TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Maschinenwesen, Institut für Strömungsmechanik, Professur für Strömungsmechanik

Diplomarbeit

Kopplung eines künstlichen neuronalen Netzwerks mit LES-LBM zur Verbesserung einer Windpark-Steuerung

vorgelegt zur Erlangung des akademischen Grades '"Diplomingenieur"'

Henry Torsten Korb

geboren am 02. März 1996

in Ratingen

eingereicht am 00. Monat 2020

1. Gutachter Prof. Dr.-Ing. habil. J. Fröhlich

2. Gutachter Dipl.-Ing. R. Jain

Kurzfassung

Kopplung eines künstlichen neuronalen Netzwerks mit LES-LBM zur Verbesserung einer Windpark-Steuerung

15 zeilen

Abstract

Coupling of an artificial neural network with LES-LBM to improve wind farm control

15 lines

Contents

Nomenclature

Latin Symbols	Unit	Description
$\mathbf{c}$	$\mathrm{m/s}$	Constant Velocity vector
c	$\mathrm{m/s}$	Constant Velocity
f	$\mathrm{kgs^3/m^6}$	Distribution function
E	$\mathrm{J/m^3}$	Energy Density
H	m	Channel half-height
Ma	-	Mach Number
p	Pa	Pressure
R	$\mathrm{J/kg/K}$	Specific gas Constant
Re	-	Reynolds Number
t	s	Time
T	K	Temperature
$\mathbf{u}$	$\mathrm{m/s}$	Macroscopic Velocity Vector
u	$\mathrm{m/s}$	Velocity in streamwise Direction
v	$\mathrm{m/s}$	Velocity in wallnormal Direction
w	$\mathrm{m/s}$	Velocity in spanwise Direction
$\mathbf{x}$	m	Vector of position
x	m	Coordinate in streamwise Direction
y	m	Coordinate in wallnormal Direction
z	m	Coordinate in spanwise Direction

Greek Symbols	Unit	Description
κ	-	Adiabatic Index
μ	$\mathrm{kg/m/s}$	Dynamic Viscosity
ν	$\mathrm{m^2/s}$	kinematic Viscosity
$\boldsymbol{\xi}$	$\mathrm{m/s}$	Microscopic Velocity
ρ	$\mathrm{kg/m^3}$	Density
Ω	$\mathrm{kgs^2/m^6}$	Collision Operator

Indices	Description
τ	Friction
b	Bulk
cp	Centerplane
f	Fluid
m	Mean
pwm	Plane-wise mean
rms	Root-mean square
s	Solid, Sound
w	Wall

Additional Symbols	Description
$\|\mathbf{a}\|$	Euclidian Norm
∇	Nabla Operator
Δ	Step
$\mathbf{a} \cdot \mathbf{b}$	Scalar Product, Matrix multiplication
a'	Fluctuation

Abbreviations	Description
BGK	Bhatnagar-Gross-Krook
DNS	Direct numerical Simulation
LBM	Lattice-Boltzmann-Method
LBE	Lattice-Boltzmann Equation
LES	Large-Eddy Simulation
MRT	Multiple Relaxation Times Operator
NSE	Navier-Stokes-Equations
pdf	Particle-distribution Function

1. Theoretical Background

1.1. Basic Windturbine Stuff

1.1.1. Description

1.1.2. Actuator Line Model

1.1.3. Greedy Control

1.2. Machine Learning for a Continuous Control Problem

1.2.1. Reinforcement Learning

Markov Decision Process

The mathematical formulation on which RL is based is the Markov Decision Process (MDP). Its two main components are the agent that given a state $\mathbf{S}_t$ takes an action $\mathbf{A}_t$, and the environment, that responds to the action $\mathbf{A}_t$ with changing its state to $\mathbf{S}_{t+1}$ and giving feedback to the agent in form of a reward R_t. The interaction takes place at discrete time steps t and the sequence of state, action and reward is referred to as the trajectory. The dynamics of the MDP are described by the function p that is defined in (1.1). It defines the probability of the state $\mathbf{s}'$ with reward r occuring, given the state $\mathbf{s}$ and action $\mathbf{a}$. Note that the following derivations will be constricted to finite MDPs, meaning that state and action space are discrete, however the concepts all are transferable to continuous action and state space.

$$p(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \doteq Pr(\mathbf{S}_t = \mathbf{s}', R_t = r|\mathbf{S}_{t-1} = \mathbf{s}, \mathbf{A}_{t-1} = \mathbf{a}) \tag{1.1}$$

For a process to be a Markov Decision Process, the p must only depent on $\mathbf{s}$ and $\mathbf{a}$. Therefore $\mathbf{s}$ must include all information necessary to determine the future behaviour of the environment. This is not limited to information currently present in the environment, when thinking of this in terms of the wind farm problem at hand, the state could include data about wind speeds at the current time but also from time steps in the past. This approach allows to model virtually any interaction as a MDP, simply by including every bit of information from the beginning of time into the state. Obviously this is not feasible and therefore a careful choice of the information in the state is necessary.
The goal of the learning process is to maximize the sum of the rewards in the long run. Therefore

a new quantity is defined, the return G_t that includes not only R_t but also the rewards received in future time steps. While in many applications of RL, the process naturally comes to an end, referred to as the terminal state $\mathbf{S}_T$, in problems of continuous control this is not the case. Therefore the timeline is broken up into episodes. This allows for a finite computation of G_t. A typical formulation of G_t, referred to as a discounted return is given in (1.2). It includes a discount rate γ, that emphasizes rewards in the near future. If $\gamma = 0$, $G_t = R_t$, if $\gamma = 1$, the return is the sum of all future rewards. [14, p. 47- 57]

$$G_t \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} ... = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}, \quad \gamma \in [0, 1] \tag{1.2}$$

Now the goal of the learning process is defined, but not what to learn. There exist three possible answers to this question: a model of the environment, a value function or a policy. Also combinations of these components is possible. In the case of continuous control, most common approaches are model-free, meaning either learning a value function, a policy or both. Therefore model-based methods will not be discussed further and the reader is referred to the book by Sutton and Barto [14].

The policy π is the mapping from states to actions with a set of adjustable parameters. Under the policy π with its vector of parameters set to $\boldsymbol{\theta}$, the probability of Action $\mathbf{A}_t = \mathbf{a}$ given a state $\mathbf{S}_t = \mathbf{s}$ is denoted as $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$. The value function of a state s under a policy π is denoted as $v_\pi(\mathbf{s})$ and is the expected return if the agent acts according to π, starting from state $\mathbf{s}$. Note that for convenience the parameters of π were be dropped. For MDPs it can be defined as (1.3).

$$v_\pi \doteq \mathrm{E}_\pi \left[G_t | \mathbf{S}_t = \mathbf{s} \right] = \mathrm{E}_\pi \left[\sum_{t'=t}^{T} \gamma^{t'-t} R_{t'} | \mathbf{S}_t = \mathbf{s} \right] \tag{1.3}$$

$$= \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s}) \sum_{\mathbf{s}'} \sum_{r} p(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) \left(r + \gamma v_\pi(\mathbf{s}') \right) \tag{1.4}$$

In the form of (1.4), the equation is referred to as the Bellmann equation and its unique solution is the value function v_π. Analogously, the action-value function is the expected reward of taking action $\mathbf{a}$ at state $\mathbf{s}$ under the policy π, denoted by $q_\pi(\mathbf{s}, \mathbf{a})$. It is defined by (1.5). [14, p. 58-59]

$$q_\pi(\mathbf{s}, \mathbf{a}) \doteq \mathrm{E}_\pi \left[G_t | \mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a} \right] = \mathrm{E}_\pi \left[\sum_{t'=t}^{T} \gamma^{t'-t} R_{t'} | \mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a} \right] \tag{1.5}$$

Policy Gradient Methods

With a known value function, it is possible to construct a policy and by improving the value function, the policy can be improved. However, there exist advantages to directly improve the policy, especially for continuous state and action space. Policy gradient methods use the gradient of a

perfomance measure $J(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$ of a policy. This gradient can be used in optimization algorithms, such as stochastic gradient descent [14, p. 201] or its extensions such as Adam [8]. In its basic form, SGD performs an update according to 1.6. The parameter α is called the learning rate.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{1.6}$$

The policy gradient methods differ now only in the performance measure. The first such algorithm proposed is the REINFORCE algorithm [15]. Its definition of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is presented in (1.7). It follows from the policy gradient theorem and substituting all values of $\mathbf{A}$ and $\mathbf{S}$ with the actions and states from one trajectory. Thus all the values necessary for the computation of the gradient are known. [14, p.324-328]

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathrm{E}_{\pi}\left[\sum_{a} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{S}_t) q_{\pi}(\mathbf{S}_t, \mathbf{a}) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{S}_t)}{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{S}_t)}\right] \tag{1.7}$$

$$= \mathrm{E}_{\pi}\left[G_t \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{A}_t|\mathbf{S}_t)}{\pi_{\boldsymbol{\theta}}(\mathbf{A}_t|\mathbf{S}_t)}\right] \tag{1.8}$$

While this algorithm makes a computation possible, it is inefficient. Therefore newer methods have been devised such as the Trust Region Policy Optimization (TRPO) [12] and the Proximal Policy Optimization (ppo) [11]. They are closely related and both propose a surrogate performance measure that limits the size of the gradient to ensure monotonic improvement in the case of TRPO and a close approximate in the case of ppo. However, the computation of the surrogate in ppo is simpler and more efficient, which helped policy gradient methods to become one of the most used algorithms in continuous control problems. One version of the surrogate performance measure, which is also referred to as objective, is given in (1.10). The probability ratio r_t compares the policy after the update to the policy before the update. Therefore $\pi_{\boldsymbol{\theta}}$ has to be approximated. $a_{\pi}(\mathbf{A}_t|\mathbf{S}_t)$ is an estimator of the advantage function, which is defined as $a_{\pi}(\mathbf{A}_t|\mathbf{S}_t) = q_{\pi}(\mathbf{A}_t|\mathbf{S}_t) - v_{\pi}(\mathbf{S}_t)$. This estimator also has to be found, however, this is a regular optimization problem, which can be solved via SGD or Adam.

$$r_t(\boldsymbol{\theta}) \doteq \frac{\pi_{\boldsymbol{\theta}}(\mathbf{A}_t|\mathbf{S}_t)}{\pi_{\boldsymbol{\theta}_{old}}(\mathbf{A}_t|\mathbf{S}_t)} \tag{1.9}$$

$$J \doteq \mathrm{E}_{\pi}\left[\min\left(r_t(\boldsymbol{\theta}) a_{\pi}(\mathbf{A}_t|\mathbf{S}_t), \mathrm{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) a_{\pi}(\mathbf{A}_t|\mathbf{S}_t)\right)\right] \tag{1.10}$$

In addition to the ratio probability clipping, other regularizations can be added to the objective function, for example an l_2 regularization, that adds a penalty proportional to the size of $\boldsymbol{\theta}$.

Table 1.1.: Activation functions commonly used in neural network

Name	Domain	Range	Definition
$\tanh$	$(-\infty, +\infty)$	$(-1, -1)$	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^- x}$
σ	$(-\infty, +\infty)$	$(0, -1)$	$\sigma(x) = \frac{1}{1+e^{-x}}$
relu	$(-\infty, +\infty)$	$[0, +\infty)$	$\mathrm{relu}(x) = \max(x, 0)$

1.2.2. Artificial Neural Networks

Feed-forward networks

In the section above, a way to update parameters of the policy or value function were described, however, no description of the function itself was given. In principal any function with a set of parameters can be used, but usually, an artificial neural network (ANN) is used. ANNs are comprised of layers of neurons. More precisely a layer k of § neurons takes as input a vector $\mathbf{p}$ of length R. The output of the layer is a new vector $\mathbf{a}^k$, wich then is the input for the next layer of the network. In a simple feed-forward layer a_j^k is computed according to (1.11). The entries $w_{i,j}^k$ of the matrix $\mathbf{W^k}$ are called the weights of the layer and the entries b_j^k of the vector $\mathbf{b}^k$ are called its bias. f is called the activation function, which can be chosen freely, but typical choices include the hyperbolic tangent (tanh), the sigmoid-function (σ) or the rectified linear unit (relu) function. A small overview over some of the key features of these functions is given in Table 1.1. Thus a network of two layers with $\tanh$ as an activation function describes the function given in (1.12), with $\mathbf{p}$ being the input to the network and $\mathbf{a}$ its output and the activation function being applied element-wise to the vectors. [2, p. 2-2 - 2-12].

$$a_j^k = f(w_{i,j}^k p_i^k + b_j^k) \tag{1.11}$$

$$\mathbf{a} = \mathbf{tanh}(\mathbf{b^1} + \mathbf{W^1} \cdot \mathbf{tanh}(\mathbf{b^0} + \mathbf{W^0} \cdot \mathbf{p}) \tag{1.12}$$

Recurrent neural networks

It is obvious that a feed-forward network only produces an output influenced by the current activation, there can be no influence of previous activations for example in a time-series. Yet there exist many applications for which such a feature would be useful, for example the field of natural language processing [5] or transient physical processes. This led to the development of recurrent neural networks, in which the network includes has a hidden state, which is preserved. Especially the development of the long short-term memory (lstm) cell has had great impact on the success of recurrent neural networks due to its ability to preserve hidden states over a longer period of

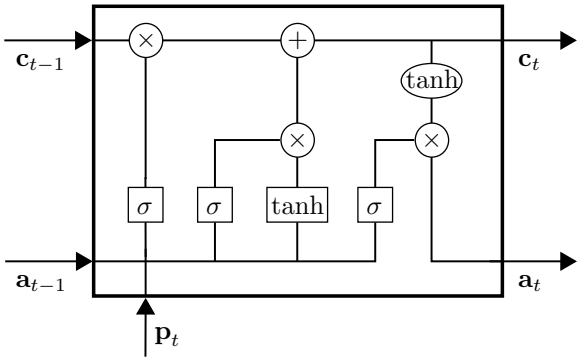

Figure 1.1.: Schematic representation of an LSTM cell, operation in boxes represent layers with trainable weights, operations in ellipses are pointwise operations.

time [6]. An lstm layer consists of one or multiple cells, with the input being the entire or parts of the activation. Each cell passes a vector called the cell state $\mathbf{c}$ as well as its output $\mathbf{a}$ to itself in the next time step. Thus a cell has as inputs at time t the cell state and output of the previous time step, $\mathbf{c}_{t-1}$ and $\mathbf{a}_{t-1}$, as well as the current activation $\mathbf{p}_t$. On the inside, the cell consists of a forget-gate, an input-gate and an output-gate. The forget-gate determines the influence of the cell-state, the input-gate determines the influence of the current time-step on the cell-state. The output-gate then computes the output based on the updated cell-state. A schematic of the cell is given in 1.1. From left to right the sigmoid layers are the forget-, input- and output-gate, whereas the $\tanh$-layer corresponds most closely to the layer of a feed-forward-network.

Training of a Neural Network

To use a neural network as policy in a policy gradient method algorithm, a way to compute the gradient of the objective J has to be found. The parameters of the policy are the weights and biases of the network, therefore the partial derivatives of J with respect to all weights and biases have to be found. Both described policy gradient methods express the objective gradient as a funtion j of the gradient of the policy as shown in (1.13). Under the assumption that each entry in the action vector $\mathbf{A}$ is a statistically indepedent random variable with the probability density function pdf, parameterized by μ_i, the policy can be written as (1.14). Thus the gradient term is (1.15). [14, p. 335]
If μ_i is the output of a K-layered network, it can be written as (1.16). Computing the gradient (1.17) can now be done efficiently via backpropagation. The name refers to the fact, that due to the chain rule, the gradient of a layer is easily expressed through the gradient of the previous layer, which allows for an efficient computation. The sensitivity $s_{i,j}^k$ determines how sensible the action is to

changes of this parameter. [2, p.11-7 - 11-13]

$$\nabla_{\boldsymbol{\theta}} J = j\left(\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{S})}{\pi_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{S})}\right) \tag{1.13}$$

$$\pi_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{S}) = \prod_i pdf(A_i, \mu_i(\mathbf{S}, \boldsymbol{\theta})) \tag{1.14}$$

$$\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{S})}{\pi_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{S})} = \prod_i \frac{\partial pdf(A_i, \mu_i(\mathbf{S}, \boldsymbol{\theta}))}{\partial \mu_i(\mathbf{S}, \boldsymbol{\theta})} \frac{\nabla_{\boldsymbol{\theta}}\mu_i(\mathbf{S}, \boldsymbol{\theta})}{pdf(A_i, \mu_i(\mathbf{S}, \boldsymbol{\theta}))} \tag{1.15}$$

$$\mu_i(\mathbf{S}, \boldsymbol{\theta}) = f^K(b_i^K + w_{i,j}^K f^{K-1}(...f^0(b_l^0 + w_{l,m}^0 S_m)...)) \tag{1.16}$$

$$\nabla_{\boldsymbol{\theta}}\mu_i|_{\mathbf{S}} = \left[\left.\frac{\partial \mu_i}{\partial b_j^k}\right|_{\mathbf{S}}, \left.\frac{\partial \mu_i}{\partial w_{j,l}^k}\right|_{\mathbf{S}}\right]^T \tag{1.17}$$

$$x_i^k = b_i^k + w_{i,j}^k p_j^k, \quad p_i^{k+1} = f^k(x_i^k), \quad p_i^0 = S_i \tag{1.18}$$

$$s_{i,j}^K = \left.\frac{\partial f^K(x)}{\partial x}\right|_{x_i^K} \delta_{i,j}, \quad s_{i,j}^k = s_{i,l}^{k+1} w_{l,j}^k \left.\frac{\partial f^k(x)}{\partial x}\right|_{x_i^k} \tag{1.19}$$

$$\left.\frac{\partial \mu_i}{\partial b_j^k}\right|_{\mathbf{S}} = s_{i,j}^k, \quad \left.\frac{\partial \mu_i}{\partial w_{j,l}^k}\right|_{\mathbf{S}} = s_{i,j}^k p_l^k \tag{1.20}$$

1.3. The Lattice Boltzmann Method

1.3.1. basics

To conduct the training of the agent, it needs to interact with the environment. In the case studied in this thesis the environment is a wind farm with multiple turbines. Therefore the fluid field within the wind park has to be calculated. The traditional approach would be to discretize the Navier-Stokes-Equations (NSE), however, solvers based on the NSE require many CPU-hours. A newer approach is the Lattice Boltzmann Method. The basics of its theory will be layed out in this chapter.
The Lattice Boltzmann Method (LBM) is based on a discretization of the Boltzmann Equation derived from the kinetic theory of gases. This description is often referred to as a mesoscopic description, since it stands in between the scale of continuum theory and the scale of single particles, which will be referred to as macroscopic and microscopic scale respectively. While the NSE describe the medium through density ρ, velocity $\mathbf{u}$ and pressure p, the Boltzmann Equation considers the particle density function (pdf) f, which describes the distribution of particles in six dimensional phase space. It is therefore a function of space $\mathbf{x}$, microscopic velocity $\boldsymbol{\xi}$ and time. However, the macroscopic quantities can easily be recovered from the pdf by taking its zeroth and first moments in velocity space, as shown in (1.21) and (1.22). Similarly the energy can be found by the third moment. The pressure can not be recovered directly, instead it is calculated by a state equation such as the ideal gas law. The pdf tends towards an equilibrium, which is given by the Maxwell equilibrium. It can be described by only the macroscopic quantities and is shown in (1.23), with R being

the specific gas constant and T the temperature. The velocity $\mathbf{v}$ is defined as $\mathbf{v} = \boldsymbol{\xi} - \mathbf{u}$.[9, p. 15- 21] The Boltzmann Equation describes the change of the pdf over time. The change in time is governed by the collision operator Ω. It describes the change of f due to collisions of particles. In the Boltzmann Equaiton it is an integral operator, that accounts for all possible collisions. Therefore it is mathematically rather cumbersome which renders it unuseful for numerical discretization, thus in LBM another formulation for Ω is used, which is an active area of research in the LBM community [1]. The Boltzmann Equation is shown in (1.24). The first form is the total derivative of f with repsect to time. The second term in the second form is a convection term, while the third term corresponds to a forcing term. In comparison to the NSE the Boltzmann Equation lacks a diffusive term. Diffusion occurs only through the collision operator. [9, p. 21]

$$\rho(\mathbf{x}, t) = \int_{-\infty}^{\infty} f(\mathbf{x}, \boldsymbol{\xi}, t) \mathrm{d}\boldsymbol{\xi} \tag{1.21}$$

$$\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \int_{-\infty}^{\infty} \boldsymbol{\xi} f(\mathbf{x}, \boldsymbol{\xi}, t) \mathrm{d}\boldsymbol{\xi} \tag{1.22}$$

$$f^{eq}(\mathbf{x}, \boldsymbol{\xi}, t) = \rho \left(\frac{1}{2\pi RT} \right) e^{-\|\mathbf{v}\|^2/(2RT)} \tag{1.23}$$

$$\frac{Df(\mathbf{x}, \boldsymbol{\xi}, t)}{Dt} = \frac{\partial f(\mathbf{x}, \boldsymbol{\xi}, t)}{\partial t} + \frac{\partial f(\mathbf{x}, \boldsymbol{\xi}, t)}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial t} + \frac{\partial f(\mathbf{x}, \boldsymbol{\xi}, t)}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial t} = \Omega(f) \tag{1.24}$$

In order to use the Boltzmann Equation for numerical simulations it needs to be discretized in space, velocity and time and a collision operator has to be defined.

The velocity discretization is based on the Hermite Series expansion, since its generating function is of the same form as the equilibrium distribution. To recover the first three moments of the distribution correctly, i.e. density, velocity and energy, truncation of the series after the first three terms is sufficient and the roots of the Hermite polynomials up to order three are the necessary discrete velocities $\boldsymbol{\xi}_i$. Together with the weights w_i, that are used in the Gauss-Hermite quadrature, the velocities form a velocity set. Velocity sets are denoted in the DdQq-notation, with d being the number of spatial dimensions and q the number of discrete velocities. In three dimensions D3Q15, D3Q19 and D3Q27 can be used to recover the Navier-Stokes-Equations, but only D3Q27 is suitable for high Reynolds number flows [7].[9, p. 73-93]

Time is discretized via the explicit Euler forward scheme, which can be shown to be second order accurate. Space is discretized uniformly into a cubic grid, so that discrete pdfs move from one node of the grid to the other in one timestep. The ratio of timestep to lattice width is called the lattice speed of sound c_s. The timestep is usually separated into two parts, the streaming step, in which populations are advected from one node to the other, and the collision step, in which the collision operator is applied. Applying the entire discretization gives the Lattice Boltzmann Equation (LBE), which is given in (1.25), with $\mathbf{c}_i$ being $\boldsymbol{\xi}_i/\sqrt{3}$ for convenience. It is, compared to discretized NSE, very simple and the separation into collision and streaming makes it easily parallelizible. Furthermore the equations to compute density and velocity are given in (1.26) and (1.27). [9, p. 94-98]

$$f_i(\mathbf{x} + \mathbf{c}_i\Delta t, \mathbf{c}_i, t + \Delta t) = \Omega(f)_i + f_i(\mathbf{x}_i, \mathbf{c}_i, t) \tag{1.25}$$

$$\rho(\mathbf{x}, t) = \sum_i f_i(\mathbf{x}, t) \tag{1.26}$$

$$\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \sum_i \mathbf{c}_i f_i(\mathbf{x}, t) \tag{1.27}$$

Lastly a collision operator omega has to be found. The first applicable collision operator proposed is the Bhatnager-Gross-Krook operator. It is based on the fact, that the distributions tend to the equilibrium distribution, therefore the BGK operator relaxes the distributions towards equilibrium with a constant relaxation frequency ω. This leads to (1.28). Via Chapmann Enskog analysis this relaxation frequency is related to the kinetic viscosity ν by (1.29). [9, p. 98-100, 112]

$$\Omega_i^{BGK} = \omega\left(f_i - f_i^{eq}\right) \tag{1.28}$$

$$\nu = c_s^2\left(\frac{1}{\omega} - \frac{\Delta t}{2}\right) \tag{1.29}$$

While the BGK operator is sufficient in low Reynolds number flows, it becomes unstable at higher Reynolds numbers. Therefore more sophisticated methods were found. One approach is to transform the distributions into moment space and relaxe the moments independently, leading to multiple relaxation time (MRT) methods. Geier et al. argue, that, since these moments are not statistically independent, they cannot be relaxed separately. However, cumulants of a distribution are statistically independent by design, therefore they can also be relaxed indepentdently. Furthermore, they fulfill Galilean invariance, which is not always the case for MRT methods. To transform the discrete distributions into cumulant space, the discrete distributions are redefined in terms of continuous velocity and then transformed into frequency space for Galilean invariance. From the transformed distributions the cumulants are generated. These quantities can now be relaxed independently, although usually only a few cumulants are relaxed. It can be shown that with a parametrization this method can be fourth order accurate. However in underresolved flows, this parametrization is used to add numerical diffusivity, acting like an inexplicit subgrid scale model for Large-Eddy-Simulations (LES). Yet the exact behaviour of the underresolved cumulant operator is not yet known.

1.3.2. boundary conditions

2. Setup of Simulation

2.1. interaction

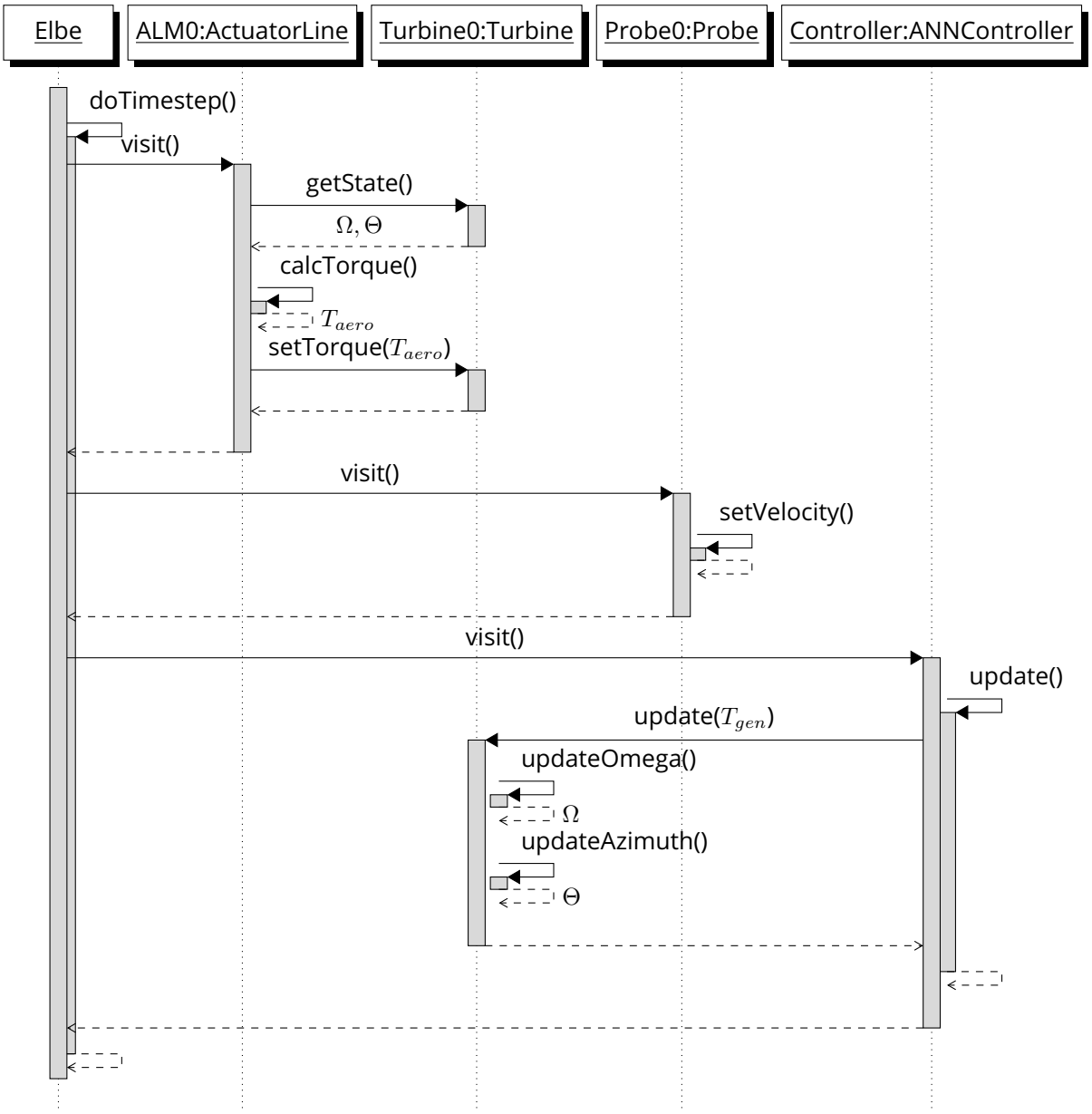

Figure 2.1.: UML sequence diagram of simulation timestep.

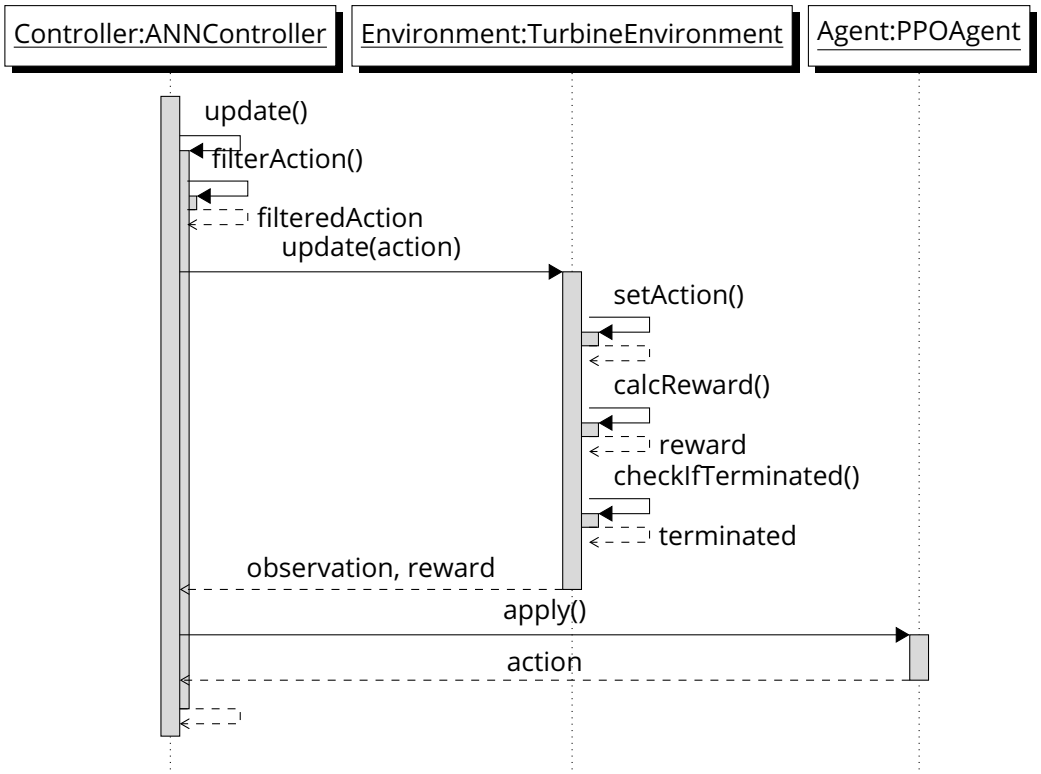

Figure 2.2.: UML sequence diagram of update step of controller.

Bibliography

[1] C. Coreixas, B. Chopard, and J. Latt. “Comprehensive comparison of collision models in the lattice Boltzmann framework: Theoretical investigations”. en. *Phys. Rev. E* 100.3 (Sept. 2019), p. 033305. URL: https://link.aps.org/doi/10.1103/PhysRevE.100.033305 (visited on 02/28/2020).

[2] H. B. Demuth et al. *Neural Network Design*. 2nd. Stillwater, OK, USA: Martin Hagan, 2014.

[3] M. Geier, A. Pasquali, and M. Schönherr. “Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part I: Derivation and validation”. en. *Journal of Computational Physics* 348 (Nov. 2017), pp. 862–888. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999117304230 (visited on 02/28/2020).

[4] M. Geier et al. “The cumulant lattice Boltzmann equation in three dimensions: Theory and validation”. en. *Computers & Mathematics with Applications* 70.4 (Aug. 2015), pp. 507–547. URL: https://linkinghub.elsevier.com/retrieve/pii/S0898122115002126 (visited on 02/28/2020).

[5] S. Ghosh et al. “Contextual LSTM (CLSTM) models for Large scale NLP tasks”. en (Feb. 2016). URL: https://arxiv.org/abs/1602.06291v2 (visited on 03/14/2020).

[6] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. en. *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735 (visited on 03/03/2020).

[7] S. K. Kang and Y. A. Hassan. “The effect of lattice models within the lattice Boltzmann method in the simulation of wall-bounded turbulent flows”. en. *Journal of Computational Physics* 232.1 (Jan. 2013), pp. 100–117. URL: http://www.sciencedirect.com/science/article/pii/S0021999112003968 (visited on 04/06/2020).

[8] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. en. *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 02/13/2020).

[9] T. Krüger et al. *The Lattice Boltzmann Method: Principles and Practice*. en. Graduate Texts in Physics. Cham: Springer International Publishing, 2017. URL: http://link.springer.com/10.1007/978-3-319-44649-3 (visited on 03/31/2020).

[10] K. Kutscher, M. Geier, and M. Krafczyk. “Multiscale simulation of turbulent flow interacting with porous media based on a massively parallel implementation of the cumulant lattice Boltzmann method”. en. *Computers & Fluids* 193 (Oct. 2019), p. 103733. URL: http://www.sciencedirect.com/science/article/pii/S0045793018300653 (visited on 03/26/2020).

[11] J. Schulman et al. “Proximal Policy Optimization Algorithms”. en. *arXiv:1707.06347 [cs]* (Aug. 2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 02/12/2020).

[12] J. Schulman et al. “Trust Region Policy Optimization”. en. *arXiv:1502.05477 [cs]* (Apr. 2017). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477 (visited on 02/27/2020).

[13] M. Schönherr. “Towards reliable LES-CFD computations based on advanced LBM models utilizing (Multi-) GPGPU hardware”. PhD Thesis. July 2015. URL: http://www.digibib.tu-bs.de/?docid=00062945.

[14] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. en. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[15] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. en. *Mach Learn* 8.3-4 (May 1992), pp. 229–256. URL: http://link.springer.com/10.1007/BF00992696 (visited on 02/27/2020).

A. The Cumulant Lattice Boltzmann Method

A.1. Derivation of cumulants

To provide a more detailed explanation of the cumulant lattice boltzmann method, a more thorough derivation is given here. First, the discrete distribution is written as in a continuous form as shown in (A.1). This is transformed to frequency space, to make the cumulants independent of frame of reference as shown in (A.2). This distribution is then used in the cumulant generating function, displayed in (A.4). However, from that form, the cumulants are not directly computable. Comparing the cumulant generating function to the moment generating function in (A.3) shows their similarity. Computing the cumulants of a arbitrary function shows how they can be expressed in terms of moments, two examples for second order cumulants are given in (A.5) and (A.6). Cumulants can also be computed from central moments, which is requires less computations [4]. Cumulants of zeroth and first order stay constant throughout the collision. The cumulants upwards from order two can now be relaxed indivdually like (A.7), therefore each has its own relaxation frequency. The first frequency ω_1 governs the shear viscosity and ω_2 the bulk viscosity of the fluid by the same relation as (1.29). The rest of the frequencies can be chosen freely and are usually set to one, resulting fully relaxing the cumulants, however a parametrized version of the collision operation exists, that changes some of the higher order relaxation frequencies, leading to fourth order accurate diffusion in a fully resolved simulation [3]. In underresolved simulations the parameter introduced in the parametrization can be used to influence the diffusivity and therefore acting as an implicit subgrid scale model, however so far only empirical evidence exists for this.[4]

$$f(\xi, \upsilon, \zeta) = \sum_{i,j,k} f_{i,j,k} \delta(ic_s - \xi)\delta(jc_s - \upsilon)\delta(kc_s - \zeta) \tag{A.1}$$

$$F(\boldsymbol{\Xi}) = \int_{-\infty}^{\infty} f(\xi, \upsilon, \zeta) e^{-\boldsymbol{\Xi}\cdot\boldsymbol{\xi}} \mathrm{d}\boldsymbol{\xi} \tag{A.2}$$

$$M_{\alpha,\beta,\gamma} = c_s^{-(\alpha+\beta+\gamma)} \left.\frac{\partial^\alpha \partial^\beta \partial^\gamma}{\partial \Xi^\alpha \partial \Upsilon^\beta \partial Z^\gamma} F(\Xi, \Upsilon, Z)\right|_{\Xi=\Upsilon=Z=0} = \sum_{i,j,k} i^\alpha j^\beta k^\gamma f_{i,j,k} \tag{A.3}$$

$$C_{\alpha,\beta,\gamma} = c_s^{-(\alpha+\beta+\gamma)} \left.\frac{\partial^\alpha \partial^\beta \partial^\gamma}{\partial \Xi^\alpha \partial \Upsilon^\beta \partial Z^\gamma} \ln F(\Xi, \Upsilon, Z)\right|_{\Xi=\Upsilon=Z=0} \tag{A.4}$$

$$C_{200} = \frac{M_{200}}{M_{000}} - \frac{M_{100}^2}{M_{000}^2} \tag{A.5}$$

$$C_{110} = \frac{M_{110}}{M_{000}} - \frac{M_{100} M_{010}}{M_{000}^2} \tag{A.6}$$

$$C_{\alpha,\beta,\gamma}* = (1 - \omega_{\alpha,\beta,\gamma})\, C_{\alpha,\beta,\gamma} \tag{A.7}$$

The density, velocity and the terms of velocity gradient tensor can be identified with cumulants as shown in (A.8) - (A.12). Note, that in this derivation the well-conditioned cumulant is used as described in the appendix of [4] and therefore the zeroth order cumulant is not the density but its deviation from unit density $\delta\rho$. Also the additional moments k_{xy} and k_{xx-yy} are introduced for use in the chapter on refinement.

$$\delta\rho = M_{000} = C_{000} \tag{A.8}$$

$$u = \frac{M_{100}}{M_{000}} = C_{100} \tag{A.9}$$

$$D_x v + D_y u = -3\omega_1 C_{110} = k_{xy} \tag{A.10}$$

$$D_x u = -\frac{\omega_1}{2\rho}\left(2C_{200} - C_{020} - C_{002}\right) - \frac{\omega_2}{2\rho}\left(C_{200} + C_{020} + C_{002} - \delta\rho\right) \tag{A.11}$$

$$D_x u - D_y v = -\frac{3\omega_1}{2}\left(C_{200} - C_{020}\right) = k_{xx-yy} \tag{A.12}$$

A.2. Refinement

As was stated before, LBM is usually used on a uniform grid. However, often a variation in grid spacing is desirable, as coarser grids need less memory and have a coarser timestep, whereas a finer grid is able to resolve finer turbulent structures. To balance these two interests, the domain can be partitioned into blocks with different levels of refinement, as areas of high turbulence, which need high resolution, usually occur only in known parts of the domain. The question is now how to treat the borders of the blocks. In [13] the compact interpolation scheme is proposed that is second order accurate, therefore it is consistent with the order of accuracy of the LBM in general. However, the derivation given was found to be neither exhaustive, nor fully correct, as there seem to be misprints, making it difficult to follow. Therefore a more detailed derivation is given here, that is based on [13] as well as [10], which gave a version for an incompressible form of LBM, but suffers from similar issues. In each coarse timestep, a layer of coarse nodes is interpolated from fine nodes and two layers of fine nodes are interpolated from coarse nodes. The node being interpolated is called the receiver node while the nodes from which is interpolated is referred to as the donor node. Each receiver node has eight donor nodes. A local coordinate system in the center of cube is defined, with the donor nodes laying in the corners at $x = -0.5, 0.5$, $y = -0.5, 0.5$ and $z = -0.5, 0.5$. First, a interpolation function for the density, $\delta\hat{\rho}$ and the velocities $\hat{u}$, $\hat{v}$ and $\hat{w}$ is defined. Note that

the density only requires to be first order accurate.

$$\delta\hat{\rho} = d_0 + d_x x + d_y y + d_z z + d_{xy} xy + d_{xz} xz + d_{yz} yz + d_{xyz} xyz \tag{A.13}$$

$$\hat{u} = a_0 + a_x x + a_y y + a_z z + a_{xx} x^2 + a_{xy} xy + a_{xz} xz + a_{yy} y^2 + a_{yz} yz + a_{zz} z^2 + a_{xyz} xyz \tag{A.14}$$

$$\hat{v} = b_0 + b_x x + b_y y + b_z z + b_{xx} x^2 + b_{xy} xy + b_{xz} xz + b_{yy} y^2 + b_{yz} yz + b_{zz} z^2 + b_{xyz} xyz \tag{A.15}$$

$$\hat{w} = c_0 + c_x x + c_y y + c_z z + c_{xx} x^2 + c_{xy} xy + c_{xz} xz + c_{yy} y^2 + c_{yz} yz + c_{zz} z^2 + c_{xyz} xyz \tag{A.16}$$

To evualate the interpolation functions, constraints have to be defined. The velocity and density in the donor node place 32 constraints, since there are eight of each, thus nine more are required for the 41 degrees of freedom. The second derivative of velocities in the center of the local coordinate system are chosen. They can be computed by taking the first order central difference of the terms of the velocity gradient tensor, which will be denoted as $D_x xu$ and so on. However, the terms in the main diagonal of the velocity gradient tensor are lengthy and include ω_2. This is somewhat undesirable and it was shown in (A.10) that differences of the terms can be expressed more directly in differences of cumulants. Therefore the remaining nine constraints are chosen like shown in (A.17) - (A.20), the missing five constraints by the off diagonal can easily be extrapolated.

$$\frac{\partial^2}{\partial x^2}\hat{v} + \frac{\partial^2}{\partial y \partial x}\hat{u} = D_{xx} v + D_{xy} u \tag{A.17}$$

$$2\frac{\partial^2}{\partial x^2}\hat{u} - \frac{\partial^2}{\partial x \partial y}\hat{v} - \frac{\partial^2}{\partial x \partial z}\hat{w} = 2D_{xx} u - D_{xy} v - D_{xz} w \tag{A.18}$$

$$2\frac{\partial^2}{\partial y^2}\hat{v} - \frac{\partial^2}{\partial x \partial y}\hat{u} - \frac{\partial^2}{\partial y \partial z}\hat{w} = 2D_{yy} v - D_{xy} u - D_{yz} w \tag{A.19}$$

$$2\frac{\partial^2}{\partial z^2}\hat{w} - \frac{\partial^2}{\partial y \partial z}\hat{v} - \frac{\partial^2}{\partial x \partial z}\hat{u} = 2D_{zz} w - D_{yz} v - D_{xz} u \tag{A.20}$$

Thus a system of linear equations is established that can be solved for the coefficients of the interpolation functions. They are listed below, in order to reduce the number length of equations only unique descriptions are given, the other coefficients can easily extrapolated. Furthermore note that

the coefficients of $\delta\hat{\rho}$ are the same as for the other interpolation functions with the exception of d_0.

$$a_0 = \frac{1}{32}\sum_{x,y,z} -4D_{xx}u - 2D_{xy}v - 4D_{yy}u - 2D_{xz}w + -4D_{zz}u + 4u_{xyz} + 4xyv_{xyz} + 4xzw_{xyz} \tag{A.21}$$

$$= \frac{1}{32}\sum_{x,y,z} -x\left(k_{xx-yy} + k_{xx-zz}\right) - 2yk_{xy} - 2zk_{xz} + 4u_{xyz} + 4xyv_{xyz} + 4xzw_{xyz} \tag{A.22}$$

$$a_x = \frac{1}{2}\sum_{x,y,z} xu_{xyz} \tag{A.23}$$

$$a_{xx} = \frac{1}{8}\sum_{x,y,z} x\left(k_{xx-yy} + k_{xx-zz}\right) + 4xyv_{xyz} + 4xzw_{xyz} \tag{A.24}$$

$$a_{yy} = \frac{1}{8}\sum_{x,y,z} yk_{xy} - 4xyv_{xyz} \tag{A.25}$$

$$a_{xy} = 2\sum_{x,y,z} xyu_{xyz} \tag{A.26}$$

$$a_{xyz} = 8\sum_{x,y,z} xyzu_{xyz} \tag{A.27}$$

$$d_0 = \frac{1}{8}\sum_{x,y,z} \delta\rho_{xyz} \tag{A.28}$$

Thus the velocities are computed to second order. To arrive at equations for the second order cumulants, (A.10) - (A.12) are solved for cumulants. A term including the divergence of the velocity is neglected, since LBM is valid in a weakly compressible range and the term is thus much smaller.To compute the cumulants, the derivatves of $\hat{u}$, $\hat{v}$ and $\hat{w}$ are used and the constant terms in the derivatives are replaced with averaged values of second order moments. Also ω_1 has to be scaled with a factor σ to account for the change in refinement, it is 2 when scaling from fine to coarse and $1/2$

when scaling from coarse to fine.

$$A_{011} = \frac{\partial \hat{v}}{\partial z} + \frac{\partial \hat{w}}{\partial y} - b_z - c_y \tag{A.29}$$

$$= b_{xz}x + b_{yz}y + 2b_{zz}z + b_{xyz}xy + c_{xy}x + 2c_{yy}y + c_{yz}z + c_{xyz}xz \tag{A.30}$$

$$A_{101} = \frac{\partial \hat{u}}{\partial z} + \frac{\partial \hat{w}}{\partial x} - a_z - c_x \tag{A.31}$$

$$= a_{xz}x + a_{yz}y + 2a_{zz}z + a_{xyz}xy + 2c_{xx}x + c_{xy}y + c_{xz}z + c_{xyz}yz \tag{A.32}$$

$$A_{110} = \frac{\partial \hat{u}}{\partial y} + \frac{\partial \hat{v}}{\partial x} - a_y - b_x \tag{A.33}$$

$$= a_{xy}x + 2a_{yy}y + a_{yz}z + a_{xyz}xz + 2b_{xx}x + b_{xy}y + b_{xz}z + b_{xyz}yz \tag{A.34}$$

$$B = \frac{\partial \hat{u}}{\partial x} - \frac{\partial \hat{v}}{\partial y} - a_x + b_y \tag{A.35}$$

$$= 2a_{xx}x + a_{xy}y + a_{xz}z + a_{xyz}yz - b_{xy}x - 2b_{yy}y - b_{yz}z - b_{xyz}xz \tag{A.36}$$

$$C = \frac{\partial \hat{u}}{\partial x} - \frac{\partial \hat{w}}{\partial z} - a_x + c_z \tag{A.37}$$

$$= 2a_{xx}x + a_{xy}y + a_{xz}z + a_{xyz}yz - c_{xz}x - c_{yz}y - 2c_{zz}y - c_{xyz}xy \tag{A.38}$$

$$C_{011} = -\frac{\sigma\rho}{3\omega_d}\left(\overline{k_{yz}} + A_{011}\right) \tag{A.39}$$

$$C_{101} = -\frac{\sigma\rho}{3\omega_d}\left(\overline{k_{xz}} + A_{101}\right) \tag{A.40}$$

$$C_{110} = -\frac{\sigma\rho}{3\omega_d}\left(\overline{k_{xy}} + A_{110}\right) \tag{A.41}$$

$$C_{200} = \frac{\delta\rho}{9} - \frac{2\sigma\rho}{9\omega_d}\left(\overline{k_{xx-yy}} + B + \overline{k_{xx-zz}} + C\right) \tag{A.42}$$

$$C_{020} = \frac{\delta\rho}{9} - \frac{2\sigma\rho}{9\omega_d}\left(-2\left(\overline{k_{xx-yy}} + B\right) + \overline{k_{xx-zz}} + C\right) \tag{A.43}$$

$$C_{002} = \frac{\delta\rho}{9} - \frac{2\sigma\rho}{9\omega_d}\left(\overline{k_{xx-yy}} + B - 2\left(\overline{k_{xx-zz}} + C\right)\right) \tag{A.44}$$

Now the cumulants can be transformed back to distributions, with the central moments up to order two and cumulants of order higher than two set to zero.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag der Professur für Strömungsmechanik eingereichte Diplomarbeit zum Thema

Kopplung eines künstlichen neuronalen Netzwerks mit LES-LBM zur Verbesserung einer Windpark-Steuerung

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Dresden, 06. Januar 2020

Henry Korb