An Introduction to Lattice-Boltzmann Methods

Gabriel d'Andrade Furlanetto*

Numerically solving the Navier-Stokes Equation (NSE) is necessary to understand a wide range of phenomena: from the airflow around the wing of an airplane to heat exchange mechanisms in nuclear reactors, a lot of modern technology is contingent on accurate simulations of fluid flow. The field of Computational Fluid Dynamics (CFD) exists essentially to solve this problem, and encompasses a wide array of different methods, a number of which (*e.g.* the Finite Difference, Finite Volume, and Finite Elements Methods) attempt to directly compute the macroscopic variables of interest (*e.g.* mass, momentum, pressure). In this paper, we will explore the Lattice Boltzmann Method (LBM), which employs methods from kinetic theory to instead simulate streaming and collisions of particle populations on a discretized space of velocities. To do this, we briefly discuss why this type of simulation is indeed equivalent to a simulation of a fluid by outlining the most important elements of Chapman-Enskog Analysis, discussing overarching advantages and problems associated with it, and then we will go over the practical implementation of the method in detail, focusing on the concrete case of flow around a cylinder and showing that the simulations display effects dependent on the Reynolds number that one would expect in typical CFD. Finally, we will mention extensions and modifications of the method that allow it to be applied to many different fields within physics.

I. INTRODUCTION

It should come as no surprise that non-linear Partial Differential Equations (PDEs) are a very hard problem: To many standard [1, 2] textbooks that focus on solving PDEs, non-linearity is spoken about in hushed tones, apologetically mentioning that they are outside the scope of this mortal textbook and treating them with the solemnity, fear and respect that Ishmael has for the Great White Whale in Moby Dick. The reasons for this are manifold, but the lack of any analytic solution (outside a few special cases) and the multitude of numerical complications that can arise act as a strong deterrent to the treatment of these topics.

While non-linearities come up naturally in different fields of physics, it is truly at the heart of fluid dynamics: no matter which way you slice it, the Navier-Stokes Equations (NSE) is non-linear. Given what was said above about the difficulty of solving these equations, the reader might be

* gabdandrade@usal.es

starting to panic about the last time they were in an airplane, but thankfully this fear is misplaced, since even though the problem is not an easy one to solve, a whole field has emerged around the numerical aspects of finding these solutions, commonly called Computational Fluid Dynamics (CFD).

In this very broad area of study, a lot of different methods exist that directly attempt to solve the NSE, amongst them Finite Difference method (FDM), Finite Volume method (FVM) and Finite Elements method (FEM), which we will call the *conventional methods* following [3], which are based on discretizing in different ways the macroscopic fluid variables (such as fluid and pressure) and directly tracking and evolving them according to the NSE.

While these conventional methods are very well-suited for a lot of different problems and situations even outside of Fluid Dynamics, we will instead focus on the Lattice Boltzmann method (LBM), which discretizes, in both position and velocity, the distribution of the fluid particles, given by the Boltzmann Equation, without ever using the NSE. While this approach may seem a bit strange, it is very physically well-motivated from kinetic theory and it can be shown to be equivalent to the *weakly compressible* NSE, and has a marked advantage in the ease of basic implementation and parallelization when compared to the standard methods.

To achieve this aim, we will begin this paper with a simple overview of kinetic theory, that will attempt to summarize its equivalence to macroscopic fluid dynamics, along with an outline of the governing equations of LBM and a brief discussion of why this method is so useful in Section II, followed by a more detailed discussion into the implementation of the method in the Julia programming language in Section III with two specific cases as benchmarks, which will then allow us to comment on extensions to this model outside of standard fluid dynamics in Section IV. Finally, we will give some brief conclusions on Section V.

II. THE *'HOWS'* AND *'WHYS'* OF LATTICE BOLTZMANN

A. Kinetic Theory

Kinetic theory exists as a sort of middleground between two possible approaches to describe a fluid: On one side, we have a purely microscopic description, which attempts to describe the movement of individual particles and their interactions and the resulting macroscopic properties of that ensemble; on the other, a purely macroscopic description using the NSE; in the middle, kinetic theory.

While we wish we could give a complete description of kinetic theory, it is in itself the subject of many lecture notes and we will defer a way more complete discussion of the subject to [3]. What we will do here is just give the most minimal introduction possible so that we can actually use LBM

Its main object of study is the Boltzmann function, also called the distribution function, $f(\mathbf{x}, \mathbf{v}, t)$, which is to be understood as the mass density[1] at position $\mathbf{x}$, with velocity $\mathbf{v}$, at an instant in time t. It works almost as a probability distribution function, but is normalized to the total number of particles, M, instead of unity.

Just with this definition, we can use its various moments to extract physical information from it. The zeroth moment gives us the mass density,

$$\rho(\mathbf{x}, t) = \int d\mathbf{v} \quad f(\mathbf{x}, \mathbf{v}, t), \tag{1}$$

while the first moment gives us the momentum density,

$$\rho(\mathbf{x}, t)\mathbf{u}(x, t) = \int d\mathbf{v} \quad \mathbf{v} f(\mathbf{x}, \mathbf{v}, t), \tag{2}$$

where $\mathbf{u}(x, t) = \langle \mathbf{v} \rangle$ is the local mean velocity, what we can actually measure in a physical system, and the second moment would give us the energy. These relations are what we will be using to calculate the macroscopic values in our numerical procedures.

Unfortunately, these are merely definitions and no physical content has been introduced yet into our description. More unfortunately, a detailed account of how to get to the dynamical equations (or even what they are in their non-approximate form) is absolutely and completely outside of the reach of this paper. Instead, we will state the result: The Boltzmann Transport Equation (BTE)

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_x f + \mathbf{a} \cdot \nabla_v f = \Omega(f) \tag{3}$$

where $\Omega(f)$ is the so-called *collision operator*, and such collision operators must verify a few properties to actually be compatible with mass, momentum and energy conservation. From equations (1) and (2), its zeroth, first and second moment must be 0. In kinetic theory, one can find an exact form of what this operator actually is, and it involves a horrendous integral of f, which is completely and utterly unusable for (our) numerical purposes. What we will use instead is the simplest

[1] It's also very usual to define it as the particle density instead of the mass density. The only change this would incur is a different normalization and sets of units.

operator which verifies the aforementioned properties, what is usually called the relaxation time approximation or the Bhatnagar-Gross-Krook (**BGK**)[2] collision operator:

$$\Omega(f) = -\frac{1}{\tau}(f - f^{eq}), \tag{4}$$

where τ is the relaxation time, which is directly related to the viscosity of the fluid, and f^{eq} is the equilibrium distribution[3], which, for our purposes, will be the Maxwell-Boltzmann distribution:

$$f^{eq}(\mathbf{x}, |\mathbf{v}|, t) = \rho \left(\frac{1}{2\pi RT}\right)^{\frac{3}{2}} e^{-\frac{|\mathbf{v}|^2}{2RT}}. \tag{5}$$

Now, before we can move on to how we can numerically solve this problem, one unsolved question remains: Is this equivalent to the NSE? A clue lies in calculating the moments of equation (3). By doing this and assuming that $f = f^{eq}$, one gets the Euler equation, i.e. the NSE for inviscid and adiabatic flow. With this as a clue, Chapman-Enskog analysis was created as a way to systematically recover macroscopic PDEs from the BTE, and it works by doing the aforementioned integration with a perturbed Boltzmann function,

$$f \approx f^{eq} + \varepsilon f^{(1)} + \mathcal{O}(\varepsilon^2), \tag{6}$$

which, as is standard within perturbation theory, will allow us to use equation (3) to be able to express $f^{(1)}$ in terms of f^{eq} and its derivatives, which will then allow us to compute the integration terms up to second order in ε. Leaving all the details to [3], this will then allow us to show that this is equivalent to:

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0, \\ \partial_t(\rho \mathbf{u}) + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nabla \cdot (\eta(\nabla \mathbf{u} + (\nabla \mathbf{u})^T). \\ \text{with } p = \rho c_s^2, \quad \eta = \rho c_s^2 \tau, \end{cases} \tag{7}$$

i.e. the NSE in the weakly compressible regime, $u^2 \ll c_s^2$.

With this, we now have established[4] what equations we have to numerically and why this is equivalent to simulating fluid flow, and now we just need to detail our discretization procedure, which is not really all that usual. This is what we shall dedicate ourselves to now.

[2] While it is often thought that this is an approximation only valid for dilute gases, that is not the case: It is actually fairly robust and fundamentally based the existence of local attractors in collisional dynamics, which is almost always true, and uniqueness of a relaxation time, τ, which is true for systems with short-range interactions. [4] really emphasizes this point.

[3] This is a bit cheaty. To find that there even is an equilibrium distribution that we converge towards and that it is indeed MB, we would need to use the full collision operator assisted by Boltzmann's H-theorem. Unfortunately, we have to limit ourselves to merely stating this as a fact.

[4] Or well, we sketched how one could do it and left details to more complete sources.

B. Discretization

It should be somewhat intuitive that equation(3) is going to be easier, at least in some sense, to discretize than equation (7): At a first glance, there fortunately are no advection terms, which are a major headache for usual FDM and FVM, and even the bothersome terms related to shear viscosity don't seem to play a role there.

However, the way that we do choose to discretize it is very important: While usually when solving a differential equation one would start with discretizing spacetime and the velocity would be treated as a time derivative of position and we would be done, this setup has been deemed inadequate for our purposes. We, instead, start by **discretizing the velocities**.

We denote the discretized velocities as $\mathbf{c}_i$, a discrete and finite set of velocities. Intuitively, this means that when we do introduce a discrete position space, these are the directions a particle can from one point in the grid to another. They all have an associated weights, related to a computation for the equilibrium distribution, w_i, and the collection of weights and velocities $\{\mathbf{c}_i, w_i\}$ is called a velocity set.

It should be obvious that these sets will be larger by the number of dimensions, but it should be said that even for the same dimension we still have different choices we can make with regards to this discretization. For one dimension, we could, for example, consider $\{0, 1, -1\}$, *i.e.* the particle can stay still, move to the right or to the left, called a D1Q3 (one dimension, 3 discrete velocities) or ignore the possibility of staying still and consider D1Q2, $\{-1, 1\}$[5]. In two dimensions, the most widely used set is D2Q9, which includes a stationary velocity, 4 for adjacent node and 4 for diagonals. Both of these are represented in Figure 1 for ease of visualization.

We will now denote $f(\mathbf{x}, \mathbf{c}_i, t) = f_i(\mathbf{x}, t)$, and we will write the discretized equations of (1) and (2) to get:

$$\rho(\mathbf{x}, t) = \sum_i f_i(x, t) \qquad \rho u(\mathbf{x}, t) = \sum_i \mathbf{c}_i f_i(x, t) \tag{8}$$

which is the way to recover macroscopic variables in this simulation. Now, we proceed to discretize the problem in spacetime as well, with discrete spacings of Δx and Δt[6] ,and we start by considering equation (3) with the BGK operator[7]:

[5] These different choices are mostly unimportant for two dimensions, but become extremely relevant in 3, in which ther are genuine computational and accuracy tradeoffs.

[6] This discretization, with a given velocity set, introduces something called the isothermal speed of sound, which is equal to $c_s^2 = \frac{1}{3}\frac{\Delta x^2}{\Delta t^2}$ for any traditional velocity set and comes up in formulas later.

[7] τ, the relaxation time, can be related to both viscosity and the stability of the solution, since it functions as an overrelaxation parameter. By standard analyses, it must be greater than $0.5\Delta t$ for any stability, and the closer it gets to that value, the more unstable the system is. For $\tau > 1$, it is unconditionally stable.

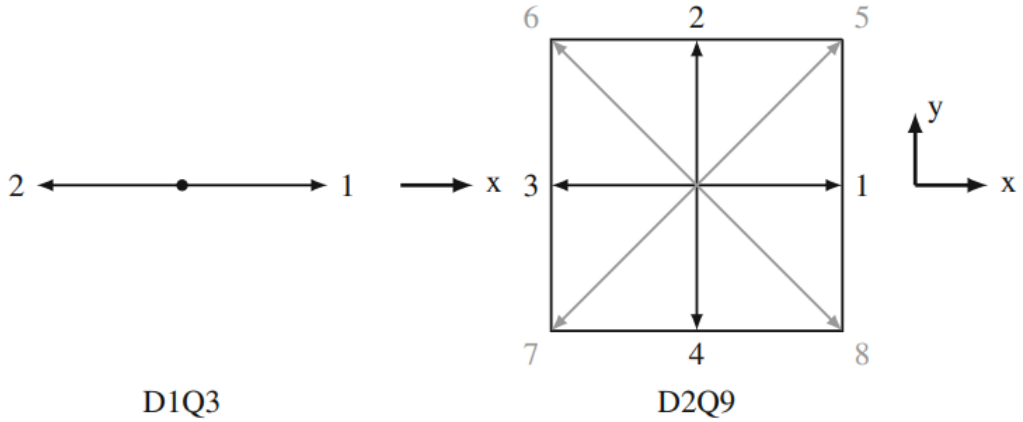

FIG. 1. D1Q3 and D2Q9 velocity sets, taken from [3].

$$f_i(\mathbf{x} + \mathbf{c}_i\Delta t, t + \Delta t) = f_i(x, t) - \frac{\Delta t}{\tau}(f_i - f_i^{eq}). \tag{9}$$

And now we would be *almost* done, if only we knew how to calculate this equilibrium distribution. The naive approach of directly using equation (5) is actually extremely inefficient, because it would require calculations of the second moment of the Boltzmann function and computing exponentials at every step of the simulation, making any possible performance advantage this method would have completely null. Instead, one can use Hermite Polynomials[8], a lot of calculations and some strength of will to find that we can calculate it to very high accuracy with:

$$f_i^{eq}(\mathbf{x}, t) = w_i\rho\left(1 + \frac{\mathbf{u}\cdot\mathbf{c}_i}{c_s^2} + \frac{(\mathbf{u}\cdot c_i)^2}{2c_s^4} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right), \tag{10}$$

where c_s is the isothermal speed of sound. In abstract terms, equations (9) and (10) already provide us with a scheme of solving the Boltzmann equation and simulating fluid flow, but it is standard to actually split (9) into two parts for computational simplicity. We consider first a collision step,

$$f_i^*(\mathbf{x}, t) = f_i(x, t)\left(1 - \frac{\Delta t}{\tau}\right) + f_i^{eq}, \tag{11}$$

and a streaming step,

$$f_i(\mathbf{x} + \mathbf{c}_i\Delta t, t + \Delta t) = f_i^*(\mathbf{x}, t). \tag{12}$$

[8] Where we must once again defer apologetically to [3] for any and all details.

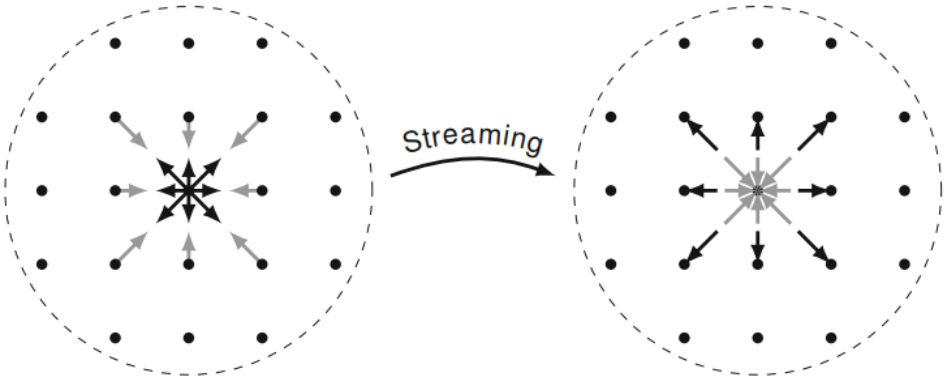

FIG. 2. A visualization of the streaming step of computation from [3], where we see particles from the central node streaming to its neighbours and particles from its neighbours streaming into the central node.

These have much nicer physical intuitions and (sometimes) computational advantages. The first step corresponds to a relaxation towards equilibrium from a state originally out of equilibrium due to collisions. The second, the flow of populations from one node into others, represented in Figure 2.

This is the full outline of the LBM, and our basis for implementation in the latter sections. Before we get there, however, we think its important to at least signal why this method is so widely used.

C. What does this achieve?

A very reasonable question one may pose after going to all this trouble to find these very different sets of equations that apparently also simulate fluid flow could be: Why? The answer is simple but nuanced: The method is easy to implement and can be made highly efficient.

To see why, we must look at where the non-linearities lies in this problem when compared to traditional methods: In traditional methods, nodes are affected by others non-linearly, with the particularly troublesome advective term, $(\mathbf{u} \cdot \nabla)\mathbf{u}$. In Lattice Boltzmann, however, can only lie in the collision step of equation (11), which is completely local to the node, and not in the streaming step (12), which describe how nodes affect others.

This means that the method is uniquely suited for massively parallel high-performance computation: The hard/computationally intensive steps of one node requires no information about other nodes, so all of those calculations can be done in parallel at the same time. Of course, writing this

highly efficient code is non-trivial, and High Performance Computing (HPC) has its very important minutia and is inherently system dependent, but it can be done.

Furthermore, it also shares a couple of features that are very desirable with some traditional methods, such as conserving mass and momentum throughout the entire domain and being well-suited for irregular grids and even being able to implement moving boundaries very easily.

Obviously, the method is not perfect and has its own sets of issues: In the HPC side, the streaming part of the code is very memory-intensive, and this can become a bottleneck for some kinds of simulations; And in the actual physical side, it is not efficient at simulating steady flows or particularly straightforward to generalize to more complex fluid simulations, specially those including temperature.

III. IMPLEMENTING THE LBM AND SOME RESULTS

In this section, we will go over a simple implementation of the LBM to simulate 2-dimensional flow on the Julia programming language, using only its plotting library for convenience. We will discuss 2 different problems, Coutte flow, which has an analytic solution we can use as a benchmark of the method, and the flow around a cylinder, which can illustrate some very interesting fluid dynamics phenomena.

The full code is available as Jupyter notebooks on `https://github.com/BielUsal/LatticeBoltzmann`, the only necessary Julia package to run it is `Plots.jl`.

A. Big ideas and general function definitions

Regardless of the problem we are in, the basic way we approach the problem is going to be very simple. Since Julia is not an object oriented language, but instead uses functional dispatch, we want to define functions such that we can run the simulation. Throughout all of this, we use what are called lattice units, $\Delta x = \Delta t = 1$.

As we discussed in the last section, the actual solution steps are very simple, and will use two functions, `collide` and `stream`, that are based on equations (11) and (12). Furthermore, the collision step and visual representations need the calculation of macroscopic variables, which we can do through a function we will call `macroscopic`. Finally, we will need functions to initialize variables, `InitialConditions`, boundary conditions `BoundaryConditions`[9] and collisions with objects in the

[9] This is the only function we are not going to discuss here, since it is highly specific to the problem.

case of cylinders, `objects`. Our final solver should look like:

```
f,rho,ux,uy = InitialConditions(u0)
for j=1:TimeScale*Nt
    #stream
    f = collide(f,rho,ux,uy)
    f = stream(f)
    f,ux,uy = object(f,ux,uy,cylinder)
    f,ux,uy = BoundaryConditions(f,ux,uy,u0)
    #macroscopic
    rho,ux,uy= macroscopic(f)
    if j%TimeScale == 0
        strength[:,:,Int(j/TimeScale)] = sqrt.(ux.^2 + uy.^2)
        vorticity[:,:,Int(j/TimeScale)] =(circshift(ux[:,:], (-1,0)) -
        ↪  circshift(ux[:,:], (1,0))) - (circshift(uy[:,:], (0,-1)) -
        ↪  circshift(uy[:,:], (0,1)))

    end
end
```

So, our objective now is to define these important functions. We start with implementing the streaming step, which is surprisingly simple with the use of a function from the standard library, `circshift`, which basically sends values at $\mathbf{x}$ to $\mathbf{x} + \mathbf{c}_i$.

```
function stream(f)
    for i=1:q
        f[:,:,i] = circshift(f[:,:,i],(cx[i],0))
        f[:,:,i] = circshift(f[:,:,i],(0,cy[i]))
    end
    return f
end
```

Then, our job is to implement equation (8), which is very straightforward with some mild headaches of reshaping arrays:

```
function macroscopic(f)
    rho = reshape(sum(f,dims=3),Nx,Ny)
    ux = reshape(sum(f[:,:,i] * cx[i] for i in 1:q)./rho,Nx,Ny)
    uy = reshape(sum(f[:,:,i] * cy[i] for i in 1:q)./rho,Nx,Ny)
    return rho,ux,uy
end
```

And then, without losing hope, we need to consider equations (11) and (10), which we implement simultaneously:

```
function collide(f,rho,ux,uy)
    feq = zeros(Nx,Ny,q)
    for i in 1:q
        feq[:,:,i] =   w[i] *rho .* (
        ↪  3*(cx[i]*ux+cy[i]*uy)+9/2*(cx[i]*ux+cy[i]*uy).^2- 3*(ux.^2 + uy.^2)/2
        ↪  .+1.0)
    end
    f = f - (f - feq)/tau
    return f
end
```

And now, we need to define the initial conditions, which will mostly consist of initializing variables and setting the initial velocity to some value, u_0. While this may seem a bit odd, most fluid dynamics problem aren't concerned with the initial conditions but the boundary conditions, so they quickly become unimportant:

```
function InitialConditions(u0)
    f= zeros(Nx,Ny,q)
    ux = zeros(Nx,Ny)
    uy = zeros(Nx,Ny)
    rho = ones(Nx,Ny)
    for i = 2:Ny-1
        ux[1,i] = u0
```

```
    end
    return f,rho,ux,uy
end
```

Finally, collisions with an object are very simple: Where the object is localized, invert all velocities in the distribution so that they are reflected[10]. It's simple to implement, but confusing. We essentially create a binary array that represents where the object is in the space, select all the points in the distribution at these points, invert them, set velocities there to 0 and done.

```
function object(f,ux,uy,BitArrayObj)
    boundary = f[BitArrayObj,:]
    boundary = boundary[:,[1,6,7,8,9,2,3,4,5]]
    ux[BitArrayObj] .= 0
    uy[BitArrayObj] .= 0
    f[BitArrayObj,:] = boundary
    return f,ux,uy
end
```

And now, once given the details of specific problems, this will update the simulation at each point in time.

B. Problem specific definitions

The method is general and can be used for any gridsize, velocity set and boundary conditions. However, both problems use the same velocity set, which we define here:

```
#definition of the D2Q9 velocity set
q = 9
cx = [0,0,1,1,1,0,-1,-1,-1]
cy = [0,1,1,0,-1,-1,-1,0,1]
w = [4/9,1/9,1/36,1/9,1/36,1/9,1/36,1/9,1/36]
```

[10] There is more than one way to do this, but this is the simplest.

1. Couette Flow

Couette flow is one of the few problems in fluid mechanics with an analytic steady state solution. It consists of two dimensional fluid between two plates in the separated by h in the y direction, one moving in the x direction with velocity U and the one below remains stationary. Its analytic solution is:

$$u(y) = U\frac{y}{h} \tag{13}$$

We will consider the following conditions, given the simplicity of the problem:

```
Nx = 100
Ny =50
tau=1.05
Nt = 8000
u0 = 0.0
```

The more complicated part is the boundary conditions, since we need to implement a moving boundary. Boundary conditions are usually complicated, and the details are, again, left to [3], but we will be using so-called bounce-back method, which are based on inverting the velocities and calculating the equilibrium distribution for the moving wall, and implies that:

$$f_{\hat{i}}(\mathbf{x}_{boundary}, t + \Delta t) = f_i^*(\mathbf{x}_{boundary}, t) - 2w_i\rho_w\frac{\mathbf{c}_i \cdot \mathbf{u}_w}{c_s^2} \tag{14}$$

```
function BoundaryConditions(f,ux,uy,u0)
    #bottom
    f[:,1,2] = f[:,1,6]
    f[:,1,3] = f[:,1,7]
    f[:,1,9] = f[:,1,5]
    ux[:,1] .= 0
    uy[:,1] .= 0
```

```
    #Top
    ux[:,Ny] .= u0
    f[:,Ny,6] = f[:,Ny,2]
    f[:,Ny,7] = f[:,Ny,3] +0.5*(f[:,Ny,4]-f[:,Ny,8]) - 0.5*ux[:,Ny]
    f[:,Ny,5] = f[:,Ny,9] + 0.5*(f[:,Ny,8]-f[:,Ny,4]) + 0.5*ux[:,Ny]
    return f,ux,uy
end
```

The results will be represented in the following sections.

2. Flow around a cylinder

In this problem, we are considering the flow inside of a rectangular box with a cylinder centered along the y direction and 1/4th of the way through the x direction. It has an inlet on the left, which effectively sets the velocity on the left, and an outlet at the right, through which flow goes out and sets the velocity to 0 there.

For this, we choose larger values for N_x and N_y for higher definitions, and for the simulation we will vary values of τ to study different regimes of the flow. We also use a one-liner to define the cylinder object we will use.

```
Nx = 1000 # number of cells in x direction
Ny =300 # number of cells in y direction
tau=0.86  # relaxation time
Nt = 4000 # number of time steps
u0 = 0.1 # initial velocity
cylinder = BitArray(metric(i,j,Nx/4,Ny/2) < Ny/4 for i in 1:Nx, j in 1:Ny)
```

Again, the boundary conditions are the complicated part. For the top and bottom, we again just use very simple bounce-back conditions which do not give us any issues. The outlet is also very simple and similar to a Neumann condition. The problem lies on the inlet, which we use the wet-node conditions, which are slightly more complicated but still very manageable:

```
function BoundaryConditions(f,ux,uy,u0)
    #Right boundary
    f[Nx,:,8] = f[Nx-1,:,8]
    f[Nx,:,7] = f[Nx-1,:,7]
    f[Nx,:,9] = f[Nx-1,:,9]

    #bottom and top
    f[:,1,2] = f[:,1,6]
    f[:,1,3] = f[:,1,7]
    f[:,1,9] = f[:,1,5]
    f[:,Ny,6] = f[:,Ny,2]
    f[:,Ny,7] = f[:,Ny,3]
    f[:,Ny,5] = f[:,Ny,9]
    ux[:,1] .= 0
    ux[:,Ny] .= 0
    uy[:,1] .= 0
    uy[:,Ny] .= 0

    #Left boundary
    for j = 2:Ny-1
        f[1,j,4] = f[1,j,8] + 2*rho[1,j]*u0/3
        f[1,j,3] = f[1,j,7] - 0.5*(f[1,j,2]-f[1,j,6])+rho[1,j]*u0/6
        f[1,j,5] = f[1,j,9] + 0.5*(f[1,j,2]-f[1,j,6])+rho[1,j]*u0/6
        ux[1,j] = u0
        uy[1,j] = 0
    end
    return f,ux,uy
end
```

And this is all we need to solve this problem!

C. Results

1. Couette Flow

For the case of Couette flow, we are able to achieve a steady flow that reproduces the analytic solution. We have represented the magnitude of the velocity field at two different points in time in Figure 3.

To check that this quantitatively reproduces the analytic solution, we have represented a cross section of the of the velocity in the x direction with the analytic solution in Figure 4, and it's very clear that, over time, the velocity profile relax towards the analytic solution and, by the end of the simulation, we have incredible agreement.

As a benchmark, this illustrates to us that the method does indeed work as a solution to the NSE. Of course, there could be myriad other ways in which the method fails that are not captured by this problem, but it at least gives us peace of mind that we are, at the very least, on the right track.

2. Flow around a cylinder

To see more interesting effects of this type of flow, we have decided to represent low and high turbulence situations, corresponding to different lower and higher Reynolds numbers, respectively, which we mostly control through the relaxation time, τ. We have represented the magnitude of the velocity field for early and late times of the simulation of both cases in Figures 5 and 6, as well as the vorticity of the field in Figures 7 and 8.

In the Figures, it's possible to see the presence of what appears to be a von Karmán vortex street[11] , and the vorticity graph is especially enlightening in order to see the difference in flow regimes, where we can see essentially no vorticity in the low Reynolds number flow, while this is very much not the case for anywhere in our other situation.

While this problem is not as clearly a benchmark of the method as the last, it does show that the method is capable of reproducing complex fluid-dynamical behaviour that further helps us gain confidence in it as a general simulation method.

[11] The reflections on the upper and lower walls and short simulation time do make it less of a clear cut example of the phenomena, but we believe the basic idea is still correct.

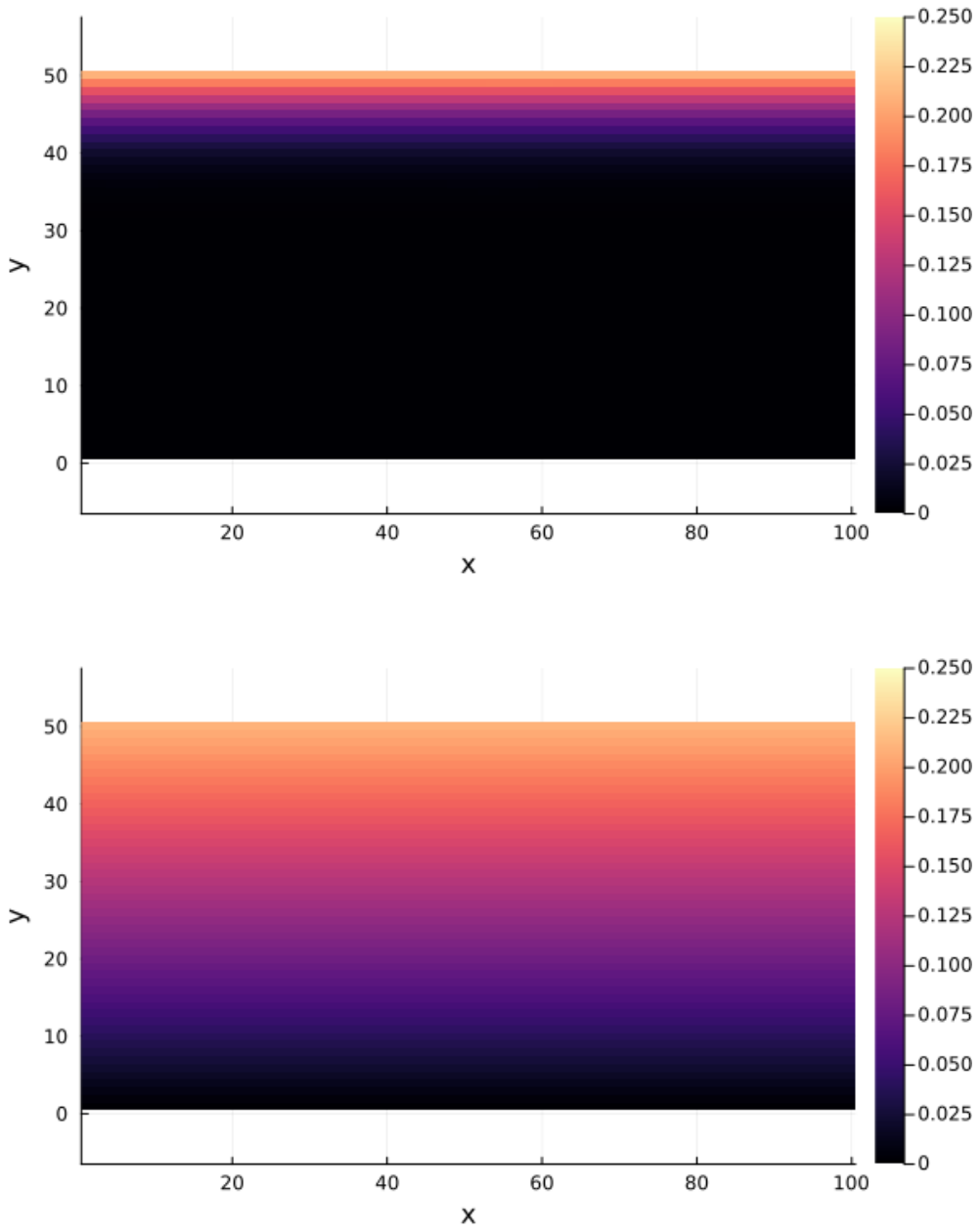

FIG. 3. Magnitude of the velocity field in Couette flow at (Upper) 100 steps and (Lower) 8000 steps. Qualitatively, we can observe that the flow forms a gradient that is independent of x.

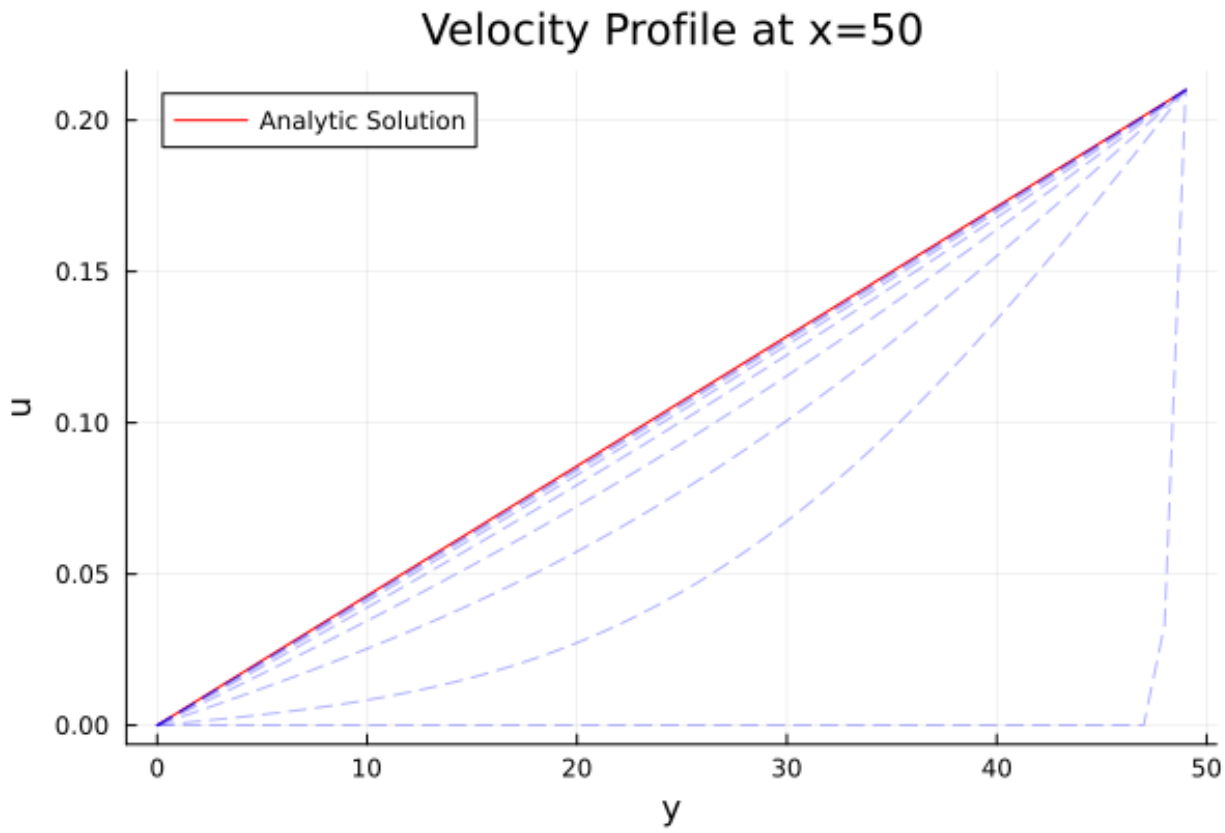

FIG. 4. Velocity profile of Couette flow in different steps of the simulation, from 1 to 8000. It's possible to observe the convergence towards the analytic solution.

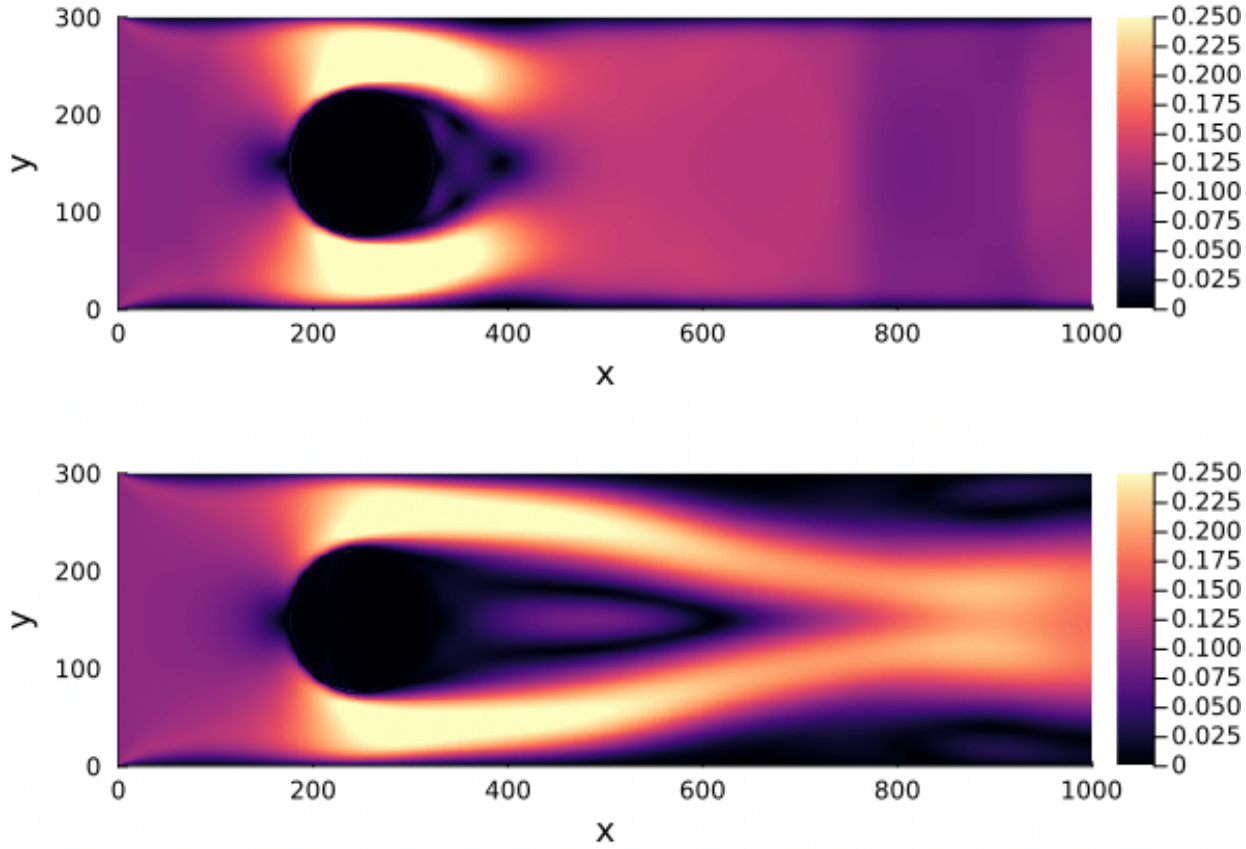

FIG. 5. Magnitude of velocity for a quasi-laminar flow around a cylinder at early (Above) and late (Below) times in the simulation. Qualitatively, we can observe a steady flow behind the cylinder, with a tail of lower velocity right behind it that forms an interface with the higher velocity above, and no vortexes.

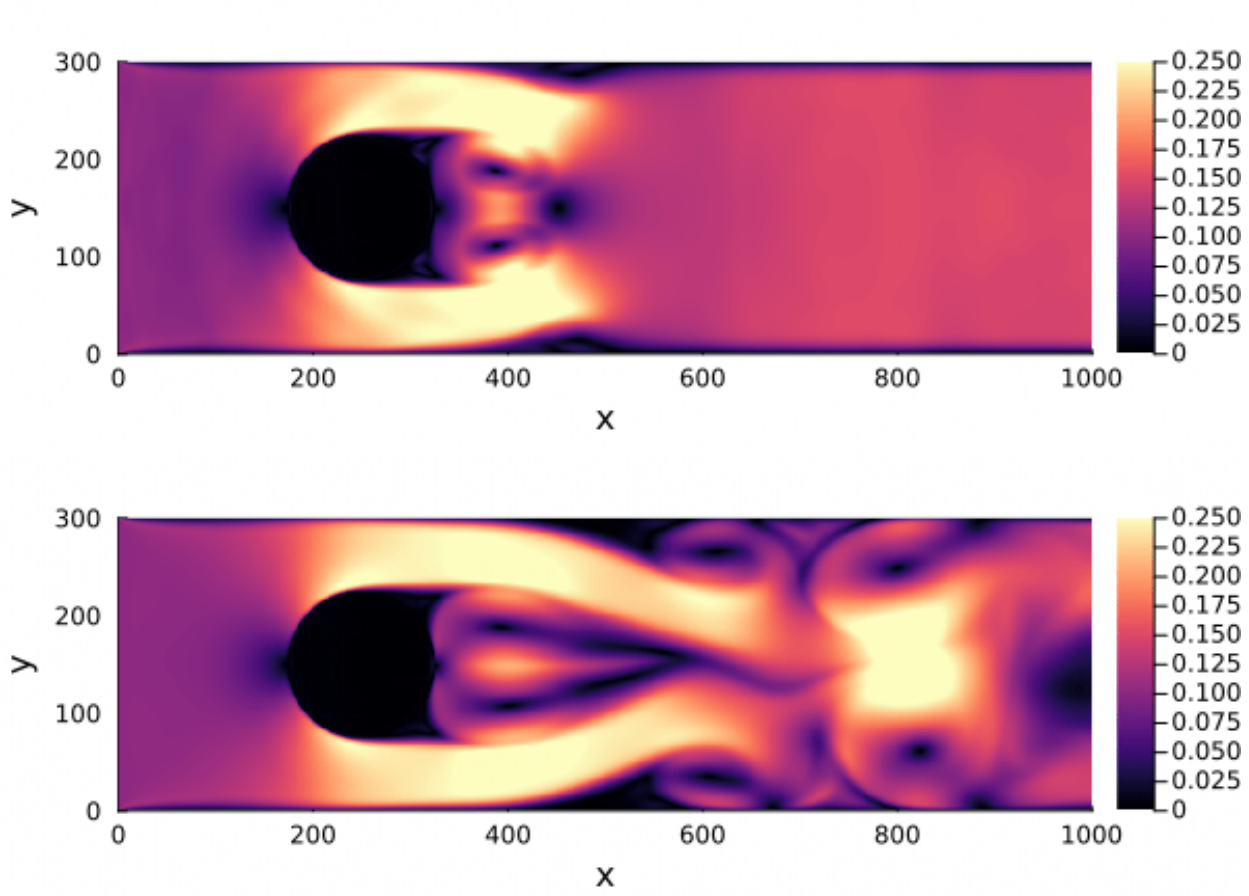

FIG. 6. Magnitude of velocity for a turbulent flow around a cylinder at early (Above) and late (Below) times in the simulation. It's possible to observe the formation of vortexes in both the low-velocity tail of the cylinder and behind it, where they flow chaotically to the outlet.

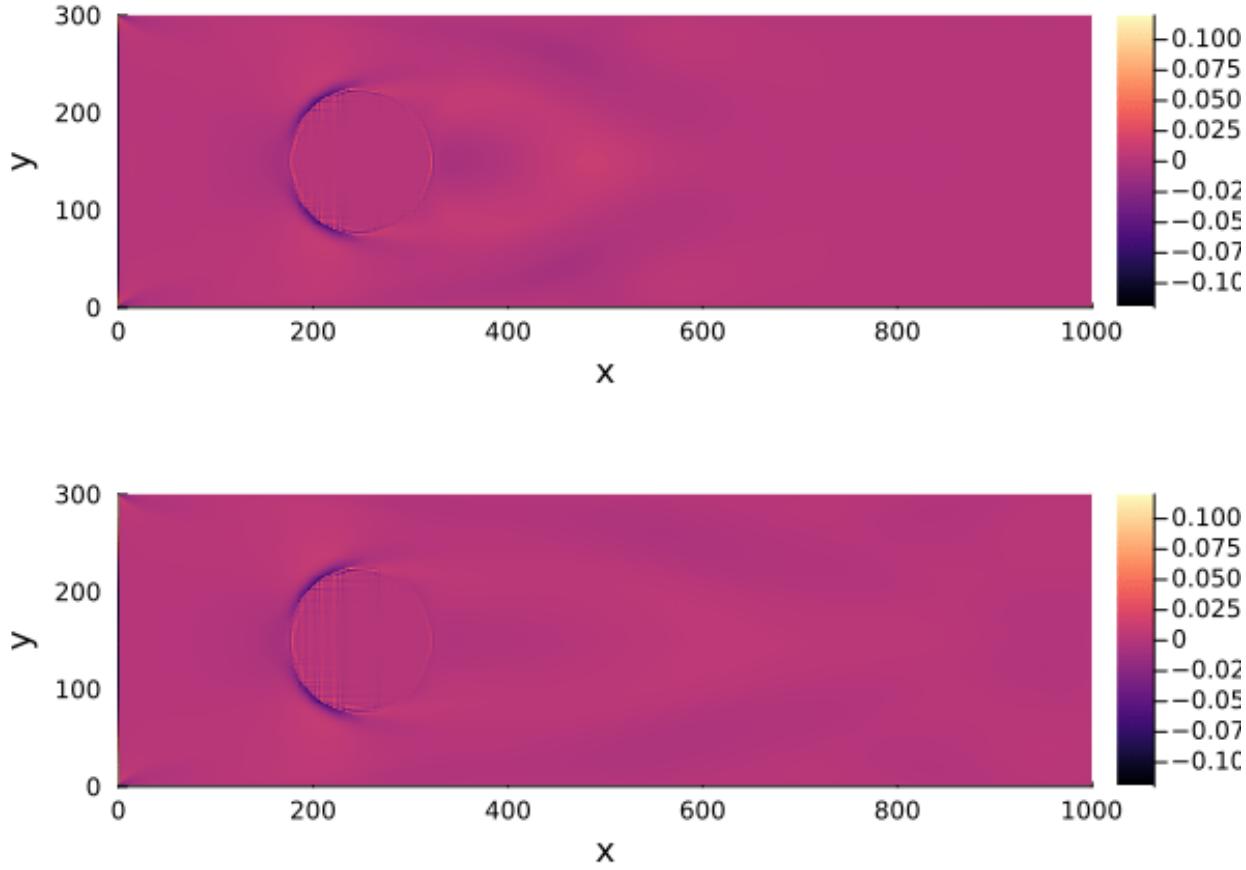

FIG. 7. Vorticity for a quasi-laminar flow around a cylinder at early (Above) and late (Below) times in the simulation. It is clear that the vorticity is essentially zero almost everywhere except the very boundary of the cylinder, which is what we expect in laminar flow.

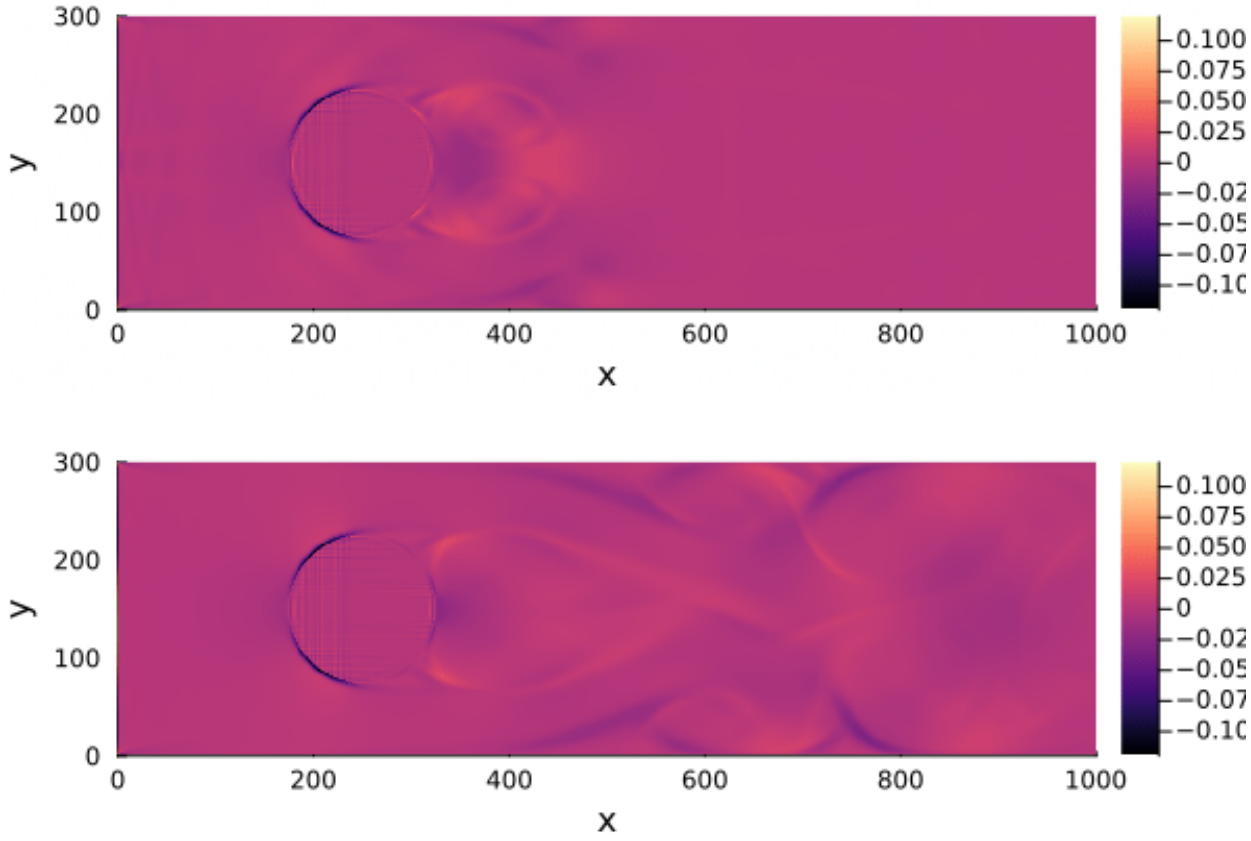

FIG. 8. Vorticity for a turbulent flow around a cylinder at early (Above) and late (Below) times in the simulation. The vorticity definitely not zero for most of the domain: We can see the formation of chaotic vortexes behind the cylinder.

IV. BEYOND LBM

It would be a shame to end this at the results: While we have thoroughly explored a few examples with the LBM, there are many applications of the general ideas in the method that go beyond it in some way or another. Some are slightly different approaches to the same classical problems, while others attempt to solve completely different physics with our beloved methods.

This section will work as almost a collection of these different extensions to the method, to mention their existence and show how broad and powerful LBM can be more than anything. Unless otherwise stated, more details on any of these topics can always be found in [4]

A. Without BGK?

One might wonder if it is possible to do anything by simulating the full non-linear Lattice Boltzmann Equation. The answer is yes... but the details are complicated. Historically, this actually came before the now widely used BGK approximation, and a some progress was made with the fully non-linear equation that was based on the underlying boolean dynamics, *i.e.* each node did have an actual integer number of particles that collided with a collision matrix. The method was successful in its time, but was only useable for low-Reynolds number simulations.

Not long after, the quasi-linear equation was explored in the LB models with enhanced collisions, which allowed for extremely high Reynolds numbers, which had great success in its time but is plagued at the theoretical and implementation levels with what I understand as very bizarre details, such as the appearance of ghost fields very similar to the ones in QFT.

It's spooky stuff! Unfortunately, the reward for this heavy price of deeper understanding and harder implementation is generally worse performance and, while it does have advantages when compared to basic BGK, it is generally inferior to its more sophisticated versions.

B. Beyond BGK?

The main problem with traditional BGK methods is, undoubtedly, the unphysicalness of a single relaxation parameter. It implies that mass, momentum and energy transfers all happen at the same rate, meaning it is constrained to ideal gases if we were to be strict.

A simple improvement was found by using two relaxation times, which allowed for different heat and momentum transfers, a method which later evolved into the Multiple-Relaxation Time approximation, which enhances numerical stability and allows simulation of more realistic liquids, at the cost of $10 - 20\%$ more computational cost for D2Q9 schemes. Ghost pop up here again, it's genuinely a bit scary.

The regularized LGBK takes a slightly different approach, and slightly modifies the collision operator to filter out more problematic contributions of the streaming step before collisions are made. This enhances numerical stability at a relatively low computational cost.

C. Quantum Mechanics?

It is surprisingly not very well-known that Quantum Mechanics admits a hydrodynamical formulation with the so-called Madelung transformation:

$$|\psi| = \sqrt{\frac{\rho(\mathbf{r},t)}{m}}$$
$$\mathbf{u} = \frac{1}{m}\nabla\theta(\mathbf{r},t),$$

that when applied to the Schrödinger equation, yields a hydrodynamical equation:

$$\partial_t\rho + \nabla\cdot(\rho\mathbf{v}) = 0$$

$$\rho\partial_t\mathbf{u} + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla(P_{QP})$$

with

$$P_{QP} = -\rho\nabla\left(\frac{1}{2m}\frac{\nabla^2\sqrt{\rho}}{\sqrt{\rho}}\right).$$

Because of this, it's very reasonable to ask if some variant of the Lattice Boltzmann Method can work here and the answer is a resounding yes, for one dimension. The very surprising thing, however, is that the Quantum Lattice Boltzmann method is not merely a solution to the Schrödinger Equation: Just as the LBM is a solver for the BTE that ends up solving the NSE, the QLBM is a solver for the Dirac Equation that ends up solving the Schrödinger Equation.

By trying to solve the hydrodynamical SE, we end up almost accidentally (and historically, it was an actual accident) solving its relativistic version! The story becomes severely more complicated in higher dimensions, but it can still be done.

Generalizations to Quantum Field Theory and Many-Body situations also exist, and this is a developing field with a lot of exciting news coming out.

D. Relativity?

Relativistic fluids are actually slightly simpler to implement and understand than quantum mechanics in the LBM context. In the end, it's almost exactly the same story with a relativistic BGK operator (There are two, the Marle operator for weakly relativistic fluids and Anderson-Witting operator for fully relativistic systems) and a relativistic equilibrium distribution (the Juttner distribution), which are both slightly more bothersome than BGK but not too terrible.

RLBM has been applied to study Quark-Gluon plasmas, flow of electrons in graphene and much more. A very complete and practically minded review can be found in [5].

Some rumblings can even be heard within the field of a Lattice Boltzmann Method for General Relativity[6], and while it does surely need more results and confirmation, it certainly looks promising!

V. CONCLUSIONS

The Lattice Boltzmann Method presents an interesting strategy to solving the Navier-Stokes Equation: Trembling with fear and solving something else which is much nicer. Of course, not every non-linear PDE will allow for such a brilliancy, but it's very important to study how to proceed when one does indeed allow us to do so.

The big advantage of this method, other than all of the High Performance Computing benefits that are completely beyond the scope of this paper, have been laid bare: It is extremely simple to implement. Some code wizardry that simplifies our lives notwithstanding, all of the implementation is essentially the direct copying of simple formulas that pop out from a very simple discretization scheme, with almost no important caveats that other methods hold.

The problems are illustrative kind of very classic examples that can benchmark fluid simulations, and the results provide a very concise argument in a few pictures that this is a genuine fluid simulation that does indeed work.

Finally, it is important to remember that this is a very active area of research, and modern methods have unbelievable computational and theoretical sophistication with a broad range of applications outside of strict classical and perfect fluids, and hopefully this can serve as a first step towards a fuller understanding of these beautiful methods.

[1] S. J. Farlow, *Partial Differential Equations for Scientists and Engineers* (Dover Publications, 1993).

[2] A. Sommerfield, *Partial Differential Equations in Physics* (Academic Press, 1949).

[3] T. Kruger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen, *The lattice Boltzmann method principles and practice* (Springer International Publishing, 2017).

[4] S. Succi, *The Lattice Boltzmann Equation For Complex States of Flowing Matter* (Oxford University Press, 2018).

[5] A. Gabbana, D. Simeoni, S. Succi, and R. Tripiccione, **863**, 1, 1909.04502.

[6] E. Ilseven and M. Mendoza, Physical Review E **93**, 10.1103/physreve.93.023303 (2016).