Lattice Boltzmann Method

Internship Work Report

by

Nakul D. Ghate (p110691)

under the guidance of

Prof. Jean-Yves Trépanier

and

Sami Ammar

Research Internship in the field of

Compuational Fluid Dynamics

at

Ecolé Polytechnique Montréal

Montréal H3C

Contents

List of Figures

Acknowledgement

I would like to thank my parents for giving me such a wonderful guidance and teachings without which it would not have been possible. Also, I would like to thank my guide Prof. Jean-Yves Trépanier for giving me this wonderful opportunity to work with him and Sami Ammar for all the help I needed. Thank you.

Abstract

In recent years, the lattice Boltzmann method (LBM) has developed into an alternative and promising numerical scheme for simulating fluid flows and modeling physics in fluids. The scheme is particularly successful in fluid flow applications involving interfacial dynamics and complex boundaries. Unlike conventional numerical schemes based on discretizations of macroscopic continuum equations, the lattice Boltzmann method is based on microscopic models and mesoscopic kinetic equations. The fundamental idea of the LBM is to construct simplied kinetic models that incorporate the essential physics of microscopic or mesoscopic processes so that the macroscopic averaged properties obey the desired macroscopic equations. The basic premise for using these simplied kinetic-type methods for macroscopic fluid flows is that the macroscopic dynamics of a fluid is the result of the collective behavior of many microscopic particles in the system and that the macroscopic dynamics is not sensitive to the underlying details in microscopic physics (Kadanoff 1986). By developing a simplied version of the kinetic equation, one avoids solving complicated kinetic equations such as the full Boltzmann equation, and one avoids following each particle as in molecular dynamics simulations. Even though the LBM is based on a particle picture, its principal focus is the averaged macroscopic behavior. The kinetic equation provides many of the advantages of molecular dynamics, including clear physical pictures, easy implementation of boundary conditions, and fully parallel algorithms. Because of the availability of very fast and massively parallel machines, there is a current trend to use codes that can exploit the intrinsic features of parallelism. The LBM fullls these requirements in a straightforward manner.

The project report is sub-divided in four parts. In the first part, the conceptual framework from statistical mechanics to fluid dynamics and the theoretical foundation for the lattice Boltzmann method are constructed. The emphasis is made on the implementation of BGK collision scheme, investigating different types of boundary

conditions and implementation of single and multi-relaxation time schemes. This is followed with simulations of common benchmark flow problems of Lid-Driven cavity and Poisuelle flow.

The second part focuses on the simulation of multiphase flows using Lattice Boltzmann Method involving the pseudopotential model. The case of falling droplet in a viscous medium under the effect of gravity is considered to explain the dynamics of multiphase flows which involves fluid-fluid and fluid-solid interactions and descretization of force terms.Shan-Chen model is considered to implement the case of falling droplet.

The third part of this project report discusses the implementation of phase-transition Heat transfer using Lattice Boltzmann Method. The Stefan's problem is taken. Heat transfer and fluid flow processes of natural convection melting of a phase change material are simulated inside a partially heated square cavity. The momentum and energy equations are solved by using enthalpy-based lattice Boltzmann method combined with multi distribution function model. In this communication, the dependence of liquid fraction, temperatures of vertical nodes and average Nusselt number on the positions of heated plates is investigated quantitatively.

The final part of this report combines the multiphase and the Stefan's problem to simulate the impact and freezing of droplet on a cold plate at cryogenic temperature. The effect of wall temperature and the gravitational force on freezing speed is also investigated quantitatively.

Chapter 1

Lattice Boltzmann Method for fluid Simulations

1.1 Introduction

In the last two decades, the Lattice Boltzmann method (LBM) has emerged as a promising tool for modelling the Navier-Stokes equations and simulating complex uid ows. LBM is based on microscopic models and mesoscopic kinetic equations. In some perspective, it can be viewed as a finite difference method for solving the Boltzmann transport equation. Moreover the Navier-Stokes equations can be recovered by LBM with a proper choice of the collision operator. In Section 2 and 3, we first introduce this method and describe some commonly used boundary conditions. In Section 4, the validity of this method is confirmed by comparing the numerical solution to the exact solution of the steady plane Poiseuille flow and convergence of solution is established. Some interesting numerical simulations, including the lid-driven cavity flow with forced and Natural convection are carried out in Section 5, 6 and 7. In Section 8, we briey highlight the procedure of recovering the Navier-Stokes equations from LBM. A summary is provided in Section 9.

1.2 Lattice Boltzmann Method

The Lattice Boltzmann method [1, 2, 3] was originated from Ludwig Boltzmanns kinetic theory of gases. The fundamental idea is that gases/uids can be imagined as consisting of a large number of small particles moving with random motions. The exchange of momentum and energy is achieved through particle streaming

and billiard-like particle collision. This process can be modelled by the Boltzmann transport equation, which is

$$\frac{\partial f}{\partial t} + \vec{u}.\nabla f = \Omega \tag{1.1}$$

where $f(\vec{u}, t)$ is the particle distribution function, $\vec{u}$ is the particle velocity, and Ω is the collision operator. The LBM simplies Boltzmanns original idea of gas dynamics by reducing the number of particles and conning them to the nodes of a lattice. For a two dimensional model, a particle is restricted to stream in a possible of 9 directions, including the one staying at rest. These velocities are referred to as the microscopic velocities and denoted by $\vec{e_i}$, where i = 0,...,8. This model is commonly known as the D2Q9 model as it is two dimensional and involves 9 velocity vectors. Figure 1 shows a typical lattice node of D2Q9 model with 9 velocities $\vec{e_i}$ dened by

$$\vec{e_i} = \left\{ \begin{array}{cc} (0,0) & \text{i} = 0 \\ \frac{\delta x}{\delta t}\left(cos\left[(i-1)\frac{\pi}{2}\right], sin\left[(i-1)\frac{\pi}{2}\right]\right) & \text{i} = 1,2,3,4 \\ \sqrt{2}\frac{\delta x}{\delta t}\left(cos\left[(2i-1)\frac{\pi}{4}\right], sin\left[(2i-1)\frac{\pi}{4}\right]\right) & \text{i} = 5,6,7,8 \end{array} \right\} \tag{1.2}$$

For each particle on the lattice, we associate a discrete probability distribution function $f_i(\vec{x}, \vec{e_i}, t)$ or simply $f_i(\vec{x}, t)$, i = 0...8, which describes the probability of streaming in one particular direction.

The macroscopic uid density can be dened as a summation of microscopic particle distribution function,

$$\rho(\vec{x}, t) = \sum_{n=1}^{9} f_i(\vec{x}, t) \tag{1.3}$$

Accordingly, the macroscopic velocity $\vec{u}(\vec{x}, t)$ is an average of microscopic velocities ei weighted by the distribution functions f_i,

$$\rho\vec{u}(\vec{x}, t) = \sum_{n=1}^{9} c f_i \vec{e_i} \tag{1.4}$$

The key steps in LBM are the streaming and collision processes which are given by

$$f_i(\vec{x} + c\vec{e_i}\Delta t, t + \Delta t) - f_i(\vec{x}, t) = -\frac{[f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)]}{\tau} \tag{1.5}$$

In the actual implementation of the model, streaming and collision are computed separately, and special attention is given to these when dealing with boundary lattice nodes. Figure 2 shows graphically how the streaming step takes place for the interior nodes.

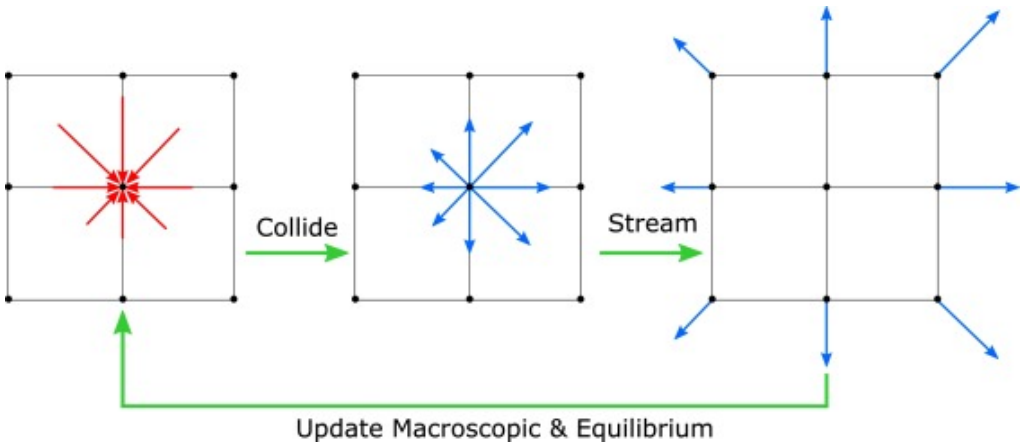


Figure 1.1: Schematic Representation of Collision and streaming process

In the collision term of (5), $f_i^{eq}(\vec{x},t)$ is the equilibrium distribution, and is considered as the relaxation time towards local equilibrium. For simulating single phase ows, it suces to use BhatnagarGross-Krook (BGK) collision, whose equilibrium distribution f_i^{eq} is dened by

$$f_i^{eq}(\vec{x},t) = w_i\rho + \rho s_i(\vec{u}(\vec{x},t)) \tag{1.6}$$

where $s_i(\vec{u})$ is defineed as

$$s_i(\vec{u}) = w_i\left[3\frac{\vec{e_i}.\vec{u}}{c} + \frac{9}{2}\frac{(e_i.\vec{u})^2}{c^2} - \frac{3}{2}\frac{\vec{u}\vec{u}}{c^2}\right] \tag{1.7}$$

and the weights w_i is given by

$$w_i = \left\{\begin{array}{cc} 4/9 & \text{i} = 0 \\ 1/9 & \text{i} = 1,2,3,4 \\ 1/18 & \text{i} = 5,6,7,8 \end{array}\right\} \tag{1.8}$$

and c = $\frac{\Delta x}{\Delta t}$ is the lattice speed. The fluid kinematic viscosity ν in the D2Q9 model is related to the relaxation time τ by

$$nu = \frac{2\tau - 1}{6}\frac{(\Delta x)^2}{\Delta t} \tag{1.9}$$

The algorithm can be summarized as follows:

1. Initialize $\rho, \vec{u}, f_i, f_i^{eq}$
2. Streaming step: move f_i to f_i*
3. Compute macroscopic ρ and $\vec{u}$ from f_i* using (2.3) and (2.4)
4. Compute f_i using (2.6)
5. Collision step: calculate the updated distribution function $f_i = f_i - \frac{f_i - f_i^{eq}}{\tau}$
6. Repeat step 2-5

Notice that numerical issues can arise as $\tau \to 1/2$. During the streaming and collision step, the boundary nodes require some special treatments on the distribution functions in order to satisfy the imposed macroscopic boundary conditions. We discuss these in details in Section 3.
Here are the matlab codes to understand how to implement streaming, collision and calculation of macroscopic variables in LBM BGK model

1.3 Boundary conditions

Boundary conditions (BCs) are central to the stability and the accuracy of any numerical solution. For the lattice Boltzmann method, the discrete distribution functions on the boundary have to be taken care of to reflect the macroscopic BCs of the fluid. In this project, we explore two of the most widely used BCs: Bounce-back BCs [4] and Zou-He velocity and pressure (density) BCs [5].

1.3.1 Bounce Back Boundary Conditions

Bounce-back BCs are typically used to implement no-slip conditions on the boundary. By the so-called bounce-back we mean that when a fluid particle (discrete distribution function) reaches a boundary node, the particle will scatter back to the fluid along with its incoming direction. Bounce-back BCs come in a few variants and we focus on two types of implementations: the on-grid and the mid-grid bounce-back [4].
The idea of the on-grid bounce-back is particularly simple and preserves a decent numerical accuracy. In this configuration, the boundary of the fluid domain is aligned with the lattice points (see Figure 3). One can use a boolean mask for

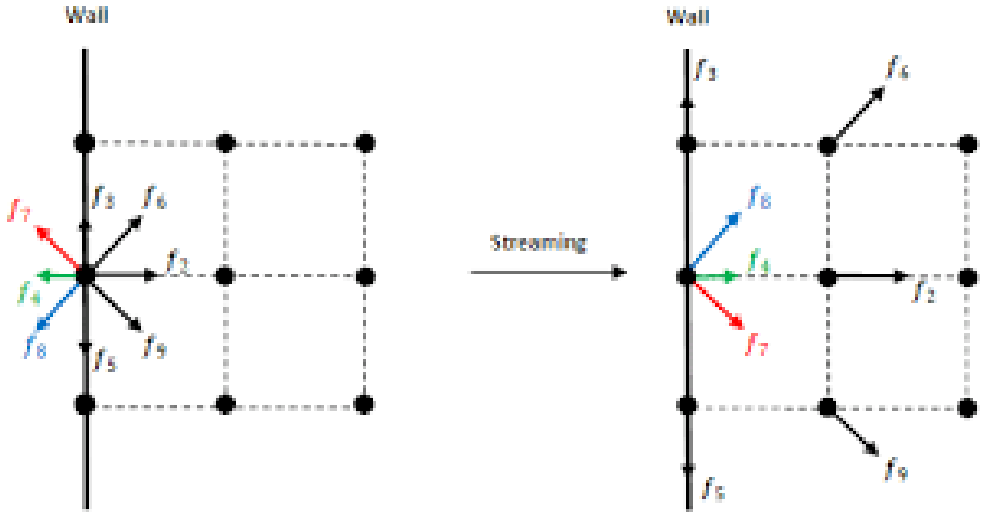


Figure 1.2: Schematic Representation of Nodes at various Boundaries

the boundary and the interior nodes. The incoming directions of the distribution functions are reversed when encountering a boundary node. This implementation does not distinguish the orientation of the boundaries and is ideal for simulating fluid flows in complex geometries, such as the porous media flow.

1.3.2 Zou-He Velocity and Pressure BCs

In many physical situations, we would like to model flows with prescribed velocity or pressure (density) at the boundary. This particular velocity/pressure BC we discuss here was originally developed by Zou and He. For illustration, we consider that the velocity $\vec{u_L} = (u, v)$ is given on the left boundary. After streaming, f_0, f_2, f_3, f_4, f_6 and f_7 are known. Whats left undetermined are f_1, f_5, f_8 and ρ (see Figure).

The idea of Zou-He BCs is to formulate a linear system of f_1, f_5, f_8 and ρ using (3) and (4). After rearranging:

$$f_1 + f_5 + f_8 = \rho - (f_0 + f_2 + f_3 + f_4 + f_6 + f_7) \tag{1.10}$$

$$f_1 + f_5 + f_8 = \rho u + (f_3 + f_6 + f_7) \tag{1.11}$$

$$f_5 - f_8 = \rho v - f_2 + f_4 - f_6 - f_7 \tag{1.12}$$

By considering (2.10) and (2.11), we can determine

$$\rho = \frac{1}{u}[f_0 + f_2 + f_4 + 2(f_3 + f_6 + f_7)] \tag{1.13}$$

However, we need a fourth equation to close the system and solve for f_1, f_5 and f_8. The assumption made by Zou and He is that the bounce-back rule still holds for the non-equilibrium part of the particle distribution normal to the boundary. In this case, the fourth equation is

$$f_1 - f_1^{eq} = f_3 - f_3^{eq} \tag{1.14}$$

With $_1$ solved by (2.6) and (2.14), f_5, f_8 are subsequently determined:

$$f_1 = \frac{2}{3}\rho v \tag{1.15}$$

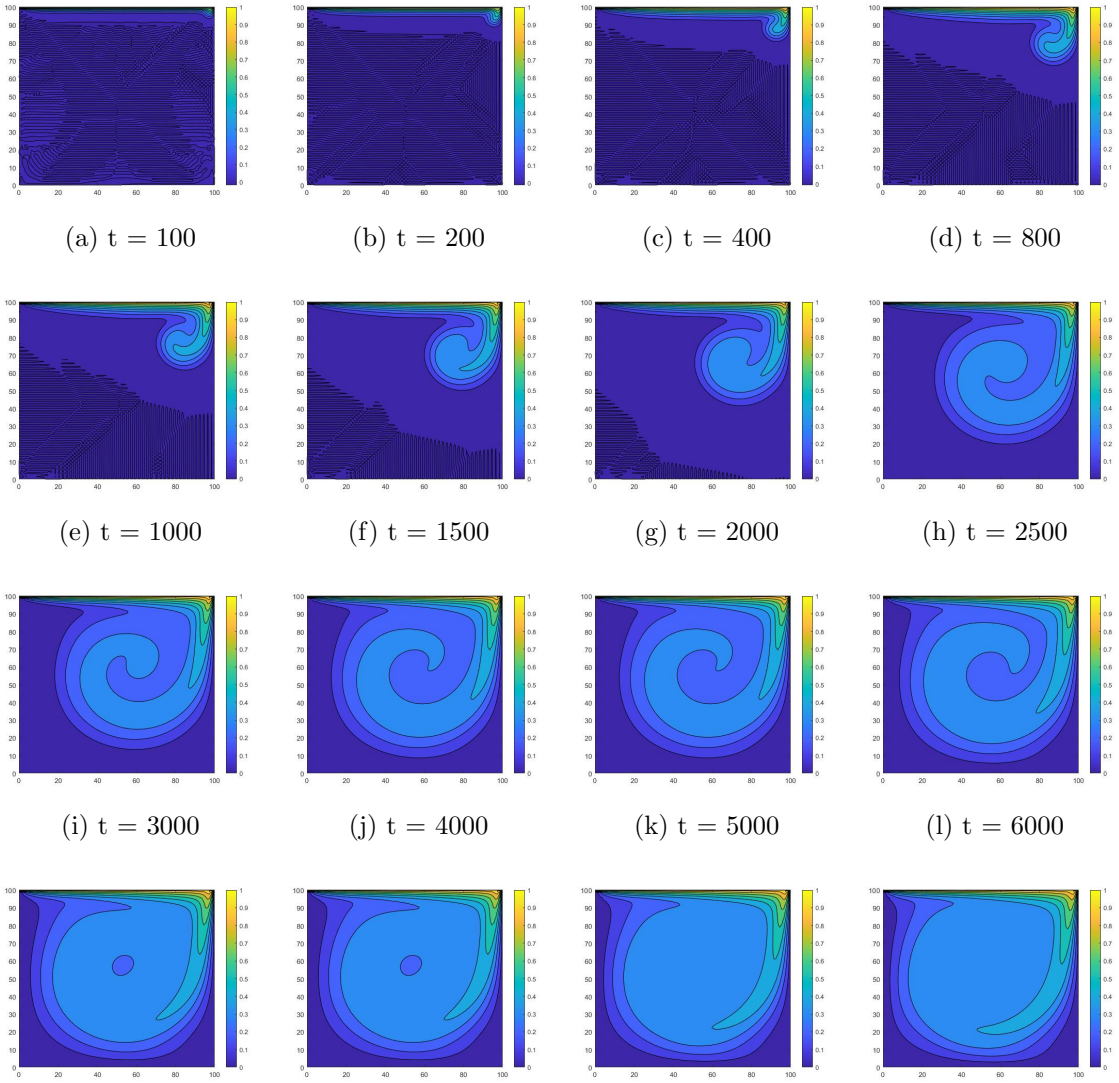
$$f_5 = f_7 - \frac{1}{2}(f_2 - f4) + \frac{1}{6}\rho u + \frac{1}{2}\rho v \tag{1.16}$$

$$f_8 = f_6 - \frac{1}{2}(f_2 - f4) + \frac{1}{6}\rho u - \frac{1}{2}\rho v \tag{1.17}$$

A similar procedure is taken if a given pressure (density) is imposed on the boundary. Here we notice that this type of BC depends on the orientation of the boundary and thus is hard to generalize for complex geometries.

1.3.3 Lid Driven Cavity Flow with heat transfer

The figures below display the results of lid-driven cavity flow with heat transfer. The heat transfer takes place from the heated top wall and the flow of the fluid occurs through Natural Convection. Please refer to the code in the appendix for more help.



(a) t = 100 (b) t = 200 (c) t = 400 (d) t = 800

(e) t = 1000 (f) t = 1500 (g) t = 2000 (h) t = 2500

(i) t = 3000 (j) t = 4000 (k) t = 5000 (l) t = 6000

(m) t = 7000 (n) t = 8000 (o) t = 9000 (p) t = 10000

Figure 1.3: Temperature field at various time steps

Chapter 2

Multiphase flows

This chapter presents the theory and applications of a multiphase model of the lattice Boltzmann method (LBM), the pseudopotential model proposed by Shan and Chen (1993) [1], which has been successfully applied to a wide range of multiphase flow problems during the past two decades. It begins with a description of the LBM and the original pseudopotential model. The distinct features and the limitations of the original model are described in detail. Then various enhancements necessary to improve the pseudopotential model in terms of decreasing the spurious currents, obtaining high density/viscosity ratio, reducing thermodynamic inconsistency, unraveling the coupling between surface tension and equations of state (EOS), and unraveling the coupling between viscosity and surface tension, are reviewed. Then the fluidsolid interactions are presented and schemes to obtain different contact angles are discussed. The final section of this part focuses on the multi-component multiphase pseudopotential model. The second part of this review describes fruitful applications of this model to various multiphase flows. Coupling of this model with other models for more complicated multiple physicochemical processes are also introduced in this part

2.1 Lattice Boltzmann Method

In the LB method, the motion of fluidis described by a set of particle distribution functions. Based on the simple and popular BhatnagarGrossKrook (BGK) collision operator [3], the standard LB equation with a force term can be expressed as follows

$$f_{\sigma,\alpha}(\mathbf{x}+c\mathbf{e}_\alpha\Delta t, t+\Delta t)-f_{\sigma,\alpha}(\mathbf{x},t) = -\frac{1}{\tau_{\sigma,\nu}}(f_{\sigma,\alpha}(\mathbf{x},t)-f_{\sigma,\alpha}(\mathbf{x},t)^{eq})+F_{\sigma,\alpha}(\mathbf{x},t) \quad (2.1)$$

where $f_{\sigma,\alpha}(\mathbf{x},t)$ is the density distribution function of the component σ at the lattice site $\mathbf{x}$ and time t, f^{eq} is the equilibrium distribution function, $c = \frac{\Delta x}{\Delta t}$ is the lattice speed with Δx and Δt as the lattice spacing and time step (Both x and Δt are equal to 1 in the lattice system), respectively, and τ_ν is the dimensionless relaxation time. The left-hand side (LHS) of Eq. (3.1) represents the streaming step whereas the right-hand side (RHS) stands for the collision operator which leads the system to the local Maxwellian equilibrium on a time scale τ.

2.2 The original Shen-Chen pseudopotential method

In order to introduce non local interaction among particles, Shan and Chen [1,2] defined the force experienced by the particles of component σ at $\mathbf{x}$ from the particles σ at $\mathbf{x}'$ as the following form

$$\mathbf{F}(\mathbf{x},\mathbf{x}') = -G(|\mathbf{x}-\mathbf{x}'|)\psi_\sigma(\mathbf{x})\psi_\sigma(\mathbf{x}')((\mathbf{x}-\mathbf{x}') \tag{2.2}$$

where G is a Greens function and w is an effective mass depending on the local density. The components σ and σ can be different or the same. Here we first discuss the single-component system. After it is thoroughly discussed we then move to multi-component system. The structure of the force along the vector between two lattice locations given by Eq. (2.7) is delicately designed. It is perhaps the most general one that meets the Newtons third law and conserves momentum globally [4]. The force is basically pairwise point coupling between the effective mass at the location $\mathbf{x}$ and that at the neighboring location $\mathbf{x}'$. This type of interaction force among the particles does not conserve the local momentum during the collision process because the force behaves as an external force acting on the site. However, it has been proved that the total momentum of the system is conserved and no net momentum is introduced into the system [2]. The total force acting on the particles at $\mathbf{x}$ is thus the sum of all the interaction forces

$$\mathbf{F}(\mathbf{x},\mathbf{x}') = \psi_\sigma(\mathbf{x}')\sum -G(|\mathbf{x}-\mathbf{x}'|)\psi_\sigma(\mathbf{x})(\mathbf{x}-\mathbf{x}') \tag{2.3}$$

In the lattice space system, if we consider N neighbor sites that interact with the current site $\mathbf{x}$ and let $G(|\mathbf{e}_\alpha|)$ be a function of $\mathbf{e}_\alpha$ only (which indicates that the interaction is isotropic), the interaction force can be further expressed as

$$\mathbf{f}(\mathbf{x},\mathbf{x}') = -g\psi(\mathbf{x})c_s^2\sum_{\alpha=1}^{N} w(|{e_\alpha}^2|)\psi(\mathbf{x}+\mathbf{e}_\alpha)\mathbf{e}_\alpha \tag{2.4}$$

where g is the interaction strength. $w(|\mathbf{e}_a|^2)$ are the weights used for the calculation of the isotropic interaction force, and it is different from that in Eq. (3).
There are three important elements in the force given by Eq. (9): the interaction strength g, the effective mass ψ, and the determination of the neighbor nodes. The g controls the interaction strength between particles, with a positive (negative) value leading to a repulsive (attractive) force between particles. The original form of effective mass in the work of Shan and Chen is [1,2]

$$\psi = \rho_0 e^{(1-\frac{\rho}{\rho_0})} \tag{2.5}$$

where ρ_0 is a normalization constant which is usually chosen as 1. This form of effective mass reduces to density ρ itself when ρ is low and reaches a saturation value at high density, as can be seen in Fig. 2. In Fig. 2, the EOS corresponding to Eq. (2.10) is named Shan - Chen EOS The feature of approaching a saturation value at high density helps prevent density collapse of the phase with high density and thus can increase the simulation stabilities [5,6]. Besides, such an effective mass can produce an exponentially vanishing force as the density increases.
Through Taylor expansion, the leading terms of the interaction force expressed by Eq. (9) can be obtained as follows [4]

$$\mathbf{F} = -g(c_s^2\psi\nabla\psi + \frac{c_s^4}{2}\psi\nabla(\Delta\psi) + ...) \tag{2.6}$$

2.3 Calculation of the interaction Force

For most of the applications of Shan and Chen pseudopotential model, only the nearest and next-nearest nodes are considered in the interaction force. For the nearest nodes, namely nodes in the lattice directions 1, 2, 3, 4 as shown in Fig. 1, the weight $w(|\mathbf{e}_\alpha|^2)$ in Eq. (9) is w(1) = 1/3, and for the next-nearest one, namely nodes in the lattice direction 5, 6, 7 and 8, the weight $w(|\mathbf{e}_\alpha|^2)$ is w(2) = 1/12 [30]. Therefore, for the current computational node (i, j), Eq. (9) can be written as

$$\begin{aligned} F_x = -g\psi(i,j)c_s^2[&\frac{1}{3}(\psi(i+1,j)-\psi(i-1,j)) + \frac{1}{12}(\psi(i+1,j+1)-\psi(i-1,j+1)) \\ &+ \frac{1}{12}(\psi(i+1,j-1)-\psi(i-1,j-1))] \end{aligned} \tag{2.7}$$

$$F_x = -g\psi(i,j)c_s^2\big[\frac{1}{3}(\psi(i+1,j)-\psi(i-1,j)) + \frac{1}{12}(\psi(i+1,j+1)-\psi(i-1,j+1)) + \frac{1}{12}(\psi(i+1,j-1)-\psi(i-1,j-1))\big] \tag{2.8}$$

where F_x and F_y are the force in the x and y direction, respectively.c_s is the speed of sound in lattice units and is calculated as $c_s = \frac{c}{\sqrt{3}}$ where $c = \frac{dx}{dt}$ and is usually taken as 1 in simulations. Calculating the interaction force this way yields an isotropy order of 4 and hence is called E4 force scheme.

2.4 Initialization of density field

In the simulation, a liquid droplet with initial radius r_0 is placed at the center of the domain. The density field is initialized using the method in Ref. [7].

$$\rho(i,j) = \frac{\rho_{liquid} + \rho vapour}{2} + \frac{\rho_{liquid} - \rho vapour}{2} \times tanh\big[\frac{2(\sqrt{(i-i_{center})^2 + (i-i_{center})^2}}{w}\big] \tag{2.9}$$

where $(_{center}, j_{center})$ is the center position of the domain. $'tanh'$ is the hyperbolic tangent function and $tanh(x) = (e^{2x}-1)/(e^{2x}+1)$. w is the prescribed width of the phase interface and is usually varied by 1-5 lattices.

2.5 Velocity shift force scheme

The velocity shift force scheme is the original force scheme that Shan and Chen used [1]. In this scheme, the interaction force is incorporated into the model by shifting the velocity $\mathbf{u}$ in the equilibrium distribution function, with which $\mathbf{u}$ is replaced by $\mathbf{u}^{eq}$

$$\mathbf{u}^{eq} = \mathbf{u} + \frac{\tau}{\rho}\mathbf{F}\Delta t \tag{2.10}$$

where $\mathbf{u}$ is given by Eq. (2.4). By averaging the momentum before and after the collision step, the actual physical velocity is given by

$$\mathbf{u}_p = \mathbf{u} + \frac{1}{2\rho}\mathbf{F}\Delta t \tag{2.11}$$

Note that the fore term in the RHS of Eq. (3.1) is not needed when the velocity shift force scheme is adopted. This velocity shift force scheme is a unique feature of the original pseudopotential model.

2.6 Boundary conditions

The computational domain consists of three air walls on the sides and top and one solid wall at the bottom. Bounce back Boundary conditions were used at all the boundaries while the solid-liquid interaction force was taken care as a boundary condition in the density field. The density field at the solid boundary is given by

$$\rho_{boundary} = 0.3(\rho_{liquid} - \rho_{vapor}) \tag{2.12}$$

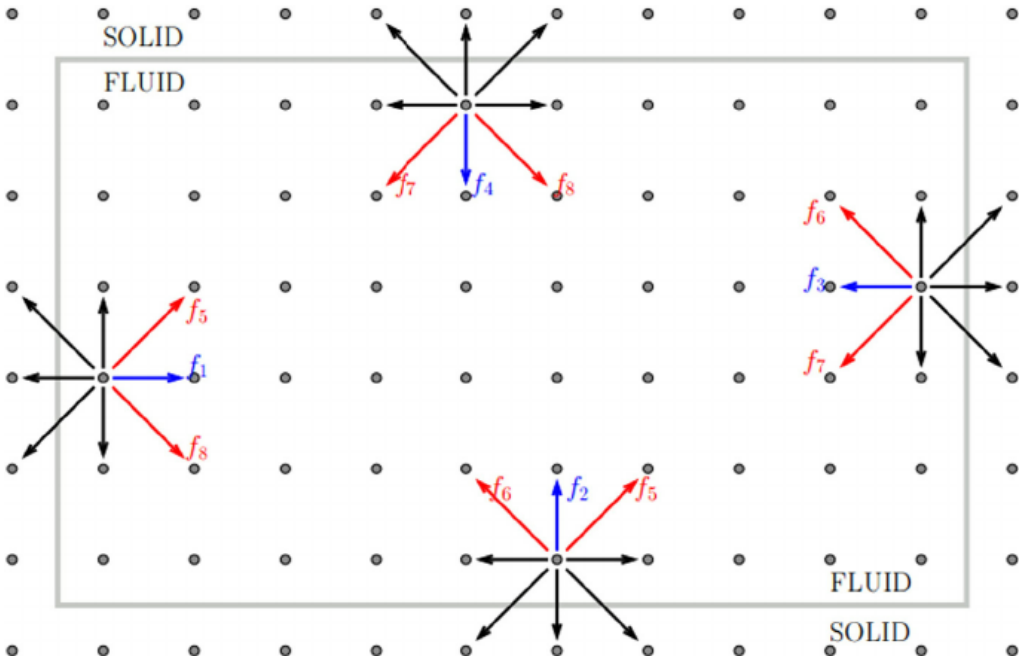


Figure 2.1: Schematic Representation of BounceBack condition

2.6.1 Boundary condition at the bottom solid wall

At the bottom solid wall, the distribution functions f_0, f_1, f_3, f_4, f_7 and f_8 are known while f_2, f_5, f_6 and ρ are unknown. Since, the velocities are zero near the walls, From Eq. (2.15-2.17), we can find the unknown distribution functions by

$$f_2 = f_4 \tag{2.13}$$

$$f_5 = f7 \tag{2.14}$$

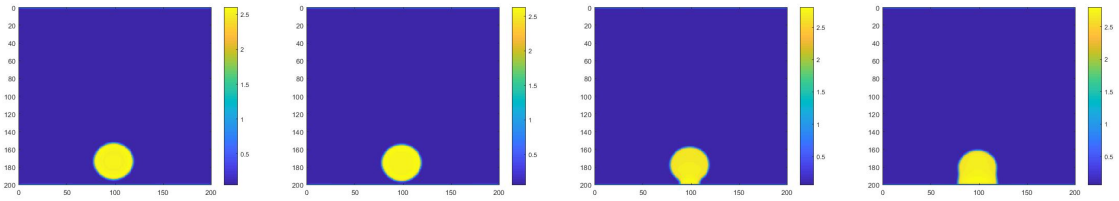
$$f_6 = f8 \tag{2.15}$$

2.7 Results

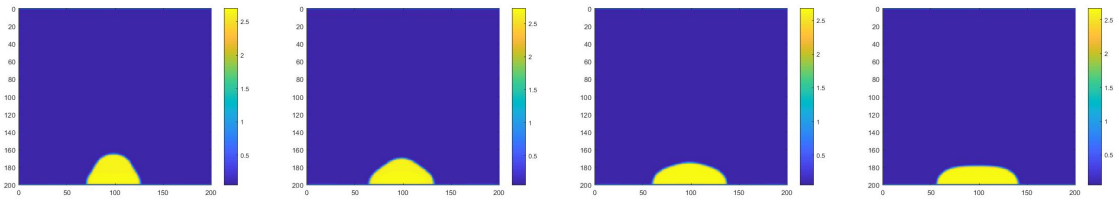
The simulation was carried in MATLAB environment with grid size 202×202 lu^2 . An unknown liquid of density 2.64 in lattice units of density, initial radius of 20 lu was dropped in the surrounding vapor of density 0.0734 lattice units of density. The initial centre was chosen to be $i_{center} = 101$, $j_{center} = 176$. The simulation was carried out for a total of 10000 time steps. The table below provides all the values of the physical constants used.

Physicalconstants	**Valuesinlatticeunits**
ρ_{liquid}	2.64
ρ_{vapor}	0.0734
ν	0.02
$force_{body}$	10^{-4}
fluid-fluid Interaction Potential (G)	-6
solid-fluid interaction Potential (H)	-0.3

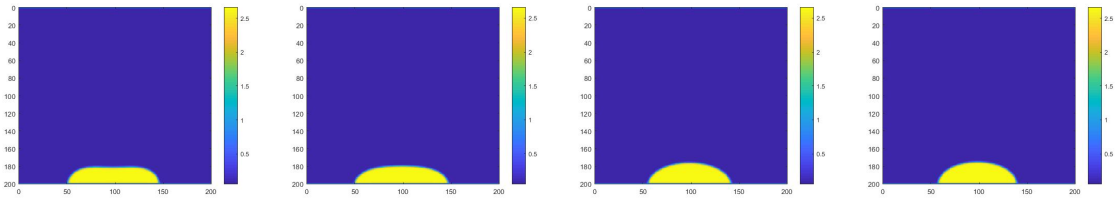
The figure 2.2 shows the density field distribution of the water droplet moving downward towards the wall and its subsequent deformation. The vapor is indicated in dark blue color and the droplet is indicated in yellow color. As you can see with the increase in time steps, the droplet first spreads to form a liquid film and then shrink back to form a droplet. The shape of the droplet changes dramatically during the initial time steps of t<1500, while its deformation nearly stops after t>1500.



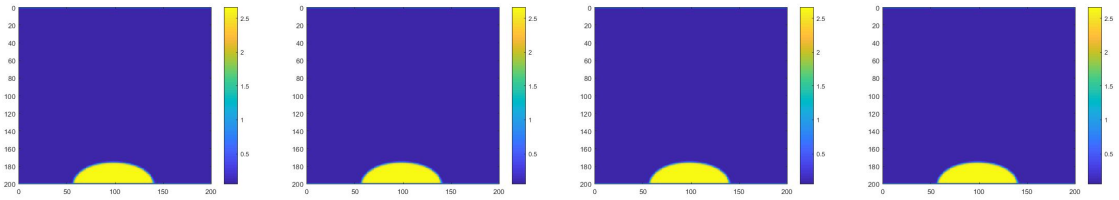
(a) t = 100 (b) t = 200 (c) t = 300 (d) t = 400



(e) t = 500 (f) t = 600 (g) t = 700 (h) t = 800



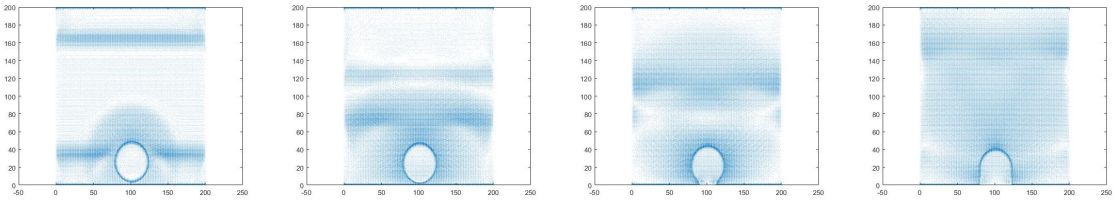
(i) t = 1000 (j) t = 1500 (k) t = 2000 (l) t = 3000



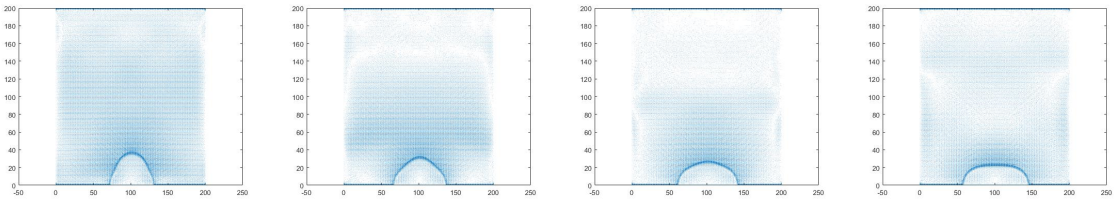
(m) t = 4000 (n) t = 6000 (o) t = 8000 (p) t = 10000

Figure 2.2: Density field of the droplet at various time steps

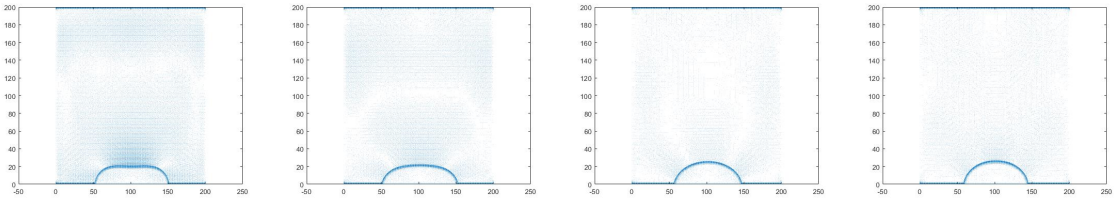
The figures 2.3 and 2.4 show the velocity inside the droplet and the near surroundings as the droplet falls under the influence of the body force and the fluid-fluid interaction.



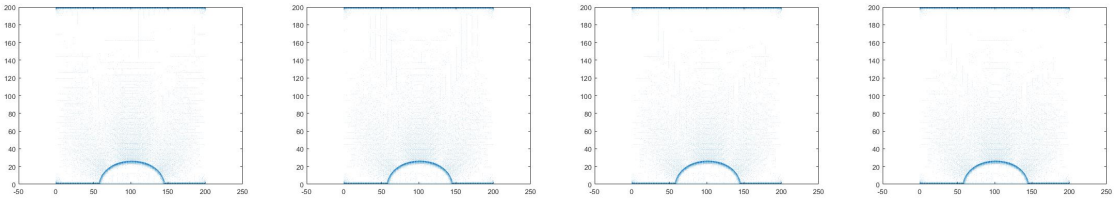
(a) t = 100 (b) t = 200 (c) t = 300 (d) t = 400



(e) t = 500 (f) t = 600 (g) t = 700 (h) t = 800

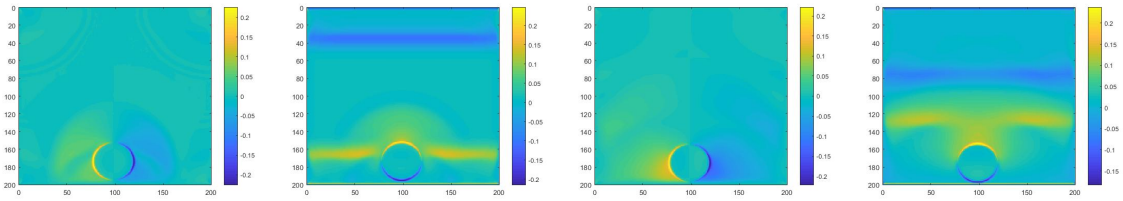


(i) t = 1000 (j) t = 1500 (k) t = 2000 (l) t = 3000

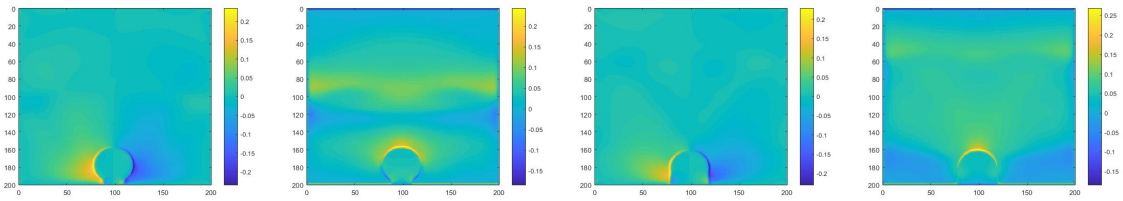


(m) t = 4000 (n) t = 6000 (o) t = 8000 (p) t = 10000

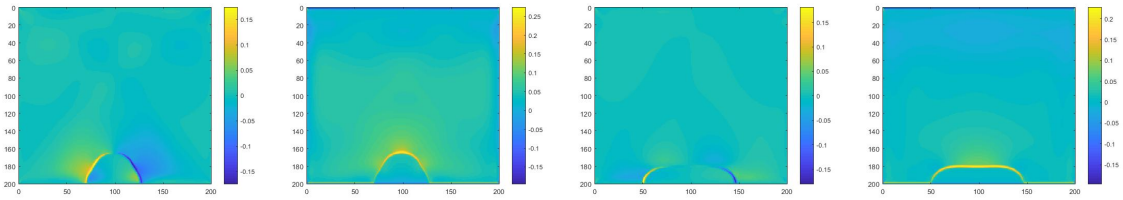
Figure 2.3: Velocity quivers at different time steps



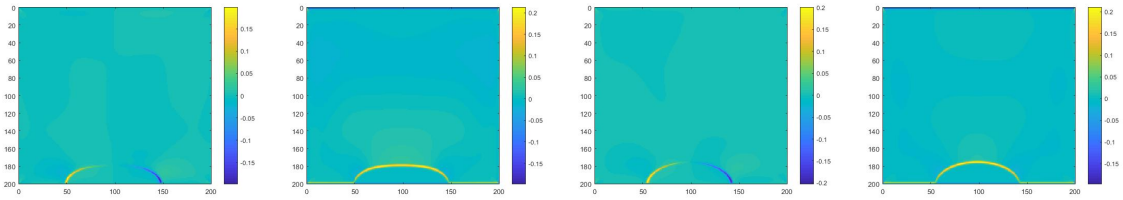
(a) u at t = 100 (b) v at t = 100 (c) u at t = 200 (d) v at t = 200



(e) u at t = 300 (f) v at t = 300 (g) u at t = 400 (h) v at t = 400



(i) u at t = 500 (j) v at t = 500 (k) u at t = 1000 (l) v at t = 1000



(m) u at t = 1500 (n) v at t = 1500 (o) u at t = 2000 (p) v at t = 2000

Figure 2.4: X and Y velocity distibution at different time steps. Note 'u' denotes the velocity in X-direction and 'v' denotes the velocity in Y-direction.

2.8 Conclusion

The dynamics of the falling droplet under the effect of gravity in a fluid medium is studied. The Lattice Boltzmann Method was utilised for modelling the Multi-phase flow. Shan-Chen pseudopotential model was used to solve the multi-phase flow dynamics and the fourth order isotropy (E4) of descretization was applied for modelling the force term. Our model results agrees well with the results from the

reference paper cited below. Further study includes application of Equation of states (EOS) and the adhesion-cohesion forces in the model to make it more physical.

2.9 References

[1] X. Shan, H. Chen, Lattice Boltzmann model for simulating flows with multiple phases and components, Phys. Rev. E 47 (1993) 18151819.
[2] X. Shan, H. Chen, Simulation of nonideal gases and liquidgas phase transitions by the lattice Boltzmann equation, Phys. Rev. E 49 (4) (1994) 29412948.
[3] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems, Phys. Rev. 94 (3) (1954) 511525.
[4] X. Shan, Pressure tensor calculation in a class of nonideal gas lattice Boltzmann models, Phys. Rev. E 77 (6) (2008) 066702.
[5] M. Sbragaglia, H. Chen, X. Shan, S. Succi, Continuum free-energy formulation for a class of lattice Boltzmann multiphase models, EPL 86 (2009) 24005.
[6] M. Sbragaglia, K. Sugiyama, L. Biferale, Wetting failure and contact line dynamics in a Couette flow, J. Fluid Mech. 614 (2008) 471493.
[7] H. Huang, M. Krafczyk, X. Lu, Forcing term in single-phase and ShanChentype multiphase lattice Boltzmann models, Phys. Rev. E 84 (4) (2011) 046710.

Chapter 3

Lattice Boltzmann Method and phase transition

The transient heat conduction problem involving a change of phase is of particular interest in many engineering problems, such as the casting of metals, melting and solidification of alloys, artificial crystal growth from melts, and storage of energy. Due to the inherent non linearities, mainly caused by formation and propagation of the moving phase-change interface, the analytical solutions to these problems are limited to problems with simple geometry and boundary conditions [1-3]. Numerical methods are, therefore, more frequently employed in solving such phase-change problems and have led to impressive results [4-10].
An extended lattice Boltzmann (LB) equation was developed for the simulation of the phase-change problem (Stefan's problem) governed by the heat conduction equation incorporated with enthalpy formation. Illustrative examples include one-dimensional solidification as well as two-dimensional corner solidification. The walls are treated as a constant temperature boundaries at sub-cryogenic temperatures. Effect of relaxation times on the freezing rates is also depicted. All numerical results obtained by the present scheme agree very well with previous analytical or numerical results in the literature.

3.1 Problem Definition

The problem is named after Josef Stefan (Joef Stefan), the Slovenian physicist who introduced the general class of such problems around 1890, in relation to problems of ice formation.[5].

The problem can be viewed as a fixed and finite volume of liquid in a con-

tainer at room temperature while the walls of container are at a temperature lower than the freezing point of the liquid (cryogenic temperature). In one-dimensional solidification problem, only one wall is kept at cryogenic temperature while for two-dimensional solidification, the adjacent walls are kept at cryogenic temperatures. The solidification occurs directionally and the growth of the ice occurs perpendicular to the walls. The flow of liquid occurs due to natural convection in the container due to temperature difference and no external force is applied.

3.2 Enthalpy formation during Phase change

In the enthalpy formulation approach, the energy equation is initially expressed in terms of the total enthalpy. The governing equation for a heat conduction problem can thus be cast in the form

$$\frac{\partial \rho H}{\partial t} = \nabla.(k \nabla T) \tag{3.1}$$

The total enthalpy, H, is further split into the sensible enthalpy and the latent heat for the phase-change problem [4]. In this manner, the phase boundary can be solved naturally and determined as part of the solution [5]. H can be written as

$$H = CT + f_l L \tag{3.2}$$

where C and L represent the constant-pressure specific heat and the latent heat of phase change; f_l is the volume-phase fraction of the liquid phase and is zero for the solid region and unity for the liquid region. The phase fraction lies between zero and unity when the region being considered is undergoing phase change. Substituting Eq. (4.2) into Eq. (4.1) yields

$$\frac{\partial (\rho C T)}{\partial t} = \nabla.(k \nabla T) - L \frac{\partial (\rho f_l)}{\partial t} \tag{3.3}$$

where the latent heat appears as a heat source term in the governing equation. Once the enthalpy is determined, the solid phase fraction is then defined as

$$f_l = \frac{H_l - H}{H_l - H_s} \tag{3.4}$$

3.3 Lattice Boltzmann Method

3.3.1 LBM for velocity field

The fluid velocities arise due to Natural convection in the container due to temperature differences along the propagation of ice. The theory and implementation of

distribution function and calculation of macroscopic variables ($\rho and \mathbf{u}$) is explained in detail in (2.3.6).

3.3.2 LBM for temperature field

LBM models the temperature field consisting of fictive particles, which streams and collies over a discrete lattice mesh. Figure shows the Cartesian lattice and the velocities e_i where i = 0, 1 8 are a direction index and $e_0 = 0$ denotes particles at rest. This model is known as D2Q9 as it is 2 dimensional and contains nine directions. The lattice unit (lu) is the fundamental measure of length in the LBM models and time steps (ts) are the time unit. Discretized lattice Boltzmann equation is written as:

$$g_i(\mathbf{x} + c\mathbf{e_i}\delta t, t + \delta t) - g_i(\mathbf{x}, t) = \Omega_i \tag{3.5}$$

where g_i is the distribution function for the temperature which have discrete velocities indicated e_i defined in Eq. (1.2) and $c = \frac{dx}{dt}$ is the lattice speed. For simplicity we assume that $dx = dt = 1$. Right hand side of Eq. (4.5) includes the collision term, and BGK approximation is used for evaluate this term as follows:

$$\Omega_i = -\frac{g_i(\mathbf{x}, t) - g_i(\mathbf{x}, t)^{eq}}{\tau} \tag{3.6}$$

In Eq. (3.6) τ is the relaxation time and g_i^{eq} is the equilibrium distribution function. The equilibrium distribution function, g_i^{eq} is evaluated by

$$g_i^{eq} = w_i T(i,j)\left[1 + 3\frac{e_i.\mathbf{u}}{c^2} + \frac{9}{2}\frac{(e_i.\mathbf{u})^2}{c^4} - \frac{3}{2}\frac{\mathbf{u}.\mathbf{u}}{c^4}\right] \tag{3.7}$$

where T is the macroscopic temperature, and w_i are weight factors in D2Q9 lattice and are presented in Eq. (1.8).
According to Yu et al. [12] the above equation can be explicitly divided into two parts. One part refers to streaming and the other one addresses the collision. For collision step we can write:

$$\tilde{g}_i(\mathbf{x}, t + \delta t) - g_i(\mathbf{x}, t) = -\frac{g_i(\mathbf{x}, t) - g_i(\mathbf{x}, t)^{eq}}{\tau} \tag{3.8}$$

and for the streaming step we have:

$$g_i(\mathbf{x} + c\mathbf{e_i}t, t + \delta t) = \tilde{g}_i(\mathbf{x}, t + \delta t) \tag{3.9}$$

Eq.(3.9) yields quantities of g_i for adjacent areas after one moment. Fluid thermal diffusivity is given by relaxation time and the lattice sound velocity, as the

following form:

$$\tau = \frac{3}{c^2}\alpha + \frac{1}{2} \tag{3.10}$$

For positive values of thermal diffusivity, it is necessary that the relaxation time parameter, τ be more than 0.5. But stability conditions forces that its value has to be enough larger than 0.5. Finally macroscopic temperature at each point will be calculated by the following equation

$$T(i,j) = \sum_i g_i \tag{3.11}$$

3.4 Latent heat as source term

The obtained temperature field does not correspond to the heat source term. For the solidification/melting problem, the latent heat behaves as a source term and must be taken into account in the heat equation. Eshraghi and Felicelli [11] introduced a novel implicit LBM to take into account the source term. In contrast to previous explicit algorithms, their model considers the latent heat source implicitly in the energy equation, avoiding iteration steps and improving the formulation stability and efficiency. Energy equation with a source term for constant thermoephysical properties is expressed as:

$$\frac{\partial T}{\partial t} = \nabla.(\alpha \nabla T) - \psi \tag{3.12}$$

where ψ denotes the thermal sources. For the phase-change problem ψ is calculated as:

$$\psi = \frac{L}{C_p}\frac{\partial f_l}{\partial t} \tag{3.13}$$

Eq. (3.4) gives a linear approximation for liquid fraction where the fluid enthalpy is between H_l and H_s (or the fluid temperature is between liquidus (T_l) and solidus (T_S) temperature). To use Eq. (4.4), it is necessary that the values of T_l and T_s differ slightly for numerical stability. In this paper the liquidus and solidus temperatures are set to 1.005 and 0.995, respectively. The final step for this section is correcting the collision step respect to the source term. It is shown that the collision term for a specific particle can be written as [11]:

$$g_i(\mathbf{x} + c\mathbf{e_i}\delta t, t + \delta t) - g_i(\mathbf{x}, t) = -\frac{g_i(\mathbf{x}, t) - g_i(\mathbf{x}, t)^{eq}}{\tau_T} - \Delta t w_i \psi \tag{3.14}$$

3.5 Boundary Conditions

Constant temperature boundary condition was applied at the walls through which the solidification occured. At the other walls, bounceback boundary conditions were applied. Here, we explain it in the context of D2Q9 lattice.

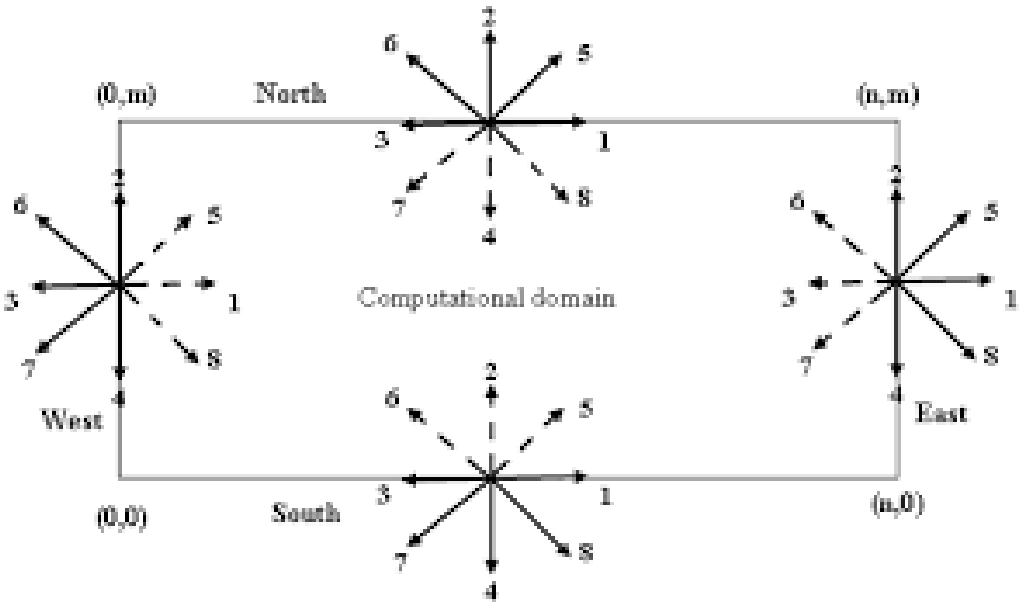


Figure 3.1: Schematic Representation of Nodes at various Boundaries

3.5.1 Isothermal wall

Suppose that the bottom wall shown in Figure is fixed in a constant temperature T_b. After streaming, the values of g_2, g_5 and g_6 are unknowns. Assume these unknown particle distribution functions (PDFs) are equal with their equilibrium distribution at some unknown temperature T'. Summing these three PDFs together, we have [13]:

$$g_2 + g_5 + g_6 = \frac{T'}{6} \tag{3.15}$$

Knowing T', we will be able to solve for g_2, g_5, and g_6. Meanwhile, it is noticed that for the isothermal wall, $\sum_{i=0}^{8} g_i = T_b$. Substituting Eq. (3.15) into this, T' then may be calculated as follows:

$$T' = 6 \times (T_b - (g_0 + g_1 + g_3 + g_4 + g_7 + g_8) \tag{3.16}$$

Finally, g_2, g_5 and g_6 may be obtained by substituting T' into $gi = T' \times w_i$.

3.6 Solution Procedure

The initial density field (ρ), temperature field (T), velocity field ($\mathbf{u}$) and solid fraction f_s are defined first along with other physical properties. The sensible, latent and total enthalpy is calculated using the above parameters. Then the distribution functions f and g are initialized using the known values of ρ, T and $\mathbf{u}$. Distribution functions are then evolved in a time loop according to the following algorithm:

Time loop:

{

1. Update temperature and solid-fraction fields
2. Impose boundary conditions
3. Calculate equilibrium DFs f_i^{eq} and g_i^{eq}
4. Calculate new values of DFs (Collision step) using previous time step values of velocity, temperature and solid fraction
5. Propagate fluid particles (Streaming)
6. Obtain density, velocity, temperature
7. Update nodal enthalpy
8. Calculate solid fraction from the updated enthalpy
9. Go back to step 2 until convergence
   }
10. Obtain macroscopic variables

The convergence criteria is chosen as:

$$min\left[\left|\frac{f_s^N - f_s^{N-1}}{f_s^N}\right|, \left|\frac{T^N - T^{N-1}}{T^N}\right|\right] < 10^{-8} \tag{3.17}$$

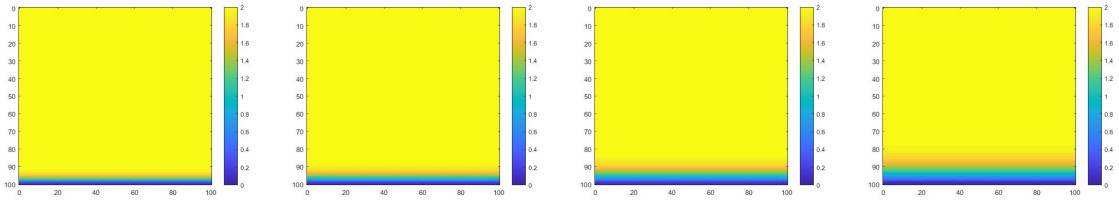
3.7 Results

The computational domain is a rectangular space of grid size 201×202. It contains fluid with density 1 (non dimensional temperature) and the melting point temperature $T_m = 1$ (lattice units). The wall is kept at a temperature $T_b = 0$. The gravitational force is kept at $g = 10^{-4}$. All the other physical properties and simulation parameters are given in the table below.

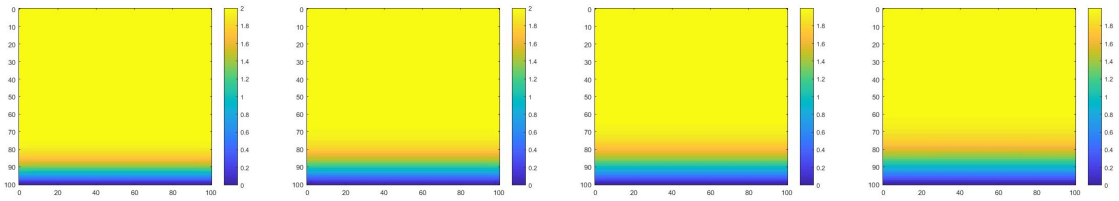
Physicalconstants	Valuesinlatticeunits
ρ_{liquid}	1
ρ_{ice}	0.9
ν	0.02
Pr	0.71
$\alpha = \frac{nu}{Pr}$	0.028
g	10^{-4}
Ra	10^3
T_0	2
T_m	1
T_s	0.995
T_l	1.005
T_b	0
Latent heat (L)	1
C_w	1
C_{ice}	0.5

3.7.1 One Dimensional solidification

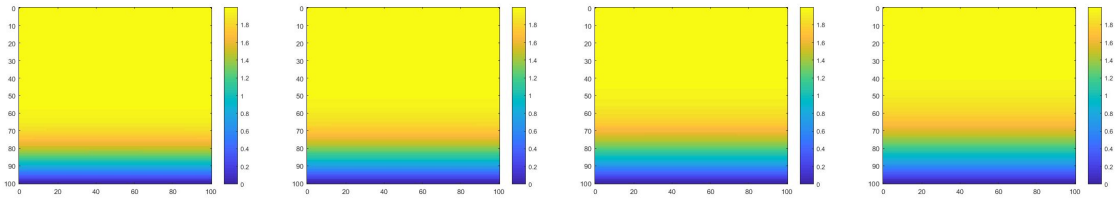
Figure 3.2 shows the temperature distribution with the time as the solidification starts at t=0. Figure 3.3 displays the growth of the solid with the progressing time. It can be noted that the growth of the ice occurs in the direction normal to the wall which concludes that the code follows the physics correctly. In figure 3.3, the yellow region indicates that the fluid is completely solidified and as the color moves towards the blue region, it shows that the fluid acts as a porous medium in that region.



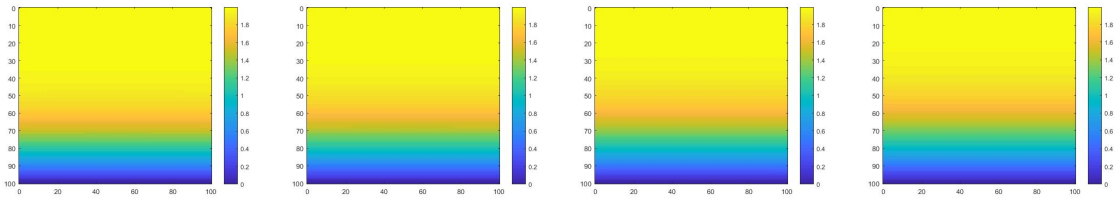
(a) t = 100 (b) t = 200 (c) t = 400 (d) t = 800



(e) t = 1000 (f) t = 1600 (g) t = 2000 (h) t = 2500



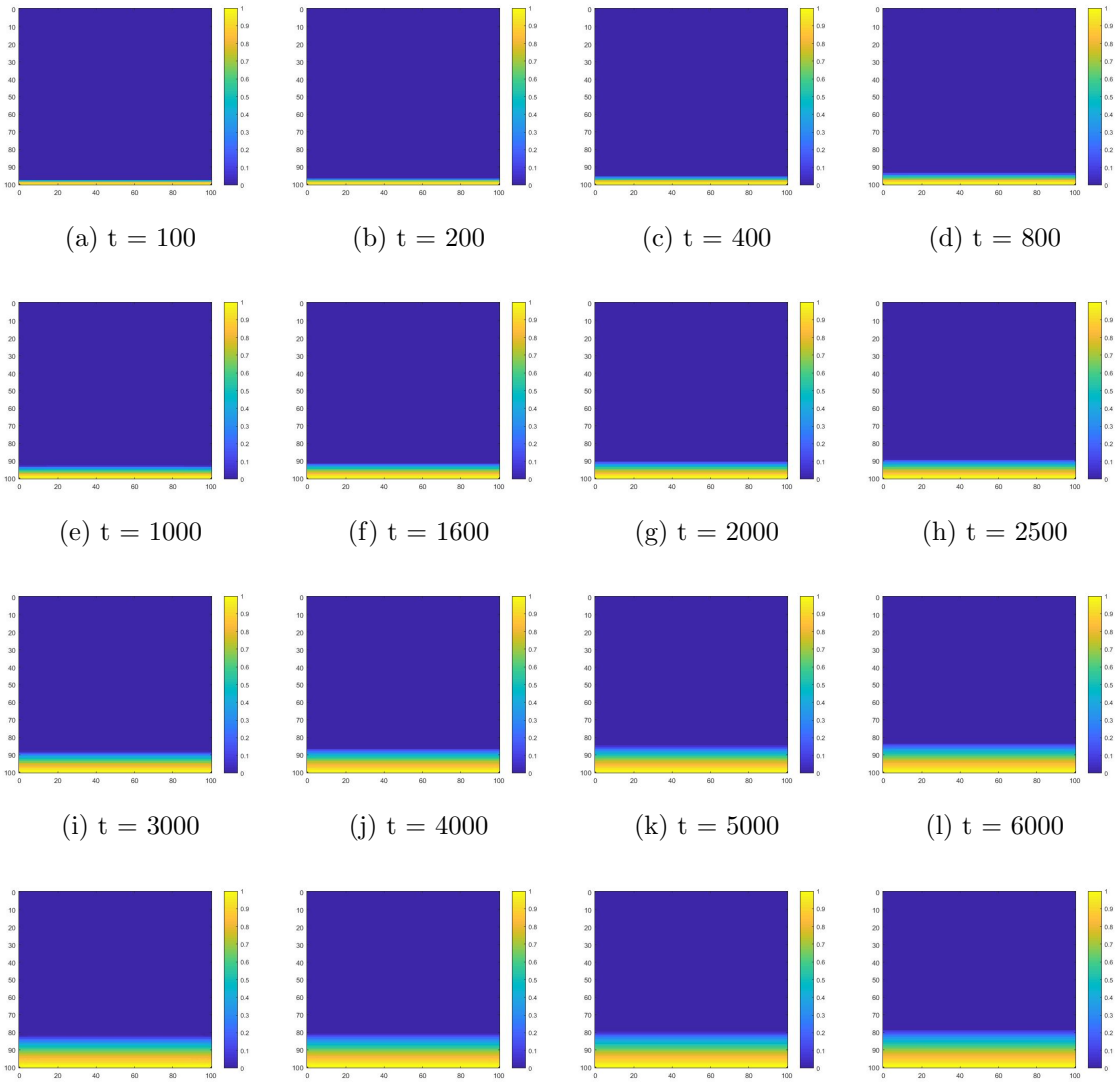
(i) t = 3000 (j) t = 4000 (k) t = 5000 (l) t = 6000



(m) t = 7000 (n) t = 8000 (o) t = 9000 (p) t = 10000

Figure 3.2: Temperature field at various time steps

3.7.2 One Dimensional solidification



(a) t = 100 (b) t = 200 (c) t = 400 (d) t = 800

(e) t = 1000 (f) t = 1600 (g) t = 2000 (h) t = 2500

(i) t = 3000 (j) t = 4000 (k) t = 5000 (l) t = 6000

(m) t = 7000 (n) t = 8000 (o) t = 9000 (p) t = 10000

Figure 3.3: Solid Fraction field at various time steps

Figures 3.4 and 3.5 display the variation of fluid temperature and solid fraction w

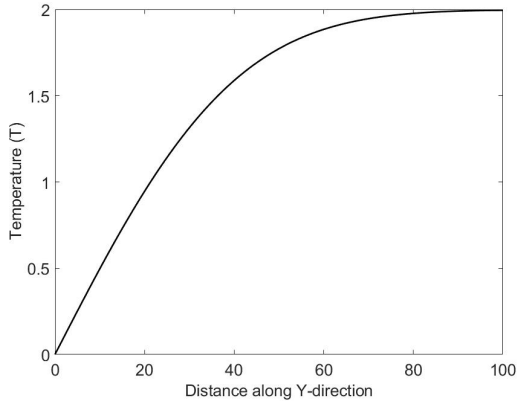


Figure 3.4: Temperature distribution through the line along Y-axis passing through the center of the bubble

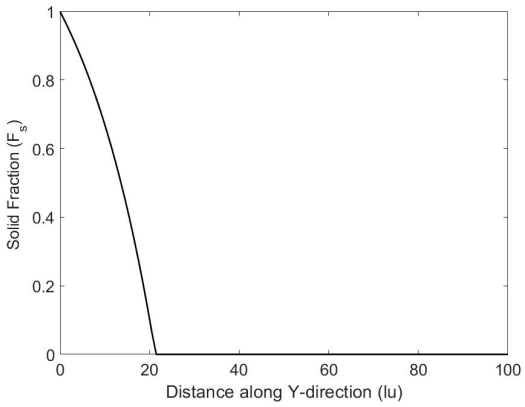
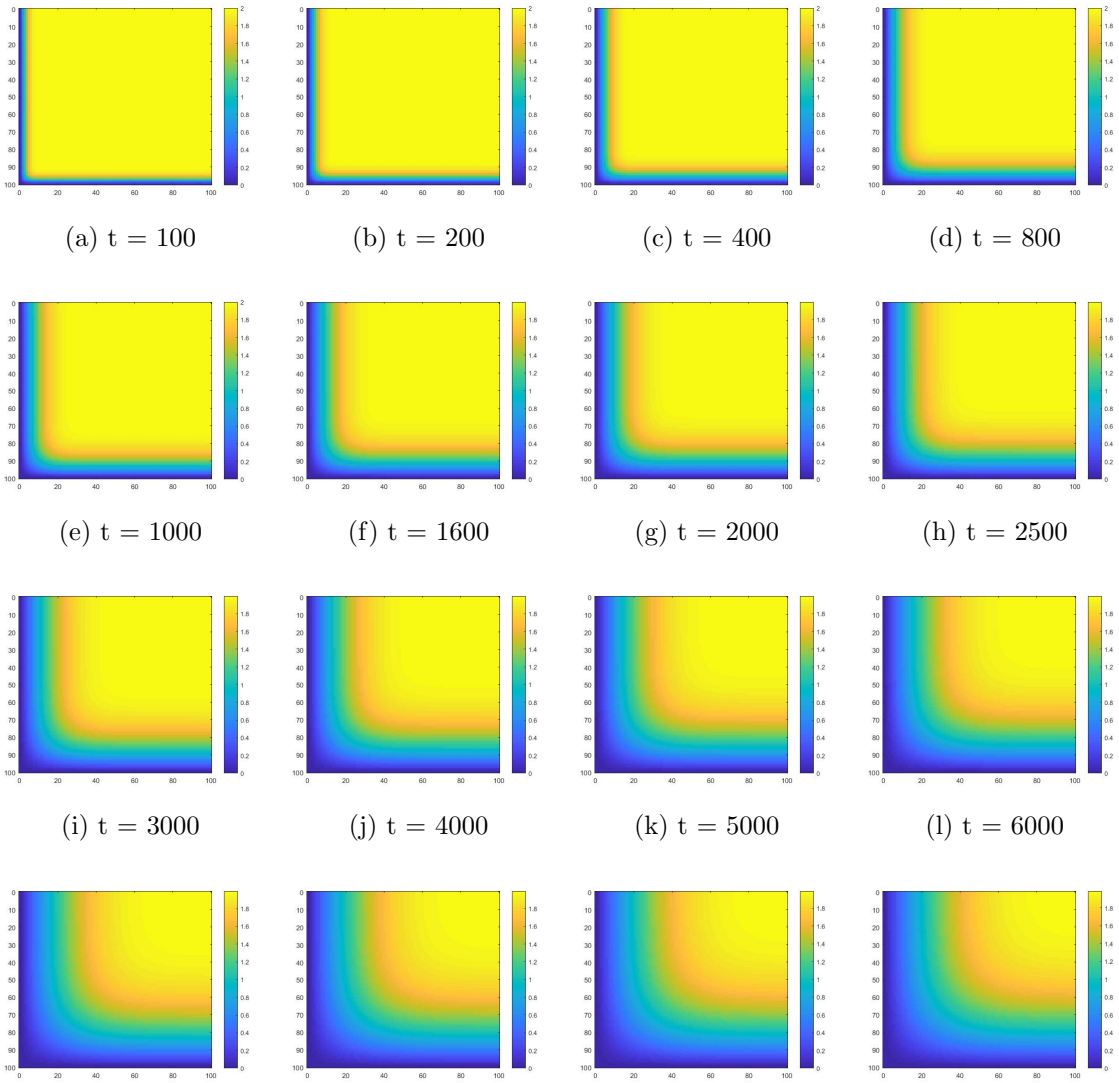


Figure 3.5: Solid Fraction distribution through the line along Y-axis passing through the center of the bubble

3.7.3 Two Dimensional solidification



(a) t = 100 (b) t = 200 (c) t = 400 (d) t = 800

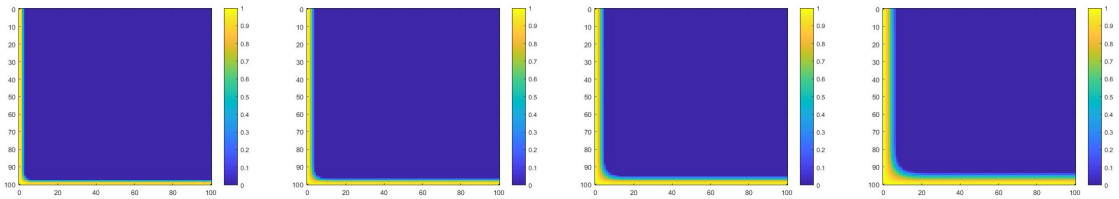
(e) t = 1000 (f) t = 1600 (g) t = 2000 (h) t = 2500

(i) t = 3000 (j) t = 4000 (k) t = 5000 (l) t = 6000

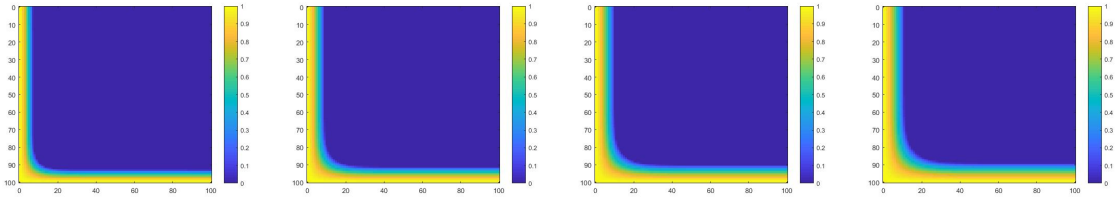
(m) t = 7000 (n) t = 8000 (o) t = 9000 (p) t = 10000

Figure 3.6: Temperature field at various time steps

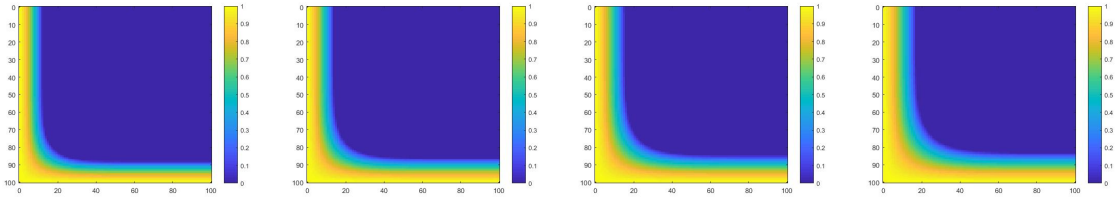
3.7.4 One Dimensional solidification



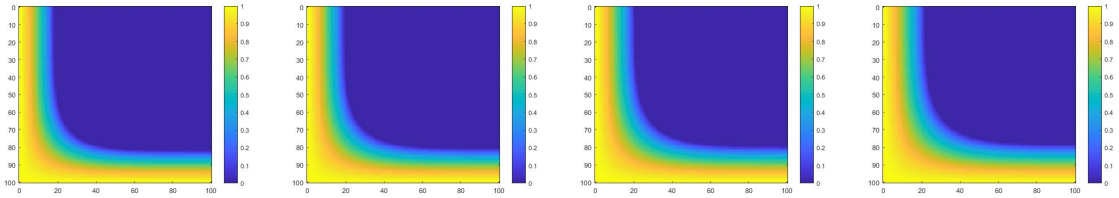
(a) t = 100 (b) t = 200 (c) t = 400 (d) t = 800



(e) t = 1000 (f) t = 1600 (g) t = 2000 (h) t = 2500

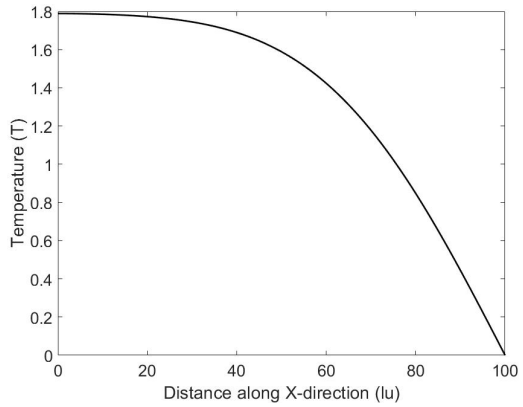


(i) t = 3000 (j) t = 4000 (k) t = 5000 (l) t = 6000

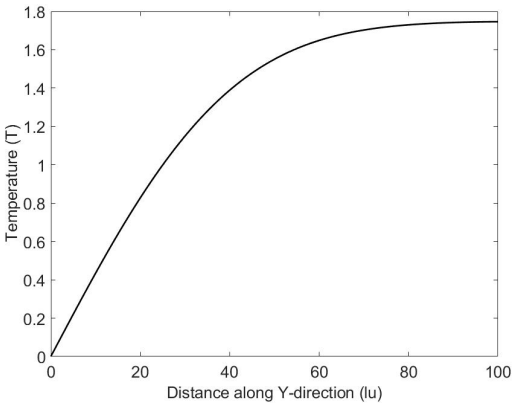


(m) t = 7000 (n) t = 8000 (o) t = 9000 (p) t = 10000

Figure 3.7: Solid Fraction field at various time steps

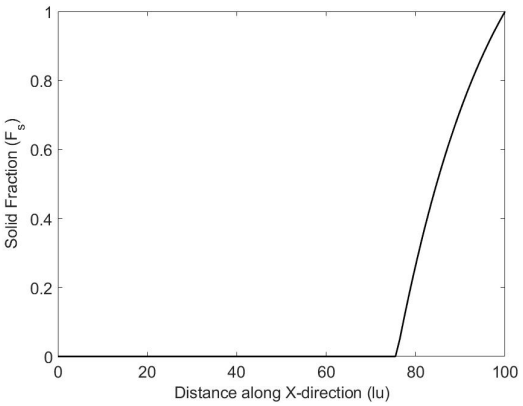


(a) Temperature variation along X-direction

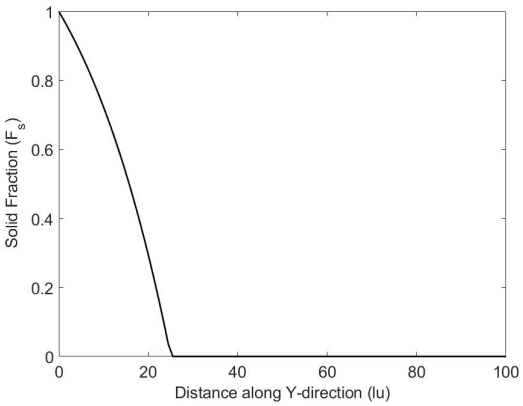


(b) Temperature variation along Y-direction

Figure 3.8: Temperature variation along X and Y direction along the lines passing through the center



(a) Solid Fraction variation along X-direction



(b) Solid Fraction variation along Y-direction

Figure 3.9: Solid Fraction variation along X and Y direction along the lines passing through the center

3.8 Conclusion

Stefan's problem of phase transition was modelled using Lattice Boltzmann Method. The model was applied to a two-dimensional lattice (D2Q9) system. Both uni-directional and multi-directional growth of ice was presented. The results from uni-directional growth matched the analytical results while the latter were qualitative.

3.9 References

1. M. N. Ozisik, Heat Conduction, Wiley, New York, 1980.
2. S. H. Cho and J. E. Sunderland, Heat-Conduction Problems with Melting or Freezing, J.
Heat Transfer, vol. 91, pp. 421-426, 1969. 3. K. A. Rathjen and L. M. Jiji, Heat Conduction with Melting or Freezing in a Corner, J. Heat Transfer, vol. 93, pp. 101-109, 1971.
4. V. R. Voller, M. Cross, and N. C. Markatos, An Enthalpy Method for Convection/Diffusion Phase Change, Int. J. Numer. Math. Eng., vol. 24, pp. 271-284, 1987.
5. W. Shyy, H. S. Udaykumar, M. M. Rao, and R. W. Smith, Computational Fluid Dynamics with Moving Boundaries, Taylor & amp Francis, Washington, DC, 1996.
6. R. Grifth and B. Nassersharif, Comparison of One-Dimensional Interface-Following and Enthalpy Methods for the Numerical Solution of Phase Change, Numer. Heat Transfer B, vol. 18, pp. 169-187, 1990.
7. Z.-X. Gong and S. Mujumdar, Noniterative Procedure for the Finite-Element Solution of the Enthalpy Model for Phase-Change Heat Conduction Problems, Numer.Heat Transfer B, vol. 27, pp. 437-446, 1995.
8. J.-Y. Lin and H.-T. Chen, Numerical Analysis for Phase Change Problems with the Mushy Zone, Numer. Heat Transfer A, vol. 27, pp. 163-177, 1995.
9. S. L. Lee and R. Y. Tzong, An Enthalpy Formulation for Phase Change Problems with a Large Thermal Diffusivity Jump across the Interface, Int. J. Heat Mass Transfer, vol. 34, pp. 1491-1502, 1991.
10. J.-Y. Lin and H.-T. Chen, Hybrid Numerical Scheme for Nonlinear Two-Dimensional Phase-change Problems with the Irregular Geometry, Heat and Mass Transfer, vol. 33, pp. 51-58, 1997.
11. U. Frish, D. dHumieres, B. Hasslacher, P. Lallemand, Y. Pomeau, and J.-P. Rivet, Lattice Gas Hydrodynamics in Two and Three Dimensions, Complex Syst., vol. 1, pp. 649-707, 1987
12. S. V. Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere, Washington, DC, 1980.
13. R. M. Furzeland, A Comparative Study of Numerical Methods for Moving Boundary Problems, J. Inst. Math. Appl., vol. 26, pp. 411-429, 1980

Chapter 4

Solidification of Falling Droplet

This chapter presents the impact and freezing of a falling droplet on a cold surface in 2D computational space. A model is developed by the lattice Boltzmann method (LBM) that applies the velocity and temperature distribution functions to investigate the solidification process of water droplet on cold flat plate. The thermal transport and liquid-solid phase transition in the present model are both based on the pseudo-potential model combined with the enthalpy formation. By this LB model, the solidification process is simulated in form of temperature and solid phase variations in water droplet on cold flat plate, and the shape of solid phase in freezing can also be predicted. Further the effect on freezing rate due to the wall temperature and relaxation time is also presented.

Note: The results are all qualitative and no experiments are done to validate them.

4.1 Mathematical Modelling

In this section we will describe the modelling of the problem using previously described Multiphase model and the existing enthalpy based solid-liquid phase change LB model.

4.1.1 Multiphase flow scheme in LBM

The mass and momentum conservation equations are covered in the density distribution function f using BGK collision scheme [1]:

$$f_i(\mathbf{x}+\mathbf{e}_i\delta t, t+\delta t) - f_i(\mathbf{x},t) = -\frac{f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t)}{\tau} + \delta t F_i \tag{4.1}$$

where f_i is the particle distribution function with discrete velocities e_i at position $\mathbf{x}$ and time t.We adopted the BhatnagarGrossKrook (BGK) (Guo and Zheng, 2009; He et al., 2009) approximation with two-dimension and nine-velocity LB model. Schematic illustration of the lattice Boltzamnn structure is shown in Fig. 1. The discrete velocities e_i in Eq. (4.1) are therefore written as:

$$\vec{e_i} = \begin{Bmatrix} (0,0) & \text{i} = 0 \\ \frac{\delta x}{\delta t}\left(cos\left[(i-1)\frac{\pi}{2}\right], sin\left[(i-1)\frac{\pi}{2}\right]\right) & \text{i} = 1,2,3,4 \\ \sqrt{2}\frac{\delta x}{\delta t}\left(cos\left[(2i-1)\frac{\pi}{4}\right], sin\left[(2i-1)\frac{\pi}{4}\right]\right) & \text{i} = 5,6,7,8 \end{Bmatrix} \tag{4.2}$$

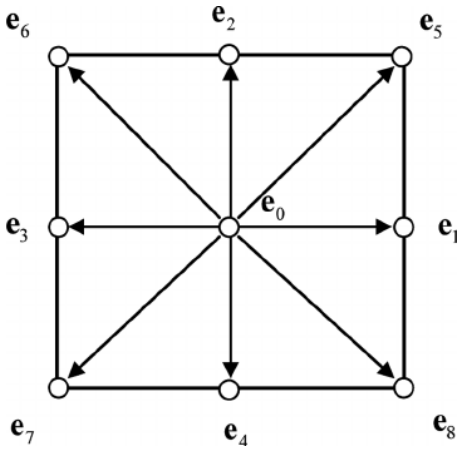


Figure 4.1: D2Q9 Lattice Schematic Representation

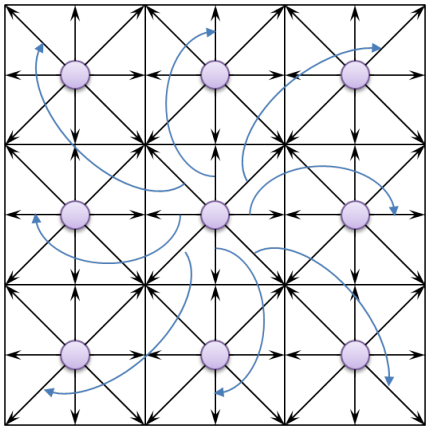


Figure 4.2: Collision and streaming Schematic Representation

where δx is the lattice spacing, δt is the time step, and τ is the dimensionless relaxation time connected with the kinematic viscosity $\nu = \frac{1}{3}(\tau - \frac{1}{2})\delta t$. The equilibrium distribution function (EDF) f_i^{eq} in Eq. (4.1) is described as follows:

$$f_i^{eq} = w_i\rho\left(1 + 3\frac{\mathbf{e}_i.\mathbf{u}}{c^2} + \frac{9}{2}\frac{(\mathbf{e}_i.\mathbf{u})^2}{c^4} - \frac{3}{2}\frac{\mathbf{u}.\mathbf{u}}{c^4}\right) \tag{4.3}$$

where ρ and $\mathbf{u}$ are the density and velocity and the weight coefficients for different directions $_i$ are given as follows:

$$w_i = \left\{ \begin{array}{cc} 4/9 & \text{i} = 0 \\ 1/9 & \text{i} = 1,2,3,4 \\ 1/18 & \text{i} = 5,6,7,8 \end{array} \right\} \tag{4.4}$$

In the pseudo-potential model, the local fluid density and flow velocity are calculated according to the following equations:

$$\rho = \sum_{i=0}^{8} f_i \tag{4.5}$$

$$\mathbf{u} = \frac{1}{\rho}\left(\sum_{i=0}^{8} \mathbf{e}_i f_i + \tau F \right) \tag{4.6}$$

where the total force F = F_{sigma} + F_t + F_g, it contains three forces including the interaction force F_{sigma} between fluids, the interaction force F_t between fluid and solid, and the gravity force F_g.
The interaction force F_{sigma} between fluids is written as follows:

$$F_{sigma} = -\phi(\mathbf{x}) \sum_{\mathbf{x}'} G(\mathbf{x}, \mathbf{x}')\phi(\mathbf{x})(\mathbf{x}' - \mathbf{x}) \tag{4.7}$$

where the function of local densities $\phi(\rho) = \rho_0[1 - exp(-\rho/\rho_0)]$ (Shan and Chen, 1994). Greens function $G(\mathbf{x}, \mathbf{x})$ in Eq. (4.7) is written as follows:

$$G(\mathbf{x}, \mathbf{x}') = \left\{ \begin{array}{cc} \text{G} & |\mathbf{x} - \mathbf{x}'| = \delta x \\ \text{G}/4 & |\mathbf{x} - \mathbf{x}'| = \sqrt{2}\delta x \\ 0 & \text{otherwise} \end{array} \right\} \tag{4.8}$$

where G is the interactive strength coefficient between fluids.
The interaction force F_t between fluid and solid (Martys and Chen, 1996) is expressed as follows:

$$F_t = -\phi(\mathbf{x}) \sum_{\mathbf{x}'} G_t(\mathbf{x}, \mathbf{x}')s(\mathbf{x} + \mathbf{e}_i)(\mathbf{x}' - \mathbf{x}) \tag{4.9}$$

where $s(\mathbf{x} + \mathbf{e}_i)$ is the indicator function of the solid phase. For the fluid phase $s(\mathbf{x} + \mathbf{e}_i) = 0$ and for the solid phase $s(\mathbf{x} + \mathbf{e}_i) = 1$. The function $G_t(\mathbf{x}, \mathbf{x})$ in Eq. (4.9) is written as follows:

$$G(\mathbf{x}, \mathbf{x}') = \left\{ \begin{array}{cc} G_t & |\mathbf{x} - \mathbf{x}'| = \delta x \\ G_t/4 & |\mathbf{x} - \mathbf{x}'| = \sqrt{2}\delta x \\ 0 & \text{otherwise} \end{array} \right\} \tag{4.10}$$

where G_t is the interactive strength coefficient between fluid and solid.
Performing a Chapman-Enskog expansion in the pseudopotential model, the continuity and momentum equations at the Navier-Stokes level are derived from the LBE.

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{u}) = 0 \tag{4.11}$$

$$\frac{\partial u}{\partial t} + \mathbf{u}\nabla\mathbf{u} = -\frac{\nabla p}{\rho} + \frac{F}{\rho} + \nu\nabla^2\mathbf{u} \tag{4.12}$$

4.1.2 LBM for temperature field

The evolution of temperature field in the LB model is governed by the evolution of distribution function g given by:

$$g_i(\mathbf{x} + c\mathbf{e}_i\delta t, t + \delta t) - g(\mathbf{x}, t) = -\frac{[g_i(\mathbf{x}, t) - g_i^{eq}(\mathbf{x}, t)]}{\tau_T} + \delta T \Phi_t \tag{4.13}$$

where g_i is the temperature distribution function and c is the dimensionless relaxation time. In the D2Q9 lattice, the thermal diffusivity $\alpha = \frac{1}{3}(\tau T - \frac{1}{2})$. The equilibrium temperature distribution function $g_i^(eq)$ in Eq. (4.13) is written as follows:

$$g_i^{eq} = w_i T\left(1 + 3\frac{\mathbf{e}_i.\mathbf{u}}{c^2} + \frac{9}{2}\frac{(\mathbf{e}_i.\mathbf{u})^2}{c^4} - \frac{3}{2}\frac{\mathbf{u}.\mathbf{u}}{c^4}\right) \tag{4.14}$$

where the weight coefficients $_i$ and the isothermal sound velocity c are the same as those for the velocity field in Eq. (5.3), and T is the macroscopic temperature defined as follows:

$$T = \sum_{i=0}^{8} g_i \tag{4.15}$$

In Eq. (4.13), the discretized form of the source term is established as the following equation:

$$\Phi_t = w_i \frac{L}{C_p}\frac{\partial f_s}{\partial t} \tag{4.16}$$

where L is the latent heat of phase transition, C_p is the heat capacity at constant pressure and f_s is the volume fraction of solid phase calculated using the following form:

$$f_s = \left\{ \begin{array}{cc} 0 & \mathrm{H} > H_l \\ \frac{H_s - H}{H_s - Hl} & H_s \leq H \leq H_l \\ 1 & \mathrm{H} < H_s \end{array} \right\} \tag{4.17}$$

where H_s and H_l represent the enthalpy values of solidus and liquidus temperature of the fluid respectively, and H is the total enthalpy calculated as follows:

$$H = C_p T + f_s L \tag{4.18}$$

On the basis of the ChapmanEnskog expansion, we can derive the temperature macroscopic equation at the NavierStokes level (Jiaung et al., 2001) from the LBE.

4.2 Evolution of LBE

The evolution of LBM includes two independent particle movements of the collision and streaming. These two movements alternate with time and the sequence is equivalent, but there are differences for simulation in phase transition region.When external temperature is not considered, particles maintain the dynamic equilibrium.The collision and migration in this state will occur simultaneously. But computationally, one the two processes can't be computed simultaneously. So, we have devised an algorithm which first perform collision step then undergoes streaming procesess. [See Appendix for the codes].

Collision:

$$f_i(\mathbf{x}, t+\delta t) = (1 - \frac{1}{\tau}) f_i(\mathbf{x}, t) + \frac{1}{\tau} f_i^{eq}(\mathbf{x}, t) \tag{4.19}$$

$$g_i(\mathbf{x}, t+\delta t) = (1 - \frac{1}{\tau_T}) g_i(\mathbf{x}, t) + \frac{1}{\tau_T} g_i^{eq}(\mathbf{x}, t) + \delta t \Phi_t \tag{4.20}$$

Streaming:

$$f_i(\mathbf{x} + \mathbf{e}_i \delta t, t+\delta t) = f_i(\mathbf{x}, t+\delta t) \tag{4.21}$$

$$g_i(\mathbf{x} + \mathbf{e}_i \delta t, t+\delta t) = g_i(\mathbf{x}, t+\delta t) \tag{4.22}$$

To solve the flow problem in solidliquid mixture and solid region, the percolation theory in porous media is adopted where the streaming step for solving the density and velocity field is modified using the equation given below:

$$f_i(\mathbf{x} + \mathbf{e}_i \delta t, t+\delta t) = (1 - f_s) \times f_i(\mathbf{x}, t+\delta t) + f_s \times f_i(\mathbf{x} + \mathbf{e}_i \delta t, t) \tag{4.23}$$

where f_s is the solid fraction. If $f_s = 0$, the node is totally fluid and the streaming of particle is the same as that in Eq. (4.19). If $f_s = 1$, the node is totally solid and particle remains stationary. If $0 < f_s < 1$, the node is solidliquid mixture and only a part of $f_i(\mathbf{x}, t)$ is streamed.

4.3 Boundary Conditions

Appropriate boundary conditions must be implemented through the distribution functions in the LBM, which will significantly influence the stability and accuracy of simulation. Here, we have used constant wall temperature boundary condition at the bottom wall and bounceback conditions for other walls. For more details refer to section 3.6 and 4.5 for velocity and temperature boundary conditions respectively. For implementation details refer to the code in Appendix.

4.4 Numerical Procedure and Algorithm

Before running the simulation, the droplet size and position, its initial density field (ρ), temperature field (T) of the surrounding liquid and the drop, velocity field ($\mathbf{u}$) and solid fraction f_s are defined first along with other physical properties. The sensible, latent and total enthalpy is calculated using the above parameters. Then the distribution functions f and g are initialized using the known values of ρ, T and $\mathbf{u}$. Distribution functions are then evolved in a time loop according to the following algorithm given in the figure below.

4.5 Results

4.5.1 Computational Domain and parameters

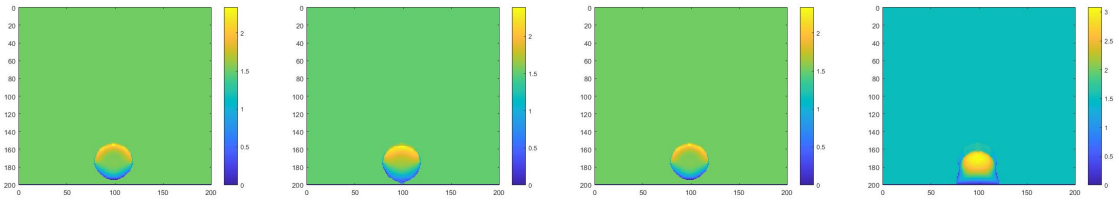
The computational domain is a 2D square cavity of grid size 202×202. The droplet of density 2.64 (lattice units) and initial radius, 20 lu is placed in a surrounding fluid of density 0.07 (lattice units) at center coordinates ($i_{center} = 101$,$j_{center} = 176$). The bottom wall is maintained at a constant temperature,T_b=0 (non-dimensional), lower than its melting point temperature ($T_m = 1$). Other properties and simulation parameters are listed in the table given below.

Physical constants	Values in lattice units
ρ_{liquid}	2.64
ρ_{vapor}	0.9
ν	0.02
Pr	0.71
$\alpha = \frac{nu}{Pr}$	0.028
g	10^{-4}
Ra	10^3
T_0	2
T_m	1
T_s	0.995
T_l	1.005
T_b	0
Latent heat (L)	1
C_w	1
C_{ice}	0.5
fluid-fluid Interaction Potential (G)	-6
solid-fluid interaction Potential (H)	-0.3

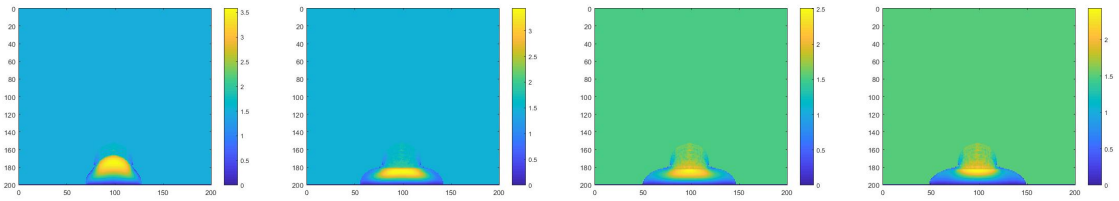
4.5.2 Numerical results for falling droplet solidification

Temperature field

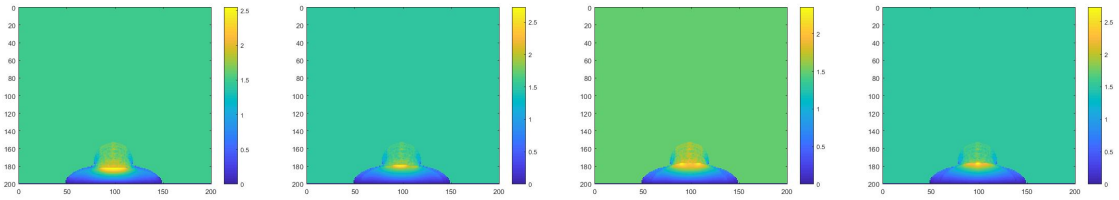
Figure 4.3 depicts the evolution of the temperature field as the droplet moves closer and closer towards the solid boundary. We noticed the temperature increase in the topmost part of the droplet as the solidification begins. This can be explained as follows: As the droplet touches the solid wall, the lower part cools down. And when it reaches below the solidification temperature, it releases its Latent heat which is absorbed by the topmost part of the droplet resulting in the increase in temperature. We also noticed some slight deformation of the bubble in the topmost part. It is due to the fact that simulation was designed in this way that the temperature field will be solved only in the droplet region ($\rho \geq \rho_{liq}/2$). Hence when the latent heat is released, some part is absorbed by the interface layer resulting in the jet shaped temperature field. This can be reduced by solving the temperature field in both liquid and vapor region.



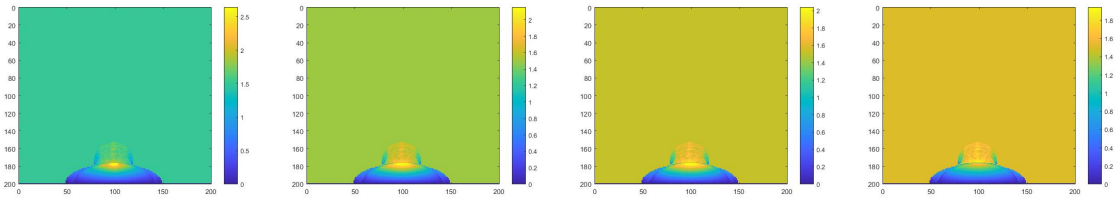
(a) t = 100 (b) t = 200 (c) t = 300 (d) t = 400



(e) t = 500 (f) t = 800 (g) t = 1000 (h) t = 1200



(i) t = 1400 (j) t = 2000 (k) t = 2500 (l) t = 3000



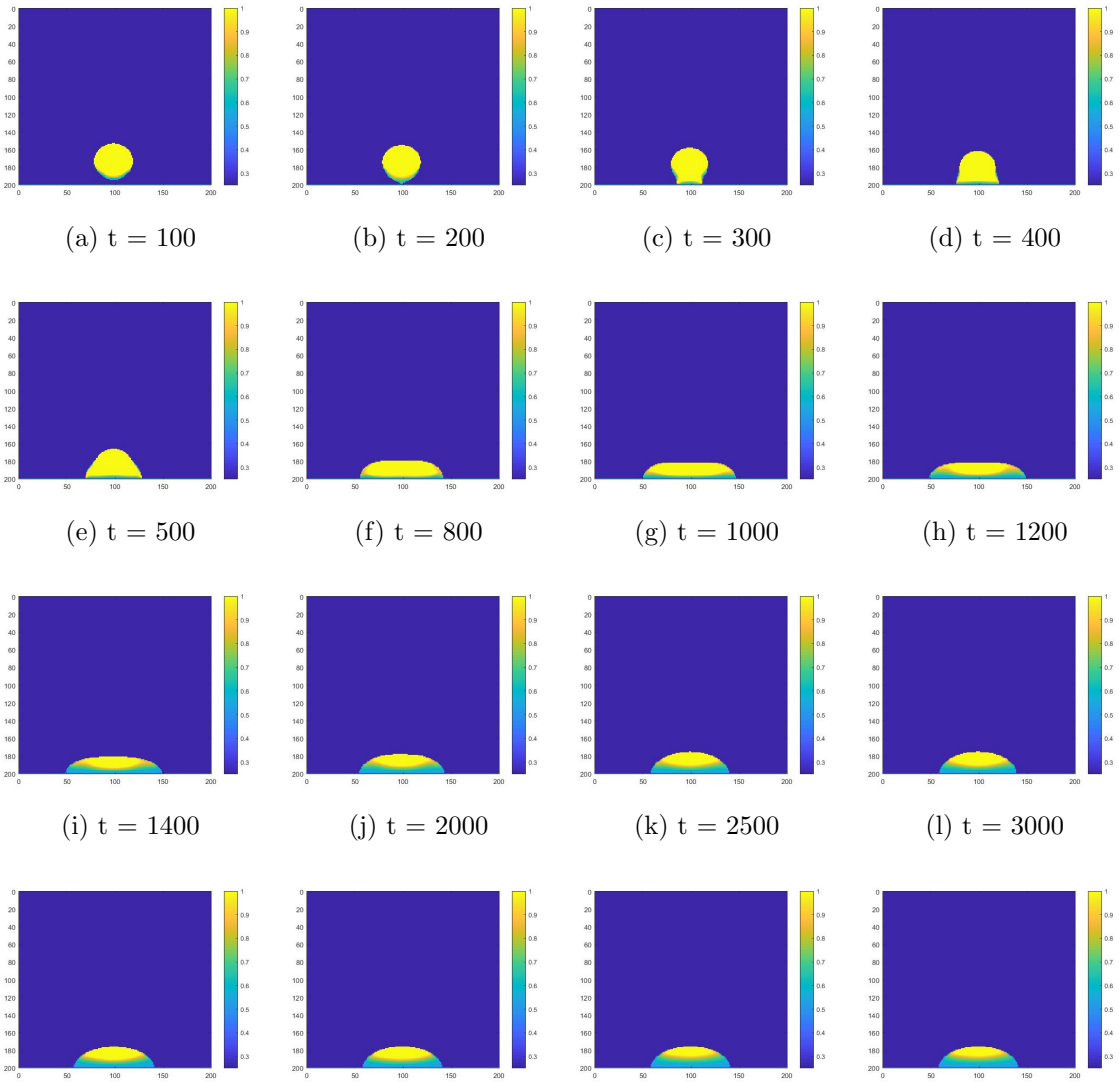
(m) t = 4000 (n) t = 8000 (o) t = 10000 (p) t = 20000

Figure 4.3: Temperature field at various time steps

Heat capacity

Figure 5.4 shows the solidification of the droplet using the heat capacity evolution with the progressing time. You can notice that the droplet solidification starts as the droplet just touches the wall. This is attributed to higher temperature relaxation time (τ_T) in the simulation. It is noticed that the droplet changes its shape during solidification process. It flattens as it touches the wall and nearly bounces back. It is due to the incomplete solidification in the earlier stages (t<2000), the medium is

porous and the distribution function streams partially according governed by the equation 4.23. But as the time progresses, the liquid fraction in the lower part of the drop approaches 1 and hence the distribution function bounces back resulting in the permanent shape of the drop (t>2000).



(a) t = 100 (b) t = 200 (c) t = 300 (d) t = 400

(e) t = 500 (f) t = 800 (g) t = 1000 (h) t = 1200

(i) t = 1400 (j) t = 2000 (k) t = 2500 (l) t = 3000

(m) t = 4000 (n) t = 8000 (o) t = 10000 (p) t = 20000

Figure 4.4: Heat capacity evolution as the time progresses

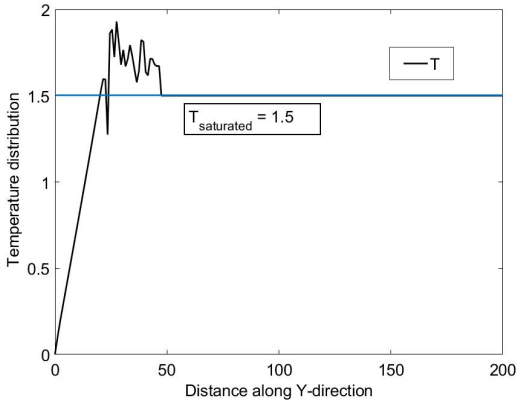


Figure 4.5: Temperature distribution through the line along Y-axis passing through the center of the bubble

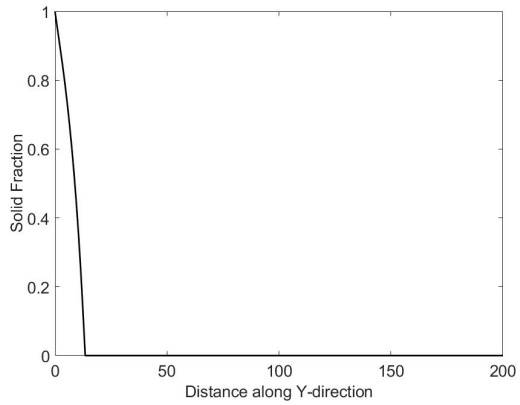


Figure 4.6: Solid-fraction distribution through the line along Y-axis passing through the center of the bubble

The above figures (4.5 and 4.6) show the variation of Temperature and Solid fraction respectively along a line parallel to Y-axis passing through the center co-ordinates of the bubble. You can clearly notice the peak in the temperature plot depicting the increase in droplet temperature at the topmost part due to absorption of Latent heat. As the interface gets thinner and thinner, the temperature saturates to the surrounding fluid temperature.

Figure 4.7 shows the variation of heat capacity along the same line. It is clearly visible that the heat capacity first increases along the line as the solid fraction decreases (since $C_w > C_s$), saturates at the liquid region and then goes down as it

reaches the vapor region.

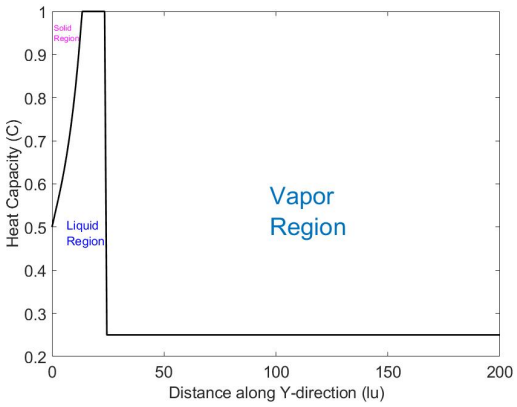


Figure 4.7: Heat Capacity distribution through the line along Y-axis passing through the center of the bubble

4.6 Conclusion

Numerical simulation of impact and solidification of falling droplet on a cold wall is presented using Lattice Boltzmann Method. The simulation parameters were taken in lattice units and has no reference to the real units. The trends match the expectations but further study is required to match it up with the real results. The simulation drawbacks can be easily improved upon by selecting proper parameters and with a little modification of code. Experimental study is a must for validation.

4.7 References

1. B.K. Cook, D.R. Noble, J.R. Williams, A Direct Simulation Method for Particle-Fluid Systems, Engineering Computations, 21(2/3/4), (2004), pp. 151−168.
2. Zhao, Jianan, Xiaoping Li, and Ping Cheng. "Lattice Boltzmann simulation of a droplet impact and freezing on cold surfaces." International Communications in Heat and Mass Transfer 87 (2017): 175-182.
3. Sun, Jinjuan, Jianying Gong, and Guojun Li. "A lattice Boltzmann model for solidification of water droplet on cold flat plate." International journal of refrigeration 59 (2015): 53-64.

Chapter 5

Appendix

5.1 MATLAB Code for flow in Lid Driven Cavity

```
%% Lattice Boltzmann code for Lid driven cavity

% grid size

Lx = 1;
Ly = 1;
nx = 100;
ny = 100;

dx = Lx/nx;
dy = Ly/ny;

%time step

dt = dx;
% vectors in x and y

x_vec(1) = 0; x_vec(2:nx+1) = dx/2:dx:Lx-dx/2; x_vec(nx+2) = Lx;
y_vec(1) = 0; y_vec(2:ny+1) = dy/2:dy:Ly-dy/2; y_vec(ny+2) = Ly;


% density and x-y velocities

u = zeros(nx+2,ny+2);
v = zeros(nx+2,ny+2);
rho = ones(nx+2,ny+2);

rho_old = zeros(nx+2,ny+2);
u_old = zeros(nx+2,ny+2);
v_old = zeros(nx+2,ny+2);

% velocity at top and Reynolds number
u_north= 10; v_north = 0;
u_south= 0; v_south = 0;
u_east= 0; v_east = 0;
u_west= 0; v_west = 0;

% velocity boundary conditions

u(1,:) = u_west; u(nx+2,:) = u_east; u(:,1) = u_south; u(:,ny+2) = u_north;
v(1,:) = v_west; v(nx+2,:) = v_east; v(:,1) = v_south; v(:,ny+2) = v_north;

%fluid properties
tau = 1;
nu = (2*tau-1)/6;
Re = u_north*Ly/nu;

f = zeros(nx+2,ny+2,9);
ftemp = f; feq = f;

% microscopic velocity term
c = dx/dt;

%d2q9 meshgird
```

(a) part 1

```
ex = [0 1 0 -1 0 1 -1 -1 1]*c;
ey = [0 0 1 0 -1 1 1 -1 -1]*c;

%% initial distribution function

c1 = 3; c2 = 9/2; c3 = 3/2;
w0 = 4/9; w1 = 1/9; w2 = 1/36;

for i = 1:nx+2
    for j = 1:ny+2

        feq(i,j,1) = w0*rho(i,j)*(1
- c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,2) = w1*rho(i,j)*(1 + c1*(u(i,j)/c^2)           + c2*(u(i,j)/c^2)^2
- c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,3) = w1*rho(i,j)*(1 + c1*(v(i,j)/c^2)           + c2*(v(i,j)/c^2)^2
- c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,4) = w1*rho(i,j)*(1 + c1*(-u(i,j)/c^2)          + c2*(u(i,j)/c^2)^2
- c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,5) = w1*rho(i,j)*(1 + c1*(-v(i,j)/c^2)          + c2*(v(i,j))^2
- c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,6) = w2*rho(i,j)*(1 + c1*(u(i,j) + v(i,j))/c^2  + c2*( (u(i,j) + v(i,
j))/c^2 )^2  - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,7) = w2*rho(i,j)*(1 + c1*(-u(i,j) + v(i,j))/c^2 + c2*( (-u(i,j) + v(i,
j))/c^2 )^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,8) = w2*rho(i,j)*(1 + c1*(-u(i,j) - v(i,j))/c^2 + c2*( (-u(i,j) - v(i,
j))/c^2 )^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,9) = w2*rho(i,j)*(1 + c1*(u(i,j) - v(i,j))/c^2  + c2*( (u(i,j) - v(i,
j))/c^2 )^2  - c3*(u(i,j)^2 + v(i,j)^2)/c^2);

    end
end

f = feq;

%% % real code

 error = 1; epsilon = 1e-7;

%while error>epsilon

n_T = 10;

for steps = 1:n_T


%% streaming

% move the direction specific fs to the nearest neighbour lattices

for i = 2:nx+1
    for j = 2:ny+1
        ftemp(i,j,1) = f(i,j,1);
        ftemp(i+1,j,2) = f(i,j,2);
        ftemp(i,j+1,3) = f(i,j,3);
```

(b) part 2

Figure 5.1: code

```
        ftemp(i-1,j,4) = f(i,j,4);
        ftemp(i,j-1,5) = f(i,j,5);
        ftemp(i+1,j+1,6) = f(i,j,6);
        ftemp(i-1,j+1,7) = f(i,j,7);
        ftemp(i-1,j-1,8) = f(i,j,8);
        ftemp(i+1,j-1,9) = f(i,j,9);
    end
end

%% equilibrium distribution function

% equation -> f_eq(x) = wa*rho(x)*(1 + 3*(ea.u)c^2 + (9/2)*(ea.u)^2/c^4  -(3/2)*(u/c)^2)

c1 = 3; c2 = 9/2; c3 = 3/2;
w0 = 4/9; w1 = 1/9; w2 = 1/36;

c = 1;
for i = 1:nx+2
    for j = 1:ny+2

        feq(i,j,1) = w0*rho(i,j)*(1 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,2) = w1*rho(i,j)*(1 + c1*(u(i,j))/c^2           + c2*(u(i,j)/c^2)^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,3) = w1*rho(i,j)*(1 + c1*(v(i,j)/c^2)           + c2*(v(i,j)/c^2)^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,4) = w1*rho(i,j)*(1 + c1*(-u(i,j)/c^2)          + c2*(u(i,j)/c^2)^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,5) = w1*rho(i,j)*(1 + c1*(-v(i,j)/c^2)          + c2*(v(i,j))^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,6) = w2*rho(i,j)*(1 + c1*(u(i,j) + v(i,j))/c^2  + c2*( (u(i,j) + v(i,j))/c^2 )^2  - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,7) = w2*rho(i,j)*(1 + c1*(-u(i,j) + v(i,j))/c^2 + c2*( (-u(i,j) + v(i,j))/c^2 )^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,8) = w2*rho(i,j)*(1 + c1*(-u(i,j) - v(i,j))/c^2 + c2*( (-u(i,j) - v(i,j))/c^2 )^2 - c3*(u(i,j)^2 + v(i,j)^2)/c^2);
        feq(i,j,9) = w2*rho(i,j)*(1 + c1*(u(i,j) - v(i,j))/c^2  + c2*( (u(i,j) - v(i,j))/c^2 )^2  - c3*(u(i,j)^2 + v(i,j)^2)/c^2);

    end
end


%% collision

for i = 1:nx+2
    for j = 1:ny+2
        for a = 1:9
            f(i,j,a) = ftemp(i,j,a) - (ftemp(i,j,a) - feq(i,j,a))/tau;
        end
    end
end
```

(a) part 3

```
% Von neuman velocity boundary condition

% Zou and He velocity boundaries

% north boundary
for i = 1:ny+2
    rho_0 =  ( f(i,ny+2,1) + f(i,ny+2,2) + f(i,ny+2,4) + 2.*( f(i,ny+2,3) + f(i,ny+2,6) + f(i,ny+2,7))) / ( 1 - u_north);
    ru = rho_0*u_north;
    rv = rho_0*v_north;
    f(i,ny+2,5) = f(i,ny+2,3) + (2/3)*rv;
    f(i,ny+2,8) = f(i,ny+2,6) + (1/6)*ru + (1/2)*rv + (1/2)*(f(i,ny+2,2) - f(i,ny+2,4));
    f(i,ny+2,9) = f(i,ny+2,7) + (1/6)*ru - (1/2)*rv + (1/2)*(f(i,ny+2,4) - f(i,ny+2,2));
end

% south boundary u = 0 v = 0
for i = 1:ny+2
    rho_0 =  ( f(i,ny+2,1) + f(i,ny+2,2) + f(i,ny+2,4) + 2.*( f(i,ny+2,5) + f(i,ny+2,8) + f(i,ny+2,9))) / ( 1 - u_south);
    ru = rho_0*u_south;
    rv = rho_0*v_south;
    f(i,1,3) = f(i,1,5) + (2/3)*rv;
    f(i,1,6) = f(i,1,8) + (1/6)*ru + (1/2)*rv + (1/2)*(f(i,1,4) - f(i,1,2));
    f(i,1,7) = f(i,1,9) + (1/6)*ru + (1/2)*rv - (1/2)*(f(i,1,2) - f(i,1,4));
end

% west boundary u = 0 v = 0

for j = 1:ny+2
    rho_0 = ( f(1,j,1) + f(1,j,3) + f(1,j,5) + 2*( f(1,j,2) + f(1,j,6) + f(1,j,9))) / ( 1 - u_west);
    ru = rho_0*u_west;
    rv = rho_0*v_west;
    f(1,j,2) = f(1,j,4) + (2/3)*rv;
    f(1,j,6) = f(1,j,8) + (1/6)*ru + (1/2)*rv + (1/2)*(f(1,j,5) - f(1,j,3));
    f(1,j,9) = f(1,j,7) + (1/6)*ru - (1/2)*rv + (1/2)*(f(1,j,3) - f(1,j,5));
end

% east boundary u = 0 v = 0
for j = 1:ny+2
    rho_0 = ( f(1,j,1) + f(1,j,3) + f(1,j,5) + 2*( f(1,j,4) + f(1,j,8) + f(1,j,7))) / ( 1 - u_east);
    ru = rho_0*u_east;
    rv = rho_0*v_east;
    f(nx+2,j,4) = f(nx+2,j,2) + (2/3)*ru;
    f(nx+2,j,8) = f(nx+2,j,6) + (1/6)*ru + (1/2)*rv + (1/2)*(f(nx+2,j,3) - f(nx+2,j,5));
    f(nx+2,j,7) = f(nx+2,j,9) + (1/6)*ru - (1/2)*rv + (1/2)*(f(nx+2,j,5) - f(nx+2,j,3));
end

for i = 1:nx+1
    for j = 1:ny+1
```

(b) part 4

Figure 5.2: code

```
        for a = 1:9
            rho(i,j) = rho(i,j) + f(i,j,a);
            u(i,j) = u(i,j) + ax(a)*f(i,j,a);
            v(i,j) = v(i,j) + ay(a)*f(i,j,a);
        end
        u(i,j) = u(i,j)/rho(i,j);
        v(i,j) = v(i,j)/rho(i,j);
    end
end


error = max(max(u-u_old).^2);
error = error + max(max(v - v_old).^2)

u_old = u;
v_old = v;
end
```

(a) part 5

Figure 5.3: code

5.2 Multi−Relaxation time scheme MATLAB code for flow induced due to Natural Convection

5.3 MATLAB code for Heat transfer using LBM

```
imax = 100; jmax = 100;
dx = 1; dy = dx;
Lx = dx*imax; Ly = dy*jmax;
x_vec(1) = 0; x_vec(2:imax+1) = dx/2:dx:Lx-dx/2; x_vec(imax+2) = Lx;
y_vec(1) = 0; y_vec(2:jmax+1) = dy/2:dy:Ly-dy/2; y_vec(jmax+2) = Ly;

dx_vec(1) = dx/2; dx_vec(2:imax) = dx; dx_vec(imax+1) = dx/2;
dy_vec(1) = dy/2; dy_vec(2:jmax) = dy; dy_vec(jmax+1) = dy/2;

u_lid = 0.2; v_lid = 0;
nu = 0.02;
density = 5;
Re = u_lid*Lx/nu;
Pr = 0.71;
alpha = nu/Pr;
%% velocity and density fields
u = zeros(imax+2,jmax+2);
v = zeros(imax+2,jmax+2);
rho = density*ones(imax+2,jmax+2);

% Temperature fields and boundary conditions
T = zeros(imax+2,jmax+2);
T_north = 1; T_west = 0; T_east = 0;
T(1,:) = T_west; T(imax+2,:) = T_east; T(:,jmax+2) = T_north;

% time step and grid velocity
dt = dx;
c = dx/dt;
% boundary conditions
u(2:imax+1,jmax+2) = u_lid;
v(2:imax+1,jmax+2) = v_lid;
% D2Q9 lattice
n_a = 9;
ex = [0 1 0 -1 0 1 -1 -1 1]*c;
ey = [0 0 1 0 -1 1 1 -1 -1]*c;

% Distribution functions
f = zeros(imax+2,jmax+2,n_a);
g = zeros(imax+2,jmax+2,n_a);
% initialization of distribution function

% weights
w = [4/9 1/9 1/9 1/9 1/9 1/36 1/36 1/36 1/36];
c1 = 3; c2 = 9/2; c3 = 3/2;
% cs = 1/sqrt(3);

for i = 1:imax+2
    for j = 1:jmax+2
        t1 = u(i,j)^2 + v(i,j)^2;
        for a = 1:n_a
            t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
            f(i,j,a) = w(a)*rho(i,j)*(1 + c1*t2 + c2*t2^2 - c3*t1);
        end
    end
end
```

(a) part 1

```
        f(i,j,3) = f(i,j-1,3);
    end
    for i = imax+2:-1:2
        f(i,j,6) = f(i-1,j-1,6);
    end
    for i = 1:imax+1
        f(i,j,7) = f(i+1,j-1,7);
    end
end

% bottom to top
for j = 1:jmax+1
    for i = 1:imax+2
        f(i,j,5) = f(i,j+1,5);
    end
    for i = 1:imax+1
        f(i,j,8) = f(i+1,j+1,8);
    end
    for i = imax+2:-1:2
        f(i,j,9) = f(i-1,j+1,9);
    end
end

%% Boundary conditions

% left wall bounce back

for j = 1:jmax+2
    f(1,j,2) = f(1,j,4);
    f(1,j,6) = f(1,j,8);
    f(1,j,9) = f(1,j,7);
end

% right wall bounce back

for j = 1:jmax+2
    f(imax+2,j,4) = f(imax+2,j,2);
    f(imax+2,j,8) = f(imax+2,j,6);
    f(imax+2,j,7) = f(imax+2,j,9);
end

% bottom wall bounce back

for i = 1:imax+2
    f(i,1,3) = f(i,1,5);
    f(i,1,6) = f(i,1,8);
    f(i,1,7) = f(i,1,9);
end

% top wall constant velocity

for i = 2:imax+1
    rho_0 = (1/(1+v_lid))*( f(i,jmax+2,1) +f(i,jmax+2,2) + f(i,jmax+2,4) + 2*(f(i,
jmax+2,3) + f(i,jmax+2,7) + f(i,jmax+2,6)) );
    f(i,jmax+2,5) = f(i,jmax+2,3) - (2/3)*rho_0*v_lid;
```

(b) part 2

Figure 5.4: code

```
f_eq = f;

for i = 1:imax+2
    for j = 1:jmax+2
        for a = 1:n_a
            t2 = ax(a)*u(i,j) + ay(a)*v(i,j);
            g(i,j,a) = w(a)*T(i,j)*(1 + c1*t2);
        end
    end
end

g_eq = g;

% %% LBM implementation
% %omega =  1/(3*nu*dt/dx^2 + 1/2);
omega = 1/(3*nu + 1/2);
omega_s = 1/(3*alpha + 1/2);
%
%n_T = 1000;
error = 1; epsilon = 1e-7;
%for steps = 1:n_T
while error>epsilon
    u_old = u;
    v_old = v;
    T_old = T;
    %% equilibrium distribution function and collision

    for i = 1:imax+2
        for j = 1:jmax+2
            t1 = u(i,j)^2 + v(i,j)^2;
            for a = 1:n_a
                t2 = ax(a)*u(i,j) + ay(a)*v(i,j);
                f_eq(i,j,a) = w(a)*rho(i,j)*(1 + c1*t2 + c2*t2^2 - c3*t1);
                f(i,j,a) = omega*f_eq(i,j,a) + (1-omega)*f(i,j,a);
            end
        end
    end


    %% streaming


    for j = 1:jmax+2
        for i = imax+2:-1:2
            f(i,j,2) = f(i-1,j,2);      % right to left
        end
        for i = 1:imax+1
            f(i,j,4) = f(i+1,j,4);      % left to right
        end
    end

    % top to bottom
    for j = jmax+2:-1:2
        for i = 1:imax+2
```

(a) part 3

```
        %f(i,jmax+2,9) = f(i,jmax+2,7) + (1/2)*rho_0*u_lid - (1/6)*rho_0*v_lid  +↙
(1/2)*(f(i,j,2) - f(i,j,4));
        %f(i,jmax+2,8) = f(i,jmax+2,6) - (1/2)*rho_0*u_lid - (1/6)*rho_0*v_lid  -↙
(1/2)*(f(i,j,2) - f(i,j,4));

         f(i,jmax+2,9) = f(i,jmax+2,7) + rho_0*u_lid/6;
         f(i,jmax+2,8) = f(i,jmax+2,6) - rho_0*u_lid/6;
    end

   %% mass and momentum conservation
    % mass conservation
    for i = 1:imax+2
        for j = 1:jmax+2
            rho_sum = 0;
            p_x_sum = 0;
            p_y_sum = 0;
            for a = 1:n_a
                rho_sum = rho_sum + f(i,j,a);
            end
              rho(i,j) = rho_sum;
        end
    end

    for i = 2:imax+2
        for a = 1:n_a
            rho(i,jmax+2) = f(i,j,1) + f(i,j,2) + f(i,j,4) + 2*( f(i,j,3) + f(i,j,6) +↙
f(i,j,7) );
        end
    end

    % momentum conservation
    for i = 2:imax+1
        for j = 2:jmax+1
            p_x_sum = 0;
            p_y_sum = 0;
            for a = 1:n_a
                p_x_sum = p_x_sum + ax(a)*f(i,j,a);
                p_y_sum = p_y_sum + ay(a)*f(i,j,a);
            end
              u(i,j) = p_x_sum/rho(i,j);
              v(i,j) = p_y_sum/rho(i,j);
        end
    end

     %% equilibrium distribution function

    for i = 1:imax+2
        for j = 1:jmax+2
            for a = 1:n_a
                t2 = ax(a)*u(i,j) + ay(a)*v(i,j);
                g_eq(i,j,a) = w(a)*T(i,j)*(1 + c1*t2);
                g(i,j,a) = omega_s*g_eq(i,j,a) + (1 - omega_s)*g(i,j,a);
            end
        end
    end
```

(b) part 4

Figure 5.5: code

```
%% streaming

for j = 1:jmax+2
    for i = imax+2:-1:2
        g(i,j,2) = g(i-1,j,2);      % right to left
    end
    for i = 1:imax+1
        g(i,j,4) = g(i+1,j,4);      % left to right
    end
end

% top to bottom
for j = jmax+2:-1:2
    for i = 1:imax+2
        g(i,j,3) = g(i,j-1,3);
    end
    for i = imax+2:-1:2
        g(i,j,6) = g(i-1,j-1,6);
    end
    for i = 1:imax+1
        g(i,j,7) = g(i+1,j-1,7);
    end
end

% bottom to top
for j = 1:jmax+1
    for i = 1:imax+2
        g(i,j,5) = g(i,j+1,5);
    end
    for i = 1:imax+1
        g(i,j,8) = g(i+1,j+1,8);
    end
    for i = imax+2:-1:2
        g(i,j,9) = g(i-1,j+1,9);
    end
end

% boundary conditions

% west boundary
for j = 1:jmax+2
    g(1,j,2) = -g(1,j,4);
    g(1,j,6) = -g(1,j,8);
    g(1,j,9) = -g(1,j,7);
end

% east boundary
for j = 1:jmax+2
    g(imax+2,j,4) = -g(imax+2,j,2);
    g(imax+2,j,7) = -g(imax+2,j,9);
    g(imax+2,j,8) = -g(imax+2,j,6);
    g(imax+2,j,3) = -g(imax+2,j,5);
    g(imax+2,j,1) = 0;
end
```

(a) part 5

```
% top boundary
for i = 1:imax+2
    g(i,jmax+2,9) = T_north*(w(9) + w(7)) - g(i,jmax+2,7);
    g(i,jmax+2,8) = T_north*(w(8) + w(6)) - g(i,jmax+2,6);
    g(i,jmax+2,5) = T_north*(w(5) + w(3)) - g(i,jmax+2,3);
    g(i,jmax+2,2) = T_north*(w(2) + w(4)) - g(i,jmax+2,4);
end

% Bottom wall adiabatic
for i = 1:imax+2
    g(i,1,1) = g(i,2,1);
    g(i,1,2) = g(i,2,2);
    g(i,1,3) = g(i,2,3);
    g(i,1,4) = g(i,2,4);
    g(i,1,5) = g(i,2,5);
    g(i,1,6) = g(i,2,6);
    g(i,1,7) = g(i,2,7);
    g(i,1,8) = g(i,2,8);
    g(i,1,9) = g(i,2,9);
end


 %% temperatura calculation
for i = 2:imax+1
    for j = 2:jmax+1
        sum = 0;
        for a = 1:n_a
           sum = sum + g(i,j,a);
        end
        T(i,j) = sum;
    end
end

error = max(max(abs(u-u_old)^2));
error = error + max(max(abs(v-v_old)^2));

error_T = max(max(T-T_old))
%contourf(x_vec,y_vec,u);
contour(x_vec,y_vec,T);
pause(0);
end
```

(b) part 6

Figure 5.6: code

5.4 MATLAB code for Multiphase modelling

```
%% a simulation code for falling droplet using multiphase LBM

%% grid
dx = 1; dy = 1;
imax = 200; jmax = 200;
Lx = dx*imax; Ly = dy*jmax;

x_vec(1) = 0; x_vec(2:imax+1) = dx/2:dx:Lx-dx/2; x_vec(imax+2) = Lx;
y_vec(1) = 0; y_vec(2:jmax+1) = dy/2:dy:Ly-dy/2; y_vec(jmax+2) = Ly;

dx_vec(1) = dx/2; dx_vec(2:imax) = dx; dx_vec(imax+1) = dx/2;
dy_vec(1) = dy/2; dy_vec(2:jmax) = dy; dy_vec(jmax+1) = dy/2;

% droplet
rho_d = 2.6429;                    %% density of droplet
r_d = 20;
x_d = x_vec(imax/2);
y_d = y_vec(jmax*7/8);
dropl = 1;                         %% 1 for droplet and -1 for bubble
% surrounding fluid
rho_l = 0.0734;                    %% liquid density (lower than droplet)

% interface
w_i = 4;                           %% interface width
rho_i = (rho_l + rho_d)/2;         %% interface density

% solid
solid = zeros(imax+2,jmax+2);
solid(:,1) = 1; solid(:,jmax+2) = 1;
rho_s = 0.3*(rho_d - rho_l) + rho_l;
% forces
f_body = 1e-4;                     %% body force (gravitational)
f_body_dir = 180;                  %% body force direction (0=down 90=east 180=north↙
270=west)
G = -6;                            %interparticular interaction potential (G<0 for↙
singlephase)

%% 2D lattice (D2Q9)
dt = dx;
tau = 0.9;
omega = 1/tau;
n_a = 9;
ex = [0 1 0 -1 0 1 -1 -1 1]*(dx/dt);
ey = [0 0 1 0 -1 1 1 -1 -1]*(dx/dt);

w0 = [4/9 1/9 1/9 1/9 1/9 1/36 1/36 1/36 1/36];

%% physical matrices
u = zeros(imax+2,jmax+2);
v = zeros(imax+2,jmax+2);
rho = zeros(imax+2,jmax+2);
psi = zeros(imax+2,jmax+2);
Liq = zeros(imax+2,jmax+2);
```

(a) part 1

```
%% initialization of density field

for i = 1:imax+2
    for j = 1:jmax+2
        rho(i,j) = rho_l;
    end
end

rho(:,1) = rho_s; rho(:,jmax+2) = rho_s;


%% initialize droplet

for i = 1:imax+2                   %% i == 1 and imax+2 are solid nodes
    for j = 1:jmax+2               %% j == 1 and jmax+2 are solid nodes
        if (solid(i,j) == 0)
            distance = sqrt((x_vec(i) - x_d)^2 + (y_vec(j) - y_d)^2);
            tmp = (1/2)*( (rho_d + rho_l) - dropl*(rho_d - rho_l)*tanh(2*(distance -↙
r_d)/w_i) );
            rho(i,j) = tmp;
        end
    end
end


%% body force vectors
body_force_x = f_body*sin(f_body_dir*pi/180);
body_force_y = -f_body*cos(f_body_dir*pi/180);
F_x = zeros(imax+2,jmax+2);
F_y = zeros(imax+2,jmax+2);
%% distribution function
f = zeros(imax+2,jmax+2,n_a);

% initialization of distribution function
for i = 1:imax+2
    for j = 1:jmax+2
        if (solid(i,j)==0)
            t1 = u(i,j)^2 + v(i,j)^2;
            for a = 1:n_a
            t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
            f(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);
            end
        end
    end
end
f_eq = f;

%% LBM loop
n_T = 10000;
for steps = 1:n_T

    %% calculation of density and velocities
    for i = 1:imax+2
        for j = 1:jmax+2
            if (solid(i,j)==0)
```

(b) part 2

Figure 5.7: code

```
            sum = 0;
            for a = 1:n_a
                sum = sum + f(i,j,a);
            end
            rho(i,j) = sum;
        end
    end
end

%% calculation of liquid

for i = 1:imax+2
    for j = 1:jmax+2
        if (solid(i,j)==0)
            if (rho(i,j) >= rho_d/2)
                Liq(i,j) = 1;
            else
                Liq(i,j) = 0;
            end
        end
    end
end

for i = 1:imax+2
    for j = 1:jmax+2
        if (solid(i,j)==0)
            p_x_sum = 0;
            p_y_sum = 0;
            for a = 1:n_a
                p_x_sum = p_x_sum + ex(a)*f(i,j,a);
                p_y_sum = p_y_sum + ey(a)*f(i,j,a);
            end
            u_temp = p_x_sum/rho(i,j);
            v_temp = p_y_sum/rho(i,j);

            % body force
            u_temp = u_temp + tau*body_force_x;
            v_temp = v_temp + tau*body_force_y;

            u(i,j) = u_temp;
            v(i,j) = v_temp;
        end
    end
end

% calculation of psi
for i = 1:imax+2
    for j = 1:jmax+2
            psi(i,j) = 1 - exp(-rho(i,j));
    end
end

%% calculation of psi-gradient (E4)
for i = 2:imax+1
    for j = 2:jmax+1
```

(a) part 3

```
            if (solid(i,j)==0)
                u_temp = u(i,j);
                v_temp = v(i,j);

%                 west = -rem(imax+2-i+1,imax+2)+(imax+2);
%                 east = rem(i,imax+2) + 1;
%                 south = -rem(jmax+2-j+1,jmax+2) + (jmax+2);
%                 north = rem(j,jmax+2) + 1;
                F_x(i,j) = -G*psi(i,j)*(1/3)*( (psi(i+1,j) - psi(i-1,j))/3 + ( psi(i+1,j+1) - psi(i-1,j+1) )/12 + ( psi(i+1,j-1) - psi(i-1,j-1) )/12 );
                F_y(i,j) = -G*psi(i,j)*(1/3)*( (psi(i,j+1) - psi(i,j-1))/3 + ( psi(i+1,j+1) - psi(i+1,j-1) )/12 + ( psi(i-1,j+1) - psi(i-1,j-1) )/12 );

                % interparticula potential in equilibrium velocity
                u_temp = u_temp + tau*F_x(i,j)/rho(i,j);
                v_temp = v_temp + tau*F_y(i,j)/rho(i,j);
                u(i,j) = u_temp;
                v(i,j) = v_temp;
            end
        end
    end

    %% calculation of equilibrium function and collision

    for i = 1:imax+2
        for j = 1:jmax+2
            if (solid(i,j)==0)
                t1 = u(i,j)^2 + v(i,j)^2;
                for a = 1:n_a
                    t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
                    f_eq(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);
                    f(i,j,a) = (1-omega)*f(i,j,a) + omega*f_eq(i,j,a);
                end
            end
        end
    end

    % streaming
    for j = 1:jmax+2
        for i = imax+2:-1:2
                f(i,j,2) = f(i-1,j,2);      % east to west
        end
        for i = 1:imax+1
                f(i,j,4) = f(i+1,j,4);      % west to east
        end
    end

    % north to south
    for j = jmax+2:-1:2
        for i = 1:imax+2
                f(i,j,3) = f(i,j-1,3);
        end
        for i = imax+2:-1:2
                f(i,j,6) = f(i-1,j-1,6);
        end
```

(b) part 4

Figure 5.8: code

```
    for i = 1:imax+1
            f(i,j,7) = f(i+1,j-1,7);
    end
end

% south to north
for j = 1:jmax+1
    for i = 1:imax+2
            f(i,j,5) = f(i,j+1,5);
    end
    for i = 1:imax+1
            f(i,j,8) = f(i+1,j+1,8);
    end
    for i = imax+2:-1:2
            f(i,j,9) = f(i-1,j+1,9);
    end
end

%% bounceback boundary condition

% west wall bounce back

for j = 1:jmax+2
    f(1,j,2) = f(1,j,4);
    f(1,j,6) = f(1,j,8);
    f(1,j,9) = f(1,j,7);
end

% east wall bounce back

for j = 1:jmax+2
    f(imax+2,j,4) = f(imax+2,j,2);
    f(imax+2,j,8) = f(imax+2,j,6);
    f(imax+2,j,7) = f(imax+2,j,9);
end

% south wall bounce back

for i = 1:imax+2
    f(i,1,3) = f(i,1,5);
    f(i,1,6) = f(i,1,8);
    f(i,1,7) = f(i,1,9);
end

%% north wall bounce back

for i = 1:imax+2
    f(i,jmax+2,5) = f(i,jmax+2,3);
    f(i,jmax+2,8) = f(i,jmax+2,6);
    f(i,jmax+2,9) = f(i,jmax+2,7);
end

imagesc(rho');
%imagesc(Liq');
%quiver(u',v');
```

(a) part 5

```
    pause(0);
    disp(steps);
end
```

(b) part 6

Figure 5.9: code

5.5 MATLAB code for One dimensional Stefan's problem

```
%% code for PCM solidification

dx = 1; dy = 1;
imax = 100; jmax = 100;
Lx = imax*dx; Ly = jmax*dy;

x_vec(1) = 0; x_vec(2:imax+1) = dx/2:dx:Lx-dx/2; x_vec(imax+2) = Lx;
y_vec(1) = 0; y_vec(2:jmax+1) = dy/2:dy:Ly-dy/2; y_vec(jmax+2) = Ly;

%% physical properties and constants
% T_critical = 647;
% rho_water = 1000; rho_ice = 919;
% Cp_water = 4180;    Cp_ice = 2000;
% L_melting = 334000;
% nu = 0.02;
 Ra = 1e3;
 g_r = 9.8;
% k_w = 0.61; k_ice = 2.3;
% alpha_ice = k_ice/(rho_ice*Cp_ice);
% alpha_water = k_w/(rho_water*Cp_water);

% in lattice units

% T_cr = 0.0729;
% T_f = 0.9; T_m = 0.7; T_b = 0.55;
% T_s = 0.695; T_l = 0.705;
% rho_w = 5.91; rho_i = 0.9*rho_w;
% time_steps = 10^4;
% Cp_w = Cp_water*(Lx)^2*T_critical/((time_steps)^2);
% Cp_i = Cp_ice*(Lx)^2*T_critical/((time_steps)^2);
% L = L_melting*(Lx)^2/(time_steps)^2;

% alpha_i = alpha_ice*(1e4);
% alpha_w = alpha_water*(1e4);
% Ste = 0.06;                %% Stefan number = Cp*(Tm-T0)/L


%% all values
T0 = 2; T_m = 1; T_w = 0;
T_s = 0.995; T_l = 1.005;
C_W = 1; L = 0.5; C_ice = 0.5;
rho_w = 1; rho_i = 0.9;
St = C_w*(T0- T_m)/L;          %% Stefan's number

Hl = C_w*T_m;
Hs = C_ice*T_m;
nu = 0.02;
alpha_w = 0.05;
alpha_i = 0.4/3;

gr = g_r*(1e-6);
 deltaT = 1;
 beta = Ra*nu*alpha_w/(Lx^3*deltaT*gr);
 f_dir = 180;
```

(a) part 1

```
%% physical matrices
T = T0*ones(imax+2,jmax+2);
u = zeros(imax+2,jmax+2);
v = zeros(imax+2,jmax+2);
rho = rho_w*ones(imax+2,jmax+2);
wall = zeros(imax+2,jmax+2);
Ps = zeros(imax+2,jmax+2);
H = zeros(imax+2,jmax+2);
C = C_w*ones(imax+2,jmax+2);
%% D2Q9 matrix
ex = [0 1 0 -1 0 1 -1 -1 1];
ey = [0 0 1 0 -1 1 1 -1 -1];
w0 = [4/9 1/9 1/9 1/9 1/9 1/36 1/36 1/36 1/36];

%% wall
wall(1,:) = 1;
%wall(:,1) = 1;
for i = 1:imax+2
    for j = 1:jmax+2
        if(wall(i,j)==0)
            T(i,j) = T_w;
            rho(i,j) = 0;
            Ps(i,j) = 1;
            C(i,j) = C_ice;
        end
    end
end

%% Distribution function
n_a = 9;
f = zeros(imax+2,jmax+2,n_a);
g = zeros(imax+2,jmax+2,n_a);

%% initialization of distribution function

for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            if(Ps(i,j)>=0 && Ps(i,j)<1)
                t1 = u(i,j)^2 + v(i,j)^2;
                for a = 1:n_a
                    t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
                    f(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);
                end
            end
        end
    end
end
f_eq = f;

for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            t1 = u(i,j)^2 + v(i,j)^2;
            for a = 1:n_a
```

(b) part 2

Figure 5.10: code

```
                t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
                g(i,j,a) = w0(a)*T(i,j)*(1 + 3*t2 + (9/2)*t2^2 - t1^2);
            end
        end
    end
end
g_eq = g;

%% LBM loop
n_t = 10000;
tau_v = 3*nu + 1/2;
tau_T_ice = 3*alpha_i + 1/2;
tau_T_w = 3*alpha_w + 1/2;
omega_v = 1/tau_v;
omega_T_w = 1/tau_T_w;
Fx = zeros(imax+2,jmax+2); Fy = zeros(imax+2,jmax+2);

for steps = 1:n_t

    u_old = u;
    v_old = v;
    rho_old = rho;
    T_old = T;

    %% mass and momentum conservation
    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                %if (Fs(i,j)>=0 && Fs(i,j)<1)
                    rho_sum = 0;
                    for a = 1:n_a
                        rho_sum = rho_sum + f(i,j,a);
                    end
                    rho(i,j) = rho_sum;
                %elseif Fs(i,j)==1
                 %       rho(i,j) = rho_ice;
                %end
            end
        end
    end

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                if (Fs(i,j)>=0 && Fs(i,j)<1)
                    Fx(i,j) = gr*beta*(T(i,j)-T0)*sin(f_dir*pi/180);
                    Fy(i,j) = gr*beta*(T(i,j)-T0)*cos(f_dir*pi/180);
                    p_x_sum = 0;
                    p_y_sum = 0;
                    for a = 1:n_a
                        p_x_sum = p_x_sum + ex(a)*u(i,j);
                        p_y_sum = p_y_sum + ey(a)*v(i,j);
                    end
                    u_temp = p_x_sum/rho(i,j);
                    v_temp = p_y_sum/rho(i,j);
```

(a) part 3

```
                    % body force
                    u(i,j) = u_temp + tau_v*Fx(i,j);
                    v(i,j) = v_temp + tau_v*Fy(i,j);

                elseif (Fs(i,j)==1)
                    u(i,j) = 0;
                    v(i,j) = 0;
                end
            end
        end
    end

    %% calculation of temperature

%     for i = 1:imax+2
%         for j = 1:jmax+2
%             if (wall(i,j)==0)
%                 T_sum = 0;
%                 for a = 1:n_a
%                     T_sum = T_sum + g(i,j,a);
%                 end
%                 T(i,j) = T_sum;
%             end
%         end
%     end

    %% calculation of temperature gradient
%     for i = 1:imax+2
%         for j =1:jmax+2
%             if (wall(i,j)==0)
%                 delta_phi(i,j) = (T(i,j) - T_old(i,j))/(T_f - T_b);
%             end
%         end
%     end
    %% equilibrium distribution function and collision

    %f
    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                if (Fs(i,j)>=0 && Fs(i,j)<1)
                    t1 = u(i,j)^2 + v(i,j)^2;
                    for a = 1:n_a
                        t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
                        f_eq(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);

                        f(i,j,a) = (1-omega_v)*f(i,j,a) + omega_v*f_eq(i,j,a);
                    end
                end
            end
        end
    end

%     %g (explicit equation)
%     for i = 1:imax+2
```

(b) part 4

Figure 5.11: code

```
%         for j = 1:jmax+2
%             if (wall(i,j)==0)
%                 for a = 1:n_a
%                     t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
%                     g_eq(i,j,a) = w0(a)*T(i,j)*(1 + 3*t2);
%                     g(i,j,a) = (1-omega_T)*g(i,j,a) + omega_T*g_eq(i,j,a);
%                 end
%             end
%         end
%     end

    % implicit
    epsilon  = 1e-4; error = 1;
    Fs_old =Fs;
    while error>epsilon
        Fs_old_iter = Fs;
        T_old_iter = T;
        for i = 1:imax+2
            for j = 1:jmax+2
                if (wall(i,j)==0)
                    t1 = u(i,j)^2 + v(i,j)^2;
                    for a = 1:n_a
                        t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
                        g_eq(i,j,a) = w0(a)*T(i,j)*(1 + 3*t2 + (9/2)*t2^2 - t1^2);
                        g(i,j,a) = (1 - omega_T_w)*g(i,j,a) + omega_T_w*g_eq(i,j,a) -↙
w0(a)*(L/C_w)*(Fs(i,j) - Fs_old(i,j));
                    end
                end
            end
        end

        %% streaming g

         %g
    for j = 1:jmax+2
        for i = imax+2:-1:2
            g(i,j,2) = g(i-1,j,2);      % east to west
        end
        for i = 1:imax+1
            g(i,j,4) = g(i+1,j,4);      % west to east
        end
    end

    % north to south
    for j = jmax+2:-1:2
        for i = 1:imax+2
            g(i,j,3) = g(i,j-1,3);
        end
        for i = imax+2:-1:2
            g(i,j,6) = g(i-1,j-1,6);
        end
        for i = 1:imax+1
            g(i,j,7) = g(i+1,j-1,7);
        end
    end
```

(a) part 5

```
% south to north
for j = 1:jmax+1
    for i = 1:imax+2
        g(i,j,5) = g(i,j+1,5);
    end
    for i = 1:imax+1
        g(i,j,8) = g(i+1,j+1,8);
    end
    for i = imax+2:-1:2
        g(i,j,9) = g(i-1,j+1,9);
    end
end

%% calculation of temperature

for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            T_sum = 0;
            for a = 1:n_a
                T_sum = T_sum + g(i,j,a);
            end
            T(i,j) = T_sum;
        end
    end
end

%% calculate Enthalpy

for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j) ==0)
            H(i,j) = C(i,j)*T(i,j) + Fs(i,j)*L;
        end
    end
end

%% calculate solid fraction

for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            if H(i,j)>Hl
                Fs(i,j) = 0;
            elseif H(i,j)<Hl && H(i,j)>Hs
                Fs(i,j) = (Hl-H(i,j))/(Hl - Hs);
                C(i,j) = Fs(i,j)*C_ice + (1-Fs(i,j))*C_w;
            elseif H(i,j)<Hs
                Fs(i,j) = 1;
                C(i,j) = C_ice;
            end
        end
    end
end
```

(b) part 6

Figure 5.12: code

```
error_Fs = abs( max(max((Fs(i,j) - Fs_old_iter(i,j)))) );
error_T = abs(max(max((T(i,j) - T_old_iter(i,j))/(T(i,j)))));
error = max([error_Fs, error_T])
end


%% streaming
%f
for j = 1:jmax+2
    for i = imax+2:-1:2
        f(i,j,2) = (1-Fs(i,j))*f(i-1,j,2) + Fs(i,j)*f(i,j,2);      % east to west
    end
    for i = 1:imax+1
        f(i,j,4) = (1-Fs(i,j))*f(i+1,j,4) + Fs(i,j)*f(i,j,4);      % west to east
    end
end

% north to south
for j = jmax+2:-1:2
    for i = 1:imax+2
        f(i,j,3) = (1-Fs(i,j))*f(i,j-1,3) + Fs(i,j)*f(i,j,3);
    end
    for i = imax+2:-1:2
        f(i,j,6) = (1-Fs(i,j))*f(i-1,j-1,6) + Fs(i,j)*f(i,j,6);
    end
    for i = 1:imax+1
        f(i,j,7) = (1-Fs(i,j))*f(i+1,j-1,7) + Fs(i,j)*f(i,j,7);
    end
end

% south to north
for j = 1:jmax+1
    for i = 1:imax+2
        f(i,j,5) = (1-Fs(i,j))*f(i,j+1,5) + Fs(i,j)*f(i,j,5);
    end
    for i = 1:imax+1
        f(i,j,8) = (1-Fs(i,j))*f(i+1,j+1,8) + Fs(i,j)*f(i,j,8);
    end
    for i = imax+2:-1:2
        f(i,j,9) = (1-Fs(i,j))*f(i-1,j+1,9) + Fs(i,j)*f(i,j,9);
    end
end

%    %g
%    for j = 1:jmax+2
%        for i = imax+2:-1:2
%            g(i,j,2) = g(i-1,j,2);      % east to west
%        end
%        for i = 1:imax+1
%            g(i,j,4) = g(i+1,j,4);      % west to east
%        end
%    end
%
%    % north to south
%    for j = jmax+2:-1:2
%        for i = 1:imax+2
%            g(i,j,3) = g(i,j-1,3);
%        end
%        for i = imax+2:-1:2
%            g(i,j,6) = g(i-1,j-1,6);
%        end
%        for i = 1:imax+1
%            g(i,j,7) = g(i+1,j-1,7);
%        end
%    end
%
%    % south to north
%    for j = 1:jmax+1
%        for i = 1:imax+2
%            g(i,j,5) = g(i,j+1,5);
%        end
%        for i = 1:imax+1
%            g(i,j,8) = g(i+1,j+1,8);
%        end
%        for i = imax+2:-1:2
%            g(i,j,9) = g(i-1,j+1,9);
%        end
%    end

%% boundary condition
%f
% west wall bounceback

for j = 1:jmax+2
    f(1,j,2) = f(1,j,4);
    f(1,j,6) = f(1,j,8);
    f(1,j,9) = f(1,j,7);
end

% east wall bounce back

for j = 1:jmax+2
    f(imax+2,j,4) = f(imax+2,j,2);
    f(imax+2,j,8) = f(imax+2,j,6);
    f(imax+2,j,7) = f(imax+2,j,9);
end

% south wall bounce back

for i = 1:imax+2
    f(i,1,3) = f(i,1,5);
    f(i,1,6) = f(i,1,8);
    f(i,1,7) = f(i,1,9);
end

% north wall bounce back

for i = 1:imax+2
    f(i,jmax+2,5) = f(i,jmax+2,3);
```

(a) part 7

(b) part 8

Figure 5.13: code

```
        f(1,jmax+2,8) = f(1,jmax+2,6);
        f(1,jmax+2,9) = f(1,jmax+2,7);
    end

    % g
    % west wall T = T_b

    for j = 1:jmax+2
        T_residual_west = (1/(w0(2) + w0(6) + w0(9)))*(T_w - (g(1,j,1) + g(1,j,3) + g↙
(1,j,4) + g(1,j,5) + g(1,j,7) + g(1,j,8)) );
        %g(1,j,2) = T_b*(w0(2) + w0(4)) - g(1,j,4);
        %g(1,j,6) = T_b*(w0(6) + w0(8)) - g(1,j,8);
        %g(1,j,9) = T_b*(w0(9) + w0(7)) - g(1,j,7);
        g(1,j,2) = w0(2)*T_residual_west;
        g(1,j,6) = w0(6)*T_residual_west;
        g(1,j,9) = w0(9)*T_residual_west;
    end

%     for j = 1:jmax+2
%         g(1,j,2) = g(1,j,4);
%         g(1,j,6) = g(1,j,8);
%         g(1,j,9) = g(1,j,7);
%     end

    % east wall bounce back

    for j = 1:jmax+2
        g(imax+2,j,4) = g(imax+2,j,2);
        g(imax+2,j,8) = g(imax+2,j,6);
        g(imax+2,j,7) = g(imax+2,j,9);
    end

    % south wall T = T_b

%     for i = 1:imax+2
%         T_residual_south = (1/(w0(3) + w0(6) + w0(7)))*(T_w - (g(1,1,1) + g(1,1,2) +↙
g(1,1,4) + g(1,1,5) + g(1,1,8) + g(1,1,9)) );
%         %g(1,1,3) = T_b*(w0(3) + w0(5)) - g(1,1,5);
%         %g(1,1,6) = T_b*(w0(6) + w0(8)) - g(1,1,8);
%         %g(1,1,7) = T_b*(w0(7) + w0(9)) - g(1,1,9);
%         g(1,1,3) = w0(3)*T_residual_south;
%         g(1,1,6) = w0(6)*T_residual_south;
%         g(1,1,7) = w0(7)*T_residual_south;
%     end

    for i = 1:imax+2
        g(1,1,3) = g(1,1,5);
        g(1,1,6) = g(1,1,8);
        g(1,1,7) = g(1,1,9);
    end
    % north wall bounce back

    for i = 1:imax+2
        g(1,jmax+2,5) = g(1,jmax+2,3);
        g(1,jmax+2,8) = g(1,jmax+2,6);
```

(a) part 9

```
        g(1,jmax+2,9) = g(1,jmax+2,7);
    end

    %% calculation of solid fraction
%     eps = 0/05;
%     for i = 1:imax+2
%         for j = 1:jmax+2
%             if (wall(i,j)==0)
%                 if ( T(i,j) < (1-eps)*T_m )
%                     Fs(i,j) = 1;
%                 elseif ( T(i,j) > (1+eps)*T_m )
%                     Fs(i,j) = 0;
%                 elseif ( T(i,j) > (1-eps)*T_m && T(i,j) < (1+eps)*T_m )
%                     Fs(i,j) = (T_m*(1 + eps) - T(i,j))/(2*eps);
%                 end
%             end
%         end
%     end

    imagesc(Fs);
    %contourf(Fs);
    %contourf(T);
    %imagesc(rho);
    %quiver(x_vec,y_vec,u',v');
    colorbar;
    pause(0);
    disp(strcat(eps));
%     max(max(max(f)));

end
```

(b) part 10

Figure 5.14: code

5.6 MATLAB code for Droplet Solidification

```
%% a simulation code for falling droplet using multiphase LBM

%% grid
dx = 1; dy = 1;
imax = 200; jmax = 200;
%imax = 400; jmax = 400;
Lx = dx*imax; Ly = dy*jmax;

x_vec(1) = 0; x_vec(2:imax+1) = dx/2:dx:Lx-dx/2; x_vec(imax+2) = Lx;
y_vec(1) = 0; y_vec(2:jmax+1) = dy/2:dy:Ly-dy/2; y_vec(jmax+2) = Ly;

dx_vec(1) = dx/2; dx_vec(2:imax) = dx; dx_vec(imax+1) = dx/2;
dy_vec(1) = dy/2; dy_vec(2:jmax) = dy; dy_vec(jmax+1) = dy/2;

% droplet
rho_w = 2.6429;                    %% density of droplet
%rho_w = 1;
r_d = 20;
%r_d = 40;
x_d = x_vec(imax/2);
y_d = y_vec(jmax*7/8);
%y_d = y_vec(jmax*3/4);
dropl = 1;                         %% 1 for droplet and -1 for bubble
% surrounding fluid
rho_air = 0.0734;                      %% liquid density (lower than droplet)

% interface
w_i = 4;                           %% interface width
rho_i = (rho_air + rho_w)/2;         %% interface density

% solid
wall = zeros(imax+2,jmax+2);
%wall(:,1) = 1;
wall(:,jmax+2) = 1;
rho_s = 0.3*(rho_w - rho_air) + rho_air;
% forces
f_body = 1e-4;                     %% body force (gravitational)
f_body_dir = 180;                  %% body force direction (0=down 90=east 180=north↙
270=west)
G = -6;                            %interparticular interaction potential (G<0 for↙
singlephase)

%% all values
T0 = 1.5; T_m = 1; T_w = 0;
T_s = 0.995; T_l = 1.005;
C_w = 1; L_w = 0.5; C_ice = 0.5;
C_air = 0.25; L_air = 100;
St = C_w*(T0- T_m)/L_w;             %% Stefan's number

Hl = C_w*T_m;
Hs = C_ice*T_m;
nu = 0.02;
alpha_w = 0.05;
alpha_i = 0.4/3;
```

(a) part 1

```
g_r= 9.81;
Ra = 1e3;
gr = g_r*(1e-6);
deltaT = 1;
beta = Ra*nu*alpha_w/(Lx^3*deltaT*gr);


%% 2D lattice (D2Q9)
dt = dx;
tau = 0.9;
omega = 1/tau;
n_a = 9;
ex = [0 1 0 -1 0 1 -1 -1 1]*(dx/dt);
ey = [0 0 1 0 -1 1 1 -1 -1]*(dx/dt);

w0 = [4/9 1/9 1/9 1/9 1/9 1/36 1/36 1/36 1/36];

%% physical matrices
u = zeros(imax+2,jmax+2);
v = zeros(imax+2,jmax+2);
rho = zeros(imax+2,jmax+2);
psi = zeros(imax+2,jmax+2);
Liq = zeros(imax+2,jmax+2);
T = T0*ones(imax+2,jmax+2);
H = zeros(imax+2,jmax+2);
C = zeros(imax+2,jmax+2);
Fs = zeros(imax+2,jmax+2);
L = zeros(imax+2,jmax+2);

%% extra

u1 = u; v1 = v;
u2 = u; v2 = v;
%% initialization of density field

for i = 1:imax+2
    for j = 1:jmax+2
        rho(i,j) = rho_air;
        C(i,j) = C_air;
        L(i,j) = L_w;
    end
end

%rho(:,1) = rho_s; rho(:,jmax+2) = rho_s;
%% wall density
for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==1)
            T(i,j) = T_w;
            rho(i,j) = rho_s;
            Fs(i,j) = 1;
            C(i,j) = C_ice;
            L(i,j) = 0;
        end
```

(b) part 2

Figure 5.15: code

```
    end
end

%% initialize droplet

for i = 1:imax+2                    %% i == 1 and imax+2 are solid nodes
    for j = 1:jmax+2                %% j == 1 and jmax+2 are solid nodes
        if (wall(i,j) == 0)
            distance = sqrt((x_vec(i) - x_d)^2 + (y_vec(j) - y_d)^2);
            tmp = (1/2)*( (rho_w + rho_air) - drop1*(rho_w - rho_air)*tanh(2*(distance↙
- r_d)/w_i) );
            rho(i,j) = tmp;
            tmp_C = (1/2)*( (C_w + C_air) - drop1*(C_w - C_air)*tanh(2*(distance -↙
r_d)/w_i) );
            C(i,j) = tmp_C;
            tmp_Liq = (1/2)*( (1 + 0) - drop1*(1 - 0)*tanh(2*(distance - r_d)/w_i) );
            Liq(i,j) = tmp_Liq;
        end
    end
end


%% body force vectors
body_force_x = f_body*sin(f_body_dir*pi/180);
body_force_y = -f_body*cos(f_body_dir*pi/180);
F_x = zeros(imax+2,jmax+2);
F_y = zeros(imax+2,jmax+2);

%% distribution function
f = zeros(imax+2,jmax+2,n_a);
g = zeros(imax+2,jmax+2,n_a);

% initialization of distribution function
for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            t1 = u(i,j)^2 + v(i,j)^2;
            for a = 1:n_a
            t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
            f(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);
            end
        end
    end
end
f_eq = f;
f_eq1 = f;
f_eq2 = f;
delta_f = f_eq2 - f_eq1;


for i = 1:imax+2
    for j = 1:jmax+2
        if (wall(i,j)==0)
            t1 = u(i,j)^2 + v(i,j)^2;
            for a = 1:n_a
```

(a) part 3

```
            t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
            g(i,j,a) = w0(a)*T(i,j)*(1 + 3*t2 + (9/2)*t2^2 - t1^2);
        end
    end
    end
end
g_eq = g;

%% LBM loop
tau_v = 3*nu + 1/2;
tau_T_ice = 3*alpha_i + 1/2;
tau_T_w = 3*alpha_w + 1/2;
tau_T_w = 0.99;
omega_v = 1/tau_v;
omega_T_w = 1/tau_T_w;


%% LBM loop
n_T = 20000;
for steps = 1:n_T

    u_old = u;
    v_old = v;
    rho_old = rho;
    T_old = T;


   % implicit
    epsilon  = 1e-4; error = 1;
    Fs_old =Fs;
    while error>epsilon

        Fs_old_iter = Fs;
        T_old_iter = T;

%% bounceback boundary condition

    % west wall bounce back

    for j = 1:jmax+2
        f(1,j,2) = f(1,j,4);
        f(1,j,6) = f(1,j,8);
        f(1,j,9) = f(1,j,7);
    end

    % east wall bounce back

    for j = 1:jmax+2
        f(imax+2,j,4) = f(imax+2,j,2);
        f(imax+2,j,8) = f(imax+2,j,6);
        f(imax+2,j,7) = f(imax+2,j,9);
    end

    % south wall bounce back
```

(b) part 4

Figure 5.16: code

```
    for i = 1:imax+2
        f(i,1,3) = f(i,1,5);
        f(i,1,6) = f(i,1,8);
        f(i,1,7) = f(i,1,9);
    end

    %% north wall bounce back

    for i = 1:imax+2
        f(i,jmax+2,5) = f(i,jmax+2,3);
        f(i,jmax+2,8) = f(i,jmax+2,6);
        f(i,jmax+2,9) = f(i,jmax+2,7);
    end

 %% calculation of density
    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                if (Fs(i,j)<1)
                    sum = 0;
                    for a = 1:n_a
                        sum = sum + f(i,j,a);
                    end
                    rho(i,j) = sum;
                    %rho(i,j) = sum*(1-Fs(i,j)) + Fs(i,j)*rho_ice;
%                     elseif (Fs(i,j)==1)
%                       rho(i,j) = rho_s;
                end
            end
        end
    end

 %% calculation of liquid

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                if (rho(i,j) >= rho_l)
                    Liq(i,j) = 1;
                else
                    Liq(i,j) = 0;
                end
            end
        end
    end

 %% calculation of velocity

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0 && Fs(i,j)<1)
                p_x_sum = 0;
                p_y_sum = 0;
                for a = 1:n_a
                    p_x_sum = p_x_sum + ex(a)*f(i,j,a);
```

(a) part 5

```
                    p_y_sum = p_y_sum + ey(a)*f(i,j,a);
                end
                u_temp = p_x_sum/rho(i,j);
                v_temp = p_y_sum/rho(i,j);

                % body force
                u_temp = u_temp + tau*body_force_x;
                v_temp = v_temp + tau*body_force_y;

                u(i,j) = u_temp;
                v(i,j) = v_temp;
                u1(i,j) = u(i,j);
                v1(i,j) = v(i,j);
%                 elseif (Fs(i,j)==1)
%                   u(i,j) = 0;
%                   v(i,j) = 0;
            end
        end
    end

    % calculation of psi
    for i = 1:imax+2
        for j = 1:jmax+2
            psi(i,j) = 1 - exp(-rho(i,j));
        end
    end

    %% calculation of psi-gradient (E4)
    for i = 2:imax+1
        for j = 2:jmax+1
            if (wall(i,j)==0 && Fs(i,j)<1)
                u_temp = u(i,j);
                v_temp = v(i,j);

                F_x(i,j) = -G*psi(i,j)*(1/3)*( (psi(i+1,j) - psi(i-1,j))/3 + ( psi(i+1,j+1) - psi(i-1,j+1) )/12 + ( psi(i+1,j-1) - psi(i-1,j-1) )/12 );
                F_y(i,j) = -G*psi(i,j)*(1/3)*( (psi(i,j+1) - psi(i,j-1))/3 + ( psi(i+1,j+1) - psi(i+1,j-1) )/12 + ( psi(i-1,j+1) - psi(i-1,j-1) )/12 );

                % interparticule potential in equilibrium velocity
                u_temp = u_temp + tau*F_x(i,j)/rho(i,j);
                v_temp = v_temp + tau*F_y(i,j)/rho(i,j);
                u(i,j) = u_temp;
                v(i,j) = v_temp;
                u2(i,j) = u(i,j);
                v2(i,j) = v(i,j);
            elseif (Fs(i,j)==1)
                u(i,j) = 0;
                v(i,j) = 0;
            end
        end
    end

    %% calculation of equilibrium function and collision
```

(b) part 6

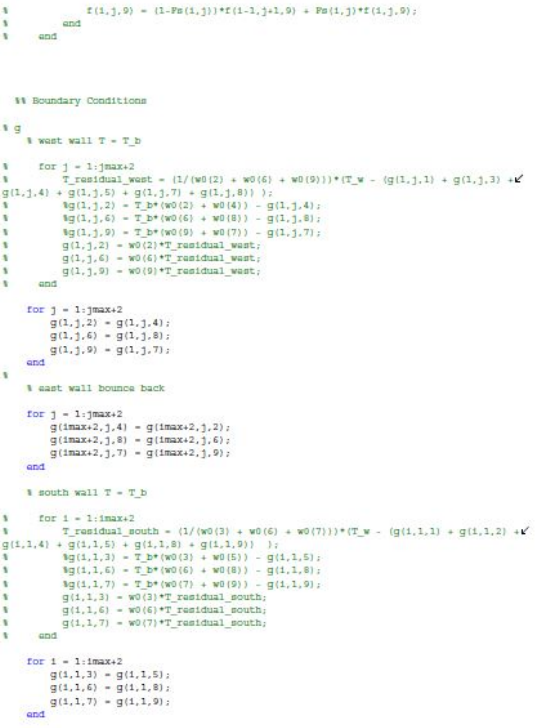
Figure 5.17: code

```
    B = 0.4;
    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0 )
              t1 = u(i,j)^2 + v(i,j)^2;
             % a1 = u1(i,j)^2 + v1(i,j)^2;
             % b1 = u2(i,j)^2 + v2(i,j)^2;
              for a = 1:n_a
%                   if (a==1)
%                       a_bar = 1;
%                   elseif (a==2 || a==3 || a ==6 || a==7)
%                       a_bar = a+2;
%                   elseif (a==4 || a==5 || a ==8 || a==9)
%                       a_bar = a-2;
%                   end
                  t2 = ex(a)*u(i,j) + ey(a)*v(i,j);
%                   a2 = ex(a)*u1(i,j) + ey(a)*v1(i,j);
%                   b2 = ex(a)*u2(i,j) + ey(a)*v2(i,j);
                  f_eq(i,j,a) = w0(a)*rho(i,j)*(1 + 3*t2 + (9/2)*t2^2 - (3/2)*t1);
%                   f_eq1(i,j,a) = w0(a)*rho(i,j)*(1 + 3*a2 + (9/2)*a2^2 - (3/2)↙
*a1);
%                   f_eq2(i,j,a) = w0(a)*rho(i,j)*(1 + 3*b2 + (9/2)*b2^2 - (3/2)↙
*b1);
%                   f_eq3(i,j,a) = w0(a)*rho(i,j);
%                   Delta(i,j,a) = f(i,j,a_bar) - f(i,j,a) + f_eq3(i,j,a) - f_eq1(i,↙
j,a_bar);
%                   delta_f(i,j,a) = f_eq2(i,j,a) - f_eq1(i,j,a);
                  f(i,j,a) = (1-omega)*f(i,j,a) + omega*f_eq(i,j,a);
                 %f(i,j,a) = f(i,j,a) - ((1 - B)*omega)*(f(i,j,a)-f_eq1(i,j,a)) +↙
B*Delta(i,j,a) + (1-B)*delta_f(i,j,a);
              end
          end
      end

       %             % streaming
     %f
      for j = 1:jmax+2
          for i = imax+2:-1:2
                f(i,j,2) = f(i-1,j,2);      % east to west
          end
          for i = 1:imax+1
                f(i,j,4) = f(i+1,j,4);      % west to east
          end
      end

    % north to south
     for j = jmax+2:-1:2
          for i = 1:imax+2
                f(i,j,3) = f(i,j-1,3);
          end
          for i = imax+2:-1:2
                f(i,j,6) = f(i-1,j-1,6);
          end
          for i = 1:imax+1
                f(i,j,7) = f(i+1,j-1,7);
          end
      end
  end

  % south to north
  for j = 1:jmax+1
      for i = 1:imax+2
            f(i,j,5) = f(i,j+1,5);
      end
      for i = 1:imax+1
            f(i,j,8) = f(i+1,j+1,8);
      end
      for i = imax+2:-1:2
            f(i,j,9) = f(i-1,j+1,9);
      end
  end



%     %% streaming
%     %f
%     for j = 1:jmax+2
%         for i = imax+2:-1:2
%             f(i,j,2) = (1-Fs(i,j))*f(i-1,j,2) + Fs(i,j)*f(i,j,2);      % east to↙
west
%         end
%         for i = 1:imax+1
%             f(i,j,4) = (1-Fs(i,j))*f(i+1,j,4) + Fs(i,j)*f(i,j,4);      % west to↙
east
%         end
%     end
%
%     % north to south
%     for j = jmax+2:-1:2
%         for i = 1:imax+2
%             f(i,j,3) = (1-Fs(i,j))*f(i,j-1,3) + Fs(i,j)*f(i,j,3);
%         end
%         for i = imax+2:-1:2
%             f(i,j,6) = (1-Fs(i,j))*f(i-1,j-1,6) + Fs(i,j)*f(i,j,6);
%         end
%         for i = 1:imax+1
%             f(i,j,7) = (1-Fs(i,j))*f(i+1,j-1,7) + Fs(i,j)*f(i,j,7);
%         end
%     end
%
%     % south to north
%     for j = 1:jmax+1
%         for i = 1:imax+2
%             f(i,j,5) = (1-Fs(i,j))*f(i,j+1,5) + Fs(i,j)*f(i,j,5);
%         end
%         for i = 1:imax+1
%             f(i,j,8) = (1-Fs(i,j))*f(i+1,j+1,8) + Fs(i,j)*f(i,j,8);
%         end
%         for i = imax+2:-1:2
```

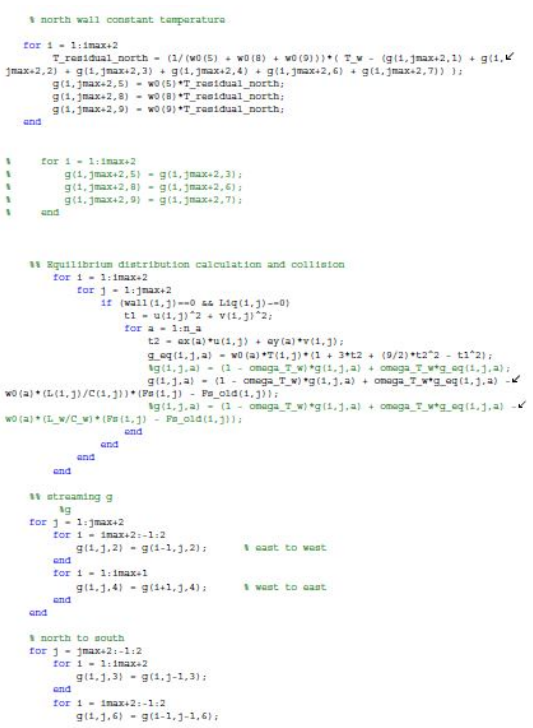
(a) part 7

(b) part 8

Figure 5.18: code



(a) part 9



(b) part 10

Figure 5.19: code

```
        end
        for i = 1:imax+1
            g(i,j,7) = g(i+1,j-1,7);
        end
    end

    % south to north
    for j = 1:jmax+1
        for i = 1:imax+2
            g(i,j,5) = g(i,j+1,5);
        end
        for i = 1:imax+1
            g(i,j,8) = g(i+1,j+1,8);
        end
        for i = imax+2:-1:2
            g(i,j,9) = g(i-1,j+1,9);
        end
    end

    % calculation of temperature

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0 && Liq(i,j)~=0)
                T_sum = 0;
                for a = 1:n_a
                    T_sum = T_sum + g(i,j,a);
                end
                T(i,j) = T_sum;
            end
        end
    end

    %% calculate Enthalpy

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j) ==0)
                if (Liq(i,j)==1)
                   % H(i,j) = C(i,j)*T(i,j) + Fs(i,j)*L;
                    H(i,j) = Fs(i,j)*C_ice*T(i,j) + (1-Fs(i,j))*C_w*T(i,j) + Fs(i,j)*L_w;
                else
                    H(i,j) = 0;
                end
            end
        end
    end

    %% calculate solid fraction

    for i = 1:imax+2
        for j = 1:jmax+2
            if (wall(i,j)==0)
                if (Liq(i,j)==1)
```

(a) part 11

```
                if H(i,j)>Hl
                    Fs(i,j) = 0;
                    C(i,j) = C_w;
                    L(i,j) = L_w;
                elseif H(i,j)<Hl && H(i,j)>Hs
                    Fs(i,j) = (Hl-H(i,j))/(Hl - Hs);
                    C(i,j) = Fs(i,j)*C_ice + (1-Fs(i,j))*C_w;
                    L(i,j) = L_w;
                elseif H(i,j)<Hs
                    Fs(i,j) = 1;
                    C(i,j) = C_ice;
                    L(i,j) = 0;
                end
            elseif (Liq(i,j)==0)
                C(i,j) = C_air;
                L(i,j) = L_air;
            end
        end
    end
end


error_Fs = max(max(abs((Fs - Fs_old_iter))) );
max_Fs = abs(max(max(Fs)));
error_Fs = error_Fs/max_Fs;
error_T = min(min(abs(((T - T_old_iter)))))
max_T = abs(max(max(T)));
error_T = error_T/max_T;
error = min([error_Fs, error_T]);
end

%imagesc(x_vec,y_vec,rho');
%subplot(1,2,1);
%imagesc(x_vec,y_vec,T');
imagesc(x_vec,y_vec,C');
%subplot(1,2,2);
%imagesc(x_vec,y_vec,Liq');
%imagesc(x_vec,y_vec,Fs');
%quiver(x_vec,y_vec,u',v');
colorbar;
pause(0);
disp(steps);
end
```

(b) part 12

Figure 5.20: code