# Hledání optimálního tvaru stěn matematického modelu proudění krve v problematice úplného kavopulmonálního cévního napojení

# Optimal shape design of walls of blood flow mathematical model focusing on the total cavopulmonary connection

Master thesis

Author:          **Bc. Jan Bureš**

Supervisor:      **doc. Ing. Radek Fučík, Ph.D.**

Consultant:      **MUDr. Mgr. Radomír Chabiniok, Ph.D.**

Academic year:   2024/2025

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bureš**  Jméno: **Jan**  Osobní číslo: **494688**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Matematické inženýrství**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Optimální tvar stěn idealizovaného úplného kavopulmonálního spojení.**

Název diplomové práce anglicky:

**Optimal wall geometry of an idealized total cavopulmonary connection.**

Pokyny pro vypracování:

1. S použitím dostupné literatury a konzultací s odborníky sestavte matematický model cévního proudění v problematice úplného kavopulmonálního spojení (TCPC) založený na mřížkové Boltzmannově metodě (LBM) a navrhněte vhodně parametrizovanou testovací úlohu, která aproximuje charakteristiky a funkčnost systému TCPC.
2. Formulujte optimalizační úlohu zaměřenou na hledání optimálního tvaru stěn s využitím parametricky popsané geometrie testovací úlohy z bodu 1. Pokuste se navrhnout vhodnou účelovou funkci použitelnou v rámci studované problematiky.
3. Proveďte rešerši optimalizačních metod vhodných pro řešení problémů charakterizovaných zvýšenou časovou náročností potřebnou pro vyhodnocení účelové funkce.
4. Pro simulaci proudění vhodně upravte výpočetní kód LBM vyvíjený na KM FJFI ČVUT v Praze. Navrhněte a implementujte obecný optimalizační rámec, jehož součástí bude výpočetní kód LBM. Využijte volně dostupný software nebo sám implementujte vybrané metody matematické optimalizace z bodu 3.
5. Aplikujte metody matematické optimalizace k hledání optimálního řešení pro uvažovanou optimalizační úlohu z bodu 2. Diskutujte získané výsledky a výpočetní náročnost jejich získání.

Seznam doporučené literatury:

[1] T. Krüger, et al., The lattice Boltzmann method: Principles and Practice. Springer International Publishing, 2017.
[2] Z. Guo, S. Chang, Lattice Boltzmann method and its application in engineering. World Scientific, 2013.
[3] J. D. Anderson, Computational Fluid Dynamics. McGraw-Hill series in mechanical engineering. McGraw-Hill Professional, 1995.
[4] F. M. Rijnberg, et al., Energetics of blood flow in cardiovascular disease. Circulation,137(22), 2018, 2393–2407.
[5] S. Boyd, L. Vandenberghe, Convex optimization. Cambridge University Press, 2004.
[6] C. Audet a W. Hare. Derivative-free and blackbox optimization. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, Switzerland, 1.edice, 2017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Ing. Radek Fučík, Ph.D.  katedra matematiky  FJFI**
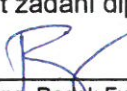
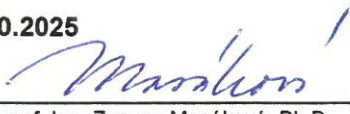Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

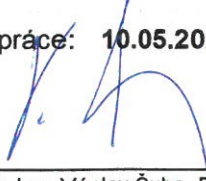**Mudr. Mgr. Radomír Chabiniok, Ph.D.  University of Texas Southwestern Medical Center, USA**

Datum zadání diplomové práce: **31.10.2023**  Termín odevzdání diplomové práce: **10.05.2024**

Platnost zadání diplomové práce: **31.10.2025**

| doc. Ing. Radek Fučík, Ph.D. | prof. Ing. Zuzana Masáková, Ph.D. | doc. Ing. Václav Čuba, Ph.D. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

# III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

_____                          _____
Datum převzetí zadání                                Podpis studenta

# Contents

# Introduction

# Chapter 1

# Mathematical model

# Chapter 2

# Lattice Boltzmann method

We can consider a fluid as a continuum and use a macroscopic description, meaning we view the fluid as a whole and utilize macroscopic quantities such as density, flow velocity, or pressure for the state description, with the governing equations being (**??**). On the other hand, we can describe the dynamics of each particle in a given volume and thus use a microscopic description. Describing the dynamics of a particle at a microscopic scale is not difficult, but a significant drawback of this approach is its evident computational complexity, which is directly proportional to the number of particles being studied.

A middle ground between the above-mentioned descriptions is the mesoscopic description [1], which is based on kinetic theory. The fluid is described using one-particle probability distribution functions of density $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$ [kg s$^3$ m$^{-6}$], which describe the system in the space of coordinates $\boldsymbol{x}$, microscopic velocities $\boldsymbol{\xi}$, and time $t$. The distribution functions $f$ give the density of particles occurring in the vicinity of $\boldsymbol{x}$ at time $t$ with a microscopic velocity $\boldsymbol{\xi}$.

The one-particle distribution functions in the vicinity of $\boldsymbol{x}$ satisfy the Boltzmann transport equation [2]

$$\frac{\partial f}{\partial t} + \sum_{i=1}^{3} \xi_i \frac{\partial f}{\partial x_i} + \sum_{i=1}^{3} g_i \frac{\partial f}{\partial \xi_i} = C(f), \tag{2.1}$$

where $\boldsymbol{g}$ [m s$^{-2}$] is the acceleration vector of external forces, and $C(f)$ [kg s$^2$ m$^{-6}$] is the collision operator, which will be discussed further in section **??**.

Using the distribution functions $f$, certain macroscopic quantities can be expressed as statistical moments [2], such as

$$\rho(\boldsymbol{x}, t) = \int_{\mathbb{R}^3} f(\boldsymbol{x}, \boldsymbol{\xi}, t) \, \mathrm{d}\boldsymbol{\xi}, \tag{2.2a}$$

$$\rho(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) = \int_{\mathbb{R}^3} f(\boldsymbol{x}, \boldsymbol{\xi}, t) \boldsymbol{\xi} \, \mathrm{d}\boldsymbol{\xi}. \tag{2.2b}$$

The lattice Boltzmann method (LBM) is a numerical method based on the mesoscopic description of fluids. The numerical scheme of LBM can be derived by discretizing (2.1). In LBM, the spatial discretization is performed using a discrete equidistant lattice, and the discretization of velocity space is carried out using a finite set of microscopic velocities. We introduce velocity models denoted as D$d$Q$q$, where $d$ and $q$ represent the dimension of the considered space and the number of different directions in which movement from each lattice node is possible, respectively.

In this thesis, we consider the D3Q27 velocity model, which is schematically shown in Figure 2.1.
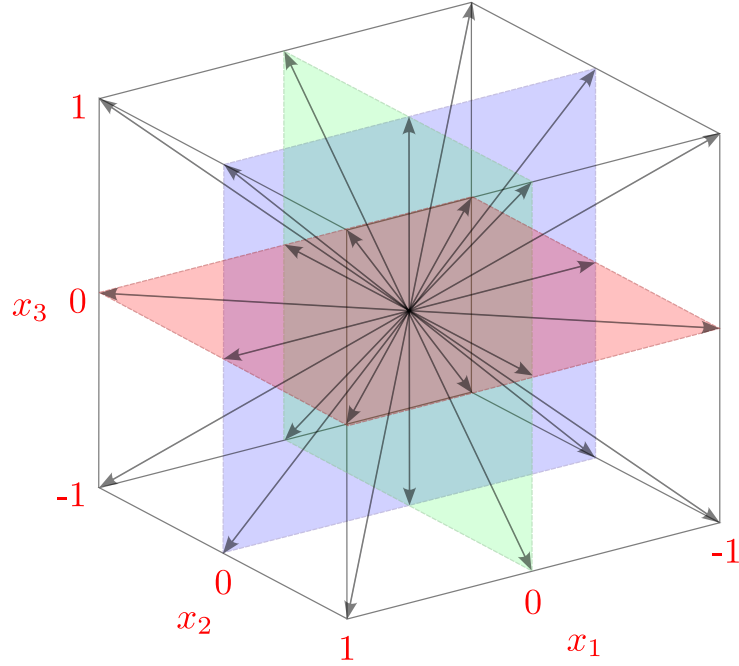
Figure 2.1: Geometric representation of the D3Q27 velocity model.

The velocity space of the microscopic particle velocity is discretized by a finite set of microscopic velocities, which, in the case of the D3Q27 model, is defined as

$$
(\xi_k)_{k=1}^{27} = \left( \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}, \right.
$$
$$
\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix},
$$
$$
\left. \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \right).
\tag{2.3}
$$

## 2.1 Non-dimensionalization and discretization

The computational domain $(0; L_1) \times (0; L_2) \times (0; L_3)$, where $L_i$ [m], $i = 1, 2, 3$, are the dimensions of the domain, is discretized using an equidistant grid with a spatial step size $\Delta l$ [m]. The time interval is then uniformly divided with a time step size $\Delta t$ [s].

In LBM, it is common to work with non-dimensional quantities instead of the original physical ones, as discussed, for example, in [2]. We will now introduce the basic conversion relationships for transitioning to a non-dimensional system of units. We also define the grid used for the discretization of the domain in LBM and describe the discretized Boltzmann transport equation.

10

### 2.1.1 Non-dimensional units

As mentioned, in LBM, it is advantageous to consider all quantities as non-dimensional. We transition from physical units to non-dimensional lattice units. It should be emphasized that the physical principles remain valid regardless of the choice of the unit system. The transition between unit systems can be performed using the law of similarity for fluid dynamics, where the equivalence of systems requires that the values of relevant non-dimensional quantities remain conserved [2]. One such non-dimensional quantity that can be used during the transition is, for example, the Reynolds number (**??**).

In the following conversion relations, all quantities in lattice units are marked with the superscript $L$. It can be shown [2] that the following holds:

$$l_0^L = \frac{l_0}{\Delta l}, \tag{2.4}$$

$$t_0^L = \frac{t_0}{\Delta t}, \tag{2.5}$$

$$\nu^L = \nu\frac{\Delta t}{\Delta l^2}, \tag{2.6}$$

$$u_i^L = \frac{\Delta t}{\Delta l}u_i, \ i \in \{1, 2, 3\}. \tag{2.7}$$

The characteristic length, time, and velocity values are chosen in accordance with the given physical problem. The dimension of the computational domain or an obstacle is typically chosen as $l_0$. Details of the derivation of the above conversion relations can be found in [2].

It should be noted that from the conversion relation for kinematic viscosity, we can see that for a given grid with a spatial step $\Delta l$, the time step $\Delta t$ is linked to the value of $\nu^L$.

For simplicity, in this work, we will assume that the spatial step $\Delta l^L$ and the time step $\Delta t^L$ in lattice units are $\Delta l^L = \Delta t^L = 1$. Let us emphasize that in this chapter, we will work exclusively with quantities in lattice units and will thus omit the use of the superscript $L$, although the non-dimensional description will still be considered.

### 2.1.2 Computational Domain and Discrete Grid

From this point forward, we assume that all quantities are expressed in lattice units, meaning $\Delta l$ and $\Delta t$ now represent non-dimensional spatial and time steps, respectively. We consider a cuboidal computational domain $\Omega \subset \mathbb{R}^3$, see section **??**. This domain in LBM is discretized using an equidistant grid

$$\hat{\Omega} = \left\{ \boldsymbol{x}_{i,j,k} = (i\Delta l, \ j\Delta l, \ k\Delta l)^T \ \big| \ i \in \{1, \ldots, N_1 - 1\}, j \in \{1, \ldots, N_2 - 1\}, k \in \{1, \ldots, N_3 - 1\} \right\}, \tag{2.8a}$$

$$\overline{\hat{\Omega}} = \left\{ \boldsymbol{x}_{i,j} = (i\Delta l, \ j\Delta l)^T \ \big| \ i \in \{0, \ldots, N_1\}, j \in \{0, \ldots, N_2\}, k \in \{0, \ldots, N_3\} \right\}, \tag{2.8b}$$

where $N_i$ [-] denotes the number of grid points in the $x_i$ direction, $i = 1, 2, 3$.

The boundary of the domain $\Omega$ will be denoted by $\partial\Omega$ and understood as the closure of the disjoint union of the individual parts

$$\partial\Omega = \overline{\partial\Omega_W \cup \partial\Omega_E \cup \partial\Omega_N \cup \partial\Omega_S \cup \partial\Omega_F \cup \partial\Omega_B}, \tag{2.9}$$

where the meanings of $\partial\Omega_W, \partial\Omega_E, \partial\Omega_N, \cup\partial\Omega_S, \cup\partial\Omega_F, \partial\Omega_B$ are illustrated in Figure 2.2. We will further consider the discretization of the boundary of the computational domain as

$$\partial\hat{\Omega} = \overline{\hat{\Omega}} \setminus \hat{\Omega}, \tag{2.10}$$

11

with the corresponding parts of the discrete boundary denoted as $\partial\hat{\Omega}_W, \partial\hat{\Omega}_E, \partial\hat{\Omega}_N, \partial\hat{\Omega}_S, \partial\hat{\Omega}_F$, and $\partial\hat{\Omega}_B$.
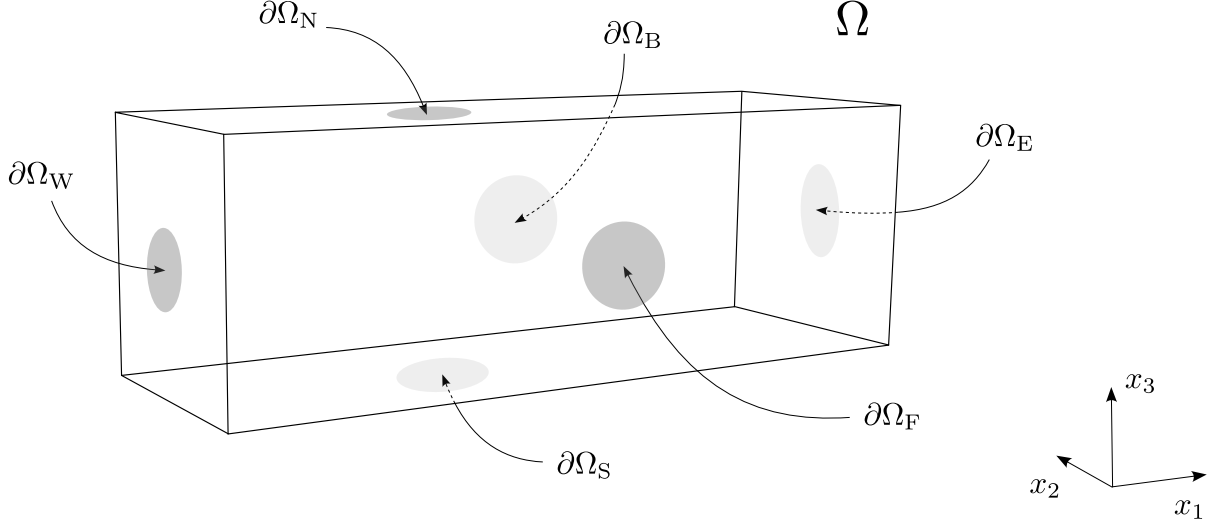


Figure 2.2: Schematic representation of the computational domain $\Omega$ and its boundary $\partial\Omega$.

The time interval over which the problem will be investigated will be denoted by $\mathcal{I}$, where $\mathcal{I} = \langle 0, T \rangle$, and $T > 0$. The interval $\mathcal{I}$ is discretized as

$$\hat{\mathcal{I}} = \left\{ t_i = i\frac{T}{N_t} \,\middle|\, i \in \{0, \dots, N_t\} \right\}, \tag{2.11}$$

where $N_t$ represents the number of discrete time steps for the discretization of the interval $\mathcal{I}$.

### 2.1.3 Discrete Boltzmann transport equation

When using the D3Q27 model, we work with a set of distribution functions

$$\{f_k(\boldsymbol{x}, t) \mid k = 1, \dots, 27\}, \; \forall \boldsymbol{x} \in \hat{\Omega}, \; \forall t \in \hat{\mathcal{I}}, \tag{2.12}$$

where the indices correspond to the directions of microscopic velocities from (2.3).

It can be shown that by discretizing the equation (2.1), we obtain the form

$$f_k(\boldsymbol{x} + \Delta t\boldsymbol{\xi}_k, t + \Delta t) = f_k(\boldsymbol{x}, t) + C_k(\boldsymbol{x}, t) + S_k(\boldsymbol{x}, t), \quad k \in \{1, \dots, 27\}, \; \forall \boldsymbol{x} \in \hat{\Omega}, \; \forall t \in \hat{\mathcal{I}}, \tag{2.13}$$

where $C_k$ represents the discrete collision operator and $S_k$ is the discrete forcing term. Details of the derivation of the discrete form of the Boltzmann transport equation can be found in [2].

The discrete collision operator $C_k$ in equation (2.13) defines the specific variant of LBM. Several choices for $C_k$ exist, each corresponding to different LBM formulations. These include, for instance, the single relaxation time (SRT-LBM) [3], multiple relaxation time (MRT-LBM) [4], central moment (CLBM) [5], entropic (ELBM) [6], and cumulant-based (CuLBM) [3] approaches. In this work, we use the cumulant collision operator, as detailed in [3].

We can further introduce the post-collision distribution functions $f_k^*$ as

$$f_k^*(\boldsymbol{x}, t) = f_k(\boldsymbol{x}, t) + C_k(\boldsymbol{x}, t) + S_k(\boldsymbol{x}, t), \quad k \in \{1, \dots, 27\}, \; \forall \boldsymbol{x} \in \hat{\Omega}, \; \forall t \in \hat{\mathcal{I}}. \tag{2.14}$$

Using $f_k^*$, we can express (2.13) in the form

$$f_k(\boldsymbol{x} + \Delta t\boldsymbol{\xi}_k, t + \Delta t) = f_k^*(\boldsymbol{x}, t), \quad k \in \{1, \dots, 27\}, \; \forall \boldsymbol{x} \in \hat{\Omega}, \; \forall t \in \hat{\mathcal{I}}, \tag{2.15}$$

which can be interpreted as an explicit prescription for calculating the distribution functions.

### 2.1.4 Macroscopic Quantities

Finally, we present the relations for calculating some of the macroscopic quantities. Some of these relations can be derived through the process of discretization from the general relations (2.2) [2]. The relations for calculating density, momentum, and pressure are as follows:

$$\rho = \sum_{k=1}^{27} f_k, \tag{2.16a}$$

$$\rho\boldsymbol{u} = \sum_{k=1}^{27} f_k \boldsymbol{\xi_k} + \rho \frac{\Delta t}{2} \boldsymbol{g}, \tag{2.16b}$$

$$p = p_0 + c_s^2(\rho - \rho_0), \tag{2.16c}$$

where $p_0$ [−] is the non-dimensional reference value of pressure, $c_s$ [−] is the non-dimensional (lattice) speed of sound, and $\rho_0$ is the non-dimensional reference value of density. For $c_s$ in the D3Q27 model, $c_s = \frac{1}{\sqrt{3}}$. Furthermore, we consider $\rho_0 = 1$. A detailed description of the calculation of macroscopic quantities is provided in [2].

## 2.2 LBM Algorithm

The LBM algorithm can be summarized in several steps:

1. **Initialization** of initial conditions on the grid, discussed in section 2.3.

2. **Cycle** ending with the fulfillment of a user-specified termination condition.

   (a) **Streaming** of post-collision distribution functions $f_k^*$ in the respective directions $\boldsymbol{\xi_k}$.
   (b) **Calculation of macroscopic quantities** using the relations (2.16).
   (c) **Collision**, where the post-collision state of the distribution function is calculated using (2.15), and **application of boundary conditions**, discussed in section 2.3.

3. **End of the algorithm.**

A flowchart of the LBM algorithm is shown in Figure 2.3.

## 2.3 Initial and boundary conditions

The appropriate choice of initial and boundary conditions, consistent with the studied problem, is an integral part of the subsequent numerical simulation. Due to the specific mesoscopic description of the fluid within LBM, careful attention must be given to the initial and boundary conditions used in the practical part of this work. Therefore, the chosen initial and boundary conditions are described in more detail below.
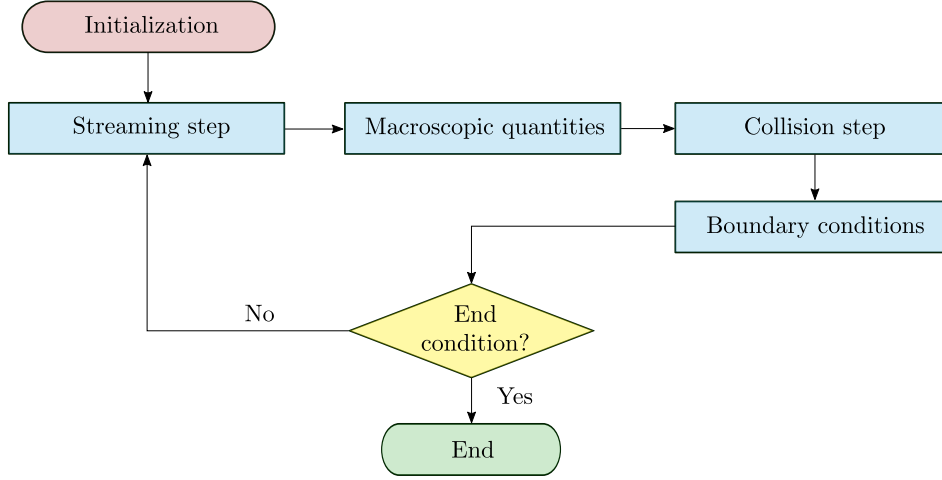
Figure 2.3: Flowchart of the LBM algorithm.

### 2.3.1 Initial Condition

Let us now consider the domain defined by relations (2.8). To set the initial condition, we use the equilibrium distribution function $f^{\text{eq}}$, which has the form

$$f_k^{\text{eq}} = \rho w_k \left( 1 + \frac{\boldsymbol{\xi}_k \cdot \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{\xi}_k \cdot \boldsymbol{u})^2}{2c_s^4} - \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2c_s^2} \right), \quad k \in \{1, \ldots, 27\}, \tag{2.17}$$

where $w_k$ represents the weights specific to the chosen velocity model. For the D3Q27 model, these weights are given as [2]

$$w_k = \begin{cases} \frac{8}{27}, & k = 1, \\ \frac{2}{27}, & k = 2, 3, \ldots, 7, \\ \frac{1}{54}, & k = 8, 9, \ldots, 19, \\ \frac{1}{216}, & k = 20, 21, \ldots, 27. \end{cases} \tag{2.18}$$

The initial distribution $\rho$, and the velocity $\boldsymbol{u}$, are defined as $\rho_{\text{ini}}$ and $\boldsymbol{u}_{\text{ini}}$, respectively. Then, for each node $\boldsymbol{x} \in \hat{\Omega}$ at time $t = 0$, we have

$$f_k(\boldsymbol{x}, 0) = f_k^{\text{eq}}(\rho_{\text{ini}}(\boldsymbol{x}), \boldsymbol{u}_{\text{ini}}(\boldsymbol{x})), \quad k \in \{1, \ldots 27\}. \tag{2.19}$$

In this approach to setting the initial condition, we assume that the non-equilibrium part of the distribution functions $f_k^{\text{neq}} = f_k - f_k^{\text{eq}}$ can be neglected, and the distribution functions can be approximated by their equilibrium part. A significant advantage of this choice of initial condition approximation is its straightforward implementation, and therefore, it is used in this work, although other more sophisticated options exist, such as those discussed in [1].

### 2.3.2 Boundary Conditions

#### 2.3.2.1 Bounce-back Boundary Condition

The first boundary condition discussed is the bounce-back boundary condition, specifically its *full-way* variant. The bounce-back boundary condition is typically used at the interface between a fluid and a solid. Its advantage is that it satisfies the no-slip boundary condition at the interface, and its implementation is straightforward. The principle of the bounce-back boundary condition is that at the interface, the

14

distribution functions corresponding to particles with microscopic velocity $\xi_k$ are reflected back into the directions from which they arrived at the node, with velocity $\xi_{\bar{k}} = -\xi_k$.

When using this boundary condition, the interface is located halfway between the fluid and solid nodes. This does not cause issues when modeling flow around straight walls parallel to the grid, but for curved boundaries that are not parallel to the grid, the bounce-back method leads to a "staircase" shape of the boundary, which does not provide a good approximation of the actual interface position.

The fullway variant of the bounce-back boundary condition requires two time steps for the reflection of the hypothetical particles. During this time, the particles travel to the solid nodes, where their direction is reversed in the next streaming step, as schematically shown in Figure **??**.



(a) Step before the streaming at time $t - \Delta t$.

(b) Step after streaming at time $t$.

(c) The distribution functions are reversed.

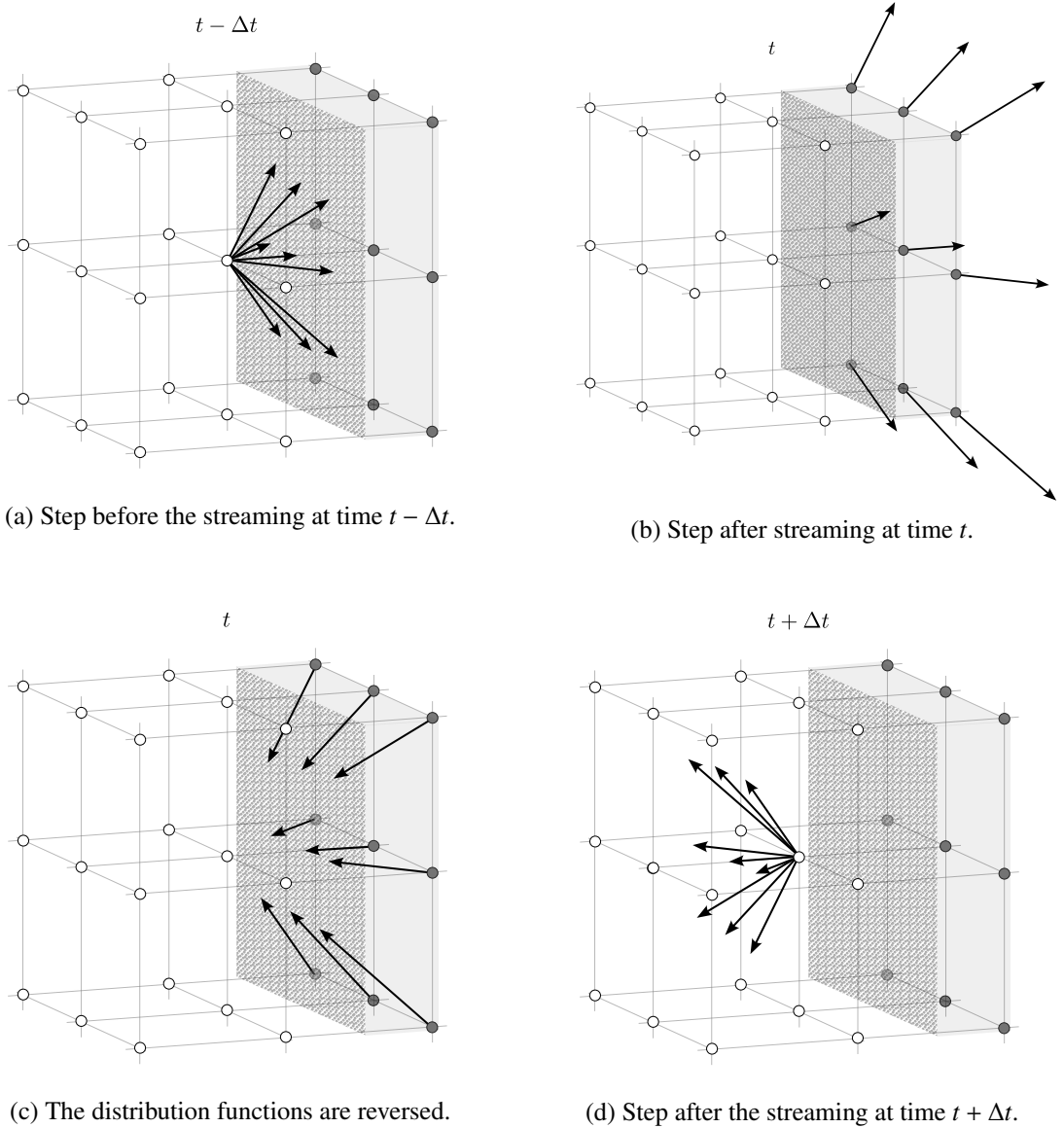(d) Step after the streaming at time $t + \Delta t$.

Figure 2.4: Schematic representation of the fullway bounce-back boundary condition for the D3Q27 velocity model. White points represent the fluid sites and gray points represent the wall sites, with the wall shown as a gray plane.

It is worth noting that there is also a *halfway* variant of the bounce-back boundary condition, which

requires only one time step for its realization. Details can be found, for example, in [2]. However, in this work, we will limit ourselves to the fullway variant.

#### 2.3.2.2  Equilibrium Boundary Condition

One option for approximating unknown distribution function values at the boundary nodes is to use the equilibrium distribution function, defined as [1]

$$f_i(\boldsymbol{x}, t) = f_i^{(\text{eq})}(\rho(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}, t)), \quad \forall k \in \{1, \ldots, 27\}, \forall t \in \hat{\mathcal{I}}. \tag{2.20}$$

The advantage of this approximation is its simple implementation, while the disadvantage is the neglect of the non-equilibrium part of the distribution function [1].

## 2.4   Notes on the implementation

As mentioned in the introduction, the numerical solution using LBM was based on a code developed at the Department of Mathematics of FNSPE, CTU in Prague, which is used to solve the Navier-Stokes equations for a Newtonian incompressible fluid. The program is implemented in C++ using the TNL library [7, 8] and employs parallelization on a GPU using the CUDA platform. The used variant of the lattice Boltzmann method CuLBM is implemented in the code for the D3Q27 model.

For the purposes of this work, the code developed within a previous bachelor's thesis [9] was extended, in which several modifications were made compared to the code developed at the Department of Mathematics of FNSPE, specifically:

- implementation of the stress tensor integration method for force calculation using difference,

- implementation of various methods for the local calculation of the stress tensor,

- implementation of interpolation boundary conditions,

- calculation of monitored quantities and their subsequent output to files.

# Chapter 3

# Mathematical optimization

Mathematical optimization, also known as mathematical programming, is a broad field that covers various disciplines, including linear and nonlinear optimization, convex programming, integer programming, and more. The goal of this chapter is to summarize the key concepts and techniques relevant to the scope of this work. Specifically, we will focus on methods for solving optimization problems with constraints and those where the objective function is generally unknown and its evaluation is computationally expensive. Classic methods applicable to unconstrained optimization, such as the Davidon-Fletcher-Powell [10] and Broyden-Fletcher-Goldfarb-Shanno [11] algorithms, while fundamental, fall outside the scope of this work and details on these methods can be found for instance in [12].

This chapter will focus on methods that address the specific challenges posed by constrained and generally unknown objective functions. First, a general optimization problem is defined, followed by an introduction to basic techniques for solving constrained problems. Next, black-box optimization methods are discussed. Finally, the proposed optimization framework used in this work is presented.

## 3.1 General optimization problem

Let $m, n, q \in \mathbb{N}$. Define the continuous functions $f : \mathbf{D} \to \mathbb{R}$, $g : \mathbf{D} \to \mathbb{R}^m$, $h : \mathbf{D} \to \mathbb{R}^q$, where $\mathbf{D} = \mathrm{Dom}\,(f) \cap \mathrm{Dom}\,(g) \cap \mathrm{Dom}\,(h)$, i.e., $\mathbf{D}$ is the intersection of the domains of the given functions. Next, define the set

$$\mathbf{X} = \{\, x \in \mathbf{D} \subseteq \mathbb{R}^n \mid g(x) \leq \mathbf{0} \wedge h(x) = \mathbf{0} \,\}, \tag{3.1}$$

where the inequality $g \leq \mathbf{0}$ and equality $h = \mathbf{0}$ are understood component-wise. The general goal of mathematical optimization is to solve the problem

$$\min_{x \in \mathbf{X}} f(x). \tag{3.2}$$

The function $f$ being minimized is called the objective function, $\mathbf{D}$ is referred to as the domain of the problem, and $\mathbf{X}$ is called the set of admissible solutions of the problem [12]. Note that, henceforth, $f$ denotes only the objective function and not the distribution function discussed in Chapter **??**.

When classifying optimization problems, we refer to what are known as constraints. These are determined by the definition of the set $\mathbf{X}$, i.e., the equality and inequality conditions for the functions $g$ and $h$, and by the domain $\mathbf{D}$. Constraints defined by $g(x) \leq \mathbf{0} \wedge h(x) = \mathbf{0}$ are called explicit constraints, while those determined by the domain $\mathbf{D}$ are called implicit constraints.

The optimal solution of the problem (3.2) is denoted by $x^\star \in \mathbf{X}$ and is defined as

$$x^\star = \underset{x \in \mathbf{X}}{\arg\min} \; f(x). \tag{3.3}$$

Note that the optimal solution may not be unique, and we refer to the set of all optimal solutions as the optimal set. It is also important to recognize that the search for an optimal solution can equivalently be formulated as finding the maximum of the function $-f$ over the same set $\mathbf{X}$, enabling the use of the same techniques for solving maximization problems [12, 13].

## 3.2   Solving constrained problems

We will now consider the problem 3.2, where the set of admissible solutions is given by 3.1. In this section, we describe how to generally solve this problem by converting it into a sequence of unconstrained optimization problems. There exist several other methods for solving constrained problems, but here we will focus on the penalty and the barrier methods.

### 3.2.1   Penalty methods

As mentioned earlier, penalty methods convert constrained optimization problems into unconstrained ones. The fundamental principle of penalty methods is to incorporate the conditions that define the set of admissible solutions by adding a penalty term to the objective function, which reflects the degree of violation of those conditions [12]. The penalty function is defined as a continuous scalar function on $\mathbb{R}^n$ that satisfies

$$\begin{aligned} p(x) &= 0, \quad \forall x \in \mathbf{X}, \\ p(x) &> 0, \quad \text{otherwise.} \end{aligned} \tag{3.4}$$

A typical choice for the penalty function is, e.g.,

$$p(x) = \sum_{j=1}^{m} \Big( \max\big\{ g_j(x), 0 \big\} \Big)^2 + \sum_{j=1}^{q} h_j(x)^2. \tag{3.5}$$

Using the penalty function $p$, we construct the modified objective function

$$\phi(x, r) = f(x) + r p(x), \tag{3.6}$$

where $r > 0$ is called the penalty coefficient [12]. From the definition of the penalty function, it can be seen that the values of the modified objective function $\phi(x, r)$ differ from the original function $f$ only for such $x$ that violate the specified condition.

To solve the constrained problem, we iteratively construct a new term in an increasing sequence of penalty coefficients $r_1, r_2, \ldots$, for which we solve the unconstrained optimization problem for the modified function $\phi(x, r_k)$. The optimal solution $x_k$ of this unconstrained problem is then used as the starting point for the next iteration. This process is repeated until the condition $r_k p(x_k) < \varepsilon$ for some $\varepsilon > 0$ is satisfied. When this condition is met, $x_k$ can be considered a sufficiently accurate approximation of the solution to the constrained problem. It should be noted that penalty methods allow searching for an optimal solution outside the set of admissible solutions during the iterations [13], and thus are

categorized as exterior point methods. A key assumption of penalty methods is that the domain of the problem satisfies $\mathbf{D} = \mathbb{R}^n$.

The construction of the sequence of penalty parameters and the principle of the penalty method are illustrated in Figure 3.1 on a trivial example of minimizing the function $f(x) = 0.5x$ with the constraint $g(x)$ :
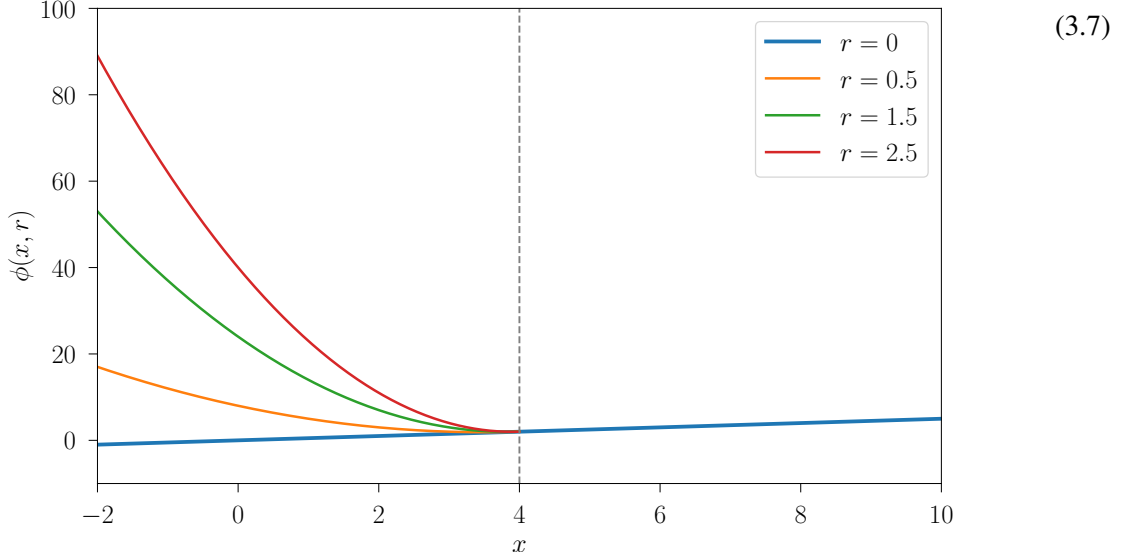
$$(3.7)$$



Figure 3.1: An illustration of the penalty method applied to minimizing the function $f(x) = 0.5x$ with the constraint $g(x) = 4 - x \leq 0$. Different shapes of the modified objective function $\phi(x, r)$ depending on the value of the penalty parameter $r$ are distinguished by color. The condition defining the set of admissible solutions is indicated by the gray dashed line. The set of admissible solutions lies in the half-plane to the right of this gray dashed line.

### 3.2.2 Barrier method

The barrier method operates on a principle similar to that of the penalty method, but its main difference is that, during the iterative process of finding an optimal solution to a constrained problem, it ensures that the solution estimates always remain within the interior of the feasible set, which is defined as

$$\mathbf{X}^{\mathrm{o}} = \left\{ \boldsymbol{x} \in \mathbf{D} \subseteq \mathbb{R}^n \,\middle|\, \boldsymbol{g}(\boldsymbol{x}) < \mathbf{0} \right\}. \tag{3.8}$$

Methods that satisfy this condition are generally referred to as interior-point methods [13].

As with penalty methods, we account for the conditions defining the feasible set by adding a new term to the objective function, which reflects the degree to which these conditions are violated. The barrier function $B$ is a continuous scalar function on $\mathbf{X}^{\mathrm{o}}$ that satisfies the condition

$$(\exists j \in \{1, 2, \ldots, m\})(\lim_{\substack{x \to y \\ \mathbf{X}^{\mathrm{o}}}} g_j(\boldsymbol{x}) = 0) \Rightarrow \lim_{\substack{x \to y \\ \mathbf{X}^{\mathrm{o}}}} B(\boldsymbol{x}) = +\infty. \tag{3.9}$$

A typical choice for the barrier function is the logarithmic barrier function

$$B(\boldsymbol{x}) = -\sum_{j=1}^{m} \ln\left(-g_j(\boldsymbol{x})\right), \tag{3.10}$$

19

or the reciprocal barrier function

$$B(\boldsymbol{x}) = -\sum_{j=1}^{m} \frac{1}{g_j(\boldsymbol{x})}. \tag{3.11}$$

Using the barrier function $B$, we construct the modified objective function

$$\phi(\boldsymbol{x}, r) = f(\boldsymbol{x}) + rB(\boldsymbol{x}), \tag{3.12}$$

where $r > 0$ [13].

To solve the constrained problem using the barrier method, at each iteration we construct a new term in a strictly decreasing sequence of positive parameters $r_1, r_2, \ldots$, for which we solve the unconstrained optimization problem for the modified function $\phi(\boldsymbol{x}, r_k)$. The vector $\boldsymbol{x}_k = \text{argmin}_{\boldsymbol{x} \in \mathbf{X}^\circ}(f(\boldsymbol{x}) + r_k B(\boldsymbol{x}))$, obtained by optimizing the unconstrained problem, is used as the starting point for the next iteration. This process is repeated until the condition $r_k < \varepsilon$ for some $\varepsilon > 0$ is satisfied, at which point $\boldsymbol{x}_k$ is considered a sufficient approximation to the solution of the constrained problem [13].

The construction of the parameter sequence $(r_k)_{k \in \mathbb{N}}$ and the principle of the barrier method are illustrated in Figure 3.2 using the example of minimizing the function $f(x) = 0.5x$ with the constraint $g(x) = 4 - x \le 0$ and the choice of the reciprocal barrier function. The modified objective function in this case is

$$\phi(x, r) = 0.5x - \frac{r}{4 - x}. \tag{3.13}$$
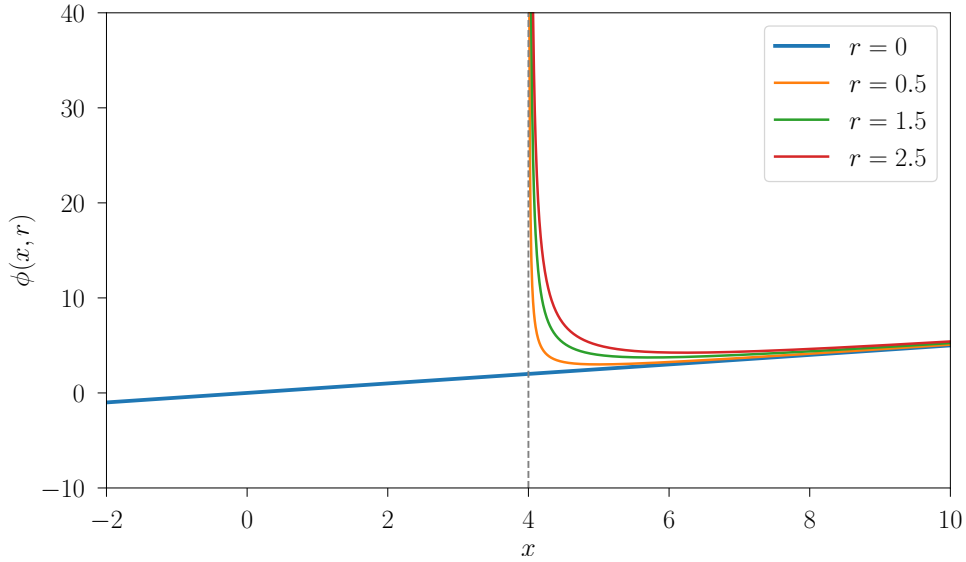


Figure 3.2: An illustration of the barrier method applied to minimizing the function $f(x) = 0.5x$ with the constraint $g(x) = 4 - x \le 0$. Different shapes of the modified objective function $\phi(x, r)$ depending on the value of the barrier parameter $r$ are distinguished by color. The condition defining the set of feasible solutions is indicated by the gray dashed line. The set of feasible solutions lies in the half-plane to the right of this gray dashed line.

Finally, we define a specific choice of the barrier function $B_\infty$ called the extreme barrier function [14]. The extreme barrier function mirrors the asymptotic behavior of the aforementioned barrier functions and

is given by

$$B_\infty(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \mathbf{X},$$
$$B_\infty(\boldsymbol{x}) = +\infty, \quad \text{otherwise.}$$
(3.14)

The modified objective function is then given by

$$f_\infty(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbf{X},$$
$$f_\infty(\boldsymbol{x}) = +\infty, \quad \text{otherwise.}$$
(3.15)

## 3.3 Black-box optimization

In practice, we often need to optimize an objective function $f$ whose exact form and derivative are unknown. This is common in numerical simulations, where the function can only be evaluated at specific points. Moreover, evaluating the function at a point may be difficult, time-consuming, or computationally expensive. As a result, standard optimization algorithms are not well-suited for these problems.

The discipline that deals with problems where the objective function (or constraints) is given by a so-called black-box[1], is called black-box optimization (hereafter referred to as BBO). In BBO, it is typically not assumed that the objective function is continuous or differentiable [14, 15, 16].

It is worth noting that black-box optimization is often confused with derivative-free optimization (DFO), which encompasses methods and techniques for objective functions whose derivatives are unknown or difficult to compute [14, 15, 17]. These two disciplines share many common characteristics, but they differ primarily in that, within DFO, the formula for calculating the derivative of the objective function may still be known. Furthermore, BBO includes heuristic methods, whereas DFO focuses mainly on methods that can be reliably analyzed mathematically in terms of convergence and stopping criteria, which is often not possible for BBO methods [14]. Therefore, although the terms BBO and DFO are often used interchangeably, in this work, we will treat them as two distinct disciplines [14].

Additionally, it should be noted that various classifications of methods within BBO can be found in the literature. In this work, we will adhere to the classification presented in [14], distinguishing between heuristic methods, direct search methods, and methods based on surrogate models. Each of these classes will be briefly described in this section.

### 3.3.1 Heuristic Methods

Heuristic optimization methods often rely on different predefined rules or even trial and error when seeking the solution of an optimization problem. These methods usually do not guarantee optimal solutions, but they are often effective for finding near-optimal results in a reasonable amount of time. Heuristic methods include genetic algorithms, detailed in [14], along with various other heuristic approaches.

In this section, however, we will focus on a different widely used heuristic method, the Nelder-Mead method, also known as the simplex method[2][3][19].

The Nelder-Mead method finds a solution to an optimization problem by iteratively constructing simplexes. The process begins by initializing a starting simplex. The objective function is then evaluated

---

[1]In programming, a black-box refers to a system whose internal mechanisms are unknown to the user. This means that the user generally has access only to the system's input and output [14].

[2]A simplex in $\mathbb{R}^n$ is defined as a bounded convex polytope (a generalization of a polyhedron to any dimension) with a non-empty interior and exactly $n + 1$ vertices [14].

[3]The term "simplex method" more commonly refers to the algorithm used to find the optimal solution in linear programming. This algorithm was developed by George Dantzig [18].

at each vertex of this simplex. In each subsequent iteration, the simplex is transformed in order to move closer to the position of the sought stationary point of the objective function. The transformation of the simplex involves manipulating its points using predefined operations – expansion, reflection, contraction (inner and outer), and shrinking, which are schematically illustrated in Figure 3.3.



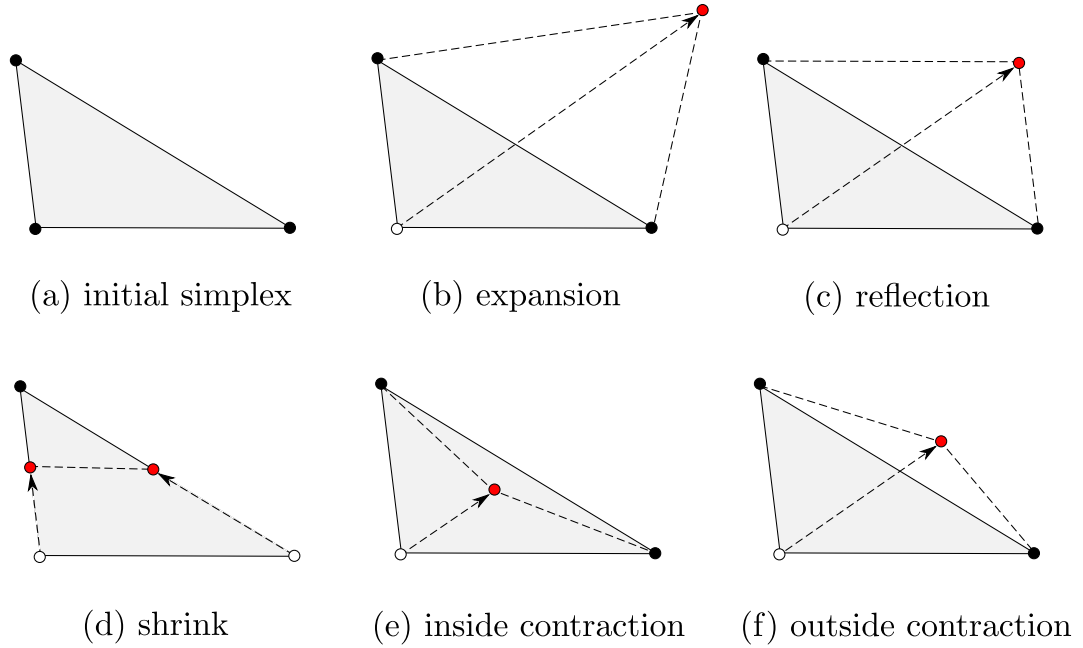| (a) initial simplex | (b) expansion | (c) reflection |
| (d) shrink | (e) inside contraction | (f) outside contraction |

Figure 3.3: A schematic representation of the operations used to transform simplexes in the Nelder-Mead method. The vertices generated by applying each operation are shown in red. For clarity, the operations are depicted in $\mathbb{R}^2$.

The transformations performed during each iteration are determined by comparing the function values at the vertices of the simplex. The newly formed simplex shares either exactly one vertex or exactly $n$ vertices with the simplex from the preceding iteration. The algorithm continues to iteratively transform the simplex until a stopping condition (specified by the user) is met [14]. Details of the Nelder-Mead method, including the algorithm's description and the choice of stopping condition, are discussed in [14, 15, 19].

The heuristic nature of the Nelder-Mead method stems from the fact that its principle is based on a somewhat random search of the space using predefined rules. Several iterations of space exploration using simplexes, for a specific choice of initial simplex and a specific function, are shown in Figure 3.4. While the convergence of this method has been proven, it is not guaranteed that the method will always converge to a stationary point [14]. It should be noted that the Nelder-Mead method was primarily developed for unconstrained optimization problems, but it can be adapted for constrained optimization problems.

---

**Algorithm 1** Nelder-Mead algorithm

---

**Require:** Initial simplex $Y^0 = \{y^0, y^1, \ldots, y^n\}$, function $f : \mathbb{R}^n \to \mathbb{R}$, parameters $\delta^e, \delta^{oc}, \delta^{ic}, \gamma$, iteration counter $k \leftarrow 0$

**Ensure:** Approximate minimum of $f$

1: **procedure** NELDER-MEAD($Y^0$)
2:      Reorder $Y^k$ so that $f(y^0) \leq f(y^1) \leq \cdots \leq f(y^n)$
3:      Set $f_{\text{best}}^k = f(y^0)$
4:      **while** stopping condition not met **do**

---

**Reflection**

---

5:          Compute centroid $x^c = \frac{1}{n} \sum_{i=0}^{n-1} y^i$
6:          Set reflection point $x^r = x^c + \delta^r(x^c - y^n)$
7:          Compute $f^r = f(x^r)$
8:          **if** $f_{\text{best}}^k \leq f^r < f(y^{n-1})$ **then**
9:             Set $Y^{k+1} = \{y^0, y^1, \ldots, y^{n-1}, x^r\}$
10:             Increment $k \leftarrow k + 1$ and continue
11:          **end if**

---

**Expansion**

---

12:          **if** $f^r < f_{\text{best}}^k$ **then**
13:             Set expansion point $x^e = x^c + \delta^e(x^r - x^c)$
14:             Compute $f^e = f(x^e)$
15:             **if** $f^e < f^r$ **then**
16:                 Set $Y^{k+1} = \{y^0, y^1, \ldots, y^{n-1}, x^e\}$
17:             **else**
18:                 Set $Y^{k+1} = \{y^0, y^1, \ldots, y^{n-1}, x^r\}$
19:             **end if**
20:             Increment $k \leftarrow k + 1$ and continue
21:          **end if**

---

**Contraction**

---

22:          **if** $f^r \geq f(y^n)$ **then**
23:             **Outside Contraction:** Compute $x^{oc} = x^c + \delta^{oc}(x^c - y^n)$
24:             Compute $f^{oc} = f(x^{oc})$
25:             **if** $f^{oc} < f(y^n)$ **then**
26:                 Set $Y^{k+1} = \{y^0, y^1, \ldots, y^{n-1}, x^{oc}\}$
27:             **else**
28:                 **Shrink:** Set $Y^{k+1} = \{y^0, y^0 + \gamma(y^1 - y^0), \ldots, y^0 + \gamma(y^n - y^0)\}$
29:             **end if**
30:             Increment $k \leftarrow k + 1$ and continue
31:          **else**
32:             **Inside Contraction:** Compute $x^{ic} = x^c + \delta^{ic}(x^c - y^n)$
33:             Compute $f^{ic} = f(x^{ic})$
34:             **if** $f^{ic} < f(y^n)$ **then**
35:                 Set $Y^{k+1} = \{y^0, y^1, \ldots, y^{n-1}, x^{ic}\}$
36:             **else**
37:                 Set $Y^{k+1} = \{y^0, y^0 + \gamma(y^1 - y^0), \ldots, y^0 + \gamma(y^n - y^0)\}$
38:             **end if**
39:             Increment $k \leftarrow k + 1$ and continue
40:          **end if**
41:      **end while**
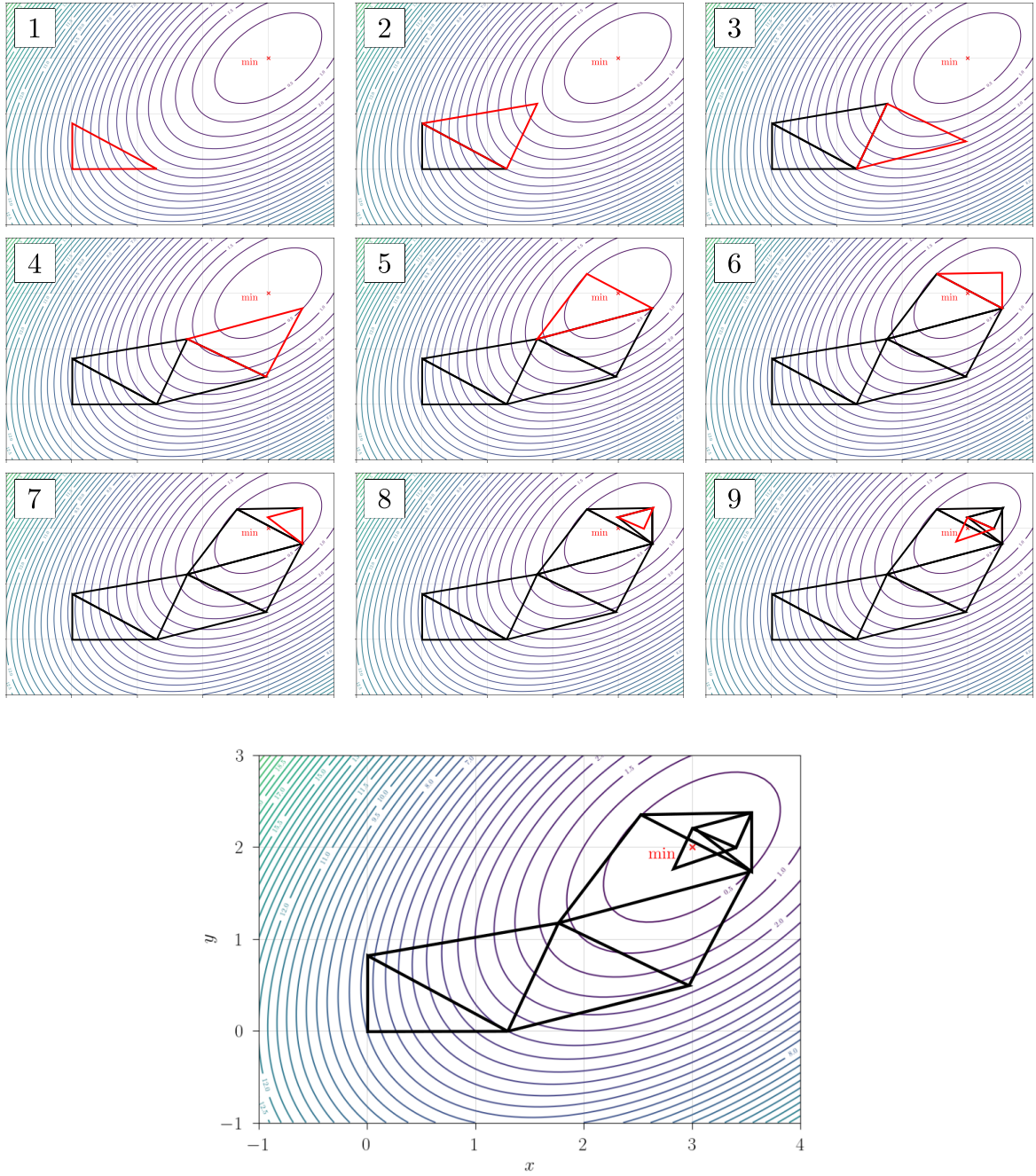42: **end procedure**

---

23

Figure 3.4: Several iterations of the Nelder-Mead method for a specific choice of the initial simplex when minimizing the function $x^2 - 4x + y^2 - y - xy + 7$, with the minimum at the point (3,2) marked by a red cross.

### 3.3.2 Direct search methods

Among direct search methods, we will describe the *generalized pattern search* (hereafter GPS) method [20] and the *mesh adaptive direct search* (hereafter MADS) method [21].

To describe the GPS algorithm, it is necessary to define a mesh, which is used to describe the search sets within the GPS algorithm. Let $\mathbf{G} \in \mathbb{R}^{n \times n}$ be invertible and $\mathbf{Z} \in \mathbb{Z}^{n \times p}$, $n, p \in \mathbb{N}$. Assume that every vector from $\mathbb{R}^n$ can be expressed as a linear combination of the columns of matrix $\mathbf{Z}$ (treated as vectors),

such that all the coefficients in this linear combination are non-negative. Furthermore, let $\mathbf{D} = \mathbf{GZ}$. The mesh $\mathbf{M}$ generated by $\mathbf{D}$ centered at point $\boldsymbol{x}$ is defined as

$$\mathbf{M} = \{\boldsymbol{x} + \delta\,\mathbf{D}y\,|\,y \in \mathbb{N}^p\}, \tag{3.16}$$

where $\delta$ is called the mesh size parameter [14, 20]. In each iteration of the GPS algorithm, the shape of the mesh generally changes, as it is always centered at the point representing the best estimate in that iteration, and the size of the mesh step also changes. Let $\boldsymbol{x}_k$ and $\delta_k$ represent the estimate of the solution and the mesh size in the $k$-th iteration, respectively. We then define the mesh in the $k$-th iteration, denoted as $\mathbf{M}_k$, as

$$\mathbf{M}_k = \{\boldsymbol{x}_k + \delta_k\,\mathbf{D}y\,|\,y \in \mathbb{N}^p\}. \tag{3.17}$$

Note that the columns of matrix $\mathbf{D}$, as defined above, can be interpreted as the possible directions in which the GPS algorithm searches the optimization space [14, 20]. Examples of search directions and meshes generated by different matrices $\mathbf{G}$ and $\mathbf{Z}$ are presented in Figure 3.5.
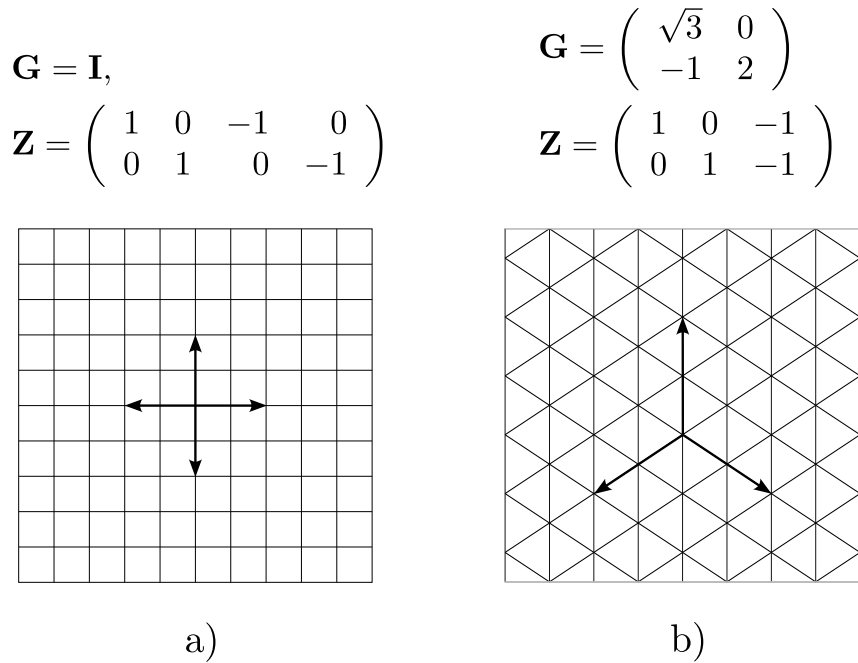
$$\mathbf{G} = \mathbf{I},$$

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} \sqrt{3} & 0 \\ -1 & 2 \end{pmatrix}$$

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$



a)　　　　　　　　　　　　　b)

Figure 3.5: Examples of search directions and meshes in $\mathbb{R}^2$ with obtained by different choices of $\mathbf{G}$ and $\mathbf{Z}$. The mesh points are at the intersections of the lines, the arrows represent possible search directions.

After initializing the necessary starting parameters, each iteration of the GPS algorithm is divided into two main steps. The first step is called the search step. During the search step, a finite set $S_k$ of candidate mesh points, selected according to a strategy specified by the user, is evaluated by computing the objective function at each one of the points. If none of the evaluated points represents an improvement over the value $f(\boldsymbol{x}_k)$, the poll step follows. In the poll step, the objective function is evaluated at all

neighboring mesh points of $x_k$. The poll set in $k$-th iteration is defined as $P_k = \{x_k + \delta_k d \,|\, d \in \mathbf{D}\}$. If none of the evaluated points represents an improvement over the value $f(x_k)$, we set $x_{k+1} = x_k$ and decrease the mesh size, i.e., $\delta_{k+1} < \delta_k$. However, if a point that improves the estimate of the solution is found in either the search or the poll step, this point is set as $x_{k+1}$, and the mesh size is increased, i.e., $\delta_{k+1} > \delta_k$ [14, 20].

The changes described above define a new mesh $\mathbf{M}_k$ in each iteration. The mesh changes throughout the GPS algorithm. The algorithm terminates when $\delta_{k+1} < \varepsilon$ for some user-specified $\varepsilon > 0$. It can be shown that the mesh step converges to zero, and under appropriate assumptions, the solution estimates converge to a stationary point of the objective function. Details can be found in [14]. It should be noted that the convergence of GPS has been proven for unconstrained problems [14]. The full GPS algorithm is presented in Algorithm 2.

---

**Algorithm 2** Generalized Pattern Search (GPS) for unconstrained optimization

---

**Require:** Function $f : \mathbb{R}^n \to \mathbb{R}$, initial point $x_0$, initial mesh size parameter $\delta_0$, positive spanning matrix $\mathbf{D}$, mesh size adjustment parameter $\tau \in (0, 1)$, stopping tolerance $\epsilon_{\text{stop}}$, iteration counter $k \leftarrow 0$

**Ensure:** Approximate solution $x^*$

 1: **procedure** GPS($x_0$)
 2:      **while** $\delta_k > \epsilon_{\text{stop}}$ **do**

**1. Search**

 3:          Define a finite subset $S_k$ of the mesh $\mathbf{M}_k$
 4:          **if** $f(t) < f(x_k)$ for some $t \in S_k$ **then**
 5:             Set $x_{k+1} \leftarrow t$ and $\delta_{k+1} \leftarrow \tau^{-1}\delta_k$
 6:             **continue**
 7:          **else**
 8:             Go to Poll step
 9:          **end if**

**2. Poll**

10:          Select a positive spanning set $\mathbf{D}_k \subseteq \mathbf{D}$
11:          Define $P_k = \{x_k + \delta_k d : d \in \mathbf{D}_k\}$
12:          **if** $f(t) < f(x_k)$ for some $t \in P_k$ **then**
13:             Set $x_{k+1} \leftarrow t$ and $\delta_{k+1} \leftarrow \tau^{-1}\delta_k$
14:          **else**
15:             $x_k$ is a mesh local optimizer
16:             Set $x_{k+1} \leftarrow x_k$ and $\delta_{k+1} \leftarrow \tau\delta_k$
17:          **end if**

**3. Termination**

18:          **if** $\delta_{k+1} \leq \epsilon_{\text{stop}}$ **then**
19:             **terminate**
20:          **else**
21:             Increment $k \leftarrow k + 1$ and continue
22:          **end if**
23:      **end while**
24: **end procedure**

---

We now turn our attention to the MADS algorithm, which generalizes the GPS algorithm by allowing for a different set of polling directions. The key difference is that MADS introduces the concept of a frame, which allows the poll directions to form a dense subset in $\mathbb{R}^n$ as the algorithm progresses [14, 15]. The frame $\mathbf{F}_k$ at iteration $k$ is defined as

$$\mathbf{F}_k = \{\boldsymbol{x} \in \mathbf{M}_k \mid \|\boldsymbol{x} - \boldsymbol{x}_k\|_\infty \le \Delta_k b\}, \tag{3.18}$$

where $M_k$ represents the current mesh, $\Delta_k$ is the frame size parameter, satisfying $\delta_k \le \Delta_k$, and $b = \max\{\|d\|_\infty \mid d \in \mathbf{D}\}$. The extent of the frame is determined by the parameter $\Delta_k$, and the polling directions are taken from this frame.

Each iteration of the MADS algorithm begins similarly to GPS, with a search step where a finite set $S_k$ of candidate points, selected based on the current mesh, is evaluated by computing the objective function at each of the points. If none of these points improves upon the current best solution, the algorithm proceeds to the poll step.

The poll set $P_k$ is a subset of points selected from $\mathbf{F}_k$ and $\mathbf{M}_k$. Trivially, $P_k \subseteq \mathbf{F}_k \subseteq \mathbf{M}_k$. Importantly, the mesh size parameter $\delta_k$ is allowed to decrease more rapidly than the enabling the poll directions to asymptotically become arbitrarily dense [21]. This aspect is crucial, as it ensures that MADS can explore directions in a finer and more systematic manner than GPS, leading to better convergence properties.
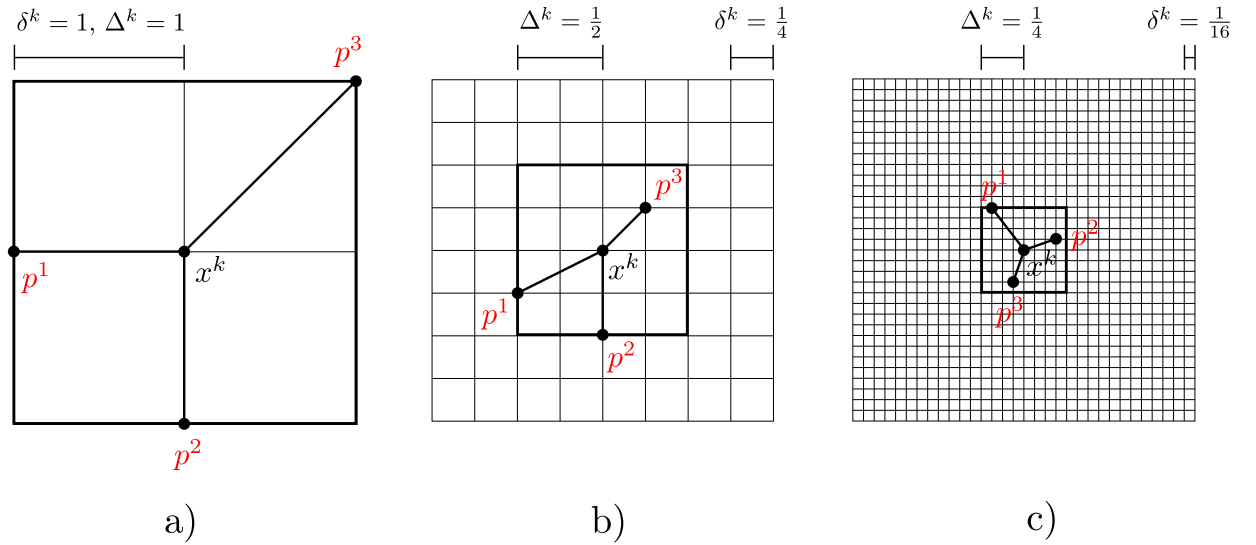


Figure 3.6: Examples of meshes and frames $\mathbb{R}^2$ for different values of $\delta_k$ and $\Delta_k$

If an improvement is found in the poll step, the new point is set as $\boldsymbol{x}_{k+1}$, and the frame size $\Delta_k$ and mesh size $\delta_k$ may be increased to encourage further exploration. Conversely, if no improvement is found, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$, and the mesh size is reduced, i.e., $\delta_{k+1} < \delta_k$, to allow for a more local search. The process of shrinking and refining the frame and mesh sizes continues until $\delta_k$ falls below a user-specified threshold, at which point the algorithm terminates.

MADS also incorporates the *extreme barrier function* to handle constraints, similarly to GPS. For constrained optimization problems, the objective function is modified into an extreme barrier function [14], which penalizes any infeasible points by assigning them an infinite objective value. This simple yet effective strategy ensures that the optimization remains focused on feasible regions of the search space, and the algorithm converges to a stationary point even for constrained problems. A full description of the MADS algorithm and its implementation details can be found in [14].

---

**Algorithm 3** Mesh Adaptive Direct Search (MADS)

---

**Require:** Function $f_\infty : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$, initial point, initial frame size parameter $\Delta_0$, positive spanning
    matrix $\mathbf{D}$, mesh size adjustment parameter $\tau \in (0, 1)$, stopping tolerance $\epsilon_{\text{stop}}$, iteration counter $k \leftarrow 0$
**Ensure:** Approximate solution $x^*$

  1: **procedure** MADS($x_0$)
  2:     **while** $\Delta_k > \epsilon_{\text{stop}}$ **do**

---

**1. Parameter Update**

---

  3:         Set the mesh size parameter $\delta_k = \min\{\Delta_k, (\Delta_k)^2\}$

---

**2. Search**

---

  4:         Define a finite set $S_k \subset \mathbf{M}_k$ such that:
  5:         **if** $f_\infty(t) < f_\infty(x_k)$ for some $t \in S_k$ **then**
  6:             Set $x_{k+1} \leftarrow t$ and $\Delta_{k+1} \leftarrow \tau^{-1}\Delta_k$
  7:             **continue**
  8:         **else**
  9:             Go to Poll step
10:         **end if**

---

**3. Poll**

---

11:         Select a positive spanning set $\mathbf{D}_k$ and define:
12:         $P_k = \{x_k + \delta_k d : d \in \mathbf{D}_k\}$, a subset of the frame $\mathbf{F}_k$ with extent $\Delta_k$
13:         **if** $f_\infty(t) < f_\infty(x_k)$ for some $t \in P_k$ **then**
14:             Set $x_{k+1} \leftarrow t$ and $\Delta_{k+1} \leftarrow \tau^{-1}\Delta_k$
15:         **else**
16:             Set $x_{k+1} \leftarrow x_k$ and $\Delta_{k+1} \leftarrow \tau\Delta_k$
17:         **end if**

---

**4. Termination**

---

18:         **if** $\Delta_{k+1} \leq \epsilon_{\text{stop}}$ **then**
19:             **terminate**
20:         **else**
21:             Increment $k \leftarrow k + 1$ and continue
22:         **end if**
23:     **end while**
24: **end procedure**

---

### 3.3.3 Optimization Using a Surrogate Model

In cases where evaluating the objective function at a specific point is time-consuming or computationally expensive, it can be useful to employ a surrogate for the objective function during optimization. We define a surrogate model of the given problem as the problem

$$\min_{x \in \tilde{\mathbf{X}}} \tilde{f}(x), \tag{3.19}$$

where

$$\tilde{\mathbf{X}} = \left\{ x \in \mathbf{D} \subseteq \mathbb{R}^n \,\middle|\, \tilde{g}(x) \leq \mathbf{0} \wedge \tilde{h}(x) = \mathbf{0} \right\}, \tag{3.20}$$

and the functions $\tilde{f}, \tilde{g}$, and $\tilde{h}$ have characteristics similar to those of the functions $f, g$, and $h$ in the original problem. The characteristics of $\tilde{f}, \tilde{g}$, and $\tilde{h}$ are intentionally left undefined, reflecting the fact that the surrogate model does not necessarily need to be an accurate approximation of the original problem [16, 14, 17]. A good approximative model may not always be a suitable surrogate for optimization purposes, a situation illustrated in Fig. 3.7.
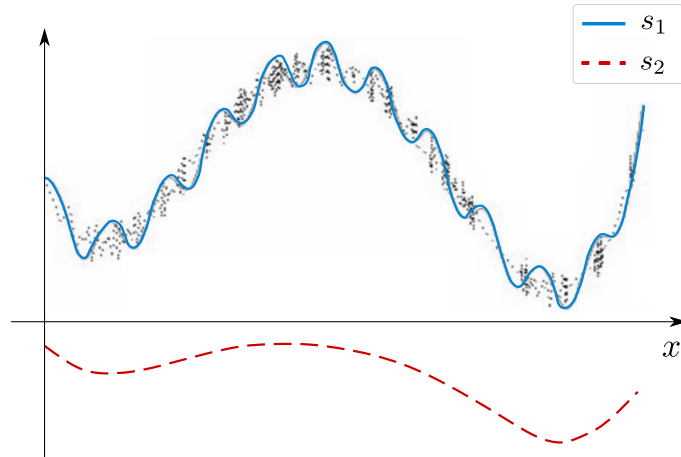


Figure 3.7: An illustration of two surrogate models, $s_1$ and $s_2$. The black points represent noisy values of the objective function. While using surrogate model $s_1$ represents a better choice for approximating the function, it is not suitable for optimization since $s_1$ contains many undesirable stationary points that the original objective function does not have. On the other hand, while surrogate model $s_2$ is not as accurate in approximating the function's values, it is a better choice for optimization because the stationary points of $s_2$ are almost identical to those of the optimized objective function.

Using a surrogate model in optimization is often part of a larger optimization method. Surrogate models can, for example, be used within GPS and MADS methods described in section 3.3.2, where, during the exploration step, we first evaluate the surrogate function $\tilde{f}$ at the same points, sort these values, and then use the sorted set of points to evaluate the original function $f$. This potentially allows us to significantly reduce the time required to complete the exploration step, as sorting the points increases the probability of finding a better estimate of the solution at one of the first examined points [14]. Surrogate models can also be used within other methods to accelerate the process, and their application is discussed in detail in [16, 14].

## 3.4   Popis optimalizačního rámce

Pro použití popsaných optimalizačních metod v rámci problematiky dynamiky tekutin a numerických simulací je potřeba vytvořit plně automatizovaný optimalizační rámec. Je vhodné navrhnout takový rámec, jehož jednotlivé části budou dostatečně modulární, tedy v případě potřeby může být pro vykonání specifického úkonu v rámci procesu optimalizace snadno použita jiná metoda. Celkový navržený rámec zahrnuje několik částí, které dále popíšeme:

1. *Optimalizace*: První část zahrnuje samotnou použitou optimalizační metodu, která řídí celý další proces. V této části je definována řešená úloha společně s případnými požadovanými vazbami. Jak

již bylo zmíněno, je vhodné, aby tato část byla co nejvíce nezávislá na ostaních částech popisovaného optimalizačního rámce. To nám dovolí použít různé metody implementované v jiných programovacích jazycích bez narušení chodu celkového procesu.

V této práci budeme pracovat s metodami L-BFGS (viz sekce **??**) a Nelderovou-Meadovou metodou (viz sekce 3.3.1). Pro implementeci obou těchto metod použijeme volně dostupný balík `Optim.jl` [22] implementovaný v programovacím jazyce Julia. Pro zahrnutí vazeb použijeme v obou případech metodu transformující účelovou funkci na extrémní bariérovou funkci, která byla pro účely této práce implementována nad rámec použitého balíku. Pro výpočet gradientu v rámci metody L-BFGS použijeme automatickou diferenciaci, která je dostupná v rámci balíku `Optim.jl` a jejíž detaily jsou popsány v [22]. Metodu L-BFGS s toutou volbou výpočtu gradientu budeme dále značit L-BFGS(A).

Dále v rámci optimalizačního rámce zařazujeme metodu přímého vyhledávání MADS (viz sekce 3.3.2) implementovanou v programovacím jazyce C++ ve volně dostupné knihovně `NOMAD` [23]. Pro zarhnutí vazeb rovněž použijeme extrémní bariérovou funkci, která je v případě knihovny `NOMAD` její součástí. Podotkněme, že v rámci této práce metoda MADS není využita.

2. *Generování geometrie*: Optimalizační parametry, jejichž hodnota se v každé iteraci optimalizace mění, jsou předány do generátoru geometrie. Pro generování geometrie, která je dále využita v numerické simulaci, byl použit balík `meshgenerator` implementovaný v jazyce Python. Zmíněný balík vznikl pro účely této práce a je blíže popsán v kapitole **??**.

3. *Numerická simulace*: Poslední částí optimalizačního rámce je řešič umožňující vyčíslení optimalizované účelové funkce. V rámci této práce se zabýváme pouze problémy týkajícími se dynamiky tekutin. Pro numerickou simulaci proudění byla použita metoda LBM, která je společně s implementačními poznámkami popsána v kapitole **??**.

Schématicky je propojení jednotlivých částí optimalizačního rámce zobrazeno na obr. 3.8.



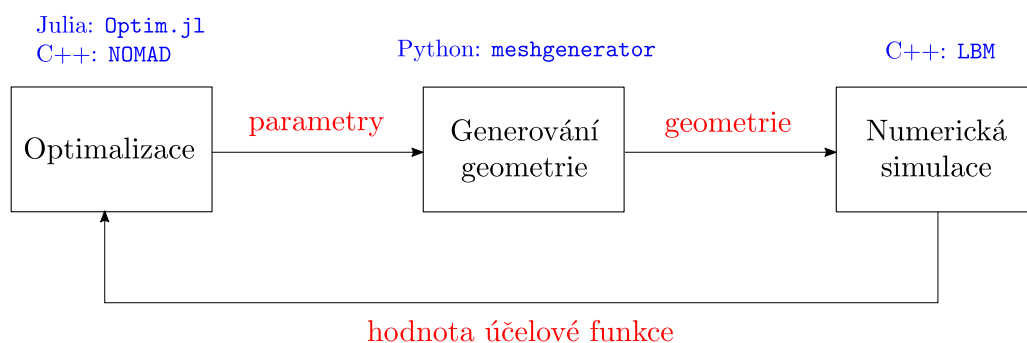Figure 3.8: Schématické znázornění navrženého optimalizačního rámce. Šipkami je znázorněno propojení jednotlivých částí, nad šipkou je pak zdůrazněno, co jednotlivé části předávají v procesu dále.

# Bibliography

[1] P. Eichler. Mathematical modeling of fluid flow using lattice boltzmann method. *Dizertační práce*, 2023.

[2] T. Krüger, et al. *The Lattice Boltzmann Method*. Springer International Publishing, 2017.

[3] M. Geier, M. Schönherr, A. Pasquali a M. Krafczyk. „The cumulant lattice boltzmann equation in three dimensions: Theory and validation". *Computers and Mathematics with Applications*, 70(4):507–547, 2015.

[4] D. D'Humières. „Generalized lattice-boltzmann equations". In *Rarefied Gas Dynamics: Theory and Simulations*, pages 450–458. American Institute of Aeronautics and Astronautics, 1994.

[5] M. Geier, A. Greiner a J. G. Korvink. „Cascaded digital lattice boltzmann automata for high reynolds number flow". *Physical Review E*, 73(6), 2006.

[6] I. V. Karlin, A. Ferrante a H. C. Öttinger. „Perfect entropy functions of the lattice boltzmann method". *Europhysics Letters (EPL)*, 47(2):182–188, 1999.

[7] Tomáš Oberhuber, Jakub Klinkovský, and Radek Fučík. Tnl: Numerical library for modern parallel architectures. *Acta Polytechnica*, 61(SI):122–134, February 2021.

[8] Jakub Klinkovský, Tomáš Oberhuber, Radek Fučík, and Vítězslav Žabka. Configurable open-source data structure for distributed conforming unstructured homogeneous meshes with gpu support. *ACM Transactions on Mathematical Software*, 48(3):1–30, September 2022.

[9] J. Bureš. Matematické modelování proudění krve v cévách. *Bakalářská práce*, 2022.

[10] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, August 1963.

[11] C. G. BROYDEN. The convergence of a class of double-rank minimization algorithms. *IMA Journal of Applied Mathematics*, 6(3):222–231, 1970.

[12] Dimitri Bertsekas. *Nonlinear programming*. Athena Scientific, September 2016.

[13] David G Luenberger and Yinyu Ye. *Linear and nonlinear programming*. International series in operations research & management science. Springer, New York, NY, 3 edition, July 2008.

[14] Charles Audet and Warren Hare. *Derivative-free and blackbox optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, Switzerland, 1 edition, December 2017.

[15] Jeffrey Larson, Matt Menickelly, and Stefan M. Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, May 2019.

[16] Stéphane Alarie, Charles Audet, Aïmen E. Gheribi, Michael Kokkolaras, and Sébastien Le Digabel. Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, 9:100011, 2021.

[17] Oliver Kramer, David Echeverría Ciaurri, and Slawomir Koziel. Derivative-free optimization. In *Computational Optimization, Methods and Algorithms*, pages 61–83. Springer Berlin Heidelberg, 2011.

[18] George B. Dantzig. Origins of the simplex method, June 1990.

[19] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[20] Charles Audet and J. E. Dennis. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, January 2002.

[21] Charles Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, January 2006.

[22] Patrick Kofod Mogensen and Asbjørn Nilsen Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, 2018.

[23] C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes. Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 48(3):35:1–35:22, 2022.