Application of numerical optimization techniques for the interfaces reconstruction in two-phase flows

University of Zurich[UZH]

Bachelor thesis in Computational Science

Completed at the Department of Informatics

Candidate:

Gioele Luca Monopoli

Student ID: 17-923-293

Supervisors:

Dr. Barbara Re, Dr. Davide Scaramuzza,

Élise Delphine Le Mélédo

Submission date:

January 31, 2021

Acknowledgements

This work would not have been possible without my supervisors and my assistant.

I would like to thank Barbara Re for giving me the opportunity to undertake this thesis and deepen my knowledge into numerical optimization and fluid dynamics. She has given me confidence and this was extremely appreciated.

Then, I would like to thank Dr. Davide Scaramuzza for the will of being my supervisor from the Computer Science department for this work, as well as mentor.

Lastly, I would like to express my gratitude toward my assistant Élise Le Mélédo, who has followed me through the entire work, giving me assistance and being always available when needed, through long mails and meetings.

Abstract

Two-phase flow problems are widely present in physics and can be investigated in computer science through numerical optimization. In fluid dynamics, coexisting fluids are separated by dynamical interfaces, curves that move and deform as time evolves, according to the velocity field given at the interface by the state of interacting phases. This thesis aims to improve the spatial accuracy in the reconstruction of the interface separating fluids in two-phase flow numerical simulations, assuming that the two flows are governed by the same set of equations, i.e., Euler equations of fluid dynamic. The whole is accomplished through the application of constrained optimization to reconstruct the shape of these interfaces at a specific time, using a Bézier curve interpolation characterised by energy minimization. The fluids are surrounded by a given velocity field resulting in constraints imposed on the curve as energy costs. Thanks to the implementation of the given dataset we find ways to calculate these energy costs and to insert them in the optimization problem, which thanks to the exploitation of existing algorithms an energy minimised curve will result from.

Zusammenfassung

Zweiphasenströmungsprobleme sind in der Physik sehr verbreitet und können in der Informatik durch numerische Optimierung analysiert werden. In der Fluiddynamik werden koexistierende Fluide durch dynamische Grenzflächen getrennt, Kurven, die sich im Laufe der Zeit bewegen und verformen, entsprechend dem Geschwindigkeitsfeld, das an der Grenzfläche durch den Zustand der interagierenden Phasen gegeben ist. Diese Arbeit zielt auf die Verbesserung der räumlichen Genauigkeit bei der Rekonstruktion der Grenzfläche zwischen Flüssigkeiten in numerischen Simulationen von Zweiphasenströmungen, wobei davon ausgegangen wird, dass die beiden Strömungen durch den gleichen Satz von Gleichungen, d.h. die Euler-Gleichungen der Fluiddynamik, geregelt werden. Dies wird durch die Anwendung einer eingeschränkten Optimierung erreicht, um die Form dieser Grenzflächen zu einem bestimmten Zeitpunkt zu rekonstruieren, wobei eine Interpolation von Bézier-Kurven verwendet wird, die durch Energieminimierung gekennzeichnet ist. Die Fluide sind von einem vorgegebenen Geschwindigkeitsfeld umgeben, das als Energiekosten zu Einschränkungen der Kurve führt. Dank der Implementierung des gegebenen Datensatzes finden wir einen Weg, diese Energiekosten zu berechnen und in das Optimierungsproblem einzufügen, aus dem sich, dank der Ausnutzung bestehender Algorithmen, eine energieminimierte Kurve ergeben wird.

Contents

Glossary and notation **8**

About the problem 8

Mesh structure 8

Math symbols 9

1 Introduction **10**

1.1 Motivation 10

1.2 Problem definition 11

1.3 Thesis outline 14

2 Background and related work **15**

2.1 Background in curve optimization 15

2.1.1 Bézier curves 15

2.1.2 Constrained optimization 17

2.2 Related work 18

2.2.1 Interactive modelling of constrained curves 18

3 Interface reconstruction based on energy-minimization **20**

3.1 Static control approach 20

3.1.1 Domain of interest 21

3.1.2 Curve modelling 21

3.1.3 Control of the curve 21
3.1.4 Curve's energy and resulting cost 23
3.1.5 Optimization statement 23
3.2 Dataset implementation 23
3.2.1 Dataset naming 24
3.2.2 Workflow setup 25
3.2.3 Data extraction 26
3.3 Optimization algorithm 28
3.3.1 Approximations of energy costs 28
3.3.2 Optimization methods 29
3.3.3 Optimizer 32

4 Evaluation of the interface reconstruction 33

4.1 Results from mesh at interface 33
4.2 Evaluation of the energy-based constrained optimization 35
4.2.1 Exemplary effects of multiple constraints on curve optimization 35
4.2.2 Possible variations of proposed optimization approach . . 37
4.3 Encountered issues and adjustments 37
4.4 Restriction 38

5 Conclusions and future work 39

5.1 Conclusions 39
5.2 Future work 40

List of Figures

1 Domain Ω with the two phases or fluids, separated by the interface. 13

2 Bézier curve with 5 control points, thus degree 4. 16

3 Triangular function spanning from 0 to 1 over $I \in [l_1, l_2]$ 22

4 Given mesh plotted using the Python library *Matplotlib*. 25

5 Given mesh with interfacial elements colored in red, plotted using the Python library *Matplotlib*. An heuristic curve is drawn in yellow. 26

6 The area ϵ where the elements are considered. Here four red, three green and two blue elements are shown. 27

7 Coarse mesh from the given dataset plotted using the Python library *Matplotlib*. The lighter part is the range of ϵ. 28

8 Nelder-Mead method. Figure reproduced from [6]. 31

9 Visual representation of two random intervals of the curve with their velocity constraints. 34

10 Energy cost functional highlighting adaptation to velocity constraints, plotted using the Python library *Matplotlib*. The red dots are the control points. 35

11 B-spline curves restricted by constraints or operators and shaped in an energy minimising way. The operators are imposed on the curve over the intervals described by the white dots. Figure reproduced from [16]. 36

12 Graph highlighting the strength influence of the director operator of the left curve when highly increased. Figure reproduced from [16]. 36

Glossary and notation

About the problem

- *Velocity-constraints*: the constraints imposed on the curve, derived from the velocity field of the original problem, retrieved from the set of *Euler* equations.
- *Euler equations*: the equations used in fluid dynamics to prescribe the motion of a fluid, neglecting the viscous and thermal effects.
- *Equation of state*: equation used to relate the properties of the fluids, such as pressure, temperature, and volume, over the domain.
- *Interface*: the region occupied by each fluid has its own boundaries, and their intersection is the interface.
- *Level-Set (LS) values*: the distance of a given point from the interface. The values are positive in one fluid, and negative in the other fluid. They are supposed to be zero on the interface location.
- *Advection equation*: the motion of a scalar as it is advected by a known velocity field. The equation is needed to retrieve the properties vector from the *Euler* equations.

Mesh structure

- *Elements*: geometrical entities, e.g., triangles, whose collection forms the mesh.
- *Degrees of freedom*: virtual points located within elements (exactly six per element) that characterize the discretization of the velocity field.

Math symbols

- $\phi(x)$: Level Set field.
- Ω: Study domain. The two fluids subdomains are described by Ω_1 and Ω_2.
- x_i: generic point belonging to an element $\in \Omega$.
- $\phi(x_i)$: Level Set value at the point x_i.
- v_i: average value of velocity at the point x_i.
- $x(s)$: coordinate of the point (x, y) parametrised by the curvilinear coordinate s along the Bézier curve.
- $x'(s)$: first order derivative of Bézier curve at the point parametrised by s.
- w : weight function; local influence of a velocity v_i on the curve.
- α_i: weight representing the global influence of a velocity v_i on the curve.
- α_{cvx}: weights of the linear combination between the two energies of the curve, E_{bend} and $E_{stretch}$.
- ℓ: Bézier curve.
- ζ: estimated length of curve ℓ.
- $[l_1, l_2]$: subinterval of curve ℓ.
- n: degree of curve ℓ.
- Γ: velocity field domain.
- Υ_c: control space of the Bézier curve

Chapter 1

Introduction

1.1 Motivation

Numerical methods are important tools to model and investigate phenomena present in all ranges of sciences. Their fundamental goal is to analyse and solve the mathematical problem describing the phenomenon of interest by exploiting numerical approximations and algorithms. A particular important role is played by numerical optimization, which aims to identify the "best available solution", according to some given constraints. Numerical optimization is broadly applied in various branches of computer science such as image processing and medical image analysis, and it is one of the core techniques of machine learning, where a particular goal is to maximize the likelihood of some training data under a prescribed model by finding the optimal parameters, or to minimize the divergence between the model and the data [14]. Furthermore, optimizations include many subjects passing through linear programming. Today, having a rich set of mathematical skills in numerical optimization and a broad knowledge of programming skills is a must in computation research as well as highly required in industry.

Numerical optimization also lies in the optimization of curves' shape by minimizing their energy. When on a curve a set of constraints are imposed to attract and drag the curve, energy minimization is the effect to adjust against these constraints. Representation of curves that fit the best a set of points via interpolation, meaning piecewise polynomial (parametric) curves, are widely used in computer-aided design and computer graphics, as especially the Bézier curve.

In this work, optimization of curves is applied to develop a new numerical tool to be integrated into an existing software for the simulation of two - phase

flows, meaning an heterogeneous mixture of two immiscible fluids, e.g oil in water, which are separated by dynamical interfaces. The goal is to reconstruct the shape of these interfaces by looking for the curvature that leads to energy minimization. This is accomplished by minimizing a cost functional that will be given, taking as inputs a vectorial displacement field and initial values of a Level Set (generally speaking, discrete data points). The output will be a spline in the previously given representation which has the least energy and is the smoothest[1] possible. A possible strategy to address this problem, which is also the one this thesis is based upon, is an interactive approach based on variational curves, that are curves that minimize some energy functional under certain interpolation constraints, with the possibility to use deformation operators with built-in energy terms [16]. More precisely, the idea of the paper from Wesselink and Veltkamp is to let the user interactively modify the range of influence, on a curve, of a certain constraint and other properties of the so-called operators. In this work, the focus is on prescribed velocity constraint but the considered paper and possible future work include additional operators, as point and curve attractors, to interactively modify the shape of the curve. This will be further discussed in Chapter 2.

1.2 Problem definition

Two-phase flow problems are encountered in a variety of physical and industrial problems ranging from biomedicine to pipeline design. Those problems involve two different natures of fluid, specifically two different physical phases of the same component as ice motion in water or two different coexisting components of the same physical phase as oil in water, which are separated by dynamical interfaces. If we consider a two-dimensional simulation (2D), these interfaces are essentially dynamic curves that move and deform as the time evolves. This means that each of the phases is subject to its own dynamic and interacts with the other phase through the interface. This interface evolves in time according to the velocity field resulting from the properties of the interacting flows. The set of the governing equations is

$$\begin{cases} \frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho E\end{pmatrix} + \nabla\cdot\begin{pmatrix}\rho u\\ \rho u^2+p\\ u(E+p)\end{pmatrix} = 0 \\ \\ \frac{\partial}{\partial t}(\rho\phi) + \nabla\cdot(\rho\vec{u}\phi) = 0, \end{cases} \tag{1}$$

where the first system expresses the *Euler* equations, which describe the dynamics of the fluid, while the last equation describes the evolution of the interface. The

[1] A smooth function is continuous and differentiable in every point up to a certain order n, $f(x) \in C^n$, higher is n smoother is the function.

state vector describes the fluid over the domain, and reads

$$\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \tag{2}$$

where ρ describes the density, u the velocity in the x axis, v the velocity in the y axis, and E the energy, which are all space-time-dependent.

For each phase, the equation of state (EOS) links the energy E with the pressure p and impacts the fluid dynamics. Thus, resolving the interface is a key point of any numerical scheme, granting that the right equation of state and the right fluid properties are instilled to the equation (2) at the degrees of freedom[2] associated to the corresponding fluid.

The interface is represented by a Level Set ϕ. The Level Set method is a popular numerical approach which aim is to capture interfaces as well as to resolve surface tension force, and has become one of the most popular interface tracking methods for simulating multiphase flows. This method represents the interface implicitly through the zero level set of a signed distance function $\phi(x)$ [17], that is

$$\Sigma = \{x | \phi(x,t) = 0\}. \tag{3}$$

In the two-phase flow field, the domain Ω is separated by the interface Σ into two parts that are respectively occupied by fluid 1 (Ω_1) and fluid 2 (Ω_2). The Level Set $\phi(x)$ gives the distance of a given point x from the interface Σ. It has the property to have positive values, i.e.

$$\phi > 0$$

at points x inside the region Ω_1 delimited by the curve (meaning in one fluid, in this case Phase 1) and negative values, i.e.

$$\phi < 0$$

at points x outside Ω_1 (meaning in the other fluid). As shown in equation (3), for x lying on the interface, we have

$$\phi = 0,$$

which describes the position the interface itself lies, which is, as aforementioned, the intersection of the regions occupied by the two fluids [12] (see Figure 1).

For optimization purposes, we consider the surface energy of the fluid boundary. Indeed, from a physical point of view, the surface energy is minimised under the vector field constraint: this is one of the properties forcing a big cube

[2]The location (point-values) at which the velocity field and other properties are retrieved from the state of a physical system

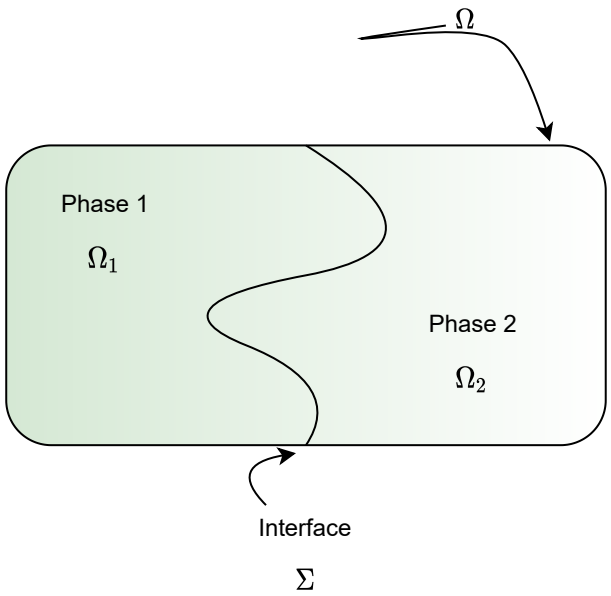

Figure 1: Domain Ω with the two phases or fluids, separated by the interface.

of water to tend to a ball in its melting process, besides fighting heat transfer. As seen from equation (2), since ϕ evolves over time and the distance function is updated continuously to track the interfacial movement, the problem is tackled by performing regular advection of the Level Set

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (u\phi) \tag{4}$$

by the underlying velocity field $u \in \mathbb{R}^2$ which is evolved according the *Euler* equations. The numerical solution of the governing equations can be obtained by any Finite Volume or variational formulation based method. The optimization technique presented in this thesis focuses on the reconstruction of the interface, given the fluid state at the neighboring degrees of freedom. Note that the dataset and the routine solving eq. (2) and eq. (4) are already given, so they are not entering the scope of this work, but directly used instead. The minimization yields an optimal curve, resembling the interface, which minimises the surface energy of the considered phase's boundary under the constraint of the discrete values of the Level Set, and improves the Level Set re-distancing process. Thanks to the application of iterative optimization algorithms, the curve will result as smooth as possible.

1.3 Thesis outline

The remainder of this work is structured in the following way. In Chapter 2, an introduction into Bézier curves is given, following with an opening insight into optimization and the highlight of the interactive approach solution introduced in the Introduction. Then, in Chapter 3, a detailed description of the interface reconstruction approach is given and the core implementation is highlighted and described. Finally, in Chapter 4 an evaluation of the results, issues and possible adjustments is performed, concluding with Chapter 5 where possible future work is discussed.

Chapter 2

Background and related work

In this chapter, the curve representation through Bézier interpolation, the constrained optimization technique and an interactive approach for user design will be presented to provide foundation of understanding for the next chapters.

2.1 Background in curve optimization

2.1.1 Bézier curves

To represent interfaces in two-phase flow problems, interpolation techniques can be used. This requires to find an approximation of the original curve, which is described through a set of points, by an interpolation function $f(x)$, which, in order to be smooth and thus have at least a continuous derivative, requires to be of a degree two or more. A type of piecewise polynomial interpolation of degree $n \geq 2$ is the Bézier curve. This type of curve has been introduced in [2] and is widely used in computer graphics and in computer-aided design system, ranging from computer font-rendering technologies like Adobe Illustrator and Postscript, to designing curves and surfaces in the automobile sector.

A Bézier curve (see Figure 2) is a type of interpolating curve, also called interpolant, quadratic or of higher order. It is defined as a parametrized curve over the interval $s \in [0, 1]$, precisely forming a path in the xy-plane traced out by the point $(s, x(s))$, also known as *coordinate functions*, as the parameter s ranges over an interval I. The Bézier curve uses as basis the *Bernstein polynomials* (also called *Bézier basis functions*). To formulate, a Bézier curve of degree n is

represented by

$$r(s) = \sum_{i=0}^{n} p_i B_{n,i}(s), \qquad 0 \leq s \leq 1, \tag{5}$$

where p_i are the $n+1$ *control points* or *Bézier points* together with the basis functions $B_{i,n}$, which are *Bernstein polynomials* defined as

$$B_{n,i}(s) = \frac{n!}{i!(n-i)!} s^i (1-s)^{n-i}. \tag{6}$$

The *end point interpolation property* states that the curve starts at the control points $r(0)$ and ends at $r(n)$.

Consider the example of a quadratic Bézier curve of degree 2 with 3 control points. The curve function $r(s)$ is defined as follow:

$$r(s) = (1-s)^2 p_0 + 2(1-s)(s)p_1 + s^2 p_2 \tag{7}$$

Speaking about derivatives, one of the most interesting observations about Bézier curves is that their derivatives are, in fact, also Bézier curves. The first derivative of the Bézier curve $r'(s)$ is called the *hodograph*. It is another Bézier curve whose degree is lower than the origin curve by one $(n-1)$ and has new control points $p'_i = n\Delta p_i$, $i = 0, \ldots n-1$ and $p'_n = p_n$. The second derivative of Bézier $r''(s)$ is similar to the first one, with one degree lower $(n-2)$ and new control points $p''_i = n-1\Delta p'_i$, $i = 0, \ldots n-2$ and $p''_n = p'_n$.

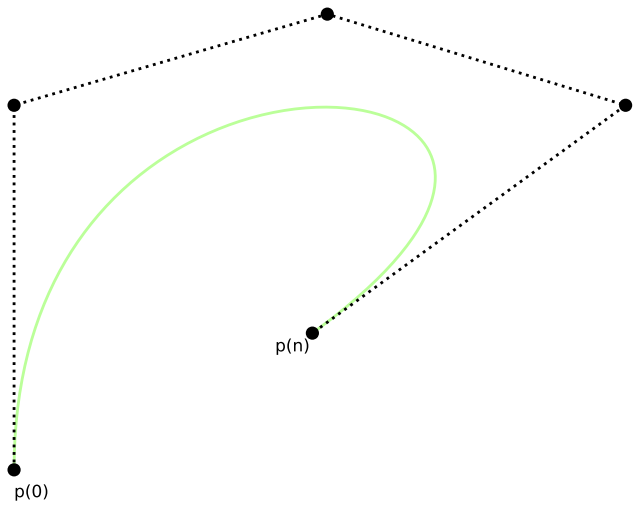

Figure 2: Bézier curve with 5 control points, thus degree 4.

2.1.2 Constrained optimization

Constrained optimization refers to a process of selecting the best element within an admission set which optimizes (minimizes or maximixes) some sorts of functionals, under the restriction of so-called constraints (the opposite would be unconstrained optimization).

Consider the minimization problem

$$\begin{aligned} \min_{x} \quad & f(x) && (8)\\ \text{subject to constraints} \quad & h(x_i) \geq 0, && (9)\\ & h(x_i) \leq 0 \text{ or} && (10)\\ & h(x_i) = 0. && (11) \end{aligned}$$

where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is the cost functional and the *objective function* to minimize, x the minimizer (also noted as the *optimal solution*), and $h(x) : \mathbb{R}^n \to \mathbb{R}^m$ the constraint function, which can be an *inequality* constraint ((9), (10)) or an *equality* one (11). The purpose of this problem is to find a solution which satisfies the aforementioned constraints and is optimal. In our case, the focus will be on minimizing a cost functional under equality constraint only.

David G. Luenberger and Yinyu Ye published a book [8] covering the central concepts of practical optimization techniques and highlighting the major and popular linear and non-linear programming problems, as well as the most effective numerical algorithms used to solve this kind of optimization problems (constrained and unconstrained). The approaches range from constrained minimization via descent methods to penalty / barrier methods and methods using Lagrange multipliers as the augmented Lagrangian. This last one, in particular, will be described and used in the Approach Section of this thesis.

An interesting application of constrained optimization is to machine learning. The majority of the machine learning algorithms, from supervised learning to reinforcement learning, can in fact be formulated as optimization problems in which the goal is to find the optimized objective function. Through iterative optimization of the objective function, these algorithms learn a function which best approximates the mapping from the input to the output, precisely minimizing the loss function of training samples. Made simple, considering the equation

$$y = f(x), \tag{12}$$

the goal is to get the best predicted output y from new input data x. As of today, a series of effective optimization methods have been developed which have improved the performance and efficiency of the aforementioned machine learning methods. These are mainly separated into three categories: first-order optimization methods as the stochastic gradient methods, high-order optimization methods as the Newton's method, and heuristic derivative-free optimization

methods as the Nelder-Mead method [15]. This last will be exposed in detail later. From this, we perceive an high contribution from numerical optimization method to machine learning techniques.

2.2 Related work

2.2.1 Interactive modelling of constrained curves

This section is based on the work of Wesselink and Veltkamp [16], which presents an interactive approach for user design in modelling a variational curve, that is a curve which minimizes a given functional representing the energy of the curve. The interaction in the design process is made of constraints or so-called operators, which in order to increase flexibility, deform the curve locally by minimizing a given energy functional. The paper highlights the consideration of two energies: an internal, which depends on property of the curve itself, and an external one, which depends on operators or constraints imposed on the curve, so "externally".

Firstly, the internal energy is defined as a combination of a bend and a stretch energy, with the latter one restricting the length of the total curve. They are defined respectively as

$$E_{bend}(x) = \int \kappa^2(s) \, \|x'(s)\| \, ds, \tag{13}$$

with κ the curvature of the curve x and

$$E_{stretch}(x) = \int \|x'(s)\| \, ds. \tag{14}$$

The two internal energies are incorporated into a convex combination, which allows the user to determine the relative weight of the two described energies via a parameter α_{cvx}.

Secondly, the design operators, which are influencing terms for the model to curve as wanted, can be of three types: a *director* n, pushing the tangent along a segment of the curve in a defined direction via a norm vector on a parameter interval $[l_1, l_2]$, formulated as

$$E_{dir}(x) = \int_{l_1}^{l_2} (x'(s) \cdot n)^2 ds, \tag{15}$$

a *point* attractor, pulling the curve towards a point, formulated as

$$E_{pat}(x) = \|x - p\|^2, \tag{16}$$

and a *curve* attractor, pulling the curve towards another curve, formulated as

$$E_{cat}(x) = \int_{l_1}^{l_2} \|x(s) - \ell(s)\|^2 \, ds. \tag{17}$$

Each operator has its own influence depending on the weight factor imposed, using various ways, for example by multiplying it with a constant factor or using weight function as the *triangular / hat function* or a *Gaussian*.

It is worth noting that the original paper makes use of B-spline curves as main spline where the operators are applied, and it minimises the whole quadratic energy functional by converting the energy functional and the constraints in vector and matrix form and then solving the system using LU decomposition. Closely related, in this work Bézier curve are considered and only the director operator will be studied and applied.

Chapter 3

Interface reconstruction based on energy-minimization

The description of the approach followed in this work for the interface shape reconstruction is subdivided into three sections, starting from approach definition, an overall subsection into the curve and its controlling possibilities, the optimization statement and then guiding through the data set manipulation and the constrained optimization algorithm implementation.

3.1 Static control approach

The approach is called static to highlight that the optimization occurs at a fixed time, i.e., fluid boundaries are not moving, thus the Level Set ϕ is not evolving over time according to the given velocity field. It consists in a direct representation and interpolation of the discrete field on the curve points. In other words, the solution of the governing equations is already computed at the time when interface optimization is performed and the two-phase flow field is known at a discrete level, namely the fluid properties (e.g., velocity) are given at each degree of freedom. Then, based on this discrete field, on the velocity information on this field, and on the level set constraints of each of the phases, an energy minimizing curve representing the optimal curve describing the interface shape is to be found.

3.1.1 Domain of interest

The numerical domain, over which the discrete field is considered, is a two-dimensional space, represented as a triangular mesh with set of elements connected by their common edges. Each of the discrete points is located on a Cartesian plane and thus has (x, y) values. Clearly, elements are constructed by three of these points, as the shape of a triangle. The degrees of freedom are lying over these elements and thus also the velocity field retrieved at these points has (x, y) location for each of its values.

3.1.2 Curve modelling

Many spline representation could have been used instead of the Bézier one, as the example of [16], which make use of B-spline instead. Indeed, there are various interpolation curves which were considered but at the end not chosen in this work, such as Catmull-Roll spline, which is a type of *Hermite* spline or *cardinal* spline with the characteristic that the specified curve will pass through all of the control points, and two additional control points are required on either end of the curve.

The choice of Bézier is motivated by flexibility and better applicability to this precise problem. In the next section and also in the implementation, the formula considered for the curve $x(s)$ is the following:

$$x(s) = \sum_{i=0}^{n} \underbrace{\binom{n}{i}(1-s)^{n-i}s^{i}}_{B(n,i)} \underbrace{\begin{pmatrix} p_{ix} \\ p_{iy} \end{pmatrix}}_{\text{control points } (x,y)} \tag{18}$$

where p_{ix} and p_{iy} are the (x, y) values of control point i. We denote the space of control, e.g. control points shaping the curve, as Υ_c:

$$\Upsilon_c = \left\{ p_{ix}, p_{iy} \right\}_{i=0}^{n} \in \mathbb{R}^{2(n+1)}. \tag{19}$$

3.1.3 Control of the curve

As highlighted in Subsection 2.2.1, many ways, or *operators*, can be exploited to control the curve. In this work two main ways of controlling the curve are imposed, as *constraints*: i.e., (i) the Level Set values, and (ii) the velocity field.

The constraint (i), seen as the distances to be preserved between the point where the Level Set value $|\phi(x_i)|$ is computed and the interface curve itself (i.e. *Euclidean distance*), can be written as

$$d(x_i, \ell) = |\phi(x_i)| \quad \forall x_i \in \mathbb{R}^2. \tag{20}$$

This distance has to be respected by the energy minimising curve.

The velocity constraint (ii) is not defined as a constraint for the optimization problem, but more of an energy cost added to the total energy functional. For the curve inflection to be consistent with the surrounding velocity field, this constraint is seen as a directional operator pulling the curve. The magnitude of the velocity is used to weight, using any existing weight function, the "attraction" of this operator, i.e. its local influence on the curve. Furthermore, a parameter α is chosen for describing how much each segment will affect the shape of the curve, i.e. its global influence on the curve. Therefore, its energy cost is defined as

$$\alpha \int_{l_1}^{l_2} w(s)(x'(s) \cdot v)^2 ds, \tag{21}$$

where l_1 and l_2 are the start and end points of a subinterval of the curve considered onto which the velocity constraint is applied. Furthermore, the actual weight function $w(x)$ used in this work is an *hat* or *triangular* function, i.e with the graph of a triangle with its value spanning from 0 to 1, where 1 is the top of the "triangle" (see Figure 3). The maximum value, 1, is considered to be at the midpoint between l_1 and l_2, expressing the approximated position where the velocity vector lies. On the other hand, parameter α is calculated using the

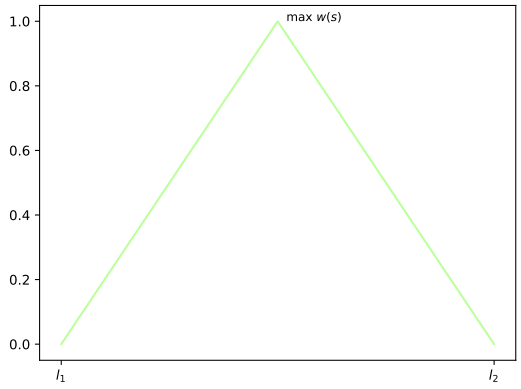

Figure 3: Triangular function spanning from 0 to 1 over $I \in [l_1, l_2]$

same weight function but depending on the velocity distance to the curve itself. An extensive description will be provided in the next sections.

3.1.4 Curve's energy and resulting cost

From Subsection 2.2.1 and eq. (21) we can now derive the curve total energy cost for the constrained optimization problem. The cost function $c(x)$ is quadratic (second-degree polynomial) and convex[1], therefore the existence of a minimum is ensured. The optimized curve will minimize the curve's blend energy to which the aforementioned external cost driven constraint are added, i.e. the velocity-driven director constraints. Thanks to eq. (13), (14), (21) and by defining the curve being of length ζ over $s \mapsto x(s)$, we can define the entire energy function (the cost) as

$$\begin{aligned} c(x) = \quad & \alpha_{cvx} \int_0^{\zeta} \kappa^2(s) \, \|x'(s)\| \, ds \quad + \quad (1 - \alpha_{cvx}) \int_0^{\zeta} \|x'(s)\| \, ds \\ & + \quad \sum_{i \in \Gamma} \Big[\alpha_i \int_{l_{i1}}^{l_{i2}} w_i(s) (x'(s) \cdot v_i)^2 ds \Big], \end{aligned} \tag{22}$$

where Γ is the velocity field domain, α_{cvx} is the parameter defining the convex combination between the two internal energies (bend and stretch), and i represents one of the intervals the curve is subdivided according to the external velocity constraint costs.

3.1.5 Optimization statement

The optimization problem can be formulated as follow: Find the curve $\ell : [0, 1] \to \mathbb{R}^2, s \to x(s)$, such that it realizes

$$\begin{aligned} & \min_x \quad c(x) \quad (22) \\ & \text{subject to constraints} \quad (20) \\ & \text{over the control space} \quad \Upsilon_c \quad (19) \\ & \text{and the parametrisation} \quad x(s) \quad (18) \end{aligned}$$

3.2 Dataset implementation

This work is built on top of an existing Python library which solves multi-phase problems according to the Level Set method. This library includes the relevant data of the fluids, the governing equations, and the entire discrete field represented over a triangular mesh structure. For this work, fine and coarse meshes have been used, where mesh properties, Cartesian points, elements details, neighbouring information and degrees of freedom are provided. Moreover, the

[1] $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for all x, y and all $\lambda \in [0, 1]$ we have $f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$.

library contains the solver and reconstruction routines that are needed in order to retrieve the Level Set and velocity values. From now on, consider LS values as being the Level Set values. An introduction to the dataset will follow.

3.2.1 Dataset naming

The main dataset is constructed from the following data structures (the attribute name is given in bold, while the Python object type in italics):

- The elements of the triangular mesh, i.e. each *triangle*, easily accessible by **Mesh.InnerElements** *(2D Numpy Array)* and is the main starting point for the next data structures,
- The diameter and other element properties, retrieved by **Mesh.InnerElementsDiameter**, **Mesh.InnerElementsVolume** *(2D Numpy Array)*,
- the Cartesian points location, retrieved via **Mesh.CartesianPoints** *(2D Numpy Array)*,
- the neighbouring information of each element, retrieved via **Mesh.Neighborhoods** *(2D Numpy Array)*,
- the location of the degrees of freedom, provided from the mesh by **Mesh.DofsXY** *(2D Numpy Array)*,
- the coordinate of the center of mass of an element, i.e its center, retrieved by **Mesh.DualCenters** *(2D Numpy Array)*,
- the LS values $\phi(x_i)$ at every vertex of an element in the mesh, retrieved by inputting to the reconstruction routine the Cartesian position (recalling that is 2D, so $[x, y]$ $\forall x_i$) of each of the vertices and the global index of the element itself, which will allow to retrieve the corresponding LS value allocated to a particular vertex on the Cartesian plane,
- the average of the velocity values v_i, retrieved by inputting to the reconstruction routine the degrees of freedom at each vertex and averaging the velocities to a single vector. This will output a velocity value (v_x, v_y) which lies on (or near) a certain vertex.

Reconstruction routine

As described in Section 1.2, the fluid dynamic is ruled by a joint evolution of the Euler Equations and the Level Set equation. The Euler equations evolve

the fluid state, from which the fluid velocity field is inferred. Simultaneously, the Level Set equation evolves the interface location according to the velocity field and furnishes implicitly the spatial location of the interface.

At the discrete level, this dynamic is retrieved on the mesh by a Continuous Galerkin type scheme. The numerical solutions for both the state and the Level Set are given on each element as values of so-called degrees of freedom, here point-values. Hence, the solution is only available at some precise locations on each mesh cell. The continuous velocity field is then retrieved by reconstructing the solution within each cell from all the degrees of freedom values accordingly to the variational discretization (i.e. basis function shapes). This process is already implemented in the reconstruction routine.

3.2.2 Workflow setup

In this subsection we present the setup of the data needed later for the optimization algorithm. At first we introduce the mesh for better understanding.

Introduction to the mesh

As mentioned, the given mesh consist in triangles, called elements, which are connected with each other by edges or vertices (see Figure 4). Since the

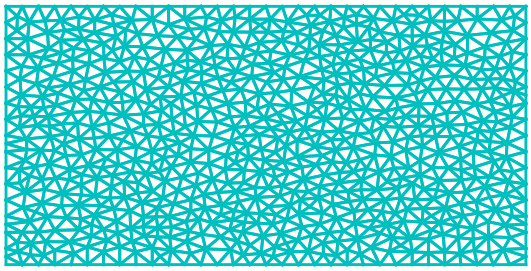

Figure 4: Given mesh plotted using the Python library *Matplotlib*.

mesh covers both fluids, the interface (where the curve passes through) is located in a section / part of the mesh itself, as you can see by the red elements from Figure 5. Each part of the mesh has elements which we can examine or drop, thus only an area of the mesh will be taken into consideration, respectively near the interface. In Figure 5, an heuristic interfacial curve is also drawn.

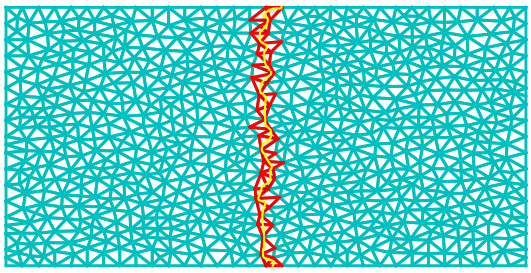

Figure 5: Given mesh with interfacial elements colored in red, plotted using the Python library *Matplotlib*. An heuristic curve is drawn in yellow.

3.2.3 Data extraction

Step 1. Getting the initial data: defining $\phi(x_i)$, v_i, α_i, ζ. Thanks to the reconstruction routine we can retrieve the LS values $\phi(x_i)$ at each element i. As we are interested in the interface where the two fluids intersect, that is where $\phi(x_i) = 0$, see Section 1.2, we consider only the elements nearby this area (the neighbourhood of the interface), limiting it to a max distance ϵ (see Figure 6). Consider these elements within ϵ as *blue elements* (which include all the colored ones). Consider the *red elements* being where the interface passes through, defined as elements where we have at least one vertex i with $\phi(x_i) \leq 0$ and a vertex j with $\phi(x_j) \geq 0$. Consider the *green elements* being the neighbouring elements of the *red* ones and part of blue elements. Because of this, we define ϵ as being of value equal to the maximum diameter of the red elements plus the maximum diameter of the green elements. This gives us an abundant element space and it ensures that each interface neighbouring element is considered. Moreover, we estimate the curve length ζ of being the sum of all the diameters of the red elements, indicating an upper bound estimate. This ensures that our optimized curve length will not exceed ζ and helps defining the intervals of the curve in the next step. Figure 6 and 7 should provide a better visualisation of the discussed procedure.

Now that we have the elements with their LS values, thanks to the degrees of freedom at these elements, we retrieve the corresponding velocities. Since, for simplicity, we are not considering multiple velocities per element, we average them to obtain a single velocity constraint v_i (per element).

By having v_i, we can now find out α_i, recalling that it is describing the global influence of eq. (21) (its importance) on the total curve. It is set via a weight function depending on how far the element i, to which the velocity is attached, is from the curve. This is described as the distance between the mean of the LS values of the element (since there are three LS values per element, i.e.

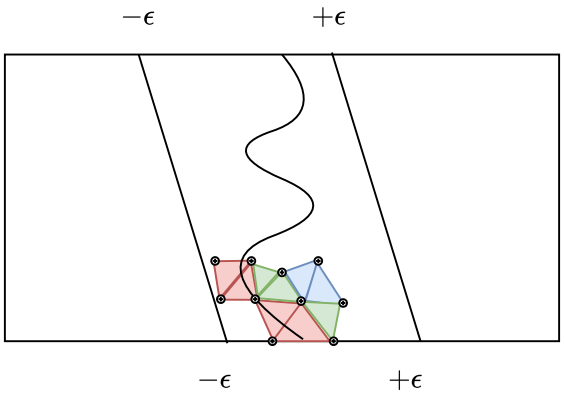

Figure 6: The area ϵ where the elements are considered. Here four red, three green and two blue elements are shown.

three vertices) and the zero LS value (where the curve lies).

Step 2. Defining l_1 and l_2 for each element. The above mentioned velocities v_i obtained are going to be imposed on intervals of the curve. l_1 and l_2 are the starting and endpoint of a interval on which a velocity is applied. Since we have one velocity per element, we have to define this interval for each element. And since the curve has to pass through the red elements (recalling that it is where the interface lies, thus the interfacial elements), they have to be connected with each other by either edges, i.e. they share an edge together, or vertices, i.e. a vertex is in common between two neighbouring red elements. The whole length of ζ, containing intervals for each red element, is defined by firstly taking the starting point of the curve (the red element on the boundary of the domain), checking its neighbouring elements (by checking its edges or vertices) to find the corresponding red element, and finally setting its $[l_1, l_2]$ by calculating its diameter. This procedure is repeated till all the neighbouring red elements are resolved (the last red element will correspond to the other boundary where the curve ends). A list with the $[l_{i1}, l_{i2}]$ of all red elements i will be obtained.

Since all the elements we consider are the blue ones (the ones within ϵ, including red and green), we need also to find the intervals for the blue - red elements (which includes the green elements plus the remaining ones within ϵ). We do this by setting their interval equal to the nearest red element interval (found by taking the minimum euclidean distance between a red element center of mass and the considered element center position).

Step 3. Setting w_i. As the weight function w in eq. (21) depends on the interval, we define it ranging from l_1 to l_2 and having its max value at the

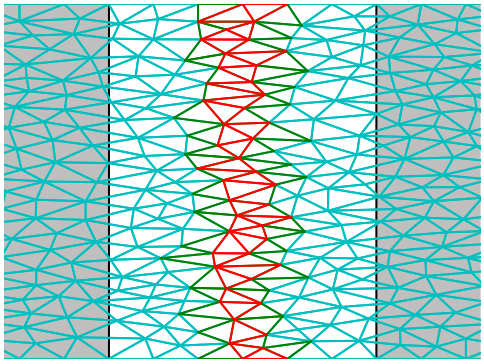

Figure 7: Coarse mesh from the given dataset plotted using the Python library *Matplotlib*. The lighter part is the range of ϵ.

midpoint $\frac{l_1+l_2}{2}$, which is the approximated position where our velocity lies.

Step 4. Determining order of Bézier Curve n. Number of control points and thus dimension of control space have to be defined in order to determine the degree n. As we have n blue elements with one velocity attached to them, we define the number of control points equal the number of velocities or constraints, i.e. number of elements. Then n is equal to it minus 1.

3.3 Optimization algorithm

To start with, the first part of the energy cost is formulated, then quadrature to integral of the energy cost is explained and then the introduction to the optimizer follows.

3.3.1 Approximations of energy costs

Step 1. Determine order derivatives of Bézier curve. Since E_{bend} can be approximated as

$$E_{bend}(x) = \int \|x''(s)\|^2 \, ds \tag{23}$$

and $E_{stretch}$ as

$$E_{stretch}(x) = \int \|x'(s)\|^2 \, ds, \tag{24}$$

we need to express first and second derivative of Bézier curves. We thus recall from Subsection 2.1.1 that the first order derivative $x'(x)$ can be formulated as

$$x'(s) = \sum_{i=0}^{n-1} B_{n-1,i}(s)\{n(p_{i+1} - p_i)\} \tag{25}$$

and the second derivative $x''(s)$ as

$$x''(s) = \sum_{i=0}^{n-2} B_{n-2,i}(s)\{n - 1\{n(p_{i+1} - p_i)\}\}. \tag{26}$$

Step 2. Approximation of integral via quadrature. Since the integrals in our energy function have large computation costs and it is impractical to compute the integral analytically, we use numerical integration and approximate it by performing a *Chebyshev–Gauss* quadrature with ten nodes. This consists in a quadrature over the interval $[-1, 1]$ with weight function $W(x) = (1 - x^2)^{-1/2}$, evaluated at ten points and considered with ten associated weights.

Since our considered interval for each constraint in eq. (21) is not $[-1, 1]$ but $[l_1, l_2]$ instead, and for E_{bend}, $E_{stretch}$ is $[0, 1]$, we have to parametrize the points to fit into these intervals (by proportions). Consider the ith parametrized point p_i as pz_i $\forall_i\{0, \ldots, p\}$, and the ith weight as wc_i. We can then formulate the quadrature as

$$\int_a^b f(x)dx \approx \sum_{i=0}^{p} wc_i f(pz_i) \tag{27}$$

3.3.2 Optimization methods

First, for completeness, we introduce the *Augmented Lagrangian* method and the *Nelder–Mead* method, following the optimization algorithm presentation.

Augmented Lagrangian Method

This method is similar to the penalty method, widely used in linear programming [8], which replaces the original constrained optimization problem by a sequence of unconstrained optimization sub-problems. Its idea is to eliminate some constraints and add to the objective function a penalty term which becomes costly for infeasible points, adding up as a multiple of the square of the violation of each constraint. Moreover, a parameter μ is to be chosen to impose the severity of the penalty. Since this problem is of unconstrained form now, an higher μ would mean that the resulting problem approximates in a precise way the original constrained problem, yielding accuracy [1]. In spite of strong

convergence properties of the quadratic penalty method, ill-conditioning is the main problem associated to them, expected to cause numerical problems when solving the linear equations to calculate the Newton step or another numerical method for finding minimums.

A modification of the penalty method is thus the *method of multipliers* or *augmented Lagrangian method* [13], in which explicit Lagrange multiplier guesstimates λ_i are used to reduce the possibility of ill-conditioning which is, as aforementioned, inherent in the quadratic penalty function. Considering our equality-constrained problem, the objective function is thus formulated as [11]

$$\mathcal{L}_A(x, \lambda; \mu) = f(x) - \sum_{i \in \varepsilon} \lambda_i h_i(x) + \frac{\mu}{2} \sum_{i \in \varepsilon} h_i^2(x), \tag{28}$$

where the length of ε depends on the number of constraints. That is for each constraint i, λ_i and the constraint function $h_i(x)$ are considered. Furthermore, constraint violations are penalized by squaring the infeasibilities and scaling them by $\frac{\mu}{2}$. Since in this iterative algorithm we have to minimize the objective function, the algorithm stops when

$$\nabla_x \mathcal{L}_A(x_k, \lambda^k; \mu_k) \approx 0 \tag{29}$$

is satisfied up to a tolerance τ. At every iteration k, x_k, i.e. the approximate minimizer of eq. (28), λ^k and μ_k are updated accordingly.

It is worth nothing that ill-conditioning is reduced by using for each k a starting point x_k for the algorithm which is close to a minimizing point of $\mathcal{L}_A(x_k, \lambda^k; \mu_k)$, normally the last point x_{k-1} of the previous iteration. In order for x_{k-1} to be near a minimizing point, μ_k has to be close to μ_{k-1}, thus the rate of increase of μ_k has to be relatively small. If μ_k is increased at fast rate, then convergence of the algorithm would be faster, but at the expenses of ill-conditioning. Normally, a sequence $\mu_{k+1} = \beta\mu_k$ with $\beta \in [4, 10]$ works well [1]. We can obtain a good estimate of x^* by minimizing $\mathcal{L}_A(x, \lambda; \mu)$ even when μ is not particularly large, provided that λ is a reasonably good estimate of λ^*.

Since the execution of the algorithm requires to find a minimum at each iteration, an additional numerical method has to be used to perform the minimization with respect to x, which is introduced below.

Nelder–Mead Method

The gradient of a function is not easily computed and is included as a main step in most of the optimization algorithms, i.e. with the search directions defined by the gradient of the function, e.g gradient descent and the conjugate gradient method. In this work we referred to a numerical method which is derivative-free.

The Nelder-Mead method [10] is a commonly known numerical method and is the most widely used *direct search* method for solving unconstrained optimization problems, minimizing a function f using only function values, without any derivative information. The main idea of the algorithm is that a simplex, i.e. a geometric figure as a triangle or a line, starts adapting itself to the local landscape, elongating, changing direction, and contracting in the neighbourhood of a minimum, in order to terminate with a vertex which is in fact the minimum. More precisely, it considers the simplex in n dimensions that is the convex hull of $n+1$ vertices, denoted as $x_1, x_2, \ldots, x_{n+1}$. The method iteratively generates a sequence of simplices to approximate an optimal point of $f(x)$. At each iteration, the vertices $\{x_j\}_{j=1}^{n+1}$ of the simplex are ordered according to the objective function values

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1}). \tag{30}$$

Consider x_1 as the best vertex, and x_{n+1} as the worst one [5]. What the algorithm can perform at each iteration is one of the following four operations: *reflection*, *expansion*, *contraction*, and *shrink*, each being associated with a scalar parameter: α (reflection), β (expansion), γ (contraction), and δ (shrink). The execution of these operations depends on whether they contribute the point to be nearer the minimum.

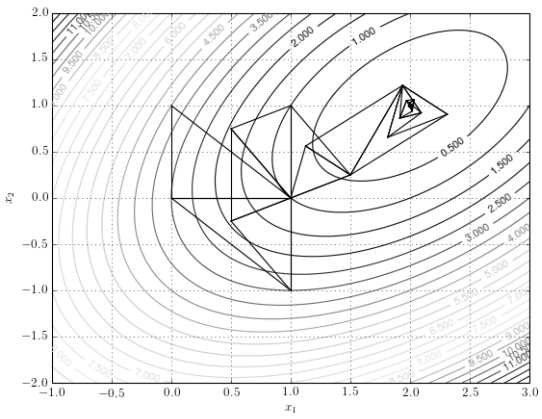

Figure 8: Nelder-Mead method. Figure reproduced from [6].

Even if the advantage of not calculating the gradient is present, there are some drawbacks. The method may fail to converge to a critical point of f, e.g successfully converge to a non-critical point of f, as it was highlighted back in the 90's by McKinnon [9]. However, to date multiple alternative methods have been explored to avoid this problem, as [7] which omits the expansion step to take place, or [3],[4] using convergence variants.

3.3.3 Optimizer

Having introduced the two main methods which construct the optimization algorithm, we can now define the final algorithm in pseudocode. Since what we want to minimize is the energy of the curve, this means to "minimize" its control points p_i (corresponding to our x of the function, *1D Numpy array* of dimension equal to the number of elements considered, i.e., $n+1$) so that the ending curve is aligned with the velocity and Level Set field. At every iteration of the algorithm a new set of control points nearer the minimum energy is to be found.

Recalling that f, i.e. the cost functional, is equal to eq. (22), consider h as the constraint function which is calculated according to eq. (20), which recalculates at every iteration (when x changes) the distance to the curve for each of the LS values of the elements considered.

At every iteration of the augmented Lagrange method, a minimizer x_k will be found from Nelder-Mead method. Depending on the tolerance τ_k, at the end of the algorithm the optimal x^* will be returned, representing these control points for which the energy functional f is minimised.

As initial guess x_0 we can consider the vector containing the centers of mass of the blue elements introduced in Section 3.2. The algorithm will follow.

Algorithm 1 Augmented Lagrangian with Nelder-Mead

Require: f, h, x_0, λ_0, μ_0, τ_0, k_{max}
  while $err > \tau_k$ and $k < k_{max}$ **do**
    $\mathcal{L}_A = \mathcal{L}_f(f, x_k, \lambda_k, \mu_k, h)$
    $x_{k+1} = NelderMead(\mathcal{L}_A, x_k, \tau_k)$
    $err = \|x_{k+1} - x_k\|$
    $\lambda_{k+1} = \lambda_k - \mu_k h(x_{k+1})$
    $\mu_{k+1} = \mu_k \beta$, with $\beta \in [4, 10]$
    $\tau_{k+1} = \max(\tau_k, \frac{\tau_k}{10})$
    $k = k+1$, $x_k = x_{k+1}$, $\lambda_k = \lambda_{k+1}$, $\mu_k = \mu_{k+1}$, $\tau_k = \tau_{k+1}$
  end while
  return x_k^*

Chapter 4

Evaluation of the interface reconstruction

Firstly, this chapter includes relevant information on the implementation of the given dataset to reconstruct the interface, where results over the mesh domain and the external energy cost (thus the one coming from the velocity constraints) are presented. Then, an evaluation of the energy minimizing curve is executed, highlighting the impact of these external constraints. Finally, issues faced during the work and possible adjustments are exposed.

4.1 Results from mesh at interface

Result 1. Sampling neighbouring elements and interfacial elements. From the given mesh we can retrieve the elements needed. We found that there are a total of 23047 elements. From these, 922 are considered, chosen within an ϵ, which was found to be $\epsilon \approx 0.28$, therefore considering only the elements in the neighbourhood of the interface. These 922 elements represent all the elements with their respective averaged velocity attached (retrieved as explained in Step 1. of subsection 3.2.3) that we consider as input for the optimization algorithm. From these, 122 are the elements lying on the interface (*red* elements, also found thanks to Step 1. of subsection 3.2.3), while the rest are our *blue* elements.

Result 2. Intervals of influence on curve for each element (velocity constraint). Each of the red elements which compose the whole curve, from which we retrieve each velocity constraint, corresponds to an interval of our curve. Thus we have to choose the number of intervals equal the number of

interfacial elements, i.e., 122. They are ordered and the intervals of the curve on which the blue elements are imposed are retrieved from the red intervals, as described in Step 2. of Subsection 3.2.3.

Result 3. Computation of external energy costs. We now compute the external cost which derives from the velocity constraints. Each cost derived from each velocity constraint is weighted (α and w) according to the hat functions described in the Data extraction subsection. Since the whole length of the curve has to span from 0 to 1 (as it is a Bézier curve), a parametrization of the intervals from each element (and thus velocity) is done. For example, taking into consideration the curve approximative length being $\ell \approx 14.6$, an element with its interval on the curve being $[7, 7.5]$ would then be parametrized and become $\approx [0.48, 0.51]$. See Figure 9 for a visual representation of two picked intervals.

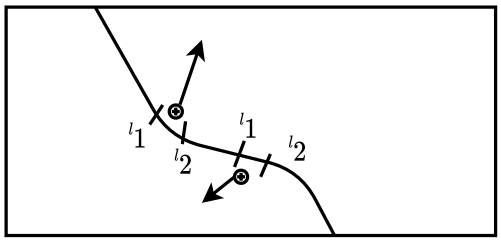

Figure 9: Visual representation of two random intervals of the curve with their velocity constraints.

Thanks to the parametrization process, we can obtain the external energy costs (21) for the entire curve. Since it is an integral, we perform the quadrature (27). Here, we recall that the quadrature points that originally span from $[-1, 1]$ also have to be parametrized to satisfy each interval at which the integral is evaluated. For example, considering the parametrized interval $[0.48, 0.51]$, each quadrature point (in our case 10) has to be in the range $\in [0.48, 0.51]$.

Now that each integral is computed, we can resolve the total external energy cost. Using the initial guess vector x_0 of control points with size 922, we sum each velocity constraint of each element and find the cost being ≈ 418.6. In the following we present the results regarding the external energy cost obtained with the first iteration of the developed algorithm. Figure 10 shows the resulting external cost to demonstrate the influence the constraints have on the curve, at each interval. The Bézier curve has adapted against the velocity constraints imposed (influencing energy terms). In the figure, 922 control points are marked in red and the velocity arrows have been omitted for simplicity.

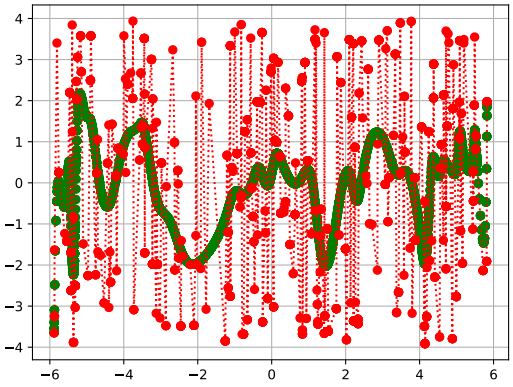

Figure 10: Energy cost functional highlighting adaptation to velocity constraints, plotted using the Python library *Matplotlib*. The red dots are the control points.

4.2 Evaluation of the energy-based constrained optimization

The optimization algorithm is still in progress, but nonetheless, we can define the solutions to be obtained and their motivation. Thanks to the optimization algorithm, the optimal solution x^* is found and a Bézier curve can then be formulated from the respective control points. These control points resemble the exact Bézier curve that is in fact describing and delineating the interface between the two phases. It is energy minimising and respects the Level Set constraints, thus optimally resolving the interface. Additionally, this curve improves the Level Set redistancing process as the separation between the two fluid is consistent with the LS discrete field values (the curve considers the surrounding velocity field).

4.2.1 Exemplary effects of multiple constraints on curve optimization

As shown in [16], the final curve respects the surrounding constraints previously imposed. An example of curves under multiple constraints is shown in Figure 11. As it can be seen, multiple operators are imposed on the curves, ranging from *director* (with arrow and without), *point* attractor to *curve* operators. The dark (filled) dots are the control points of the B-spline curve and the white (unfilled) ones are the parameter intervals on the curve to which an operator is applied. Each of these operators influences the curve local shape. The remaining part of the shape is determined by minimizing the internal energy, while the total curve still minimizes an energy functional. The curves model and adapt themselves in order to minimise that energy and be consistent with the

surrounding field, i.e. the operators.

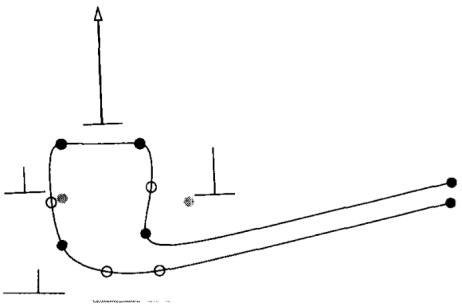

Figure 11: B-spline curves restricted by constraints or operators and shaped in an energy minimising way. The operators are imposed on the curve over the intervals described by the white dots. Figure reproduced from [16].

As having chosen an interactive approach, it is possible to change the parameters α in the convex combination of the internal energies to see the impact on the results. With a higher value on the bend energy, since it reflects the quantity of energy needed to bend the curve, we expect high resistance at control points with respect to the curvature, and a stronger local impact on the shape of the curve at these points. On the other hand, having a higher value on the stretch, which can be seen as the effort for the curve to stretch itself (the energy needed), will result in less tendency to lengthen itself, making the curve resist stretching.

The effect of energy minimising can be noticed by the fact that, as can be seen from Figure 12 and as mentioned by [16], when the weight factor w of an external energy functional is highly increased, e.g. the director in the figure, the corresponding part of the curve will almost exactly meet the specifications of the operator, but with the addition that the neighborhood of this segment of the curve can be heavily disturbed (right side of figure). This is due to the energy

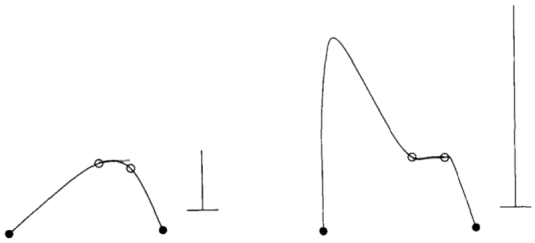

Figure 12: Graph highlighting the strength influence of the director operator of the left curve when highly increased. Figure reproduced from [16].

in this portion of the curve (where the director lies) dominating the total energy of the curve so much that the energy of the portions in the neighborhood is ignored, leading to the neighborhood of this portion doing anything to decrease the energy, even if this causes them to oscillate.

4.2.2 Possible variations of proposed optimization approach

Specific numerical methods have been applied to solve this problem, but there are other algorithms that could have been adopted. The augmented Lagrangian could have been replaced by a constrained linear programming method, the simplex algorithm. Furthermore, the Nelder-Mead method could have been replaced by any numerical methods for finding minimums as the gradient descent or line search methods.

4.3 Encountered issues and adjustments

Throughout the work issues have occurred and this section will highlight some of those. First of all, the quantity of the velocities Γ to consider in the blue elements has been approximated. Since considering velocities at each degree of freedom of each element would have given a quite large set of velocities constraints, they were averaged up to just one per element. Having included all the set of velocities in the cost functional, a slight change in the final curve would be expected.

Another issue appeared in the geometry analysis process of the mesh, explained in Step 2 of Subsection 3.2.2. The goal is to find the elements of interest by their LS value sign so to group connected elements. As the interfacial elements were not always connected by edges but instead by a simplex, i.e. join of two vertices, the setup process had to consider that and check both edges and vertices.

An improvement and possible adjustment regards the weight w_i of each velocity in eq. (21) which was described via an *hat* function. This weight could have been refined in a better way by changing it depending on the guessed curvature from the optimization algorithm at each iteration. Additionally, as we considered the center of the *hat* as $\frac{l_1+l_2}{2}$, it would have been more accurate if, considered the geometry of each element and where the point of application of each averaged velocity lies, a projection of that point on the diameter of the element and a respective estimation of the "center" of the *hat* would have been performed, depending on where the projected point lies.

4.4 Restriction

The focus of the problem considered in this thesis is about fluids which have an open interface, thus a curve which does not intersect itself. An interesting case to study is when two fluids approach each other causing droplet breakout from which droplets result in one of the other fluid. This would mean to find a closed curve, i.e. a close representation of the interface, which needs further constraints.

Chapter 5

Conclusions and future work

5.1 Conclusions

Having a discrete dataset of point values, an energy minimising curve representation fitting the Level Set constraint, i.e. positive and negative values and matching the velocity information can be found. As we have seen, we can determine the interfaces of two-phase flows problems via direct representation of a Bézier curve. By considering the Level Set ϕ distinguishing the surrounding domains finding the path the curve is supposed to pass through, and declaring the energy functionals of the problem as the objective function of the minimization problem, we can formulate the problem as an optimization statement. Thanks to constrained optimization techniques, we can minimize the energy of that curve depending on the surrounding velocity field, leading to an optimized interface with respect to energy and velocity constraints. Energy minimization is achieved via augmented Lagrangian method, an algorithm similar to the penalty method, that reformulates the constrained optimization in an unconstrained form. In practice, it has been solved with the help of Nelder-Mead method.

Contribution and applications of this work lie in the simulation of multi-phase flows, physical phenomena such as rising bubbles, merging droplets, and more generally to problems for which two initially simple subdomains cohabitate and are driven by the same set of state equations, the key issue being to choose appropriate models to capture interfaces as well as to resolve surface tension force.

5.2 Future work

Future work may include a few features that could be included in the optimization scope. In the first place, different constraints could be added to the cost functional. As highlighted in Subsection 2.2.1 and by [16], a *curve* attractor

$$E_{cat} = \int_{l_1}^{l_2} w(s) \left\| x - \ell(s) \right\|^2 ds \tag{31}$$

could be added to the initial interface (at $\phi = 0$) to remain close to the current curve, so that the optimized curve is now attracted by the initial one. Additionally, a *point* attractor could be used instead of rigid constraint to allow sloppy curves that are still consistent with the Level Set field though not equal pointwise with the LS values. The optimized curve would then be attracted by the point p. The constraint with the considered inferred point p on the Level Set would then be defined as

$$E_{pat} = c \left\| x - p \right\|^2 . \tag{32}$$

Another approach that would be of major interest is to find a curve representation which, other than fitting the LS values, also evolves in time according to the surrounding velocity field. This means to have control points which are time-dependent, obtaining an optimized curve which takes care of the final time and also considers the history in a time-continuous setting. The cost functional $c(x)$ defined in eq. (22) would then be surrounded by

$$\int_0^T c(x,t)dt \tag{33}$$

and all the curve equations would be defined according to the time t.

Bibliography

[1] D.P. Bertsekas and W. Rheinboldt. *Constrained Optimization and Lagrange Multiplier Methods*. Computer science and applied mathematics. Elsevier Science, 2014.

[2] Pierre Bézier. *The Mathematical Basis of the UNISURF CAD System*. Butterworth-Heinemann, 1986.

[3] David Byatt. Convergent variants of the nelder-mead algorithm. 2000.

[4] I. D. Coope and C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, 11(4):859–869, 2001.

[5] F. Gao and L. Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51(1):259–277, 2012.

[6] Jakub Konka. Nelder-mead simplex algorithm. `https://www.jakubkonka.com/2013/10/22/nelder-mead-simplex.html`. Accessed: 2021-01-14.

[7] J. C. Lagarias, B. Poonen, and M. H. Wright. Convergence of the restricted nelder-mead algorithm in two dimensions. *SIAM Journal on Optimization*, 22(2):501–532, 2012.

[8] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated, 2015.

[9] K. I. M. McKinnon. Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.

[10] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[11] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[12] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*, volume 153. Springer Science & Business Media, 2006.

[13] Michael J. D. Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14(1):224–248, 1978.

[14] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Neural information processing series. MIT Press, 2012.

[15] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A survey of optimization methods from a machine learning perspective. *IEEE Transactions on Cybernetics*, 2019.

[16] W. Wesselink and R. C. Veltkamp. Interactive design of constrained variational curves. *Computer Aided Geometric Design*, 12(5):533 – 546, 1995.

[17] H. Yuan, Z. Chen, C. Shu, Y. Wang, and S. Shu. A free energy-based surface tension force model for simulation of multiphase flows by level-set method. *Journal of Computational Physics*, 345:404–426, 2017.