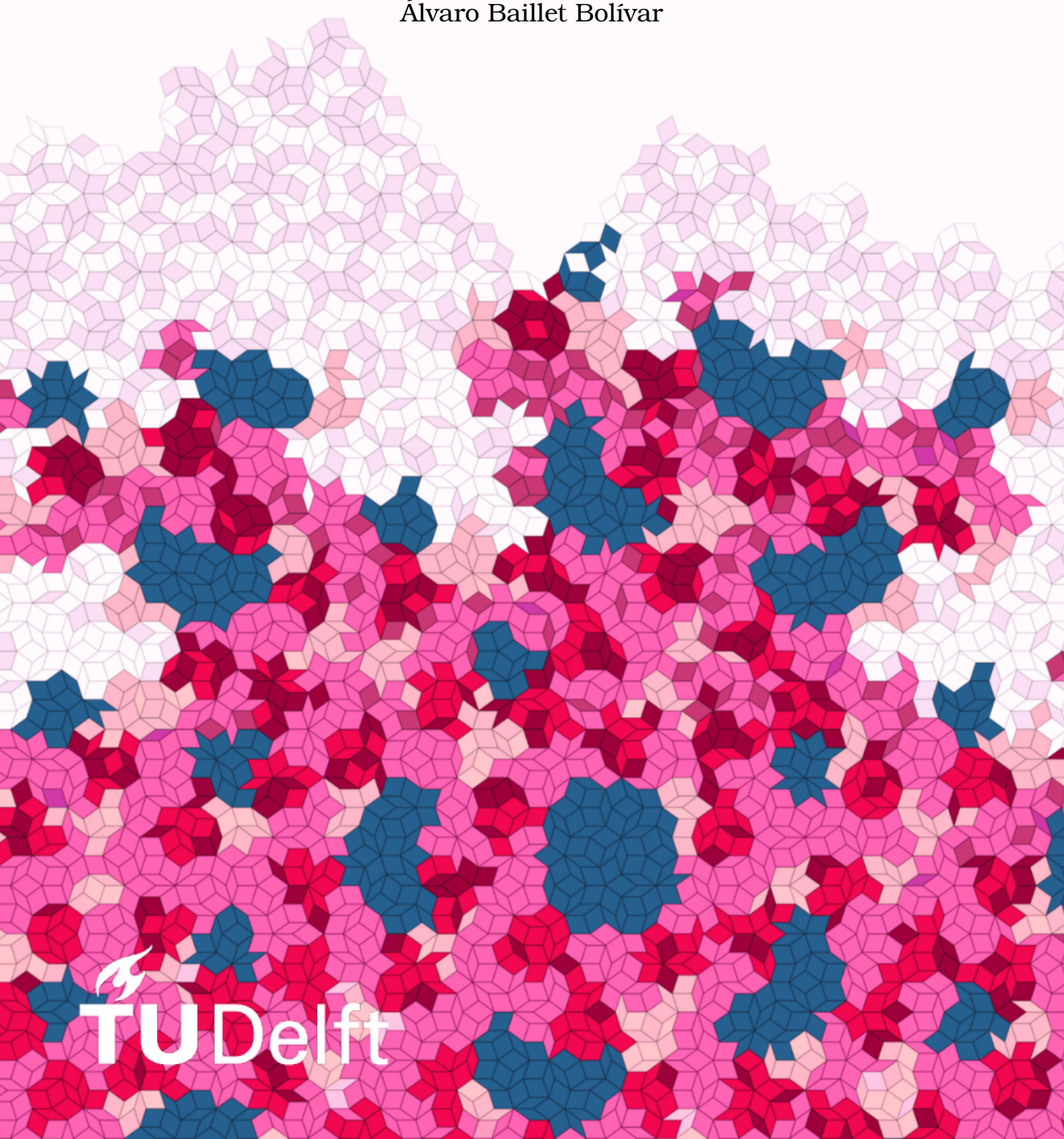
A Discrete Adjoint Lattice Boltzmann Solver for Aerodynamic Shape Optimization

Álvaro Baillet Bolívar



A Discrete Adjoint Lattice Boltzmann Solver for Aerodynamic Shape Optimization

by

Álvaro Baillet Bolívar

In partial fulfilment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology

To be defended on Friday January 30, 2026 at 09:15

Track:	Aerodynamics, Flow Physics and Technology (FPT)
Thesis committee:	Dr. R. P. Dwight
	Dr. F. de Prenter
	Dr. M. Möller
	M. Lăcătuș MSc.
	K. Hoefnagel MSc.

Cover: Penrose tiling by Siha Silva Aguilar.

TU Delft Delft University of Technology

"All this the world well knows; yet none knows well
To shun the heaven that leads men to this hell."
— William Shakespeare, *Sonnet 129*

"He should stop running things. It's a computer—he'll get a different result each time."
— My dad to my girlfriend

Abstract

A Lattice Boltzmann (LBM) solver is presented and differentiated with the discrete adjoint method. A shape optimization pipeline is implemented based on the obtained gradients. The following features are considered, aiming for applicability in industrially relevant simulation and optimization, and possible integration with a quantum LBM code in the future:

1. Recursive regularized collision operator.
2. Volumetric grid refinement with arbitrary grid geometries.
3. A mass conserving, differentiable boundary condition for no-slip boundaries and a methodology to deduce physical shape gradients from the bounce back boundary condition (for integration with a quantum code).
4. Momentum inlet and pressure outlet boundary conditions.
5. Pressure relaxation absorption region (sponge) to dampen acoustic fluctuations.
6. Coefficients of lift and drag as quantities of interest, computed with the momentum exchange method.
7. Shape parametrization with free form deformation.

The forward solver is validated on steady and unsteady flow conditions, as are the adjoint-based gradients, these obtained with a general methodology by Tekitek et al. [1]. The adjoint-based gradients present relative errors to finite differences of $\mathcal{O}(10^{-5})$. A steady L/D optimization case is run, presenting improvements on the base shape in the order of 20%. It is demonstrated that the adjoint methodology supports parametrization of both the collision operator and streaming matrix in a simple, extensible manner.

Contents

1. Introduction 2
- 1.1. Review of adjoint LBM and LBM-based optimization literature . . . 5
- 1.2. Gaps in the literature and research questions . . . 10

2. The Forward Lattice Boltzmann Method 14
- 2.1. The Lattice Boltzmann Method . . . 14
- 2.2. The collision operator . . . 16
- 2.3. Boundary conditions . . . 20
  - 2.3.1. Solid walls . . . 20
  - 2.3.2. Domain boundary conditions . . . 24
  - 2.3.3. Sponge . . . 25
- 2.4. Grid refinement . . . 25
  - 2.4.1. Multiple refinement levels . . . 31
- 2.5. Cost function definition and computation . . . 31

3. The Discrete Adjoint Lattice Boltzmann Method Applied to Shape Optimization 33
- 3.1. Adjoint derivation . . . 33
  - 3.1.1. Developed flow state gradients . . . 37
  - 3.1.2. Steady Gradients . . . 37
- 3.2. Adjoint boundary conditions, source terms, and miscellaneous derivatives . 38
  - 3.2.1. Source terms . . . 39
  - 3.2.2. Partial derivatives to the optimization parameters . . . 40
  - 3.2.3. Adjoint sponge . . . 40
- 3.3. Adjoint grid refinement . . . 41
- 3.4. Checkpointing . . . 41
- 3.5. Shape parametrization with Free-Form Deformation . . . 43
  - 3.5.1. Implementation . . . 44
  - 3.5.2. Derivative of FFD . . . 45
- 3.6. Main application cases . . . 46
  - 3.6.1. Parameter selection . . . 48

4. Results 49
- 4.1. Forward solver validation . . . 49
  - 4.1.1. Consistency check for PBB and verification of RR . . . 49
  - 4.1.2. Validation of the steady case . . . 52
  - 4.1.3. Validation of the unsteady case and grid refinement scheme . . . 55
  - 4.1.4. Convergence studies and sponge analysis . . . 56
  - 4.1.5. Verification and validation of the grid refinement scheme . . . 60
- 4.2. Adjoint solver validation . . . 62
  - 4.2.1. Validation of adjoint-based gradients for the steady case . . . 62
  - 4.2.2. Shape optimization results for the steady case . . . 63
  - 4.2.3. Validation of adjoint-based gradients for the unsteady case . . . 66
- 4.3. Preliminary extension to higher order boundary conditions . . . 72

5. Conclusions and Recommendations 75

A. Algorithm for time-stepping on multiple refinement levels 83

B. Recursive regularized collision operator with porosity and pressure relaxation 85

C. Comparison between Tekitek et al.'s and Vergnault and Sagaut's adjoint formulations 88

D. Example of source terms arising due to non-local boundary conditions 92

E. Can a gradient based optimization algorithm successfully run with the halfway bounce back boundary condition? 94
E.1. Methodology 94
E.2. Results 95
E.2.1. Mesh convergence study 96
E.2.2. Gradient verification 96
E.2.3. L/D optimization 96
E.3. Conclusion 98

F. Alternative example of the effect of a homogenous outlet adjoint boundary condition on gradient accuracy 99

List of symbols and abbreviations

Abbreviation	Definition
2D	2-dimensional
3D	3-dimensional
BGK	Bhatnagar-Gross-Krook
CAD	Computer aided design
$coal$	Coalescence
CFD	Computational fluid dynamics
cpc, cpd	Cells per chord, per diameter
DNS	Direct numerical simulation
DVBE	Discrete velocity Boltzmann equation
FD	Finite difference
FFD	Free form deformation
FWH	Ffowcs-Williams-Hawkings
LES	Large eddy simulation
LBM	Lattice Boltzmann method
MDO	Multidisciplanry optimization
MRT	Multiple relaxation time
NACA	National Advisory Committee for Aeronautics
PDE	Partial differential equation
PBB	Porous bounce back
QoI	Quantity of interest
RAM	Random access memory
VLES	Very large eddy simulation

Symbol	Definition
a, b	Free parameters for PBB
$A_{pq}, A^{eq}_{pq}, A^{neq}_{pq}$	Hermite moments of the distribution functions
$\mathcal{B}$	Non-linear boundary condition operator
$\mathcal{B}^b_a$	Bernstein polynomial of degree b
c	Chord length
C_{RAM}	Ram capacity
C_L, C_D	Lift and drag coefficients
$C_{L_{max}}, C_{D_{max}}$	Maximum lift and drag coefficients
C_p	Pressure coefficient
c_s	Speed of sound
d	Minimum distance to wall
D	Drag
$\overline{D}$	Time averaged drag
e	Error
$\mathcal{F}$	Local vector of aerodynamic forces
f_i	Particle distribution function (otherwise referred to as population)
f^{eq}_i, f^{neq}_i	Equilibrium and non-equilibrium distribution functions
f^*_i	Adjoint distribution function
$\hat{f}_i$	Post-collision distribution function
$\hat{\mathbf{f}}$	Vector of post-collision distribution functions in a given node
$\mathbf{F}^n$	n-th flattened state vector

Symbol	Definition
$\mathbf{F}^{*,n}$	n-th flattened vector of the adjoint distribution functions
H	Channel height
h	Finite difference spacing
I	Cost function
$\mathcal{I}$	Cost functional
J	Lagrangian function
L	Lift
L	FFD rectangle dimension vector
L/D	Lift to drag ratio
Ma	Mach number
M_{pq}, M_{pq}^{eq}	$p+q$-th (equilibrium) moment of the distribution functions
mu_x	x-momentum
n_{check}	Number of checkpoints
n_{start}	Starting iteration
N	Final discrete timestep or total number of timesteps
N_{levs}	Number of embedded grids
N_x, N_y	Cells in cartesian directions
$\mathbf{n}$	Normal vector
p	Pressure
q	Normalized distance to wall
R	Transformation matrix from Hermite to raw moment space
Re	Reynolds number
$\mathbf{R}$	Residual vector
S	Streaming matrix
t	Time
t_c	Convective time
t_n	n-th timestep
T	Transformation matrix from raw moment space to post-collision distributions
u_α	Velocity component (Einstein notation)
u_{ex}, v_{ex}	Exact x and y velocity components
u_{sp}	Spline parameter
$\mathbf{u}$	Velocity vector
v	Adjoint variable for the Poisson equation
w_i	Lattice quadrature weights
X_b	Set of boundary nodes
X_f	Set of fluid nodes
$\mathcal{X}$	Transformed coordinate after FFD
$\boldsymbol{x}$	Spatial location vector (vector notation)
$\mathbf{x}$	Coordinate vector
$\mathbf{x}_b$	Node adjacent to solid wall
$\mathbf{x}_k$	k-th mesh node
$\tilde{\mathbf{x}}_{k_j}$	$\mathbf{x}_k + \boldsymbol{\xi}_j$
$\mathscr{C}$	Collision operator
$\mathcal{C}$	Relaxation operator in the Hermite moment space

Greek symbol	Definition
β	Brinkmann porosity parameter
$\delta_{\alpha\beta}$	Kronecker delta
$\Delta x, \Delta t$	Discrete intervals in space and time
Δ_{check}	Checkpointing interval
ε	Porosity

Greek symbol	Definition
$\boldsymbol{\mu}$	Vector of optimization parameters
μ	Dynamic viscosity
ν	Kinematic viscosity
Ω_i	Generic collision operator
$\Pi_{\alpha\beta}$	Viscous stress tensor
Φ	Composition of streaming and collision, LBM step operator
ρ	Density
$\bar{\rho}$	Time averaged density
ρ'	Density fluctuation field
ρ_a	Acoustic source amplitude
ρ_s	Relaxed density, or density on a sponge node
ρ_∞, u_∞	Freestream density, velocity
σ	Sponge strength parameter
σ_ε	Free parameter for PBB
τ	Relaxation time
τ_f, τ_c	Relaxation times on fine and parent coarse grid
ω	$1/\tau$
$\boldsymbol{\xi}_i$	Lattice velocity
ξ	Continuous velocity
ξ_x, ξ_y	x and y components of the discrete lattice velocity

Chapter 1: Introduction

Two of the primary goals of aerodynamic research are to explain how an object's geometry affects the motion of air around it and to propose design improvements to ensure performance and efficiency. Attempting the first objective using an approximate model of reality, be it mathematical or experimental, is called simulation. The second objective is called optimization.

Computational methods are widely used for both of these practices. Computational simulation, more commonly known as computational fluid dynamics (CFD), consists of obtaining the density, momentum, and energy fields around an object (otherwise referred to as an obstacle) by numerically approximating the governing Navier-Stokes equations [2].

Conventionally, computational optimization reduces the complex fluid motions output by the simulation code to a single number—known as a quantity of interest (QoI)—and studies how changes to the simulation's parameters (i.e. the input variables that define the obstacle's geometry) affect it. Based on this sensitivity, changes are proposed to either increase or decrease its value (i.e. maximization or minimization) [3]. QoIs may be a measure of the output of a design—for example, the downforce (lift) produced by a Formula 1 car—a measure of the cost, such as the resistive drag force, or a quantification of efficiency, such as the lift-to-drag ratio.

To ensure the level of fine geometric control required to accurately optimize an aerodynamic design, the optimization algorithm must typically handle a large number of parameters [4], [5]. For such high-dimensional situations, gradient-based approaches are the most computationally efficient methods. Non-gradient based techniques, such as evolutionary algorithms, may require a prohibitive amount of evaluations of the costly CFD code (often referred to as the 'forward code' or 'forward problem') to assess and propose new designs [5].

Nevertheless, efficiently obtaining the QoI gradient (the aforementioned sensitivity) is in itself a challenge. The simplest approach, numerical approximation using finite differences, requires (at least) as many evaluations of the forward problem as there are inputs for a single gradient computation. A potentially better practice is to obtain exact gradients of the simulation code using the standard rules of analytical differentiation. In the process of doing so, one can define define a procedure called the adjoint problem[1], that yields auxiliary quantities called adjoint variables, to ensure that the QoI's gradient can be obtained at a cost independent of the number of optimization parameters [3]. In the context of sensitivity evaluation of a scalar function, the adjoint method yields an exactly equivalent procedure to backpropagation or reverse-mode automatic differentiation.

So far, at a very high level, we have defined what simulation and optimization are in an aerodynamic context, how they are coupled together in a computational setting, the challenge of obtaining gradients efficiently, and the concept of an adjoint as a solution to this problem. Given that the adjoint is the main object of study of this work (as can be deduced from the title), it is worth introducing some relevant terminology. The adjoint problem that arises from analytical differentiation of a discrete method is called the discrete adjoint problem. An alternative technique is to derive adjoint equations at a continuous level and then discretize them to yield an efficient, computable estimate of the QoI gradient. Naturally, this is called a continuous adjoint [3]. Each approach has its relative strengths, which will be assessed in the specific context of this thesis in the forthcoming literature review.

Using either of the adjoint approaches, coupled with a conventional Navier-Stokes or

[1] Like eigenvalue problems (for example), adjoints are a class of problems in themselves that can arise in many applications, e.g. error estimation [6].

Euler CFD code, aerodynamic gradient-based shape optimization is a mature and very successful field. Three classical examples are the optimization of complete aircraft configurations using the adjoint Euler equations by Reuther [7] and Elliott [8], and the design of a wing with an adjoint Navier-Stokes code [9]. More recent examples include the optimization of wind turbines blades for maximum torque [10], and airfoil optimization — including laminar to turbulent transition — for minimum drag [11].

Context for the present thesis This work is an addition to the adjoint based aerodynamic shape optimization literature. However, its main twist is that it does not use a Navier-Stokes or Euler solver for the fluid flow simulation, but an alternative approach called the Lattice Boltzmann Method (LBM). Rather than directly solving for the macroscopic fields (density, momentum, energy), the LBM evolves a mesoscopic fluid model that deals with a statistical representation of the microscopic movement of individual fluid molecules, called a distribution function [12]. The method consists of a two-step scheme where the dissipative non-linear character of viscous flows is modelled by a local operation called collision, followed by a linear advective step called streaming. The macroscopic variables are obtained in post-processing, and it can be shown that the recovered fields agree with the compressible Navier-Stokes model at low Mach numbers [12].

One of the active research areas related to the LBM concerns the development of algorithms to implement it on quantum computers. Among CFD methods, the LBM is particularly well-suited for this purpose due to its explicit time integration, high degree of parallelization, and local update rules. The streaming operation is unitary and therefore naturally compatible with quantum computation, while the locality of the collision operator reduces the difficulty of approximating the non-unitary, dissipative aspects of fluid dynamics. In contrast, implementing a conventional incompressible Navier-Stokes method in a quantum computer would be very challenging and inefficient given the long-range dependencies introduced by the elliptic pressure Poisson equation.

The quantum LBM research group based at TU Delft, in partnership with Fujitsu, has the long-term goal of aiding industrial design cycles through computational optimization centred around a quantum LBM simulation framework coupled with an adjoint-based optimizer. The promise of the quantum LBM code is polylogarithmic memory scaling that would make high-fidelity simulations far more affordable than current Navier-Stokes, Euler, or classical LBM implementations [13].

Top-level objective In this context, the quantum group proposed this thesis with the following objective:

> To develop a general, extensible, and validated adjoint LBM framework capable of treating flows over a broad range of Reynolds numbers, and suitable for industrially relevant shape optimization problems.

Given the very early development stage of the quantum LBM technology, the focus is not on developing a new adjoint architecture with a native quantum interface (which would likely be most computationally efficient), but on identifying the most general adjoint methodologies and coupling them with an optimizer that is itself also suitable for industrial cases.

Context, re-examined Regardless of the promise of quantum LBM technology, the classical scheme offers several competitive advantages that merit its consideration as an alternative fluid solver for shape optimization. It conserves mass and momentum by construction, it is simple to implement for complex geometries, and is well suited for parallel algorithms for high-performance computing [14]. Furthermore, it naturally propagates

acoustic waves and generally introduces very little numerical dissipation [15], [16]. Comparative studies against Navier-Stokes codes by Maerie et al. [15] and Suss et al. [17] have both found that, for under-resolved, unsteady, turbulent flows (i.e. LES or VLES) the LBM is as fast or faster than the Navier-Stokes codes for a given error tolerance — for aeroacoustic flows, the LBM is much superior. Thus, for these conditions, the LBM appears as a competitive candidate.

Nevertheless, the LBM still faces some developmental needs. Primarily, its accuracy in wall bounded flows should be improved without sacrificing its conservation properties. As with the quantum LBM, there is a TU Delft group currently spearheading this effort [18].

A priori requirements Returning to the top-level objective, a brief review of some LBM applications to high-reynolds number cases [14], [19]–[23], yields the following common features of classical LBM codes that should be considered in the adjoint and optimization pipeline:

A priori requirements

1. Collision operators that ensure stability at high Reynolds number.
2. Local grid refinement areas to affordably increase resolution near walls.
3. High-order no-slip boundary conditions on the obstacle walls (the obstacle is the embedded object of interest).
4. Velocity (or momentum) boundary conditions at the inlet and density or Neumann boundary conditions at the outlet for well-posedness.
5. Absorption regions in the far-field to dampen acoustic waves reflected from outflow (or inflow) boundaries.
6. Steady and unsteady flow conditions.
7. Turbulence modelling based on the eddy viscosity assumption.
8. Coefficients of lift and drag and the lift to drag ratio as possible quantities of interest.
9. 3D geometries.
10. Concerning the optimization, a shape parametrization suitable for analytical differentiation and readily extended to more complex cases.

This list incorporates some considerations that would be unnecessary for a quantum solver, specifically the sophisticated collision operator (given that instabilities arise due to high grid Reynolds numbers [24]) and the local grid refinement, given the polylogarithmic scaling. Other considerations are currently not possible, such as the absorption region or the inlet/outlet boundary conditions. Nevertheless, all of these remain in the work to enable it to be used for classical optimization while it lies in wait for potential quantum applications.

For now, only the third requirement is modified to incorporate a quantum consideration. The simple halfway bounce back boundary condition enforces no-slip boundaries with generally first-order accuracy [12], but can be implemented as a mass-conserving, unitary operation, which are key considerations for quantum computing. However, this boundary condition is not suitable for gradient-based optimization (cf. section 2.3), so the quantum group proposed implementing a mass-conserving but not necessarily unitary boundary condition on the obstacle. The order of accuracy was also not specified. Aside from this single quantum oriented constraint, the remainder of this work solely addresses classical adjoint LBM problems.

The imposition of this top-level objective and associated list of considerations (read requirements) makes the thesis more akin to an engineering design project than a research one. If all the necessary methodologies existed in the adjoint LBM literature, the top-level objective could be achieved by reviewing, compiling, and reproducing existing methods

and results. More pragmatically, this thesis would exist if a satisfactory open-source adjoint LBM solver did not. However, the field of adjoint lattice Boltzmann methods and LBM-based shape optimization is still developing, and a number of research areas aligned with the requirements can be identified. The following literature review does this and proposes research questions (that incorporate the requirements) from which to develop the thesis.

1.1. Review of adjoint LBM and LBM-based optimization literature

The following review presentes a chronological overview of the adjoint LBM literature. For a clearer illustration of the contributions made by each paper, a descriptive header is employed. After discussing the most significant papers, the gaps in the literature are identified and research questions are proposed.

Initial development of the adjoint LBM The adjoint LBM field was initiated by Tekitek et al. in 2006 [1]. They developed an unsteady discrete adjoint based on a global formulation of the LBM. In this formulation, collision, streaming, and boundary-condition enforcement are represented as operators acting on a vector that concatenates the LBM state variables over all grid points. This differs from the typical presentation of the forward LBM, where the scheme is given for a node in the bulk and explicit exceptions (boundary conditions) are also detailed for the specific nodes to which they apply. However, with this operator-based abstraction, the adjoint derivation is simple, especially if the forward boundary conditions can be expressed as a linear operation.

Tekitek et al. applied their adjoint to optimize the parameters of a multiple relaxation time (MRT) collision model with the objective of minimising the error to known analytical solutions. The application cases consisted of both steady and unsteady laminar flows.

The main strengths of this work are that the adjoint solver remains a two-step method[2] with the locality and parallelisation properties of the forward code, yielding a conceptually simple and efficient scheme, and that the adjoint boundary conditions are derived in a general and simple-to-implement manner. Nevertheless, recent benchmarking by several groups [24]–[27] has shown that the MRT model offers reduced stability and accuracy compared to state-of-the-art collision models, limiting its applicability at higher Reynolds numbers. Furthermore, Tekitek et al. do not address the well-known drawback of using unsteady adjoint formulations, which is their high storage costs [3]. Other limitations of this work are that the adjoint is restricted to uniform grids, without an absorption region, and with periodic boundary conditions.

An alternative discrete adjoint method applied to topology optimization After Tekitek et al., several other papers (primarily by Georg Pingen and co-authors) were written based on a marginally different discrete adjoint [28]–[32]. This approach is limited to steady problems and consists of differentiating the fixed point iteration describing a converged LBM simulation (arrived at by unsteady time-stepping). As in Tekitek et al., the collision and streaming steps are considered as operators on a complete state vector, leading to a straightforward implementation of adjoint boundary conditions. However, the linearization of the steady-state problem leads to a large matrix that must be inverted, making this approach very computationally expensive for large problems. This deficiency was noted by the authors themselves and was partially addressed with advanced linear algebra techniques [29]. In contrast, Tekitek et al.'s approach can be efficiently applied to steady problems by running the unsteady adjoint to convergence.

[2] For which it is possible to identify adjoint collision and adjoint streaming steps.

In all of these papers, the methodology was applied to steady, laminar topology optimization problems, which are related to, but fundamentally different from, shape optimization problems. Topology optimization determines the best material layout within a design space (including creating or removing holes), while shape optimization fine-tunes the boundaries of an existing geometry without changing its connectivity. Although formally a generalization of shape optimization, topology optimization is better suited for proposing coarse, baseline designs, given its typically low geometric resolution [31]. For the industrial applications of the TU Delft/Fujitsu group, where small improvements on an existing baseline design are sought, shape optimization is the more appropriate approach.

Because it will play a part in my proposed approach for shape optimization, it is worth discussing how these papers tackle the topology optimization. [28]–[30] seek to determine whether a particular node on the grid should be treated as fluid or solid in order to minimize a particular QoI (for example, the pressure drop across an inlet and outlet). To do so, they describe each node with a certain porosity value between zero and one (zero is a fluid node, one a solid node). The value of the porosity is controlled by the optimizer. As implemented in these papers, where the collision step is modified, the name of the method is 'Brinkmann penalization'. The approach is simple to implement but has been shown to result in incorrect pressure fields for unsteady problems [33] as it allows for pressure variations *within* the obstacle (or solid region, in the case of topology optimization). Furthermore, its geometric resolution is very poor as it results in a staircase representation of the geometry (similarly to the bounce back rule). However, it has the important mass-conservation property sought by the quantum group. Thus, it is worth investigating how it can be adapted for shape optimization.

An alternative topology parametrization approach was proposed by Kreissl et al. [31], claiming to be capable of generalized shape optimization (i.e. it can support both fine changes to the geometry and emergence of new topological features). They consider an explicit level set method[3] where the optimization parameters are the interpolation weights of the radial basis functions that define the level set function. To overcome the geometric modelling deficiencies of the penalization approach, no-slip boundary conditions are implemented via a higher order interpolated bounce back boundary condition (by Bouzidi et al.) that incorporates the smooth geometry of the radial basis functions. This is easily mapped to the adjoint thanks to the global derivation of the adjoint LBM, similar to Tekitek et al.

Despite offering a potential parametrization technique for shape optimization, the approach has several limitations. Primarily, it is complex to implement. Radial basis functions must be defined on an alternate grid with a spacing that has a significant effect on the results of the optimization; a supplementary algorithm must be provided to identify the zero-contour of the level set function, explicit smoothing is required for stability, and within all of these considerations non-trivial, case-specific decisions must be made on their internal parameters (for example, the smoothing radius of the smoothing algorithm).

These papers considered either the BGK or MRT collision operators, both of which have limited stability at higher Reynolds numbers, and do not consider grid refinement, turbulence, or an absorption region. They do contribute a treatment of velocity and pressure inlets and outlets, the former with equilibrium boundary conditions, the latter with non-equilibrium boundary conditions. It is not explicitly detailed how they are mapped to the adjoint though it is implied that both only modify the collision operation and thus are incorporated into the adjoint collision step.

[3] In the level set method the geometry of the obstacle or fluid domain is described by the zero-contour of an analytical function called the level set function.

A continuous adjoint The proposed discrete adjoint methodologies by Tekitek et al. and Pingen et al. were followed by a continuous formulation by Krause et al. [34], [35] based on the continuous Lattice Boltzmann BGK equation. Their primary motivation for doing so was to offer an adjoint with the same parallelization properties as the forward code, which had already been done by Tekitek et al. in a discrete setting but not Pingen et al. The discretization of the continuous adjoint does yield an easily parallel scheme, but poses significant challenges in the derivation of adjoint boundary conditions. Nevertheless, the formulation has been quite widely adopted, primarily for topology optimization problems (e.g., [36], [37]), sometimes coupled with thermal flows [38], [39]. However, in a recent study, Luo et al. [40] have shown that boundary terms obtained from the continuous formulation are significantly less stable than those obtained from the discrete form, even at low Reynolds numbers.

Krause et al.'s approach does not offer any (new) developments regarding the list of considerations I proposed. They applied their adjoint to a very simple flow control problem where a body force is controlled to achieve a desired (steady) flow state. The reduced stability (and gradient accuracy) of the continuous adjoint, the difficulty of obtaining adjoint boundary conditions, and the prior existence of a simple and local adjoint implementation, makes the discrete adjoint far more attractive.

An alternative derivation of Tekitek et al.'s adjoint The discrete adjoint LBM was revisited by Vergnault and Sagaut in 2014 [41]. Their application case is an acoustic, unsteady control problem, with periodic and bounce back boundary conditions, on a uniform grid, with the BGK collision operator. They offer a strategy for reducing the storage costs associated with the unsteady adjoint. This consists of only storing the underlying mean flow and then running the unsteady adjoint to convergence (i.e. using a steady adjoint). The accuracy of the gradients is reduced but, for their test cases, remains sufficient for a successful optimization. They also propose a novel way of estimating the Hessian for the optimization algorithms using complex differentiation. However, their most significant contribution is a novel derivation of the discrete adjoint LBM.

This formulation underpins the few papers on adjoint based shape optimization in the LBM field, so it merits further discussion. Firstly, we must note that it yields an algorithm exactly equivalent to Tekitek et al.'s. Therefore, it is not a variation on the discrete adjoint LBM, aside from the fact that it uses the BGK operator whereas Tekitek et al. use the MRT model. However, the nature of the derivation is quite different, leading to several limitations (and some advantages) which should be examined.

To do so, given that we have not fully discussed the LBM yet, I propose the following example. Suppose the 1D Poisson equation is to be solved with a Neumann boundary condition on the left end of the domain $x = 0$ and a Dirichlet boundary condition on the right $x = 1$:

$$\begin{cases} -u''(x) = f \\ u'(0) = 0 \\ u(1) = 0. \end{cases} \tag{1.1.1}$$

Using central differences on a uniformly spaced grid with 5 unknowns yields the system

$$A\mathbf{u} = \mathbf{f}, \tag{1.1.2}$$

where A incorporates the boundary conditions and takes the form:

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.1.3}$$

$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 \end{bmatrix}^\intercal$ is the vector of unknowns and $\mathbf{f}$ is left unspecified as it is not needed for this explanation. It can be shown that the discrete adjoint system is of the form

$$A^\intercal \mathbf{v} = \mathbf{g}, \tag{1.1.4}$$

with $\mathbf{v}$ the adjoint variable and $\mathbf{g}$ an appropriate source term [3]. The adjoint operator $A^\intercal$ directly incorporates the adjoint boundary conditions by simple transposition. Going from (1.1.2) to (1.1.4) by (linearizing) and transposing the forward operator is precisely what Tekitek et al. do for the LBM.

On the other hand, Vergnault and Sagaut consider the local form of the LBM. In essence, they consider a general node in the bulk, which in our example would follow the equation:

$$-u_{i-1} + 2u_i - u_{i+1} = f_i \tag{1.1.5}$$

where $i = 3$, for example. Working exclusively in this local frame, they would derive that the adjoint equation for a node in the bulk is

$$-v_{i-1} + 2v_i - v_{i+1} = g_i, \tag{1.1.6}$$

which can be verified by writing out the third row of $A^\intercal$.

The primary advantage of this approach is that it allows the reader to easily grasp and devise implementation strategies for the adjoint LBM, especially since the forward LBM is usually presented as a local, explicit time stepping method.

However, there are two very significant disadvantages. The first is Vergnault and Sagaut do not derive the non-trivial adjoint terminal condition[4], presenting an ultimately incomplete methodology. The second has to do with the derivation of adjoint boundary conditions. Writing out the adjoint operator $A^\intercal$ yields:

$$A^\intercal = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}. \tag{1.1.7}$$

This leads to the following observations:

1. There are three bulk nodes, with indices 2, 3, and 4, but (1.1.6) only holds for v_3.
2. The other bulk nodes, which neighbour boundaries, are altered due to the transposition of the boundary conditions.
3. The adjoint boundary conditions are not obvious from the form of the forward operator.

Thus, accurately obtaining a complete adjoint operator using Vergnault and Sagaut's local derivation is much harder. One has to first derive the adjoint boundary condition by considering the local rule

$$-u_1 + u_2 = f_1 \tag{1.1.8}$$

and then identify and quantify its influence on neighbouring nodes, which can be a very thorny process. Nevertheless, his derivation has been adopted for shape optimization with varying success, as evidenced by the following papers.

[4]The terminal condition does not arise in the Poisson example but is an important consideration for any unsteady adjoint solver.

Adjoint-based shape optimization Cheylan et al.'s 2019 paper [42] is particularly significant for this work as it shares the top level objective, namely to use the adjoint LBM for optimization at a wide range of Reynolds numbers and for industrial applications. Additionally, it is the first paper to do pure shape optimization (i.e., with no topological variations) for the standard LBM.[5] They address my proposed a priori requirements in the following manner:

1. The adjoint of the double relaxation time (DRT) operator is proposed to overcome the stability deficiencies of BGK at high Reynolds numbers. However, in a recent effort to systematically benchmark the performance of collision operators, particularly by Astoul [25], this operator has been shown to have much poorer stability than more state-of-the-art models such as the Recursive Regularization (RR) operator.
2. The adjoint of an interpolation based grid refinement scheme is implemented. This is a valuable contribution, especially since it shows how to incorporate grid refinement schemes into the adjoint LBM. However, the interpolation scheme has also recently shown comparatively poor performance in systematic benchmarking tests [27], and has disadvantages concerning ease of implementation, particularly for non-rectangular refinements [46]. These would be ideal for an industrial case where the refinement geometry wraps around the obstacle to reduce cost.
3. A high-order interpolated bounce back scheme was implemented, but not correctly mapped to the adjoint. The error occurred because the influence on neighbouring nodes in the adjoint was not considered [47], as a consequence of Vergnault and Sagaut's locally-centred theory. This results in large errors on the computed gradients, with a maximum relative error of approximately 30% compared to finite differences. Furthermore, the scheme does not conserve mass.
4. Joint pressure and velocity Dirichlet boundary conditions were imposed on all edges of the domain, overdetermining the problem. The main motivation for this was to simplify the treatment of adjoint boundary conditions, although more physically consistent approaches have been successfully implemented using Tekitek et al.'s theory. For external aerodynamics this may be acceptable as the boundaries are far from the domain. However, for generality—and to reduce the domain's size—more consistent approaches should be sought.
5. No absorption region was considered.
6. Only steady adjoints are used although unsteady problems are considered. To avoid using an unsteady adjoint the fields are time averaged and then used in the steady adjoint (which marches in time until it converges). This approach may not be valid, or even feasible, if large oscillations occur in the flow field, as the adjoint is a linearization over a base state and accounts only for small variations near this.
7. The Smagorinsky model is used for turbulence modelling and mapped to the adjoint under the frozen turbulence assumption.
8. The dimensional drag force is used as a cost function, computed using a far field method (i.e. with a surface integral on a contour far from the obstacle). There are several limitations here. Firstly, a non-dimensional quantity should be used as an aerodynamic cost function to ensure minimization does not occur due to a smaller scale. Secondly, in their implementation of the far-field formulation, several viscous terms are removed. This simplification may not always be physically appropriate (especially if the drag is not pressure-dominated, i.e. in the absence of separated flow) possibly leading to a poorer optimization.
9. The implementation is 3D.

[5] Curiously, several papers have been written on aerodynamic shape optimization using a Lattice Boltzmann method on curvilinear grids or with a finite volume discretization. See for examples papers by Li et al. [43] and, more recently, Jalali Khouzani [44], [45]. These constitute an independent field, however.

10. The shape is parametrized by its coordinates with an explicit smoothing algorithm. Though simple and general, this greatly increases the dimensionality of the problem, making the optimization more difficult. Furthermore, obtaining analytical sensitivities to the geometry can be complicated given the non-smooth geometrical representation and the fact that many LBM boundary schemes incorporate the obstacle geometry through the distance from the boundary node to the boundary.

To date, Cheylan et al.'s paper is the most ambitious LBM-based aerodynamic shape optimization text. The field is only significantly complemented by Kazuya Kusano's contributions. These consist of a paper establishing the (discrete) adjoint methodology [47] and several applications to aeroacoustic optimization problems (e.g., [48], [49]). With regards to the list of requirements, he proposes:

1. The BGK collision operator.
2. The same interpolated grid refinement scheme as Cheylan et al.
3. A correct adjoint implementation of an interpolated bounce back rule, obtained by carefully and very diligently considering the influence on neighbouring nodes. The adjoint is successfully validated showing relative errors of $\mathcal{O}(10^{-4})$.
4. The same overdetermining boundary conditions as Cheylan et al.
5. A viscosity based absorption region that is successfully mapped to the adjoint. In this region, which is far from the obstacle, the viscosity is progressively increased to completely attenuate any acoustic waves that would (non-physically) reflect from the outflow boundary. This approach is valid but will not be used for reasons detailed in section 1.2.
6. A naive unsteady adjoint implementation that requires terabytes of storage for moderate Reynolds numbers of $\mathcal{O}(100)$.
7. No turbulence.
8. Aeroacoustic cost functions based on direct noise computation. This may only be feasible for moderate Reynolds numbers. A more general approach would be to use an acoustic analogy for the sound intensity computation.
9. 2D implementation, though the proposed methods are easily extensible to 3D.
10. A more sophisticated parametrization than Cheylan et al. using an Akima spline for the parametrization; thus reducing the dimensionality and allowing for analytical sensitivities (although he computes the required gradients with finite differences).

1.2. Gaps in the literature and research questions

Based on the literature review above, the following gaps are identified and potential solutions are given when they exist in literature:

1. Several of the necessary (or best available) features for industrially relevant simulations do not have an adjoint formulation. Recently, Coreixas and Latt [14] presented a robust 2D implementation of a forward LBM solver validated for a wide range of Reynolds numbers (up to $Re = 10^7$). They use the recursive regularized (RR) collision operator, volumetric grid refinement, momentum inlet and zero-gradient outlet boundary conditions, and a pressure-relaxation sponge (see subsection 2.3.3 and the following entry in this list). Given their excellent results on both subsonic and supersonic flows[6], this architecture will be used for the forward solver.
2. In the context of shape optimization, a well-posed system of inlet/outlet boundary conditions has not been considered. Although several papers, primarily relating

[6]The collision operator must change for supersonic flows but the grid refinement can be easily adapted.

to topology optimization, have successfully implemented a consistent adjoint inlet/outlet boundary treatment (e.g., [28], [50]), accurate and robust forward boundary conditions for external aerodynamic simulations have not been considered. One such set is the one proposed in [14] with a momentum inlet, extrapolation outlet, and pressure relaxation sponge. The pressure relaxation sponge drives the pressure field to a freestream value over a region near the outlet. This dampens acoustic waves *and* effectively imposes a Dirichlet pressure outlet, which is required for well-posedness especially since the combination of a momentum inlet and extrapolation outlet leads to an underdetermined problem.

3. Despite the existence of a general methodology for the treatment of adjoint boundary conditions, it has not been implemented for shape optimization. For robustness, this should be achieved.

4. The lift force has not been used as a cost-function in an adjoint LBM pipeline. Furthermore, although the drag force has been considered as an objective function [30], [42], implementation of the momentum exchange algorithm (MEA) has not been realized in an adjoint framework. The MEA is a widely-used, general (meaning it is not restricted to bounce back, cf. [18], [51] for its application to high-order boundary conditions), and accurate [12] method for computing surface forces. Therefore, it is a good inclusion for an adjoint solver that seeks wide applicability in aerodynamic LBM-based design. Furthermore, it has already been implemented for a quantum LBM solver, where it is likely the most efficient (and viable) way to compute surface forces [52].

5. Despite the high storage cost of obtaining unsteady adjoint gradients, only one paper [37] in the LBM field has proposed a robust[7] low storage technique and the proposed methodology sacrifices gradient accuracy. It is necessary to find a method that does not have this limitation. A simple and commonly used approach is the so-called checkpointing technique [53].

6. For shape optimization, the shape parametrization strategies are fairly rudimentary, especially compared to the more mature field of Navier-Stokes based shape optimization [54].

It was also determined that a discrete adjoint will be employed given its efficient implementation and exact gradients.

[7] The technique of time-averaging the flow fields will not yield accurate gradients and may not even lead to a converged adjoint field.

Having surveyed the relevant literature and identified research gaps, it is now possible to formulate the research questions that drive the thesis. The main overarching research question is:

Main research question

What are the essential components of a discrete, general, and extensible adjoint LBM framework capable of industrially relevant shape optimization across a wide range of Reynolds numbers?

I propose the following sub-questions and link them to the a priori requirements and identified research gaps:

Research sub-questions

1. How can state of the art LBM features — specifically collision models, grid refinement strategies, and absorption regions — be incorporated into a discrete adjoint LBM? *This question addresses research gap 1 and requirements 1, 2 and 5.*
2. How can adjoint boundary conditions be expressed in a general and systematic way that aids the extensibility of future solver developments? *This question addresses research gap 2 and 3, and requirements 3 and 4.*
3. How can steady and unsteady adjoint-based gradients of the lift and drag coefficients be obtained within a robust, memory efficient, and accurate LBM framework? *This question addresses research gap 4 and 5, and requirements 6 and 8.*
4. In an LBM setting, how can a robust and flexible parametrization of the geometry be incorporated into an adjoint-based shape optimization pipeline? *This question addresses research gap 6, and requirement 10.*

Requirements 7 and 9 are not addressed in this work, as the associated technical challenges were deemed unattainable within the time frame of the thesis.

To address these questions, this thesis is structured in the following way: In chapter 2, the features of the forward LBM code are described, based on implementation by Coreixas and Latt [14]. To satisfy the requirement of a quantum friendly boundary condition on the obstacle, a novel treatment based on the Brinkman penalisation approach is presented.[8] In chapter 3, the discrete adjoint methodology is presented and applied to the forward code to obtain a discrete adjoint solver. A checkpointing routine is described for low storage unsteady adjoint gradients as is the integration of free form deformation shape parametrization into the optimization pipeline.

Then, in chapter 4, the proposed adjoint solver and shape optimization pipelines are validated and assessed. Principally, this is done on an external, steady airfoil case as defined in [18] and the internal (i.e. wind tunnel), unsteady vortex-shedding cylinder from [55]. The respective QoIs are the lift to drag ratio (which should be maximised) and drag coefficient (which should be minimised). To demonstrate the simplicity with which the solver can be extended to higher order no-slip boundary conditions (which are better suited for a purely classical high Reynolds number optimization case), a toy problem using an interpolated bounce back boundary condition is studied. The work concludes by reassessing the research questions and providing recommendations for future research.

[8] Towards the end of the project, a method to obtain gradients using halfway bounce back on the obstacle was partially, but successfully, tested. This may allow for a very straightforward quantum implementation of the forward boundary condition. Preliminary results are given in Appendix E.

A note on implementation All the methods in this work are implemented in Python. The forward and adjoint LBMs use the just-in-time compiler JAX [56] for increased performance and GPU support. However, optimal programming techniques have not been employed so no results on the solver's computational performance are presented.

Chapter 2: The Forward Lattice Boltzmann Method

In this chapter, the features of the forward LBM code are described. In section 2.1 the basic LBM algorithm is presented. Then, the collision operator is specified (section 2.2), followed by the boundary conditions (section 2.3). Here, a quantum-friendly treatment for the obstacle is proposed. Section 2.4 details the grid-refinement strategy, and the chapter concludes with a description of the methods used to compute the QoIs (section 2.5).

2.1. The Lattice Boltzmann Method

Lattice Boltzmann methods are a class of numerical schemes used for computational fluid dynamics. The most commonly used Lattice Boltzmann method (LBM) describes isothermal fluids at low Mach numbers [12]. Macroscopically, these flows are governed by the weakly compressible Navier-Stokes equations:

$$\begin{cases} \partial_t \rho + \partial_\alpha(\rho u_\alpha) = 0, \\ \partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta + p\delta_{\alpha\beta}) = \partial_\beta(\Pi_{\alpha\beta}), \\ p = c_s^2 \rho \end{cases} \tag{2.1.1}$$

where ρ is the density and ρu_α a component of the momentum. Instead of the energy equation, the isothermal equation of state links the pressure to the density via the speed of sound c_s. $\Pi_{\alpha\beta}$ is the viscous stress tensor (see [12] for its definition), $\delta_{\alpha\beta}$ the Kronecker delta and α, β are tensor notation stand-ins for the spatial coordinates x, y (in 2D).

The LBM does not directly solve for ρ and ρu_α, rather, it evolves the mesoscopic discrete-velocity Boltzmann equation (DVBE), (2.1.2). The DVBE describes the behaviour of a particle distribution function, f_i, representing the density of fluid particles moving with a discrete velocity $\boldsymbol{\xi}_i$ at a given location x and time t[1]. Alternatively, f_i is sometimes referred to as a population. (2.1.2) states that f_i evolves due to advection and the interactions of microscopic particles, modelled by a general collision operator Ω_i.

$$\partial_t f_i + \boldsymbol{\xi}_i \cdot \nabla f_i = \Omega_i \tag{2.1.2}$$

The DVBE can asymptotically recover the weakly compressible Navier-Stokes equations if a discrete set of velocities (called a *lattice*) of sufficiently high order (i.e. with enough velocities) is used. For the 2D flows that concern this work, the most commonly used lattice contains 9 velocities so is referred to as D2Q9. Assuming dimensionless *lattice units*, defined such that $\Delta x = \Delta t = 1$, D2Q9 can be specified as: $i \in \{0, 1, 2, ..., 8\}$ and

$$\boldsymbol{\xi}_i = \begin{cases} (0,0), & i = 0, \\ (\pm 1, 0),\ (0, \pm 1), & i = 1, 2, 3, 4, \\ (\pm 1, \pm 1), & i = 5, 6, 7, 8. \end{cases}$$

Figure 2.1 graphically represents it.

[1] The DVBE is a simplification of the Boltzmann equation, where the state variable is the continuous velocity particle distribution function representing the density of particles per unit of velocity space.

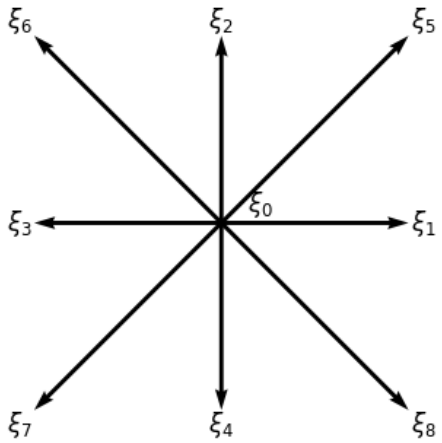


Figure 2.1.: D2Q9 lattice

ρ and ρu_α are computed from f_i by taking moments, as in (2.1.3) and (2.1.4). The second equality, stating that the discrete moments are equal to the moments of the continuous-velocity distribution function (which is modelled by the Boltzmann equation) is a necessary condition to ensure that the DVBE accurately recovers macroscopic conservation laws.

$$\rho = \sum_i f_i = \int_{-\infty}^{\infty} f_(x, \boldsymbol{\xi}, t)\, \mathrm{d}^3\xi \tag{2.1.3}$$

$$\rho \mathbf{u} = \sum_i \boldsymbol{\xi}_i f_i = \int_{-\infty}^{\infty} \boldsymbol{\xi}\, f_(x, \boldsymbol{\xi}, t)\, \mathrm{d}^3\xi \tag{2.1.4}$$

(2.1.3) and (2.1.4) can be rewritten in a more general notation that will be helpful to define the collision operator in section 2.2. Let

$$M_{pq} = \sum_i \xi_{i,x}^p \xi_{i,y}^q f_i \tag{2.1.5}$$

denote a moment of the distribution function with order $p+q$. Then, $\rho = M_{00}$, $\rho u_x = M_{10}$ and $\rho u_y = M_{01}$.

Following a discretization in space and time using the method of characteristics and the trapezoidal rule, the fully discrete Lattice Boltzmann equation reads (in lattice units):

$$f_i(\mathbf{x}_k + \boldsymbol{\xi}_i\, t_{n+1}) - f_i(\mathbf{x}_k, t_n) = \Omega_i(\mathbf{x}_k, t_n) \tag{2.1.6}$$

where $\mathbf{x}_k$ is a node on the computational mesh and t_n a discrete time. In this work, for reasons related to the implementation of boundary conditions and mesh refinement schemes, I consider $\mathbf{x}_k$ to denote the centre of the k-th square cell, as shown in Figure 2.6. Under this interpretation, the mass contained in the cell is given (in 2D) by $f_i(\mathbf{x}_k, t_n)\Delta x^2$.

Implementing (2.1.6) is straightforward. For every time step, an intermediate variable called the post-collision distribution function, $\hat{f}_i(\mathbf{x}_k, t_n)$, is computed for each node as:

$$\hat{f}_i(\mathbf{x}_k, t_n) = f_i(\mathbf{x}_k, t_n) + \Omega_i(\mathbf{x}_k, t).$$

This step, which is generally non-linear, is called collision. In the next step, called streaming, the post-collision distribution functions are advected to the neighbouring cells following the rule (2.1.7).

$$f_i(\mathbf{x}_k + \boldsymbol{\xi}_i\, t_{n+1}) = \hat{f}_i(\mathbf{x}_k, t_n) \tag{2.1.7}$$

Boundary conditions are imposed after streaming to determine populations that are unknown because they originate from a fluid node, thus completing an LBM iteration.

At each time step, macroscopic variables can be reconstructed from (2.1.3) and (2.1.4). The pressure can be obtained from the equation of state in (2.1.1) with $c_s^2 = 1/3$ (in lattice units) for D2Q9. Although in principle the computational mesh must be Cartesian and uniform to ensure that populations always advect exactly to the neighbouring nodes, there exist methods for local grid refinement. Section2.4 presents the one developed by Rohde et al. [57].

For the adjoint derivation, it will be useful to express (2.1.7) as an iterative equation for the flattened vector $\mathbf{F}^n = (f_i(\mathbf{x}_k, t_n))_{(0\leq i\leq 8, 0\leq k\leq N_x N_y - 1)}$, where N_x and N_y are the number of nodes in the x and y directions. The collision step is represented by the general non-linear operation $\hat{\mathbf{F}} = \mathscr{C}(\mathbf{F})$. The advective step is linear, so it can be expressed as a matrix S. In the most general case, boundary conditions are non-linear transformations of the post-collision distributions. However, all the rules employed in this work, along with a wide array of high-order boundary treatments[2], are linear. Therefore, I assume that the boundary conditions are incorporated directly into the matrix S. The resulting iteration is the composition of collision and streaming, complemented in the current implementation with a zero velocity and uniform density (equilibrium) initial condition.

$$\begin{cases} \mathbf{F}^0 = \mathbf{F}^{eq}(\mathbf{u} = 0, \rho = 1) \\ \mathbf{F}^{n+1} = S \cdot \mathscr{C}(\mathbf{F}^n) \equiv \Phi^n(\mathbf{F}^n), \quad n = 0, 1, ..., N-1 \end{cases} \tag{2.1.8}$$

2.2. The collision operator

The purpose of a collision operator is to model the effect of microscopic particle collisions on the distribution function. According to kinetic theory, these collisions drive f_i to an equilibrium state f_i^{eq} [12]. The simplest and most widely-used model, (2.2.1), known as the Bhatnagar-Gross-Krook (BGK) collision operator, describes this process by relaxing populations to their equilibrium state with a single characteristic time τ.

$$\Omega_i = -\frac{1}{\tau}(f_i - f_i^{eq}) \tag{2.2.1}$$

The form of the equilibrium distribution function determines the macroscopic equations recovered by the DVBE. The fully continuous Boltzmann equation specifies f^{eq} as the Maxwell-Boltzmann distribution (or Maxwellian). The BGK-DVBE, however, only uses an expansion of the Maxwellian in Hermite polynomials up to second order, as shown in (2.2.2). With such a representation, the first two moments of f_i^{eq}, $M_{00}^{eq} = \rho$, $M_{10}^{eq} = \rho u_x$ and $M_{01}^{eq} = \rho u_y$, exactly match the first two moments of the Maxwellian. This is the necessary condition to recover (2.1.1), precisely the macroscopic evolution equation of these first two moments. In (2.2.2), $w_i = \{w_0 = 4/9, w_{1-4} = 1/9, w_{5-8} = 1/36\}$ are Gaussian quadrature weights, specific to D2Q9, that enforce the equality between discrete and continuous moments.

$$f_i^{eq} = w_i \rho \left[1 + \frac{\boldsymbol{\xi}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\boldsymbol{\xi}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u} \cdot \mathbf{u}}{2c_s^2} \right] \tag{2.2.2}$$

In the context of the fully discrete LBM, the BGK collision step is:

$$\hat{f}_i(\mathbf{x}_k, t_n) = f_i - \frac{1}{\tau}(f_i - f_i^{eq}) = f_i - \frac{1}{\tau} f_i^{neq} = f_i^{eq} + \left(1 - \frac{1}{\tau}\right) f_i^{neq}, \tag{2.2.3}$$

[2] For example, the interpolated bounce back boundary conditions by Bouzidi et al. [51], Yu et al. [58] and Ginzburg et al. [59], or the volumetric scheme by Hoefnagel et al. [18].

where f_i^{neq} are the non-equilibrium distribution functions. For the D2Q9 lattice, τ is related to the fluid's kinematic viscosity (in lattice units) by (2.2.4).

$$\nu = \frac{2\tau - 1}{6} \tag{2.2.4}$$

Although the BKG operator has been used very successfully in many applications, it suffers from stability issues in the low viscosity limit, $\tau \to 1/2$ [3], and for non-vanishing Mach numbers (greater than 0.1) [24]. These are conditions that arise for under-resolved weakly-compressible aerodynamic simulations, primarily at high Reynolds numbers [16], [24], [25], as would be expected in classical industrial applications.

The origin of these instabilities is best understood in the context of linear stability analysis. The linearization of the 2D weakly compressible Navier-Stokes equations results in 3 waves (2 acoustic and one shear) carrying hydrodynamic information. In contrast, the linearized D2Q9 LBM results in 9 waves, 3 hydrodynamic and 6 originating from higher order moments of the distribution function. Works such as [16], [24], [60] have shown that discretization errors result in spurious couplings of the hydrodynamic and non-hydrodynamic modes, eventually leading to instabilities. Furthermore, due to the truncation of the equilibrium function, a modelling error of $\mathcal{O}(Ma^3)$ exists in the reconstruction of the viscous stress tensor [12]. This limits the maximum stable Mach number for the LBM to $Ma_{max} = \sqrt{3} - 1 \approx 0.73$, although significant errors in the transport and dissipation of the hydrodynamic modes start occurring for lower Mach numbers (around 0.4) [24].

The objective of this thesis is to take the first step towards implementing an adjoint solver and shape optimization pipeline applicable to the full range of Reynolds and Mach numbers that are well described by the weakly-compressible Navier-Stokes based LBM. To achieve this objective, it is necessary to implement a robust collision operator that overcomes the stability issues of BGK.

To this end, I chose the recursive regularized (RR) collision operator [24], [61]. Thorough investigations using linear stability analysis, plus non-linear benchmarking using the double shear layer as a test case [26], [61], and applications to acoustics studies [27], have demonstrated that this operator consistently offers a good compromise between stability and accuracy.[4]

In a recent paper, Coreixas and Latt claim good agreement between LBM-RR simulations and standard benchmarking cases including the 3030N three element airfoil (for which local Mach numbers exceed 0.5 and the Reynolds number is $Re = 1.7 \times 10^6$) and a cylinder in a uniform flow at a wide range of Reynolds and Mach numbers [14] (including a run where the local Mach number reaches 0.72 (!) although the accuracy of the simulation is not assessed due to lack of reference data).

Another recent study by Schukmann et al. demonstrates that this operator emits fewer spurious acoustic waves at grid refinement interfaces than the BGK and raw moment models [27], while maintaining a low artificial viscosity. The PhD thesis of Astoul shows that at moderate Mach numbers with low viscosity the DRT collision model (used by Cheylan et al. in a previous shape optimization paper [42]) is unable to match the performance of RR, even with added filtering [25].

The RR operator is a member of the class of multiple relaxation time (MRT) collision operators [24].[5] These relax all non-conserved moments (M_{pq} with $p+q > 1$) of the distri-

[3] τ is a free parameter when setting up an LBM simulation that bears no influence on the real, physical viscosity being simulated. In practice, τ is chosen (in lattice units) based on the mesh resolution and Reynolds number. This determines a viscosity, also in lattice units, that is converted to the the physical viscosity using a unit conversion factor. It is the vanishing viscosity in lattice units that leads to issues in the BGK-LBM, which can therefore appear at any Re, as long as $\tau \to 1/2$.

[4] Compared to the raw moment, Hermite moment, central moment, central Hermite moment, and cumulant operators.

[5] In the adjoint LBM literature review MRT always referred to the raw moment collision operator

bution functions toward their equilibrium values, each with its own characteristic time instead of using a single τ for all populations. Thus, the user can directly control the influence of higher order moments on the solution, increasing damping if necessary to improve stability. Furthermore, MRT operators employ an extended equilibrium distribution (partially) up to fourth order, as shown below.

$$
\begin{aligned}
f_0^{eq} &= \frac{4}{9}\left[\rho - \frac{3}{2}\left(A_{20}^{eq} + A_{02}^{eq}\right) + \frac{9}{4}A_{22}^{eq}\right], \\
f_1^{eq} &= \frac{1}{9}\left[\rho + 3\rho u_x + \frac{3}{2}(2A_{20}^{eq} - A_{02}^{eq}) - \frac{9}{2}A_{12}^{eq} - \frac{9}{2}A_{22}^{eq}\right], \\
f_2^{eq} &= \frac{1}{9}\left[\rho + 3\rho u_y + \frac{3}{2}(2A_{02}^{eq} - A_{20}^{eq}) - \frac{9}{2}A_{21}^{eq} - \frac{9}{2}A_{22}^{eq}\right], \\
f_3^{eq} &= \frac{1}{9}\left[\rho - 3\rho u_x + \frac{3}{2}(2A_{20}^{eq} - A_{02}^{eq}) + \frac{9}{2}A_{12}^{eq} - \frac{9}{2}A_{22}^{eq}\right], \\
f_4^{eq} &= \frac{1}{9}\left[\rho - 3\rho u_y + \frac{3}{2}(2A_{02}^{eq} - A_{20}^{eq}) + \frac{9}{2}A_{21}^{eq} - \frac{9}{2}A_{22}^{eq}\right], \\
f_5^{eq} &= \frac{1}{36}\left[\rho + 3\rho(u_x + u_y) + 3(A_{20}^{eq} + A_{02}^{eq}) + 9A_{11}^{eq} + 9(A_{21}^{eq} + A_{12}^{eq}) + 9A_{22}^{eq}\right], \\
f_6^{eq} &= \frac{1}{36}\left[\rho + 3\rho(-u_x + u_y) + 3(A_{20}^{eq} + A_{02}^{eq}) - 9A_{11}^{eq} + 9(A_{21}^{eq} - A_{12}^{eq}) + 9A_{22}^{eq}\right], \\
f_7^{eq} &= \frac{1}{36}\left[\rho + 3\rho(-u_x - u_y) + 3(A_{20}^{eq} + A_{02}^{eq}) + 9A_{11}^{eq} + 9(-A_{21}^{eq} - A_{12}^{eq}) + 9A_{22}^{eq}\right], \\
f_8^{eq} &= \frac{1}{36}\left[\rho + 3\rho(u_x - u_y) + 3(A_{20}^{eq} + A_{02}^{eq}) - 9A_{11}^{eq} + 9(-A_{21}^{eq} + A_{12}^{eq}) + 9A_{22}^{eq}\right].
\end{aligned}
\tag{2.2.5}
$$

This has been shown to reduce the error of $\mathcal{O}(Ma^3)$ in the reconstruction of the viscous stress tensor, lowering the transport and dissipation errors of the hydrodynamic modes [24].

Variations in the different types of MRT models arise in the space in which moments are relaxed. One can relax the velocity moments of f_i: (2.1.5)—charmingly referred to as *raw* moments—or the Hermite moments $A_{pq} = \sum_i \mathcal{H}_{pq}(f_i)i$, central Hermite moments, cumulants, or central moments. The RR operator relaxes the Hermite moments but explicitly reconstructs the non-equilibrium moments with $p + q > 2$ (which do not play a part in hydrodynamics), using recursive formulae derived from Chapman-Enksog expansion (hence the name). Because three higher order moments are reconstructed as recursive functions of lower order moments, RR reduces the dimension of the moment space from 9 to 6. This diminishes the likelihood of spurious interactions between the acoustic, shear, and non-hydrodynamic modes [16], [24], [25], [60].

(2.2.7) shows the collision operation in vector form [24], with $\hat{\mathbf{f}}$ the vector of post collision distribution functions as defined in (2.2.10) and $\mathbf{f}$ the analogous distribution function vector. At a high level, the collision consists of a non-linear function $\mathcal{C}(\mathbf{f})$ whereby the moments of f are relaxed to their equilibrium values, followed by two linear transformations. The first of these, (2.2.9), translates the Hermite moments to the raw moment space, the second results in the post collision functions, (2.2.10).

(2.2.8), closed by (2.2.11), (2.2.12) and (2.2.13), shows the relaxation procedure. To enforce conservation laws, the first three moments cannot be relaxed. A_{11} follows a relaxation time $\tau_2 = 1/\omega_2$ which is related to the kinematic viscosity by

$$
\nu = \frac{2\tau_2 - 1}{6}. \tag{2.2.6}
$$

A_{20} and A_{02} are both governed by $\omega_+ = \frac{1}{2}(\omega_4 + \omega_1)$ and $\omega_- = \frac{1}{2}(\omega_4 - \omega_1)$, where $\omega_1 = \omega_2$ must be satisfied for proper reconstruction of the momentum equation. Higher order moments are relaxed with the free parameters ω_3 and ω_4, where ω_4 is related to the bulk

viscosity following (2.2.4). The regularization strategy is to set $\omega_3 = \omega_4 = 1$, thus critcally damping the higher order moments. In this work, however, I use the single relaxation time approach $\omega_1 = \omega_2 = \omega_3 = \omega_4 = 1/\tau$, which has been shown to reduce the amount of numerical dissipation in the system while maintaining good stability [26], [27]. Note, the terms explicitly reconstructed in the RR model are A_{21}^{neq}, A_{21}^{neq} and A_{22}^{neq} in (2.2.11).

$$\hat{\mathbf{f}} = T \cdot R \cdot \mathcal{C}(\mathbf{f}) \tag{2.2.7}$$

$$\mathcal{C}(\mathbf{f}) = \hat{\mathbf{A}} = \begin{cases} \hat{A}_{00} &= \rho, \\ \hat{A}_{10} &= \rho u_x, \\ \hat{A}_{01} &= \rho u_y, \\ \hat{A}_{11} &= (1-\omega_2)A_{11}^{\text{neq}} + A_{11}^{\text{eq}}, \\ \hat{A}_{20} &= A_{20} - \omega_+ A_{20}^{\text{neq}} - \omega_- A_{02}^{\text{neq}}, \\ \hat{A}_{02} &= A_{02} - \omega_- A_{20}^{\text{neq}} - \omega_+ A_{02}^{\text{neq}}, \\ \hat{A}_{21} &= (1-\omega_3)A_{21}^{\text{neq}} + A_{21}^{\text{eq}}, \\ \hat{A}_{12} &= (1-\omega_3)A_{12}^{\text{neq}} + A_{12}^{\text{eq}}, \\ \hat{A}_{22} &= (1-\omega_4)A_{22}^{\text{neq}} + A_{22}^{\text{eq}}. \end{cases} \tag{2.2.8}$$

$$\hat{\mathbf{M}} = R\hat{\mathbf{A}} = \begin{pmatrix} \hat{M}_{00} \\ \hat{M}_{10} \\ \hat{M}_{01} \\ \hat{M}_{11} \\ \hat{M}_{20} \\ \hat{M}_{02} \\ \hat{M}_{21} \\ \hat{M}_{12} \\ \hat{M}_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ c_s^2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ c_s^2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & c_s^2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & c_s^2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ c_s^4 & 0 & 0 & 0 & c_s^2 & c_s^2 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{A}_{00} \\ \hat{A}_{10} \\ \hat{A}_{01} \\ \hat{A}_{11} \\ \hat{A}_{20} \\ \hat{A}_{02} \\ \hat{A}_{21} \\ \hat{A}_{12} \\ \hat{A}_{22} \end{pmatrix} \tag{2.2.9}$$

$$\hat{\mathbf{f}} = T\hat{\mathbf{M}} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \\ \hat{f}_5 \\ \hat{f}_6 \\ \hat{f}_7 \\ \hat{f}_8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & -\frac{1}{4} & 0 & 0 & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} \hat{M}_{00} \\ \hat{M}_{10} \\ \hat{M}_{01} \\ \hat{M}_{11} \\ \hat{M}_{20} \\ \hat{M}_{02} \\ \hat{M}_{21} \\ \hat{M}_{12} \\ \hat{M}_{22} \end{pmatrix} \tag{2.2.10}$$

$$\begin{cases} A_{20}^{\text{neq}} &= A_{20} - A_{20}^{\text{eq}}, \\ A_{02}^{\text{neq}} &= A_{02} - A_{02}^{\text{eq}}, \\ A_{11}^{\text{neq}} &= A_{11} - A_{11}^{\text{eq}}, \\ A_{21}^{\text{neq}} &= u_y A_{20}^{\text{neq}} + 2u_x A_{11}^{\text{neq}}, \\ A_{12}^{\text{neq}} &= u_x A_{02}^{\text{neq}} + 2u_y A_{11}^{\text{neq}}, \\ A_{22}^{\text{neq}} &= u_y^2 A_{20}^{\text{neq}} + u_x^2 A_{02}^{\text{neq}} + 4u_x u_y A_{11}^{\text{neq}}, \end{cases} \tag{2.2.11}$$

$$\begin{cases} A_{00} &= \rho, \\ A_{10} &= \rho u_x, \\ A_{01} &= \rho u_y, \\ A_{11} &= \sum_i \xi_{i_y} \xi_{i_y} f_i, \\ A_{20} &= \sum_i (\xi_{i_x}^2 - c_s^2) f_i, \\ A_{02} &= \sum_i (\xi_{i_y}^2 - c_s^2) f_i. \end{cases} \tag{2.2.12}$$

$$\begin{cases} A_{11}^{\text{eq}} &= \rho u_x u_y, \\ A_{20}^{\text{eq}} &= \rho u_x^2, \\ A_{02}^{\text{eq}} &= \rho u_y^2, \\ A_{21}^{\text{eq}} &= A_{20}^{\text{eq}} u_y, \\ A_{12}^{\text{eq}} &= A_{02}^{\text{eq}} u_x, \\ A_{22}^{\text{eq}} &= \frac{A_{20}^{\text{eq}} A_{02}^{\text{eq}}}{\rho}. \end{cases} \tag{2.2.13}$$

2.3. Boundary conditions

As stated in chapter 1, one of the requirements placed on the solver is to incorporate a mass conserving boundary condition on the obstacle for compatibility with a quantum LBM solver. In this section, a novel approach for LBM shape optimization — called porous bounce back (PBB) — is detailed, followed by a presentation of the standard rules used on the domain's edges. In a nutshell, PBB combines the Brinkman porosity model used in e.g. [30], with the normal bounce back boundary condition.

2.3.1. Solid walls

For walls with a macroscopic no-slip boundary condition, the simplest mesoscopic approach is halfway bounce back. This method assumes that the wall lies halfway between a fluid node and a solid node. Then, during the streaming step, populations moving from the fluid to the solid node hit the wall, reverse direction, and stream back to the fluid node where they came from, all in the space of one time step. As an example, Figure 2.2, adapted from [12], shows how the post-collision population $\hat{f}_1(\mathbf{x}_b, t)$ streams toward the solid node $\mathbf{x}_{b+1}$, hits the wall (dashed line), reverses direction, and streams back to $\mathbf{x}_b$, becoming $f_3(\mathbf{x}_b, t_{n+1} = t + \Delta t)$.

Following a collide-then-stream implementation of the LBM, the bounce back rule is:

$$f_{i^-}(\mathbf{x}_b, t_{n+1}) = \hat{f}_i(\mathbf{x}_b, t) \tag{2.3.1}$$

where f_{i^-} is the (unknown) population such that $\boldsymbol{\xi}_{i^-} = -\boldsymbol{\xi}_i$ and $\mathbf{x}_b$ is a node adjacent to the wall. For walls exactly halfway between the fluid and solid node, the method is second-order accurate, but this degrades to first-order for a general boundary. The method always conserves mass because it simply reflects populations off the wall.

Inspecting (2.3.1) and observing that it does not depend on the boundary geometry leads to the conclusion that flow sensitivities with respect to the shape cannot be obtained using bounce back. Instead, a commonly used boundary condition is the model introduced by Spaid and Phelan for porous flow problems [62] and then adapted by Pingen et al. for shape and topology optimization [30].

Pingen et al.'s approach defines the object's boundary, the set of fully solid nodes—which is a subset of the nodes within the boundary—and the set of porous nodes, as shown in Figure 2.3. Then, for the porous *and* solid nodes, the method modifies the collision step as follows.

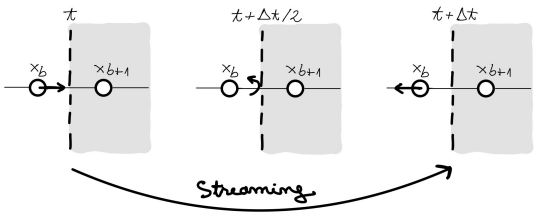


Figure 2.2.: Illustration of the bounce back process that transforms the post-collision population $\hat{f}_1$ into f_3 at the next time step.

The post collision distributions are generally a function of ρ, $\mathbf{u}$ and $\mathbf{f} = [f_0, ..., f_8]$:

$$\hat{f}_i = \hat{f}_i(\rho, \mathbf{u}, \mathbf{f}, \mathbf{x}_k, t_n) \tag{2.3.2}$$

Defining a porosity $\varepsilon \in [0, 1]$, where $\varepsilon = 0$ is a solid node and $\varepsilon = 1$ a fluid node, (2.3.2) changes to

$$\hat{f}_i(\mathbf{x}_k, t_n) = \hat{f}_i(\rho, \varepsilon\mathbf{u}, \mathbf{f}, \mathbf{x}_k, t_n). \tag{2.3.3}$$

Intermediate porosities are assigned based on the distance from the porous node to the boundary. Shape changes propagate into the flow field via these points. (2.3.3) is particularized for RR in Appendix B.

We can verify that this approach conserves mass by checking if $\rho_{\text{pre-collision}}(\mathbf{x}_k) = \rho_{\text{post-collision}}(\mathbf{x}_k)$. $\rho_{\text{post-collision}}$ is the first moment of the post collision distribution functions (as given in (2.2.7)):

$$\rho_{\text{post-collision}} = \hat{\mathbf{f}}^\intercal \mathbf{1} = \mathcal{C}(\mathbf{f})^\intercal R^\intercal T^\intercal \mathbf{1} \tag{2.3.4}$$

where $\intercal$ denotes a transposition and $\mathbf{1} \in \mathbb{R}^9 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$. By direct computation,

$$\rho_{\text{post-collision}} = \hat{A}_{00} = \rho_{\text{pre-collision}} \tag{2.3.5}$$

$\hat{A}_{00}$ is directly extracted, $\varepsilon\mathbf{u}$ never appears in the calculation, so the operation conserves mass.

The momentum extracted from the flow due to (2.3.3) is:

$$\rho\mathbf{u}_{ex} = \sum_{\text{All nodes}} \rho(\mathbf{u}_{\text{pre-collision}} - \mathbf{u}_{\text{post-collision}}) \tag{2.3.6}$$

where for fluid nodes the difference in momenta is zero [30].

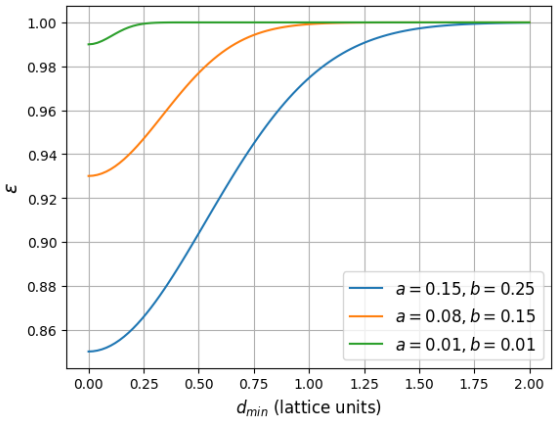


Figure 2.4.: $\varepsilon(d)$, σ_ε is fixed at 1.5.

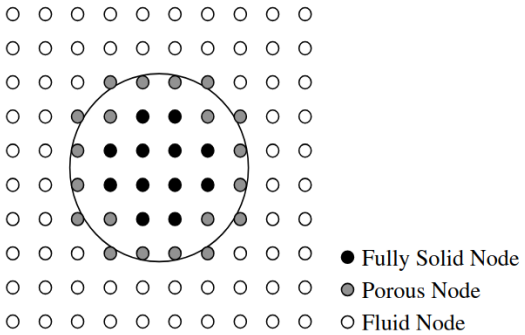


Figure 2.3.: Sketch of solid, porous and fluid nodes around an object. [28]

My proposed approach combines Pingen et al.'s method with the normal bounce back rule so I refer to it as porous bounce back (PBB). Let

$$\tilde{\mathbf{x}}_{k_i} = \mathbf{x}_k + \boldsymbol{\xi}_i. \tag{2.3.7}$$

Then, the set of boundary nodes, X_b, is:

$$X_b = \{\tilde{\mathbf{x}}_{k_i} | \tilde{\mathbf{x}}_{k_i} \notin X_f, \mathbf{x}_k \in X_f\} \tag{2.3.8}$$

where X_f is the set of fluid nodes. For $\mathbf{x}_k \in X_b$ I apply

$$f_{i^-}(\mathbf{x_k}, t_{n+1}) = \hat{f}_i(\rho, \varepsilon\mathbf{u}, \mathbf{x}_k, t) \tag{2.3.9}$$

with

$$\varepsilon = 1 - a \exp\left[-\frac{d^2}{\sigma_\varepsilon^2 b}\right], \tag{2.3.10}$$

where d is the (minimum) distance from $\mathbf{x}_k$ to the boundary (in lattice units) and a, b and σ_ε are free parameters. Figure 2.4 shows how the free parameters influence the porosity ε. Choosing a, b, and σ_ε comes down to balancing two considerations. Firstly, the $\varepsilon(d)$ curve should not be too steep as this may lead to very small cost-function gradients given small values of $\frac{d\varepsilon}{dd}$ as $d \to \infty$. This may slow down the optimization. Secondly, the porosity should not be too large (i.e., $\varepsilon \to 0$) or too small since, from experience, this may lead to strong over- or under-predictions of the drag. In theory, it is possible to choose constant parameters such that drag is accurately computed even for coarse meshes, but this requires a priori knowledge of the mesh-independent solution. In the

end, this second consideration is not too important as, regardless of the parameters chosen, all PBB solutions converge to the same mesh-independent result. Thus, with the first consideration as a priority, the values chosen are:

$$\begin{cases} a = 0.15 \\ b = 0.25 \\ \sigma_\varepsilon = 1.5. \end{cases} \tag{2.3.11}$$

The exact form of the modified RR operator is reported in Appendix B.

Macroscopically, PBB is equivalent to imposing a no-slip boundary with (generally) first order accuracy and then, in the layer of boundary nodes, simulating the flow through a material with variable porosity using the Brinkman model (shown here in incompressible form):

$$\nu \nabla^2 \mathbf{u} - \beta \mathbf{u} = \frac{1}{\rho} \nabla p$$

where $\beta(\varepsilon)$ is a parameter relating to the material properties [62]. The Brinkman model has been shown to be compatible with a flow governed by the Navier-Stokes equation, meaning the velocity and stress are continuous over the porous-open fluid interface. Thus, PBB is a fully consistent model for a Navier-Stokes-Brinkman flow with no-slip boundary conditions, suggesting that in the limit of a fine mesh, as the porous layer vanishes, it is also a consistent no-slip boundary condition for the Navier-Stokes equations.

PBB is more attractive than Pingen et al.'s approach, which, though also physically consistent, assumes the object is made entirely of the porous material and that the flow eventually gets stuck, without clearly delineating the boundary. Kreissl et al. have reported diminished accuracy for unsteady flows when using the purely porous method due to spurious pressure fluctuations within the obstacle [33]. In their case, these fluctuations contaminated their pressure drop cost function, leading to a poor(er) optimization. In the present work, the integration of the surface forces may be compromised.

Furthermore, PBB is more in line with this thesis' goal of an extensible adjoint solver. To implement Pingen et al.'s approach, one has to:

1. Identify the points within the obstacle (solid nodes) and those near the boundary that make up the porous nodes.
2. Compute the distance of the porous nodes to the boundary to assign ε.
3. If surface forces are desired, apply (2.3.6) to compute the extracted momentum and divide by Δt to convert to a force.

For PBB one must also implement the following routines:

1. Identify the boundary nodes for each lattice velocity (i.e. the nodes $\mathbf{x}_{k_j}$ for which the velocity $\boldsymbol{\xi}_{j^-}$ is unknown). This is also a necessary routine for most, if not all, higher order boundary conditions.
2. Surface forces must be obtained by combining (2.3.6) with the momentum exchange algorithm (MEA) to account for bounce back's contribution. This lends the adjoint pipeline extensibility to both quantum and classical architectures [18], [52].

PBB exists to fulfil the mass-conservation requirement set by the quantum group. However, this remains an attractive feature for classical codes. The mass leakage associated with some high-order boundary conditions, specifically interpolated bounce back schemes (such as the ones used by Cheylan et al. and Kusano), has been shown to significantly affect the flow fields and related QoIs. For example, this boundary condition results in a time dependent mean drag force for a vortex shedding cylinder — as studied

in [63] — with deviations of up to 40% (!) from the true value. The effect worsens for under-resolved simulations, as would be expected in (classical) industrial applications.

Furthermore, the scheme has some attractive properties from a computational standpoint. Particularly when compared to interpolated bounce back schemes, the local nature of PBB is much more in line with the single instruction multiple data (SIMD) concept, leading to a much more efficient implementation on a GPU architecture.

Nevertheless, the ideal boundary condition for a quantum solver remains halfway bounce back due to its unitary nature. It is even unclear whether PBB could be successfully implemented for quantum LBM. The TU Delft group is currently approximating the collision operator with a data-driven surrogate model that would need to learn the locally scaled-velocity collision operator, adding even more complexity to the problem [13]. However, a promising methodology (inspired by PBB) for obtaining gradients of halfway bounce back was discovered late in the project. Preliminary results are shown in Appendix E.

2.3.2. Domain boundary conditions

All simulations in this work are carried out on a rectangular grid with the following standard boundary conditions.

On the western edge of the domain, a momentum inlet [12], [14] is imposed using the bounce back rule with a correction:

$$f_i(\mathbf{x}_b, t_{n+1}) = \hat{f}_i(\mathbf{x}_b, t) - 2w_i\rho_{in}\frac{\boldsymbol{\xi}_i \cdot \mathbf{u}_{in}}{c_s^2}. \tag{2.3.12}$$

The exact value of $\mathbf{u}_{in}$ is specific to each simulation and can be found in section 3.6. For all cases, $\rho_{in} = 1$. Note that, although both ρ_{in} and $\mathbf{u}_{in}$ are specified in (2.3.12), the rule does not impose either of these quantities at the inlet—rather, it fixes the momentum $\rho_{in}\mathbf{u}_{in}$.

Two different boundary conditions are applied on the northern and southern edges, depending on the simulation. For the airfoil case, I use the free-slip boundary condition [12], (2.3.13), enforcing $u_n = 0$:

$$f_a(\mathbf{x}_b, t_{n+1}) = f_i(\mathbf{x}_b, t_{n+1}). \tag{2.3.13}$$

a and i are related via $\boldsymbol{\xi}_{a,n} = -\boldsymbol{\xi}_{i,n}$ and $\xi_{a,t} = \xi_{i,t}$ where t and n denote tangential and normal vector components. For the cylinder case, bounce back is used, (2.3.1), to enforce a no-slip velocity condition.

On the eastern edge, zero-gradient of *all* populations (following [14]) is enforced via a simple extrapolation in the direction normal to the boundary:

$$f_i(x_N, t_{n+1}) = f_i(x_{N-1}, t_{n+1}). \tag{2.3.14}$$

For the internal flow of the vortex-shedding cylinder, it was observed that the combination of bounce back on the top and bottom walls and extrapolation at the outlet led to a small mass divergence. This was not observed for the free-slip/extrapolation combination applied to the airfoil case. Therefore, for the cylinder problem, a free-slip boundary condition is specified on the second to last node (in the x-direction), followed by the extrapolation boundary condition:

$$\begin{cases} f_{i^-}(\mathbf{x}_b, t_{n+1}) = \hat{f}_i(\mathbf{x}_b, t), \text{ if } b < N_x - 1 \\ f_a(\mathbf{x}_b, t_{n+1}) = f_i(\mathbf{x}_b, t_{n+1}), \text{ if } b = N_x - 1 \\ f_i(x_N, t_{n+1}) = f_i(x_{N-1}, t_{n+1}), \text{ if } b = N_x. \end{cases} \tag{2.3.15}$$

For this case, this approach ensured that the mass drain at the outlet correctly balanced the mass inflow at the inlet.

The combination of a momentum inlet and an extrapolation outlet was chosen because it is simple to implement, it is used in the paper on which the entire forward solver is based (cf. chapter 1 and [14]), and, more importantly, simple to map to the adjoint solver[6]. However, it is not a consistent boundary treatment for the weakly compressible Navier-Stokes equations, which require a Dirichlet boundary condition for at least one variable at the outlet, typically the density. LBM boundary conditions exist for constant pressure boundaries but are non-linear and thus harder to differentiate. Therefore, to fully determine the problem the pressure relaxation approach described in the following section is essential.

2.3.3. Sponge

The addition of an absorption region, or sponge, to an LBM code is a widely used method typically employed to eliminate spurious acoustic reflections at boundaries for unsteady flows [47], [64]–[67]. Usually, the sponge is a region of increased viscosity in far-field; however, I use the pressure-relaxation approach proposed in [14] as it also eliminates the indeterminateness issue introduced by the inlet and outlet boundary conditions. In this way, the sponge can be seen as an extended outlet boundary condition.

Similarly to the PBB approach, this method modifies one of the macroscopic variables that is input into the collision operator, in this case the density. For a portion of the simulation domain several nodes thick, the collision step follows (2.3.16) - (2.3.18).

$$\hat{f}_i(\mathbf{x}_s, t_n) = \hat{f}_i(\rho_s, \mathbf{u}, \mathbf{f}) \tag{2.3.16}$$

$$\rho_s = \rho(1-\sigma) + \sigma\rho_\infty \tag{2.3.17}$$

$$\sigma = \begin{cases} \sigma_{max}(x - x_{min})/(x_{max} - x_{min}), & x \geq x_{min} \\ \sigma_{max} & x \geq x_{max} \end{cases} \tag{2.3.18}$$

The exact modifications made to the RR operator can be found in Appendix B. The geometry of the sponge is sketched in Figure 2.5 for a general rectangular grid. In this work, the sponge is only applied downstream of the object given the relatively small domains used and the proximity of the obstacle to the inlet. For larger domains, the absorption region is typically defined about the object using a radial basis function. The effect of (2.3.16) is to relax the value of ρ to a fixed freestream value ρ_∞ with a relaxation parameter σ, thus eliminating fluctuations and determining the problem. σ is a function of space which linearly increases from x_{min} to x_{max} and remains constant at $\sigma = \sigma_{max}$ from x_{max} to the outlet. A study of the sensitivity of the flow fields to this parameter is presented in subsection 4.1.4.

2.4. Grid refinement

In many practical computations, it is necessary to use local grid refinement to achieve well-resolved simulations at an affordable cost. This work uses Rohde et al.'s [57] mass and momentum-conserving algorithm. Primarily, this is done as recent benchmarking has shown that mass-conserving schemes limit the spurious noise caused by vortical

[6] This is because the extrapolation outlet is a linear operation and the bounce back rule is affine. For further details see section 3.2.

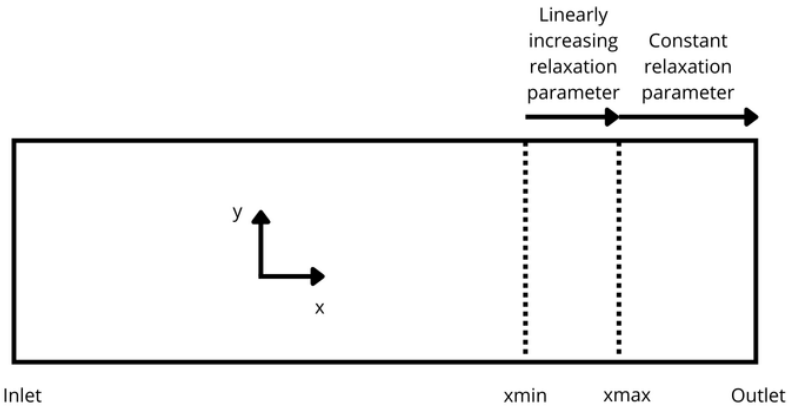


Figure 2.5.: Sketch of sponge geometry for a narrow rectangular grid. The dashed lines indicate the location of the parameters in (2.3.18).

structures crossing refinement interfaces. Although Rohde et al.'s algorithm is only first order accurate it remains more accurate in these aeroacoustic benchmarks than second order interpolation based schemes, such as the one used by Cheylan et al. [27], [68]. There are other, more practical reasons to employ the method. It is very simple to implement since it only uses local information (unlike interpolation schemes which have a wide stencil) and requires no explicit filtering (which also adds to its robustness in high Reynolds number flows). Given its local stencil, it is also very well-suited for adaptive mesh refinement and can handle arbitrary refinement levels.[7] Its robustness and simplicity have made it the scheme of choice for the commercial software PowerFLOW [69] and the open source software waLBerla, and as such, have yielded a wide number of successful, industrially relevant applications (e.g. [64]–[66]).

Rohde et al.'s scheme assumes that the nodes on the computational mesh represent cell centres. Refining a cell means uniformly splitting the cell into $n^2 = \left(\frac{\Delta x_{coarse}}{\Delta x_{fine}}\right)^2$ children (in 2D), as shown in Figure 2.6. In this work only factors of $n = 2$ are considered — accuracy may degrade if higher factors are considered [69] — but multiple refinement levels are allowed.

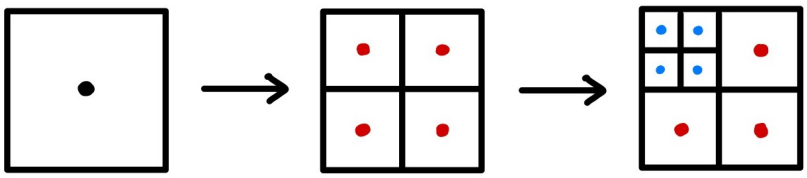


Figure 2.6.: Process of (partially) twice refining a coarse cell by a factor of 2.

To ensure physically consistent simulations, the first consideration for any LBM refinement scheme is to ensure continuity of the non-dimensional parameters, Ma (Mach number) and Re (Reynolds number), over the domain. Ma can be handled by ensuring that the ratio $\Delta x/\Delta t$ is constant between different refinement levels. This means that parameters follow the so-called acoustic scaling and solutions of the LBM will always converge to the weakly compressible Navier-Stokes equations (with a discrepancy of $\mathcal{O}(Ma^2)$ to the

[7] At one point, it was considered to implement an adjoint based adaptive mesh refinement routine, so this was a major consideration

incompressible Navier-Stokes equations) [12], [14]. Re must be dealt with by rescaling the local relaxation time, τ, over the refinement levels to ensure the kinematic viscosity remains constant. Given the relaxation time on a coarse grid, τ_c, the relaxation time on its fine child grid, τ_f, is:

$$\tau_f = 2\tau_c - 0.5. \tag{2.4.1}$$

Rohde et al.'s scheme describes how, over one coarse timestep (or two fine steps), a coarse grid is coupled to a nested fine grid through a grid interface layer, illustrated in Figure 2.7. The coarse and fine grids are made up of regular coarse and fine nodes, x^c, x^f, and the interface layer of coarse interface nodes, $x^{c,I}$, and fine interface nodes, $x^{f,I}$. At a high level, the method achieves conservation of mass and momentum through the following couplings. Firstly, to create the populations that stream into the fine grid from the coarse grid, the density stored in $f_i^{c,I}$ is simply copied to each nested $f_i^{f,I}$. Thus, the macroscopic moments in the coarse cell stay constant. Secondly, populations streaming from the fine grid to the coarse grid are obtained by averaging $f_i^{f,I}$ over a coarse cell, which is also a conservative operation.

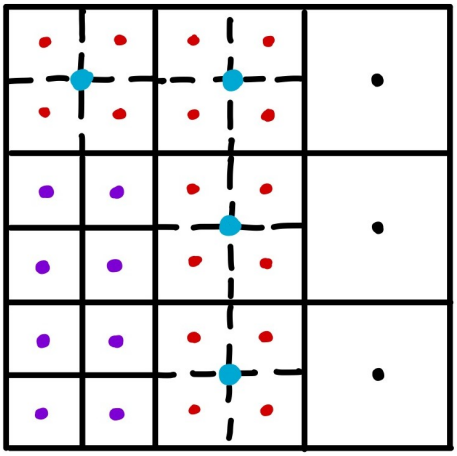


Figure 2.7.: Example of a coarse grid x^c (cells with black centres), a fine grid x^f (solid cells with purple centres, on the left), and an interface layer. The interface consists of blue coarse nodes $x^{c,I}$ and red fine nodes $x^{f,I}$. Each coarse and fine node contains populations f_i^c and f_i^f, respectively.

The following steps, taken directly from [27], summarize the algorithm in greater detail. Note, this quoted explanation is included to clarify the forthcoming discussion concerning the treatment of corners in the refinement interface. To the best of my knowledge, how to deal with corners has only been described in [69], with an inconsistency that violates the algorithms conservation properties.

1. Collision of all coarse nodes, $x^c + x^{c,I}$, and regular fine nodes x^f. Fine interface nodes $x^{f,I}$ do not take part in any collision step during this algorithm.
2. Coarse to fine coupling (termed *explosion*). Uniform redistribution of all coarse post-collision populations among fine interface nodes [8], following:

[8] Strictly speaking, it is only necessary to explode the populations pointing into the fine grid, however, for ease of implementation it is more convenient to explode all the populations. Populations not pointing into the fine grid play no part in the algorithm – they will neither stream into the fine domain nor be accounted for in the coalescence step.

$$\hat{f}_i^f(x^{f,I}, t) = \hat{f}_i^c(x^{c,I}, t) \tag{2.4.2}$$

3. Streaming of all coarse and fine nodes. Coarse populations reach $t^c = t + \Delta t^c$, fine populations reach $t^f = t + \frac{1}{2}\Delta t^c$.
4. Collision at regular fine nodes x^f. Since two lines of fine interface nodes are updated during explosion with coarse post-collision states, the corresponding states remain valid for two consecutive fine streaming steps.
5. Streaming of all fine nodes. Populations on the fine grid reach $t^f = t + \Delta t_c$.
6. Fine to coarse coupling (*coalescence*). Populations from fine grid interface nodes directed toward the coarse grid, $f_{i,coal}^{f,I}$, are averaged over each coarse cell and transferred to the corresponding coarse interface node:

$$f_{i,coal}^c(x^c, t) = \frac{1}{4}\sum_{x^{f,I}} f_{i,coal}^f(x^{f,I}, t) \tag{2.4.3}$$

7. Go to step 1 and repeat to the end of the simulation.

Special care must be taken to determine which populations are coalesced in step 6. The general rule is:

$f_{i,coal}^{f,I}$ *are* always *the populations pointing into the coarse grid that underwent at least one collision step in the fine grid.*

For a case such as the one shown in Figure 2.8, where the domain is assumed to extend to infinity, it is clear why the rule holds. During the coarse streaming step (step 3), only populations 3, 6, and 7 will stream into the coarse grid. Therefore, only these populations have to be coalesced from the fine grid. Furthermore, these populations clearly stream into the interface layer exclusively from the fine grid, satisfying the second condition. However, for a more general case where the interface layer has corners, the rule requires some clarification. Indeed, for such a case, an incorrect description of the algorithm has been reported in figure 1 of [69].

Consider the scenario sketched in Figure 2.9, where the grid interface (highlighted in orange) has a corner. The coalescence rule tells us that for the coarse interface cell marked in orange, we should coalesce populations 1, 5, and 8. However, at first glance, it seems that averaging population 8 is unnecessary. Examining Figure 2.10, we could conclude that after the coarse streaming (step 3), population 8 advects from the neighbouring coarse grid to our node of interest and so no communication with the fine grid is necessary. This is what figure 1 in [69] suggests. However, this violates the algorithm's conservation properties, as shown in Figure 2.11.

We begin by exploding population 8 (for clarity we only show the exploded state for the node adjacent to the orange cell). After steps 3 and 4, one of the exploded states (bottom left) has undergone an additional collision step compared to its brothers — this is the arrow drawn in green. Recall that the collision step redistributes mass among the populations in a given node in a conservative manner. After the second fine streaming, the four states that arrive at the interface nodes consist of 3 blue arrows (containing the original density of the coarse cell) and one green arrow, with a slightly different density due to the additional collision step. During the coalescence step, if we do not coalesce population 8 (i.e. we ignore the effect of the green arrow), the additional collision step will become a source of mass, and the algorithm will suffer from reduced accuracy and stability.

To conclude this discussion, Figure 2.12 shows the correct interpretation of the coalescence rule for a cornered grid interface.

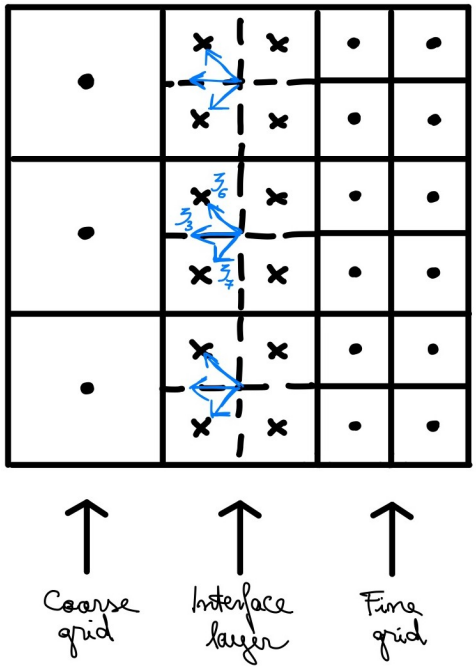


Figure 2.8.: Example of the coalesced populations (shown as blue arrows) at a grid interface layer which stretches to infinity in the vertical axis (or has periodic boundary conditions at the edges). Coalesced populations are $\xi_{3,6,7}$.

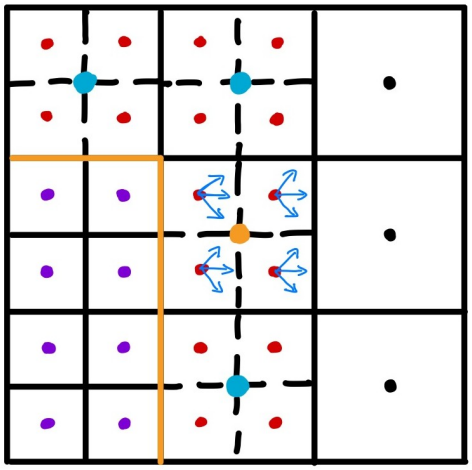


Figure 2.9.: Grid refinement interface with a corner.

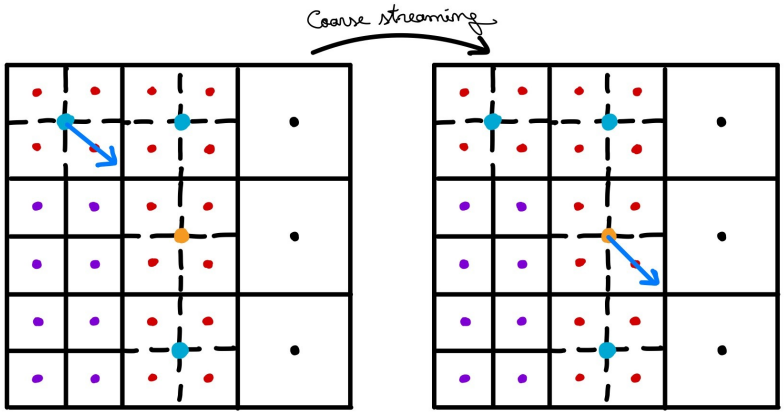


Figure 2.10.: Incorrect interpretation of the grid refinement algorithm. The coarse population ξ_8 (shown as a blue arrow) streams from the teal coarse interface node to the orange interface node with no interaction with the fine grid.

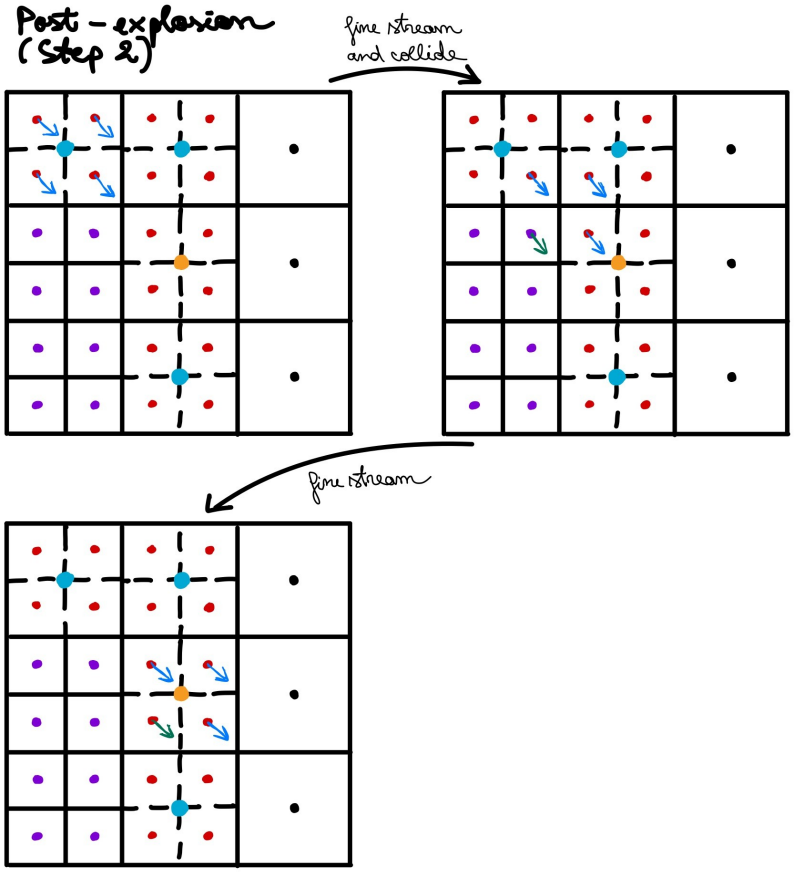


Figure 2.11.: Series of steps that results in the correct conservation rules.

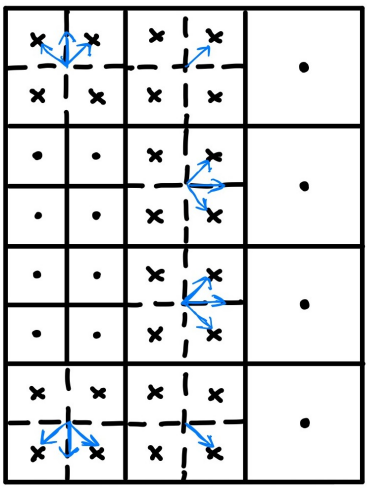


Figure 2.12.: Correct coalescence rule for a grid interface with two corners (highlighted in blue). Coalesced populations shown as blue arrows.

2.4.1. Multiple refinement levels

The above algorithm applies to a single embedded grid and requires an extension when multiple refinement levels are used, as in the case of the two refinement levels in Figure 3.5. For generality, I consider an arbitrary case with N_{levs} refinement levels (if the grid is uniform, $N_{levs} = 0$).

The only paper that describes how this can be done for Rohde et al.'s algorithm proposes a recursive formulation [70]. However, it is not possible to use this formulation for the current implementation as the compiler JAX does not support recursion. Therefore, I developed the iterative algorithm shown and explained in Appendix A, omitted here for brevity.

2.5. Cost function definition and computation

The cost function is the quantity of interest for which gradients will be obtained and which the optimizer seeks to minimize (or maximize). As stated in chapter 1, it is desirable to use non-dimensional QoIs to assess aerodynamic performance. Specifically, this work is limited to studying the surface forces around a given obstacle. Two flow cases are considered in this work, the steady flow past a NACA 0012 airfoil and the unsteady flow past a cylinder.

Each case considers its own cost function I. For the steady problem, I the lift to drag ratio:

$$I = \frac{L}{D}. \tag{2.5.1}$$

For the unsteady problem, I is the time averaged chord based drag coefficient:

$$I = C_{\overline{D}} = \frac{2\overline{D}}{\rho_0 u_0^2 c} \tag{2.5.2}$$

where

$$\overline{D} = \frac{1}{N - n_{start}} \sum_{n=n_{start}}^{N} D(t_n). \tag{2.5.3}$$

The subscript 0 indicates reference quantities, c is the chord and the temporal bounds n_{start} and N are selected by the user. It is expected that the unsteady problem converges to a periodic vortex shedding state. Therefore, $N - n_{start}$ should span an integer multiple of periods.

Both quantities are computed by a direct integration of the surface forces using the momentum exchange algorithm (MEA) with a correction to account for the porous boundary treatment [12], [30]. The MEA is also applicable to higher order boundary conditions (with a very slight modification, see [12]), so its integration into an adjoint pipeline allows for extensibility of the solver.

Assuming that the inflow is horizontal, the object is stationary, and the boundary treatment is PBB, the aerodynamic forces can be computed from:

$$\boldsymbol{\mathcal{F}}(t_n) = \begin{bmatrix} L \\ D \end{bmatrix} = \sum_{\tilde{\mathbf{x}}_{k_j} \in X_b} \left(2\hat{f}_j(\mathbf{x}_k, t_n) \cdot \boldsymbol{\xi}_j + \rho(\mathbf{x}_k)(\mathbf{u}_{\text{pre-collision}}(\mathbf{x}_k) - \mathbf{u}_{\text{post-collision}}(\mathbf{x}_k)) \right). \quad (2.5.4)$$

The first term represents the regular MEA, where forces equal the sum of the momentum exchanged at each boundary node in one time step due to the reflection of the incoming population f_j^*. The second term accounts for the added porosity by considering the momentum extracted from the flow, as per (2.3.6).

At first glance, (2.5.4) has units of momentum density. This is because I assume lattice units, where $\Delta x = \Delta t = 1$, so omit the factors Δx^2 and $1/\Delta t$ that would make the equations dimensionally consistent. Given that both cost functions are non-dimensional, it is not necessary to specify a conversion to physical units.

Chapter 3: The Discrete Adjoint Lattice Boltzmann Method Applied to Shape Optimization

This chapter shows how to obtain the cost function gradient using an adjoint LBM solver and how to integrate it into a shape optimization pipeline. Sections 3.1 to 3.3 show how the adjoint LBM is derived, how to map forward boundary conditions to their adjoint counterparts, how to particularize some of the case-specific terms, and how to incorporate the grid refinement scheme. This is followed by a description of the checkpointing routine employed to reduce the storage cost associated with the unsteady adjoint (section 3.4). Section 3.5 presents how the shape is parametrized and how to incorporate this into the adjoint pipeline. The chapter concludes with a presentation of the test cases used for gradient computation and optimization (section 3.6).

3.1. Adjoint derivation

The adjoint derivation closely follows Tekitek et al.'s pioneering work [1], beginning with the most general case—involving a time dependent cost function—followed by the reduction to the steady problem.

The unsteady shape optimisation problem consists in minimising a scalar cost function I that is the time average of a functional $\mathcal{I}^n$:

$$I = \frac{1}{N}\sum_{n=0}^{N} \mathcal{I}^n(\boldsymbol{\mu}, \mathbf{F}^n(\boldsymbol{\mu})). \tag{3.1.1}$$

This is in line with the definitions presented in section 2.5 and can be adapted for different temporal bounds. $\boldsymbol{\mu} \in \mathbb{R}^m$ is the vector of optimization parameters and $\mathbf{F}^n(\boldsymbol{\mu})$ the flattened vector of state variables given by the generic and fully discrete residual equation (also referred to as the forward problem):

$$\mathbf{R}^n(\boldsymbol{\mu}, \mathbf{F}^n(\boldsymbol{\mu})) = 0. \tag{3.1.2}$$

For the LBM we can specify the residual by rearranging (2.1.8):

$$\mathbf{R}^n(\boldsymbol{\mu}, \mathbf{F}(\boldsymbol{\mu})) = \begin{cases} \mathbf{F}^0 - \mathbf{F}^{init} = 0 \\ \mathbf{F}^{n+1} - \Phi^n(\mathbf{F}^n) = \mathbf{F}^{n+1} - S \cdot \mathscr{C}(\mathbf{F}^n) = 0, \quad n = 0, 1, ..., N-1 \end{cases} \tag{3.1.3}$$

where the boundary conditions are assumed to be incorporated in S.

The optimization is carried out using an iterative algorithm that requires access to the cost function gradient:

$$\frac{dI}{d\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=0}^{N}\left[\frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} + \frac{\partial \mathcal{I}^n}{\partial \mathbf{F^n}}\frac{d\mathbf{F^n}}{d\boldsymbol{\mu}}\right]. \tag{3.1.4}$$

It is possible to directly evaluate this quantity by deriving explicit expressions for $\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}}$; however, the computational cost of doing so scales with the number of optimization parameters. For a problem with m parameters, obtaining the gradients with such a forward[1] pass would have a cost similar to m evaluations of the forward code. This may be prohibitive for the typically large dimensional aerodynamic shape optimization problems.

[1] I chose this term because I am describing forward-mode automatic differentiation.

The adjoint approach allows evaluating the gradient at a cost that is independent of the number of optimization parameters. To do so, we define the Lagrangian function:

$$J = I + \sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\mathbf{R}^n \tag{3.1.5}$$

where the line vector $\mathbf{F}^{*,n+1}$, called the adjoint variable, is free to choose given (3.1.2). For our problem

$$J = \frac{1}{N}\sum_{n=0}^{N} \mathcal{I}^n(\boldsymbol{\mu}, \mathbf{F}^n(\boldsymbol{\mu})) + \sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\left(\mathbf{F}^{n+1} - \Phi^n(\mathbf{F}^n)\right) \tag{3.1.6}$$

and $\mathbf{F}^{*,n} = (f_i^*(\mathbf{x}_k, t_n))_{(0\leq i\leq 8, 0\leq k\leq N_x N_y - 1)}$.

Recalling that the total derivative of the residual equations is 0 because they hold for any set of parameters $\boldsymbol{\mu}$, we obtain

$$\frac{dJ}{d\boldsymbol{\mu}} = \frac{dI}{d\boldsymbol{\mu}}.$$

Writing out $\frac{dJ}{d\boldsymbol{\mu}}$ yields

$$\frac{dJ}{d\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=0}^{N} \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} + \frac{1}{N}\sum_{n=0}^{N} \frac{\partial \mathcal{I}^n}{\partial \mathbf{F^n}}\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}} + \sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\left(\frac{d\mathbf{F}^{n+1}}{d\boldsymbol{\mu}} - \frac{\partial \Phi^n}{\partial \mathbf{F}^n}\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}} - \frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}\right). \tag{3.1.7}$$

Now we employ the following identity:

$$\sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\left(\frac{d\mathbf{F}^{n+1}}{d\boldsymbol{\mu}} - \frac{\partial \Phi^n}{\partial \mathbf{F}^n}\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}}\right) = \tag{3.1.8}$$

$$\left(\mathbf{F}^{*,N}\frac{d\mathbf{F}^N}{d\boldsymbol{\mu}} - \mathbf{F}^{*,1}\frac{\partial \Phi^0}{\partial \mathbf{F}^0}\frac{d\mathbf{F}^0}{d\boldsymbol{\mu}}\right) + \sum_{n=1}^{N-1}\left[\left(\mathbf{F}^{*,n} - \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \mathbf{F}^n}\right)\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}}\right]. \tag{3.1.9}$$

Substituting into (3.1.7) yields:

$$\begin{aligned}\frac{dJ}{d\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=0}^{N} \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} + \frac{1}{N}\sum_{n=0}^{N} \frac{\partial \mathcal{I}^n}{\partial \mathbf{F^n}}\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}} + \left(\mathbf{F}^{*,N}\frac{d\mathbf{F}^N}{d\boldsymbol{\mu}} - \mathbf{F}^{*,1}\frac{\partial \Phi^0}{\partial \mathbf{F}^0}\frac{d\mathbf{F}^0}{d\boldsymbol{\mu}}\right) \\ + \sum_{n=1}^{N-1}\left[\left(\mathbf{F}^{*,n} - \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \mathbf{F}^n}\right)\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}}\right] - \sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}.\end{aligned} \tag{3.1.10}$$

The initial condition is independent of $\boldsymbol{\mu}$ so all terms containing $\frac{d\mathbf{F}^0}{d\boldsymbol{\mu}}$ vanish. To minimize the computational cost, we seek to eliminate all terms involving $\frac{d\mathbf{F}^n}{d\boldsymbol{\mu}}$. This yields the following equations. Firstly, the adjoint terminal condition:

$$\mathbf{F}^{*,N} = -\frac{1}{N}\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^n}. \tag{3.1.11}$$

Secondly, the adjoint evolution equation:

$$\mathbf{F}^{*,n} = \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \mathbf{F}^n} - \frac{1}{N}\frac{\partial \mathcal{I}^n}{\partial \mathbf{F^n}} \tag{3.1.12}$$

And thirdly, the cost function gradient (which we were after all along):

$$\frac{dJ}{d\boldsymbol{\mu}} = \frac{dI}{d\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=0}^{N} \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \sum_{n=0}^{N-1} \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}} \tag{3.1.13}$$

$$= \frac{1}{N}\left(\frac{\partial \mathcal{I}^N}{\partial \boldsymbol{\mu}} + \sum_{n=0}^{N-1}\left[\frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n+1}\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}\right]\right) \tag{3.1.14}$$

Since $\Phi^n(\mathbf{F}^n) = S \cdot \mathscr{C}(\mathbf{F}^n)$ we have:

$$\frac{\partial \Phi^n}{\partial \mathbf{F}^n} = S \cdot \frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}. \tag{3.1.15}$$

Substituting this result into (3.1.12) and transposing the equation so that the adjoint becomes a column vector yields the iterative procedure:

$$\begin{cases} \mathbf{F}^{*,N} = -\frac{1}{N}\left(\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^n}\right)^\intercal \\ \mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot S^\intercal \cdot \mathbf{F}^{*,n+1} - \frac{1}{N}\left(\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}\right)^\intercal, n = N-1, ..., 0 \end{cases} \tag{3.1.16}$$

Theoretically, since the adjoint variable has the same dimensions as the forward state and the iteration runs for the same number of time steps, obtaining the adjoint has a cost on the order of one forward problem solve. The partial derivatives $\frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}}$ and $\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$ can be determined by hand and evaluated inexpensively (see section 3.2). However, to compute these derivatives, the collision Jacobian, and the source term, the forward state vector must be loaded from disk at every timestep, representing a significant memory and computational cost. A method to allay this is presented in section 3.4.

(3.1.16) describes an iterative process that is very similar to the forward LBM, but, whereas the LBM follows a collide-then-stream pipeline, the adjoint LBM is stream-then-collide. The nature of the adjoint streaming and collision is also akin to the forward problem. As shown by Luo et al. [40] and Verganult and Sagaut [41], for a node in the bulk sufficiently far removed from any boundaries, the transposed streaming operation can be written as backwards-in-time streaming:

$$f_i^{*,s}(\mathbf{x}_k - \boldsymbol{\xi}_i, t_{n+1}) = f_i^*(\mathbf{x}_k, t_{n+1}) \tag{3.1.17}$$

where the superscript s denotes post-streaming adjoint populations, analogously to the post collision populations $\hat{f}_i$ in the forward problem. Furthermore, if the collision operator is local[2], the matrix $\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}$ and its transpose will be block diagonal. Therefore, the adjoint collision can be written entirely locally as:

$$f_i^*(\mathbf{x}_k, t_n) = \sum_{j=0}^{q-1} f_j^{*,s}(\mathbf{x}_k, t_{n+1}) \frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n) - \frac{1}{N} \frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)} \tag{3.1.18}$$

The local derivative of the post collision distribution functions, $\frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n)$, is particularized for RR in Appendix B. The local source term generally takes the form given in (3.1.19), further specified for the cost functions from section 2.5 in section 3.2.

$$\frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)} = \begin{cases} \frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)}, \text{if computing } \mathcal{I} \text{ involves the node } x_k \text{ at time } t_n \\ 0, \text{otherwise} \end{cases} \tag{3.1.19}$$

Representing (3.1.16) using equations (3.1.18) and (3.1.19) shows that implementing the adjoint LBM can be done with the same programming and parallelization techniques as the forward code. From a performance standpoint, the only (albeit very) significant difference between the solvers is the adjoint's need to load the state $\mathbf{F}^n$ from disk at every timestep. Another important consideration is the implementation of adjoint boundary conditions, which is generally not as straightforward as for the LBM, especially if the rules contain a degree of non-locality (meaning they use information from neighbouring nodes and/or more than one post-collision distribution function[3]). This is further discussed in section 3.2.

[2] This is the case for most collision operators with the notable exception of the hybridized Recursive Regularization model, which has recently been the focus of several works due to its excellent stability.

[3] Of the boundary conditions used in this work, only the extrapolation boundary condition has this property. However, some examples of non-local boundary conditions worth mentioning are interpolated bounce back methods, high order volumetric schemes and non-equilibrium bounce back.

The following algorithms show how the forward and adjoint solvers must be synchronized to evaluate (3.1.14). Note, $\mathbf{F}^*$ is now assumed to be a column vector.

Algorithm 1 Forward LBM simulation

Initialize the distribution functions. $\mathbf{F}^0 = \mathbf{F}^{init}$
Initialize the cost function. $I = 0$
for $n = 0$ to $n = N - 1$ **do**
    $\mathbf{F}^{n+1} = S \cdot \mathscr{C}(\mathbf{F}^n)$
    $I \leftarrow I + \mathcal{I}(\mathbf{F}^{n+1})$
    Save $\mathbf{F}^{n+1}$ to disk.
end for
$I \leftarrow \frac{I}{N}$
Outputs: I, states $\mathbf{F}^1, ..., \mathbf{F}^N$ saved to disk.

Algorithm 2 Adjoint LBM run

Load the state $\mathbf{F}^N$
$\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}}$
Initialize the adjoint vector. $\mathbf{F}^{*,N} = -\frac{1}{N}\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^N}$
for $n = N - 1$ to $n = 0$ **do**
    Load the state $\mathbf{F}^n$
    $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{dI}{d\boldsymbol{\mu}} + \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n+1\intercal}\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$
    $\mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot S^\intercal \cdot \mathbf{F}^{*,n+1} - \frac{1}{N}\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}$
end for
$\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{1}{N}\frac{dI}{d\boldsymbol{\mu}}$
Outputs: $\frac{dI}{d\boldsymbol{\mu}}$

Tekitek et al.'s derivation is not the only way to obtain a discrete-adjoint LBM. The few existing papers on LBM gradient-based (aerodynamic) shape optimization use the derivation proposed by Vergnault and Sagaut [41], which directly arrives at a collide-then-stream adjoint equation, written in local form. This is an attractive property because one can immediately see that the backward and forward LBMs have the same form, and the local equation of the adjoint collision and source terms, akin to (3.1.18), is immediately obtained—offering a simpler path towards implementation. As shown in Appendix C, under certain conditions this derivation yields a method exactly equivalent to Tekitek et al.'s—both in the final gradient and in the manner of implementation.

However, Vergnault and Sagaut's approach suffers from two significant issues. Firstly, the adjoint is derived focusing on a node in the bulk so the method lacks a general treatment of adjoint boundary conditions—a deficiency noted by the authors themselves. This is further discussed in section 3.2 and Appendix C. Secondly (and more fundamentally), they do not derive, or in any way specify, the adjoint terminal condition. As shown in (3.1.16), this is generally non-zero and has a significant effect on $\frac{dI}{d\boldsymbol{\mu}}$ since the gradient is accumulated from the first adjoint timestep. Thus, the derivation is ultimately incomplete—to obtain accurate gradients the user has to arrive at the terminal condition themselves. Moreover, Vergnault and Sagaut's gradient equation only holds for the case where the optimization parameters affect the collision operator. In the alternative case (which would occur for interpolated bounce back, see section 4.3), a different equation must be used, as shown in Appendix C. Given these deficiencies, I suggest using Tekitek et al.'s approach as the theoretical basis for the discrete-adjoint LBM.

3.1.1. Developed flow state gradients

Since simulations are initialized to a macroscopic zero-velocity condition, there is an initial transient period where the flow adjusts to the obstacle before reaching its fully developed state. For the unsteady flow past a cylinder, the flow is expected to eventually exhibit periodic vortex shedding. We are interested in the average drag (and its gradient to the shape) over a set of periods and wish to ignore the initial transient for the gradient computation. Achieving this also means a significant reduction of the storage cost associated with the adjoint.

The only modification necessary to the adjoint solver is that it no longer runs until $n = 0$. Algorithm 3 and Algorithm 4 show the modified forward and adjoint pipeline. The truncation of the adjoint solver introduces an error in the gradients which reduces if a longer time integration is used. This is examined in chapter 4.

Algorithm 3 Forward LBM simulation

Initialize the distribution functions. $\mathbf{F}^0 = \mathbf{F}^{init}$
Initialize the cost function. $I = 0$
Define the first time for which the cost function will be calculated: n_{start}
for $n = 0$ to $n = N - 1$ **do**
    $\mathbf{F}^{n+1} = S \cdot \mathscr{C}(\mathbf{F}^n)$
    if $n + 1 \geq n_{start}$ **then**
        $I \leftarrow I + \mathcal{I}(\mathbf{F}^{n+1})$
        Save $\mathbf{F}^{n+1}$ to disk.
    end if
end for
$I \leftarrow \frac{I}{N - n_{start}}$
Outputs: I, states $\mathbf{F}^{n_{start}}, ..., \mathbf{F}^N$ saved to disk.

Algorithm 4 Truncated Adjoint LBM run

Load the state $\mathbf{F}^N$
$\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}}$
Initialize the adjoint vector. $\mathbf{F}^{*,N} = -\frac{1}{N}\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^N}$
for $n = N - 1$ to $n = n_{start} - 1$ **do**
    Load the state $\mathbf{F}^n$
    $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{dI}{d\boldsymbol{\mu}} + \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n+1^\intercal}\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$
    $\mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot S^\intercal \cdot \mathbf{F}^{*,n+1} - \frac{1}{N}\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}$
end for
$\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{1}{N - n_{start}}\frac{dI}{d\boldsymbol{\mu}}$
Outputs: $\frac{dI}{d\boldsymbol{\mu}}$

3.1.2. Steady Gradients

In case the forward flow field reaches a stationary state, it is only necessary to store the converged field $\mathbf{F}^N$ to run the adjoint solver [1], [41], [42]. This represents a huge reduction of the computational and memory cost of obtaining gradients. For this case, the forward and adjoint solvers interface as:

Algorithm 5 Steady forward LBM simulation

1: Initialize the distribution functions. $\mathbf{F}^0 = \mathbf{F}^{init}$
2: **for** $n = 0$ to convergence **do**
3: $\quad \mathbf{F}^{n+1} = S \cdot \mathscr{C}(\mathbf{F}^n)$
4: **end for**
5: $I \leftarrow I(\mathbf{F^N})$
6: Outputs: I, state $\mathbf{F}^N$ saved to disk.

Algorithm 6 Steady adjoint LBM run

1: Load the state $\mathbf{F}^N$
2: Precompute $\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^N}$, $\left(\frac{\partial \mathscr{C}(\mathbf{F}^N)}{\partial F^N}\right)^\intercal$
3: Initialize the adjoint vector. $\mathbf{F}^{*,N} = -\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^N}$
4: **for** $n = N-1$ to convergence ($n = n_{conv}$) **do**
5: $\quad \mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^N)}{\partial F^N}\right)^\intercal \cdot S^\intercal \cdot \mathbf{F}^{*,n+1} - \frac{1}{N}\frac{\partial \mathcal{I}^N}{\partial \mathbf{F}^N}$
6: **end for**
7: $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{\partial \mathcal{I}^N}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n_{conv}{}^\intercal}\frac{\partial \Phi^N}{\partial \boldsymbol{\mu}}$
8: Outputs: $\frac{dI}{d\boldsymbol{\mu}}$

Note, the cost function is not a time average, hence the absence of the $1/N$ term. The convergence criterion used for the LBM and adjoint solvers is

$$\frac{||\mathbf{F}^{n+1} - \mathbf{F}^n||_1}{||\mathbf{F}^n||_1} < 10^{-15}. \tag{3.1.20}$$

3.2. Adjoint boundary conditions, source terms, and miscellaneous derivatives

Tekitek et al.'s formulation very straightforwardly leads to adjoint boundary conditions. The only necessary assumption is that S is a matrix with entries that are independent of the distribution functions (i.e. S is not a function of $\mathbf{F}$). Then:

> Assuming that the forward boundary conditions can be incorporated into the streaming matrix S in a manner independent of $\mathbf{F}$, the adjoint boundary conditions are trivially implemented by the operation $S^\intercal$.

For this reason, I recommend implementing the forward and adjoint collision steps using their local representations Equation 2.2.7 and (3.1.18), but keeping the streaming steps as explicit matrix-vector products. In this way, if S is implemented correctly, the adjoint boundary conditions are a foolproof step. The matrices S and $S^\intercal$ are very sparse with approximately $9N_xN_y$ non-zero elements, depending on the exact form of the boundary conditions. Therefore, they can be stored and manipulated efficiently using a standard sparse matrix representation (CSR, COO). Furthermore, if one really wants to implement the adjoint using local rules, a program can easily be written to generate them from the matrix by looking for non-zero entries in the rows of $S^\intercal$.

The extension to affine or non-linear boundary conditions (typically imposed at the inlet or outlet as, for example, anti-bounce back pressure boundary condition or non-equilibrium bounce back [12]) is straightforward. The forward LBM step must be modified to:

$$\mathbf{F}^{n+1} - \Phi^n(\mathbf{F}^n) = \mathbf{F}^{n+1} - \mathcal{B}(S \cdot \mathscr{C}(\mathbf{F}^n)) = 0, \quad n = 0, 1, ..., N-1 \tag{3.2.1}$$

with $\mathcal{B}$ a general non-linear operator [40]. The adjoint step will then be:

$$\mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot S^\intercal \cdot \left(\frac{\partial \mathcal{B}(\mathbf{F}^{n,s})}{\partial F^{n,s}}\right)^\intercal \mathbf{F}^{*,n+1} - \frac{1}{N}\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}, n = N-1, ..., 1 \tag{3.2.2}$$

where $F^{n,s} = S \cdot \mathscr{C}(\mathbf{F}^n)$. The terminal condition remains unchanged. The only added complexity is taking the derivative $\frac{\partial \mathcal{B}(\mathbf{F}^{n,s})}{\partial \mathbf{F}^{n,s}}$.

Recalling section 2.3, I employ standard halfway bounce back on solid walls not subject to optimization, bounce back with a (constant) momentum correction on inlets, PBB (which is just bounce back with a modified post-collision distribution function) on solid walls subject to optimization, free-slip on northern and southern boundaries for the airfoil case, and extrapolation of all populations at the outlet. Halfway bounce back, PBB, and free-slip all share the convenient property that they are simply index permutations of the post-collision distribution functions. In other words, they may be written as

$$f_\alpha(x_b, t_{n+1}) = \hat{f}_\beta(x_b, t_n)$$

translating to a single entry per row and column in S. Therefore, the transposition merely swaps the indices, such that the adjoint boundary conditions are

$$f^{*,s}_\beta(x_b, t_{n+1}) = f^*_\alpha(x_b, t_{n+1}) \tag{3.2.3}$$

This is the previously identified result that bounce back results in adjoint bounce back [40]–[42] — extended to the free-slip and PBB boundary conditions. In the last case, the effect of modifying the collision operator is incorporated into $\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}$. For bounce back with a constant (i.e. independent of the distribution functions) momentum correction, the adjoint boundary condition remains bounce back, with no added terms. To see why, define

$$\mathcal{B}(\mathbf{F}) = I \cdot \mathbf{F} + M$$

with I the identity matrix and M the vector containing the momentum correction. If M does not depend on $\mathbf{F}$,

$$\frac{\partial \mathcal{B}(\mathbf{F}^{n,s})}{\partial \mathbf{F}^{n,s}} = I.$$

In the code, (3.2.3) is only exploited for PBB. The domain boundary conditions are represented via an explicit matrix-vector product, allowing for a robust treatment of the non-local extrapolation boundary condition. For this rule, the rows in S corresponding to the outlet nodes contain two entries. Therefore, after transposing, nodes in the vicinity of the outlet (which are not the outlet nodes) contain source terms originating from the forward boundary condition. An example of this is provided in Appendix D.

As discussed in the previous section, the discrete-adjoint LBM has also been derived by Vergnault and Sagaut. Their derivation considers individual nodes and populations. Thus, obtaining (linear) non-local adjoint boundary conditions is a thorny bookkeeping exercise, since one must explicitly identify which boundary-adjacent nodes receive source terms and from where they arise. This complexity has led to incorrect derivations of interpolated bounce back adjoint conditions in [42], as noted by [47], or otherwise very case-specific formulas that are considerably harder to implement than their forward counterparts (for an example, equations (19)-(22) in [47]).

3.2.1. Source terms

As opposed to the forward LBM, a source term appears in the adjoint collision step. This term, $\frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)}$, can be specified for the two considered cost functions by taking the derivative of the vector force function (2.5.4):

$$\frac{\partial \boldsymbol{\mathcal{F}}(t_n)}{\partial f_i(\mathbf{x}_k, t_n)} = 2\frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n)\boldsymbol{\xi}_j + \boldsymbol{\xi}_i - \frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n)\boldsymbol{\xi}_i = (2\boldsymbol{\xi}_j - \boldsymbol{\xi}_i)\frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n) + \boldsymbol{\xi}_i \tag{3.2.4}$$

where the derivative of the post-collision distribution functions incorporate the effect of the porosity as shown in Appendix B.

For the unsteady optimization case, we obtain the derivative of the drag coefficient by taking the x component of (3.2.4):

$$\begin{cases} \frac{\partial \mathcal{I}(t_n)}{\partial f_i(\mathbf{x}_k, t_n)} = \frac{2}{\rho_0 u_0^2 c}\left[(2\xi_{x_j} - \xi_{x_i})\frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n) + \xi_{x_i}\right], \tilde{\mathbf{x}}_{k_j} \in X_b \\ 0, \text{otherwise} \end{cases} \tag{3.2.5}$$

The source term for the steady case follows by applying the quotient rule (note the time dependence is gone):

$$\begin{cases} \frac{\partial \mathcal{I}}{\partial f_i(\mathbf{x}_k)} = \frac{\partial\left(\frac{L}{D}(\mathbf{x}_k)\right)}{\partial f_i(\mathbf{x}_k)} = \frac{\left(\frac{\partial \mathcal{F}_y}{\partial f_i}\right)\mathcal{F}_x - \left(\frac{\partial \mathcal{F}_x}{\partial f_i}\right)\mathcal{F}_y}{\mathcal{F}_x^2}, \tilde{\mathbf{x}}_{k_j} \in X_b \\ 0, \text{otherwise} \end{cases} \tag{3.2.6}$$

3.2.2. Partial derivatives to the optimization parameters

Recalling the definition of PBB, (2.3.9), we can write out the terms $\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$ and $\frac{\partial \mathcal{I}^N}{\partial \boldsymbol{\mu}}$.

Firstly,

$$\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}} = S\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial d} \cdot \frac{\partial d}{\partial \boldsymbol{\mu}} \tag{3.2.7}$$

where $\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial \varepsilon}$ can be found in Appendix B and

$$\frac{\partial \varepsilon}{\partial d} = \frac{2ad}{\sigma_\varepsilon^2 b}\exp\left[-\frac{d^2}{\sigma_\varepsilon^2 b}\right]. \tag{3.2.8}$$

$\frac{\partial d}{\partial \boldsymbol{\mu}}$ is detailed in subsection 3.5.2.

$\frac{\partial \mathcal{I}^N}{\partial \boldsymbol{\mu}}$ is specific to each cost function. The derivative of the vector force function is:

$$\frac{\partial \boldsymbol{\mathcal{F}}(t_n)}{\partial \boldsymbol{\mu}} = \sum_{\tilde{\mathbf{x}}_{k_j} \in X_b}\left(2\frac{\partial \hat{f}_j(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}} \cdot \boldsymbol{\xi}_j - \sum_{j=0}^{8}\boldsymbol{\xi}_j\frac{\partial \hat{f}_j(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}\right). \tag{3.2.9}$$

where $\frac{\partial \hat{f}_j(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}$ is the local representation of $\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial d} \cdot \frac{\partial d}{\partial \boldsymbol{\mu}}$. For the drag coefficient, take the x component of (3.2.9):

$$\frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} = \frac{2}{\rho_0 u_0^2 c}\frac{\partial \mathcal{F}_x(t_n)}{\partial \boldsymbol{\mu}} \tag{3.2.10}$$

For the L/D ratio, apply the quotient rule:

$$\frac{\partial \mathcal{I}}{\partial \boldsymbol{\mu}} = \frac{\partial\left(\frac{L}{D}\right)}{\partial \boldsymbol{\mu}} = \frac{\left(\frac{\partial \mathcal{F}_y}{\partial \boldsymbol{\mu}}\right)\mathcal{F}_x - \left(\frac{\partial \mathcal{F}_x}{\partial \boldsymbol{\mu}}\right)\mathcal{F}_y}{\mathcal{F}_x^2} \tag{3.2.11}$$

3.2.3. Adjoint sponge

The forward sponge modifies the density that goes into the collision operator. Therefore, its effect on the adjoint variable is incorporated into the term $\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}$. Appendix B gives the explicit formulas for this term, taking into account the pressure relaxation sponge and the velocity relaxation for PBB.

3.3. Adjoint grid refinement

Following the results of [42], [47]—obtained for an interpolation based grid refinement scheme—it was assumed that the adjoint grid refinement strategy is identical to the forward grid refinement, except that the adjoint coalesced populations follow $f^*_{i,coal} = f_{i^-,coal}$ due to the transposed streaming. This assumption is likely justified as the coalescence and explosion steps consist of purely local information, both spatially and in velocity space.

Algorithm 10 is based on a collide-then-stream implementation of the forward LBM. Since the adjoint iteration is nominally stream-then-collide it is handy to make a modification to the adjoint terminal condition that results in a collide-then-stream iteration, allowing us to use Algorithm 10 with minimal modifications. This is straightforwardly achieved by starting the adjoint loop with the post-streaming populations at time $N+1$ set to $\mathbf{F}^{*,\mathsf{s}}(N+1) = 0$. The adjoint loop then becomes:

Algorithm 7 Collide-then-stream adjoint LBM run

1: Initialize the adjoint vector. $\mathbf{F}^{*,s,N+1} = S^\intercal \mathbf{F}^{*,N+1} = 0$
2: $\frac{dI}{d\boldsymbol{\mu}} = 0$
3: Store (in RAM) $\mathbf{F}^{*,N+1}$ (it is needed for the gradient computation in the next iteration)
4: **for** $n = N$ to $n = 0$ **do**
5: Load the state $\mathbf{F}^n$
6: Collide: $\mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot \mathbf{F}^{*,s,n+1} - \frac{1}{N}\left(\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}\right)^\intercal$
7: Store $\mathbf{F}^{*,n}$
8: $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{dI}{d\boldsymbol{\mu}} + \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n+1^\intercal} \frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$
9: Stream: $\mathbf{F}^{*,s,n} = S^\intercal \mathbf{F}^{*,n}$
10: **end for**
11: $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{1}{N}\frac{dI}{d\boldsymbol{\mu}}$
12: Outputs: $\frac{dI}{d\boldsymbol{\mu}}$

For multiple refinement levels, the only addition to Algorithm 10 is that the gradient should be updated after every collision on a boundary node.

3.4. Checkpointing

As shown by Algorithm 2 or Algorithm 4, obtaining gradients using the unsteady adjoint formulation requires loading (and thus storing) the state vector $\mathbf{F}^n$ at every timestep. Given a simulation on a fine mesh, the memory storage cost may become prohibitive and the computational cost of reading the stored file from disk will greatly degrade the performance of the adjoint solver [37], [42], [47], [50].

To ameliorate this situation, I have implemented a simple checkpointing routine that greatly reduces the memory cost while exactly retaining the accuracy of the gradients [53]. The idea is to store only a certain number of timesteps—called checkpoints—and, while evaluating the adjoint, *recompute* the necessary forward states from each checkpoint and store them in RAM. In this way, the disk memory requirement is reduced as is the cost of loading the state (given that accessing RAM is much faster than disk). The downside is it comes at the added cost of recomputing the forward solution (though, if the gradients are sought for the developed flow field only a subset of timesteps must be recomputed).

The following algorithms show the implementation of the forward and adjoint loops with checkpointing. The algorithms are shown for gradients accounting for the transient flow

evolution period but the extension to the developed state is simple. For simplicity I follow the initialization convention $\mathbf{F}^{*,N+1} = 0$

Algorithm 8 Forward LBM simulation (storing checkpoints)

Initialize the distribution functions. $\mathbf{F}^0 = \mathbf{F}^{init}$
Initialize the cost function. $I = 0$
Define the checkpointing interval and the number of checkpoints: Δ_{check}, n_{check}. The first checkpoint is the first time step.
$i \leftarrow 0$
for $n = 0$ to $n = N - 1$ **do**
    $\mathbf{F}^{n+1} = S \cdot \mathscr{C}(\mathbf{F}^n)$
    $I \leftarrow I + \mathcal{I}(\mathbf{F}^{n+1})$
    if $n + 1 = 1 + i\Delta_{check}$ **and** $i < n_{check}$ **then**
        Save $\mathbf{F}^{n+1}$ to disk.
        $i \leftarrow i + 1$
    end if
end for
$I \leftarrow \frac{I}{N}$
Outputs: I, states $\mathbf{F}^1, \mathbf{F}^{1+\Delta_{check}}, ..., \mathbf{F}^{1+(n_{check}-1)\Delta_{check}}$ saved to disk.

Algorithm 9 Adjoint LBM with checkpointing

$\frac{dI}{d\boldsymbol{\mu}} = 0$
Initialize the adjoint vector. $\mathbf{F}^{*,s,N+1} = S^\intercal \mathbf{F}^{*,N+1} = 0$
for $i = n_{check-1}$ to $i = 0$ **do**
    Load the last available checkpoint $\mathbf{F}^{1+i\Delta_{check}}$.
    Store the checkpoint in RAM by appending it to an array `States`.
    for $n = 1 + i\Delta_{check}$, to $n = \min(N - 1, (i + 1)\Delta_{check} - 1)$ **do**
        Compute $\mathbf{F}^{n+1}$ using the LBM iteration.
        Append $\mathbf{F}^{n+1}$ to `States`
    end for
    for $n = \min(N, (i + 1)\Delta_{check})$ to $n = \max(i\Delta_{check}, 0)$ **do**
        Load the state $\mathbf{F}^n$ from `States`
        Collide: $\mathbf{F}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal \cdot \mathbf{F}^{*,s,n+1} - \frac{1}{N}\left(\frac{\partial \mathcal{I}^n}{\partial \mathbf{F}^n}\right)^\intercal$
        $\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{dI}{d\boldsymbol{\mu}} + \frac{\partial \mathcal{I}^n}{\partial \boldsymbol{\mu}} - \mathbf{F}^{*,n+1^\intercal} \frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$
        Stream: $\mathbf{F}^{*,s,n} = S^\intercal \mathbf{F}^{*,n}$
    end for
end for
$\frac{dI}{d\boldsymbol{\mu}} \leftarrow \frac{1}{N}\frac{dI}{d\boldsymbol{\mu}}$
Outputs: $\frac{dI}{d\boldsymbol{\mu}}$

Given this strategy, the memory cost reduces from N to n_{check} timesteps. Recomputing a forward state is much cheaper than loading a state from disk, so a minimum number of checkpoints should be sought. An approximate (lower bound) formula for this minimum, assuming a D2Q9 LBM with M total grid points, evolved with double precision, and a computer RAM capacity C_{RAM} (in bytes) is

$$0 \leq C_{RAM} - \text{LBM memory requirement} \times \Delta_{check} = C_{RAM} - 72M\Delta_{check}. \tag{3.4.1}$$

With optimal programming, the computational cost of obtaining gradient is twice the cost of evolving the forward LBM (due to the recomputation) plus the cost of the adjoint (which should be the same or less than the LBM, depending on whether the gradient includes the transient period) plus overhead from reading the checkpoints.

This checkpointing routine is very simple and assumes the number of timesteps is known a priori. Dynamic checkpointing routines have been developed if this assumption is not met [53]. Although it is more expensive than the time-averaging approach of Cheylan et al. [42] it guarantees exact gradients, speeding up the optimization. Furthermore, for a field with large variations about the mean, such as the vortex shedding cylinder, linearization about the mean may be excessively inaccurate. It was attempted to obtain gradients with this approach but the adjoint solver did not converge, likely due to the strong unsteady nonlinearities.

3.5. Shape parametrization with Free-Form Deformation

The only remaining ingredient left to specify for the adjoint solver (and optimization pipeline) is the optimization parameters themselves. These are a set of variables that govern the mathematical representation of the object. The optimization algorithm then looks for the parameter values that minimize the chosen objective function, determining the optimal shape.

A wide variety of shape parametrization methods have been studied for aerodynamic shape optimization with non-LBM codes. Considerations for a good parametrization include:

1. Analytical differentiability.
2. Smooth geometric perturbations.
3. Simple setup for complex geometries.
4. Local control over the geometry.
5. Applicability to multidisciplinary optimization (MDO).
6. Interface with CAD.

This list, originally from [54], has been sorted to align with the goal of this project: to define the adjoint pipeline for industrially relevant optimization cases, in the absence of a well-defined optimization problem. Thus, the most important consideration is seamless integration of the parametrization into the adjoint (i.e. differentiability and smoothness), both for the simple tests in this work, and for complex, industrial, 3D problems down the line (i.e. simple and local setup for complex geometries, MDO capability, and integrability with CAD).

Perhaps the best (or most natural) parametrization for industrial optimization is the CAD system used to design the prototype itself. However, interfacing the LBM and CAD solvera and integrating them into the adjoint is a task outside the scope of this project, especially since no specific CAD software was given as a requirement.

The LBM literature has considered parametrizing the geometry by its coordinates (Cheylan et al. [42]), or using an Akima spline (Kusano, [47]). The former approach is not ideal for the methodologies I have presented, primarily because the surface representation is not smooth, leading to cases where the derivative of the minimum distance to the coordinates (needed by PBB) is undefined.[4]. Thus, some of the boundary nodes may be neglected in the gradient calculation, deteriorating the entire pipeline. Furthermore, the parametrization must be coupled with a smoothing algorithm for the gradients, and may lead to a slow and inefficient optimization for large problems [54].

The latter approach belongs to the more popular, widely successful, and entirely valid family of spline parametrization methods, although Akima splines have not been well studied for 3D aerodynamic shape optimization. However, Kusano's implementation has a significant flaw, namely that he numerically evaluates the sensitivities (despite these

[4] e.g., if the lattice node lies on the bisector of two concave line segments.

being analytically defined) with a finite difference. Although he does not report deteriorated gradients, this is not a high-fidelity approach for more complex problems.

Thus, in an effort to expand the LBM-based shape optimization literature, I chose to implement the free form deformation (FFD) mehtod [71], as implemented in PyGeM[5]. FFD can be intuitively understood by imagining that the original object is embedded in a parallelepiped of clear, flexible plastic. One then deforms the plastic in a given way, and the embedded object deforms with it. As such, FFD is not a method that directly defines the shape, rather, it specifies a transformation of the space around it.

This approach satisfies the requirements of analytical differentiability and smoothness and compares competitively with a spline-based method (such as B-splines or NURBS). Although spline methods offer more direct control over specific geometric features (which may result in marginally better designs), FFD can be much simpler to set up for complex geometries [72]. With this method, the user can: define how much control they have over shape perturbations irrespective of the underlying geometry's complexity (i.e., specify coarse deformations on detailed surfaces while maintaining topology, or vice versa), modify only sub-parts of a design with more control than splines, and adaptively increase the number of optimization parameters in a region of interest without changing the underlying shape [72]. Furthermore, the method is suitable for MDO as it can consistently deform a design with both external and internal components [54].

3.5.1. Implementation

For the LBM with PBB it is necessary to define an underlying continuous mathematical representation of the geometry. In my implementation, I use a B-Spline for this initial definition. The code assumes that the object is provided as a set of Cartesian coordinates, with sufficient geometric resolution and an appropriate ordering, and then fits a periodic spline through these points using SciPy's native B-spline implementation[6]. With this, the shape is now represented as a parametric function:

$$\begin{cases} x_s(u_{sp}) = S_x(u_{sp}) \\ y_s(u_{sp}) = S_y(u_{sp}) \end{cases} \tag{3.5.1}$$

where x_s and y_s are the x and y coordinates lying on the spline for a given value of u_{sp}, the spline parameter. The exact form of the spline is captured by the function $S(u_{sp})$, which need not be specified further as it can be evaluated directly using Scipy's `splev` function.

Returning to FFD, Figure 3.1 shows how an environment is defined. The object is placed inside a rectangle with a specified origin $\mathbf{x}_0 = (x_0, y_0)$ and dimensions $L = (L_x, L_y)$. The optimization parameters $\boldsymbol{\mu}_{ij} = \left[\mu_{x_{ij}}, \mu_{y_{ij}}\right]^T$ are the translations, in x and y, of a set of control points arranged uniformly throughout the cylinder, with $(i, j) \in \{0, 1, \ldots, l-1\} \times \{0, 1, \ldots, m-1\}$, and $l = 4$, $m = 3$ (for this example). For clarity, if all of the control points are in their original positions $\boldsymbol{\mu}_{ij} = 0 \; \forall (i, j)$.

[5] https://mathlab.github.io/PyGeM/
[6] The spline is made to pass exactly through all the points.

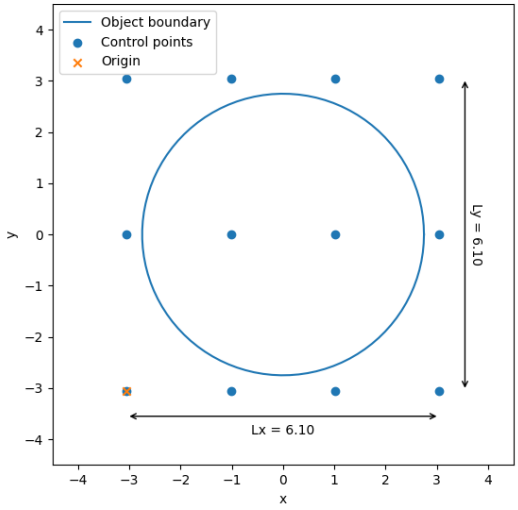


Figure 3.1.: Example definition of an FFD environment around a circle. All control points are undisplaced.

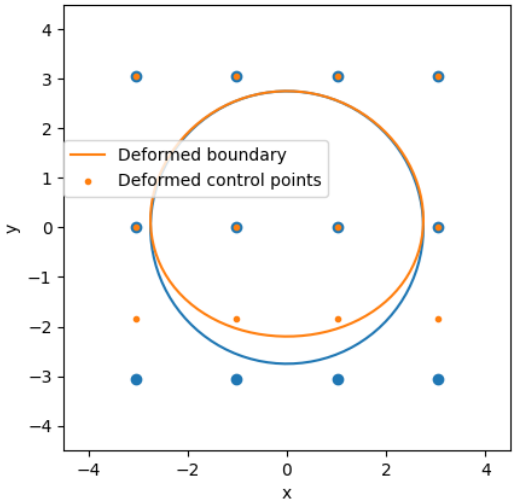


Figure 3.2.: Effect of displacing the lower three FFD control points on the object boundary.

Constraining the method to work on 2D geometries, FFD transforms the curves x_s, y_s, and $z_s = 0$ into:

$$\boldsymbol{\mathcal{X}}(\boldsymbol{\mu}, u_{sp}) = \begin{cases} \mathcal{X}_x(\mu_{x_{ijk}}, u_{sp}) = \left[\mathcal{B}_i^l\left(\frac{x_s(u_{sp})-x_0}{L_x}\right)\mathcal{B}_j^m\left(\frac{y_s(u_{sp})-y_0}{L_y}\right)\mathcal{B}_k^n\left(\frac{-z_0}{L_z}\right)\mu_{x_{ijk}} + \frac{x_s(u_{sp})-x_0}{L_x}\right]L_x + x_0 \\ \mathcal{X}_y(\mu_{y_{ijk}}, u_{sp}) = \left[\mathcal{B}_i^l\left(\frac{x_s(u_{sp})-x_0}{L_x}\right)\mathcal{B}_j^m\left(\frac{y_s(u_{sp})-y_0}{L_y}\right)\mathcal{B}_k^n\left(\frac{-z_0}{L_z}\right)\mu_{y_{ijk}} + \frac{y_s(u_{sp})-y_0}{L_y}\right]L_y + y_0 \\ \mathcal{X}_z = 0 \end{cases} \tag{3.5.2}$$

where Einstein summation convention is employed and $\mathcal{B}_a^b(s) = \binom{b}{a}(1-s)^{b-a}s^a$ is the Bernstein polynomial of degree b. (3.5.2) is a composition of three steps:

- The physical domain is linearly transformed to a reference domain.
- In the reference domain, a deformation is imposed based on the displacement of the control points. The deformation function is a trivariate Bernstein polynomial.
- The deformed points are translated back to the physical domain.

Figure 3.2 shows how a uniform translation in the y direction of the control points leads to a deformation of the boundary.

3.5.2. Derivative of FFD

The derivative $\frac{dd}{d\boldsymbol{\mu}}$ is required for the gradient evaluation (see section 3.2). d is the minimum distance from a fluid node $\mathbf{x}_k = [x_{k_x}, x_{k_y}, 0]$ to the boundary $\boldsymbol{\mathcal{X}}(\boldsymbol{\mu}, u_{sp}) = [\mathcal{X}_x, \mathcal{X}_y, 0]$.

Let

$$\mathbf{r}(u_{sp}, \boldsymbol{\mu}) = \boldsymbol{\mathcal{X}}(u_{sp}, \boldsymbol{\mu}) - \mathbf{x}_k.$$

Then,

$$d = \min_{u_{sp}} ||\mathbf{r}(u_{sp}, \boldsymbol{\mu})|| = \min_{u_{sp}} \mathcal{R}(u_{sp}, \boldsymbol{\mu}).$$

The minimum of $\mathcal{R}$ occurs for a value $u_{min}(\boldsymbol{\mu})$ that we can assume to be a smooth function. Furthermore, we assume $d > 0$. This yields the stationarity condition

$$\left.\frac{\partial \mathcal{R}}{\partial u}\right|_{u_{sp_{min}}} = \frac{\mathbf{r}}{\mathcal{R}} \cdot \frac{\partial \mathbf{r}}{\partial u_{sp}} = 0 \implies \mathbf{r}(u_{sp_{min}}, \boldsymbol{\mu}) \cdot \frac{\partial \mathbf{r}}{\partial u} = 0.$$

With this condition we obtain:

$$\frac{dd}{d\boldsymbol{\mu}} = \frac{1}{d}\left(\mathbf{r} \cdot \frac{\partial \mathbf{r}}{\partial \boldsymbol{\mu}} + \cancel{\mathbf{r} \cdot \frac{\partial \mathbf{r}}{\partial u_{sp}} \cdot \frac{\partial u_{sp}}{\partial \boldsymbol{\mu}}}^{0}\right) = \frac{\mathbf{r}(u_{sp_{min}}, \boldsymbol{\mu})}{d} \cdot \frac{\partial \boldsymbol{\mathcal{X}}}{\partial \boldsymbol{\mu}}(u_{sp_{min}}, \boldsymbol{\mu}) \tag{3.5.3}$$

where

$$\frac{\partial \boldsymbol{\mathcal{X}}}{\partial \boldsymbol{\mu}}(u_{sp}, \boldsymbol{\mu}) = \begin{cases} L_x \mathcal{B}_i^l\left(\frac{x_s(u_{sp})-x_0}{L_x}\right) \mathcal{B}_j^m\left(\frac{y_s(u_{sp})-y_0}{L_y}\right) \mathcal{B}_k^n\left(\frac{-z_0}{L_z}\right) \\ L_y \mathcal{B}_i^l\left(\frac{x_s(u_{sp})-x_0}{L_x}\right) \mathcal{B}_j^m\left(\frac{y_s(u_{sp})-y_0}{L_y}\right) \mathcal{B}_k^n\left(\frac{-z_0}{L_z}\right) \\ 0 \end{cases} \tag{3.5.4}$$

The resulting gradient $\frac{dd}{d\boldsymbol{\mu}}$ has a full dimension $l \times m \times n$ and a non-trivial dimension $l \times m$.

3.6. Main application cases

The adjoint solver is primarily validated by implementing two cases concerning aerodynamic flows. The first case is the steady laminar flow around a NACA 0012 airfoil at Reynolds number $Re = 100$. Reference data for this case is available in [18], consisting of a 2D incompressible direct numerical simulation using OpenFOAM. The airfoil has chord c and is placed at $5°$ angle of attack.

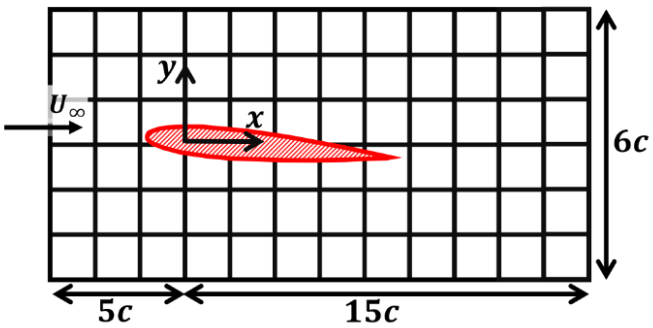


Figure 3.3.: Airfoil test case setup. Figure taken from [18].

A constant momentum $\rho_\infty u_\infty$ is imposed at the inlet. $\rho_\infty = 1$ and the exact value of u_∞ is determined by the considerations detailed in subsection 3.6.1. The northern and southern boundaries impose a free slip boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$ and a zero-gradient extrapolation condition is imposed at the outlet. A density-relaxation sponge is applied with its geometry defined by $x_{min} = 15c$, $x_{max} = 18.75c$ (see subsection 2.3.3).

The second case is the (incompressible) unsteady flow around a cylinder with $Re = 100$ as presented in [55]. Figure 3.4, taken from [55], shows the basic geometric setup.

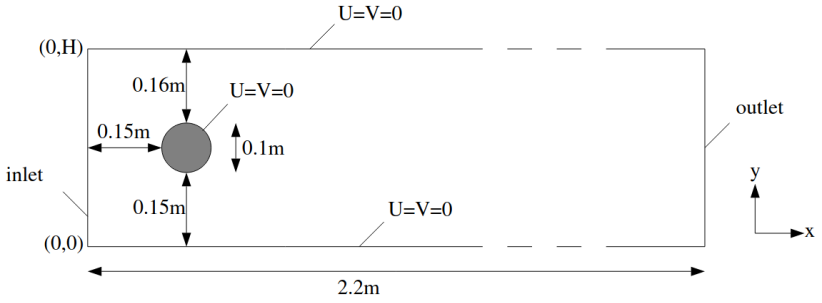


Figure 3.4.: Cylinder case geometry [55]

The obstacle, northern, and southern edges impose a no-slip boundary condition that make the case akin to a wind-tunnel experiment. At the inlet, a parabolic velocity profile is imposed:

$$\mathbf{u}_{in} = \begin{cases} u(x=0,y,t) = 4U_m y \frac{H-y}{H^2} \\ v(x=0,y,t) = 0 \end{cases} \tag{3.6.1}$$

with U_m determined by LBM parameters (see subsection 3.6.1), with the constraint of resulting in $Re = 100$.

The outlet boundary condition is unspecified in the reference paper. In the current implementation, I use a zero-gradient outlet coupled with a sponge with $x_{min} = 1.7m$ and $x_{max} = 2m$. This case also features two levels of grid refinement employed to expedite the calculation and to demonstrate the forward and adjoint solver's multigrid capabilities. The geometry of the grids is shown in Figure 3.5.

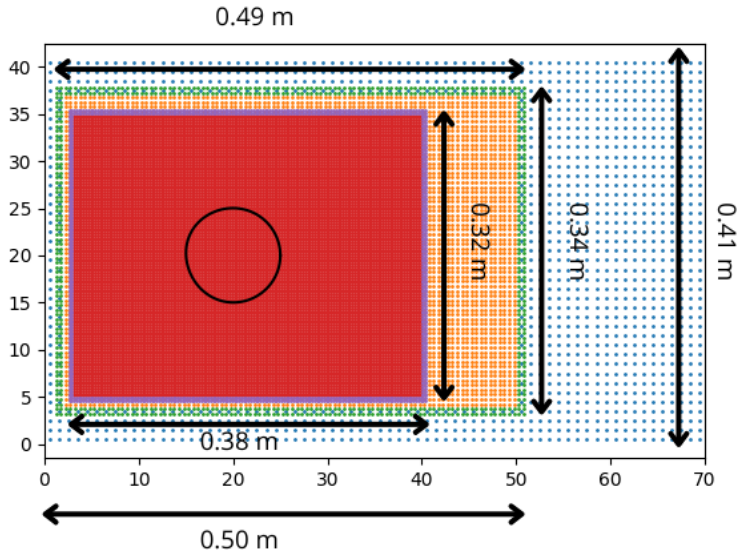


Figure 3.5.: Geometry of hierarchical grids for the cylinder case.

3.6.1. Parameter selection

In the above discussion, it was stated that the values of the inflow momentum and inflow velocity are determined by LBM parameters. This section shows the two main ways this can be done.

Recall that the LBM simulates weakly compressible flows, governed by the non-dimensional Reynolds and Mach numbers:

$$Re = \frac{u \cdot c}{\nu}, \tag{3.6.2}$$

$$Ma = \frac{u}{c_s}, \tag{3.6.3}$$

where u and c are a characteristic velocity and distance, and ν and c_s the kinematic viscosity and speed of sound. In this work c is the chord of the airfoil or the diameter of the cylinder, measured in lattice units and predetermined by the problem definition. $c_s = 1/\sqrt{3}$ (in lattice units) is also fixed for the D2Q9 lattice.

Note that for both test cases Re was given by the problem definition (as it governs incompresisble flows). Thus, the LBM parameters we must choose are u (which determines Ma) and ν. Furthermore, recall the relationship between ν and the relaxation time τ:

$$\nu = \frac{2\tau - 1}{6}. \tag{3.6.4}$$

From this, it is clear that choosing ν is equivalent to choosing τ.

There are two main methods to choose these parameters. The first approach is called *diffusive scaling*. Here, τ is chosen, determining ν, and u follows from

$$u = \frac{Re \cdot \nu}{c}. \tag{3.6.5}$$

If u is too high (the LBM will fail if the maximum Mach number in the flow exceeds $Ma = 0.73$, and may become unstable well below that value), τ must be chosen differently (taking a lower value). If τ gets too close to 0.5, which also negative impacts the LBM's stability, it is necessary to refine the mesh such that c increases (in lattice units).

Importantly, we must consider what happens as we refine the mesh, for example, if running a mesh convergence study. Under diffusive scaling τ remains fixed, so as c increases with every refinement, u decreases. This means the Mach number decreases with each mesh refinement and so does the discrepancy of $\mathcal{O}(Ma^2)$ between the weakly compressible and incompressible Navier-Stokes equations. Thus, diffusive scaling leads the LBM to converge to the incompressible Navier-Stokes equations.

The second method is called *acoustic scaling*. Now, u (and so Ma) is chosen and ν varies as

$$\nu = \frac{u \cdot c}{Re}. \tag{3.6.6}$$

Incidentally, this is the scaling that must be employed between grid refinement levels to ensure continuity of the macroscopic fields. Now, in a mesh refinement study, the LBM does not converge to the incompressible Navier-Stokes equations. Therefore, if one is interested in quasi-incompressible flows, Ma should be chosen low ($Ma < 0.1$) to ensure the $\mathcal{O}(Ma^2)$ discrepancy is small.

Both diffusive and acoustic scaling are used in this work. For each result the method employed is specified as well as the relevant parameters.

Chapter 4: Results

This chapter presents results pertaining to the forward and adjoint solvers and the optimization pipeline. The high-level structure is as follows. Firstly, in section 4.1 the salient features of the forward code and their integration into the LBM code are studied, verified, and validated against a number of reference cases, for both steady and unsteady flows. Secondly, in section 4.2 the adjoint code is verified by comparison with the finite difference method for both steady and unsteady cases. A steady airfoil L/D maximization is run to demonstrate the functionality of the whole optimization pipeline. Finally, section 4.3 demonstrates how to incorporate higher-order boundary conditions into the solver (and more generally, optimization parameters that affect the streaming matrix instead of the collision operator), by means of a toy acoustic propagation problem.

4.1. Forward solver validation

The following results benchmark the accuracy of the LBM solver. We begin by ascertaining that PBB is a consistent no-slip boundary condition for the Navier-Stokes equations and that the RR operator is correctly implemented (subsection 4.1.1). Then, convergence studies for C_L, C_D, L/D and C_p distributions against reference data [18] are shown for the steady airfoil case (subsection 4.1.2). We continue in subsection 4.1.3 with more mesh convergence studies, this time for QoIs relevant to the unsteady vortex-shedding cylinder, again against reference data [55]. The impact of the absorption region is quantified and the mesh refinement algorithm verified.

4.1.1. Consistency check for PBB and verification of RR

The first result consists of a mesh convergence study for incompressible, laminar, quasi-1D channel flow (also known as Poiseuille flow) with PBB. This problem has a known analytical solution (denoted by the subscript ex, for exact):

$$\begin{cases} u_{ex}(y) = \frac{4U_m}{H^2} y(H-y) \\ v_{ex} = 0 \\ \rho = const \end{cases} \tag{4.1.1}$$

where U_m is determined by the magnitude of the pressure gradient (or body force) that drives the flow, v_{ex} is the y component of the velocity, and H the channel height. The main objective of this case is to confirm that PBB is a consistent boundary condition for the pure Navier-Stokes equations, which govern our desired quantities of interest, and not just the Navier-Stokes-Brinkman model. The analytical solution allows us to draw a strong conclusion on this matter, at least for this specific setup. As a secondary goal, this will also (partially) verify that the RR operator is well coded.

The simulation is carried out with periodic boundaries at the inlet and outlet, and the forcing scheme described in [73]. The top and bottom walls are subject to PBB to impose a macroscopic no-slip boundary condition. Three values of the porosity ε are studied: $\varepsilon = [0.5, 0.75, 0.95]$ and two values of the relaxation time: $\tau = [0.7, 1.0]$. Following, [57], [73] the Reynolds number is $Re = 4.8$. To ensure convergence to the incompressible Navier-Stokes equations, error curves are generated for a constant τ. A square mesh is used with $N_y \times N_y$ cells and the nodes are assumed to lie in the cell centres. Results are shown in terms of the L_1 error norm:

$$||e||_{L_1} = \sum_k |u_{ex}(y_k) - u_{num}(y_k)| \tag{4.1.2}$$

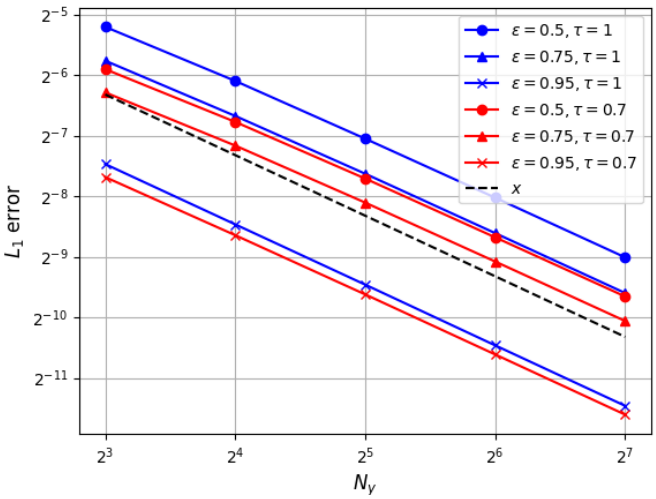


Figure 4.1.: Mesh convergence study for a laminar channel flow using PBB for a range of viscosities and porosities.

where $u_{num}(y_k)$ is the numerical solution obtained at the k-th y node. u_{num} is computed by taking the moments of the distribution functions (2.1.5). Further details of the setup are omitted for brevity (this is a standard case that is well explained in [12], [73]).

Figure 4.1 shows that, for the parameters considered, PBB is consistent and converges with first order accuracy to the true solution. This also suggests a correct implementation of RR. Furthermore, there is a dependency of the error on the relaxation time (meaning the location of the no-slip boundary depends on τ) and the scaling factor ε. The unphysical dependence on τ is expected as it is also a deficiency of halfway bounce back. It cannot be straightforwardly cured as τ is a parameter that, for a more general case, is usually heavily constrained by stability considerations, which themselves are functions of the Reynolds number and mesh size. Moreover, if Ma and Re are specified (as is necessary for the hierarchical grid refinement), then τ is no longer free. ε is a parameter with somewhat more freedom, which for this case should be chosen close to 1, given that the absolute error reduces as $\varepsilon \to 1$.

The velocity and error profiles shown in Figure 4.2 and Figure 4.3 illustrate the impact of introducing a porous layer near the channel walls. This region weakens the near-wall velocity gradients, resulting in a global under-prediction of the velocity profile. Interestingly, the error across all fluid (non-porous) nodes is (almost) constant, meaning the main loss of parabolicity occurs only within the boundary nodes. This reflects the compatibility between the Navier–Stokes and Brinkman models. At the given Reynolds number, as $Ma \to 0$, and under the applied body force (or equivalently pressure gradient), the fluid nodes satisfy an incompressible Stokes equation

$$\mu \nabla^2 \mathbf{u} = \nabla P,$$

where μ is the dynamic viscosity. With periodic boundary conditions, the solution in the fluid region must be a parabola whose shape is determined by the velocity imposed on the first fluid nodes by the Brinkman solution. Because this imposed velocity is lower than what is required for a true Poiseuille flow, the resulting parabolic profile is uniformly shifted downward but still captured very accurately in the bulk, with small compressibility discrepancies of $\mathcal{O}(Ma^2)$.

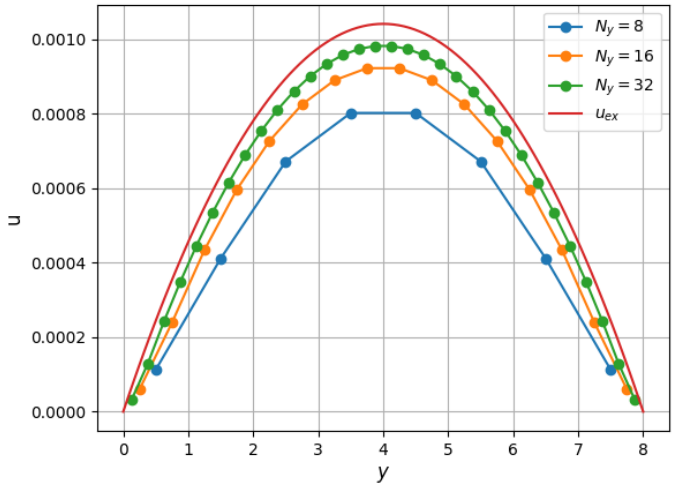


Figure 4.2.: Velocity profiles for $N_y = [8, 16, 32]$ with $\varepsilon = 0.75$ and $\tau = 0.7$.

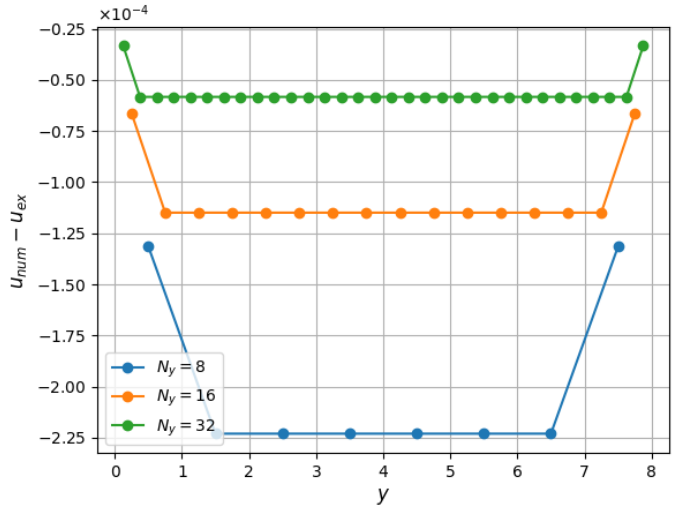


Figure 4.3.: Absolute error $e = u_{num} - u_{ex}$ profiles for $N_y = [8, 16, 32]$ with $\varepsilon = 0.75$ and $\tau = 0.7$.

4.1.2. Validation of the steady case

The following results relate to the steady and laminar airfoil case described in section 3.6. As above, the objective of this section is to show that PBB is a consistent boundary treatment and to draw some conclusions on its accuracy, now for one of the more general cases that validate the adjoint solver.

Firstly, Figure 4.4 is a diffusive-scaling mesh convergence study (with $\tau = 0.56$) for the integrated x-momentum in a region of the wake:

$$mu_x = \sum_{\mathbf{x}_k \in \Omega} \rho u_x(\mathbf{x}_k) \tag{4.1.3}$$

where $\Omega = [7.5c, 10c] \times [1.25c, 5c]$. An analytical solution is unavailable, so the error is computed relative to the numerical solution obtained on the finest mesh:

$$e = |mu_{x_{num}} - mu_{x_{fine}}| \tag{4.1.4}$$

where the finest mesh has $2^9 = 512$ points per chord. The integrated momentum was chosen as a QoI because the LBM's order of accuracy is typically defined in terms of the (moments) of the distribution functions [12], not the surface forces. The free parameters in (2.3.10) are set to $a = 0.15$, $b = 0.25$ and $\sigma_\varepsilon = 1.5$. The sponge was specified with $\sigma = 0.5$ (see subsection 4.1.3 for a discussion on the effect of this parameter on the fields). The figure shows that PBB is consistent and first order accurate (similarly to normal bounce back), with the deviation from the log-linear trend in the last point due to the fact that we are comparing against a fine mesh solution, not the true solution to which PBB is actually converging.

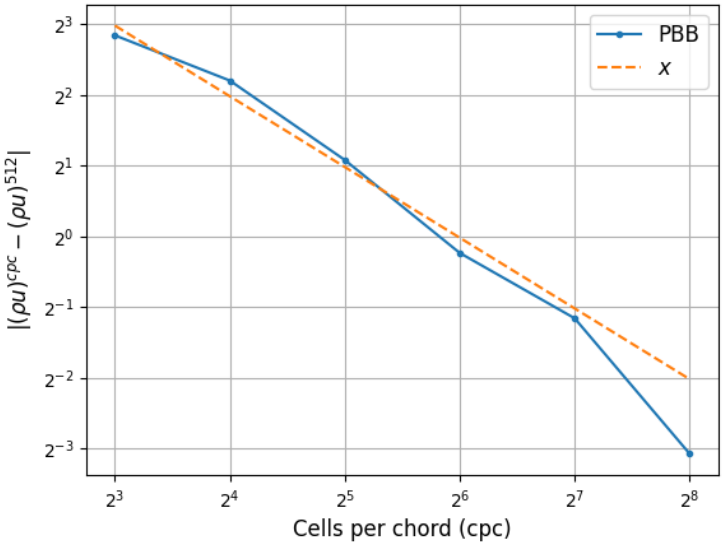


Figure 4.4.: Convergence study of the integrated momentum in the area $\Omega = [7.5c, 10c] \times [1.25c, 5c]$.

Figures 4.5-4.10 show convergence studies (with the same diffusive scaling) for C_D, C_L and L/D, the latter being the focus of the adjoint based optimization. Evidently, the coefficients tend to the reference value (recall, it comes from a 2D DNS with OpenFoam), again highlighting PBB's consistency (and verifying the collision operator). However, although the drag coefficient appears to show smooth first-order accuracy, the lift presents significant oscillations in its error curve. These are likely due to the staircase representation of the airfoil's geometry inherent to the bounce back scheme we are using. As shown by figures 4.11-4.18, the staircase boundary changes non-smoothly between refinement levels

(perhaps most obviously seen by comparing Figure 4.11 and Figure 4.13), a consequence of a staircases inability to converge to the true curvature of an arbitrary boundary. This manifests itself in the pressure distribution as spikes of negative pressure when the flow rounds a concave corner and vice versa. Note how—despite the observed convergence of the C_p curves to the reference data—as the mesh gets finer and the number of steps in the staircase increases, so does the number of spikes, albeit with reduced amplitude. These non-smooth changes in the geometry between refinement levels eventually (once the absolute error is low) lead to the oscillations[1] and large relative errors we observe on the C_L order of accuracy curve.

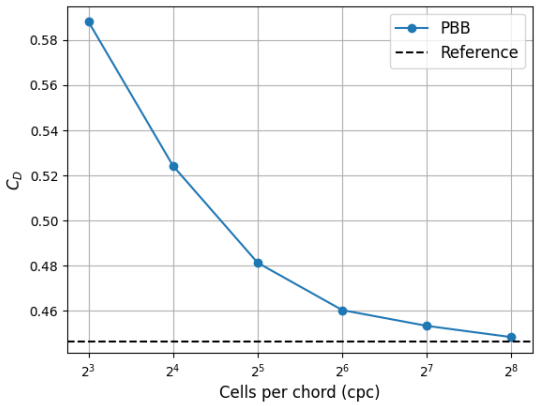


Figure 4.5.: Convergence of C_D to the reference value (dashed line) using PBB.

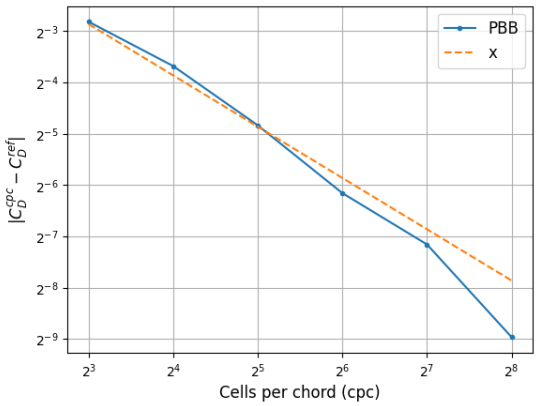


Figure 4.6.: Order of accuracy study for C_D using PBB.

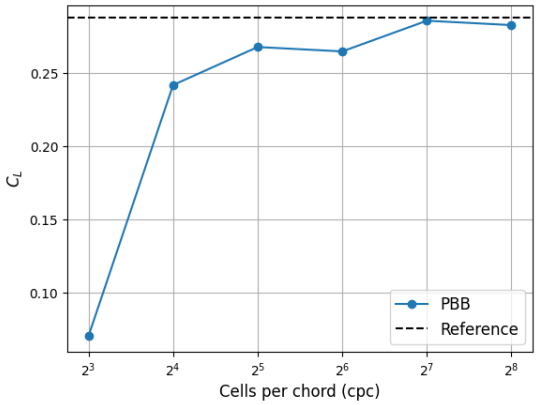


Figure 4.7.: Convergence of C_L to the reference value (dashed line) using PBB.

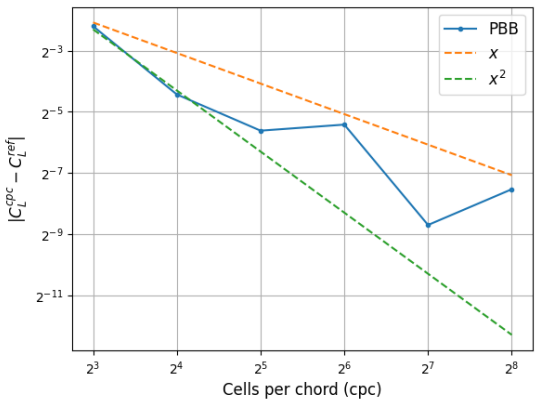


Figure 4.8.: Order of accuracy study for C_L using PBB.

[1] Oscillations occur because each modification to the staircase also changes the effective angle of attack. For example, the box in Figure 4.11 is at $0°$ angle of attack, whereas the two boxes in Figure 4.13 create a pressure difference equivalent to some positive angle of attack. Because the staircase does not change smoothly between grids, neither does the effective angle of attack, leading to large relative errors when we are near the true value.

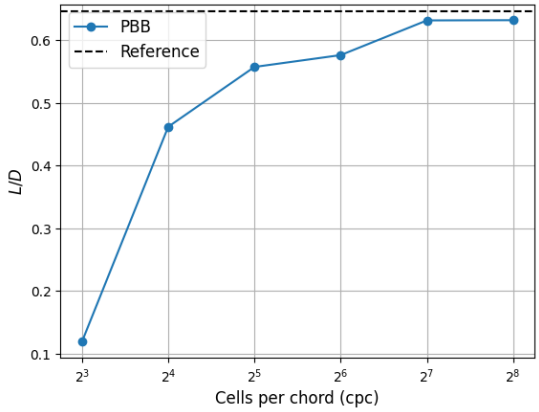


Figure 4.9.: Convergence of L/D to the reference value (dashed line) using PBB.

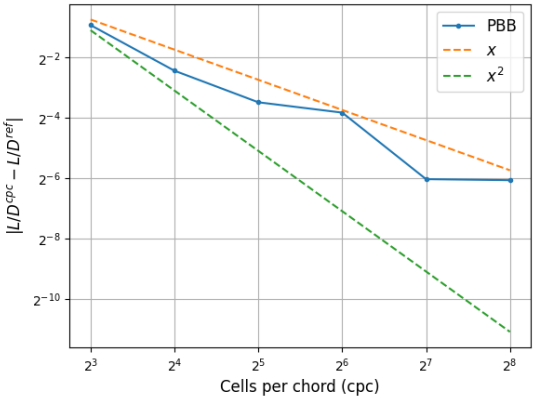


Figure 4.10.: Order of accuracy study for L/D using PBB.

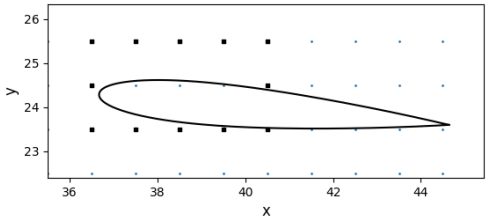


Figure 4.11.: Staircase representation of the airfoil, 8 points per chord.

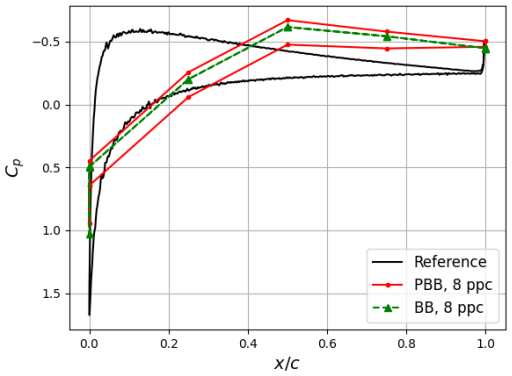


Figure 4.12.: C_p curve, 8 points per chord. Bounce back curve included to highlight the improved geometric resolution of PBB

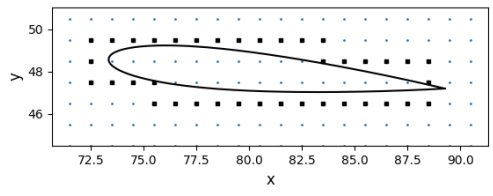


Figure 4.13.: Staircase representation of the airfoil, 16 points per chord.

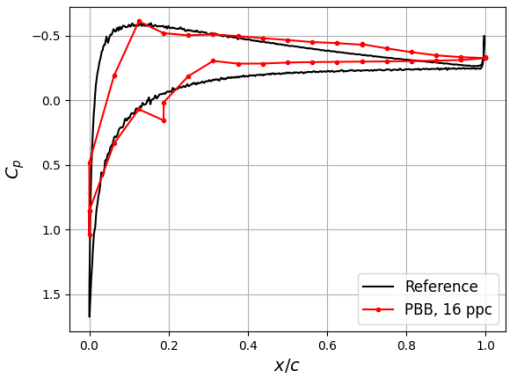


Figure 4.14.: C_p curve, 16 points per chord.

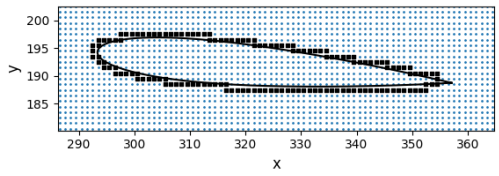


Figure 4.15.: Staircase representation of the airfoil, 64 points per chord.

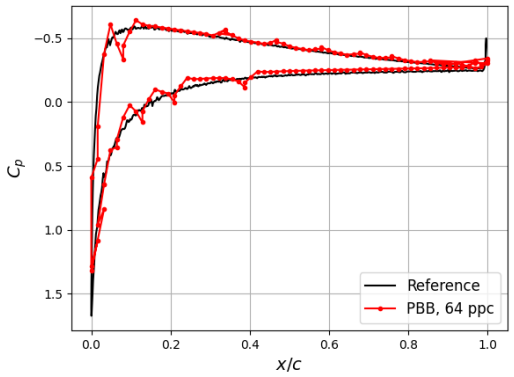


Figure 4.16.: C_p curve, 64 points per chord.

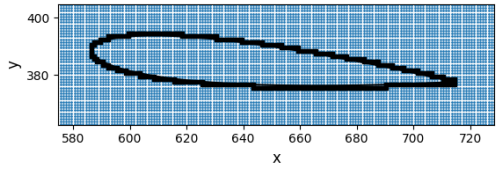


Figure 4.17.: Staircase representation of the airfoil, 128 points per chord.

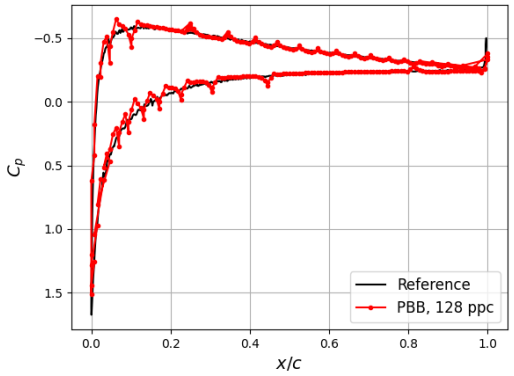


Figure 4.18.: C_p curve, 128 points per chord.

Figure 4.11 and Figure 4.12 reveal that PBB is able to (partially) account for the true shape of the boundary with greater detail than the staircase approximation that underpins bounce back—precisely the property we required for differentiability. Given the extremely coarse mesh and relatively thin airfoil, its staircase representation is a rectangular box at $0°$ angle of attack, which should lead to equal pressures on top and bottom surfaces and zero lift force. However, the pressure distributions are clearly unequal and the result is a net positive lift, consequence of the effective angle of attack imposed by the true curvature. The lift is still greatly underpredicted because we do not resolve almost half the surface, but this observation further validates PBB as a consistent model for a non-slip boundary condition on curved boundaries.

4.1.3. Validation of the unsteady case and grid refinement scheme

The following results continue the validation of the forward LBM code by studying a laminar, unsteady, vortex-shedding cylinder. Primarily, this is done to study the unsteady adjoint formulation in a later section. The case is described in section 3.6 and reference data is available in [55]. This section first presents diffusive-scaling convergence studies for the maximum drag coefficient $C_{D_{max}}$ and maximum lift coefficient $C_{L_{max}}$[2]—which are the quantities for which reference data is available—followed by an acoustic

[2]Computed over one period.

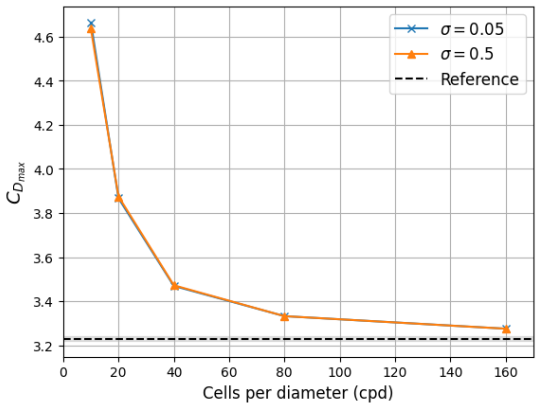


Figure 4.19.: Convergence of $C_{D_{max}}$ to the reference value (shaded region) using PBB with different sponge strengths.

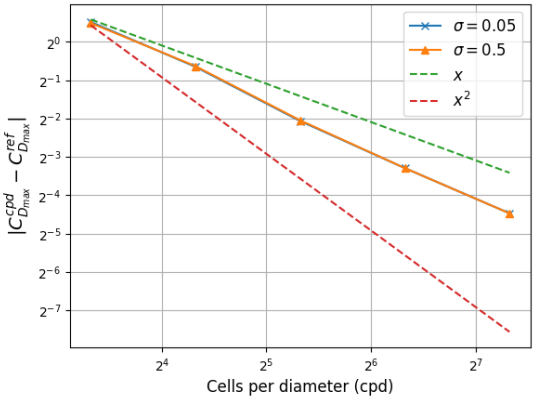


Figure 4.20.: Order of accuracy study for $C_{D_{max}}$ using PBB with different sponge strengths. The true solution is $C^{ref}_{D_{max}} = 3.23$,the midpoint of the range reported in [55].

scaling convergence study of the same variables to quantify the effect of the spurious pressure waves created at the outlet and the damping provided by the sponge. Then, I present an acoustic-scaling convergence study of the time-averaged drag coefficient $C_{\overline{D}}$ (the quantity of interest for the adjoint solver). After that, a verification of the grid refinement scheme is shown, followed by a convergence study for $C_{\overline{D}}$ using hierarchical grids. These validation checks highlight all of the essential features of the forward code which are differentiated in the adjoint pipeline. Results are only presented for the PBB boundary condition as it is the one used for the adjoint solver. For this, we set $a = 0.15$, $b = 0.25$ and $\sigma_\varepsilon = 1.5$.

4.1.4. Convergence studies and sponge analysis

Diffusive scaling convergence to the reference quantities Figures 4.19 to 4.22 show the convergence of $C_{D_{max}}$ and $C_{L_{max}}$ to the reference solutions using diffusive scaling. The relaxation time is fixed to $\tau = 0.52$. Simulations are run for 750 convective times, $t_c = D/U_m$, and analyzed for $t_c > 650$ to ensure that the flow field reaches a state of periodic oscillations about a stationary mean and the initial transient is completely skipped. Results are presented for two strengths of the absorption region at the outlet. This is done to study whether the relatively close proximity of the sponge to the obstacle (about 15 cylinder diameters) results in unwanted upstream effects[3]. For this, diffusive scaling is also essential. We are interested in examining whether the sponge leads to incorrect values of the integrated surface forces, independently of its (favourable) damping effect on acoustic waves reflected and propagated from the outlet. These scale in intensity with the Mach number[4], so we must ensure that this decreases with increasing mesh resolution, especially since the Neumann boundary condition at the outlet does not result in fewer reflections with finer meshes[5].

Evidently, both coefficients are converging to a value in agreement with the reference data. Both follow an order of accuracy slightly greater than one, similarly to the steady

[3]Since the absorption region gradually forces the pressure field towards a freestream value, it may result in an unduly short wake recovery region leading to higher-than-expected pressure gradients on the obstacle surface and inaccurate surface forces.
[4]At the finest mesh $Ma = 0.011$, whereas on the coarsest mesh $Ma = 0.173$.
[5]As opposed to a characteristic boundary condition.

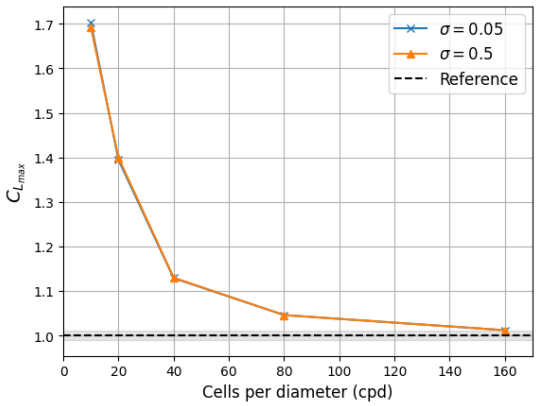


Figure 4.21.: Convergence of $C_{L_{max}}$ to the reference value (shaded region) using PBB with different sponge strengths.

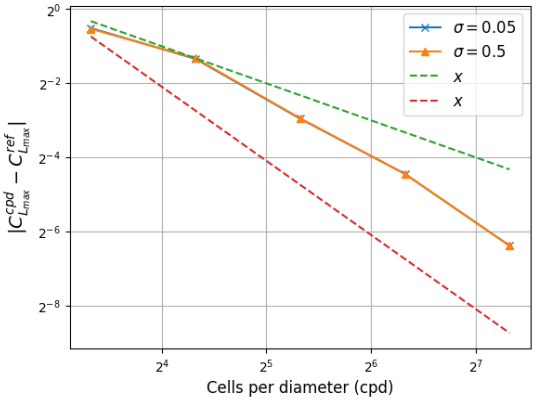


Figure 4.22.: Order of accuracy study for $C_{L_{max}}$ using PBB with different sponge strengths. The true solution is $C^{ref}_{L_{max}} = 1.00$, the midpoint of the range reported in [55].

case. However, due to the artificial porosity introduced by PBB, both present high absolute errors on the coarsest meshes. Concerning $C_{D_{max}}$, the error reduces from 44% to 1.3% for $\sigma = 0.05$; with regards to $C_{L_{max}}$, there is a reduction of nearly 70%. The observed convergence allows us to conclude that the presence of the sponge does not degrade the accuracy of the computed surface forces, when considered independently of its damping properties. Furthermore, we note the insensitivity of the surface forces to the sponge strength for all refinement levels (and thus Mach numbers) — suggesting there is freedom to choose the parameter σ.

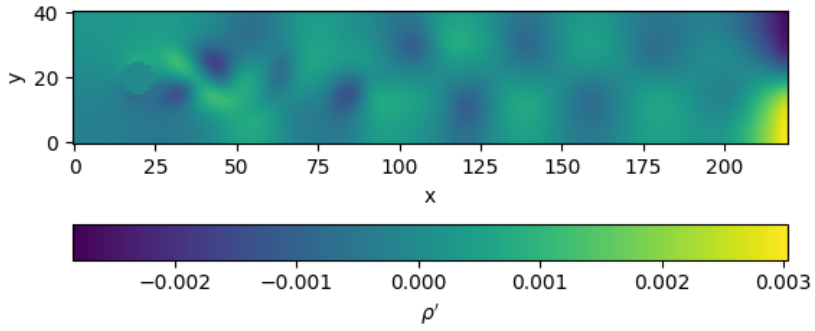
Acoustic scaling convergence studies The following results examine the performance of the solver at a fixed Mach number $Ma = U_m/c_s = 0.0866$. This is necessary to eventually validate the grid refinement scheme, as that assumes continuity of Re and Ma throughout the domain. For this Mach number, the interaction of vortices and acoustic waves with the outlet has a significant effect on the aerodynamic fields and resulting forces, allowing us to further examine the effect of the absorption region. Figure 4.23a and Figure 4.23b show snapshots of the density fluctuation field ρ' for a case with 10 cells per diameter (cpd) at two sponge strengths: $\sigma = 0$ (i.e. no sponge) and $\sigma = 0.5$[6]. The density fluctuation field is defined as

$$\rho' = \rho(\mathbf{x}, t) - \overline{\rho}(\mathbf{x})$$

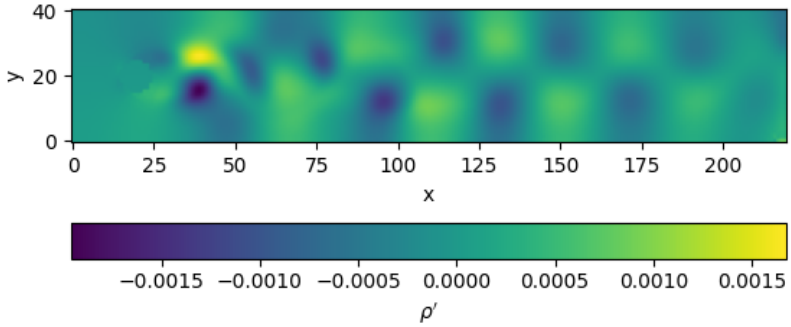
where the overline denotes the time-averaged density over a suitably large number of periods (as many periods as present in $650 < t_c < 750$). For comparability, the snapshots were taken at the instant where ρ' showed a maximum (i.e. when the highest value seen on the colorbar was maximized).

Figure 4.23a clearly shows that very large spurious density fluctuations are generated when the vortices shed by the cylinder cross the extrapolation outlet. The magnitude of these artefacts exceeds (almost by a factor of two) the strength of the hydrodynamic fluctuations created by the vortices themselves (which in turn are larger than the acoustic waves they produce). Given their unphysical nature and notable upstream effect (which we will discuss shortly), the use of a sponge for this case is highly warranted.

[6] I observed, as in the diffusive scaling case, an insensitivity of the surface forces to σ provided $\sigma \geq 0.05$ (the lowest non-zero value studied).



(a) $\rho' = \rho(\mathbf{x}, t) - \overline{\rho}(\mathbf{x})$ for a sponge with $\sigma = 0$.



(b) $\rho' = \rho(\mathbf{x}, t) - \overline{\rho}(\mathbf{x})$ for a sponge with $\sigma = 0.5$.

Figure 4.23.: Density fluctuations ρ' for different sponge strengths.

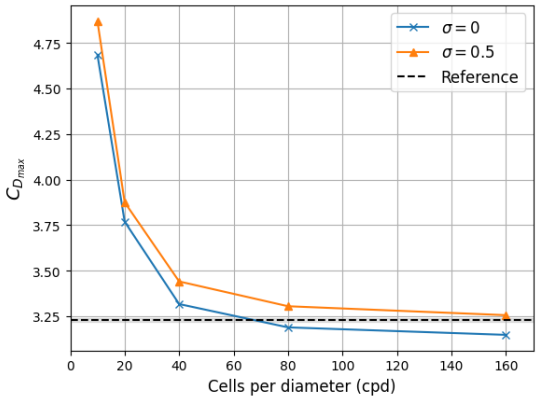


Figure 4.24.: Convergence of $C_{D_{max}}$ to the reference value (shaded region) using PBB with different sponge strengths. Mach number fixed to $Ma = 0.0866$.

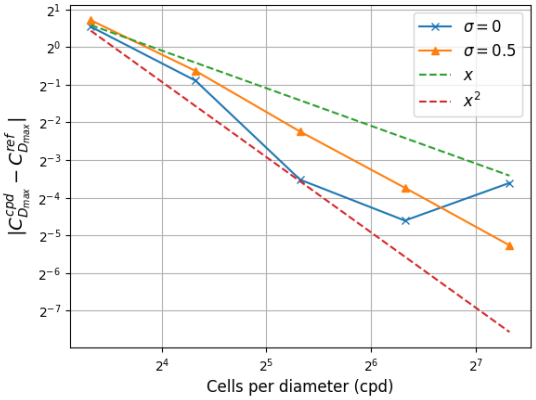


Figure 4.25.: Order of accuracy study for $C_{D_{max}}$ using PBB with different sponge strengths. $Ma = 0.0866$.

Figure 4.23b highlights the effectiveness of such an absorption region. The fluctuations at the outlet are practically eliminated without non-physically impacting the near-field around cylinder or, seemingly, excessively distorting the wake[7].

Figures 4.24 to 4.27 show mesh convergence studies of $C_{D_{max}}$ and $C_{L_{max}}$ to the reference values. Since these are given for an incompressible flow solver, we expect deviations of $\mathcal{O}(Ma^2) = \mathcal{O}(0.14^2) \approx 0.02$, where $Ma = 0.14$ is the maximum observed Mach number in the field. As reported in [69], these should not be of a sufficiently high magnitude to produce a strong deviation from the incompressible values, so we expect to see a similar convergence as that in, for example, Figure 4.20.

The figures show that, when using a sponge, the maximum lift and drag converge to the reference values with an (approximately) constant order of accuracy, indicating the validity of the outlet boundary treatment and near incompressibility of the fields[8]. Without a sponge, both quantities tend to values significantly lower than the reference (plus any compressibility error) and there is also a reduction in the amplitude of the drag oscillations. For $\sigma = 0$, the amplitude of these on the finest mesh is 0.036. In contrast, for $\sigma = 0.5$ the amplitude is 0.065 which is in much better agreement with the value of 0.066 obtained for the finest mesh with diffusive scaling (and $\sigma = 0.05$). This, combined with the lower-than-expected $C_{D_{max}}$ value, allows us to conclude that the pressure fluctuations generated at the outlet have a strong non-physical damping effect on the surface forces (for this case) that should be treated with an absorption region.

Now we can study the QoI for the unsteady case: the time-averaged drag coefficient $C_{\overline{D}}$. Results are only shown for PBB with $Ma = 0.0866$ and a sponge with relaxation parameter $\sigma = 0.5$. Figure 4.28 and Figure 4.29 show a convergence of the QoI to a certain fine-mesh value (with 128 cpd), with an order slightly higher than one. Given the lack of reference data for this solution, it is difficult to draw further conclusions.

[7] For a proper study on the impact of the sponge on the wake, a complete reference incompressible solution would be needed (or an LBM run on a very fine mesh).

[8] For C_L a cleaner convergence curve was obtained using $C^{ref}_{L_{max}} = 0.99$ as the reference value. This still lies within the range reported by [55]. The discrepancy may be due to the compressibility error.

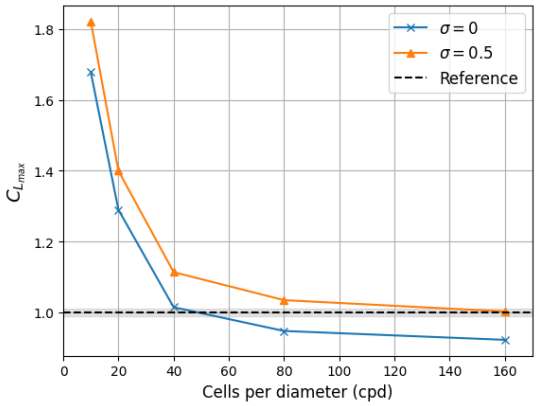


Figure 4.26.: Convergence of $C_{L_{max}}$ to the reference value (shaded region) using PBB with different sponge strengths.

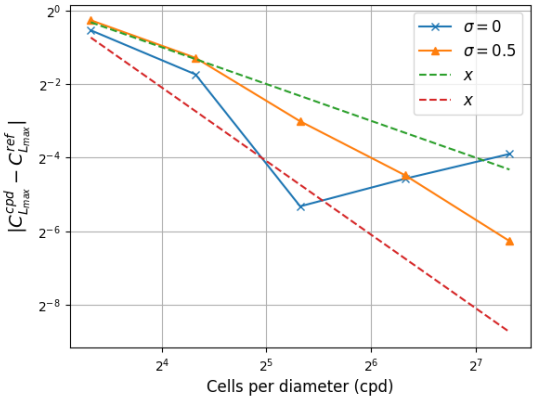


Figure 4.27.: Order of accuracy study for $C_{L_{max}}$ using PBB with different sponge strengths. $Ma = 0.0866$.

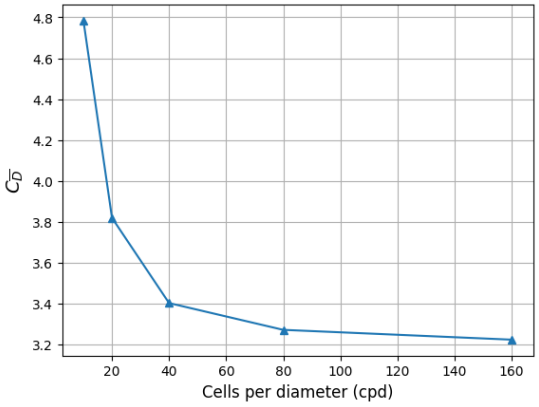


Figure 4.28.: Convergence of $C_{\overline{D}}$.

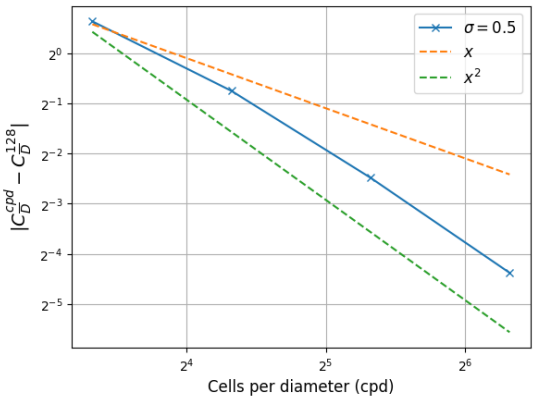


Figure 4.29.: Preliminary order of accuracy study for $C_{\overline{D}}$ using PBB.

4.1.5. Verification and validation of the grid refinement scheme

Verification Firstly, we verify the theoretical conservation of mass and momentum. For this, a Taylor-Green vortex is simulated. Given the periodic boundary conditions and lack of any source terms, this case reveals whether a particular scheme is indeed conservative. Three refinement levels are arranged symmetrically about the centre of the square domain, as shown in Figure 4.30. The outer grid is a circle to highlight—alas, only as a curiosity—that the solver can handle hierarchical grids with arbitrary geometries. The base grid consists of 40 cells in each direction, $Re = 1.0$ and $Ma = 0.001443$ (corresponding to $\tau = 0.6$). The unsteady simulation is run for 10500 coarse timesteps ($= 10500 \times 2^3 = 84000$ fine timesteps). The mass and momentum over the whole domain are computed using:

$$\begin{cases} m(t_n) = \sum_{\mathbf{x}_k} \rho(\mathbf{x}_k, t_n) \Delta x_k^2 \\ m\mathbf{u}(t_n) = \sum_{\mathbf{x}_k} \rho\mathbf{u}(\mathbf{x}_k, t_n) \Delta x_k^2, \end{cases} \tag{4.1.5}$$

where $\Delta x_k = 1$ on the coarsest level and decreases by factors of 2 with increasing mesh resolution. t_n denotes a coarse timestep.

Table 4.1 shows the evolution of the conserved moments. The final relative error for the

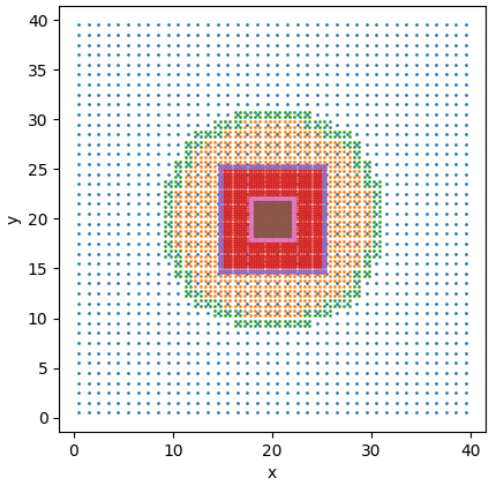


Figure 4.30.: Refinement levels for the Taylor-Green case. The green, purple, and pink areas indicate the grid interface layers.

mass is $\mathcal{O}(10^{-15})$ and both momentum components stay close (near machine precision) to their initial 0 values. This result is a positive pass/fail verification for the implementation. The forthcoming results continue the benchmarking focusing on $C_{\overline{D}}$, the QoI for the adjoint gradients.

Table 4.1.: Evolution of conserved moments for the Taylor-Green vortex with hierarchical grids.

Timestep	m	mu_x	mu_y
0	1600.0000002352992	-1×10^{-17}	-4×10^{-17}
5500	1600.0000002352986	-1×10^{-13}	-2×10^{-14}
10500	1600.000000235306	-2×10^{-13}	8×10^{-14}

Validation Finally, we study how the refinement scheme can be applied to the unsteady cylinder case. The geometry of the refinement levels is shown in section 3.6. The refinement levels are symmetric with respect to the parabolic profile at the inlet (i.e. symmetric about the domain's y-midpoint) to minimize artificial flow gradients $\frac{\partial u_y}{\partial y}$ introduced by the grids themselves[9]. The simulation is run for 750 convective times with $Ma = 0.0866$ and $Re = 100$. The cylinder is resolved with 40 cpd and the mesh has 29304 points out of a maximum 144320 (if they were all fine).

Forces are computed every fine timestep and averaged over every coarse timestep (i.e. the values of C_D for the 4 fine timesteps that make up one coarse step are averaged.). This is done to remove some staggering-in-time observed in the forces. For example, at the final coarse step, the drag coefficient for each fine step is:

$$C_D = \begin{bmatrix} 3.370 & 3.358 & 3.370 & 3.358 \end{bmatrix}. \tag{4.1.6}$$

This behaviour was identified by Rohde et al. in his original paper on LBM grid refinement [57], where he attributes it to the streaming of populations for which a coarse collision step was applied into the fine grid (i.e. the exploded populations), representing a mismatch of time scales (given the different relaxation times on each grid). He suggests

[9] Gradients in the x-direction are inevitable.

averaging-in-time as a post-processing step like I have done. Strictly speaking, the drag should be averaged every two fine steps as that is the staggering frequency. However, as we are ultimately interested in the time averaged drag coefficient (over many or all timesteps) this is not a concern.

Table 4.2 shows that excellent agreement is obtained for the drag coefficients when comparing the solution on the uniform and locally refined meshes, suggesting the implementation of the refinement scheme is correct. The agreement is less good for the lift but remains satisfactory. To study why this is the case one could compute the flow gradients on the refined mesh and compare them to the uniform grid, but this was not done in this work.

Notably, this refinement geometry results in a reduction of the computational cost by a factor of 5 (the performance of the LBM is memory access driven). Even better improvements could be obtained with an adaptive mesh refinement strategy, although special care should be given to the geometry of the refinement levels as, from experience, asymmetries in the grids may result in significant errors.

Table 4.2.: Comparison between QoIs obtained on a uniformly refined mesh with 144320 total points and one with local grid refinement and 29304 points. Both resolve the cylinder to 40 cpd. % relative error computed with the uniform refinement as the truth.

QoI	Uniform refinement	Local refinement	% error
$C_{D_{max}}$	3.441	3.440	0.03
$C_{\overline{D}}$	3.402	3.401	0.03
$C_{L_{max}}$	1.114	1.103	0.99

4.2. Adjoint solver validation

The adjoint solver is validated as follows. Section subsection 4.2.1 verifies the correct implementation of the steady adjoint solver by comparing the gradients to those obtained from a finite difference calculation. Then, subsection 4.2.2 applies the adjoint-based gradients to a L/D optimization. Finally, subsection 4.1.3 compares the gradients from the unsteady case to a finite-difference approximation and their suitability for shape optimization is preliminarily presented.

4.2.1. Validation of adjoint-based gradients for the steady case

For the gradient computation and shape optimization of the steady airfoil, the object is embedded in an FFD environment with $l = 13$ points in the x direction and $m = 2$ in the y direction, as shown in Figure 4.31.

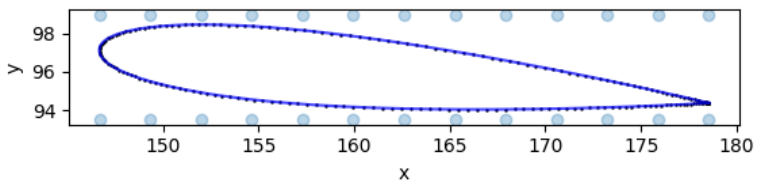


Figure 4.31.: FFD environment for the steady case.

To keep the chord length, and thus the Reynold number constant, deformations are only allowed in the y direction: $\mu_{x_{ijk}} = \mu_{z_{ijk}} = 0$. Furthermore, the two columns nearest the

leading and trailing edges are fixed: $\mu_{y_{0jk}} = \mu_{y_{1jk}} = \mu_{y_{11jk}} = \mu_{y_{12jk}}$, to prevent the airfoil from translating along the y-axis. Fixing two columns also ensures that the (deformed) shape is $\mathcal{C}^1$ continuous, by construction of the FFD method [71].

For the finite difference verification, a mesh with 32 points per chord is used as a compromise between accuracy and computational cost, especially since computing the finite difference gradients takes several evaluations of the forward code. For the undeformed case, the error to the reference value is 13%; however, to get below 5% would require a mesh four times as fine, given the rather slow convergence of L/D. Nevertheless, since the adjoint formulation is discrete, the gradients should be exact irrespective of the mesh size.

Only a subset of the adjoint-based gradients is compared to those obtained by a forward finite difference:

$$\frac{dI}{d\mu_{y_{ijk}}} \approx \frac{I(\mu_{y_{ijk}} + h) - I(\mu_{y_{ijk}})}{h} \tag{4.2.1}$$

where the spacing $h = 10^{-6}$ was determined to be sufficiently fine to yield h-independent results after a convergence study omitted here for brevity. Table 4.3 shows the successful results of this validation check. The highest percentage error is 0.00393%, a significant improvement compared to the results presented by Cheylan et al. [42] for a similar case. Those results have a maximum error of approximately 30%, as a consequence of their incorrect boundary treatment.

With respect to boundary conditions, these results partially validate the proposed treatment of the non-local extrapolation outlet using the transposed streaming matrix. However, this cannot be considered a complete corroboration because, for this case set-up, where the object is far removed from the outlet, this boundary has a minimal effect on the cost-function gradient. Indeed, adjoint gradients were computed to (practically) the same accuracy as those in Table 4.3 by setting the adjoint outlet boundary condition to zero for all (adjoint) populations. This observed insensitivity of the gradients to the (adjoint) outlet boundary conditions has been reported in literature [74], allowing some authors to disregard the treatment of inlet/outlet boundary conditions entirely. Although this may be a practical solution for a steady case, as we will see in subsection 4.2.3, this is not a valid adjoint boundary treatment for unsteady gradients.

Table 4.3.: Comparison between finite-difference (FD) reference values and adjoint results for selected entries. Percent relative error is computed with FD taken as the truth.

Entry	FD (truth)	Adjoint	Percent error (%)
$\mu_{y_{2,0}}$	0.0281954219	0.02819653	0.00393
$\mu_{y_{2,1}}$	0.0127541349	0.01275462	0.00380
$\mu_{y_{6,0}}$	0.0112698375	0.01127016	0.00286
$\mu_{y_{6,1}}$	-0.0049200743	-0.00491990	0.00354
$\mu_{y_{10,0}}$	-0.0239572275	-0.02395675	0.00199
$\mu_{y_{10,1}}$	-0.0091496929	-0.00914939	0.00331

4.2.2. Shape optimization results for the steady case

Given the adjoint gradients validated above, we can now test the solver's shape optimization capabilities. For this, the same set-up as above is used (to increase confidence in the physical validity of the optimization, results on a finer mesh are also presented), the optimizer is Scipy's SLSQP[10] implementation, and aside from fixing the leading and trailing edges and chord length, the only other constraint is a bounds constraint on the

[10]https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html

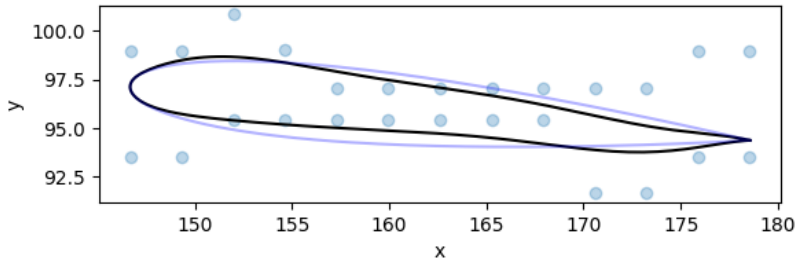


Figure 4.32.: Final shape after L/D optimization. Original shape shown in light blue.

maximum allowed deformation of $\mu_{max} = \pm 0.35$[11]. SLSQP can handle general non-linear constraints (to impose constant thickness to chord ratio, for example) but these were not explored and are instead left for future work on more sophisticated optimization cases. The cost-function is L/D, which we seek to maximize.

Figure 4.32 shows the shape obtained after the optimizer converged and Figure 4.33 shows a comparison between the original and optimized pressure distributions. The evolution of the surface force coefficients is presented in Table 4.4.

Table 4.4.: Evolution of surface force coefficients on a mesh with 32 cells per chord.

Parameter	Original	Optimized	% change
C_L	0.26820	0.32737	+22.1
C_D	0.48101	0.48095	-0.01
L/D	0.55757	0.68067	+22.1

Evidently, the optimizer has produced a better design for the specified cost-function. The lift enhancement is due to the greater convex curvature over the top surface (from $x = 150$ to $x = 155$, approximately) and the stronger concave curvature over the bottom surface (from $x = 150$ to $x = 170$). The first modification strengthens suction on the top surface, visible as a larger suction peak at $x/c = 0.13$ in the C_p curve. The second raises the pressure on the lower surface, evident from the overall downward shift of the C_p curve between $x/c = 0.25$ and $x/c = 0.72$. Together, these effects produce a larger pressure differential between the surfaces and thus a higher lift.

The drag stays essentially constant, likely because any increment in the viscous drag over the top surface due to increased supervelocities is compensated by decreased supervelocities over the lower surface. Further analysis of this phenomenon grounded in the shear stress distribution should be carried out to strengthen this conclusion. Given the 2D setup, there is no lift-induced drag. The combination of higher lift and constant drag results in improved L/D.

We can ascertain that the optimizer has converged to a local minimum in the specified design space by checking whether the norm of the gradient $||\frac{dI}{d\boldsymbol{\mu}}||_{L_2}$ is near zero and/or the bounds constraints is activated. The initial gradient norm is 0.0579 which slightly decreases to 0.0492, suggesting a turning point has not been reached. However, the matrix of (allowed) deformations is

$$\boldsymbol{\mu}_y = \begin{bmatrix} 0.35 & 0.01765153 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & -0.35 & -0.35 \end{bmatrix},$$

[11] This was determined by trial and error to yield realistic shapes. Without this constraint excessive deformations are sometimes tried which break the optimizer due to geometric degeneracy (top and bottom surfaces start to cross each other, for example).

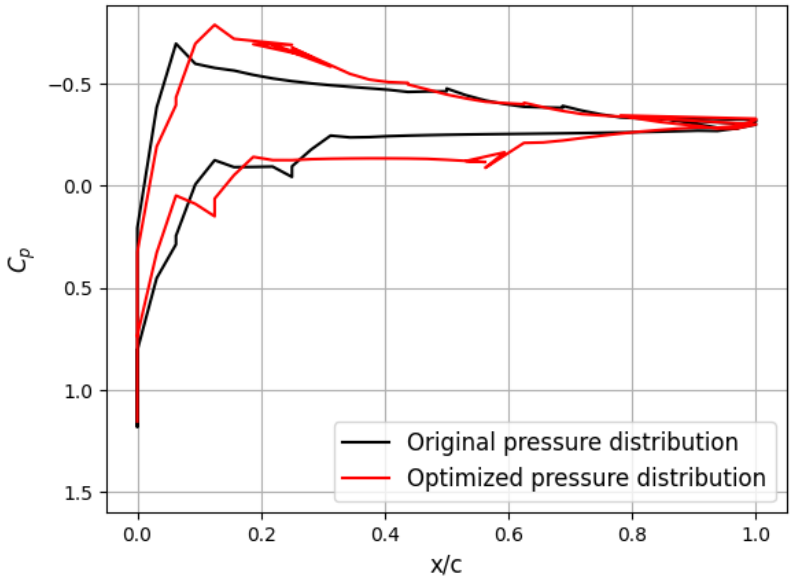


Figure 4.33.: Comparison between original and optimized pressure distributions.

where the rows correspond to the top and bottom rows of FFD nodes in Figure 4.32. The matrix shows that all parameters, bar one, have hit their constraint. The one exceptional parameter presents a sensitivity an order of magnitude smaller than its neighbours, suggesting if the optimization were for longer it would also hit its limit. This, combined with the positive convergence message displayed by Scipy[12] as well as the asymptotic L/D history shown in Figure 4.34, suggests a local minimum has been found within the constrained design space.

Optimization results on a finer mesh

The above results are for a fairly coarse mesh, where the relative error of the QoI (L/D) to the reference value is 13%, possibly resulting in a physically inaccurate optimization. To study whether the discretization error has a significant effect on the final shape, the following results are generated on a mesh 4 times as fine, with 128 cells per chord. The relative error to the reference value is now 2%.

Firstly, Table 4.5 shows the evolution of the relevant QoIs. There is still a significant, albeit slightly reduced, increase in the L/D. Given the significantly finer mesh, this provides more certainty that the optimization pipeline is yielding physically consistent results. Examining the final shape (Figure 4.35) and matrix of optimization parameters:

$$\boldsymbol{\mu}_y = \begin{bmatrix} 0.35 & 0.2564164 & -0.11199134 & -0.30049473 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & -0.16003677 & -0.35 \end{bmatrix}$$

we can conclude that the methodology on the coarser mesh identified the correct design paradigm, namely to increase concavity on the lower surface and convexity near the leading edge of the upper surface.

This is complemented by Figure 4.36, where we observe increased pressure on the lower side and increased suction on the upper side, although the pressure difference between original and optimized distributions on the upper side appears less pronounced on the fine mesh than on the coarse one (this may explain why the drag decreases more significantly on the fine mesh — the velocity gradients on the lower side are weakened more

[12] The Scipy implementation of SLSQP terminates if the gradient of the Lagrangian or sum of constraint violations or step size or cost-function changes exceed some user-specified tolerance, here `ftol = 1e-6`.

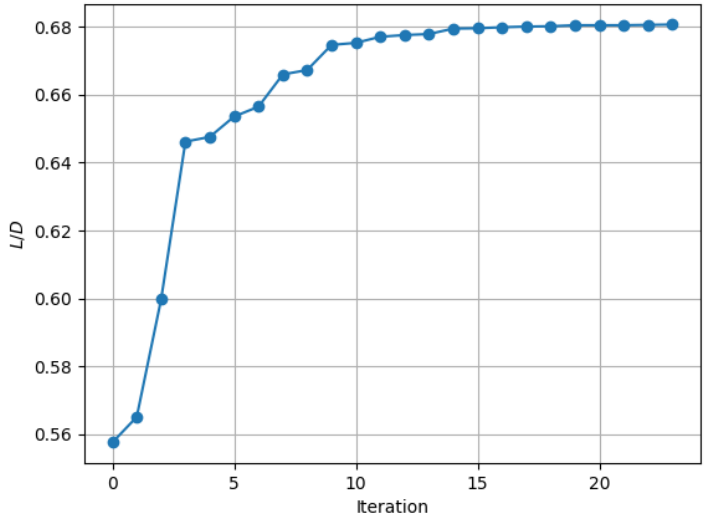


Figure 4.34.: L/D optimization history.

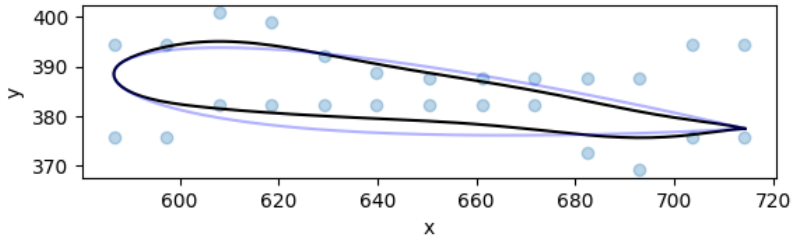


Figure 4.35.: Final shape after L/D optimization on a mesh with 128 cells per chord.

than they are strengthened on the upper side). Comparing the matrices, we can see that all elements share the same sign, and most share the same constrained value, indicating the (domain-constrained) minimum obtained on the coarse mesh is accurate. We note that the final gradient magnitude on the fine mesh is $||\frac{dI}{d\boldsymbol{\mu}}||_{L_2} = 0.0289$, compared to $||\frac{dI}{d\boldsymbol{\mu}}||_{L_2} = 0.0492$ on the coarse mesh. This suggests that running the optimization for a longer time on the fine mesh may lead to identical results between the grids, whereby all the parameters are constrained.

Table 4.5.: Evolution of surface force coefficients on a mesh with 128 cells per chord.

Parameter	Original	Optimized	% change
C_L	0.2877	0.3377	+17.4
C_D	0.4494	0.4456	-0.8
L/D	0.6403	0.7578	+18.4

4.2.3. Validation of adjoint-based gradients for the unsteady case

The next set of results serve to validate the unsteady adjoint-based gradients. These are studied for the cylinder case, with multiple refinement levels and $I = C_{\overline{D}}$. Due to time constraints, an optimization is not run for this case.

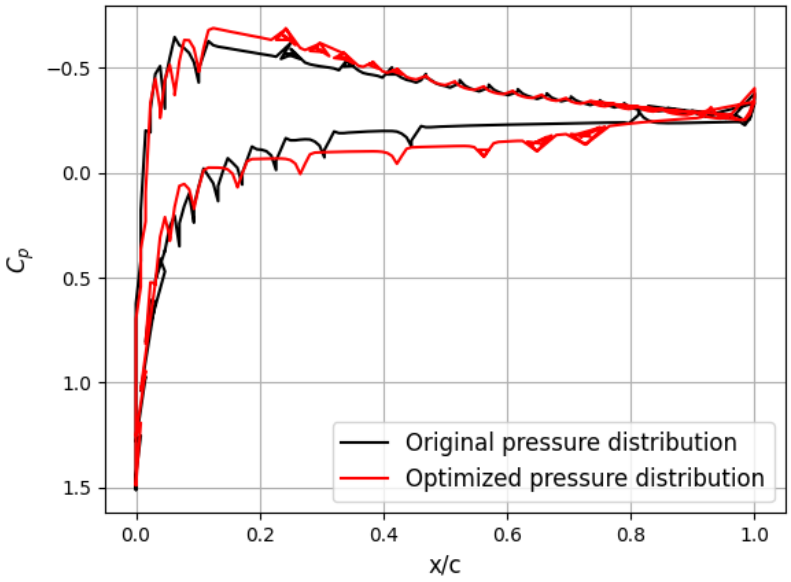


Figure 4.36.: Comparison between original and optimized pressure distributions on a mesh with 128 cells per chord.

The cylinder is embedded in a free form deformation environment with $l = m = 3$ points in the x and y directions as shown in Figure 4.37. If an optimization case were run, a larger set of points would likely yield a wider design space and a better final shape. However, for the gradient verification, these points will do. Again, to keep the diameter constant, deformations are only allowed in the y direction but no constraints are imposed at the edges.

Test case 1 The first test is a backwards in time integration, 10,000 time-steps long, of the initial transient period when the flow adjusts to the cylinder, starting at $t = 0$. For this case, there is no sponge[13] and no local grid refinement. A coarse mesh with 10 cells per diameter is used to minimize the computational cost. This should not lead to a reduction in the accuracy of the gradients since we are working with a discrete adjoint that exactly differentiates the solver. This case tests the adjoint implementation of the non-local extrapolation boundary condition more thoroughly than the steady case, since the reflections produced at the outlet have a significant effect on the cost function.

Table 4.6 presents the adjoint and finite difference based gradients, again showing good agreement between the two that suggests the adjoint method has been correctly implemented (as well as the checkpointing routine, which is employed for all the unsteady cases). For comparison, a column is also included with the gradients obtained with a homogenous adjoint outlet (as we did for the steady case). There is a noticeable reduction in accuracy, however, the error remains small, suggesting the homogenous adjoint outlet boundary condition may be a useful approximation, as in the steady case. This, however, is not a sound conclusion to draw. In Appendix F, a similar test case is shown except that the cost function is the mean square density — which may serve as a measure of sound intensity, for example — measured on a line located downstream of the cylinder.

[13]It is necessary to study the adjoint outlet boundary condition without a sponge first because its effect is to reduce the upstream influence of the outlet so much that it no longer contributes to the sensitivities and can be satisfactorily treated with a homogenous adjoint Dirichlet boundary condition. Given the necessity of the sponge we determined in our previous discussions, one could conclude that the transposed streaming matrix is an unnecessary approach for this work. Nevertheless, it is employed as it is a general methodology, easily extensible for future research.

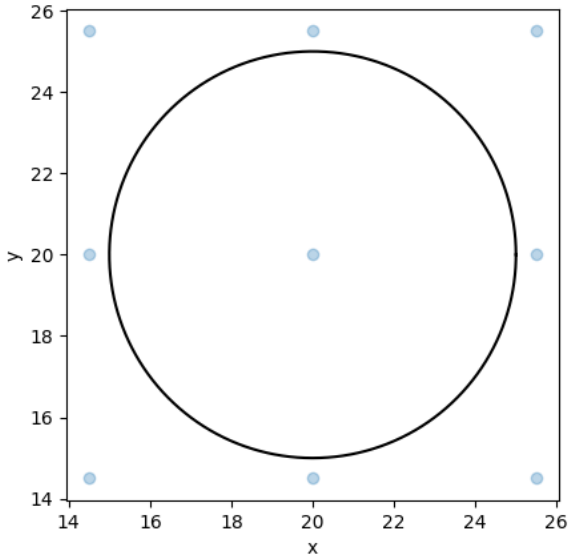


Figure 4.37.: 3×3 FFD environment for the unsteady cylinder case.

For this case, the homogenous outlet results in vastly reduced accuracy, to the point of making the resulting gradients unusable. Therefore, to ensure a reliable adjoint calculation for general cost functions, I recommend using the exact outlet treatment derived from considering a global form of the boundary condition operator.

Table 4.6.: Comparison between finite-difference (FD) reference values and adjoint results. $C_{\overline{D}}$ for the first 10000 time-steps. No grid refinement or sponge. Percent error is computed with FD taken as the truth.

Entry	FD (truth)	Adjoint	Error (%)	Adjoint (hom. outlet)	Error (hom. outlet) (%)
$\mu_{y_{0,0}}$	-0.004014	-0.004014	-0.008191	-0.004011	0.07428
$\mu_{y_{0,1}}$	-0.0001141	-0.0001140	0.02819	-0.0001142	-0.1209
$\mu_{y_{0,2}}$	0.004288	0.004288	-0.01235	0.004292	-0.1110
$\mu_{y_{1,0}}$	-0.005160	-0.005160	-0.01001	-0.005160	0.003365
$\mu_{y_{1,1}}$	-0.0001351	-0.0001351	-0.02124	-0.0001369	-1.383
$\mu_{y_{1,2}}$	0.005430	0.005431	-0.01080	0.005434	-0.06144
$\mu_{y_{2,0}}$	-0.002201	-0.002201	-0.01455	-0.002209	-0.3521
$\mu_{y_{2,1}}$	-0.0004238	-0.0004239	-0.03012	-0.0004308	-1.644
$\mu_{y_{2,2}}$	0.001413	0.001413	-0.001493	0.001406	0.4980

Test case 2 The second computation consists of a similar set-up except that a sponge is now used with $\sigma_{max} = 0.5$ and two levels of local grid refinement are employed (with the geometry shown in section 3.6). With the local grid refinement, the cylinder is represented with 40 cpd (as in Table 4.2). Table 4.7 shows the results which are of a similar accuracy as before, validating the proposed adjoint grid refinement scheme.

Table 4.7.: Comparison between finite-difference (FD) reference values and adjoint results. $C_{\overline{D}}$ for the first 10000 coarse time-steps (i.e. 40000 fine time-steps). Two levels of grid refinement and a sponge with $\sigma_{max} = 0.5$. Percent error is computed with FD taken as the truth.

Entry	FD (truth)	Adjoint	Error (%)
$\mu_{y_{0,0}}$	-0.008952	-0.008952	-0.0009657
$\mu_{y_{0,1}}$	0.0006624	0.0006626	-0.01811
$\mu_{y_{0,2}}$	0.01418	0.01418	-0.002811
$\mu_{y_{1,0}}$	-0.02001	-0.02001	-0.002114
$\mu_{y_{1,1}}$	-0.0008277	-0.0008276	0.005177
$\mu_{y_{1,2}}$	0.02079	0.02079	-0.001735
$\mu_{y_{2,0}}$	-0.01117	-0.01117	-0.001235
$\mu_{y_{2,1}}$	-0.003893	-0.003893	0.003565
$\mu_{y_{2,2}}$	0.002221	0.002221	-0.003255

Test case 3 For the last test case, we examine a truncated backwards in time integration for a range $P = 2^i, \quad i \in [0, 1, ..., 6]$ of shedding periods. This is done to reduce the computational cost of the gradient computation by leveraging the state of periodicity achieved by the flow.

Tables 4.8-4.10 and Figure 4.38 show the results. The integration truncation clearly introduces a significant and dominant error. This error decreases as the integration window is lengthened with approximately first order convergence to the FD approximation [14]. Table 4.11 presents a more global view of the error by taking the error norm $||e|| = ||e||_{L_2}/||(dI/d\boldsymbol{\mu})_{FD}||_{L_2} \times 100$ where $e = (dI/d\boldsymbol{\mu})_{FD} - (dI/d\boldsymbol{\mu})_{adj}$. This better highlights the convergence behaviour of the gradients (the error halves as the window length doubles, especially for $P > 4$) and serves as a comparison with the similar case reported by Kusano [47]. For $P = 16$ (which is the value for which the error is given by Kusano) my gradients have an error of 2.686% compared to his 19.8%. This improvement will not be further analyzed as the shape parametrization and boundary treatment are different in my case, as is the case itself. However, we can suggest that, in this case, for any P, or perhaps more safely any $P > 2$, the adjoint gradients have sufficient accuracy to run a successful optimization pipeline given Kusano's favourable results.

Table 4.8.: FD reference vs adjoint results computed for 1 and 2 periods. Percent error is computed with FD as truth.

Entry	FD	Adj $P = 1$	Error (%)	Adj $P = 2$	Error (%)
$\mu_{y_{0,0}}$	-0.01259	-0.01441	-14.44	-0.01386	-10.09
$\mu_{y_{0,1}}$	6.627×10^{-5}	-0.000327	593.1	-0.0003211	584.5
$\mu_{y_{0,2}}$	0.01287	0.01184	8.030	0.01118	13.09
$\mu_{y_{1,0}}$	-0.02191	-0.02072	5.407	-0.02109	3.739
$\mu_{y_{1,1}}$	2.297×10^{-5}	-0.0004671	2133.	-0.0004819	2197.
$\mu_{y_{1,2}}$	0.02214	0.01740	21.42	0.01756	20.69
$\mu_{y_{2,0}}$	-0.007020	-0.006861	2.277	-0.007002	0.2630
$\mu_{y_{2,1}}$	-1.779×10^{-5}	-0.0001692	-850.9	-1.577×10^{-4}	-788.0
$\mu_{y_{2,2}}$	0.006997	0.005620	19.68	0.005697	18.58

[14] Given the periodic nature of the flow the FD gradients are approximately (practically) constant regardelss the number of periods over which they are computed.

Table 4.9.: Adjoint results computed for 4 and 8 periods. Percent error is computed with FD from Table 4.8 as truth.

Entry	Adj $P = 4$	Error (%)	Adj $P = 8$	Error (%)
$\mu_{y_{0,0}}$	-0.01341	-6.521	-0.01305	-3.603
$\mu_{y_{0,1}}$	-0.0001916	388.9	-7.802×10^{-5}	217.6
$\mu_{y_{0,2}}$	0.01168	9.240	0.01219	5.269
$\mu_{y_{1,0}}$	-0.02144	2.144	-0.02166	1.141
$\mu_{y_{1,1}}$	-0.0003187	1489.	-1.692×10^{-4}	837.8
$\mu_{y_{1,2}}$	0.01909	13.77	0.02043	7.700
$\mu_{y_{2,0}}$	-0.007046	-0.3598	-0.007041	-0.2885
$\mu_{y_{2,1}}$	-1.065×10^{-4}	-491.6	-6.700×10^{-5}	-269.9
$\mu_{y_{2,2}}$	0.006150	12.11	0.006526	6.735

Table 4.10.: Adjoint results computed for 16, 32 and 64 periods. Percent error is computed with FD from Table 4.8 as truth.

Entry	Adj $P = 16$	Error (%)	Adj $P = 32$	Error (%)	Adj $P = 64$	Error (%)
$\mu_{y_{0,0}}$	-0.01282	-1.827	-0.01271	-0.9243	-0.01265	-0.4864
$\mu_{y_{0,1}}$	-6.840×10^{-6}	110.3	2.973×10^{-5}	55.18	4.803×10^{-5}	27.59
$\mu_{y_{0,2}}$	0.01253	2.671	0.01270	1.325	0.01279	0.6371
$\mu_{y_{1,0}}$	-0.02178	0.5722	-0.02184	0.2786	-0.02188	0.1217
$\mu_{y_{1,1}}$	-7.448×10^{-5}	424.8	-2.577×10^{-5}	212.4	-1.395×10^{-6}	106.1
$\mu_{y_{1,2}}$	0.02127	3.899	0.02171	1.941	0.02193	0.9516
$\mu_{y_{2,0}}$	-0.007032	-0.1532	-0.007027	-0.08189	-0.007025	-0.05344
$\mu_{y_{2,1}}$	-4.285×10^{-5}	-136.6	-3.047×10^{-5}	-68.22	-2.427×10^{-5}	-33.92
$\mu_{y_{2,2}}$	0.006759	3.409	0.006879	1.698	0.006939	0.8348

Table 4.11.: Error norms for adjoint gradients computed for increasing numbers of periods.

P	Error norm
1	14.82
2	14.24
4	9.480
8	5.304
16	2.686
32	1.337
64	0.6562

As a brief verification that the obtained gradients are physical and suited for shape optimization, a single step was run using the gradient descent algorithm

$$\boldsymbol{\mu}_y^1 = \boldsymbol{\mu}_y^0 - s\frac{dI}{d\boldsymbol{\mu}_y^0}.$$

With the step size s arbitrarily set to $s = 5$. $\boldsymbol{\mu}_y^0$ and $\boldsymbol{\mu}_y^1$ denote the initial and updated optimization parameters, with $\boldsymbol{\mu}_y^0 = 0$. The gradient obtained with $P = 16$ is used. Figure 4.39 shows the resulting deformation: a slightly streamlined cylinder, as we expect. Table 4.12 shows the impact on the drag-related QoIs—all of which are significantly reduced—further validating the gradients.

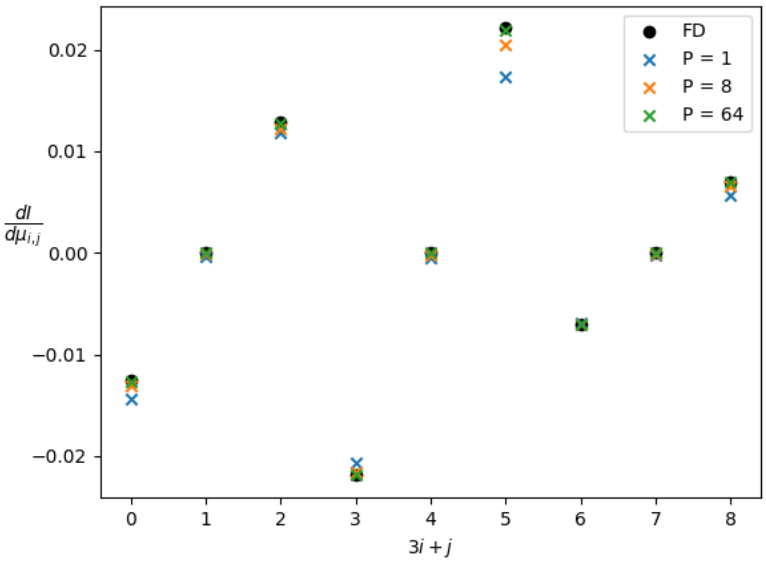


Figure 4.38.: Convergence to the finite difference gradient of adjoint-based gradients with increasing integration windows.

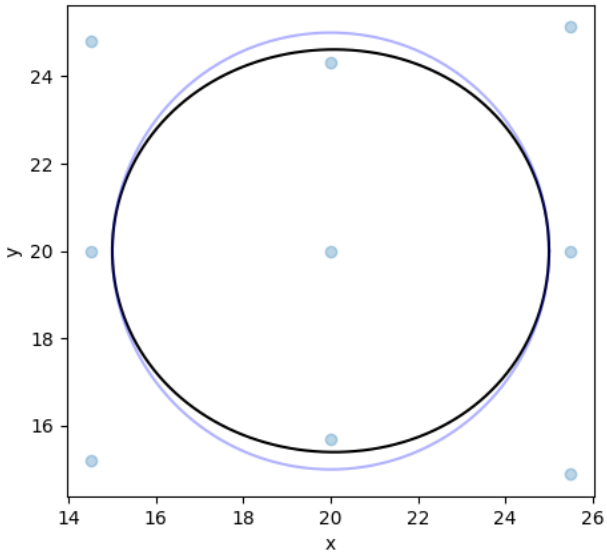


Figure 4.39.: Cylinder after one gradient descent step (black line) vs original shape (blue).

Table 4.12.: Evolution of drag-related QoIs after one gradient descent step for the cylinder case.

Quantity	Original value	Value after gradient descent step	% change
$C_{\overline{D}}$	3.402	2.911	-14.4
$C_{D_{max}}$	3.441	2.926	-15.0
Amplitude of C_D oscillations	0.066	0.030	-54.5

4.3. Preliminary extension to higher order boundary conditions

This section presents results that highlight how a high-order no-slip boundary condition can be incorporated into the adjoint pipeline. The objective is to answer research question 2: *How can adjoint boundary conditions be expressed in a general and systematic way that aids the extensibility of future solver developments?* A toy problem is employed using the second-order accurate interpolated bounce back (IBB) scheme by Bouzidi et al. [51]. This boundary condition is simple, but structurally equivalent to more promising alternatives currently under development [18] that seek second-order accuracy while satisfying conservation laws, thus overcoming the deficiencies of Bouzidi et al's. scheme (noted in section 2.3). By structurally equivalent, I mean that it is a linear combination of the post-collision distribution functions[15] that may be incorporated into the streaming matrix S — and that it shifts the impact of the optimization parameters from the collision operator (as in PBB) to the streaming matrix: $S(\boldsymbol{\mu})$.

The toy problem is an acoustic wave propagation case similar to those in [41] and [47]. In short, an acoustic source is placed inside a domain bounded by solid walls at the top and bottom (modelled with IBB), and extrapolation outlets at the left and right hand sides, from which acoustic waves (partially) reflect. The intensity of the sound is measured at another point. We are interested in evaluating the gradients of the measured sound with respect to the position of the walls.

The domain consists of 101 points in the x direction and 31 points in the y direction. A point acoustic source is placed at $x = 25$, $y = 28$. The acoustic source modifies the collision operator [41] following:

$$\hat{f}_{i_{acous}}(x = 25, y = 28, t_n) = \hat{f}_i(x = 25, y = 28, t_n) + w_i \rho_a \sin(2\pi t_n/10) \tag{4.3.1}$$

where $t_n \in [0, 1, ..., N-1]$, the frequency in inverse time lattice units is $1/10$, and $\rho_a = -0.05$. Initializing the states to equilibrium (with 0 velocity and constant density) we will observe acoustic waves (density fluctuations) propagating from the source, and interacting with the boundary conditions (see Figure 4.40).

The cost function is defined as the mean-square density evaluated at $x = 25$, $y = 30$:

$$I = \frac{1}{N} \sum_{n=0}^{N} \left[\frac{1}{2} \rho^2 (x = 25, y = 30, t_n) \right] \tag{4.3.2}$$

At the eastern and western edges the extrapolation boundary condition is applied. On the northern and southern edges (excluding points that overlap with the east and west boundaries) the Bouzidi et al. boundary condition is employed:

$$f_{i^-}(\mathbf{x}_k, t_{n+1}) = 2q_i \hat{f}_i(\mathbf{x}_k, t_n) + (1 - 2q_i) \hat{f}_i(\mathbf{x}_k - \boldsymbol{\xi}_i, t_n), \tag{4.3.3}$$

where $0 \leq (q_i = d/|\boldsymbol{\xi}_i|) \leq 1$ is the normalized distance to the wall along the vector $\boldsymbol{\xi}_i$, which, for a straight wall, is constant for all the boundary nodes. I assume that $q_i < 1/2$ (otherwise (4.3.3) must be modified) and set it to $q_i = q \in [0.05, 0.15, 0.25, 0.35, 0.45]$.

The combination of two non-local schemes (Bouzidi et al. and extrapolation) yields an interesting problem for the adjoint boundary conditions, as near the corners the IBB results in source terms for (some of) the nodes subject to the extrapolation and vice versa. To obtain a simple mapping between the forward and adjoint problems, the boundary conditions are programmed into the streaming matrix S.

We seek the gradient of I with respect to q to mimic a very simple shape optimization problem. Algorithm 2 is used for the gradient computation, where, given S, the adjoint

[15] This definition also includes the quadratic interpolation scheme proposed by Bouzidi et al. [51] (because the quadratic term is in the coefficients of the post-collision functions, not the distributions themselves) and multi-reflection boundary conditions, among others [12].

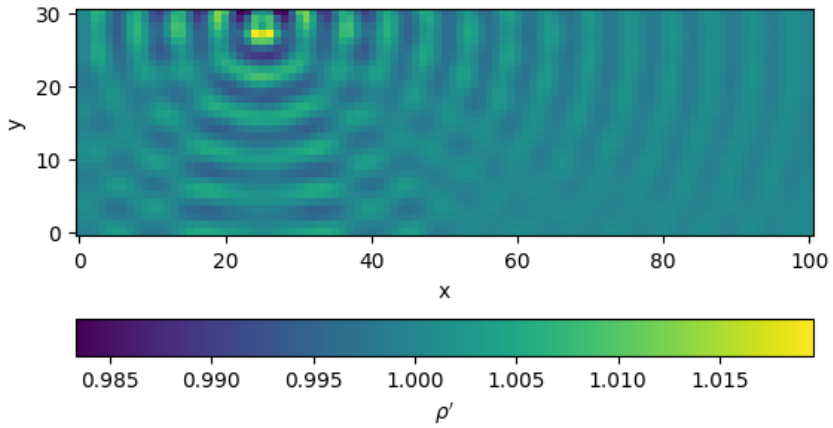


Figure 4.40.: Acoustic waves propagating throughout the domain, $t_n = 4078$.

boundary conditions are trivially and efficiently implemented by $S^\intercal$, with no need to bookkeep the source terms induced by the non-local schemes on boundary-adjacent nodes, much less the schemes' interaction near the corners. The program is run for a very long time, $N = 4900$, to fully account for the effect of the boundary conditions (i.e. multiple reflections from all the edges of the domain affect the cost function).

Since the added acoustic source is independent of the distribution functions, $\left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial F^n}\right)^\intercal$ stays the same. However, the adjoint source term must be modified to account for the new cost function. In local form, we obtain:

$$\frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_{n_0})} = \begin{cases} \rho(x = 25, y = 30, t_n), \text{if } \mathbf{x}_k = (25, 30) \text{ and } t_n = t_{n_0} \\ 0, \text{otherwise} \end{cases} \tag{4.3.4}$$

The expressions for $\frac{\partial \Phi}{\partial \boldsymbol{\mu}}$ from subsection 3.2.2 also no longer hold, as they assumed the optimization parameters propagated through the collision operator. Now, it is the entries of the streaming matrix that are functions of the optimization parameters: $S(q)$. Therefore,

$$\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}} = \frac{\partial \Phi^n}{\partial q} = \frac{\partial S}{\partial q} \cdot \mathscr{C}(\mathbf{F}) \tag{4.3.5}$$

where $\mathscr{C}$ now includes the added acoustic source. The derivative of the streaming matrix is readily available by differentiating the code that provided it.

The results of this test case are shown in Figure 4.41 and Figure 4.42. The error between finite difference and adjoint gradients is again extremely small (and decreases as q increases for some, as yet unknown, reason), once again validating the implementation.

Another conclusion to draw from this experiment, is that extending the solver requires modification only of the routine responsible for incorporating the boundary conditions into the forwrad streaming matrix. This design yields a highly modular framework. By contrast, in Kusano's locally-centred approach [47], the adjoint boundary conditions must be derived analytically, with a form tightly coupled to the structure of the forward residual. These conditions must then be implemented as separate routines, with thorny auxiliary logic to determine which degrees of freedom are influenced by non-local effects.

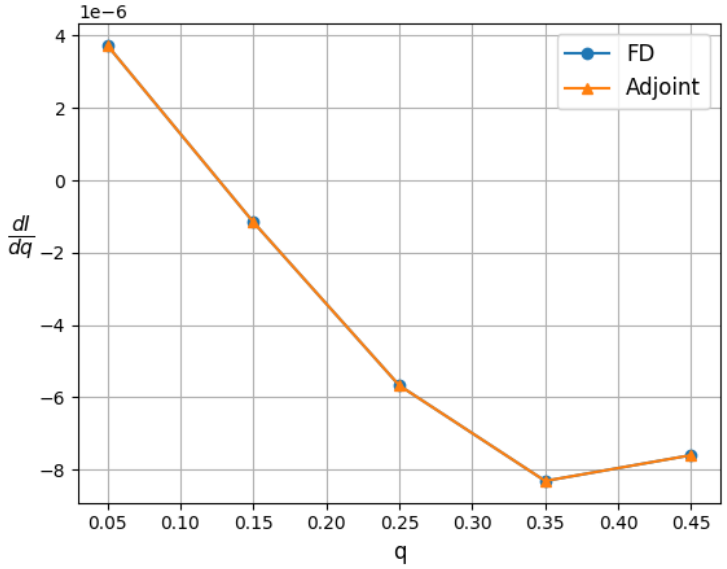


Figure 4.41.: FD vs adjoint based gradients for a range of q.

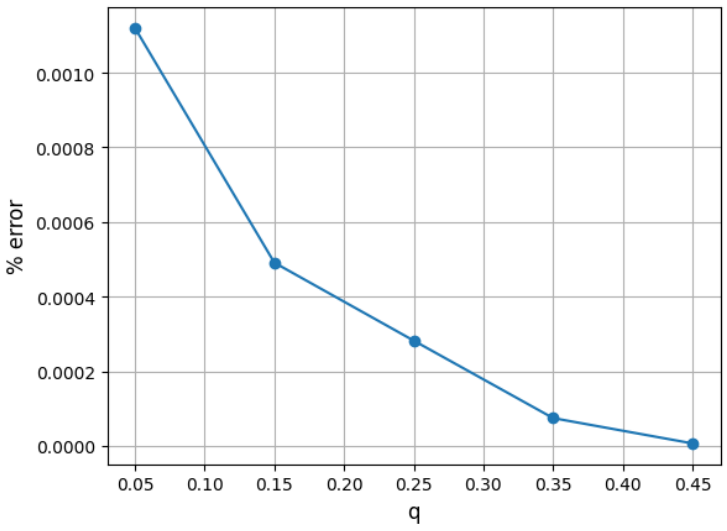


Figure 4.42.: % error between adjoint and FD gradients.

Chapter 5: Conclusions and Recommendations

To draw clear conclusions the research questions are restated below:

Main research question

What are the essential components of a discrete, general, and extensible adjoint LBM framework capable of industrially relevant shape optimization across a wide range of Reynolds numbers?

Research sub-questions

1. How can state of the art LBM features — specifically collision models, grid refinement strategies, and absorption regions — be incorporated into a discrete adjoint LBM? *This question addresses research gap 1 and requirements 1, 2 and 5.*
2. How can adjoint boundary conditions be expressed in a general and systematic way that aids the extensibility of future solver developments? *This question addresses research gap 2 and 3, and requirements 3 and 4.*
3. How can steady and unsteady adjoint-based gradients of the lift and drag coefficients be obtained within a robust, memory efficient, and accurate LBM framework? *This question addresses research gap 4 and 5, and requirements 6 and 8.*
4. In an LBM setting, how can a robust and flexible parametrization of the geometry be incorporated into an adjoint-based shape optimization pipeline? *This question addresses research gap 6, and requirement 10.*

The sub-questions were adressed as follows:

1. The recursive regularized (RR) collision model was incorporated into a discrete-adjoint LBM by differentiating it with respect to the distribution functions. *This is the first time an adjoint RR has been derived.* The resulting adjoint collision step remained a purely local operation. Similarly, the pressure relaxation sponge was incorporated as part of a modified collision operation. *No work has derived the adjoint of the pressure relaxation sponge to date.* It was observed that the sponge aids with the well-posedness of the steady problem (in the presence of a momentum inlet and extrapolation outlet) and is a necessary feature to dampen strong spurious pressure fluctuations at the outlet for the unsteady case.

   A volumetric grid refinement scheme was implemented into the adjoint framework through a transposition of its constitutive laws (i.e. it was found the adjoint algorithm has the same structure as the forward algorithm). *This is also a first-of-its kind implementation.* Methodological ambiguities concerning general shapes of embedded refined grids were addressed to ensure conservation laws. The implemented volumetrix scheme offers more robustness in terms of refinement geometry (which helps reduce computational cost at higher Reynolds numbers) and has previously been shown to reduce spurious artefacts at the grid interface compared to the interpolation scheme of [42], [47].

2. As opposed to the state-of-the-art shape optimization papers [42], [47], a discrete adjoint methodology originally proposed by Tekitek et al [1] was employed, offering a general treatment of boundary conditions. It was argued that this methodology is more consistent than the one currently employed and its extensibility to a high

order boundary condition with a general structure was demonstrated. Consistent inlet/outlet boundary conditions were proposed, representing an improvement over the ill-posed approach of the state-of-the-art.

A novel quantum-friendly boundary condition modifying the bounce back rule to account for smooth shapes, while conserving mass was proposed and validated. It can be understood as a layer of porous material coating the obstacle geometry, with the porosity dependent on the shape. Within the porous layer, a no-slip boundary condition is imposed using bounce back. The rule was found to be first order accurate but offers potential advantages over more commonly used porosity-based approaches. Specifically, it does not allow for pressure variations within the obstacle, potentially offering more accurate lift and drag predictions, particularly for unsteady flows. Based on this approach, another methodology was proposed to obtain accurate and physical gradients from the bounce back boundary condition. Preliminary results were encouraging as a steady optimization case (seemingly) converged to a local minimum. *This approach has successfully obtained non-trivial gradients from a piecewise constant function, and successfully used them to run an accurate optimization.*

3. Coefficients of lift and drag computed with the momentum exchange algorithm were incorporated into the adjoint pipeline for both steady and unsteady flow cases. *This is the first time lift has been considered as a cost function and the first time the momentum exchange algorithm has been differentiated.* A checkpointing routine was successfully implemented to reduce memory storage costs yielding computationally efficient and exact unsteady adjoint gradients. *This is one of only two works that address the high adjoint storage costs, and is the only that retains gradient accuracy.* Gradients were successfully validated against finite differences with better agreement than that reported in [42].
4. The free form deformation method was incorporated into the adjoint pipeline representing a smooth, and extensible (to 3D) shape parametrization which overcomes the differentiability issues of the cartesian coordinates in Cheylan et al. [42] and compares favorably to the spline-based approach of Kusano [47].

The main research question is answered by aggregating the subquestions, leading to the following assessment. Overall, a state-of-the-art adjoint solver is presented and tested successfully with a steady airfoil L/D optimization. The resulting shape has increased camber, as one would expect. An unsteady optimization is not run but its feasibility is tested by validating the gradients and running a single gradient-descent step.

Despite the significant progress made, I cannot fully argue that the solver is yet capable of industrially relevant shape optimization (i.e. 3D flows at high Reynolds numbers). The main aspects left to pursue are:

1. Extension of the features to 3D. This is primarily a programming challenge, although the RR operator will have to be adapted and re-differentiated. For affordable computations it is likely a high-order mass conserving scheme for the no-slip obstacle boundary, such as the one in development by [18], will have to be implemented.
2. Incorporation of a turbulence model and wall models.
3. Verification of gradients and successful optimizations.

However, these challenges can be faced standing on the contributions of this work namely:

1. A proof of concept that RR collision operators fit into the discrete adjoint methodology using a unified MRT formulation that is easily extensible.
2. Establishment of the adjoint methodology for the flexible and robust volumetric grid refinement scheme.

3. Shape parametrization with an intrinsically 3D method (FFD).
4. Implementation of checkpointing for affordable calculations, as well as a steady adjoint formulation (possibly for attached turbulent flows).
5. Establishment of the (adjoint) momentum exchange algorithm for robust surface force integration.
6. Presentation of a clear adjoint methodology for boundary conditions, resulting in easier extensibility for higher order schemes, and reducing the need for excessively large domains to deal with overdetermined systems.

The field of LBM based shape optimization is only just developing and presents very exciting opportunities. Using my code as it stands I recommend studying the low-Reynolds number airfoil—a complex system characterized by unsteady laminar separation and reattachment. This case is a nice fit for the RR-LBM with its wide stability range, unsteady time stepping scheme and limited numerical diffusion. It would pose challenges for the accuracy of the grid refinement scheme, which would need extensive validation. Furthermore, it would pose a challenge for the adjoint pipeline as it borders on a chaotic system, where gradient become unbounded.

Another possible application, with some development required, is to leverage the acoustic propagation properties of the LBM, combined with the accuracy afforded by the RR and volumetric grid refinement to run aeroacoustic optimization studies. Kusano has started to do this but with direct noise computation in the far field, and very expensive adjoint gradients. An interesting project would be to implement the Ffowcs-Williams-Hawkings (FWH) acoustic analogy into the adjoint pipeline (as a first!) and focus on spatial and temporal adjoint coarsening techniques for efficiency.

One final suggestion, with a personal connection, is to pioneer the field of adjoint based adaptive mesh refinement for the LBM. Even a simple method where the adjoint variable weights a heuristic refinement sensor (see the one in [14]) could offer valuable insights on the mesh geometry required for accurate QoI computations, and would put the general refinement capabilities of the solver to good use!. More ambitiously, a rigorous adjoint based error estimator could be derived. For this, I would recommend a starting with a laminar channel flow, as it has an analytical solution. However, care must be taken. Most goal-oriented error estimators involve a refinement of the solution space, typically the grid. However, in an LBM context, refining the grid always changes the form of the underlying DVBE, either by redefining the discrete velocities ξ_i (diffusive scaling), or the relaxation time τ (acoustic scaling). A fully discrete approach, as described in [6], seems most promising.

Bibliography

[1] M. Tekitek, M. Bouzidi, F. Dubois, and P. Lallemand, "Adjoint lattice Boltzmann equation for parameter identification," *Computers & Fluids*, vol. 35, no. 8, pp. 805–813, 2005, Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science, ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2005.07.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793005001428.

[2] J. Anderson, *Computational Fluid Dynamics* (Computational Fluid Dynamics: The Basics with Applications). McGraw-Hill Education, 1995, ISBN: 9780070016859. [Online]. Available: https://books.google.nl/books?id=dJceAQAAIAAJ.

[3] M. B. Giles and N. A. Pierce, "An introduction to the adjoint approach to design," *Flow, turbulence and combustion*, vol. 65, no. 3, pp. 393–415, 2000.

[4] D. A. Masters, N. J. Taylor, T. Rendall, C. B. Allen, and D. J. Poole, "Geometric comparison of aerofoil shape parameterization methods," *AIAA journal*, vol. 55, no. 5, pp. 1575–1589, 2017.

[5] J. R. Martins, "Aerodynamic design optimization: Challenges and perspectives," *Computers & Fluids*, vol. 239, p. 105 391, 2022, ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2022.105391. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793022000615.

[6] M. Nemec and M. Aftosmis, "Adjoint error estimation and adaptive refinement for embedded-boundary Cartesian meshes," in *18th AIAA computational fluid dynamics conference*, 2007, p. 4187.

[7] J. J. Reuther, A. Jameson, J. J. Alonso, M. J. Rimllnger, and D. Saunders, "Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 2," *Journal of aircraft*, vol. 36, no. 1, pp. 61–74, 1999.

[8] J. Elliott and J. Peraire, "Practical three-dimensional aerodynamic design and optimization using unstructured meshes," *AIAA journal*, vol. 35, no. 9, pp. 1479–1485, 1997.

[9] J. K. Elliott, "Aerodynamic optimization based on the Euler and Navier-Stokes equations using unstructured grids," Aerospace Computational Design Laboratory, Dept. of Aeronautics . . ., Tech. Rep., 1998.

[10] T. Dhert, T. Ashuri, and J. R. Martins, "Aerodynamic shape optimization of wind turbine blades using a Reynolds-averaged Navier–Stokes model and an adjoint method," *Wind Energy*, vol. 20, no. 5, pp. 909–926, 2017.

[11] G. L. Halila, J. R. Martins, and K. J. Fidkowski, "Adjoint-based aerodynamic shape optimization including transition to turbulence effects," *Aerospace Science and Technology*, vol. 107, p. 106 243, 2020.

[12] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. Viggen, *The Lattice Boltzmann Method: Principles and Practice* (Graduate Texts in Physics). Springer International Publishing, 2016, ISBN: 9783319446493. [Online]. Available: https://books.google.nl/books?id=S-d0DQAAQBAJ.

[13] M. Lăcătuş and M. Möller, "Surrogate quantum circuit design for the Lattice Boltzmann collision operator," *arXiv preprint arXiv:2507.12256*, 2025.

[14] C. Coreixas and J. Latt, *GPU-based compressible lattice Boltzmann simulations on non-uniform grids using standard c++ parallelism: From best practices to aerodynamics, aeroacoustics and supersonic flow simulations*, 2025. arXiv: 2504.04465 [physics.comp-ph]. [Online]. Available: https://arxiv.org/abs/2504.04465.

[15] S. Marié, D. Ricot, and P. Sagaut, "Comparison between lattice Boltzmann method and Navier–Stokes high order schemes for computational aeroacoustics," *Journal of Computational Physics*, vol. 228, no. 4, pp. 1056–1070, 2009.

[16] G. Wissocq, C. Coreixas, and J.-F. Boussuge, "Linear stability and isotropy properties of athermal regularized lattice Boltzmann methods," *Physical Review E*, vol. 102, no. 5, p. 053 305, 2020.

[17] A. Suss, I. Mary, T. Le Garrec, and S. Marié, "Comprehensive comparison between the lattice Boltzmann and Navier–Stokes methods for aerodynamic and aeroacoustic applications," *Computers & Fluids*, vol. 257, p. 105 881, 2023.
[18] K. Hoefnagel, D. Casalino, S. Hulshoff, and F. de Prenter, *A second-order volumetric boundary treatment for the lattice Boltzmann method*, 2025. arXiv: 2509.05035 [physics.comp-ph]. [Online]. Available: https://arxiv.org/abs/2509.05035.
[19] A. Rak, L. Grbčić, A. Sikirica, and L. Kranjčević, *Experimental and LBM analysis of medium-Reynolds number fluid flow around NACA0012 airfoil*, 2023. arXiv: 2302.04951 [physics.flu-dyn]. [Online]. Available: https://arxiv.org/abs/2302.04951.
[20] X.-P. Chen, "Applications of lattice Boltzmann method to turbulent flow around two-dimensional airfoil," *Engineering Applications of Computational Fluid Mechanics*, vol. 6, no. 4, pp. 572–580, 2012.
[21] C. M. Teixeira, "Incorporating turbulence models into the lattice-Boltzmann method," *International Journal of Modern Physics C*, vol. 9, no. 08, pp. 1159–1175, 1998.
[22] Z. Zhou, S. Moreau, and M. Sanjosé, "Installation effects on airfoil self-noise estimated by direct numerical simulations," *Journal of Sound and Vibration*, vol. 604, p. 118 978, 2025.
[23] A. F. Ribeiro and C. Muscari, "Sliding mesh simulations of a wind turbine rotor with actuator line lattice-Boltzmann method," *Wind Energy*, vol. 27, no. 11, pp. 1115–1129, 2024.
[24] C. Coreixas, G. Wissocq, B. Chopard, and J. Latt, "Impact of collision models on the physical properties and the stability of lattice Boltzmann methods," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2175, p. 20 190 397, 2020.
[25] T. Astoul, "Towards improved lattice Boltzmann aeroacoustic simulations with non-uniform grids: Application to landing gears noise prediction," Jul. 2021. DOI: 10.13140/RG.2.2.23640.01280.
[26] C. Coreixas, B. Chopard, and J. Latt, *Collision models in the lattice Boltzmann framework: Accuracy, stability, and performance comparisons on standard lattices*, Jul. 2021. DOI: 10.13140/RG.2.2.13457.35684/1.
[27] A. Schukmann, A. Schneider, V. Haas, and M. Böhle, "Analysis of hierarchical grid refinement techniques for the lattice Boltzmann method by numerical experiments," *Fluids*, vol. 8, no. 3, p. 103, 2023.
[28] G. Pingen, A. Evgrafov, and K. Maute, "Adjoint parameter sensitivity analysis for the hydrodynamic lattice Boltzmann method with applications to design optimization," *Computers & Fluids*, vol. 38, no. 4, pp. 910–923, 2009.
[29] G. Pingen, A. Evgrafov, and K. Maute, "A parallel schur complement solver for the solution of the adjoint steady-state lattice Boltzmann equations: Application to design optimisation," *International Journal of Computational Fluid Dynamics*, vol. 22, no. 7, pp. 457–464, 2008. DOI: 10.1080/10618560802238267. eprint: https://doi.org/10.1080/10618560802238267. [Online]. Available: https://doi.org/10.1080/10618560802238267.
[30] G. Pingen, A. Evgrafov, and K. Maute, "Topology optimization of flow domains using the lattice Boltzmann method," *Structural and Multidisciplinary Optimization*, vol. 34, pp. 507–524, Dec. 2007. DOI: 10.1007/s00158-007-0105-7.
[31] S. Kreissl, G. Pingen, and K. Maute, "An explicit level set approach for generalized shape optimization of fluids with the lattice Boltzmann method," *International Journal for Numerical Methods in Fluids*, vol. 65, no. 5, pp. 496–519, 2011. DOI: https://doi.org/10.1002/fld.2193. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2193. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2193.
[32] D. Makhija, G. Pingen, R. Yang, and K. Maute, "Topology optimization of multi-component flows using a multi-relaxation time lattice Boltzmann method," *Computers & Fluids*, vol. 67, pp. 104–114, 2012, ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2012.06.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793012002411.

[33] S. Kreissl, G. Pingen, and K. Maute, "Topology optimization for unsteady flow," *International Journal for Numerical Methods in Engineering*, vol. 87, no. 13, pp. 1229–1253, 2011.

[34] M. J. Krause, "Fluid flow simulation and optimisation with lattice Boltzmann methods on high performance computers - application to the human respiratory system," Ph.D. dissertation, 2010. DOI: 10.5445/IR/1000019768.

[35] M. J. Krause, G. Thäter, and V. Heuveline, "Adjoint-based fluid flow control and optimisation with lattice Boltzmann methods," *Computers & Mathematics with Applications*, vol. 65, no. 6, pp. 945–960, 2013, Mesoscopic Methods in Engineering and Science, ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2012.08.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0898122112005421.

[36] K. Yaji, T. Yamada, M. Yoshino, T. Matsumoto, K. Izui, and S. Nishiwaki, "Topology optimization using the lattice Boltzmann method incorporating level set boundary expressions," *Journal of Computational Physics*, vol. 274, pp. 158–181, 2014, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2014.06.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999114004112.

[37] C. Chen, K. Yaji, T. Yamada, K. Izui, and S. Nishiwaki, "Local-in-time adjoint-based topology optimization of unsteady fluid flows using the lattice Boltzmann method," *Mechanical Engineering Journal*, vol. 4, pp. 17–00 120, Jan. 2017. DOI: 10.1299/mej.17-00120.

[38] K. Yaji, T. Yamada, M. Yoshino, T. Matsumoto, K. Izui, and S. Nishiwaki, "Topology optimization in thermal-fluid flow using the lattice Boltzmann method," *Journal of Computational Physics*, vol. 307, pp. 355–377, 2016, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2015.12.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999115008244.

[39] J.-W. Luo, L. Chen, A. He, and W. Tao, "Topology optimization of convective heat transfer by the lattice Boltzmann method," *International Journal for Numerical Methods in Fluids*, vol. 95, no. 3, pp. 421–452, 2023.

[40] J.-W. Luo, L. Chen, K. Yaji, and W.-Q. Tao, "Improved adjoint lattice Boltzmann method for topology optimization of laminar convective heat transfer," *International Journal of Heat and Mass Transfer*, vol. 251, p. 127 315, 2025, ISSN: 0017-9310. DOI: https://doi.org/10.1016/j.ijheatmasstransfer.2025.127315. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0017931025006544.

[41] E. Vergnault and P. Sagaut, "An adjoint-based lattice Boltzmann method for noise control problems," *Journal of Computational Physics*, vol. 276, pp. 39–61, 2014, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2014.07.027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999114005208.

[42] I. Cheylan, G. Fritz, D. Ricot, and P. Sagaut, "Shape optimization using the adjoint lattice Boltzmann method for aerodynamic applications," *AIAA Journal*, vol. 57, no. 7, pp. 2758–2773, 2019. DOI: 10.2514/1.J057955. eprint: https://doi.org/10.2514/1.J057955. [Online]. Available: https://doi.org/10.2514/1.J057955.

[43] X. Li, L. Fang, and Y. Peng, "Airfoil design optimization based on lattice Boltzmann method and adjoint approach," *Applied Mathematics and Mechanics*, vol. 39, no. 6, pp. 891–904, 2018.

[44] H. Jalali Khouzani and R. Kamali Moghadam, "A novel approach of unsteady adjoint lattice Boltzmann method based on circular function scheme," *Journal of Scientific Computing*, vol. 85, no. 2, p. 38, 2020.

[45] H. Jalali Khouzani and R. Kamali-Moghadam, "Airfoil inverse design based on laminar compressible adjoint lattice Boltzmann method," *International Journal for Numerical Methods in Fluids*, vol. 95, no. 8, pp. 1197–1219, 2023.

[46] D. Lagrava, O. Malaspinas, J. Latt, and B. Chopard, "Advances in multi-domain lattice Boltzmann grid refinement," *Journal of Computational Physics*, vol. 231, no. 14, pp. 4808–4822, 2012.

[47] K. Kusano, "Adjoint sensitivity analysis method based on lattice Boltzmann equation for flow-induced sound problems," *Computers & Fluids*, vol. 248, p. 105 662,

2022, ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2022.105662. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793022002559.

[48] K. Kusano and H. Yamaguchi, "Adjoint-based shape optimization using lattice Boltzmann method for flow and sound control in tandem cylinders," *Journal of Fluids and Structures*, vol. 135, p. 104 308, 2025, ISSN: 0889-9746. DOI: https://doi.org/10.1016/j.jfluidstructs.2025.104308. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S088997462500043X.

[49] K. Kusano, "Development of aeroacoustic control devices for a square cylinder using adjoint lattice Boltzmann method," *European Journal of Mechanics - B/Fluids*, vol. 116, p. 204 413, 2026, ISSN: 0997-7546. DOI: https://doi.org/10.1016/j.euromechflu.2025.204413. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0997754625001943.

[50] S. Nørgaard, O. Sigmund, and B. Lazarov, "Topology optimization of unsteady flow problems using the lattice Boltzmann method," *Journal of Computational Physics*, vol. 307, pp. 291–307, 2016, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2015.12.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999115008426.

[51] M. Bouzidi, M. Firdaouss, and P. Lallemand, "Momentum transfer of a Boltzmann-lattice fluid with boundaries," *Physics of fluids*, vol. 13, no. 11, pp. 3452–3459, 2001.

[52] M. A. Schalkers and M. Möller, "Momentum exchange method for quantum Boltzmann methods," *Computers & Fluids*, vol. 285, p. 106 453, 2024.

[53] Q. Wang, P. Moin, and G. Iaccarino, "Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2549–2567, 2009.

[54] J. Samareh, "A survey of shape parametrization techniques, intern," in *Forum on Aeroelasticity and Structural Dynamics Conf*, 1999.

[55] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, "Benchmark computations of laminar flow around a cylinder," in *Flow simulation with high-performance computers II: DFG priority research programme results 1993–1995*, Springer, 1996, pp. 547–566.

[56] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/jax-ml/jax.

[57] M. Rohde, D. Kandhai, J. J. Derksen, and H. E. Van den Akker, "A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes," *International journal for numerical methods in fluids*, vol. 51, no. 4, pp. 439–468, 2006.

[58] D. Yu, R. Mei, and W. Shyy, "A unified boundary treatment in lattice Boltzmann method," in *41st aerospace sciences meeting and exhibit*, 2003, p. 953.

[59] I. Ginzburg, F. Verhaeghe, and D. d'Humieres, "Two-relaxation-time lattice Boltzmann scheme: About parametrization, velocity, pressure and mixed boundary conditions," *Communications in computational physics*, vol. 3, no. 2, pp. 427–478, 2008.

[60] G. Wissocq, P. Sagaut, and J.-F. Boussuge, "An extended spectral analysis of the lattice Boltzmann method: Modal interactions and stability issues," *Journal of Computational Physics*, vol. 380, pp. 311–333, 2019.

[61] C. Coreixas, G. Wissocq, G. Puigt, J.-F. Boussuge, and P. Sagaut, "Recursive regularization step for high-order lattice Boltzmann methods," *Physical Review E*, vol. 96, no. 3, Sep. 2017, ISSN: 2470-0053. DOI: 10.1103/physreve.96.033306. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.96.033306.

[62] M. A. Spaid and F. R. Phelan Jr, "Lattice Boltzmann methods for modeling microscale flow in fibrous porous media," *Physics of fluids*, vol. 9, no. 9, pp. 2468–2474, 1997.

[63] S. K. Sanjeevi, A. Zarghami, and J. T. Padding, "Choice of no-slip curved boundary condition for lattice Boltzmann simulations of high-Reynolds-number flows," *Physical Review E*, vol. 97, no. 4, p. 043 305, 2018.
[64] D. Casalino, A. Hazir, and A. Mann, "Turbofan broadband noise prediction using the lattice Boltzmann method," *AIAA Journal*, vol. 56, no. 2, pp. 609–628, 2018.
[65] D. Casalino, G. Romani, R. Zhang, and H. Chen, "Lattice-Boltzmann calculations of rotor aeroacoustics in transitional boundary layer regime," *Aerospace Science and Technology*, vol. 130, p. 107 953, 2022.
[66] C. Teruna, F. Avallone, D. Casalino, and D. Ragni, "Numerical investigation of leading edge noise reduction on a rod-airfoil configuration using porous materials and serrations," 2021.
[67] K. Kusano, "Development of aeroacoustic control devices for a square cylinder using adjoint lattice Boltzmann method," *European Journal of Mechanics-B/Fluids*, p. 204 413, 2025.
[68] A. Schukmann, V. Haas, and A. Schneider, "Spurious aeroacoustic emissions in lattice Boltzmann simulations on non-uniform grids," *Fluids*, vol. 10, no. 2, pp. 1–41, 2025.
[69] H. Chen, O. Filippova, J. Hoch, *et al.*, "Grid refinement in lattice Boltzmann methods based on volumetric formulation," *Physica A: Statistical Mechanics and its Applications*, vol. 362, no. 1, pp. 158–167, 2006, Discrete Simulation of Fluid Dynamics, ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2005.09.036. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437105009696.
[70] F. Schornbaum, "Block-structured adaptive mesh refinement for simulations on extreme-scale supercomputers," Ph.D. dissertation, Dissertation, Erlangen, Friedrich-Alexander-Universität Erlangen-Nürnberg ..., 2018.
[71] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 151–160.
[72] C. Lee, D. Koo, and D. W. Zingg, "Comparison of B-spline surface and free-form deformation geometry control for aerodynamic optimization," *AIAA Journal*, vol. 55, no. 1, pp. 228–240, 2017.
[73] M. Rohde, "Extending the lattice-Boltzmann method - novel techniques for local grid refinement and boundary conditions," Dissertation (TU Delft), Delft University of Technology, 2004, ISBN: 90-6464-113-7.
[74] G. Liu, M. Geier, Z. Liu, M. Krafczyk, and T. Chen, "Discrete adjoint sensitivity analysis for fluid flow topology optimization based on the generalized lattice Boltzmann method," *Computers & Mathematics with Applications*, vol. 68, no. 10, pp. 1374–1392, 2014.

Appendix A: Algorithm for time-stepping on multiple refinement levels

This appendix presents the algorithm developed to extend Rohde et al.'s mesh refinement scheme to multiple refinement levels in a recursion-free manner. The time stepping scheme begins with a collide-then-stream iteration on all the grids (lines 2-10). After each collision, an explosion step transfers information from the parent coarse grid to the child fine grid. Then, the while loop handles the bulk of the iterative time-stepping based on the following rules:

- If a fine grid, i, lags behind its parent grid, $i-1$, it must catch up by performing an appropriate number of collide-then-stream steps.
- When i catches up to $i-1$, it transfers information to the parent via a coalescence step. The grid under consideration changes to the parent, $i-1$.
- If i runs ahead of its child grid, $i+1$, it bequeaths information via an explosion step. The grid under consideration changes to the child, $i+1$.
- The loop ends after the coalescence step from grid 1 to grid 0.

Algorithm 10 Multi-level single coarse timestep for Rohde et al.'s mesh refinement scheme

Initialize `counters`$[i] \leftarrow 0$ for all $i = 0, \ldots, N_{levs}$
▷ Initial collide on all levels
for $i = 0$ to $i = N_{levs}$ **do**
    COLLIDE(i)
end for
▷ Initial explode from coarse to fine grids
for $i = 0$ to $N_{levs} - 1$ **do**
    EXPLODE($i \rightarrow i + 1$)
end for
▷ Initial stream on all levels
for $i = 0$ to N_{levs} **do**
    STREAM(i)
    `counters`$[i] \leftarrow$ `counters`$[i] + 2^{N_{levs}-i}$
end for

$i \leftarrow N_{levs}$
`exp_check` $\leftarrow$ False
▷ Main refinement loop
while $i > 0$ **do**
    if `counters`$[i] <$ `counters[i-1]` **then**
        COLLIDE(i)
        if exp_check **then**
            EXPLODE($i - 1 \rightarrow i$)
        end if
        STREAM(i)
        `counters`$[i] \leftarrow$ `counters`$[i] + 2^{N_{levs}-i}$
    end if
    coal_check $\leftarrow$ (`counters[i]` $==$ `counters[i-1]`)
    if $i == N_{levs}$ **then**
        exp_check $\leftarrow$ False
    else
        exp_check $\leftarrow$ (`counters[i]` $>$ `counters[i+1]`)
    end if
    if exp_check **then**
        $i \leftarrow i + 1$
    else if coal_check **then**
        COALESCE($i \rightarrow i - 1$)
        $i \leftarrow i - 1$
    end if
end while

Appendix B: Recursive regularized collision operator with porosity and pressure relaxation

Considering the PBB boundary condition and pressure relaxation sponge presented in section 2.3 and subsection 2.3.3, the post-collision distribution functions can be written for a general node as:

$$\hat{f}_j(\mathbf{x}_k, t_n) = \hat{f}_j(\rho_s, \hat{\mathbf{u}} = \varepsilon\mathbf{u}, \mathbf{f}, \mathbf{x}_k, t_n) \tag{B.0.1}$$

where $\rho_s = \rho(1-\sigma) + \sigma\rho_\infty$ and

$$\hat{\mathbf{f}} = [\hat{f}_j, j \in (0, ..., 8)] = T \cdot R \cdot \mathcal{C}(\rho_s, \hat{\mathbf{u}}, \mathbf{f}) \tag{B.0.2}$$

$$\mathcal{C}(\rho_s, \hat{\mathbf{u}}, \mathbf{f}) = \hat{\mathbf{A}} = \begin{cases} \hat{A}_{00} & = \rho_s, \\ \hat{A}_{10} & = \rho_s \hat{u}_x, \\ \hat{A}_{01} & = \rho_s \hat{u}_y, \\ \hat{A}_{11} & = (1-\omega_2)A_{11}^{\text{neq}} + A_{11}^{\text{eq}}, \\ \hat{A}_{20} & = A_{20} - \omega_+ A_{20}^{\text{neq}} - \omega_- A_{02}^{\text{neq}}, \\ \hat{A}_{02} & = A_{02} - \omega_- A_{20}^{\text{neq}} - \omega_+ A_{02}^{\text{neq}}, \\ \hat{A}_{21} & = (1-\omega_3)A_{21}^{\text{neq}} + A_{21}^{\text{eq}}, \\ \hat{A}_{12} & = (1-\omega_3)A_{12}^{\text{neq}} + A_{12}^{\text{eq}}, \\ \hat{A}_{22} & = (1-\omega_4)A_{22}^{\text{neq}} + A_{22}^{\text{eq}}. \end{cases} \tag{B.0.3}$$

$$\begin{cases} A_{20}^{\text{neq}} & = A_{20} - A_{20}^{\text{eq}}, \\ A_{02}^{\text{neq}} & = A_{02} - A_{02}^{\text{eq}}, \\ A_{11}^{\text{neq}} & = A_{11} - A_{11}^{\text{eq}}, \\ A_{21}^{\text{neq}} & = \hat{u}_y A_{20}^{\text{neq}} + 2\hat{u}_x A_{11}^{\text{neq}}), \\ A_{12}^{\text{neq}} & = \hat{u}_x A_{02}^{\text{neq}} + 2\hat{u}_y A_{11}^{\text{neq}}, \\ A_{22}^{\text{neq}} & = \hat{u}_y^2 A_{20}^{\text{neq}} + \hat{u}_x^2 A_{02}^{\text{neq}} + 4\hat{u}_x\hat{u}_y A_{11}^{\text{neq}}), \end{cases} \tag{B.0.4}$$

$$\begin{cases} A_{11} & = \sum_i \xi_{i_y}\xi_{i_y} f_i, \\ A_{20} & = \sum_i (\xi_{i_x}^2 - c_s^2) f_i, \\ A_{02} & = \sum_i (\xi_{i_y}^2 - c_s^2) f_i. \end{cases} \tag{B.0.5}$$

$$\begin{cases} A_{11}^{\text{eq}} & = \rho_s \hat{u}_x \hat{u}_y, \\ A_{20}^{\text{eq}} & = \rho_s \hat{u}_x^2, \\ A_{02}^{\text{eq}} & = \rho_s \hat{u}_y^2, \\ A_{21}^{\text{eq}} & = A_{20}^{\text{eq}} \hat{u}_y, \\ A_{12}^{\text{eq}} & = A_{02}^{\text{eq}} \hat{u}_x, \\ A_{22}^{\text{eq}} & = \frac{A_{20}^{\text{eq}} A_{02}^{\text{eq}}}{\rho_s} = \rho_s \hat{u}_x^2 \hat{u}_y^2. \end{cases} \tag{B.0.6}$$

ε and σ follow:

$$\varepsilon = \begin{cases} \varepsilon(d(\boldsymbol{\mu})), \mathbf{x}_k \in X_b \\ 1, \text{otherwise} \end{cases} \tag{B.0.7}$$

$$\sigma = \begin{cases} \sigma_{max}(x - x_{min})/(x_{max} - x_{min}),\ x \geq x_{min} \\ \sigma_{max}\ x \geq x_{max} \\ 0, \text{otherwise.} \end{cases} \tag{B.0.8}$$

The LBM loop can be implemented as

Algorithm 11 Forward LBM simulation

1: Define the initial macroscopic fields: $\rho_{init} = \rho_\infty = 1$, $\mathbf{u}_{init} = 0$.
2: Compute $\hat{\mathbf{u}} = \varepsilon \mathbf{u}_{init}$, $\rho_s = \rho_{init}(1-\sigma) + \sigma\rho_\infty$
3: Initialize the distribution functions.
4: **for** $n = 0$ to $n = N - 1$ **do**
5: $\quad \hat{f}_j(\mathbf{x}_k, t_n) = \hat{f}_j(\rho_s, \hat{\mathbf{u}}, \mathbf{f}, \mathbf{x}_k, t_n)$
6: $\quad f_j(\mathbf{x}_k + \boldsymbol{\xi}_i\, t_{n+1}) = \hat{f}_j(\mathbf{x}_k, t_n)$
7: $\quad \rho = \sum_j f_j$, $\mathbf{u} = \frac{1}{\rho}\sum_j \boldsymbol{\xi}_j f_j$
8: $\quad \hat{\mathbf{u}} = \varepsilon \mathbf{u}$, $\rho_s = \rho(1-\sigma) + \sigma\rho_\infty$
9: **end for**

Derivative with respect to the state

The global term $\left(\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial \mathbf{F}^n}\right)^\intercal$, expressed locally as $\frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n)$, is given by:

$$\partial_{f_i}\hat{\mathbf{f}} = (T \cdot R \cdot \partial_{f_i}\mathcal{C}(\mathbf{f}))^\intercal \tag{B.0.9}$$

$$\partial_{f_i}\mathcal{C}(\mathbf{f}) = \begin{cases} \partial_{f_i}\hat{A}_{00} &= \partial_{f_i}\rho_s, \\ \partial_{f_i}\hat{A}_{10} &= \hat{u}_x\partial_{f_i}\rho_s + \rho_s\partial_{f_i}\hat{u}_x, \\ \partial_{f_i}\hat{A}_{01} &= \hat{u}_y\partial_{f_i}\rho_s + \rho_s\partial_{f_i}\hat{u}_y, \\ \partial_{f_i}\hat{A}_{11} &= (1-\omega_2)\partial_{f_i}A^{\text{neq}}_{11} + \partial_{f_i}A^{\text{eq}}_{11}, \\ \partial_{f_i}\hat{A}_{20} &= \partial_{f_i}A_{20} - \omega_+\partial_{f_i}A^{\text{neq}}_{20} - \omega_-\partial_{f_i}A^{\text{neq}}_{02}, \\ \partial_{f_i}\hat{A}_{02} &= \partial_{f_i}A_{02} - \omega_-\partial_{f_i}A^{\text{neq}}_{20} - \omega_+\partial_{f_i}A^{\text{neq}}_{02}, \\ \partial_{f_i}\hat{A}_{21} &= (1-\omega_3)\partial_{f_i}A^{\text{neq}}_{21} + \partial_{f_i}A^{\text{eq}}_{21}, \\ \partial_{f_i}\hat{A}_{12} &= (1-\omega_3)\partial_{f_i}A^{\text{neq}}_{12} + \partial_{f_i}A^{\text{eq}}_{12}, \\ \partial_{f_i}\hat{A}_{22} &= (1-\omega_4)\partial_{f_i}A^{\text{neq}}_{22} + \partial_{f_i}A^{\text{eq}}_{22}. \end{cases} \tag{B.0.10}$$

$$\begin{cases} \partial_{f_i}A^{\text{neq}}_{20} &= \partial_{f_i}A_{20} - \partial_{f_i}A^{\text{eq}}_{20}, \\ \partial_{f_i}A^{\text{neq}}_{02} &= \partial_{f_i}A_{02} - \partial_{f_i}A^{\text{eq}}_{02}, \\ \partial_{f_i}A^{\text{neq}}_{11} &= \partial_{f_i}A_{11} - \partial_{f_i}A^{\text{eq}}_{11}, \\ \partial_{f_i}A^{\text{neq}}_{21} &= A^{\text{neq}}_{20}\partial_{f_i}\hat{u}_y + \hat{u}_y\partial_{f_i}A^{\text{neq}}_{20} + 2(\hat{u}_x\partial_{f_i}A^{\text{neq}}_{11} + A^{\text{neq}}_{11}\partial_{f_i}\hat{u}_x), \\ \partial_{f_i}A^{\text{neq}}_{12} &= \hat{u}_x\partial_{f_i}A^{\text{neq}}_{02} + A^{\text{neq}}_{02}\partial_{f_i}\hat{u}_x + 2(\hat{u}_y\partial_{f_i}A^{\text{neq}}_{11} + A^{\text{neq}}_{11}\partial_{f_i}\hat{u}_y), \\ \partial_{f_i}A^{\text{neq}}_{22} &= \hat{u}_y^2\partial_{f_i}A^{\text{neq}}_{20} + 2\hat{u}_yA^{\text{neq}}_{20}\partial_{f_i}\hat{u}_y + \hat{u}_x^2\partial_{f_i}A^{\text{neq}}_{02} + 2\hat{u}_xA^{\text{neq}}_{02}\partial_{f_i}\hat{u}_x \\ & \quad +4(\hat{u}_x\hat{u}_y\partial_{f_i}A^{\text{neq}}_{11} + A^{\text{neq}}_{11}(\hat{u}_x\partial_{f_i}\hat{u}_y + \hat{u}_y\partial_{f_i}\hat{u}_x)). \end{cases} \tag{B.0.11}$$

$$\begin{cases} \partial_{f_i}A_{11} &= \xi_{i_y}\xi_{i_y}, \\ \partial_{f_i}A_{20} &= (\xi^2_{i_x} - c_s^2), \\ \partial_{f_i}A_{02} &= (\xi^2_{i_y} - c_s^2). \end{cases} \tag{B.0.12}$$

$$
\begin{cases}
\partial_{f_i} A_{11}^{\mathrm{eq}} = (\partial_{f_i}\rho_s)\,\hat{u}_x\hat{u}_y + \rho_s\big(\hat{u}_y\,\partial_{f_i}\hat{u}_x + \hat{u}_x\,\partial_{f_i}\hat{u}_y\big), \\
\partial_{f_i} A_{20}^{\mathrm{eq}} = (\partial_{f_i}\rho_s)\,\hat{u}_x^2 + 2\rho_s\,\hat{u}_x\,\partial_{f_i}\hat{u}_x, \\
\partial_{f_i} A_{02}^{\mathrm{eq}} = (\partial_{f_i}\rho_s)\,\hat{u}_y^2 + 2\rho_s\,\hat{u}_y\,\partial_{f_i}\hat{u}_y, \\
\partial_{f_i} A_{21}^{\mathrm{eq}} = (\partial_{f_i} A_{20}^{\mathrm{eq}})\,\hat{u}_y + A_{20}^{\mathrm{eq}}\,\partial_{f_i}\hat{u}_y, \\
\partial_{f_i} A_{12}^{\mathrm{eq}} = (\partial_{f_i} A_{02}^{\mathrm{eq}})\,\hat{u}_x + A_{02}^{\mathrm{eq}}\,\partial_{f_i}\hat{u}_x, \\
\partial_{f_i} A_{22}^{\mathrm{eq}} = (\partial_{f_i}\rho_s)\,\hat{u}_x^2\hat{u}_y^2 + 2\rho_s\,\hat{u}_x\,\hat{u}_y^2\,\partial_{f_i}\hat{u}_x + 2\rho_s\,\hat{u}_x^2\,\hat{u}_y\,\partial_{f_i}\hat{u}_y.
\end{cases}
\tag{B.0.13}
$$

where

$$
\partial_{f_i}\hat{\mathbf{u}} = \frac{1-\sigma}{\rho_s - \sigma\rho_\infty}\,(\varepsilon\boldsymbol{\xi}_i - \hat{\mathbf{u}})
\tag{B.0.14}
$$

and

$$
\partial_{f_i}\rho_s = 1 - \sigma.
\tag{B.0.15}
$$

Appendix C: Comparison between Tekitek et al.'s and Vergnault and Sagaut's adjoint formulations

The papers on LBM gradient based aerodynamic shape optimization derive a discerete adjoint LBM using the method proposed by Vergnault and Sagaut. Rather than consider the flattened vector of state variables $\mathbf{F}^n$ (as Tekitek et al. do), they derive an adjoint equation for each individual node[1]. This approach leads to an identical adjoint evolution algorithm as Tekitek et al. (after a small modification to the method, and with the exception that the terminal condition is left unspecified), but a different gradient equation. In this appendix, I show what the necessary terminal condition is, and why Vergnault and Sagaut's proposed gradient equation only holds if the optimization parameters affect the collision operator alone, making the method generally inconsistent.

Firstly, we must note that a given set of (non-)linear algebraic equations that define the evolution of a state variable from which a scalar cost function is computed (as arise from the discretization of a PDE), lead to a unique linear system defining an adjoint variable. Since both Tekitek et al. and Vergnault and Sagaut deal with exactly the same discretization of the DVBE—and treat general cost functions defined in the same way (as time averages over the same interval)—their proposed adjoint and gradient equations must be, ultimately, identical. With this in mind, and noting Tekitek et al.'s derivation is completely determined whereas Vergnault and Sagaut's leaves the terminal condition unspecified, the following discussion shows how the latter method can be modified to become a consistent discrete-adjoint LBM, given a posteriori knowledge of Tekitek et al.'s approach.

For a node in the bulk, sufficiently far removed from any boundaries, the adjoint Lattice Boltzmann Method as derived by Tekitek et al. is:

$$\begin{cases} f_i^{*,s}(\mathbf{x}_k, t_{N+1}) = 0 \\ f_i^{*}(\mathbf{x}_k, t_n) = \sum_{j=0}^{q-1} f_j^{*,s}(\mathbf{x}_k, t_{n+1}) \frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n) - \frac{1}{N} \frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)} \\ f_i^{*,s}(\mathbf{x}_k - \boldsymbol{\xi}_i, t_n) = f_i^{*}(\mathbf{x}_k, t_n), n = N, N-1, ..., 1 \end{cases} \tag{C.0.1}$$

where we follow the initialization procedure presented in section 3.3 to make the method collide-then-stream (as opposed to the nominal stream-then-collide implementation). Here f^* is the adjoint variable and $f^{*,s}$ the adjoint post-streaming populations.

Assuming that the cost function has no explicit dependence on the optimization parameters:

$$\frac{\partial I}{\partial \boldsymbol{\mu}} = 0,$$

the gradient is given by:

$$\frac{dI}{d\boldsymbol{\mu}} = -\sum_{i=0}^{q-1} \sum_{k=0}^{M-1} \sum_{n=0}^{N-1} f_i^*(\mathbf{x}_k, t_{n+1}) \frac{\partial \phi_{ki}(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}. \tag{C.0.2}$$

where ϕ_{ki} is the $k \times i$-th row of the vector

$$\Phi^n = S\mathscr{C}(\mathbf{F}^n).$$

Vergnault and Sagaut's adjoint equation reads:

[1] The method is well detailed in [41].

$$\begin{cases} \hat{f}_i^*(\mathbf{x}_k, t_{n+1}) = \sum_{j=0}^{q-1} f_j^{*,V}(\mathbf{x}_k, t_{n+1}) \frac{\partial \hat{f}_j}{\partial f_i}(\mathbf{x}_k, t_n) - \frac{1}{N} \frac{\partial \mathcal{I}^n}{\partial f_i(\mathbf{x}_k, t_n)} \\ f_i^{*,V}(\mathbf{x}_k - \boldsymbol{\xi}_i, t_n) = \hat{f}_i^*(\mathbf{x}_k, t_{n+1}), n = N-1, ..., 0. \end{cases} \tag{C.0.3}$$

There are three differences with (C.0.1). Firstly, the terminal condition is not specified. Secondly, the method is expressed in terms of the adjoint variable $f^{*,V}$ (after Vergnault) and the adjoint post-collision populations $\hat{f}^*$. Thirdly, the algorithm runs from $n = N-1$ to $n = 0$.

To make (C.0.1) and (C.0.3) match, it is only necessary to make the following symbolic reinterpretations

$$\begin{cases} f^{*,V}(\mathbf{x}_k, t_n) = f^{*,s}(\mathbf{x}_k, t_n) \\ \hat{f}^*(\mathbf{x}_k, t_{n+1}) = f^*(\mathbf{x}_k, t_n) \end{cases}$$

and use the terminal condition

$$f^{*,V}(\mathbf{x}_k, t_{N+1}) = 0.$$

The method is then exactly equal to (C.0.1). Note that this means the adjoint variables proposed by Tekitek et al. and Vergnault and Sagaut are different, separated by a streaming step. Given that Tekitek et al.'s derivation is fully consistent, I suggest taking f^* as the correct adjoint variable.

Vergnault and Sagaut's proposed gradient equation is

$$\frac{dI}{d\boldsymbol{\mu}} = \sum_{i=0}^{q-1} \sum_{k=0}^{M-1} \sum_{n=0}^{N-1} f_i^{*,V}(\mathbf{x}_k, t_{n+1}) \frac{\partial R_i(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}, \tag{C.0.4}$$

where R is the residual of the LBM (in local form, for a node in the bulk):

$$R_i(\mathbf{x}_k, t_n) = f_i(\mathbf{x}_k + \boldsymbol{\xi}_j \, t_{n+1}) - \hat{f}_i(\mathbf{x}_k, t_n) = 0. \tag{C.0.5}$$

Given that both gradient equations run for the same number of timesteps, Vergnault and Sagaut's and Tekitek et al.'s methods can only be equivalent (as they must be, given the uniqueness of the discrete adjoint) if the following equality holds

$$f_i^{*,V}(\mathbf{x}_k, t_{n+1}) \frac{\partial R_i(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}} = -f_i^*(\mathbf{x}_k, t_{n+1}) \frac{\partial \phi_{ki}(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}. \tag{C.0.6}$$

To verify this, we consider two cases. Firstly, we assume the optimization parameters affect the collision operator, $\hat{f}_i(\boldsymbol{\mu})$. For simplicity, we restrict ourselves to a node removed from any boundaries (i.e. nodes where the streaming operation is altered). This is the case for the porous node treatment of Pingen et al., for example. Then,

$$\frac{\partial R_i(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}} = -\frac{\partial \hat{f}_i}{\partial \boldsymbol{\mu}} \tag{C.0.7}$$

which can be rewritten globally as:

$$\frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}} = -\frac{\partial \mathscr{C}}{\partial \boldsymbol{\mu}} \tag{C.0.8}$$

where $\frac{\partial \mathscr{C}}{\partial \boldsymbol{\mu}}$ is only non-zero for points where the collision operator is modified.

Continuing, the global representation of $\frac{\partial \phi_{ki}(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}}$ is:

$$\frac{\partial \Phi}{\partial \boldsymbol{\mu}} = S \frac{\partial \mathscr{C}}{\partial \boldsymbol{\mu}}, \tag{C.0.9}$$

which reduces (C.0.6) to:

$$\mathbf{F}^{*,V} = \left(\mathbf{F}^{*\intercal} S\right)^{\intercal} = S^{\intercal}\mathbf{F}^{*}. \tag{C.0.10}$$

This is clearly true because we define $f^{*,V}$ as the post-streaming adjoint populations (exactly $S^{\intercal}\mathbf{F}^{*}$). Therefore, if the optimization parameters propagate into the flow field via the collision operator, the method by Vergnault and Sagaut is equivalent to Tekitek et al.'s and thus, for this case, consistent.

The second case assumes that the optimization parameters affect the streaming matrix: $S(\boldsymbol{\mu})$. We restrict ourselves to linear boundary conditions. This occurs, for example, for an interpolated bounce back boundary condition such as in section 4.3. For this case, the local LBM residual is:

$$R_i(\mathbf{x}_k, t_n) = f_i(\mathbf{x}_k, t_{n+1}) - \mathscr{F}(\hat{f}_(\mathbf{x}_k, t_n)) = 0. \tag{C.0.11}$$

where $\mathscr{F}(\hat{f}_(\mathbf{x}_k, t_n))$ is a general linear combination of a certain set of post-collision distribution functions. Importantly, this can be written globally as

$$\mathbf{R}^n = \mathbf{F}^{n+1} - S\mathscr{C}(\mathbf{F}^n) = \mathbf{F}^{n+1} - \Phi^n(\mathbf{F}^n)$$

where $R_i(\mathbf{x}_k)$ will be the $i \times k$-th row of the resulting vector. This means

$$\frac{\partial R_i(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}} = -\frac{\partial \phi_{ki}(\mathbf{x}_k, t_n)}{\partial \boldsymbol{\mu}} \tag{C.0.12}$$

which would require

$$f_i^*(\mathbf{x}_k, t_{n+1}) = f_i^{*,V}(\mathbf{x}_k, t_{n+1}). \tag{C.0.13}$$

This cannot hold as the two adjoint variables differ by a streaming operation. Thus, Vergnault and Sagaut's gradient equation is incorrect if the optimization parameters affect the streaming matrix, and his method does not yield a consistent discrete-adjoint LBM. To obtain accurate gradients using Vergnault and Sagaut's terminology, the correct gradient equation is:

$$\frac{dI}{d\boldsymbol{\mu}} = \begin{cases} \sum_{i=0}^{q-1}\sum_{k=0}^{M-1}\sum_{n=0}^{N-1} f_i^{*,V}(\mathbf{x}_k, t_{n+1})\frac{\partial R_i(\mathbf{x}_k,t_n)}{\partial \boldsymbol{\mu}}, \text{if } \hat{f}(\boldsymbol{\mu}) \\ \sum_{i=0}^{q-1}\sum_{k=0}^{M-1}\sum_{n=0}^{N-1} \hat{f}_i^{*}(\mathbf{x}_k, t_{n+1})\frac{\partial R_i(\mathbf{x}_k,t_n)}{\partial \boldsymbol{\mu}}, \text{if } S(\boldsymbol{\mu}) \end{cases} \tag{C.0.14}$$

This is just rewriting Tekitek et al.'s method with different symbols, of course.

A note on Luo's recent formulation

In a recent paper, Luo et al present an alternative derivation for the discrete-adjoint LBM. They consider a multi-stage Lagrangian:

$$J = I + \sum_{n=0}^{N} \mathbf{F}^{*,s,n+1^{\intercal}}(\hat{\mathbf{F}}^n - \mathscr{C}(\mathbf{F}^n)) + \sum_{n=0}^{N} \mathbf{F}^{*,n+1^{\intercal}}(\mathbf{F}^{n+1,s} - S\hat{\mathbf{F}}^n) + \sum_{n=0}^{N} \hat{\mathbf{F}}^{*,n+1^{\intercal}}(\mathbf{F}^{n+1} - \mathcal{B}\mathbf{F}^{n+1,s}) \tag{C.0.15}$$

where the post-boundary condition adjoint variable $\mathbf{F}^{*,n+1^{\intercal}}$ is directly associated with the post-streaming populations $\mathbf{F}^{n+1,s}$, the post-streaming adjoint $\mathbf{F}^{*,s,n+1^{\intercal}}$ controls the post collision populations, and the post-collision adjoint $\hat{\mathbf{F}}^{*,n+1^{\intercal}}$ the post-boundary condition populations $\mathbf{F}^n$. Here the boundary treatment is incorporated into the general operator $\mathcal{B}$ to allow for non-linear rules.

Using this Lagrangian—which requires a posteriori knowledge of the form of the adjoint iteration[2]—they arrive at the following method:

$$\begin{cases} \hat{\mathbf{F}}^{*,N+1} = 0 \\ \mathbf{F}^{*,n+1} = \left(\frac{\partial \mathscr{B}(\mathbf{F}^{n,s})}{\partial F^{n,s}}\right)^{\intercal} \mathbf{F}^{*,n+1} \\ \mathbf{F}^{*,s,n+1} = S^{\intercal} \cdot \mathbf{F}^{*,n+1} \\ \hat{\mathbf{F}}^{*,n} = \left(\frac{\partial \mathscr{C}(\mathbf{F}^{n})}{\partial F^{n}}\right)^{\intercal} \mathbf{F}^{*,s,n+1} - \frac{1}{N}\left(\frac{\partial \mathcal{I}^{n}}{\partial \mathbf{F}^{n}}\right)^{\intercal} n = N, ..., 1 \end{cases} \tag{C.0.16}$$

which is clearly equivalent to (3.2.2) albeit with an extra timestep to accommodate the zero initial condition (which is what we did with Vergnault). The downside of this multi-stage derivation is that it does not lead to a general gradient equation. The authors state, without proof, that $\frac{dI}{d\boldsymbol{\mu}}$ should be updated after the adjoint streaming (they consider a steady case):

$$\frac{dI}{d\boldsymbol{\mu}} = -\mathbf{F}^{*,s}\frac{\partial \mathscr{C}}{\partial \boldsymbol{\mu}}. \tag{C.0.17}$$

This, of course, only holds if the optimization parameters affect the collision operator alone. Thus, like Vergnault and Sagaut's approach, this derivation yields a method that works equivalently in practice but has a weaker theoretical basis than Tekitek et al.'s.

[2]This is true only if we want the names of the adjoint variables to make sense. One could easily name them generically (e.g. $F^{*,a}$, $F^{*,b}$, and $F^{*,c}$) and figure out their interpretation later.

Appendix D: Example of source terms arising due to non-local boundary conditions

As stated since the literature review, applying a non-local forward boundary condition at boundary nodes affects the adjoint problem. Specifically, for (some) nodes near such boundaries, the adjoint streaming operation generates source terms. This appendix shows an example of this process by considering an extrapolation boundary condition at the outlet and building the associated streaming and transposed streaming matrices.

For clarity, this example considers a 1D problem with a D1Q3 lattice:

$$\begin{cases} \boldsymbol{\xi}_0 = 0 \\ \boldsymbol{\xi}_1 = 1 \\ \boldsymbol{\xi}_1 = -1 \end{cases} \tag{D.0.1}$$

and 3 spatial coordinates x_0, x_1, x_2. x_0 is a boundary node adjacent to a wall where bounce back is applied. x_2 is the outlet node where all populations are extrapolated from x_1 (as in section 2.3). The case is shown in Figure D.1.

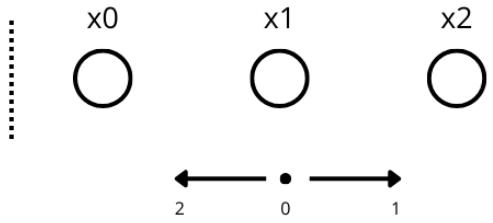


Figure D.1.: Sample 1D case geometry.

We construct the streaming matrix by considering the operation

$$\mathbf{F}^{n+1} = S\hat{\mathbf{F}}^n \tag{D.0.2}$$

where $\hat{\mathbf{F}} = \begin{bmatrix} \hat{f}_0(x_0) & \hat{f}_0(x_1) & \dots & \hat{f}_2(x_1) & \hat{f}_2(x_2) \end{bmatrix}^\intercal$ is the vector of post-collision populations, and boundary conditions are directly accounted for in S. Explicitly writing out all 9 equations that determine the problem yields

$$\begin{cases} f_0(x_0) = \hat{f}_0(x_0) & \text{Regular streaming.} \\ f_0(x_1) = \hat{f}_0(x_1) & \text{Regular streaming.} \\ f_0(x_2) = f_0(x_1) = \hat{f}_0(x_1) & \text{Extrapolation.} \\ f_1(x_0) = \hat{f}_2(x_0) & \text{Bounce back.} \\ f_1(x_1) = \hat{f}_1(x_0) & \text{Regular streaming.} \\ f_1(x_2) = f_1(x_1) = \hat{f}_1(x_0) & \text{Extrapolation.} \\ f_2(x_0) = \hat{f}_2(x_1) & \text{Regular streaming.} \\ f_2(x_1) = \hat{f}_2(x_2) & \text{Regular streaming.} \\ f_2(x_2) = f_2(x_1) = \hat{f}_2(x_2) & \text{Extrapolation.} \end{cases} \tag{D.0.3}$$

The resulting streaming matrix S is:

$$S\hat{\mathbf{F}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{f}_0(x_0) \\ \hat{f}_0(x_1) \\ \hat{f}_0(x_2) \\ \hat{f}_1(x_0) \\ \hat{f}_1(x_1) \\ \hat{f}_1(x_2) \\ \hat{f}_2(x_0) \\ \hat{f}_2(x_1) \\ \hat{f}_2(x_2) \end{bmatrix}. \tag{D.0.4}$$

Transposing, we obtain the adjoint post-streaming populations

$$\mathbf{F}^{*,s} = \begin{bmatrix} f_0^{*,s}(x_0) & f_0^{*,s}(x_1) & \dots & f_2^{*,s}(x_1) & f_2^{*,s}(x_2) \end{bmatrix}^\intercal = S^\intercal \mathbf{F}^*$$

explicitly written out as:

$$\begin{cases} f_0^{*,s}(x_0) = f_0^*(x_0) & \text{Backwards streaming.} \\ f_0^{*,s}(x_1) = f_0^*(x_1) + f_0^*(x_2) & \text{Backwards streaming} + \text{Source term.} \\ f_0^{*,s}(x_2) = 0 & \text{Adjoint extrapolation.} \\ f_1^{*,s}(x_0) = f_1^*(x_1) + f_1^*(x_2) & \text{Backwards streaming} + \text{Source term.} \\ f_1^{*,s}(x_1) = 0 & \text{Source term.} \\ f_1^{*,s}(x_2) = 0 & \text{Adjoint extrapolation.} \\ f_2^{*,s}(x_0) = f_1^*(x_0) & \text{Adjoint bounce back.} \\ f_2^{*,s}(x_1) = f_2^*(x_0) & \text{Backwards streaming.} \\ f_2^{*,s}(x_2) = f_2^*(x_1) + f_2^*(x_2) & \text{Adjoint extrapolation.} \end{cases} \tag{D.0.5}$$

We can make the following observations:

1. There is no obvious adjoint extrapolation rule. In other words, unlike for local boundary conditions (bounce back, free-slip), the adjoint boundary condition is not simply the transpose of the forward boundary condition (i.e. the rule is not $f_i^{*,s}(x_2) = f_i^{*,s}(x_1)$). This is the mistake that Cheylan et al. make in [42] for an interpolated bounce back boundary condition.
2. There is no obvious adjoint extrapolation rule. In other words, unlike for local boundary conditions (bounce back, free-slip), the adjoint boundary condition is not simply the transpose of the forward boundary condition (i.e. the rule is not $f_i^{*,s}(x_2) = f_i^{*,s}(x_1)$). This is the mistake that Cheylan et al. make in [42] for an interpolated bounce back boundary condition.
3. The population $f_1^{*,s}(x_1)$—which at first glance should receive population $f_1^*(x_2)$ via a simple backwards streaming step—is set to $f_1^{*,s}(x_1) = 0$. This reinforces the conclusion that non-local boundary conditions alter the adjoint streaming operation in a non-trivial, non-local, and non-obvious way, which (I believe) is best dealt with by representing the forward streaming with a matrix vector product. By doing so, all these complications are absorbed into an essentially foolproof transpose operation, which can then be turned into local rules by writing out the non-zero entries in the rows of $S^\intercal$, as we did above.

Appendix E: Can a gradient based optimization algorithm successfully run with the halfway bounce back boundary condition?

This work originated as part of a larger research effort to run industrially relevant shape optimization cases with a quantum LBM code, where the promise of the quantum approach is to dramatically reduce the computational cost of a forward problem solution. The quantum solver is restricted to unitary operations, so the halfway bounce back boundary condition is perhaps the most appropriate method for (no-slip) fluid-solid interfaces, and is the only method available to the TU Delft group. As I have discussed in section 2.3, this rule does not (nominally) propagate infinitesimal shape changes into the fluid, as it always assumes the boundary lies exactly halfway between the boundary node and the boundary. To overcome this, I proposed the PBB approach. This, however, requires local modifications to the collision operator that may be difficult to implement given the data-driven learning-based approach currently at the forefront of research into quantum-friendly collision operators [13].

In this appendix, I present preliminary results for a methodology—inspired by PBB—that may be able to obtain cost-function gradients for a flow where the obstacle is simulated with bounce back, requiring no modifications to the forward code. To demonstrate its viability, I rerun the airfoil L/D optimization case of chapter 4 with promising results.

E.1. Methodology

The starting point is PBB, restated as follows. Let

$$\tilde{\mathbf{x}}_{k_i} = \mathbf{x}_k + \boldsymbol{\xi}_i. \tag{E.1.1}$$

Then, the set of boundary nodes, X_b, is:

$$X_b = \{\tilde{\mathbf{x}}_{k_i} | \tilde{\mathbf{x}}_{k_i} \notin X_f, \mathbf{x}_k \in X_f\} \tag{E.1.2}$$

where X_f is the set of fluid nodes. For $\mathbf{x}_k \in X_b$, apply the bounce back rule with a modified collision operation $\hat{f}_i$:

$$f_{i^-}(\mathbf{x_k}, t_{n+1}) = \hat{f}_i(\rho, \varepsilon\mathbf{u}, \mathbf{x}_k, t) \tag{E.1.3}$$

where the modification scales the velocity as $\varepsilon\mathbf{u}$ with

$$\varepsilon = 1 - a \exp\left[-\frac{d^2}{\sigma_\varepsilon^2 b}\right]. \tag{E.1.4}$$

a, σ_s, and b are free parameters discussed in section 2.3 and d is the minimum distance from the boundary node to the boundary.

Now, we will modify $\varepsilon(d)$[1] so that $\varepsilon = 1$ for all $\mathbf{x}_k \in X_b$, thus reducing PBB to bounce back.

[1]The initial definition of $\varepsilon(d)$ as an exponential function was completely arbitrary.

Let

$$\varepsilon = \frac{d}{\sigma_s}. \tag{E.1.5}$$

d remains the minimum distance to the boundary but now, for each boundary node, choose σ_s with the *constant* value $\sigma_s = d_{const}$, such that $d_{const} = d \implies \sigma_s = d \implies \varepsilon = 1$. The key is not to choose $\sigma_s = d$, but rather to *fortuitously guess*[2] the exact constant d_{const} such that $d_{const} = d$.

The idea of this approach is that the cost-function derivative $\frac{dI}{d\boldsymbol{\mu}}$, which involves the product

$$\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}} = S\frac{\partial \mathscr{C}(\mathbf{F}^n)}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial d} \cdot \frac{\partial d}{\partial \boldsymbol{\mu}}$$

remains non-zero because

$$\frac{\partial \varepsilon}{\partial d} = \partial_d \frac{d}{\sigma_s} = \frac{1}{\sigma_s}. \tag{E.1.6}$$

The underlying assumption is that the Brinkmann porosity model has physical significance for 'super-fluid' nodes with $\varepsilon = 1$, that effectively act as momentum sources in the flow.

The final step is to replace the original equation for $\frac{d\varepsilon}{dd}$, (3.2.8), with (E.1.6) at all appropriate spots in the adjoint gradient pipeline ($\frac{\partial I}{\partial \boldsymbol{\mu}}$ and $\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$). With everything in place, it should be possible to obtain gradients of a QoI computed with the bounce back boundary condition. Henceforth, I refer to this method as synthetic bounce back (SBB).

A number of aspects merit clarification, particularly relating to how the method works in a shape optimization pipeline. Every time the shape changes, σ_s must be recalculated ('re-guessed') to ensure $\varepsilon = 1$. In other words, $\sigma_s = d_{const}$ with d_{const} only a constant for a given shape. However, when verifying the approach with finite differences, σ_s *must* stay constant. For (perhaps excessive) clarity, the following procedure details how to take finite difference gradients:

1. Define the boundary. In this work, this is done with the B-Spline + FFD approach.
2. For each boundary node, compute d and set $\sigma_s = d$.
3. For each boundary node, store the value of σ_s.
4. For each boundary node, set $\varepsilon = d/\sigma_s = 1$.
5. Run the forward solver with the unperturbed optimization parameters ($\boldsymbol{\mu}^0$).
6. Perturb one element of $\boldsymbol{\mu}$ as: $\mu = \mu^0 + h$.
7. Recompute the boundary.
8. Recompute d *but not* σ_s.
9. Set $\varepsilon = d/\sigma_s \neq 1$, using the stored value of σ_s.
10. Run the perturbed forward solver and compute the gradient with respect to μ^0.

E.2. Results

As in chapter 4, the results consist of a (very preliminary) mesh convergence study to validate the implementation of the bounce back rule, then a finite difference verification of the adjoint[3]-based gradients, followed by a demonstration of their physical validity through an optimization case. The test case is the steady airfoil described in section 3.6.

[2] Obviously in an actual implementation you do not guess but use the computed value of d.
[3] *née* chain rule

E.2.1. Mesh convergence study

The following table serves as a preliminary mesh convergence study for the lift and drag coefficients and the lift to drag ratio. To accelerate the computation, acoustic scaling is used with $Ma = 0.108$ (therefore, a small compressibility error is expected).

Table E.1.: Preliminary mesh convergence study using bounce back for the coefficient of lift, coefficient of drag, and lift to drag ratio.

Cells per chord	C_L	C_D	L/D
32	0.259	0.451	0.574
64	0.262	0.446	0.587
128	0.289	0.447	0.647
Reference	0.288	0.446	0.646

More extensive validation is required — preferably with diffusive scaling or a smaller Mach number, and certainly on at least one finer mesh — but the low error on the finest available mesh suggests the implementation is correct. Note, the big leap in the lift coefficient is similar to the behaviour of PBB (see Figure 4.7) and, again, likely due to the non-smooth staircase representation of the geometry.

E.2.2. Gradient verification

As in subsection 4.2.1, the mesh resolves the airfoil with 32 points per chord. Table E.2 shows the comparison between adjoint and finite difference based gradients.

Table E.2.: Comparison between finite-difference (FD) reference values and adjoint results for selected entries. Percent error is computed with FD taken as the truth.

Entry	FD (truth)	Adjoint	Percent error (%)
$\mu_{y_{2,0}}$	0.89628882	0.89628492	0.00043472
$\mu_{y_{2,1}}$	0.68986919	0.68985246	0.00242439
$\mu_{y_{6,0}}$	0.01517057	0.01516562	0.03260497
$\mu_{y_{6,1}}$	-0.68441464	-0.68444231	-0.00404244
$\mu_{y_{10,0}}$	-2.11865135	-2.11874692	-0.00451100
$\mu_{y_{10,1}}$	-1.04728350	-1.04731608	-0.00311091

The agreement is excellent and comparable to that of PBB (see Table 4.3), strongly suggesting the adjoint implementation is correct.[4]

E.2.3. L/D optimization

To ensure the gradients are appropriate for shape optimization, thirty[5] iterations of steepest descent are run with a stepsize $s = 1$:[6]

$$\boldsymbol{\mu}_y^{i+1} = \boldsymbol{\mu}_y^i - s\frac{dI}{d\boldsymbol{\mu}_y^i}. \tag{E.2.1}$$

[4] i.e. the derivatives $\frac{d\varepsilon}{dd}$, $\frac{\partial I}{\partial \boldsymbol{\mu}}$, and $\frac{\partial \Phi^n}{\partial \boldsymbol{\mu}}$.

[5] The number of iterations is chosen for comparability with PBB's results, cf. Figure 4.34.

[6] The stepsize $s = 1$ was chosen because it is default stepsize for the first iteration of SLSQP—the optimizer used for PBB.

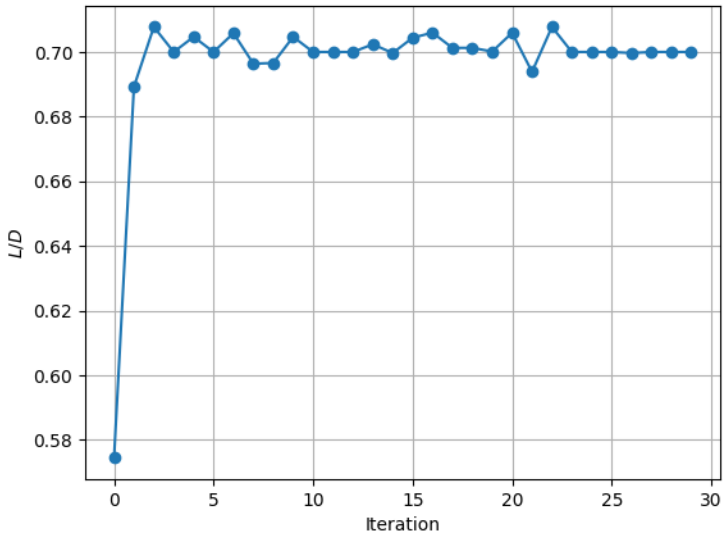


Figure E.1.: L/D optimization history using SBB.

A bounds constraint is set such that $|\boldsymbol{\mu}_{y_{l,m}}| \leq 0.35$.

Figure E.1 and Table E.3 show the optimization history. Evidently, the optimizer has achieved a significant increase in the QoI, with results similar to PBB—namely, a large increase in lift and almost no change in drag. Given the fixed stepsize and the significantly larger gradient (by two orders of magnitude), the SBB based optimization achieves its maximum much more quickly than PBB, spending the majority of the iterations making small oscillations about the optimum. The final shape (Figure E.2) is close to that obtained from PBB, as is further shown by the vectors of final optimization parameters:

$$\boldsymbol{\mu}_y^{SBB} = \begin{bmatrix} 0.35 & -0.1773067 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & -0.35 & -0.35 & -0.35 \end{bmatrix}, \quad \text{(E.2.2)}$$

$$\boldsymbol{\mu}_y^{PBB} = \begin{bmatrix} 0.35 & 0.01765153 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & -0.35 & -0.35 \end{bmatrix}. \quad \text{(E.2.3)}$$

There are two minor discrepancies, but the end result is largely similar, suggesting that the optimization is building on the same physical principle (lift incrementation through increased pressure on the lower side and increased suction on the upper side, with a balancing effect on the drag). Examining (E.2.2), we observe that the optimum value is again located at the edges of the constrained design space. From all of this evidence, we can preliminarily conclude that the gradients obtained from SBB are physical.

Table E.3.: Evolution of surface force coefficients.

Parameter	Original	Optimized	% change
C_L	0.25906	0.31567	+21.9
C_D	0.45096	0.45094	-0.004
L/D	0.57445	0.70003	+21.9

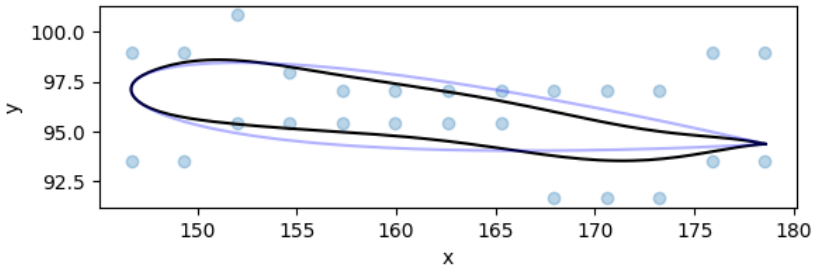


Figure E.2.: Final shape after L/D optimization using SBB. Original shape shown in light blue.

E.3. Conclusion

A promising methodology for obtaining gradients of the bounce back boundary condition was presented, based on considering the rule as a point within a continuous Brinkmann porosity model supporting momentum sources (i.e. super-fluid nodes with a porosity greater than 1). This may be of particular interest for application in quantum LBM gradient-based shape optimization, given bounce back's unitary nature. The principal remaining work is carrying out more extensive validation of the forward solver, and verifying the adjoint-based gradients for an unsteady case.

Appendix F: Alternative example of the effect of a homogenous outlet adjoint boundary condition on gradient accuracy

Table F.1 shows the adjoint and finite difference based gradients of the (half) mean squared density: $I = \overline{\rho} = \frac{1}{2N}\sum_{n=0}^{N}\sum_{k}\rho(\mathbf{x}_{obs}, t_n)$ for the unsteady cylinder case. The cost function is measured on a vertical line located 6 diameters from the centre of the cylinder. With respect to Figure 3.4 the line is located at $x_{obs} = 0.8$ m, $y_{obs} \in [0.03, 0.38]$ m. Two comparisons are presented, one using the exact adjoint outlet boundary condition, another replacing this with a homogenous outlet. Evidently, the homogenous outlet is incorrect.

Table F.1.: Comparison between finite-difference (FD) reference values and adjoint results. $\overline{\rho}$ for the first 10000 time-steps. No grid refinement or sponge. Percent error is computed with FD taken as the truth.

Entry	FD (truth)	Adjoint	Error (%)	Adjoint (hom. outlet)	Error (hom. outlet) (%)
$\mu_{y_{0,0}}$	0.004463	0.004463	-0.000633	0.003854	13.64
$\mu_{y_{0,1}}$	-0.002257	-0.002257	-0.004434	-0.0009745	56.82
$\mu_{y_{0,2}}$	0.00007204	0.00007193	0.1643	-0.001198	1763
$\mu_{y_{1,0}}$	0.005069	0.005069	0.001633	0.004648	8.308
$\mu_{y_{1,1}}$	-0.002692	-0.002692	0.002793	-0.001306	51.47
$\mu_{y_{1,2}}$	-0.0003945	-0.0003946	-0.002570	-0.002223	-463.4
$\mu_{y_{2,0}}$	-0.006529	-0.006530	-0.002438	-0.002648	59.45
$\mu_{y_{2,1}}$	-0.008242	-0.008242	-0.000634	-0.004152	49.62
$\mu_{y_{2,2}}$	-0.008590	-0.008590	-0.000143	-0.005058	41.13