High Performance Computing with Python

Final Report

Jonas Bürgel

5500163

jonas.buergel@mail.de

July 17, 2023

Contents

1

Introduction

2

Methods

2.1 Probability Density Function (PDF)

The Probability Density Function (PDF) is a concept that describes the probability of finding a particle at a certain position. In this project, the PDF is used to track the individual trajectories of particles in phase space. Usually this would require solving a very large number of equations. Because solving these equations would be too costly, only the averages over the volumes in the phase space are taken using the PDF. The PDF, denoted as $f(\mathbf{r}_i, \mathbf{v}_i, t)$, represents the probability density of finding a particle at a certain position $\mathbf{r_i}$ and velocity $\mathbf{v_i}$ at a given time t.

2.2 Boltzmann Transport Equation (BTE)

The equation formulates the evolution of motion for the PDF over time. The Boltzmann Transport Equation (BTE) consists of two parts. The first part is *streaming* and resembles only the moving of particles. The second part is called *collision* and deals with the interaction between particles while moving.

2.2.1 Streaming

The probability density of the PDF is able to move, which is described by the Boltzmann Transport Equation (BTE). BTE transports the probability density distributions at a specific velocity in real space. While transporting, the influence of the velocity and acceleration are considered. The combined effect of velocity and acceleration leads to the streaming of density.

The whole Boltzmann Transport Equation is denoted as

$$\frac{\partial f\left(\mathbf{r}, \mathbf{v}, t\right)}{\partial t} + \mathbf{v}\nabla_{\mathbf{r}} f\left(\mathbf{r}, \mathbf{v}, t\right) + \mathbf{a}\nabla_{\mathbf{v}} f\left(\mathbf{r}, \mathbf{v}, t\right) = C(f)\cdot \tag{2.1}$$

The l.h.s. of the equation denotes the streaming part that was just explained and the r.h.s. the collision, explained in the following part.

2.2.2 Collision

Only applying streaming resembles a probability of collision of 0%, which is not realistic. To account for collisions, an additional term is introduced into the equation to represent the collision process that occurs at each time step. In reality, collisions between particles result in an almost instantaneous exchange of energy and momentum. However, these collisions occur in extremely short time intervals on the order of femtoseconds (10^{-15} seconds), making it impractical to measure them directly in the model. Therefore, the collision process is approximated as an instantaneous process. Because of this instantaneous, it cannot be represented as a differential equation, which normally describes continuous changes. Instead, a probabilistic approach is taken to describe the effects of collisions.

In the previous section 2.2.1, the Boltzmann Transport Equation (BTE) was introduced (eq. (2.1)), where the right term represents the collision process. To simplify this collision term, a relaxation time approximation is commonly used. This approximation assumes that the probability density function (PDF) $f(\mathbf{r}, \mathbf{v}, t)$ relaxes towards a local equilibrium distribution, denoted as $f^{eq}(\mathbf{r}, \mathbf{v}, t)$. By interpreting the streaming term as the total time derivative of the PDF, the BTE can be reformulated as follows:

$$\frac{d}{dt} f(\mathbf{r}, \mathbf{v}, t) = -\frac{f(\mathbf{r}, \mathbf{v}, t) - f^{eq}(\mathbf{r}, \mathbf{v}, t)}{\tau}. \tag{2.2}$$

The included equilibirum function can is denoted as

$$f_i^{eq}(\rho(\mathbf{r}), \mathbf{u}(\mathbf{r})) = w_i \rho(\mathbf{r}) \left[1 + 3\mathbf{c}_i \cdot \mathbf{u}(\mathbf{r}) + \frac{9}{2}(\mathbf{c}_i \cdot \mathbf{u}(\mathbf{r}))^2 - \frac{3}{2}|\mathbf{u}(\mathbf{r})|^2\right]. \tag{2.3}$$

The equilibrium function introduces some additional quantities, namely the density $\rho(\mathbf{r})$, velocity $\mathbf{u}(\mathbf{r})$ and w_i. w_i is defined for a D2Q9 lattice as seen in eq. (2.4). The other two quantities can be calculated using the following formulas.

$$w_i = \left(\frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}\right) \tag{2.4}$$

$$\rho(\mathbf{r}) = \sum_i f_i \tag{2.5}$$

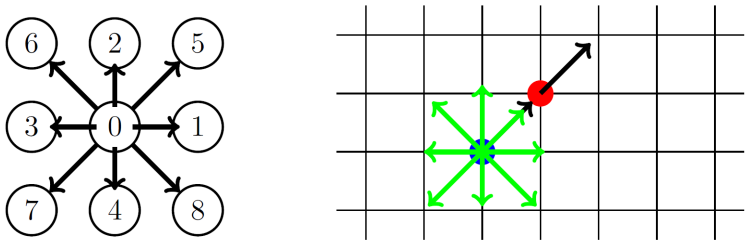
$$\mathbf{u}(\mathbf{r}) = \frac{1}{\rho(\mathbf{r})} \sum_i \mathbf{c}_i f_i(\mathbf{r}) \tag{2.6}$$

2.3 Lattice Bolzmann Scheme

In order to discretize the Boltzmann Transport Equation (BTE), it is necessary to incorporate both velocity and position space into the discrete scheme representation. An effective approach involves utilizing a 2D grid, as illustrated in fig. 2.1. The grid represents the position space as coordinates of the x and y coordinates.

Each position holds a probability density value, resembled by the value at that point. The velocities are included when introducing a third dimension, that separates the different streaming directions. To get the probability density function back, only the sum of all different directions in one point is needed. The probability density function can be reconstructed by simply summing the values for all different directions at a given point.

The streaming, as explained in section 2.2.1, is applied by moving the values of the points in one of 9 directions in their respective dimension. The collision process can be implemented by employing the functions described in section 2.2.2. During collisions, densities may be transferred between different dimensions within the scheme.



(a) (b)

Figure 2.1: Visualization of the underlying grid including labled directions.
(a) directions with given labels
(b) streaming example of one particle

3

Implementation

3.1 Setup

This section deals with the topic of setting up this project and to navigate through it. It is intended to readers who want to run the experiments themselves or help to dig into the exact implementations.

3.1.1 Environement

This project was developed using pyenv and pip. Pyenv was selected due to its ability to create a virtual Python environment while still utilizing pip as the native package manager, distinguishing it from alternatives like anaconda.

The project's requirements are outlined in the requirements.txt file, and the desired Python version is specified in the .python-version file, generated by pyenv. To install the requirements, execute the following commands in the project's top directory within a Bash environment:

```
#!/bin/bash
pyenv install 3.11
pyenv local 3.11
source venv/bin/activate
pip install -r requirements.txt
```

3.1.2 Code Structure

The project consists of three primary folders: *src/shared*, *src/experiments*, and *tests*. The *tests* folder contains code dedicated to programmatic validation of the implementations, primarily comprising unit tests. The *src/shared* folder contains code that is shared among all experiments conducted in the

project. This includes implementations for streaming and collision in the lib file, among others. Lastly, the *src/experiments* folder contains experiment-specific code, including plotting functionalities tailored to each experiment.

3.2 Probability Density Function

The probability density function is modelled as a *numpy* array with 3 dimensions, namely: channels, x-direction and y-direction in this order. While the number of channels allways has to be exactly 9, the x and y dimensions may vary in size and are independent of each other. These constraints to the probability density function are assumed by all implemented functions and have to hold at all time.

The Probability density function follows the scheme described in section 2.3. While the grid in fig. 2.1 resembles the second and third dimension of the PDF, the channels can be imagined as a third dimension on the grid. Each channel resembles one direction of moving as shown in the left part of fig. 2.1. The indices of the channels are in line with the indices of the arrows in the graphic.

3.3 Main Routine

The main routine may be seen as a function that is repeated until the experiment is over. It consists of several operators that stay the same in each experiment. For some experiments, not all operators are used or for example the *bounce-back* is only applied to certain walls. **The order of the operators always stays the same and is crucial for a successive experiment**. All possible operators are listed below:

1. collision - handles colliding particles in the simulation
2. slide - applies a steady velocity to on side
3. bounce back - bounces particles back into from certain walls
4. stream - moves all particles

An interested reader may find the exact implementation in *src/shared/boltzmann.py*. As it is redundant and maybe not inline with the exact implementation there won't be code examples at this point. However, it is to mention that all examples follow strictly the formulas explained in chapter 2, so there is no special need in doing so.

4

Results

4.1 Shear Wave Decay

The Shear Wave Decay is a common concept in computational physics to measure the kinematic viscosity of a fluid. It is set up by creating an initial sinusoidal velocity profile and measuring the decay rate. The field is set up with a periodic boundary condition on each side, which results in e.g. particles moving out of the right to appear back on the left side. The default parameters for the Shear Wave Decay experiments are shown in table 4.1 and used if not stated otherwise.

Parameter	Value
L_x	100
L_y	100
ω	1.0
ϵ	0.01
$t_{\max}$	1000

Table 4.1: Parameters for the Shear Wave Decay

4.1.1 Sinusodial Density

The initial condition is a given by the following equation where L_x resembles the size in x-direction

$$\mathbf{u}(\mathbf{r}, 0) = 0$$
$$\rho(\mathbf{r}, 0) = \rho_0 + \epsilon \sin\left(\frac{2\pi x}{L_x}\right).$$

Because of the initialization the fluid is shaped like a sinusoid wave without any velocity. It is expected that this wave collapses in itself. This is due to the fact that the system tries to reach a state of equilibrium where

the mass at each position is the same. Therefore, a flow is created from the higher density area to the lower density area. This flow continues until the mass at the previous lower density area is so dense, that no further flow is created. It now reached a state similar to the beginning however the dense and low-dense areas swapped which is why the flow will have opposite directions in the next iteration. This process can be seen in fig. 4.1.

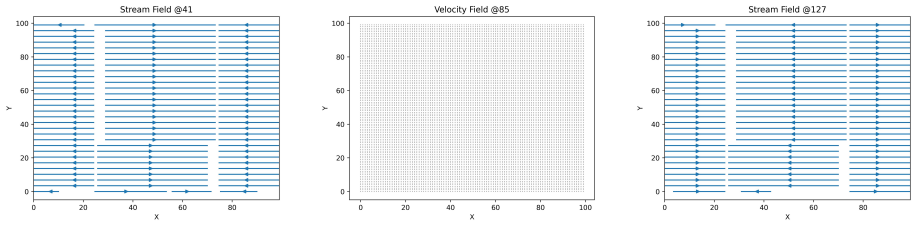


Figure 4.1: Different flow states during the simulation. From piling up at step 41 to a steady state at step 85 to the opposite flow at step 127.

This flow won't hold forever as the newly forming dense areas are always less dense as the once from the previous iteration. The system tries to reach an equilibrium. Over time the piles are shallower and shallower until it reaches the desired equilibrium function with the same density at all positions. The decay is shown through the plots in fig. 4.2.

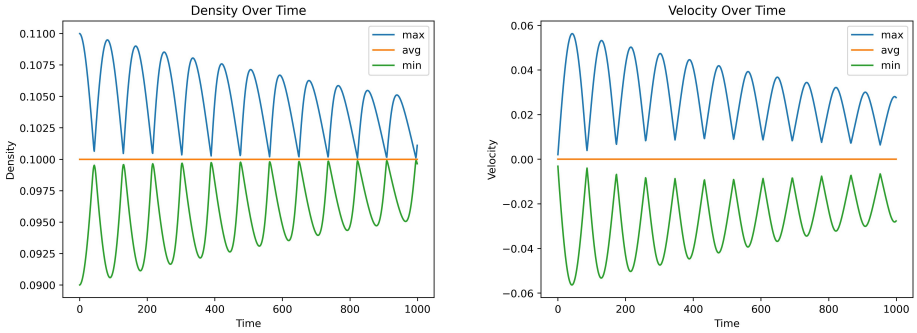


Figure 4.2: Decaying density and velocity over time.

4.1.2 Sinusoidal Velocity

This experiment ran with an epsilon of 0.5 to display the effect more drastically. The initial condition is a given by the following equation where L_y resembles the size in y-direction and ϵ resembles the initial amplitude

$$u_x(\mathbf{r}, 0) = \epsilon \sin\left(\frac{2\pi x}{L_y}\right)$$

$$\rho(\mathbf{r}, 0) = 1\cdot$$

Because of the initial condition, the flow varies throughout the field. The equilibrium state is a state without a clearly directed movement as in the initial condition. So it is expected that the flow will decrease over time.

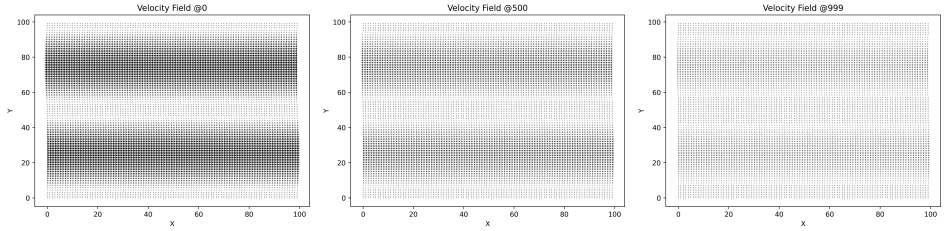


Figure 4.3: Different flow states during the simulation. From the sinusoid initial condition with a steady decrease till step 999.

Figure 4.3 gives a brief overview over the initial flow, and it's decaying over time. While at step 0, a strong sinusoid flow can be seen, at timestep 999 it decreased visibly. This decrease can be even further shown in fig. 4.4 which shows the decay at a specific column. The decay rate is also in line with the theoretical solution (as seen in fig. 4.5) given by

$$a_t = a_0 e^{-\nu t \frac{2\pi}{L_y}^2}$$

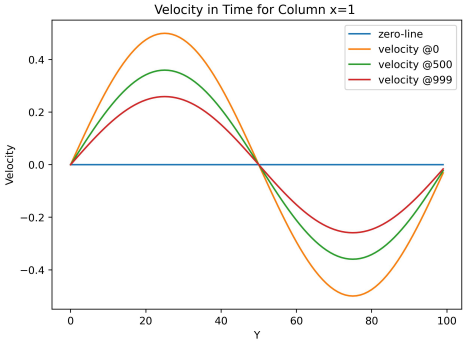


Figure 4.4: Decaying velocity at a specific column.

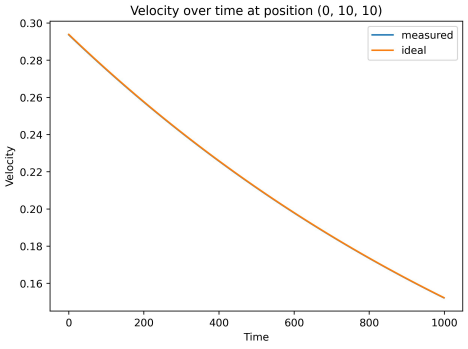


Figure 4.5: measured vs ideal decay

4.1.3 Correlation of Kinematic Viscosity and Omega

This experiment aims to measure the correlation between the kinematic viscosity and the parameter omega, that is used for the collision. The kinematic viscosity describes, how *thick* some fluid is, e.g. syrup has a higher viscosity than water. This experiment measures the viscosity by running the previous experiment section 4.1.2 multiple times with different omegas. It is expected that a lower omega has a higher viscosity. The results of the experiment are shown in fig. 4.6.

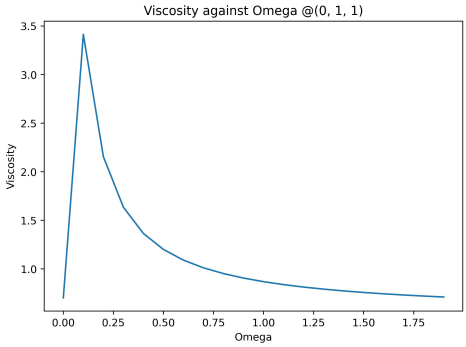


Figure 4.6: Correlation between the kinematic viscosity and omega.

Strangely, for very small omega, the viscosity does not follow the otherwise exponential decay. This is due to the model of the simulation, that fails to replicate the exact behaviour with too large or small values. In theory, this may even have an impact on the experiments, but omega was set to 1.0 to not have a negative impact of this phenomenon.

4.2 Couette Flow

The *Couette Flow* arises in a bounded field with two distinct boundaries located at the top and bottom. Notably, the lower boundary remains stationary, while the upper boundary is sliding in x-direction. Periodic boundary conditions are enforced along the right and left sides. The initial condition, depicted in fig. 4.7 and can be mathematically expressed as follows:

$$\rho(0) = 0$$
$$\mathbf{u}(0) = 0\cdot$$

Because of the friction of the boundaries, the fluid starts flow. More specifically, the fluid in the top starts to get momentum, because it is close to the upper boundary which is moving. This flow then spreads further down. But because the wall at the bottom is steady, the flow gets less the

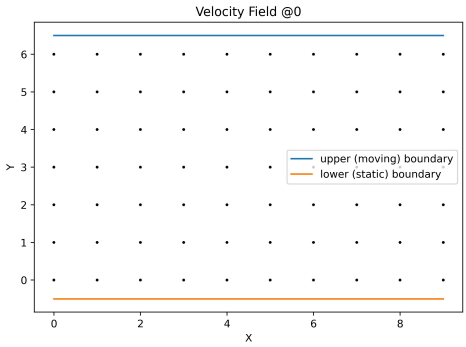


Figure 4.7: Initial condition with a sliding boundary at the top, periodic boundary conditions to the side and a hard wall to the bottom.

further it deepens. This process can be seen in the following two graphs in fig. 4.8.

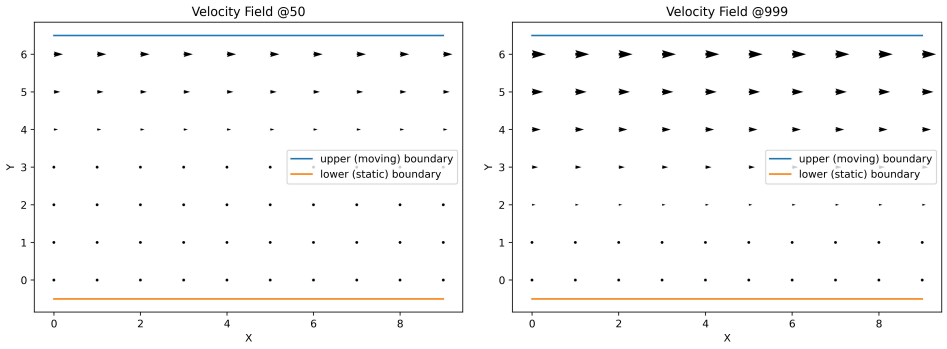


Figure 4.8: Velocity field of Couette Flow over time.

To reproduce this experiment, please use the parameters from table 4.2.

Parameter	Value
L_x	10
L_y	10
ω	1.0
ϵ	0.01
sliding u	-0.1
sliding rho	1
$t_{\max}$	1000

Table 4.2: Parameters for the Couette Flow

5

Conclusion

6

Chapter 1

This is an example of a citation [1]. The corresponding paper can be found in the bibliography section at the end of this document.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis risus ante, auctor et pulvinar non, posuere ac lacus. Praesent egestas nisi id metus rhoncus ac lobortis sem hendrerit. Etiam et sapien eget lectus interdum posuere sit amet ac urna.

Example of normal equation

$$f_i(\mathbf{x}_j + \mathbf{c}_i \cdot \Delta t, t + \Delta t) = f_i(\mathbf{x}_j, t) - \omega \left(f_i(\mathbf{x}_j, t) - f_i^{\mathrm{eq}}(\mathbf{x}_j, t) \right) \tag{6.1}$$

Example of aligned equation:

$$\rho(\mathbf{x}_j, t) = \sum_i f_i(\mathbf{x}_j, t) \tag{6.2}$$

$$\mathbf{u}(\mathbf{x}_j, t) = \frac{1}{\rho(\mathbf{x}_j, t)} \sum_i \mathbf{c}_i f_i(\mathbf{x}_j, t) \tag{6.3}$$

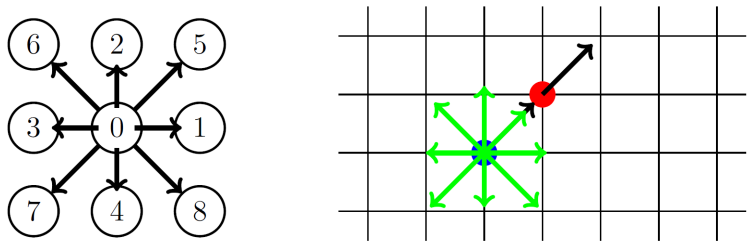
6.1 section title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis risus ante, auctor et pulvinar non, posuere ac lacus. Praesent egestas nisi id metus rhoncus ac lobortis sem hendrerit. Etiam et sapien eget lectus interdum posuere sit amet ac urna. Aliquam pellentesque imperdiet erat, eget consectetur felis malesuada quis. Pellentesque sollicitudin, odio sed dapibus eleifend, magna sem luctus turpis.

- Example of a list
- Example of a list
- Example of a list

7

Chapter 2



(a) (b)

Figure 7.1: example figure

7.1 Section title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. id convallis magna eros nec metus. Sed vel ligula justo, sit amet vestibulum dolor. Sed vitae augue sit amet magna ullamcorper suscipit. Quisque dictum ipsum a sapien egestas facilisis.

Table 7.1: Sample table

S. No.	Column#1	Column#2	Column#3
1	50	837	970
2	47	877	230
3	31	25	415
4	35	144	2356
5	45	300	556

7.2 Code listing

here we provide a short example of code listing. For further information you can take look here:

`https://www.overleaf.com/learn/latex/code_listing`

This is just meant to used if you think that there is some relevant part of code to be shown. Please do not append your whole implementation in the report.

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None # to become the incidence matrix
    VT = np.zeros((n*m,1), int)  # dummy variable
```

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Lorem ipsum list:

Bibliography

[1] Krüger Timm, H Kusumaatmaja, A Kuzmin, O Shardt, G Silva, and E Viggen. *The lattice Boltzmann method: principles and practice.* Springer: Berlin, Germany, 2016.