be useful, we need to extracts some kind of information from it, in our case we are interested in the macroscopic variables ρ and $\mathbf{u}$, which when known allows us to simulate and visualize the change in a fluid's density and velocity over time. These macroscopic variables can be recovered from the LB equation by calculating what is known as moments[8] of the particle distribution function f_i. Noting that certain quantities like mass and momentum must be preserved in order to give a physically meaningful solution, the zeroth moment is given as

$$\rho = \sum_{i=0}^{8} f_i^{\text{eq}} = \sum_{i=0}^{8} f_i. \tag{3.10}$$

This simply states that the sum of each particle distribution in node $\mathbf{x}$ equals the fluid density in that node. The first moment is given as

$$\rho\mathbf{u} = \sum_{i=0}^{8} \mathbf{c}_i f_i,$$

which can be rewritten to give us an expression for the fluid velocity:

$$\mathbf{u} = \frac{1}{\rho} \sum_{i=0}^{8} \mathbf{c}_i f_i. \tag{3.11}$$

Note that the upper and lower bound on the summations is due to the use of a D2Q9 lattice model.

[8] Moments are calculated as an integral over all velocities, with the distribution function $f(\mathbf{x}, t, \boldsymbol{\xi})$ weighted with some function of velocity as the integrand. In the discrete case, the integrals is replaced with a summation sign and the velocities $\boldsymbol{\xi}$ are replaced with a discrete set of velocities $\mathbf{c}_i$. The link provided between the mesoscopic and macroscopic scales by these moments is what allows us to recover the sought after macroscopic variables.

3.2 The lattice Boltzmann method

In a D2Q9 lattice of size $N \times M$ there are a total of $N \cdot M$ nodes, each node containing nine particle distribution functions $f_i(\mathbf{x}, t)$.

During each iteration of the method, a series of steps are performed, a flow chart to help summarize the lattice Boltzmann method is given in Figure 3.2. Note that the first step is to initialize the system

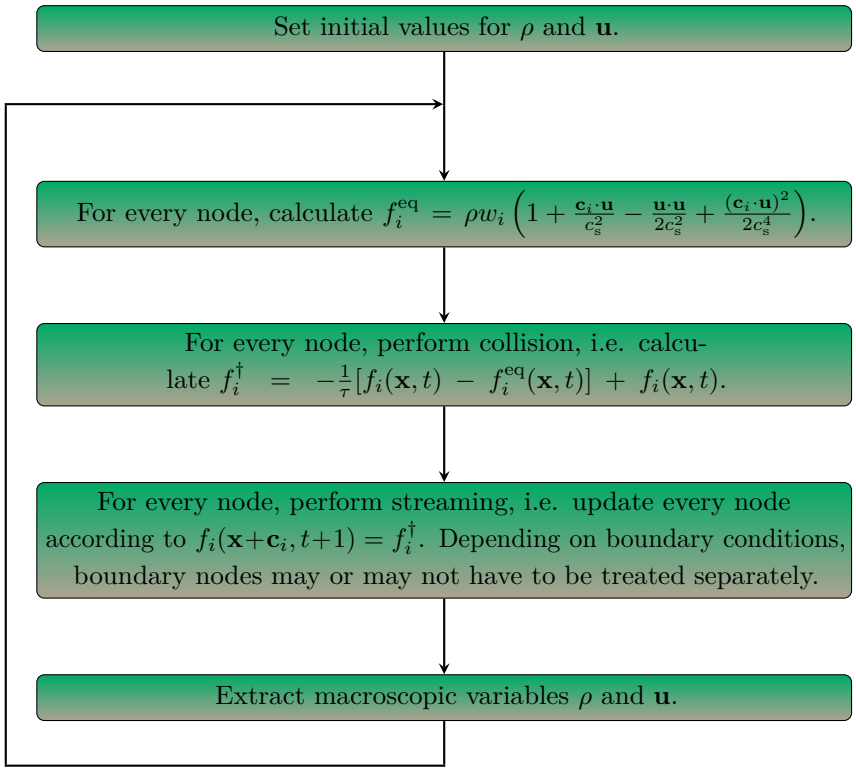


Figure 3.2: Flow chart depicting the different steps performed during each iteration of the LBM with a general boundary condition.

with appropriate values for ρ and $\mathbf{u}$. These values may depend on the problem and it is not always straightforward how to choose the initial values. A very simple, but not necessarily very accurate way is to initialize with $\rho = 1$, $\mathbf{u} = \mathbf{0}$ and $f_i = f_i^{\mathrm{eq}}$ in every node. For some additional information on how to initialize the system, see [23, Ch. 5.5].

3.3 Boundary conditions

When it comes to the nodes lining our computational domain, i.e. the boundary nodes, we encounter a problem. The nodes in our domain receive particle distribution functions who stream in from neighbor nodes. However, when it comes to boundary nodes, they do not have any neighbors outside the domain, furthermore some of the distribution functions found in a boundary node will inevitably stream out of the domain. This is highly undesirable as it would lead

to information loss and inaccurate simulations, to counter this we impose what is called boundary conditions (BCs).

The primary task of a boundary condition is to specify the value of the unknown f_is streaming into the computational domain from imaginary nodes lying outside the domain, and to decide what will happen to f_is streaming out of the domain. The whole streaming process for an arbitrary node is depicted in Figure 3.3 and Figure 3.4.

The choice of BCs greatly depend of the phenomenon being studied, but some common boundary conditions used are

- **Periodic boundary:** Distribution functions streaming out of the domain streams back inside, but on the opposite side.
- **Bounce-back BC:** Distribution functions streaming out of the domain now hits an imaginary wall, and bounce back. The f_is that bounce back stay in the nude, i.e. the do not actually stream anywhere during one iteration, but they bounce back with reversed direction. This is sometimes thought of as an attempt to replicate the physical phenomena of a particle hitting a solid wall, but according to Bennett [29, Ch. 3.1], the idea that a particle bounces back with opposite velocity components does not conform well with reality and is a remnant from the LGA method.
- **Zou-He BC:** The Zou-He BC can be said to apply the bounce-back rule to the non-equilibrium part of the distribution function [30]. Can be used to impose a given density or velocity at the boundary nodes.

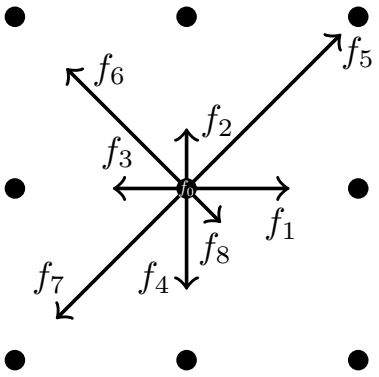


Figure 3.3: A sample of nine nodes from some $N \times M$ sized lattice. The length of the arrows indicate that the different f_is have different values. Distribution functions are only shown for the middle node.

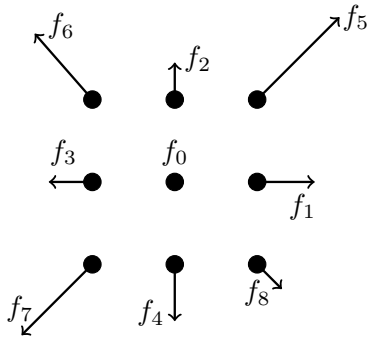


Figure 3.4: The distribution functions have, during one iteration streamed outwards from their origin node, according to their respective $\mathbf{c}_i$s. Only streaming from the center node is depicted.

3.4 The Chapman-Enskog method

In order to verify that the LBM produces behavior according to the compressible Navier-Stokes equation (2.6b), one can expand the left hand side of the LB equation

$$f_i(\mathbf{x} + \mathbf{c}_i, t + 1) - f_i(\mathbf{x}, t) = \Omega_i(f), \tag{3.12}$$

where $\Omega_i(f)$is a generic collision operator, into a second-order Taylor series:

$$\left(\frac{\partial}{\partial t} + \nabla \cdot \mathbf{c}_i\right) f_i + \frac{1}{2}\left(\frac{\partial^2}{\partial t^2} + 2\frac{\partial}{\partial t}\nabla \cdot \mathbf{c}_i + \frac{\partial}{\partial x_\alpha}\frac{\partial}{\partial x_\beta} c_{i,\alpha} c_{i,\beta}\right) f_i \approx \Omega_i(f)$$

This is followed by what is normally called a multi-scale Chapman-Enskog expansion. This procedure is quite complex and have been performed by others and will therefore not be repeated here. However, the basic steps of the Chapman-Enskog (CE) analysis will be presented to give the reader some idea as to how the Navier-Stokes equations can be derived from the LBM. Besides from the original articles by Chapman [31, 32] and Enskog [33, 34] on the theory behind

what would later become the CE expansion a more modern introduction is given by Struchtrup in [18, Ch. 4]. For a very detailed and thorough derivation of compressible Navier-Stokes from the LBM, see [10, Appendix A] by Viggen.

During a CE multi-scale expansion the functions $\Omega_i(f)$, f_i and the operators $\partial/\partial t$ and ∇ are expanded in terms of what is knows as a smallness parameter ϵ, which turns out to be the Knudsen number Kn. This results in a separation of time scales, terms in different orders of ϵ are collected and different physical phenomenon at different time scales can then be studied separately. In addition, the different equations in order of ϵ contribute individually to the equations of motion. If we pair this with the fact that macroscopic variables are given by moments of the particle distribution function, and that certain conservation laws must hold, we eventually end up with compressible Navier-Stokes from the LB equations.

3.5 Lattice units

In contrast with real life problems, which are usually given in a system of metric units, LB simulations "live in a system of lattice units". Therefore, in order to go from lattice units, hereby denoted with the subscript la, to physical units, denoted with subscript ph, we have to carry out some kind of conversion. There are more than one way to convert from physical units to lattice units and vice versa and the method used here is based on the approach taken in [20].

The time and space resolution Δt and Δx are what connects lattice units and physical units. As an example, the physical speed is connected to the lattice speed through

$$u_{\text{ph}} = \frac{\Delta x}{\Delta t} u_{\text{la}}. \tag{3.13}$$

As we know, speed has the unit $\mathrm{m\,s^{-1}}$, the right hand side of the equation above must therefore have the same dimension, this is achieved if Δx is measured in seconds s, and Δt is measured in meters m. However, we still need to know the value of the conversion factors Δx and Δt. Turns out there are two constraints on our system, the speed of sound (which is constant for our purposes) given by

$$c_{\text{s,ph}} = \frac{\Delta x}{\Delta t} c_{\text{s,la}}, \tag{3.14}$$

and viscosity given by

$$\nu_{\text{ph}} = \frac{\Delta x^2}{\Delta t} \nu_{\text{la}}, \tag{3.15}$$

where $c_{\text{s,la}=1/\sqrt{(3)}}$. Using these two equations, we can solve for Δx and

Δt and get

$$\Delta x = \frac{\nu_{\text{ph}}}{\nu_{\text{la}}} \frac{c_{\text{s,la}}}{c_{\text{s,ph}}}, \tag{3.16}$$

$$\Delta t = \frac{\nu_{\text{ph}}}{\nu_{\text{la}}} \left(\frac{c_{\text{s,la}}}{c_{\text{s,ph}}} \right)^2. \tag{3.17}$$

In addition, we have a free parameter τ through

$$\nu_{\text{la}} = \left(\tau - \frac{1}{2} \right) c_{\text{s,la}}^2. \tag{3.18}$$

Inserting (3.18) in e.g. (3.15) gives

$$\nu_{\text{ph}} = \frac{\Delta x^2}{\Delta t} \left(\tau - \frac{1}{2} \right) c_{\text{s,la}}^2. \tag{3.19}$$

The left side of (3.19) is a constant, and therefore the right side must also be a constant, from this we can conclude that adjusting τ allows us to either increase or decrease Δx or Δt by some amount. Note that $\tau > 0.5$ as $\tau \leq 0.5$ gives zero or negative viscosity, resulting in non-physical fluids[9]. In addition numerical instabilities arise when $\tau \to 0.5$, according to Sukop and Thorne [35, Ch. 4.3] $\tau = 1$ is the safest choice.

[9] Baring any quantum mechanical phenomenon like for example the superfluidity phenomenon.

To summarize, an example of the steps taken to convert from values produced by the LBM, to physical values with physical units, is given below:

1. Choose a $\tau > 0.5$.
2. Calculate viscosity in the lattice system from (3.18).
3. Calculate the value for Δt from (3.17).
4. Calculate the value for Δx from (3.15).

Lastly, as mentioned by Viggen in [10], simulation of acoustic waves in air yields extremely small values for Δx and Δt, rendering LB simulations for acoustics in air somewhat troublesome.

4 Non-reflecting boundary conditions

This chapter presents three different non-reflecting boundary conditions (NRBCs). A short background history is given as well as a detailed mathematical treatment of the different NRBCs. Each section is based on the original article(s) where these particular types of NRBCs originated. In keeping with tone of the primary goals of this thesis, equations are expanded on and explanations are provided to complement the original articles.

First of all we establish why non-reflecting boundary conditions are worth investigating: Physical wave phenomena often take place in unbounded domains. Such unbounded domains are mostly found in nature where the distance between the wave source and possible reflective obstacles is so great that for all intents and purposes, they can be regarded as being infinitely far away (and thus not reflecting). For obvious reasons it is not feasible to implement an infinitely large domain on a computer. Thus, paradoxically, to simulate an unbounded domain we have restrict the domain by imposing some kind of boundaries. This is where non-reflecting boundary conditions comes in. The aim of a NRBC is to reduce the reflection of waves from the boundaries and into the computational domain as much as possible. Thus allowing us to study the simulation without unwanted interference from the boundaries.

Note that even though the chapter title, indeed even the thesis title, contains the words non-reflecting boundary conditions, the material presented in section 4.2 and section 4.3 cannot, technically speaking, be said to describe a NRBC. They are instead what is called absorbing layers treatments. Only section 4.1 presents a "true" NRBC. The term NRBC are therefore used more like an umbrella term.

4.1 Characteristic boundary conditions

This section is based on the article by Heubes, Bartel and Ehrhardt [36].

4.1.1 Introduction and mathematical treatment

Characteristic boundary conditions (CBSs) are, as the name implies, developed with the method of characteristics. Non-reflecting characteristic boundary conditions for nonlinear hyperbolic equations[1] were first developed by Thompson [37] and Hedstrom [38].

[1] Hyperbolic partial differential equations have "wave-like" solutions, which means that disturbances in the initial data are not felt everywhere at once, instead they propagate with finite speed.

The method of characteristics is only valid for hyperbolic partial differential equations (PDEs), which presents us with a problem since only the continuity equation (2.6a) is hyperbolic, the momentum (2.6b) and energy equation (2.6b) in the NSF model are parabolic PDEs [39]. This problem is circumvented by setting the Laplacian terms in (2.10) equal to zero, turning it into the momentum equation in the Euler model. In addition, the energy equation in the NSF model is dropped. As a sort of justification for dropping the energy equation, remember that the energy equation is not actually simulated by the LBM used in this thesis, as per the isothermal assumption (section 3.1). After this, we are left with a hyperbolic PDE which can be written as

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{U}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{U}}{\partial y} = 0, \tag{4.1}$$

where $\mathbf{U} = [\rho, u_x, u_y]^\intercal$ is a vector of the characteristic variables and $\mathbf{A}$ and $\mathbf{B}$ are coefficient matrices given by

$$\mathbf{A}(\rho, u_x, u_y) = \begin{bmatrix} u_x & \rho & 0 \\ \frac{c_\mathrm{s}^2}{\rho} & u_x & 0 \\ 0 & 0 & u_x \end{bmatrix}, \quad \mathbf{B}(\rho, u_x, u_y) = \begin{bmatrix} u_y & 0 & \rho \\ 0 & u_y & 0 \\ \frac{c_\mathrm{s}^2}{\rho} & 0 & u_y \end{bmatrix}, \tag{4.2}$$

where u_x and u_y are the velocities in x and y direction. Since (4.1) is a hyperbolic equation, the coefficient matrices in (4.2) are diagonalizable [40], and we can represent them as

$$\mathbf{SAS}^{-1} = \mathbf{\Lambda}, \quad \mathbf{TBT}^{-1} = \mathbf{M}, \tag{4.3}$$

with eigenvalue matrices

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3) = \mathrm{diag}(u_x - c_\mathrm{s}, u_x, u_x + c_\mathrm{s}) \text{ and} \tag{4.4}$$

$$M = \mathrm{diag}(\mu_1, \mu_2, \mu_3) = \mathrm{diag}(u_y - c_\mathrm{s}, u_y, u_y + c_\mathrm{s}). \tag{4.5}$$

The matrices $\mathbf{S}$ and $\mathbf{T}$ used in [36] are given as

$$\mathbf{S} = \begin{bmatrix} c_\mathrm{s}^2 & -c_\mathrm{s}\rho & 0 \\ 0 & 0 & 1 \\ c_\mathrm{s}^2 & c_\mathrm{s}\rho & 0 \end{bmatrix}, \qquad \mathbf{S^{-1}} = \begin{bmatrix} \frac{1}{2c_\mathrm{s}^2} & 0 & \frac{1}{2c_\mathrm{s}^2} \\ -\frac{1}{2c_\mathrm{s}\rho} & 0 & \frac{1}{2c_\mathrm{s}\rho} \\ 0 & 1 & 0 \end{bmatrix},$$

$$\mathbf{T} = \begin{bmatrix} c_\mathrm{s}^2 & 0 & -c_\mathrm{s}\rho \\ 0 & 1 & 0 \\ c_\mathrm{s}^2 & 0 & c_\mathrm{s}\rho \end{bmatrix}, \qquad \mathbf{T^{-1}} = \begin{bmatrix} \frac{1}{2c_\mathrm{s}^2} & 0 & \frac{1}{2c_\mathrm{s}^2} \\ 0 & 1 & 0 \\ -\frac{1}{2c_\mathrm{s}\rho} & 0 & \frac{1}{2c_\mathrm{s}\rho} \end{bmatrix}.$$

With this we differentiate between ingoing and outgoing waves by simply inspecting the eigenvalues and eigenvectors. A positive eigenvalue λ_i indicates a wave traveling in positive x-direction, a positive eigenvalue μ_i indicates a wave traveling in positive y-direction and vice versa. This is very convenient as we encounter a problem at the boundaries since we usually have no information on the incoming wave, i.e. we do not know anything about waves coming from outside the boundaries of our system. However, since we would like non-reflecting boundaries, this problem is solved simply by setting the incoming wave equal to zero. This will be shown in greater mathematical detail later on.

Originally, two different NRBCs were investigated in [36], Thompson's NRBC by Thompson [37] and a NRBC developed by Izquierdo and Fueyo [41] based on the local one-dimensional inviscid equation (LODI). They found that these two approaches either overestimated or underestimated both the velocity tangential to the boundary, and the mass density. Motivated by this, Heubes et al. constructed what they call a modified Thompson boundary condition, whose solution can be interpreted as a convex combination of the solutions of the Thompson and LODI approach. The modified Thompson BCs are among the NRBCs tested in this thesis and will be referred to later simply as CBCs.

By using the definitions in (4.3) we write

$$\mathbf{A}\frac{\partial \mathbf{U}}{\partial x} = \mathbf{S}^{-1}\mathbf{\Lambda}\mathbf{S}\frac{\partial \mathbf{U}}{\partial x} = \mathbf{S}^{-1}\boldsymbol{\mathcal{L}}_x \quad \text{and} \quad \mathbf{B}\frac{\partial \mathbf{U}}{\partial y} = \mathbf{T}^{-1}\mathbf{M}\mathbf{T}\frac{\partial \mathbf{U}}{\partial y} = \mathbf{T}^{-1}\boldsymbol{\mathcal{L}}_y,$$

where

$$\boldsymbol{\mathcal{L}}_x = \begin{bmatrix} \mathcal{L}_{x,1} \\ \mathcal{L}_{x,2} \\ \mathcal{L}_{x,3} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\mathcal{L}}_y = \begin{bmatrix} \mathcal{L}_{y,1} \\ \mathcal{L}_{y,2} \\ \mathcal{L}_{y,3} \end{bmatrix}. \tag{4.6}$$

This is called expressing the x- and y-derivatives in characteristics. The different $\mathcal{L}_{x,i}$s and $\mathcal{L}_{y,i}$s express the amplitude variations of the characteristic waves, and by following the approach of Hedstrom and Thompson, they can be expressed as

$$\tilde{\mathcal{L}}_{x,i} = \begin{cases} \lambda_i \boldsymbol{\ell}_i^\intercal \frac{\partial \mathbf{U}}{\partial x} & \text{for an outing wave,} \\ 0 & \text{for an incoming wave.} \end{cases} \tag{4.7}$$

$$\tilde{\mathcal{L}}_{y,i} = \begin{cases} \mu_i \mathbf{m}_i^\intercal \frac{\partial \mathbf{U}}{\partial x} & \text{for an outing wave,} \\ 0 & \text{for an incoming wave,} \end{cases} \tag{4.8}$$

Where $\boldsymbol{\ell}_i^\intercal$ and $\mathbf{m}_i^\intercal$ denotes the ith row of $\mathbf{S}$ and $\mathbf{T}$ respectively and the tilde ($\tilde{}$) indicates that we have explicitly annihilated the incoming wave.

With this we can finally express the modified Thompson BCs for an x-boundary[2]

$$\frac{\partial \mathbf{U}}{\partial t} = -\mathbf{S}^{-1}\widetilde{\boldsymbol{\mathcal{L}}}_x - \gamma \mathbf{B}\frac{\partial \mathbf{U}}{\partial y}. \tag{4.9}$$

where $\gamma \in [0,1]$ is a parameter which decides the convex combination of solutions, demonstrated by the fact that the solution of (4.9) can be seen as the convex combination of the solution of the Thompson BC and the LODI BC:

$$\mathbf{U}_{\mathrm{mT}} = \gamma \mathbf{U}_{\mathrm{Thompson}} + (1-\gamma)\mathbf{U}_{\mathrm{LODI}}. \tag{4.10}$$

Lastly, we deal with the corners, the corners require special attention as a characteristic analysis has to be done in both x- and y-direction. We start by writing both the x- and the y-derivatives in (4.1) in characteristics, and using the definitions in (4.7) and (4.8), we get

$$\frac{\partial \mathbf{U}}{\partial t} = -\mathbf{S}^{-1}\widetilde{\boldsymbol{\mathcal{L}}}_x - \mathbf{T}^{-1}\widetilde{\boldsymbol{\mathcal{L}}}_y. \tag{4.11}$$

4.1.2 Implementation of Characteristic boundary conditions for the LBM

Assuming we have a 2D computational domain, we would now like to solve (4.9) at every boundary node for every iteration of our lattice Boltzmann algorithm. By solving (4.9) for each iteration k in our LBM, we are given ρ, u_x and u_y at time instant k.

Step 1, computing the wave amplitude variation: It is still assumed that we are looking at an x-boundary, the steps for the y boundary are for the most part similar, but slight differences will be emphasized. We start by writing out (4.7) which gives us

$$\tilde{\mathcal{L}}_{x,1} = \begin{cases} (u_x - c_\mathrm{s})\left[c_\mathrm{s}^2\frac{\partial \rho}{\partial x} - c_\mathrm{s}\rho\frac{\partial u_x}{\partial x}\right] & \text{for an outgoing wave,} \\ 0 & \text{for an incoming wave,} \end{cases}$$

$$\tilde{\mathcal{L}}_{x,2} = \begin{cases} u_x\frac{\partial u_y}{\partial x} & \text{for an outgoing wave,} \\ 0 & \text{for an incoming wave,} \end{cases}$$

$$\tilde{\mathcal{L}}_{x,3} = \begin{cases} (u_x + c_\mathrm{s})\left[c_\mathrm{s}^2\frac{\partial \rho}{\partial x} + c_\mathrm{s}\rho\frac{\partial u_x}{\partial x}\right] & \text{for an outgoing wave,} \\ 0 & \text{for an incoming wave.} \end{cases}$$

In the case of $\widetilde{\boldsymbol{\mathcal{L}}}_y$, its components have the exact same form except that $u_x \to u_y$, $\frac{\partial}{\partial x} \to \frac{\partial}{\partial y}$ and $u_y \to u_x$.

As the lattice Boltzmann is a numerical method, we need to discretize the spatial derivatives found in $\widetilde{\boldsymbol{\mathcal{L}}}_x$ and $\widetilde{\boldsymbol{\mathcal{L}}}_y$. The derivatives found therein are perpendicular to the boundary and since no nodes

[2] Note that by x-boundary, a boundary with x = const is implied, in other words, an x-boundary corresponds to the west and east boundaries of a square, free of any internal obstacles.

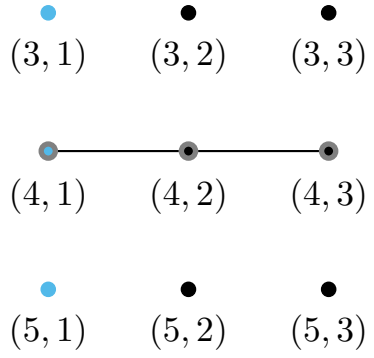


Figure 4.1: The numerical stencil on the west boundary.

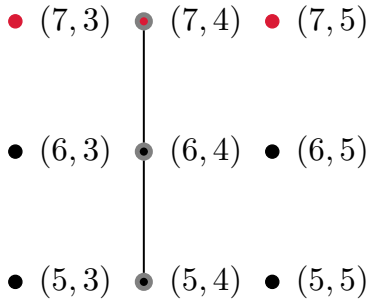


Figure 4.2: The numerical stencil on the north boundary.

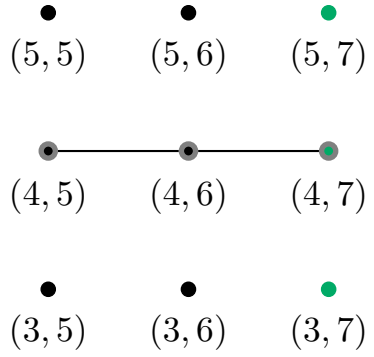
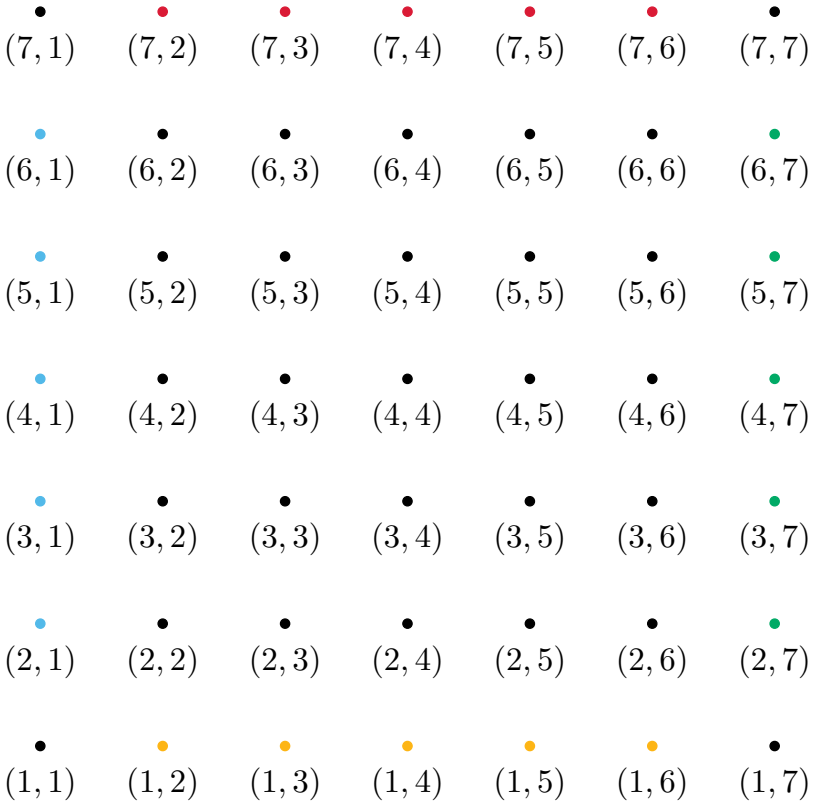


Figure 4.3: The numerical stencil on the east boundary.

are available outside the computational domain, the following one-sided second order finite difference (FD) scheme is a good choice:

$$\left.\frac{\partial \lambda}{\partial x}\right|_{x=x_i} = \frac{1}{2}\left(-3\lambda([x_i, y], t) + 4\lambda([x_{i+1}, y], t) - \lambda([x_{i+2}, y], t)\right), \quad (4.12)$$

where $\lambda(\mathbf{x}, t)$ is a general quantity, and x_i indicates the we are looking at the ith border node on an x-boundary in the lattice. Assuming our lattice is given as in figure 4.5, the finite difference stencils for the different boundaries are shown in figure 4.1, 4.2, 4.3 and 4.4.



As can be seen from (4.11), the equation to be solved for the corners is discretized by simply using a combination of difference stencils, i.e. the north-west corner will use stencils for the west and the north boundary etc. By discretizing the different $\mathcal{L}_{x,i}$s and $\mathcal{L}_{y,i}$s, the resulting characteristics can be written as

$$-\mathbf{S}^{-1}\tilde{\boldsymbol{\mathcal{L}}}_x^{\mathrm{FD}} = \begin{bmatrix} -\frac{1}{2c_s^2}\left(\tilde{\mathcal{L}}_{x,1}^{\mathrm{FD}} + \tilde{\mathcal{L}}_{x,3}^{\mathrm{FD}}\right) \\ \frac{1}{2c_s\rho}\left(\tilde{\mathcal{L}}_{x,1}^{\mathrm{FD}} - \tilde{\mathcal{L}}_{x,3}^{\mathrm{FD}}\right) \\ -\tilde{\mathcal{L}}_{x,2}^{\mathrm{FD}} \end{bmatrix}, \quad (4.13)$$

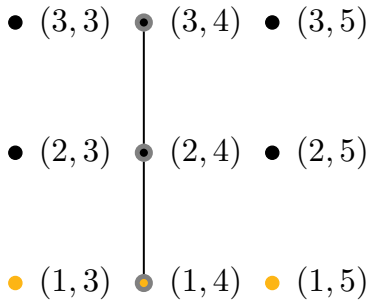


Figure 4.4: The numerical stencil on the south boundary.

Figure 4.5: An example of a 7×7 lattice.

where the superscript FD indicates that the spatial discretization is done with a finite difference scheme. The form of $\mathbf{S}^{-1}\widetilde{\boldsymbol{\mathcal{L}}}_y^{\mathrm{FD}}$ is the same except that $\tilde{\mathcal{L}}_{y,i}^{\mathrm{FD}}$ is used instead.

Step2, time integration at the boundary: Having discretized the wave amplitude variations in the previous step, we now have to discretize the spatial derivatives found in (4.9). These derivatives run parallel to the boundary in question and are discretized with the following second-order central finite difference scheme:

$$\left.\frac{\partial \lambda}{\partial y}\right|_{y=y_i} = \frac{1}{2}\left(\lambda([x, y_{i+1}], t) - \lambda([x, y_{i-1}], t)\right). \tag{4.14}$$

Finally, in order to solve (4.9) on the boundary nodes and (4.11) on the corner nodes, we have to perform some sort of numerical integration. For this task we use the very simple explicit Euler method[3]. By using symbols and notation already familiar to us, the explicit Euler can be written as

$$\mathbf{U}_{n+1} = \mathbf{U}_n + h\mathbf{F}(\mathbf{U}_n, t_n). \tag{4.15}$$

[3] The explicit Euler, or forward Euler as it is sometimes called, is a variation of the Euler method where only previous values of the function to be integrated are used. It is first order accurate.

Put in context, $\mathbf{U}_{n+1}$ represents the values of ρ, u_x and u_y that we need during the current streaming step of the LBM, $\mathbf{U}_n$ are values that are available from the previous iteration of the LBM and $\mathbf{F}(\mathbf{U}_n, t_n)$ represents the function we are integrating, expressed with values from the previous iteration step. Note also that the step size h is set to one, and that (4.15) is used only once for each boundary node during one iteration of the LBM.

Step 3, transfer of variables to the LBM: When it comes to streaming in the boundary nodes, there is no place for particle distributions f_i streaming out of the computational domain to stream to, neither are there any particle distributions streaming into the computational domain from the outside. This presents us with a problem as we need to compute the $f_i(\mathbf{x}, t)$s on the boundary. One simple way to solve this is to use equilibrium boundary conditions (EBCs). We are now operating with two boundary conditions, CBCs and EBCs. The CBC is a way to impose a Dirichlet condition[4] on the boundaries, therefore giving us the values of ρ, u_x and u_y at the boundary nodes. These values are then carried over to the LBM, where the EBCs ensures that missing populations[5] are found in such a way that desired macroscopic behavior is approximated.

[4] Dirichlet conditions are conditions that are imposed on ordinary or partial differential equations, and specifies the values that a solution needs to take on the boundary.

[5] As streaming takes place in the boundary nodes, particle distributions streaming out of the computational domain would normally be lost, and distributions that were supposed to stream into the computational domain from the outside are missing.

A simple way to "find" missing particle distribution is to use the aforementioned equilibrium boundary conditions, we denote the boundary lattice points as $\mathbf{x}_{\mathrm{B}}$, and ρ_{D} and $\mathbf{u}_{\mathrm{D}} = [u_{x,\mathrm{D}}, u_{y,\mathrm{D}}]^{\intercal}$ as the values of $\mathbf{U}$ on the boundary.

$$f_i(\mathbf{x}_{\mathrm{B}}, t) = f_i^{(\mathrm{eq})}(\mathbf{x}_{\mathrm{B}}, t) = w_i \rho_{\mathrm{D}} \left[1 + \frac{\mathbf{c}_i \cdot \mathbf{u}_{\mathrm{D}}(\mathbf{x}_{\mathrm{B}})}{c_{\mathrm{s}}^2} + \frac{(\mathbf{c}_i \cdot \mathbf{u}_{\mathrm{D}})^2}{2c_{\mathrm{s}}^4} - \frac{\mathbf{u}_{\mathrm{D}} \cdot \mathbf{u}_{\mathrm{D}}}{2c_{\mathrm{s}}^2}\right].$$

4.1.3 Flowchart for LBM with CBCs

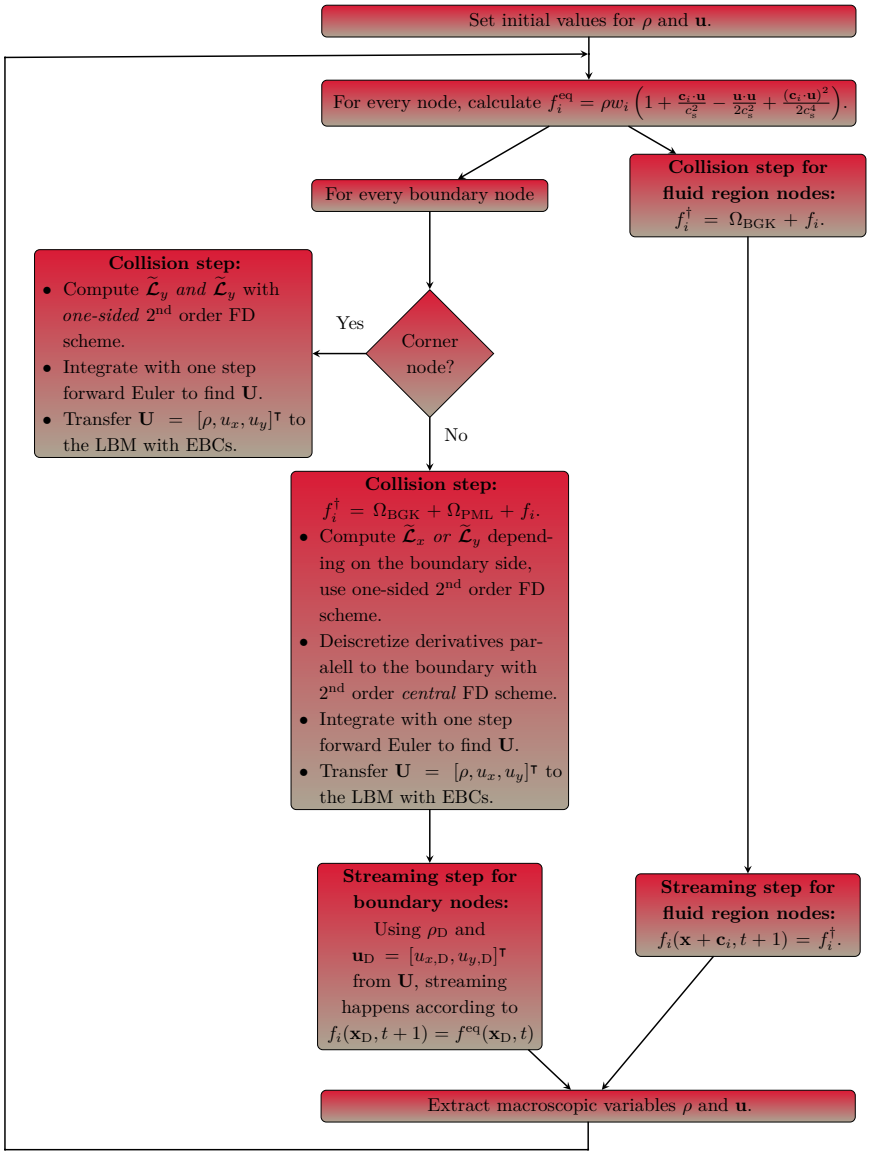


Figure 4.6: A flowchart for the LBM with CBCs outlining the key steps in the algorithm.

4.2 Sponge layer

This section is based on the articles [42, 43] by Vergnault, Malaspinas and Sagaut.

This is the first of two absorbing layer techniques reviewed in this thesis, the other can be found in section 4.3. Absorbing layers may or may not be used in combination with absorbing boundary conditions [44], in this thesis the simple mid-way bounce back (not absorbing) boundary condition is used.

4.2.1 Sponge layer, mode of operation

The idea behind sponge layers is quite simple, increasing viscosity is artificially introduced in the sponge layers so that the wave entering such a layer will experience damping. The viscosity is introduced through the relaxation frequency $\omega = 1/\tau$ which should vary quadratically across the sponge layers, in a direction perpendicular to the boundary. This relaxation frequency for the sponge layers is given as

$$\omega_{\text{sponge}} = \frac{1 - 0.999d^2}{3\nu + 0.5} \qquad \text{for} \qquad d \leq 1, \tag{4.16}$$

$$\omega_{\text{sponge}} = \frac{0.001}{3\nu + 0.5} \qquad \text{for} \qquad d > 1, \tag{4.17}$$

Although this does not conform with theory, e.g. the Chapman-Enskog expansion is no longer valid for very small values of ω, it is argued that there is no need to match the physics in the sponge layer to the real world [43].

The distance d mentioned in (4.16) needs some additional discussion. It is not clear to the author of this thesis if d is the distance in number of nodes, or in some other metric. If the distance is measured in nodes, (4.16) reduces to

$$\omega_{\text{sponge}} = \frac{0.001}{3\nu + 0.5} \tag{4.18}$$

and we get a very abrupt change in viscosity in the interface between the fluid region and the sponge layer. To investigate the effect of such an abrupt change, different variation profiles for ω_{sponge} were tested in a sponge layer of thickness $d = 10$, the profiles tested can be found in Figure 4.7

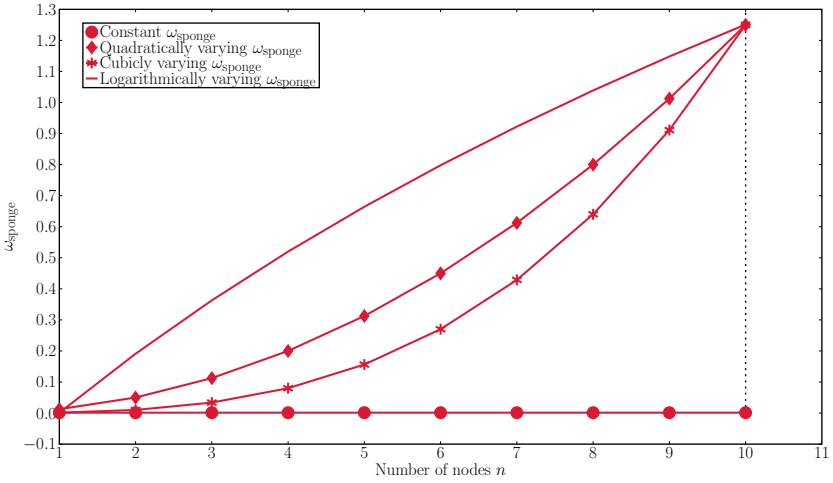


Figure 4.7: The variation profile of different ω_{sponge} tested. From a cross sectional point of view, the vertical dashed line indicates the end of the sponge layer, i.e. sponge layer on the left, fluid region on the right, in the fluid region, $\omega_{\text{sponge}} = \omega$. As a final note, although it cannot be seen from the graph the lower limit for every ω_{sponge} tested was ≈ 0.001.

From the test, it was concluded that a logarithmically varying ω_{sponge} gave the least errors and this profile is used from this point forward, it is given by the discrete function

$$\omega_{\text{sponge}}[n] = \frac{\omega}{\log(2)} \log\left(0.9999 + \frac{2 - 0.9999}{d-1}(n-1)\right) + 0.0012 \quad \text{for } n = [1, 2, \ldots, d].$$

4.3 Perfectly matched layer

This section is based on the article by Najafi-Yazdi and Mongeau [45]. Both two and three dimensional cases are mentioned in the article, but we will only look at the two dimensional version in this thesis.

4.3.1 Short introduction to perfectly matched layers

The perfectly matching layer (PML) technique was first introduced by Berenger to be used on Maxwell's electromagnetic equations [46].

Similar to the sponge layer method in section 4.2, the PML method adds an absorbing layer to the computational domain, as seen in Figure 4.8.

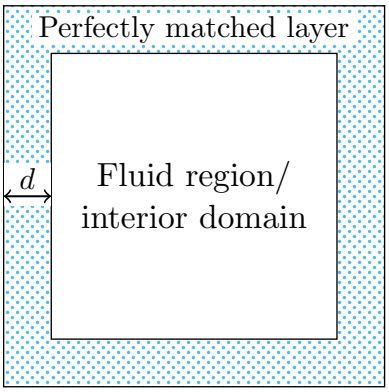


Figure 4.8: The figure is showing the entire computational domain, the hatched area represents a perfectly matching layer with thickness d.

However, unlike the sponge layer method, the governing equations of a PML is, as the name implies, matched perfectly to the governing equations of the interior domain. This results in a reflection free interface between the interior domain and the PML, contrary to what can be observed in the sponge layer method where difference in dispersion properties causes some reflection at the interface of the interior domain and the sponge layer.

To derive the equation for a PML, we start with the general equation for a linear hyperbolic PDE in two dimensions:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{U}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{U}}{\partial y} = 0, \tag{4.19}$$

where $\mathbf{U}$ is a $m \times 1$ vector and $\mathbf{A}$ and $\mathbf{B}$ are $m \times m$ matrices. By decomposing (4.19) into a system of equations, one for every spatial dimension, introducing auxiliary variables, writing the system of equations in the frequency domain to allow algebraic manipulation, and then transforming back to time domain (the derivation can be found in [46]), gives us

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{U}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{U}}{\partial y} = -(\sigma_x + \sigma_y)\mathbf{U} - \sigma_x\sigma_y\mathbf{Q} - \sigma_y\mathbf{A}\frac{\partial \mathbf{Q}}{\partial x} - \sigma_x\mathbf{B}\frac{\partial \mathbf{Q}}{\partial y},$$

where $\mathbf{Q}$ is a $m \times 1$ vector and is given as $\partial\mathbf{Q}/\partial t = \mathbf{U}$, and σ_x and σ_y are positive damping coefficients which control the rate of decay of the waves entering the PML. Based on [47] by Bécache, Fauqueux and Joly, who studied the stability of PMLs, the authors of [45] conclude

that $\sigma_x = \sigma_y = \sigma$ in order to overcome instability problems in certain lattice directions, ergo, we will only operate with one damping coefficient henceforth, and the equation above reduces to

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{U}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{U}}{\partial y} = -2\sigma\mathbf{U} - \sigma^2\mathbf{Q} - \sigma\left(\mathbf{A}\frac{\partial \mathbf{Q}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{Q}}{\partial y}\right). \quad (4.20)$$

4.3.2 Perfectly matched layers in the LBM

To incorporate PMLs for the LBM we rewrite the discrete velocity Boltzmann equation (3.3), into its more general linear hyperbolic form

$$\frac{\partial \mathbf{f}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{f}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{f}}{\partial y} = -\frac{1}{\tau}(\mathbf{f} - \mathbf{f}^{\mathrm{eq}}), \quad (4.21)$$

where $\mathbf{A}$ and $\mathbf{B}$ contains elements of $\mathbf{c}_i$ in the following way

$$\mathbf{A} = \begin{bmatrix} c_{0,\alpha} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_{8,\alpha} \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} c_{0,\beta} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_{8,\beta} \end{bmatrix}. \quad (4.22)$$

In addition, instead of writing f_i we now write $\mathbf{f}$ where $\mathbf{f} = [f_0, f_1, \ldots, f_8]^\intercal$. Next we state that $\mathbf{f}^{\mathrm{eq}}$ can be written as

$$\mathbf{f}^{\mathrm{eq}} = \bar{\mathbf{f}}^{\mathrm{eq}} + \tilde{\mathbf{f}}^{\mathrm{eq}}. \quad (4.23)$$

Here, $\bar{\mathbf{f}}^{\mathrm{eq}}$ is the mean uniform component and $\tilde{\mathbf{f}}^{\mathrm{eq}}$ is called the acoustic perturbation component. The mean uniform component is constant and is calculated once from (3.6), where the initial values of ρ and $\mathbf{u}$, mentioned in section 3.2, are used. The acoustic perturbation component, is a constantly varying component as it describes the variation from the mean, as $\mathbf{f}$ and $\bar{\mathbf{f}}^{\mathrm{eq}}$ are available to us in every iteration of the LBM, we can easily compute the perturbation component as

$$\tilde{\mathbf{f}}^{\mathrm{eq}} = \mathbf{f}^{\mathrm{eq}} - \bar{\mathbf{f}}^{\mathrm{eq}}. \quad (4.24)$$

Replacing $\mathbf{f}$ with $\mathbf{f}^{\mathrm{eq}}$ in (4.21) produces

$$\frac{\partial \mathbf{f}^{\mathrm{eq}}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{f}^{\mathrm{eq}}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{f}^{\mathrm{eq}}}{\partial y} = 0. \quad (4.25)$$

Using the decomposition from (4.23), (4.25) turns into

$$\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial t} + \mathbf{A}\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial x} + \mathbf{B}\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial y} = 0. \quad (4.26)$$

Note that that the partial derivative of $\bar{\mathbf{f}}^{\mathrm{eq}}$ equals zero since this is a constant term. As (4.26) has the same form as (4.19), we can apply the PML framework and end up with

$$\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial t} + \mathbf{A}\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial x} + \mathbf{B}\frac{\partial \tilde{\mathbf{f}}^{\mathrm{eq}}}{\partial y} = -\sigma\left(2\tilde{\mathbf{f}}^{\mathrm{eq}} + \sigma\mathbf{Q} + \mathbf{A}\frac{\partial \mathbf{Q}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{Q}}{\partial y}\right), \quad (4.27)$$

where

$$\frac{\partial \mathbf{Q}}{\partial t} = \tilde{\mathbf{f}}^{\text{eq}}.$$

Next we note that the distribution function $\mathbf{f}$ is the sum of its equilibrium function $\mathbf{f}^{\text{eq}}$ and its non-equilibrium function $\mathbf{f}^{\text{neq}}$, which gives us

$$\begin{aligned} \mathbf{f} &= \mathbf{f}^{\text{eq}} + \mathbf{f}^{\text{neq}} && (4.28)\\ &= \tilde{\mathbf{f}}^{\text{eq}} + \bar{\mathbf{f}}^{\text{eq}} + \mathbf{f}^{\text{neq}}. && (4.29) \end{aligned}$$

Inserting (4.28) into (4.21) gives the evolution equation[6] for the non-equilibrium distribution

$$\frac{\partial \mathbf{f}^{\text{neq}}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{f}^{\text{neq}}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{f}^{\text{neq}}}{\partial y} = -\frac{1}{\tau}\mathbf{f}^{\text{neq}}. \tag{4.30}$$

Ultimately, we want an equation on the form of (4.21) with the addition of an additional collision term, hereby called Ω_{PML}, which incorporates the principles of PMLs. As is evident from (4.29), $\mathbf{f}$ is the sum of three terms, we now have equations for these three terms, therefore the idea is to sum them up to obtain the sought after equation. To provide some overview, the three equations are presented individually before they are summed up:

$$\begin{aligned} &\frac{\partial \tilde{\mathbf{f}}^{\text{eq}}}{\partial t} + \mathbf{A}\frac{\partial \tilde{\mathbf{f}}^{\text{eq}}}{\partial x} + \mathbf{B}\frac{\partial \tilde{\mathbf{f}}^{\text{eq}}}{\partial y} = -\sigma\left[2\tilde{\mathbf{f}}^{\text{eq}} + \sigma\mathbf{Q} + \mathbf{A}\frac{\partial \mathbf{Q}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{Q}}{\partial y}\right] = \Omega_{\text{PML}},\\ &\frac{\partial \bar{\mathbf{f}}^{\text{eq}}}{\partial t} + \mathbf{A}\frac{\partial \bar{\mathbf{f}}^{\text{eq}}}{\partial x} + \mathbf{B}\frac{\partial \bar{\mathbf{f}}^{\text{eq}}}{\partial y} = 0,\\ &\frac{\partial \mathbf{f}^{\text{neq}}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{f}^{\text{neq}}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{f}^{\text{neq}}}{\partial y} = -\frac{1}{\tau}\mathbf{f}^{\text{neq}} \overset{(4.28)}{=} -\frac{1}{\tau}(\mathbf{f} - \mathbf{f}^{\text{eq}}) = \Omega_{\text{BGK}}. \end{aligned}$$

Summation of these three yields

$$\frac{\partial}{\partial t}(\tilde{\mathbf{f}}^{\text{eq}} + \bar{\mathbf{f}}^{\text{eq}} + \mathbf{f}^{\text{neq}}) + \mathbf{A}\frac{\partial}{\partial x}(\tilde{\mathbf{f}}^{\text{eq}} + \bar{\mathbf{f}}^{\text{eq}} + \mathbf{f}^{\text{neq}}) + \mathbf{B}\frac{\partial}{\partial y}(\tilde{\mathbf{f}}^{\text{eq}} + \bar{\mathbf{f}}^{\text{eq}} + \mathbf{f}^{\text{neq}}) = \Omega_{\text{PML}} + \Omega_{\text{BGK}}$$

which, by (4.29) reduces to

$$\frac{\partial \mathbf{f}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{f}}{\partial x} + \mathbf{B}\frac{\partial \mathbf{f}}{\partial y} = \Omega_{\text{PML}} + \Omega_{\text{BGK}}. \tag{4.31}$$

Using the $\mathbf{c}_i$ together with the Del operator, and reverting back to the more familiar scalar form of the particle distribution function we can write (4.31) as

$$\frac{\partial f_i}{\partial t} + \mathbf{c}_i \cdot \nabla f_i = \Omega_{\text{PML}} + \Omega_{\text{BGK}}, \tag{4.32}$$

where Ω_{PML} is given as

$$\Omega_{\text{PML}} = -\sigma\left(2\tilde{f}_i^{\text{eq}} + \sigma Q_i + \mathbf{c}_i \cdot \nabla Q_i\right). \tag{4.33}$$

[6] The term evolution equation is often used in the literature and simply describes the time evolutions of a system.

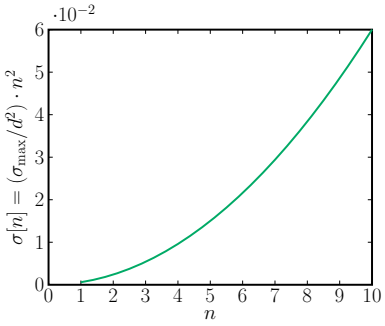


Figure 4.9: Graph of the damping coefficient $\sigma[n]$ varying from $\sigma_{\max}/d^2$ at the interface between the fluid region and PML region, to its max value $\sigma_{\max}$ at the boundary nodes. The thickness of the PML region is denoted with d. This is a depiction of the cross section of the damping profile for the east border, i.e. the at $n = 0$ we are in the fluid region, $n = 1$ we are in the first node in the PML region and $n = 10$ indicates that we are in the outermost node in a PML region of thickness 10.

The role of the PML is now neatly encapsulated in the additional collision term Ω_{PML}, and the fully discrete version can be derived by the same steps as proposed in section 3.1. As a last note, the damping coefficient σ should be varied quadratically from the first nodes in the PML to the last nodes, i.e. the boundary nodes. On a square, like the one in Figure 4.10, one possible realization of σ, perpendicular to the east wall, is

$$\sigma[n] = \begin{cases} \frac{\sigma_{\max}}{d^2} \cdot n^2, & n \in [1, \dots, d], \\ 0, & \text{otherwise}, \end{cases} \tag{4.34}$$

where n is the distance in nodes from the fluid region to the boundary. A plot of this function with $\sigma_{\max} = 0.06$ and $d = 10$ is found in Figure 4.9.

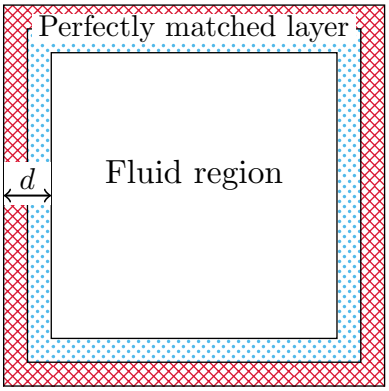


Figure 4.10: In a domain with PMLs, there are three different regions, the implementation of the LBM differs from region to region. The middle region is the fluid region, the blue dotted region is called the PML interior region and the red cross-hatched one is called the PML boundary region.

4.3.3 Implementation of the LBM with PML

When it comes to implementation, we first divide our computation domain into three regions: Fluid region, PML interior region and PML boundary region. This is visualized in Figure 4.10.

In the **fluid region** it is business as normal, i.e. implementation follows the algorithm from Figure 3.2

In the **PML interior region**, Ω_{PML} is included in the collision step, i.e., we are now solving

$$f_i^\dagger = \Omega_{\text{PML}} + \Omega_{\text{BGK}} + f_i = -\sigma \left[2\tilde{f}_i^{\text{eq}} + \sigma Q_i + \mathbf{c}_i \cdot \nabla Q_i\right] + \Omega_{\text{BGK}} + f_i. \tag{4.35}$$

To solve this numerically, we first have to integrate $\partial Q_i / \partial t = \tilde{f}_i^{\text{eq}}$ with some method to find Q, and secondly we to have to discretize $\mathbf{c}_i \cdot \nabla Q_i$ in some way.

The integration can be done by an Euler-Maclaurin quadrature[7], which by [48] is given as

$$\int_a^b f(t)\,\mathrm{d}t = \frac{h}{2}\left[f(a) + f(b)\right] + h\sum_{k=1}^{n-1} f(a+kh) \tag{4.36}$$

$$- \sum_{r=1}^{m-1} \frac{h^{2r} B_{2r}}{(2r)!}\left[f^{2r-1}(b) - f^{2r-1}(a)\right] \tag{4.37}$$

$$- n\frac{h^{2m+1} B_{2m}}{(2m)!} f^{2m}(\xi) \tag{4.38}$$

where n is the number of steps in the method, $h = (b-a)/n$ is the step length, B_r are the Bernoulli numbers and ξ is some number in (a, b). As we only have access to $\tilde{f}_i^{\text{eq}}$ at integer timesteps, the formula above reduces to the very simple

$$Q_i(\mathbf{x}, \Delta t_{k+1}) = \int_{\Delta t_k}^{\Delta t_{k+1}} \tilde{f}_i^{\text{eq}}\,\mathrm{d}t = \frac{\Delta t_{k+1} - \Delta t_k}{2}\left[\tilde{f}_i^{\text{eq}}(\mathbf{x}, \Delta t_k) + \tilde{f}_i^{\text{eq}}(\mathbf{x}, \Delta t_{k+1})\right],$$

[7] The term numerical quadrature, or simply quadrature is synonymous with numerical integration and is used interchangeably in the literature.

which is basically the one-step trapezoid rule. Here we have assumed that $\tilde{f}_i^{\text{eq}}(\mathbf{x}, 0) = 0$ and that the size of one timestep $\Delta t = 1$ in lattice units. Timestep number k is denoted with Δt_k, and $k \in \mathbb{N}$.

To discretize $\mathbf{c}_i \cdot \nabla Q_i$, the authors of [45] mention the use of a central difference scheme along the lattice directions. No order is mentioned but a second order central difference scheme, like the one in (4.14), has been adopted in this thesis, note that this second order scheme is only used for nodes in the PML interior region.

As mentioned, we need to discretize $\mathbf{c}_i \cdot \nabla Q_i$ in the different lattice directions. For Q_1, Q_2, Q_3 and Q_4 we simply use the appropriate central difference scheme seen before, but for Q_5, Q_6, Q_7 and Q_8, which points in a diagonal direction, we have to use a combination of central difference schemes in x- and y-direction. To clarify this point, a numerical stencil showing the points involved can be seen in Figure 4.11.

In the **PML boundary region** we repeat the same steps we did in the PML interior region, except when it comes to discretizing $\mathbf{c}_i \cdot \nabla Q_i$ as the lack of neighbor nodes on one side forces us to use a combination of one-sided discretization schemes.

The Q_is with directions pointing outside the domain are discretized with one-sided difference schemes, in the opposite direction of the outwards pointing Q_is. The differential terms for those Q_is whose directions point inside the domain are simply set to zero to avoid downwind discretization[8]. And finally, the Q_is who point alongside the boundaries are discretized with a first order central difference scheme. A flowchart of the process used in this thesis is presented in Figure 4.12.

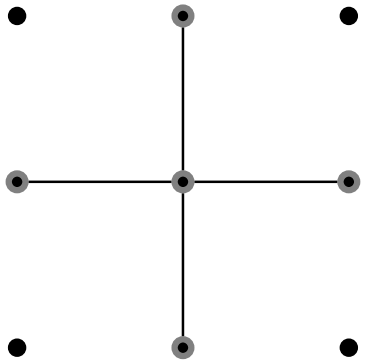


Figure 4.11: For Q_is pointing in a diagonal direction, we use a combination of horizontal and vertical central difference schemes. This is only done for nodes in the PML interior region.

[8] Upwind schemes are a class of numerical discretization methods to solve hyperbolic PDEs. The term upwind or downwind refers to the direction, in which the nodes used to calculate the derivatives point . If we use the x-axis as an example, an upwind scheme uses the points x_i, x_{i-1}, x_{i-2} etc., while a downwind scheme uses the points x_i, x_{i+1}, x_{i+2} etc.

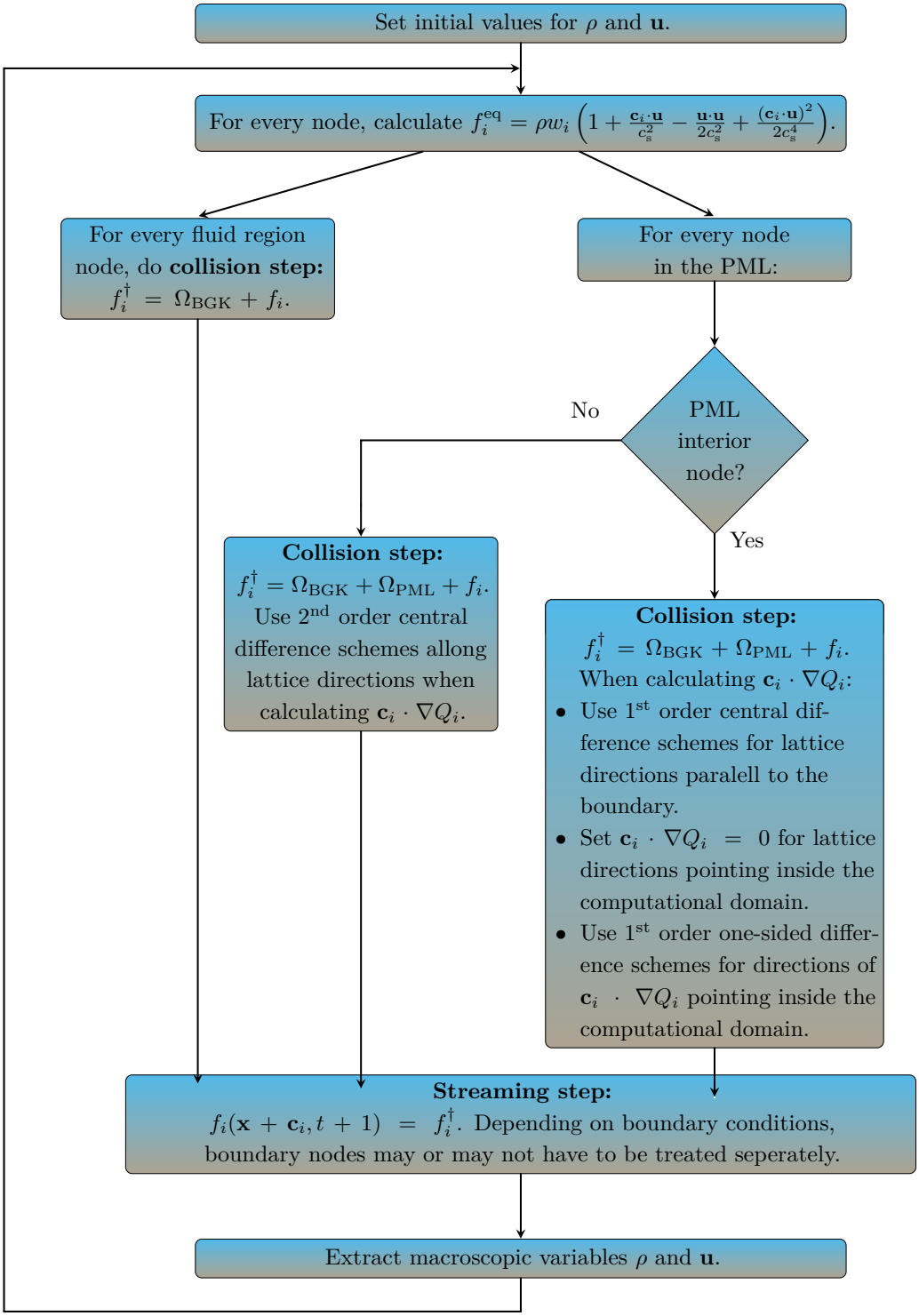


Figure 4.12: A summary of the LBM with perfectly matching layers represented in the form of a flowchart.

5 Tweaking of parameters and numerical simulations

If not otherwise stated, simulations with NRBCs were performed in a domain of 201×201 nodes. All values are given in lattice units.

Also, throughout this chapter, reference values of ρ and $\mathbf{u}$ are used in various ways. These values were obtained from simulation on a large domain measuring 3010×3010 nodes. The domain was then initialized with the initial values from section 5.4, simulation was then stopped just before hitting the walls of the domain. Values of ρ and $\mathbf{u}$ from a 201×201 sized cutout, centered around the center node, are then used as reference values. These will now be referred to as ρ_{ref}, $u_{x,\text{ref}}$ and $u_{y,\text{ref}}$ and represents values obtained from an ideal non-reflecting boundary condition.

5.1 Sponge layer thickness

The determining factor of a sponge layer's effectiveness is its thickness d. A thicker layer absorbs incoming waves to a larger degree, but also increases the computational cost as the total number of nodes increases if the fluid region (part of the computational domain not containing sponge layers) is to remain constant in size. To demonstrate the effect of different sponge layer thickness, simulations were done with $d = 10$ and $d = 50$. The density in a node situated at $\mathbf{x} = [71, 101]$ were compared against reference values of the density in the exact same node. The system was initialized with values from section 5.4. The variations in density and velocity can be found in Figure 5.1, from which we can see that a thicker sponge layer produces, in general, smaller fluctuations and lie closer to the reference values.

5.2 The damping coefficient in PML

A factor that decides the efficiency of a PML is the damping coefficient σ. As mentioned in subsection 4.3.2, σ should be varied quadratically, but out of curiosity, as was done in subsection 4.2.1, a different variation profile was tested. Values of ρ and $\mathbf{u}$ were recorded

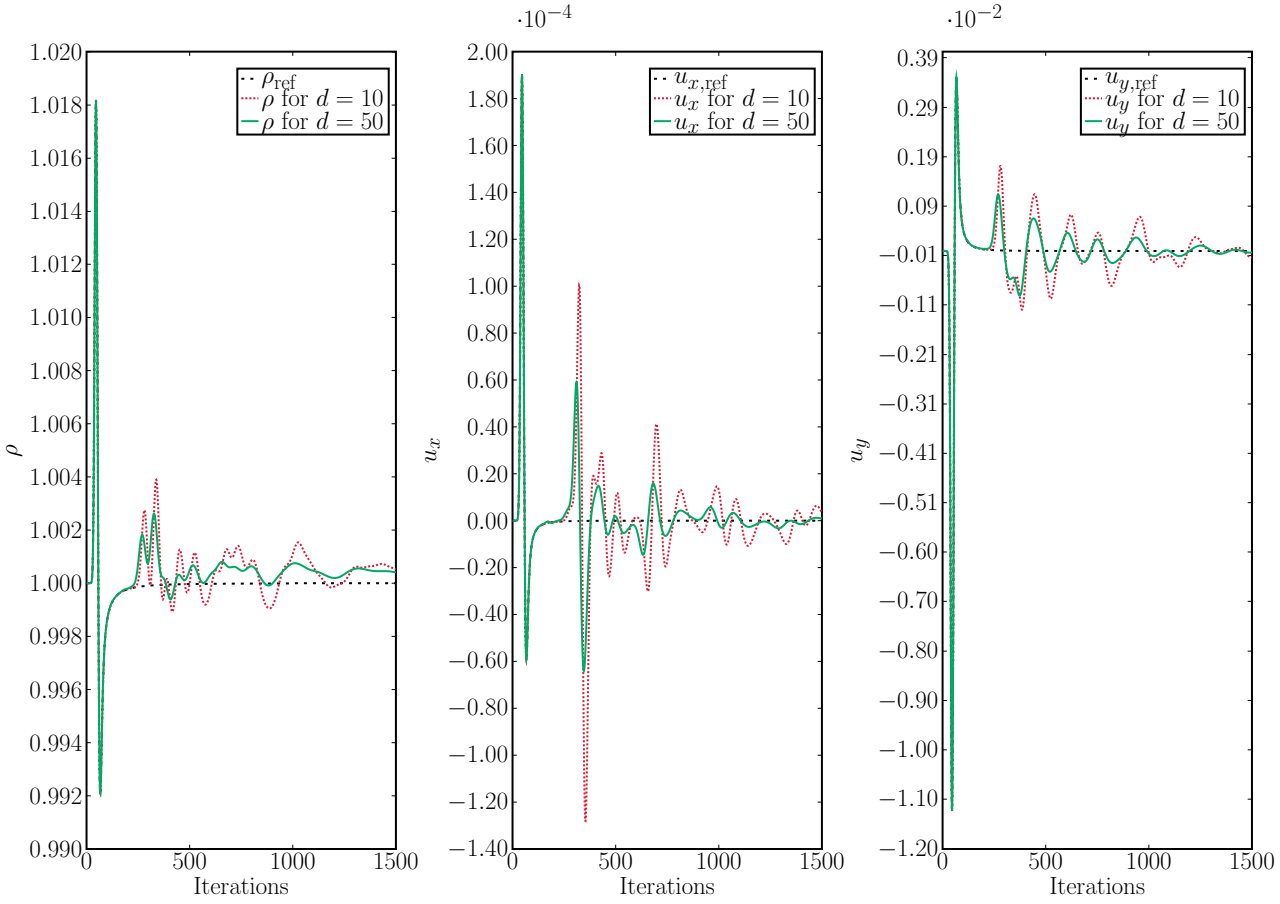


Figure 5.1: Values obtained from simulations with sponge layers of different thickness compared to reference values. Red dotted lines are values obtained from simulation with $d = 10$ and green solid lines are values obtained from simulations with $d = 50$. Simulation with $d = 50$ produces smaller fluctuations around the reference. The big spike at the beginning is the result of a Gauss pulse traveling through the measurement node.

in a certain point, with a quadratically and a logarithmically varying σ. These values were so compared with ρ_{ref} and $\mathbf{u}_{\text{ref}}$. No significant difference found, and we stick to the sigma profile originally suggested by Najafi-Yazdi and Mongeau.

The next thing to investigate is how the value of σ affects the reflections. The original article showed how different values of σ affected the reflections, but did not show how a quadratically varying σ with different values of $\sigma_{\max}$ would impact the reflected waves. Here we investigate both.

As previously, initial values from section 5.4 were used to initialize the system. First σ was held constant, but varied from from 0.01 to 0.10, increasing with 0.01 after each completion of the LB algorithm. In the second test, σ was varied according to (4.34) and $\sigma_{\max}$ was varied from 0.01 to 0.10, increasing with 0.01 after each completion of the LB algorithm. A similar test was done for the value range 0.1 to 1.0 with increases of 0.1. The error is measured as the absolute difference

$$E_\lambda = \sum_{\text{iterations}} |\lambda_{\text{ref}}(\mathbf{x}, t) - \lambda(\mathbf{x}, t)| \tag{5.1}$$

in the point $\mathbf{x} = [71, 101]$ where $\lambda(\mathbf{x}, t)$ is some general quantity. A total of 600 iterations was used to produce one E_λ value.

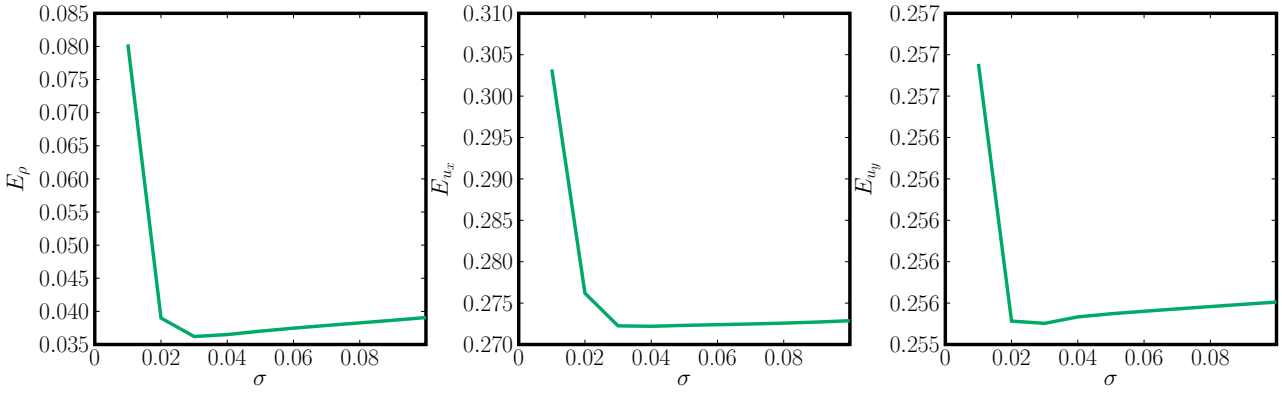


Figure 5.2: Time evolution of the error for ρ and $\mathbf{u}$ for $\sigma \in [0.01, 0.02, \ldots, 0.10]$. The profile of σ is kept constant throughout each run of the LB algorithm.

From Figure 5.2 and Figure 5.3 we can observe that there are no dramatic differences in absolute error, the lowest values, and for which value of σ and $\sigma_{\max}$, is presented in Table 5.1.

	Constant profile with $\sigma = 0.03$	Quadratically varying profile with $\sigma_{\max} = 1.0$
ρ	0.0362	0.0321
u_x	0.2723	0.2723
u_y	0.2555	0.2569

Table 5.1: The damping coefficient for each profile who gave the least absolute errors of ρ and $\mathbf{u}$.

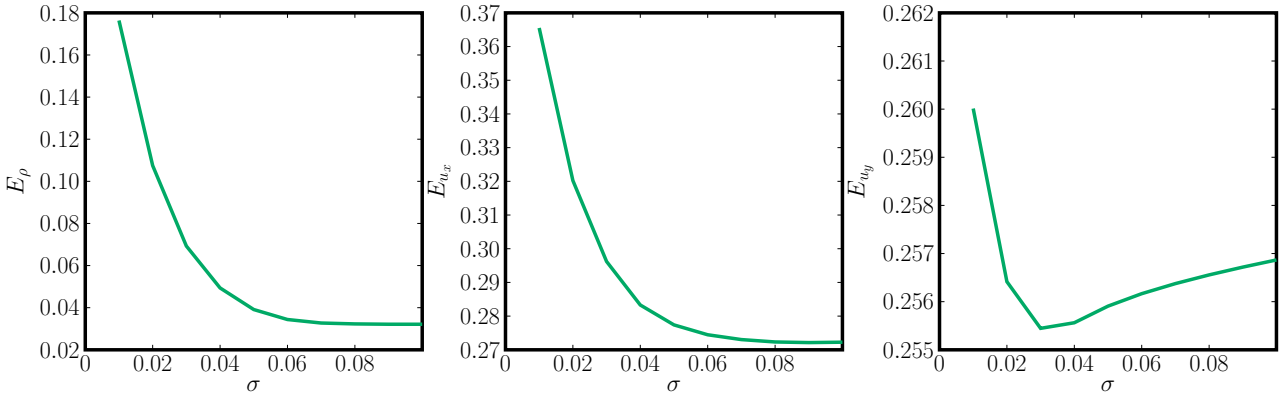


Figure 5.3: Time evolution of the error for ρ and $\mathbf{u}$ for $\sigma_{\max} \in [0.01, 0.02, \ldots, 0.10]$. The profile of σ is varied according to a quadratic function throughout each run of the LB algorithm.

From this we can conclude that a quadratically varying damping coefficient, internally in the algorithm, gives a slightly better result. Looking at figure Figure 5.4 and Figure 5.5 we see that $\sigma > 0.1$ results in an increasing error, indeed the simulations "blew up" to infinity around $\sigma = 0.7$. It was also mentioned by the authors of the original article that increasing σ above a certain optimum value may result in the growth of spurious waves. Judging by the results, the damping coefficient should be kept in the range [0.03,. . . ,0.1], in addition, a quadratically varying σ (internally in the LB algorithm) will be used in following simulations.

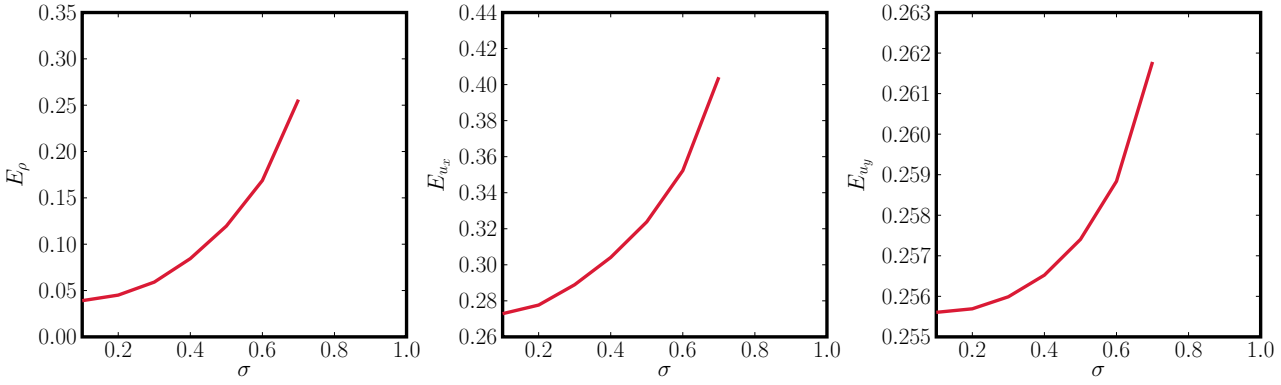


Figure 5.4: Time evolution of error for ρ and $\mathbf{u}$ for $\sigma \in [0.1, 0.2, \ldots, 1.0]$. The profile of σ is kept constant throughout each run of the LB algorithm.

5.3 The thickness of the perfectly matching layer

In addition to the damping coefficient σ, the thickness d is vital to the performance of the PML. Simulations was done to demonstrate this, and to find the best combination of σ and d. "Best combination" in this context is in terms of the absolute error in (5.1).

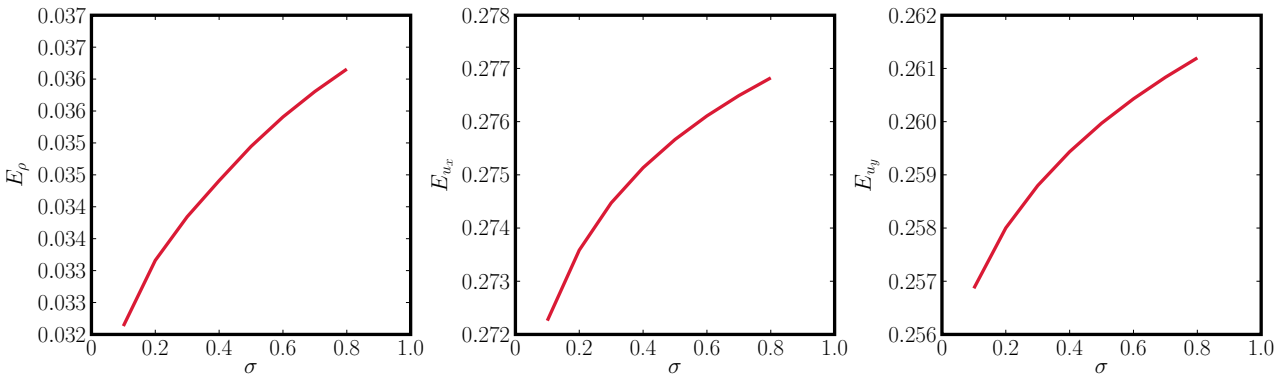


Figure 5.5: Time evolution of error for ρ and $\mathbf{u}$ for $\sigma_{\max} \in [0.1, 0.2, \ldots, 1.0]$. The profile of σ is varied according to a quadratic function throughout each run of the LB algorithm.

The test setup was identical to the tests performed in section 5.2, except that d was varied from 10 to 50, with increments of 10. For a given value of d, σ varied in the optimal range $[0, 04, \ldots, 0.1]$ (determined from section 5.2) with increments of 0.01.

From Figure 5.6 we observe that a higher value of d not necessarily means better PML performance, and as we can see, in some cases the error for a given value of d changes with increasing σ. As a reasonable trade-off, $d = 30$ with $\sigma = 0.07$ is chosen for use in future simulations.

5.4 Gaussian pulse propagation in two dimensions

To test the reflective, or rather, the non-reflective properties of the different NRBCs, the system is initialized with a Gauss pulse, and the resulting refelctions are studied. The Gauss pulse used here is given by

$$\rho(\mathbf{x}, t) = \rho_0 + 0.173\exp\left(-\frac{(x-\Xi)^2 + (y-\Upsilon)^2}{25}\right). \tag{5.2}$$

Here, the denominator in the exponential function decides how wide the pulse is, Ξ and Υ decides the placement of the pulse. In the simulations, the pulse was placed with its center aligned with the center node in the computational domain. The following initialization values for density and velocity for the system was used:

$$\rho_0 = 1, \quad \text{and} \quad \mathbf{u} = \mathbf{0}. \tag{5.3}$$

In testing the performance of the different NRBCs, we use the optimal values derived from the numerical simulations in the preceding sections. It is difficult to choose a test value for the thickness d in the case of sponge layers as an increasing d yields better results, for the sake of simplicity, d for sponge layers was set to the same value of d for PMLs. The test values are summed up in Table 5.2.

Note that $\gamma = 0.75$ was reproduced from [36], where the authors mention that their numerical tests showed best results for this value.

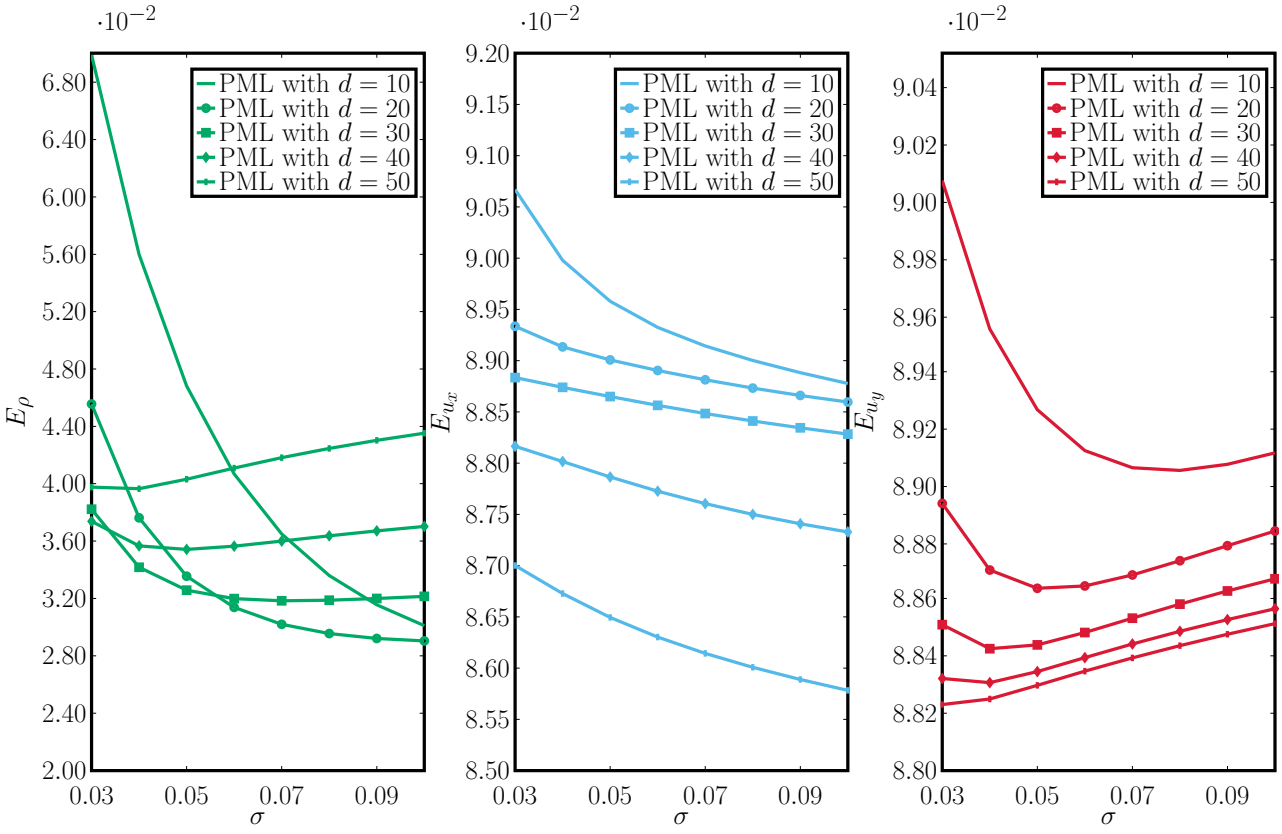


Figure 5.6: The effect of different PML thickness parameter d on the density and velocity.

NRBC	d	$\sigma_{\max}$	γ
Sponge layers	30	NA	NA
Characteristic boundary conditions	NA	NA	0.75
Perfectly matching layers	30	0.07	NA

Table 5.2: The values of performance related parameters in different NRBC used in Gaussian pulse simulation.

Using these parameter values and using the absolute difference as the error, we get Figure 5.7. From the figure we conclude that PMLs

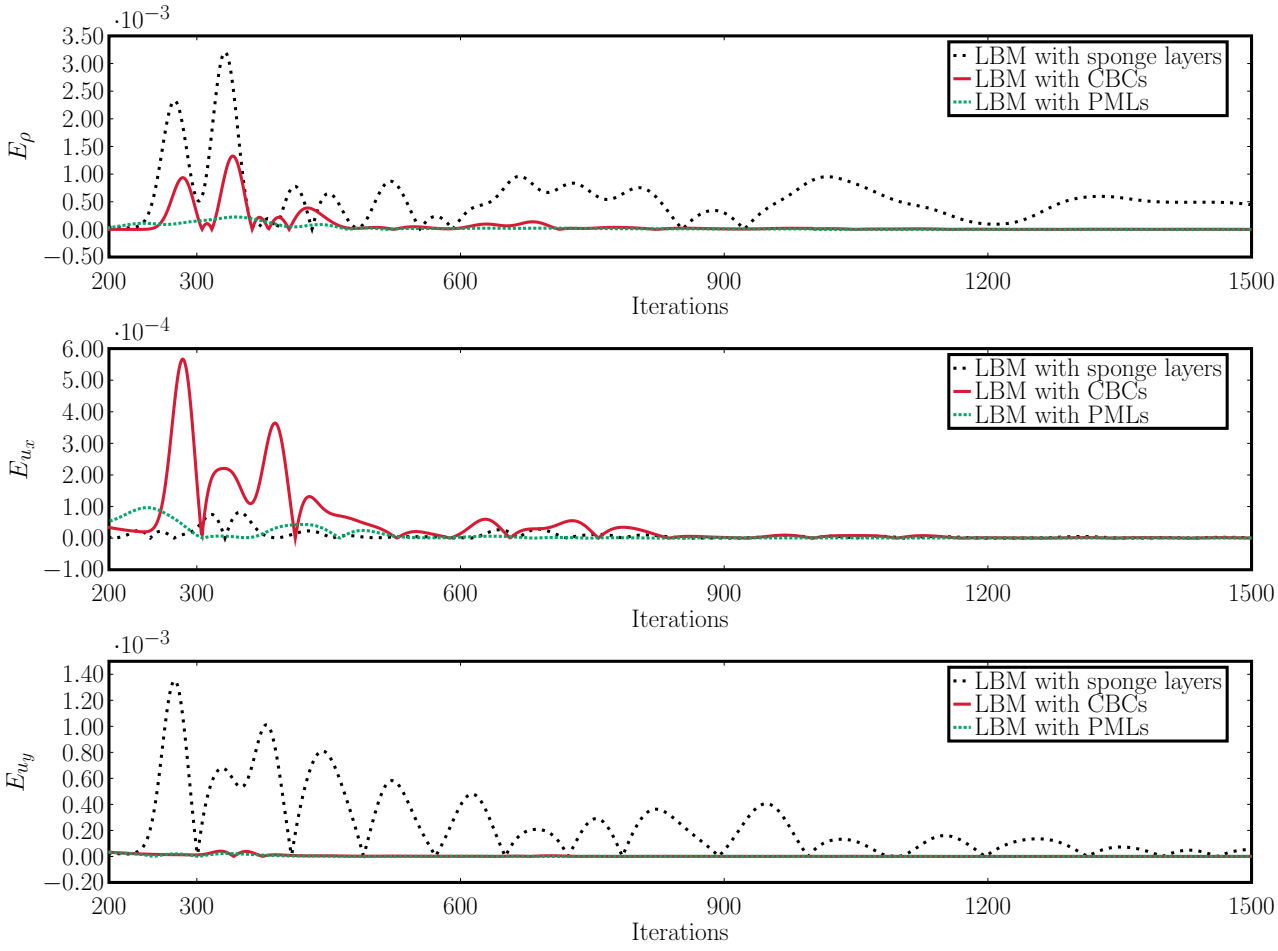


Figure 5.7: The error, measured as the absolute difference, of the LBM with three different non-reflecting boundary conditions. Only the evolution of the error from 200 iterations an beyond are shown since the reflections are not registered in the measuring point till after ≈ 200 iterations.

produce less reflections, and the reflections die out quicker. Regarding density, the fluctuations in E_ρ dies out after ≈ 540 iterations. The sponge layer method is the worst of the three, in terms of absolute difference, fluctuations are still present after 1500 iterations.

A figure covering the full range of iterations, from 1 to 1500, is presented in Figure 1 in Appendix 6. This figure is worth looking into as it shows some deviant behavior from the PMLs and CBCs in the first hundred or so iterations.

5.5 *Acoustic point source*

The point of this simulation is to mimic a sinusoidal wave emitted from a point source located in the middle of the computational domain. The varying center-node density is given by

$$\rho(\mathbf{x}, t) = \rho_0 + \rho_\mathrm{s} \sin\left(\frac{2\pi t}{T}\right), \tag{5.4}$$

where the point source amplitude $\rho_\mathrm{s} \leq \rho_0$ and T is the oscillation period in lattice units. For the simulation, $\rho_0 = 1$, $\mathbf{u} = \mathbf{0}$, $\rho_\mathrm{s} = 0.01$ and $T = 20$. As the density in the point source node is artificially varied according to (5.4), we reset the density in the node to its equilibrium value, $\rho_0 = 1$ in the collision step during the duration of the sinusoidal pulse. Doing this ensures that the correct number of particles is present in the node before streaming.

The error $\Delta\lambda$ is measured as the range in the arithmetic sense of the word, i.e.

$$\Delta\lambda = \max(\lambda) - \min(\lambda), \tag{5.5}$$

here, $\lambda(\mathbf{x}, t)$ is again some general quantity.

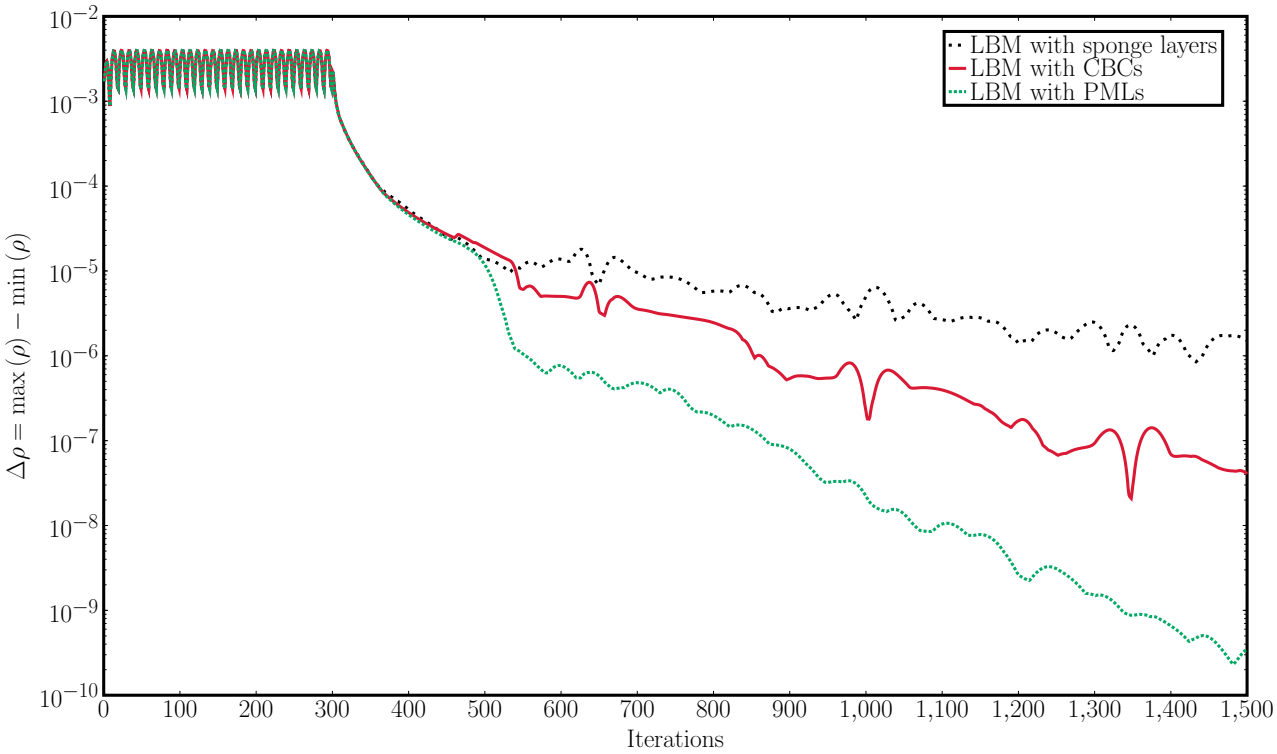


Figure 5.8: Semilogarithmic plot of the drop in the range $\Delta\rho$ as result of a sinusoidally varying point source being deactivated.

Figure 5.8 shows how the error $\Delta\rho$ drops as the point source stops producing sinusoidal waves. Ideally, $\Delta\rho$ would very quickly drop to

0, nevertheless, $\Delta\rho$ for the LBM with PMLs drops by seven orders of magnitude in approximately 1300 iterations. In accordance with the results from section 5.4, PMLs performed the best, while sponge layers performed the worst. The drop in $\Delta\rho$ for LBM with sponge layers was by three orders of magnitude in the same 1300 iterations.

5.6 Long time numerical stability

As a final test, the long time behavior of the different NRBCs were investigated. The Gaussian pulse from (5.2) and initial values from (5.3) were used to initialize the system. In the same manner as in [36], long time numerical stability was established by summing up the total mass in the system, in every iteration, for $4 \cdot 10^4$ iterations. A NRBC is said to be long time numerically stable if the total mass tends to $201 \cdot 201 \cdot \rho_0 = 40401$ (the computational domain was of size 201×201) as the number of iterations grow. The results can be seen in Figure 5.9 and Figure 5.10.

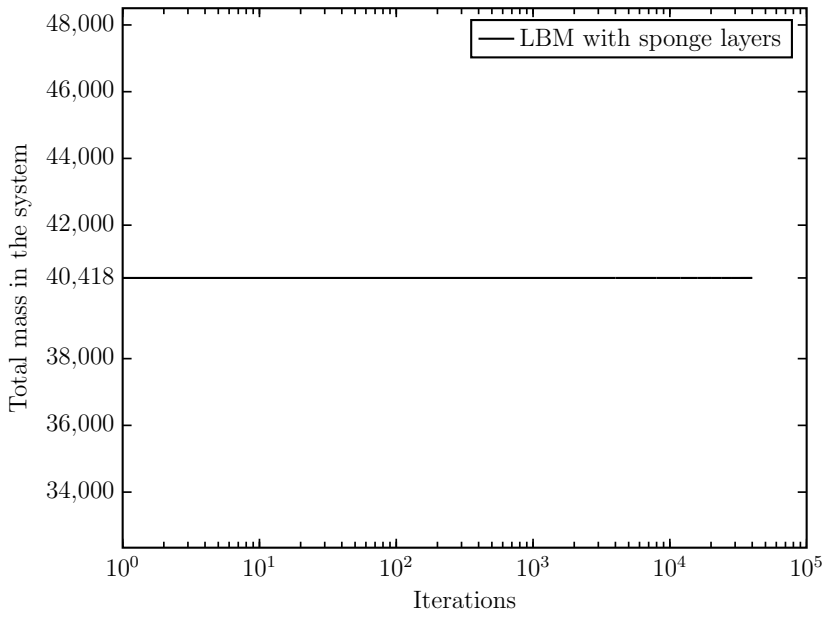


Figure 5.9: Long time numerical stability of LBM with sponge layers.

From the figures above we can see that the LBM with CBCs and PMLs reaches the expected value of 40401 after about 1600 and 1100 iterations respectively, and can therefore be considered numerically stable. On the other hand, LBM with sponge layers gave a total mass which stayed constant at a value of 40418, 417 units of mass away from the predicted value.

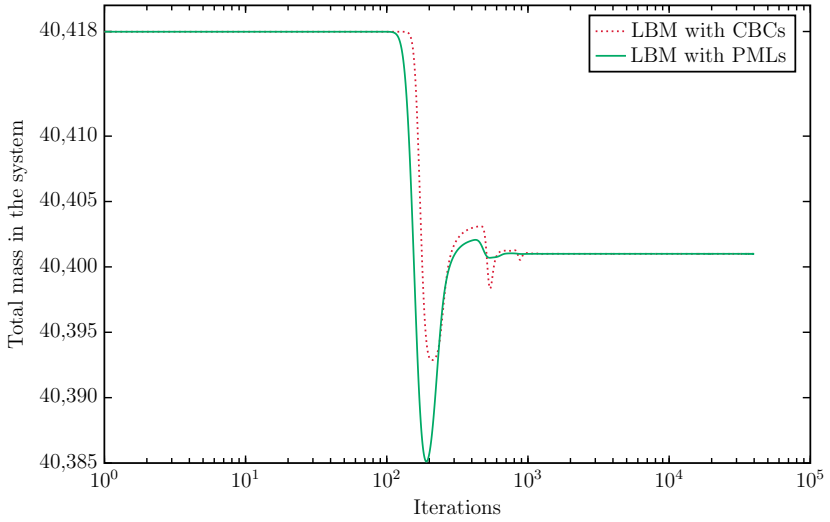


Figure 5.10: Long time numerical stability of LBM with CBCs and PMLs.

6 Conclusion

Three different non-reflecting boundary conditions (NRBCs) for the lattice Boltzmann method have been thoroughly accounted for and summarized, thereby providing a compendium of sorts for the three NRBCs perfectly matching layer (PMLs), sponge layers and characteristic boundary conditions (CBCs).

We have seen how PMLs, CBCs and sponge layers performed when tested against each other, and an idealized non-reflecting boundary condition. In terms of absolute difference between ideal reference values and values from numerical simulation with the different NRBCs, PML gave the least reflections. Of the three, the sponge layer method was the least optimal. However, the potential user needs to be aware of the fact that PMLs and sponge layers depend on damping layers with a certain thickness. Increasing the thickness improves the performance but also increases computational cost if the fluid region is to stay the same size. If this is unsuitable for the application, CBCs should be considered instead as they do not use damping layers.

A further study could include the use of absorbing boundary conditions for the sponge layer method and the PML method, instead of the simple bounce back boundary conditions used in this thesis. Ideally this would contribute to reducing reflections even more.

Bibliography

[1] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1998.

[2] E. R. Tufte. *Envisioning Information*. Cheshire, Connecticut: Graphics Press, 1990. ISBN: 0-9613921-1-8.

[3] E. R. Tufte. *Visual explanations: images and quantities, evidence and narrative*. Cheshire, Connecticut: Graphics Press, 1998.

[4] E. R. Tufte. *Beautiful Evidence*. Graphics Press, 2006. ISBN: 9780961392178.

[5] E. Ruder. *Typographie*. Visual Communication Books. Hastings House Publishers, 1981. ISBN: 9780803872233.

[6] S. Wolfram. *Cellular Automata and Complexity*. Westview Press, 1994.

[7] S. Wolfram. "Cellular automata as models of complexity." In: *Nature* 311 (1984), pp. 419 –424.

[8] J. Hardy, O. de Pazzis, and Y. Pomeau. "Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions." In: *Physical Review A: Atomic, Molecular and Optical Physics* 13 (5 1976), pp. 1949–1961. DOI: 10.1103/PhysRevA.13.1949. URL: http://link.aps.org/doi/10.1103/PhysRevA.13.1949.

[9] U. Frisch, B. Hasslacher, and Y. Pomeau. "Lattice-Gas Automata for the Navier-Stokes Equation." In: *Physical Review Letters* 56 (14 1986), pp. 1505–1508. DOI: 10.1103/PhysRevLett.56.1505. URL: http://link.aps.org/doi/10.1103/PhysRevLett.56.1505.

[10] E. M. Viggen. "The Lattice Boltzmann Method with Applications in Acoustics." Master's thesis. Norwegian University of Science and Technology, 2009.

[11] G.R. McNamara and G. Zanetti. "Use of the Boltzmann Equation to Simulate Lattice-Gas Automata." In: *Physical Review Letters* 61 (20 1988), pp. 2332–2335. DOI: 10.1103/PhysRevLett.61.2332. URL: http://link.aps.org/doi/10.1103/PhysRevLett.61.2332.

[12] R. K. Agarwal, K. Y. Yun, and R. Balakrishnan. "Beyond Navier-Stokes: Burnett equations for flows in the continuum transition regime." In: *Physics of Fluids* 13.10 (2001), pp. 3061–3085. URL: http://dx.doi.org/10.1063/1.1397256.

[13] A. V. Bobylev. "The Chapman–Enskog and Grad methods for solving the Boltzmann equations." In: *Soviet Physics - Doklady* 27 (1982).

[14] X. Zhong, R.W. Maccormack, and D.R. Chapman. "Stabilization of the Burnett equations and application to high-altitude hypersonic flows." In: *AIAA Journal* 31.6 (1993), pp. 1036–1043. DOI: 10.2514/6.1991-770. URL: http://dx.doi.org/10.2514/6.1991-770.

[15] D.A. Lockerby and J.M. Reese. "High-resolution Burnett simulations of micro Couette flow and heat transfer." In: *Journal of Computational Physics* 188.2 (2003), pp. 333 –347. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/S0021-9991(03)00162-1. URL: http://www.sciencedirect.com/science/article/pii/S0021999103001621.

[16] K. Xu and Z. Li. "Microchannel flow in the slip regime: gas-kinetic BGK–Burnett solutions." In: *Journal of Fluid Mechanics* 513.1 (2004), pp. 87–110. DOI: http://dx.doi.org/10.1017/S0022112004009826.

[17] H. Struchtrup. "Failures of the Burnett and super-Burnett equations in steady state processes." In: *Continuum Mechanics and Thermodynamics* 17.1 (2005), pp. 43–50. ISSN: 0935-1175. DOI: 10.1007/s00161-004-0186-0. URL: http://dx.doi.org/10.1007/s00161-004-0186-0.

[18] H. Struchtrup. *Macroscopic Transport Equations for Rarefied Gas Flows: Approximation Methods in Kinetic Theory*. Interaction of Mechanics and Mathematics. Springer Berlin Heidelberg, 2005. ISBN: 978-3-540-24542-1. DOI: 10.1007/3-540-32386-4.

[19] H. Struchtrup and M. Torrilhon. "$\mathcal{H}$ Theorem, Regularization, and Boundary Conditions for Linearized 13 Moment Equations." In: *Physical Review Letters* 99 (1 2007), p. 4. DOI: 10.1103/PhysRevLett.99.014502. URL: http://link.aps.org/doi/10.1103/PhysRevLett.99.014502.

[20] E. M. Viggen. "The lattice Boltzmann method: Fundamentals and acoustics." PhD thesis. Norwegian University of Science and Technology, 2014.

[21] P. L. Bhatnagar, E. P. Gross, and M. Krook. "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems." In: *Physical Review* 94 (3 1954), pp. 511–525. DOI: 10.1103/PhysRev.94.511. URL: http://link.aps.org/doi/10.1103/PhysRev.94.511.

[22] X. He and L.S Luo. "Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation." In: *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics* 56 (6 1997), pp. 6811–6817. DOI: 10.1103/PhysRevE.56.6811. URL: http://link.aps.org/doi/10.1103/PhysRevE.56.6811.

[23] J. Lätt. "Hydrodynamic Limit of Lattice Boltzmann Equations." PhD thesis. University of Geneva, 2007. URL: http://archive-ouverte.unige.ch/unige:464.

[24] D. A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*. Springer, 2000.

[25] A. J. Wagner. *A Practical Introduction to the Lattice Boltzmann Method*. Manual. North Dakota State University, 2008.

[26] A. Satoh. "8 - Theoretical Background of Lattice Boltzmann Method." In: *Introduction to Practice of Molecular Simulation*. Elsevier, 2011, pp. 255 –284. ISBN: 978-0-12-385148-2. DOI: http://dx.doi.org/10.1016/B978-0-12-385148-2.00008-2. URL: http://www.sciencedirect.com/science/article/pii/B9780123851482000082.

[27] J. G. Zhou. *Lattice Boltzmann Methods for Shallow Water Flows*. Springer Berlin Heidelberg, 2004. ISBN: 978-3-642-07393-9. DOI: 10.1007/978-3-662-08276-8.

[28] D. H. Rothman and S. Zaleski. *Lattice-Gas Cellular Automata: Simple Models of Complex Hydrodynamics*. 2004. ISBN: 9780521607605.

[29] Sam Bennet. "A Lattice Boltzmann Model for Diffusion of Binary Gas Mixtures." PhD. University of Cambridge, 2010.

[30] Q. Zou and X. He. "On pressure and velocity boundary conditions for the lattice Boltzmann BGK model." In: *Physics of Fluids* 9 (1997), pp. 1591–1598. DOI: 10.1063/1.869307.

[31] S. Chapman. "On the Law of Distribution of Molecular Velocities, and on the Theory of Viscosity and Thermal Conduction, in a Non-Uniform Simple Monatomic Gas." In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 216 (1916), pp. 279–348. ISSN: 02643952. URL: http://www.jstor.org/stable/91089.

[32] S. Chapman. "On the Kinetic Theory of a Gas. Part II: A Composite Monatomic Gas: Diffusion, Viscosity, and Thermal Conduction." In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 217 (1918), pp. 115–197. ISSN: 02643952. URL: http://www.jstor.org/stable/91120.

[33] D. Enskog. PhD dissertation. Uppsala, 1917.

[34] D. Enskog. "The numerical calculations of phenomena in fairly dense gases." In: *Arkiv för matematik, astronomi och fysik* 16.1 (1921).

[35] M. C. Sukop and D. T. Thorne. *Lattice Boltzmann Modeling, An Introduction for Geoscientists and Engineers*. Springer-Verlag, 2007. ISBN: 3-540-27981-4.

[36] D. Heubes, A. Bartel, and M. Ehrhardt. "Characteristic boundary conditions in the lattice Boltzmann method for fluid and gas dynamics." In: *Journal of Computational and Applied Mathematics* (2013). DOI: http://dx.doi.org/10.1016/j.cam.2013.09.019. URL: http://www.sciencedirect.com/science/article/pii/S0377042713004743.

[37] K. W. Thompson. "Time dependent boundary conditions for hyperbolic systems." In: *Journal of Computational Physics* 68.1 (1987), pp. 1 –24. DOI: http://dx.doi.org/10.1016/0021-9991(87)90041-6. URL: http://www.sciencedirect.com/science/article/pii/0021999187900416.

[38] G.W. Hedstrom. "Nonreflecting boundary conditions for nonlinear hyperbolic systems." In: *Journal of Computational Physics* 30.2 (1979), pp. 222 –237. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/0021-9991(79)90100-1. URL: http://www.sciencedirect.com/science/article/pii/0021999179901001.

[39] R.Kh. Zeytounian. *Navier-Stokes-Fourier equations. A rational asymptotic modeling point of view*. Vol. xvi. Berlin: Springer, 2012, p. 276. DOI: 10.1007/978-3-642-20746-4.

[40] M. Ehrhardt. "Absorbing boundary conditions for hyperbolic systems." In: *Numerical Mathematics* 3.3 (2010), pp. 295–337. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-80052234023&partnerID=40&md5=361a0de65dfed77f6462ce59e1f43dde.

[41] S. Izquierdo and N. Fueyo. "Characteristic nonreflecting boundary conditions for open boundaries in lattice Boltzmann methods." In: *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics* 78 (4 2008). DOI: 10.1103/PhysRevE.78.046707. URL: http://link.aps.org/doi/10.1103/PhysRevE.78.046707.

[42] E. Vergnault, O. Malaspinas, and P. Sagaut. "A lattice Boltzmann method for nonlinear disturbances around an arbitrary base flow." In: *Journal of Computational Physics* 231.24 (2012), pp. 8070 –8082. ISSN: 0021-9 991. DOI: http://dx.doi.org/10.1016/j.jcp.2012.07.021. URL: http://www.sciencedirect.com/science/article/pii/S0021999112003944.

[43] E. Vergnault, O Malaspinas, and P Sagaut. "Noise source identification with the lattice Boltzmann method." In: *The Journal of the Acoustical Society of America* 133.3 (2013), pp. 1293–1305. DOI: http://dx.doi.org/10.1121/1.4776181. URL: http://scitation.aip.org/content/asa/journal/jasa/133/3/10.1121/1.4776181.

[44] J. Papageorgakopoulos and S. Tsangaris. "A Numerical Method for Predicting Acoustical Wave Propagation in Open Spaces." In: *ISRN Mechanical Engineering* 2011 (2001). DOI: 10.5402/2011/174031.

[45] A. Najafi-Yazdi and A. Mongeau. "An absorbing boundary condition for the lattice Boltzmann method based on the perfectly matched layer." In: *Computers and Fluids* 68.0 (2012), pp. 203 –218. ISSN: 0045-7930. DOI: http://dx.doi.org/10.1016/j.compfluid.2012.07.017. URL: http://www.sciencedirect.com/science/article/pii/S004579301200285X.

[46] JP. Berenger. "A Perfectly Matched Layer for the Absorption of Electromagnetic Waves." In: *Journal of Computational Physics* 114.2 (1994), pp. 185–200. ISSN: 0021-9991. DOI: 10.1006/jcph.1994.1159. URL: http://dx.doi.org/10.1006/jcph.1994.1159.

[47] E. Bécache, S. Fauqueux, and P. Joly. "Stability of perfectly matched layers, group velocities and anisotropic waves." In: *Journal of Computational Physics* 188.2 (2003), pp. 399 –433. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/S0021-

9991(03)00184-0. URL: http://www.sciencedirect.com/science/article/pii/S0021999103001840.

[48] F. J. Smith. "Quadrature methods based on the Euler-Maclaurin formula and on the Clenshaw-Curtis method of integration." English. In: *Numerische Mathematik* 7.5 (1965), pp. 406–411. ISSN: 0029-599X. DOI: 10.1007/BF01436254. URL: http://dx.doi.org/10.1007/BF01436254.

Additional figures

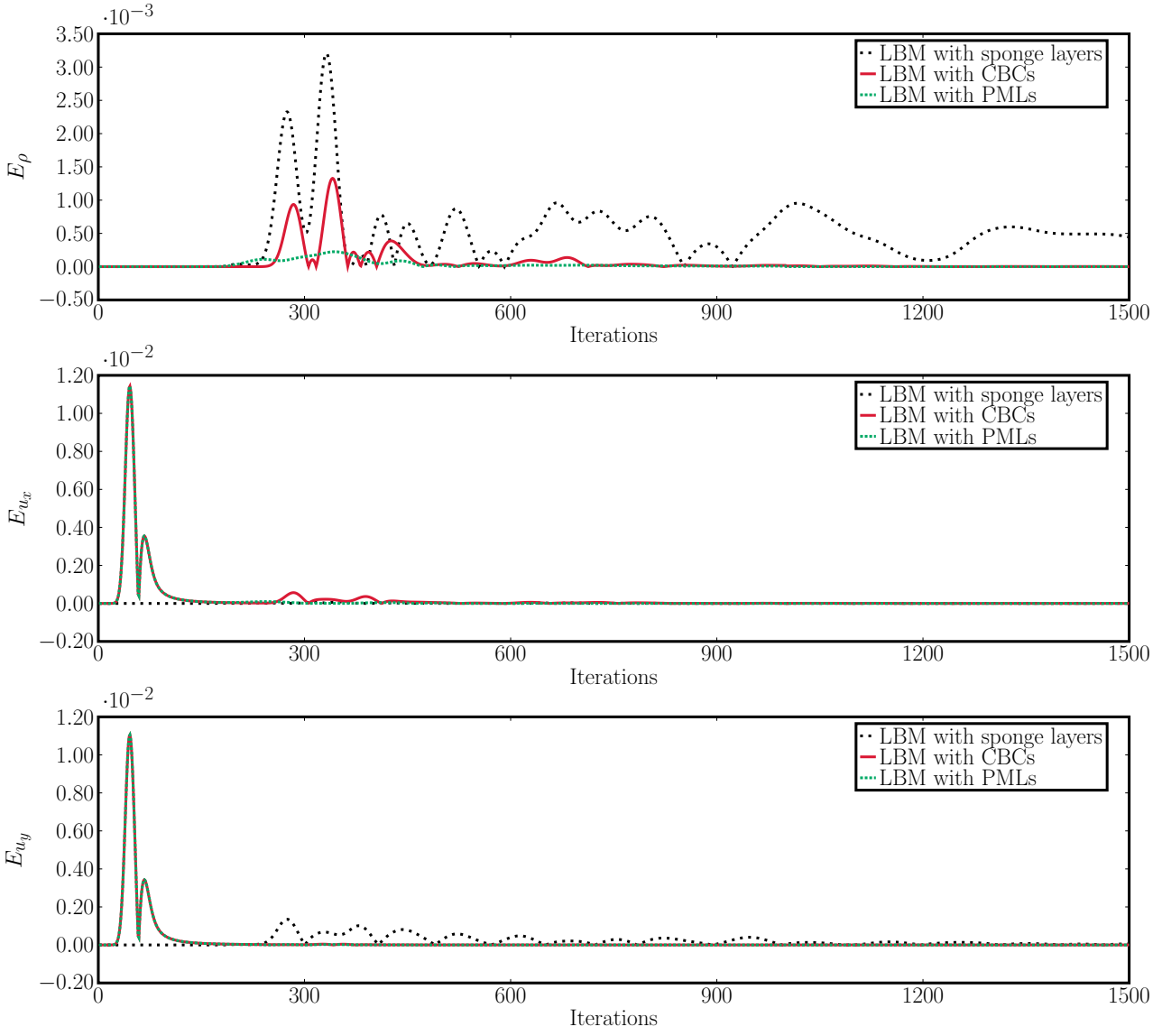


Figure 1: Error measured as absolute difference between reference values and values from simulations with different NRBCs.

Note the spikes occurring after approximately 50 iterations for E_{u_x} and E_{u_y}. Both LBM with PMLs and CBCs exhibits this behavior, while LBM with sponge layers does not. These values should not be present as they occur before the Gaussian pulse have reached the boundaries and reflections start propagating back. The absolute error between the reference values and simulation values with PMLs an CBCs should therefore be zero, as is the case with E_ρ. The author has no concrete explanation for this deviant behavior, but purely speculates that it might be connected to the actual implementation, and not the NRBCs themselves.

Matlab code

Although I have tried to be generous with comments, I would not recommend anyone to re-use or to try to adapt the code snippets presented here. They were not written with generality in mind, nor are they, I would imagine, particularly well written. For this reason, I have only included the files used to run a LB simulation with PMLs.

Although the code is quite messy, a brief inspection might provide some ideas or tricks, as for example the use of hypermatrices to represent every f_i on a 2D matrix or the use of linear indexing to avoid for loops.

Parts of the code are based on the file `cavity.m` by Jonas Lätt, found on `http://wiki.palabos.org/numerics:matlab_samples`. This site contains several LBM implementations in various languages and can serve as an additional resource to the novice LBM programmer.

PMLBC.m

```
0001 function [] = PMLBC(sigmaMax,d,T)
0002 %  c7  c3   c6  D2Q9 model. At each timestep, particle densities propagate
0003 %    \  |  /    outwards in the directions indicated in the figure. An
0004 %  c4 -c1 - c2  equivalent 'equilibrium' density is found, and thedensities
0005 %    /  |  \    relax towards that state, in a proportion governed by omega.
0006 %  c8  c5   c9
0007
0008
0009 % GENERAL SIMULATION CONSTANTS
0010 lx = 201; ly = 201;
0011 % T = 400; %number of iterations
0012 cs = 1/sqrt(3); % speed of sound in the fluid in lattice units for D2Q9
0013 nu = 0.1; % kinematic shear viscosity
0014 tau = nu/(cs^2) + 0.5; % relaxation factor
0015 % d = 25; % thickness of the PML
0016 omega = 1/tau;
0017
0018 % D29 LATTICE CONSTANTS
0019 t   = [4/9, 1/9, 1/9, 1/9, 1/9, 1/36, 1/36, 1/36, 1/36]';
```

```
0020 cx  = [0, 1, 0, -1,  0, 1, -1, -1,  1]';
0021 cy     = [0, 0, 1,  0, -1, 1,  1, -1, -1]';
0022
0023 % these are special maps used to navigate the computational domain
0024 [wWall,nWall,eWall,sWall,fluidR,PMLR,PMLIR,DMBN] =calcMaps2(lx,ly,d);
0025
0026 [dirMapDMBN] = calcDirectionMaps2(ly, lx, DMBN);
0027
0028 % INITIALIZE THE SYSTEM
0029 rho = ones(ly, lx);
0030 ux = zeros(ly, lx);
0031 uy = zeros(ly, lx);
0032
0033 f = reshape(t*ones(1, lx*ly), 9, ly, lx); % initial distribution
0034 fEq = zeros(9, ly, lx); % equilibrium distribution
0035 fTilde = zeros(9, ly, lx);      % temporary particle density distribution
0036 fEqBar = zeros(9, ly, lx);   % mean equilibrium distribution is constant
0037 fEqTilde = zeros(9, ly, lx); % acoustic perturbation distribution
0038 fEqTildeOld = fEqTilde;
0039
0040 % calculate the mean equilibrium:
0041 for i = 1:9
0042     uc = cx(i).*ux + cy(i).*uy;
0043     fEqBar(i,:,:) = t(i).*rho.*(1 + uc./(cs^2) + (uc.^2)./ ...
0044                     (2*cs^4) - (ux.^2 + uy.^2)./(2*cs^2));
0045 end
0046
0047 % assume Q(x,0) = 0;
0048 Q = zeros(9,ly,lx);
0049
0050 % INITAL VALUES
0051 % create gaussian pulse
0052 [X, Y] = meshgrid(1:ly,1:lx);
0053 Z = 0.173.*exp(-((X-ly/2).^2+(Y-lx/2).^2)./25);
0054 rho = rho + Z;
0055
0056
0057 % VISUALIZATION CONSTANTS
0058 sFrames =  10; % number of frames that is skipped in the visualization
0059 % process
0060
0061 % create sigma
0062 squares = cell(d,1);
0063 mock = ones(ly,lx);
```

```
0064 for k=2:(d+1)
0065     mock(k:(ly-k+1),k:(ly-k+1)) = 0;
0066     squares{k-1} = find(mock);
0067     % reset
0068     mock = ones(ly,lx);
0069 end
0070 sigmaMask=ones(9,ly,lx);
0071 % quadratic variation
0072 sigma=(sigmaMax/(d^2)).*((d:-1:1).^2);
0073 for k = d:-1:1
0074     sigmaMask(:,squares{k}) = sigma(k);
0075 end
0076
0077 % MAIN LOOP
0078 for iteration = 1:T
0079
0080     %%%%%%%% COLLISION STEP FOR FLUID REGION %%%%%%%%%%
0081     for i = 1:9
0082         % intermediary calculations of c vector times u vector and the
0083         % equilibrium distribution
0084         uc = cx(i).*ux + cy(i).*uy;
0085         fEq(i,:,:) = t(i).*rho.*(1 + uc./(cs^2) + (uc.^2)./ ...
0086             (2*cs^4) - (ux.^2 + uy.^2)./(2*cs^2));
0087     end
0088     fTilde(:,fluidR) = (1 - omega).*f(:,fluidR) + omega.*fEq(:,fluidR);
0089     %%%%%%%% COLLISION STEP FOR FLUID REGION END %%%%%%
0090
0091
0092     %%%%%%%% COLLISION STEP FOR THE PML %%%%%%%%%%
0093     OmegaBGK = -omega.*f(:,PMLR) + omega.*fEq(:,PMLR);
0094     fEqTilde(:,PMLR) = fEq(:,PMLR) - fEqBar(:,PMLR);
0095     Q(:,PMLR) = 0.5.*(fEqTildeOld(:,PMLR) + fEqTilde(:,PMLR));
0096     % euler-maclaurin one step
0097     cNQ = calccNablaQ(wWall,nWall,eWall,sWall,Q, PMLIR, PMLR, ly, lx);
0098
0099     OmegaPML = -sigmaMask(:,PMLR).*(cNQ(:,PMLR) + 2.*fEqTilde(:,PMLR) + ...
0100                 sigmaMask(:,PMLR).*Q(:,PMLR));
0101     fTilde(:,PMLR) = OmegaBGK + f(:,PMLR) + OmegaPML;
0102     %%%%%%%% COLLISION STEP FOR THE PML END %%%%%%%
0103
0104
0105     % STREAMING FOR FLUID REGION END
0106     % %%%%%%% STREAMING STEP FOR THE PML BOUNDARY %%%%%%%%%%
0107     % the populations f_i are computed with mid-way bounce back boundary
```

```
0108    % condition
0109
0110    % STREAMING STEP FOR FLUID DOMAIN AND PML EXCEPT BOUNDARY NODES %%%%
0111    for i = 1:9
0112        f(i, dirMapDMBN(i,:)) = fTilde(i, DMBN);
0113    end
0114    %%% STREAMING STEP FOR FLUID DOMAIN AND PML EXCEPT BOUNDARY NODES END %
0115
0116    %%%%%%%% CORNERS %%%%%%%
0117    % NW
0118    % these stay in the node:
0119    f(1,1) = fTilde(1,1);  f(2,1) = fTilde(4,1);    f(5,1) = fTilde(3,1);
0120    f(6,1) = fTilde(8,1);  f(8,1) = fTilde(6,1);    f(9,1) = fTilde(7,1);
0121    % these stream out of the node:
0122    f(5,1+1) = fTilde(5,1); f(9,1+ly+1) = fTilde(9,1);  f(2,1+ly) = ...
0123        fTilde(2,1);
0124
0125    % NE
0126    % these stay in the node:
0127    f(1,ly*(lx-1)+1-ly) = fTilde(1,ly*(lx-1)+1-ly);
0128    f(4,ly*(lx-1)+1-ly) = fTilde(2,ly*(lx-1)+1-ly);
0129    f(8,ly*(lx-1)+1-ly) = fTilde(6,ly*(lx-1)+1-ly);
0130    f(5,ly*(lx-1)+1-ly) = fTilde(3,ly*(lx-1)+1-ly);
0131    f(9,ly*(lx-1)+1-ly) = fTilde(7,ly*(lx-1)+1-ly);
0132    f(7,ly*(lx-1)+1-ly) = fTilde(9,ly*(lx-1)+1-ly);
0133    % these stream out of the node:
0134    f(4,ly*(lx-1)+1-ly) = fTilde(4,ly*(lx-1)+1);
0135    f(8,ly*(lx-1)+1-ly+1) = fTilde(8,ly*(lx-1)+1);
0136    f(5,ly*(lx-1)+1+1) = fTilde(5,ly*(lx-1)+1);
0137
0138    % SE
0139    % these stay in the node:
0140    f(4,lx*ly) = fTilde(2,lx*ly);    f(8,lx*ly) = fTilde(6,lx*ly);
0141    f(6,lx*ly) = fTilde(8,lx*ly);    f(3,lx*ly) = fTilde(5,lx*ly);
0142    f(7,lx*ly) = fTilde(9,lx*ly);    f(1,lx*ly) = fTilde(1,lx*ly);
0143    % these stream out of the node:
0144    f(3,lx*ly-1) = fTilde(3,lx*ly);   f(7,lx*ly-ly-1) = fTilde(7,lx*ly);
0145    f(4,lx*ly-ly) = fTilde(4,lx*ly);
0146
0147    % SW
0148    % these stay in the node:
0149    f(9,ly) = fTilde(7,ly);   f(2,ly) = fTilde(4,ly);
0150    f(6,ly) = fTilde(8,ly);   f(3,ly) = fTilde(5,ly);
0151    f(7,ly) = fTilde(9,ly);   f(1,ly) = fTilde(1,ly);
```

```
0152    % these stream out of the node:
0153    f(2,ly+ly) = fTilde(2,ly);   f(6,ly+ly-1) = fTilde(6,ly);
0154    f(3,ly-1) = fTilde(3,ly);
0155    %%%%%%%% CORNERS END %%%
0156
0157    %%%%%%%% BOUNDARIES EX. CORNERS %%%%%%%
0158    % W
0159    % these stay in the node:
0160    f(1,wWall) = fTilde(1,wWall);   f(9,wWall) = fTilde(7,wWall);
0161    f(2,wWall) = fTilde(4,wWall);   f(6,wWall) = fTilde(8,wWall);
0162    % these strema out of the node:
0163    f(2,wWall+ly) = fTilde(2,wWall);   f(6,wWall+ly-1) = fTilde(6,wWall);
0164    f(3,wWall-1) = fTilde(3,wWall);   f(5,wWall+1) = fTilde(5,wWall);
0165    f(9,wWall+ly+1) = fTilde(9,wWall);
0166
0167    % N
0168    % these stay in the node:
0169    f(1,nWall) = fTilde(1,nWall);   f(8,nWall) = fTilde(6,nWall);
0170    f(5,nWall) = fTilde(3,nWall);   f(9,nWall) = fTilde(7,nWall);
0171    % these strema out of the node:
0172    f(2,nWall+ly) = fTilde(2,nWall);   f(4,nWall-ly) = fTilde(4,nWall);
0173    f(8,nWall) = fTilde(8,nWall-ly+1);   f(5,nWall+1) = fTilde(5,nWall);
0174    f(9,nWall+ly+1) = fTilde(9,nWall);
0175
0176    % E
0177    % these stay in the node:
0178    f(1,eWall) = fTilde(1,eWall);   f(4,eWall) = fTilde(2,eWall);
0179    f(8,eWall) = fTilde(6,eWall);   f(7,eWall) = fTilde(9,eWall);
0180    % these strema out of the node:
0181    f(3,eWall-1) = fTilde(3,eWall);   f(7,eWall-ly-1) = fTilde(7,eWall);
0182    f(4,eWall-ly) = fTilde(4,eWall);   f(8,eWall-ly+1) = fTilde(8,eWall);
0183    f(5,eWall+1) = fTilde(5,eWall);
0184
0185    % S
0186    % these stay in the node:
0187    f(1,sWall) = fTilde(1,sWall);   f(3,sWall) = fTilde(5,sWall);
0188    f(6,sWall) = fTilde(8,sWall);   f(7,sWall) = fTilde(9,sWall);
0189    % these strema out of the node:
0190    f(2,sWall+ly) = fTilde(2,sWall);   f(6,sWall+ly-1) = fTilde(6,sWall);
0191    f(3,sWall-1) = fTilde(3,sWall);   f(7,sWall-ly-1) = fTilde(7,sWall);
0192    f(4,sWall-ly) = fTilde(4,sWall);
0193    %%%%%%%% BOUNDARIES EX. CORNERS END %%%
0194    %%%%%%%% STREAMING STEP FOR THE PML BOUNDARY END %%%%%%%
0195
```

```
0196     % EXTRACT MACROSCOPIC QUANTITIES RHO AND U
0197     rho = sum(f);  % produces a 2D matrix where the elements are the sums
0198     % of f along the first dimension, the dimension that contains the 9
0199     % different f's
0200     ux = reshape(cx'*reshape(f, 9, lx*ly), 1, ly, lx)./rho;
0201     uy = reshape(cy'*reshape(f, 9, lx*ly), 1, ly, lx)./rho;
0202
0203     % reshape to 2D matrices
0204     rho = reshape(rho, ly, lx);
0205     ux = reshape(ux, ly, lx);
0206     uy = reshape(uy, ly, lx);
0207
0208
0209 %     if(mod(iteration,sFrames)==0)
0210 %         % 2D VISUALIZATION
0211 %         imagesc(rho)
0212 %         colorbar
0213 %         title(iteration)
0214 %         drawnow
0215 %         % 3D VISUALIZATION
0216 %         %         colormap(jet)
0217 %         %         surfl(X, Y, rho,'light')
0218 %         %         shading interp;
0219 %         %         axis tight
0220 %         %         title(iteration)
0221 %         %         drawnow
0222 %     end
0223
0224     fEqTildeOld = fEqTilde;
0225 end
0226 end
```

calccNablaQ.m

```
0001 function [cNQ] = calccNablaQ(wWall,nWall,eWall,sWall,Q, PMLIR, ly, lx)
0002     % calcNablaQ calculates c_i times the spatial derivatives of the
0003     % different Q_i, the corners and boundaries (minus corners) are handled
0004     % seperately.
0005     % input:
0006     % wWall, nWall, eWall, sWall = linear indexes for the nodes on the
0007     % west, north etc. wall.
0008     % Q = the function to be derivated, Q is a hypermatrix with nine rows,
0009     % ly colums and lx "pages" (see matlab homepage/forums for description
0010     % of linear indexing in hypermatrices).
0011     % PMLIR = linear indexes of the nodes in the PML region excluding nodes
```

```
0012     % on the boundaries.
0013     % ly, lx = height and length respectively of computational domain.
0014     % Output:
0015     % cNQ = hypermatrix with the spatial derivatives of the Q_-
is, the
0016     % calculation is done for every node in the PML.
0017
0018     % for nodes in the PML not on the outer boundary (nodes facing
0019     % the outside of the computational domain)
0020     cNQ = zeros(9,ly,lx);
0021     % c_0 Nabla Q
0022     cNQ(1,PMLIR) = 0;
0023     % c_1 Nabla Q
0024     cNQ(2,PMLIR) = (Q(2,PMLIR + ly) - Q(2,PMLIR - ly))./2;
0025     % c_2 Nabla Q
0026     cNQ(3,PMLIR) = (Q(3,PMLIR - 1) - Q(3,PMLIR + 1))./2;
0027     % c_3 Nabla Q
0028     cNQ(4,PMLIR) = -cNQ(2,PMLIR);
0029     % c_4 Nabla Q
0030     cNQ(5,PMLIR) = -cNQ(3,PMLIR);
0031     % c_5 Nabla Q
0032     cNQ(6,PMLIR) = cNQ(2,PMLIR) + cNQ(3,PMLIR);
0033     % c_6 Nabla Q
0034     cNQ(7,PMLIR) = -cNQ(2,PMLIR) + cNQ(3,PMLIR);
0035     % c_7 Nabla Q
0036     cNQ(8,PMLIR) = -cNQ(2,PMLIR) - cNQ(3,PMLIR);
0037     % c_8 Nabla Q
0038     cNQ(9,PMLIR) = cNQ(2,PMLIR) - cNQ(3,PMLIR);
0039
0040     % for nodes on the outer boundary
0041     cNQ(1,wWall) = 0;
0042     cNQ(2,wWall) = 0; % inwards pointing diff terms = 0 to avoid
0043     % downwind discretization
0044     cNQ(3,wWall) = (Q(3,wWall + 1) - Q(3,wWall - 1))./2; % bussiness as
0045     % usual (central  difference schemes) when diff direction is alongside
0046     % boundary
0047     % difference scheme for diff directions pointing outside the domain:
0048     cNQ(4,wWall) = Q(4,wWall+ly) - Q(4,wWall);
0049     cNQ(5,wWall) = -cNQ(3,wWall); % bussiness as usual (
0050     % central difference schemes) when diff direction is alongside boundary
0051     cNQ(6,wWall) = 0; % inwards pointing diff terms = 0 to avoid
0052     % downwind discretization
0053     % difference scheme for diff directions pointing outside the domain:
0054     cNQ(7,wWall) = Q(7,wWall+ly) - Q(7,wWall) + Q(7,wWall+1) - Q(7,wWall);
```

```
0055     % difference scheme for diff directions pointing outside the domain
0056     cNQ(8,wWall) = Q(8,wWall+ly) - Q(8,wWall) + Q(8,wWall-1) - Q(8,wWall);
0057     cNQ(9,wWall) = 0; % inwards pointing diff terms = 0 to avoid
0058     % downwind discretization
0059
0060     cNQ(1,nWall) = 0;
0061     cNQ(2,nWall) = (Q(2,nWall + ly) - Q(2,nWall - ly))./2;
0062     cNQ(3,nWall) = Q(3,nWall+1) - Q(3,nWall);
0063     cNQ(4,nWall) = -cNQ(2,nWall);
0064     cNQ(5,nWall) = 0;
0065     cNQ(6,nWall) = Q(6,nWall-ly) - Q(6,nWall) + Q(6,nWall+1) - Q(6,nWall);
0066     cNQ(7,nWall) = Q(7,nWall+ly) - Q(7,nWall) + Q(7,nWall+1) - Q(7,nWall);
0067     cNQ(8,nWall) = 0;
0068     cNQ(9,nWall) = 0;
0069
0070     cNQ(1,eWall) = 0;
0071     cNQ(2,eWall) = Q(2,eWall-ly) - Q(2,eWall);
0072     cNQ(3,eWall) = (Q(3,eWall + 1) - Q(3,eWall - 1))./2;
0073     cNQ(4,eWall) = 0;
0074     cNQ(5,eWall) = -cNQ(3,eWall);
0075     cNQ(6,eWall) = Q(6,eWall-ly) - Q(6,eWall) + Q(6,eWall) - Q(6,eWall);
0076     cNQ(7,eWall) = 0;
0077     cNQ(8,eWall) = 0;
0078     cNQ(9,eWall) = Q(9,eWall-ly) - Q(9,eWall) + Q(9,eWall-1) - Q(9,eWall);
0079
0080     cNQ(1,sWall) = 0;
0081     cNQ(2,sWall) = (Q(2,sWall + ly) - Q(2,sWall - ly))./2;
0082     cNQ(3,sWall) = 0;
0083     cNQ(4,sWall) = -cNQ(2,sWall);
0084     cNQ(5,sWall) = Q(5,sWall-1) - Q(5,sWall);
0085     cNQ(6,sWall) = 0;
0086     cNQ(7,sWall) = 0;
0087     cNQ(8,sWall) = Q(8,sWall+ly) - Q(8,sWall) + Q(8,sWall-1) - Q(8,sWall);
0088     cNQ(9,sWall) = Q(9,sWall-ly) - Q(9,sWall) + Q(9,sWall-1) - Q(9,sWall);
0089
0090     % north west corner
0091     cNQ(1,1) = 0;
0092     cNQ(2,1) = 0;
0093     cNQ(3,1) = Q(3,1+1) - Q(3,1);
0094     cNQ(4,1) = Q(4,1+ly) - Q(4,1);
0095     cNQ(5,1) = 0;
0096     cNQ(6,1) = 0;
0097     cNQ(7,1) = Q(7,1+ly) - Q(7,1) + Q(7,1+1) - Q(7,1);
0098     cNQ(8,1) = 0;
```

```
0099     cNQ(9,1) = 0;
0100
0101     % north east corner
0102     cNQ(1,ly*(lx-1)+1) = 0;
0103     cNQ(2,ly*(lx-1)+1) = Q(2,ly*(lx-1)+1-ly) - Q(2,ly*(lx-1)+1);
0104     cNQ(3,ly*(lx-1)+1) = Q(3,ly*(lx-1)+1+1) - Q(3,ly*(lx-1)+1);
0105     cNQ(4,ly*(lx-1)+1) = 0;
0106     cNQ(5,ly*(lx-1)+1) = 0;
0107     cNQ(6,ly*(lx-1)+1) = Q(6,ly*(lx-1)+1-ly) - Q(6,ly*(lx-1)+1) + ...
0108                          Q(6,ly*(lx-1)+1) - Q(6,ly*(lx-1)+1);
0109     cNQ(7,ly*(lx-1)+1) = 0;
0110     cNQ(8,ly*(lx-1)+1) = 0;
0111     cNQ(9,ly*(lx-1)+1) = 0;
0112
0113     % south east corner
0114     cNQ(1,lx*ly) = 0;
0115     cNQ(2,lx*ly) = Q(2,lx*ly-ly) - Q(2,lx*ly);
0116     cNQ(3,lx*ly) = 0;
0117     cNQ(4,lx*ly) = 0;
0118     cNQ(5,lx*ly) = Q(5,lx*ly-1) - Q(5,lx*ly);
0119     cNQ(6,lx*ly) = 0;
0120     cNQ(7,lx*ly) = 0;
0121     cNQ(8,lx*ly) = 0;
0122     cNQ(9,lx*ly) = Q(9,lx*ly-ly) - Q(9,lx*ly) + Q(9,lx*ly-1) - Q(9,lx*ly);
0123
0124     % south west corner
0125     cNQ(1,ly) = 0;
0126     cNQ(2,ly) = 0;
0127     cNQ(3,ly) = 0;
0128     cNQ(4,ly) = Q(4,ly+ly) - Q(4,ly);
0129     cNQ(5,ly) = Q(5,ly-1) - Q(5,ly);
0130     cNQ(6,ly) = 0;
0131     cNQ(7,ly) = 0;
0132     cNQ(8,ly) = Q(8,ly+ly) - Q(8,ly) + Q(8,ly-1) - Q(8,ly);
0133     cNQ(9,ly) = 0;
0134 end
```

calcDirectionMaps2.m

```
0001 function[dirMapDMBN] = calcDirectionMaps2(ly, lx, DMBN)
0002 % calcDirectionMaps2. calculates the movement patters used to stream the
0003 % entire domain, except the boundary nodes which need special attention due
0004 % to boundary conditions
0005 % input:
0006 % ly, lx = height and length respectively of computational domain.
```

```
0007 % DMBN = linear indexes of the entire computational domain, except the
0008 % boundary nodes.
0009 % Output:
0010 % dirMapDMBN = a matrix with 9 rows and size(DMBN) columns, contains the
0011 % linear indices of the movement patterns for each direction i. See figure
0012 % in PMLBC.m for i directions
0013     dirMapDMBN = zeros(9, size(DMBN,1));
0014     dirMapDMBN(1,:) = DMBN;
0015     dirMapDMBN(2,:) = DMBN + ly;
0016     dirMapDMBN(3,:) = DMBN - 1;
0017     dirMapDMBN(4,:) = DMBN - ly;
0018     dirMapDMBN(5,:) = DMBN + 1;
0019     dirMapDMBN(6,:) = DMBN + ly - 1;
0020     dirMapDMBN(7,:) = DMBN - ly - 1;
0021     dirMapDMBN(8,:) = DMBN - ly + 1;
0022     dirMapDMBN(9,:) = DMBN + ly + 1;
0023 end
```

calcMaps2.m

```
0001 function [wWall,nWall,eWall,sWall,fluidR,PMLR,PMLIR,DMBN] = ...
0002          calcMaps2(lx,ly,d)
0003 % this function finds the linear indexes belonging to the different regions
0004 % of the computational domain. These are necessary to address the various
0005 % input:
0006 % ly, lx = height and length respectively of computational domain.
0007 % d = thickness of PML.
0008 % Output:
0009 % wWall, nWall, eWall, sWall = linear indexes for the nodes on the
0010 % west, north etc. wall.
0011 % PMLIR = linear indexes of the nodes in the PML region excluding nodes
0012 % on the boundaries.
0013 % PMLR = linear indexes of the nodes in the PML
0014 % DMBN = linear indexes of the entire computational domain, except the
0015 % boundary nodes.
0016 if(d <= 2)
0017     disp('d has to be greater than or equal to 3');
0018     return
0019 end
0020
0021 temp = zeros(ly, lx);
0022
0023 % west wall without corners
0024 wWall = temp;
0025 wWall(2:ly-1,1) = 1;
```

```
0026 wWall = find(wWall);
0027 % north wall without corners
0028 nWall = temp;
0029 nWall(1,2:lx-1) = 1;
0030 nWall = find(nWall);
0031 % east wall without corners
0032 eWall = temp;
0033 eWall(2:ly-1,lx) = 1;
0034 eWall = find(eWall);
0035 % south wall without corners
0036 sWall = temp;
0037 sWall(ly,2:lx-1) = 1;
0038 sWall = find(sWall);
0039
0040 % fluid region
0041 fluidR = temp;
0042 fluidR(d+1:ly-d,d+1:lx-d) = 1;
0043 fluidR = find(fluidR);
0044
0045 % entire PML region
0046 PMLR = ones(ly,lx);
0047 PMLR(d+1:ly-d,d+1:lx-d) = 0;
0048 PMLR = find(PMLR);
0049
0050 % only boundary nodes, i.e. those who face the exterior of the
0051 % computational domain
0052 BR = zeros(ly,lx);
0053 BR(:,1) = 1; BR(1,2:lx-1) = 1; BR(:,lx) = 1; BR(ly,2:lx-1) = 1;
0054 BR = find(BR);
0055 % Entire PML region except boundary (BR) nodes
0056 PMLIR = zeros(ly,lx);
0057 PMLIR(PMLR) = 1;
0058 PMLIR(BR) = 0;
0059 PMLIR = find(PMLIR);
0060 % entire Domain minus boundary Nodes
0061 DMBN = ones(ly,lx);
0062 DMBN(BR) = 0;
0063 DMBN = find(DMBN);
0064 end
```